Advanced Topics in CUDA

Cliff Woolley, NVIDIA Developer Technology Group

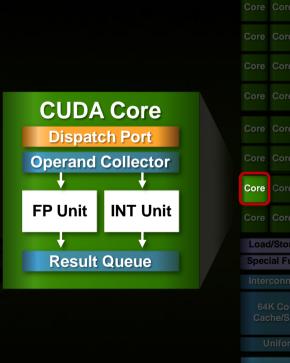




RECAP: SCHEDULING

GPU Architecture – Fermi: 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



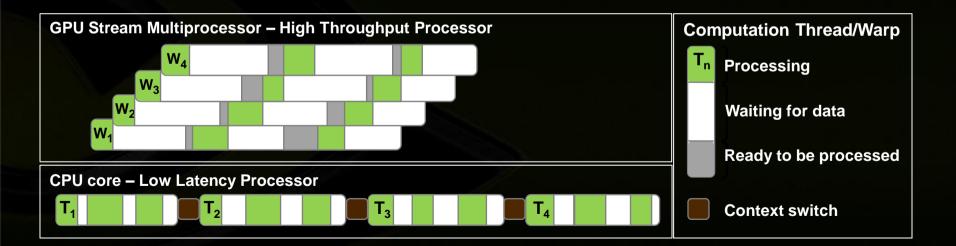




Low Latency or High Throughput?



- CPU architecture must minimize latency within each thread
- GPU architecture hides latency with computation from other thread warps



GPU Latency Hiding: Warp Switching



	Warp Scheduler	Warp Scheduler
	Instruction Dispatch Unit	Instruction Dispatch Unit
	Warp 8 instruction 11	Warp 9 instruction 11
1	Warp 2 instruction 42	Warp 3 instruction 33
	Warp 14 instruction 95	Warp 15 instruction 95
1	:	:
	Warp 8 instruction 12	Warp 9 instruction 12
	Warp 14 instruction 96	Warp 3 instruction 34
	Warp 2 instruction 43	Warp 15 instruction 96

