

**GPU** TECHNOLOGY  
CONFERENCE

# CUDA OPTIMIZATION WITH NVIDIA NSIGHT™ VISUAL STUDIO EDITION

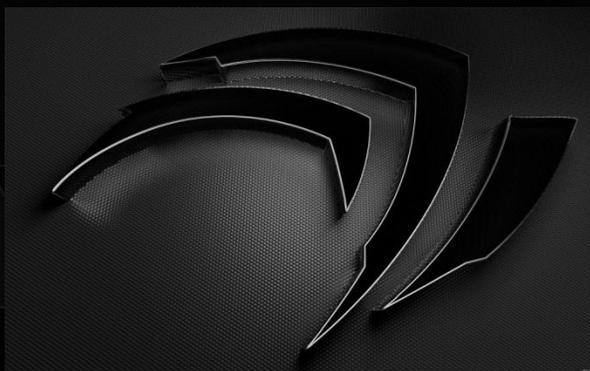
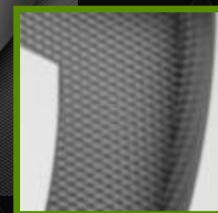
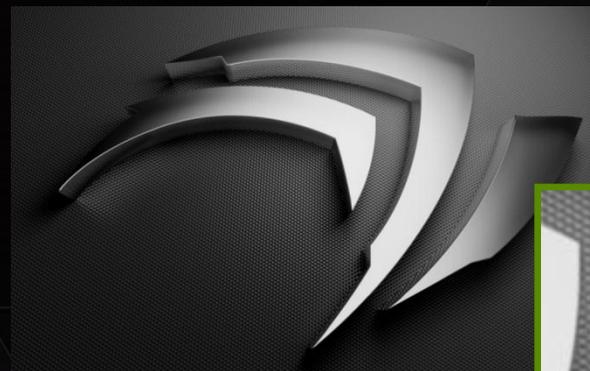
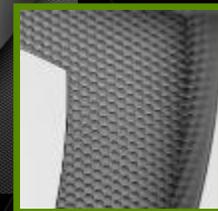