Time

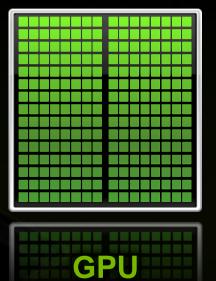


OVERLAPPING PROCESSING WITH DATA TRANSFERS

Available engines

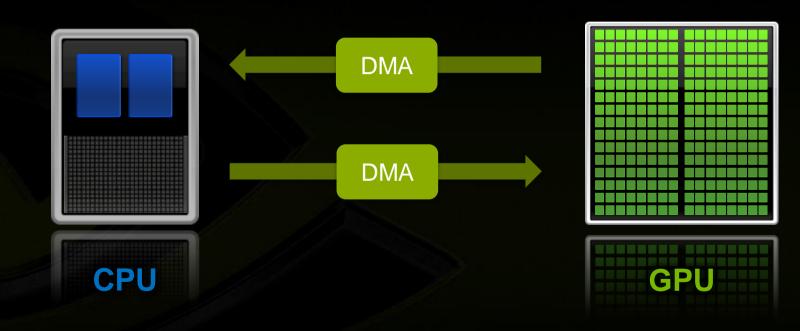






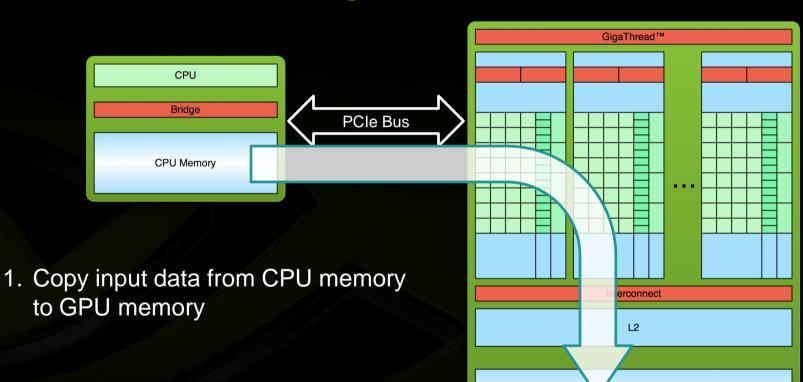
Available engines





Standard Processing Flow

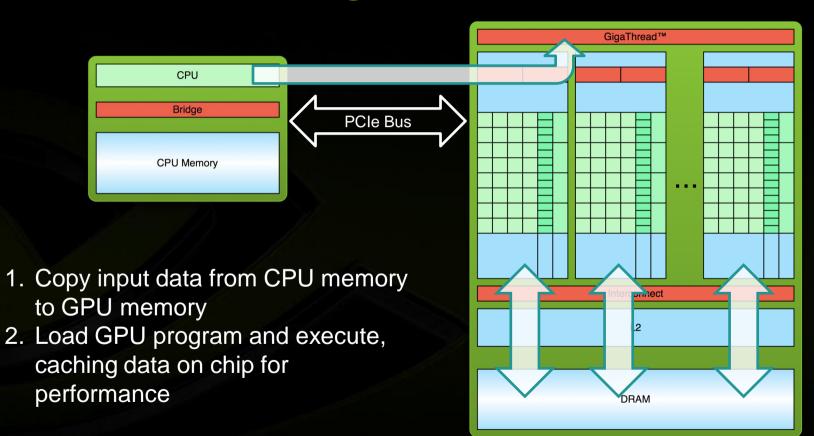




DRAM

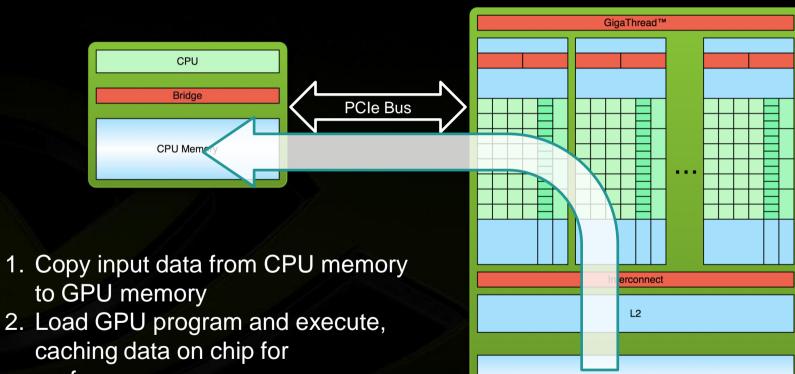
Standard Processing Flow





Standard Processing Flow





DRAM

performance

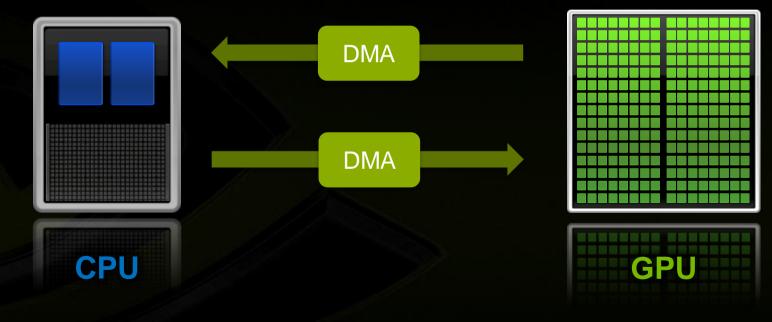
3. Copy results from GPU memory to CPU memory

Objective



- Tasks not on the critical path should be hidden
 - i.e. overlapped with other tasks
- Critical path is frequently the transfer of result data from GPU to CPU
 - e.g. path data

Objective: Overlap Processing With Transfers



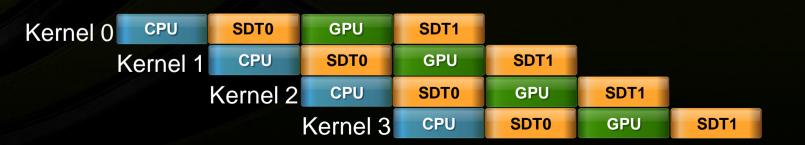


Overlapping Processing With Transfers

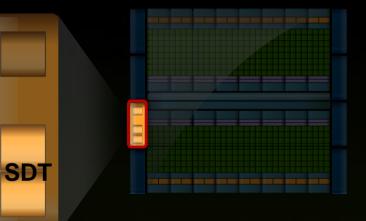
Dual DMA engines

- Simultaneous CPU→GPU and GPU→CPU data transfer
- Fully overlapped with CPU and GPU processing time

Activity Snapshot:







Phase A: No Pipelining





No overlap

- Kernel runs on GPU
- Transfer data across PCIe from device to host
- Post process on CPU

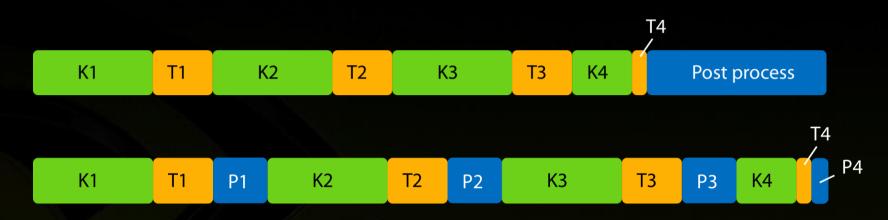
Phase A: No Pipelining



6	۰ 🌔 🔺	Timel	ine									□ •
Row Filters	Y											
	Seconds		9.465491	4 9.4754914	9.4854914	9.4954914	9.5	054914	9.5154914	9.5254914	9.53549	9.545491-
Proce	esses											
😑 Pip	elineDemo.ex											
😑 Tł	nread [5396]											
	Function Calls	-	Level 0	cudaDeviceSynchronize				cudaMem	сру			
	runction Calls	Υ.	Level 1	cuCtxSynchronize				cuMemcpyDt	toH_v2			
	Tools Extensi											
	Push / Pop	-	Level 0				GPU	version A				
	Push / Pop	W.	Level 1	Generate on GPU				Copy resu	ults			
🗆 CI	UDA											
	Context 0											
Ξ (Context 1 [0]											
	Runtime API	\mathbf{Y}	0	cudaDeviceSynchronize				cudaMem	сру			
	Driver API	\mathbf{Y}	0	cuCtxSynchronize				cuMemcpyDt	toH_v2			
	Warnings	∇	Ranges									
	Memory	Y .										
÷	Compute	Y.	-									
÷	Streams											
÷	Counters											

Phase B: Batching





- No overlap
 - Preparation for next phase...

Phase B: Batching



								_
60	🎒 🖀 🚺 🕨	Timelin	e					□ ▼
Row Filters	Y							
1	Seconds				9.5887361 9.5897361 9.5907361 9.5917361	9.5927361 9.5937361	9.5947361 9.5957361 9.5967361 9.5977361	9.5987361 9.5997361
😑 Proc								
🖃 Pip	pelineDemo.ex							
🗆 T	hread [5396]							
	Function Calls		Level 0 cudaMemcpy Level 1 cuMemcpyDtoH_v2		cudaMemcpy cuMemcpyDtoH_v2		cudaMemcpy cuMemcpyDtoH_v2	
	Tools Extensi							
1	Seconds 95817361 95827261 9587761 9587761 9587761 sses elineDemo.ex elineDemo.ex elineDemo.ex elineDemo.ex read [5396] cudaMemcpy evel 0 cudaMemcpy Tools Extensi evel 0 cudaMemcpy/DtoH_v2 Push / Pop V Level 0 context 0 Context 10 cudaMemcpy evel 0 evel 0 Priver API V o cudaMemcpyDtoH_v2 Warnings Ranges evel 0 cudaMemcpyDtoH_v2			GPU version B				
1		W.	Level 1 Copy results	Post process on CPU	Copy results	Post process on CPU	Copy results	Post process on CPU
🖃 C								
	Context 0							
	Context 1 [0]							
-	Context 1 [0] Runtime API	Y	0 cudaMemcpy		cudaMemcpy		cudaMemcpy	
	Context 1 [0] Runtime API				cudaMemcpy cuMemcpyDtoH_v2	0	cudaMemcpy cuMemcpyDtoH_v2	
	Context 1 [0] Runtime API Driver API	Ÿ.	0 cuMemcpyDtoH_v2			4 1		: :
	Context 1 [0] Runtime API Driver API Warnings	Y Y	0 cuMemcpyDtoH_v2			4		
•	Context 1 [0] Runtime API Driver API Warnings Memory	* 7 7	0 cuMemcpyDtoH_v2 Ranges	: :		8		
	Context 1 [0] Runtime API Driver API Warnings Memory Compute	* 7 7	0 cuMemcpyDtoH_v2 Ranges					
Ŧ	Context 1 [0] Runtime API Driver API Warnings Memory Compute Streams	* 7 7	0 cuMemcpyDtoH_v2 Ranges	• •		1		• •

Phase C: Overlap Kernel and Transfer





Overlap GPU compute with PCIe transfer
 Use streams to specify dependencies
 K1→T1, K2→T2 etc.

Phase C: Overlap Kernel and Transfer



	۹ 🌒 🔺 🔘	Time	line										D.
Row Filte	ers 🔽												
	Second	ds	9.7402106	9.740	3106	9.7404106	9.7405106	9.7406106	9.7407106	9.7408106	9.7409106	9.7410106	9.7411106
🖃 Pr	rocesses												
	PipelineDemo.ex												
Ξ	Thread [5396]												
	Function Calls	Y	Level 0 Level 1	cudaLaunch cuLaunchKernel	cudaMem cuMem	cudaLaunch cuLaunchKernel	cudaMemcpyAs cuMemcpyDto	cudaLaunch cuLaunchKernel	cudaMemcpyAs cuMemcpyDto	cudaLaunch cuLaunchKernel	cudaMemcpyAs cuMemcpyDto	cudaLaunch cuLaunchKernel	cudaMemcpyAs cuMemcpyDto
=	Tools Extensi.												
	Push / Pop	. 7	Level 0 Level 1	Generate on GPU	Copy results	Generate on GPU	Copy results	GPU versio Generate on GPU	on C Copy results	Generate on GPU	Copy results	Generate on GPU	Copy results
Ξ	CUDA												
±	Context 0												
-	Context 1 [0]												
	Runtime API	Y	0	cudaLaunch	cudaMem	cudaLaunch	cudaMemcpyAs	cudaLaunch	cudaMemcpyAs	cudaLaunch	cudaMemcpyAs	cudaLaunch	cudaMemcpyAs
	Driver API	∇	0	cuLaunchKernel	cuMem	cuLaunchKernel	cuMemcpyDto	cuLaunchKernel	cuMemcpyDto	cuLaunchKernel	cuMemcpyDto	cuLaunchKernel	cuMemcpyDto
	Warnings		Ranges										
	Memory	\mathbb{Y}						10000KB [DeviceToHost				
	 Compute 	∇											
	Streams												
	Stream 0	Y											
	Stream 1	Y					kernel						
	Stream 2	Y						10000KB [DeviceToHost				
	Stream 3	Y											
	Stream 4	Y											
	 Counters 												

Phase C: Overlap Kernel and Transfer



		🖀 🌒 🕨	Timelir	ie																		-
Row Filte	ers 🛛	7																				
		Seconds		9.7371898 9.7391	898 9.7411898 9	9.7431898 9.74	451898 9.7471	898 9.74918	98 9.7511898	9.7531898	9.7551898 9	.7571898 9.7	591898 9.7611	898 9.7631898 9.76	51898 9.76718	98 9.7691898	9.7711898	9.7731898 9	.7751898 9.77	771898 9.779	1898 9.7811898 9.7	831898
😑 Pi	rocess	ses																				
	Pipeli	lineDemo.ex																				
Ξ	Thre	ead [5396]																				
	Fu	unction Calls	7	Level 0										eviceSynchronize								
				Level 1									cuC	txSynchronize								
=	3 To	ools Extensi																				
	1	Push / Pop	Y	Level 0 Level 1									GPU ver: Wait	ion C for completion								Post pro
	CUE	DA																				
		ontext 0																				
	Co	ontext 1 [0]																				
	i i	Runtime API	∇										cudaD	eviceSynchronize								
		Driver API			N DA DA DA DA DAGA DAGAN DA								cuC	txSynchronize								
	1			Ranges																		
	1	Memory	\mathbb{Y}^{-}	100	00 10000	10000	10000	10000	10000	10000	10000	10000	10000	10000 10000	10000	10000	10000	10000	10000	10000	10000 10000	
	•	Compute	∇			-	_		-	-					-			-			-	
	Ξ.	Streams																				
			\mathbb{Y}^{-}																			
			∇	100	00	10000		10000		10000		10000		10000	10000		10000		10000		10000	
			\mathbb{Y}^{-}		10000		10000		10000		10000		10000	10000		10000		10000		10000	10000	
			Υ.																			
		Stream 4	∇																			
	•	Counters																				

Phase D: Overlap Kernel, Transfer and CPU

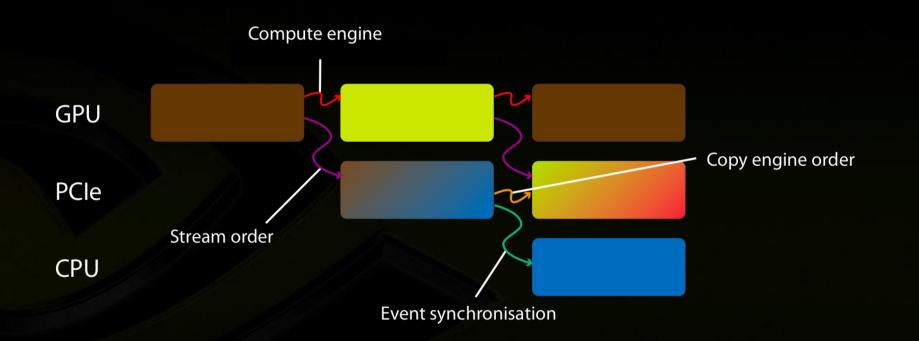




- Overlap GPU compute, PCIe transfer and CPU compute
- Use streams to specify K/T dependencies
- Use events to specify T/P dependencies
 - T1 \rightarrow P1, T2 \rightarrow P2 etc.

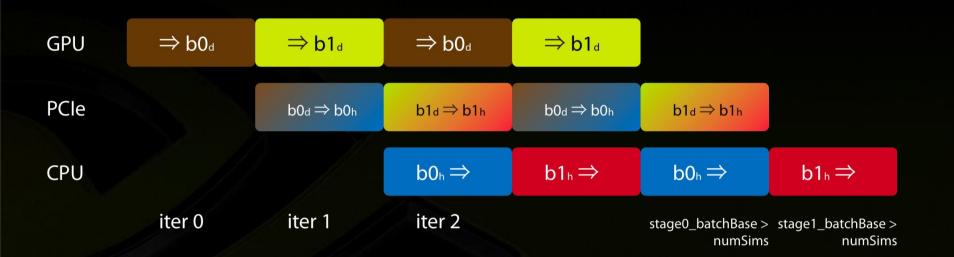
Phase D: Work Dependencies





Phase D: Use Double Buffering





Phase D: Overlap Kernel, Transfer and CPU

0	🙆 🖀 🚯 🕨	Timelir	ne														□ -
Row Filte	rs 🔽																
	Seconds		9.8420345 9.	.8440345	9.8460345	9.8480345 9.85	00345 9.852	0345 9.854	0345 9.856	0345 9.858	0345 9.860	0345 9.8620	1345 9.864(9.866	0345 9.868	0345 9.8700	0345
😑 Pr	ocesses																
	PipelineDemo.ex																
=	Thread [5396]																
	Function Calls	¥.	Level 0 cudaEvent Level 1 cuEventSy														1
	Tools Extensi																
	Push / Pop	7	Level 0 Level 1 Wait fo	or data Post	pr Post proc	e Post process	Post process	Post process	Post process	GPU version D Post process	Post process						
=	CUDA																
+	Context 0																
-	Context 1 [0]																
	Runtime API	\mathbb{Y}^{-}	0 cudaEvent	Synchron	•		N.						I I	N I		N 1	
	Driver API	\mathbb{Y}^{-}	0 cuEventSy	nchronize						N I			ll l	N.	N.	N I	
	Warnings	\mathbb{Y}^{-}	Ranges														
	Memory	Y	10	0000KB D] 100	000KB D 10000KB	D 10000KB D	10000KB D	10000KB D	10000KB D	10000KB D	10000KB D	10000KB D	10000KB D	10000KB D	10000KB D	10000KB D	10000KB D
	 Compute 	\mathbb{A}^{-}			-	_	_	-	_	_		_	_	_	_		
	 Streams 																
	Stream 0	7															
		\mathbb{Y}^{-}															
	Stream 2	\mathbb{Y}^{-}															
	Stream 3	\mathbb{Y}^{-}	kernel 10	0000KB D]	10000KB	D	10000KB D		10000KB D		10000KB D		10000KB D		10000KB D		10000KB D
	Stream 4	∇	kernel	100	000KB D	10000KB D		10000KB D		10000KB D		10000KB D		10000KB D		10000KB D	
	 Counters 																



Phase D: Overlap Kernel, Transfer and CPU

) 😂 🆀 🚯 🕨	Timeli	ne																				P]-
Row Filter	ars 🔽																							
i i	Seconds	s	9.852027	7 9.8521277	9.8522277	9.8523277	9.8524277	9.8525277	9.8526277	9.8527277	9.8528277	9.8529277	9.8530277	9.8531277	9.8532277	9.8533277	9.8534277	9.8535277	9.8536277	9.8537277	9.8538277	9.8539277	9.8540277	_ /
😑 Pro	rocesses																							
	PipelineDemo.ex																							///
	Thread [5396]																							
	Function Calls	Y	Level 0	cudaLaun cuLaunc																			001	
F	Tools Extensi																							
	Push / Pop	Y	Level 0 Level 1 Ger	enerate on	Wait for								GPU vers P	ersion D Post process o	on CPU								Gener	
Ξ						<u> </u>		<u> </u>			<u> </u>	<u> </u>	<u> </u>	<u> </u>		<u> </u>		<u> </u>						
۰																								_
=																								_ r
4		Y		cudaLaun																				<u> </u>
	Driver API	Y		cuLaunc																				
	Warnings		Ranges																					
		Y.											107	0000KB DeviceT	oHost									
	 Compute 	Y																						_ r
	Streams																							_
4	Stream 0																							
/	Stream 1																							
	Stream 2																							_ [
/	Stream 3								kernel															_
4	Stream 4	Y											107	0000KB DeviceT	oHost									
4	 Counters 																							





CONCURRENT KERNELS

Concurrent Kernel Execution





Sequential Kernel Execution

Parallel Kernel Execution



ANALYSIS-DRIVEN OPTIMIZATION

Performance Optimization Process



Use appropriate performance metric for each kernel

For example, Gflops/s don't make sense for a bandwidth-bound kernel

Determine what limits kernel performance

- Memory throughput
- Instruction throughput
- Latency
- Combination of the above

Address the limiters in the order of importance

- Determine how close to the HW limits the resource is being used
- Analyze for possible inefficiencies
- Apply optimizations
 - Often these will just fall out from how HW operates



New in CUDA Toolkit 4.0: Automated Performance Analysis using Visual Profiler

Summary analysis & hints

- Per-Session
- Per-Device
- Per-Context
- Per-Kernel

New UI for kernel analysis

- Identify the limiting factor
- Analyze instruction throughput
- Analyze memory throughput
- Analyze kernel occupancy

File View						
Analysis					8	;
Instruction Through	put	Analysis for kernel conv	olutionColumnsKern	el on device GeForce G	5TX 480	1000
 Divergent brand Control flow div Replayed Instru- Global Global Shared Shared memory Hint(s) The kernel is Shared memory Thy using a stranged memory Shared memory Conflicts can be Olarge a differer Refer to the "Ar for more details. 	cor tance bar bar cor tance point ga ng con rith cory red ppro- tance ppro- tance ppro- tance ng cor tance point cory cory cory cory cory cory cory cory	erice(%): 0.03 erice(%): 20.05 fory replay(%): 0.00 ik conflict replay(%): 0.00 ik conflict replay(%): 25.38 ik conflict per shared memor npute bound , to reduce ins if the instruction mix, as singl entails, etc. have different thin thitmetic hitmisc functions, ompler flags(-ft2=true, -prec mer precision loss; ento Linstructions' section in beank conflicts are high w used by opriate padding for data stor ank; g data in shared memory, th d Memory' section in the "Pe	struction count e precision floating poin oughputs. Use double p (34.7) are interpreted -div=false, -prec-sign= the 'Performance Guide hich causes serialization ed in shared memory so us changing access patt	recision arithmetic only wi as double precision as per false etc) to get higher pe lines [*] chapter of the CUD/ n of threads within a warp that each thread in a war ern;	hen required (E.g. C standard); sformance, but may A C Programming Guide . Shared memory bank rp accesses data from a	
		Show all columns				
The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction count •The kernel is compute bound, to reduce instruction, •The kernel is compute bound, to reduce instruction, •The kernel is completed bound, to reduce instruction count result in some precision loss. Refer to the "Airfluence Instructions" section in the "Performance Guidelines" chapter of the CUDA C Programming Guide for more details. •Shared memory bank conflicts are high which causes serialization of threads within a warp. Shared memory bank conflicts can be reduced by •Olsing appropriate padding for data stored in shared memory so that each thread in a warp accesses data from a different bank; •Refer to the "Shared Memory" section in the "Performance Guidelines" chapter of the CUDA C Programming Guide for more details. Factors that may affect analysis Show all columns <u>GPU Timestamp (us) GPU Time (us) shared load Type/SM Runs4 Type/SM Runs4 1 38718 1652.96 334560 24600 Instruction throughput <u>3 44507.4 1652.93 334560 24600 </u></u>						
	ysis Ø > truction Throughput Analysis for kernel convolutionColumnsKernel on device GeForce GTX 480 • PC: 1.56 • Maximum IPC: 2 • Outgrant Moving Stand(%): 0.00 • Control Row divergence(%): 0.03 • Control Row divergence(%): 0.00 • Shared memory bank conflict per shared memory instruction(%): 99.90 • The kernel is compute bound, to reduce instruction count • Outgrant Row divergence(%): 0.03 • The kernel is compute bound, to reduce instruction count • Understand the instruction mix, as single precision Roating point, double precision Roating point, int, mem, transcredentals, etc. have different throughputs. Use double precision attimetic only when required (E.g., floating point, thrank floating point, iterals without an f suffix (34.7) are interpreted as double precision attimetic only when required (E.g., floating point iterals without an f suffix (34.7) are interpreted as double precision as per C standard); • The kernel is compute Road, for reduce instruction. • Try using compiler flags(-flat-true, -prec-div=flage, -prec-signt=flage etc) to get higher performance, but may result in some precision loss: Refer to the "Authmetic Instructions" section in the "Performance Guidelines" chapter of the CUDA C Programming Guide for more details. • Ousing appropriate padding for data stored in shared memory so that each thread in a warp accesses data from a different bank; • Ousing appropriate padding for data stored in shared memory so that each thread in a warp accesses data from a different bank; • Ousing appropriate padd					
Analysis	2	41989.6	1652.86	334560	24600	
	3	44507.4	1652.93	334560	24600	
Analysis	4	47024.9	1652.96	334560	24600	

1653.09

100340

334560

224560

24600

245.00

49541.

C 03003 3

Occupancy Analysis

Notes on profiler



Most counters are reported per Streaming Multiprocessor (SM)

- Not entire GPU
- Exceptions: L2 and DRAM counters

A single run can collect a few counters

- Multiple runs are needed when profiling more counters
 - Done automatically by the Visual Profiler
 - Have to be done manually using command-line profiler

Counter values may not be exactly the same for repeated runs

- Threadblocks and warps are scheduled at run-time
- So, "two counters being equal" usually means "two counters within a small delta"

See the profiler documentation for more information



ANALYSIS-DRIVEN OPTIMIZATION: IDENTIFYING PERF LIMITERS

Limited by Bandwidth or Arithmetic?



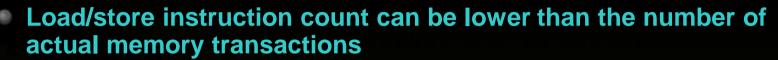
Perfect instructions:bytes ratio for Fermi C2050:

- ~4.5 : 1 with ECC on
- ~3.6 : 1 with ECC off
- These assume fp32 instructions, throughput for other instructions varies

Algorithmic analysis:

- Rough estimate of arithmetic to bytes ratio
- Code likely uses more instructions and bytes than algorithm analysis suggests:
 - Instructions for loop control, pointer math, etc.
 - Address pattern may result in more memory fetches
 - Two ways to investigate:
 - Use the profiler (quick, but approximate)
 - Use source code modification (more accurate, more work intensive)

A Note on Counting Global Memory Accesses



- Address pattern, different word sizes
- Counting requests from L1 to the rest of the memory system makes the most sense
 - Caching-loads: count L1 misses
 - Non-caching loads and stores: count L2 read requests ۲
 - Note that L2 counters are for the entire chip, L1 counters are per SM

Some shortcuts, assuming "coalesced" address patterns:

- One 32-bit access instruction -> one 128-byte transaction per warp
- One 64-bit access instruction -> two 128-byte transactions per warp
- One 128-bit access instruction -> four 128-byte transactions per warp

Analysis with Profiler



Profiler counters:

- instructions_issued, instructions_executed
 - Both incremented by 1 per warp
 - "issued" includes replays, "executed" does not
 - gld_request, gst_request
 - Incremented by 1 per warp for each load/store instruction
 - Instruction may be counted if it is "predicated out"
 - l1_global_load_miss, l1_global_load_hit, global_store_transaction
 - Incremented by 1 per L1 line (line is 128B)
- uncached_global_load_transaction
 - Incremented by 1 per group of 1, 2, 3, or 4 transactions
 - Better to look at L2_read_request counter (incremented by 1 per 32B transaction; per GPU, not per SM)

Compare:

32 * instructions_issued

/* 32 = warp size */

128B * (global_store_transaction + l1_global_load_miss)

Analysis with Modified Source Code



Time memory-only and math-only versions of the kernel

- Easier for codes that don't have data-dependent control-flow or addressing
- Gives you good estimates for:
 - Time spent accessing memory
 - Time spent in executing instructions
- Compare the times taken by the modified kernels
 - Helps decide whether the kernel is mem or math bound

Compare the sum of mem-only and math-only times to full-kernel time

- Shows how well memory operations are overlapped with arithmetic
- Can reveal latency bottleneck

Some Example Scenarios



time

mem math full

Memory-bound

Good mem-math overlap: latency likely not a problem

(assuming memory throughput is not low compared to HW theory)

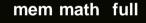


mem math full

Math-bound

Good mem-math overlap: latency likely not a problem

(assuming instruction throughput is not low compared to HW theory)



Balanced

Good mem-math overlap: latency likely not a problem

(assuming memory/instr throughput is not low compared to HW theory) mem math full Memory and latency bound Poor mem-math overlap: latency *is* a problem

Source Modification



Memory-only:

- Remove as much arithmetic as possible
 - Without changing access pattern
 - Use the profiler to verify that load/store instruction count is the same

Store-only:

Also remove the loads (to compare read time vs. write time)

Math-only:

- Remove global memory accesses
 - Need to trick the compiler:
 - Compiler throws away all code that it detects as not contributing to stores
 - Put stores inside conditionals that always evaluate to false
 - Condition should depend on the value about to be stored (prevents other optimizations)
 - Condition outcome should not be known to the compiler

Source Modification for Math-only



__global__ void fwd_3D(..., int flag)

value = temp + coeff * vsq; if(1 == value * flag) g_output[out_idx] = value;

. . .

If you compare only the flag, the compiler may move the computation into the conditional as well

Source Modification and Occupancy



Removing pieces of code is likely to affect register count

- This could increase occupancy, skewing the results
- See slide 23 to see how that could affect throughput

Make sure to keep the same occupancy

- Check the occupancy with profiler before modifications
- After modifications, if necessary add shared memory to match the unmodified kernel's occupancy

kernel<<< grid, block, smem, ...>>>(...)

Case Study: Limiter Analysis



- 3DFD of the wave equation, fp32
- Time (ms):
 - Full-kernel: 35.39
 - Mem-only: 33.27
 - Math-only: 16.25
- Instructions issued:
 - Full-kernel: 18,194,139
 - Mem-only: 7,497,296
 - Math-only: 16,839,792
- Memory access transactions:
 - Full-kernel: 1,708,032
 - Mem-only: 1,708,032
 - Math-only:

0

Analysis:

- Instr:byte ratio = ~ 2.66
 - 32*18,194,139 / 128*1,708,032
- Good overlap between math and mem:
 - 2.12 ms of math-only time (13%) are not overlapped with mem
- App memory throughput: 62 GB/s
 - HW theory is 114 GB/s, so we're off

Conclusion:

- Code is memory-bound
- Latency could be an issue too
- Optimizations should focus on memory throughput first
 - math contributes very little to total time (2.12 out of 35.39ms)

Summary: Limiter Analysis



Rough algorithmic analysis:

How many bytes needed, how many instructions

Profiler analysis:

- Instruction count, memory request/transaction count
- Analysis with source modification:
 - Memory-only version of the kernel
 - Math-only version of the kernel
 - Examine how these times relate and overlap

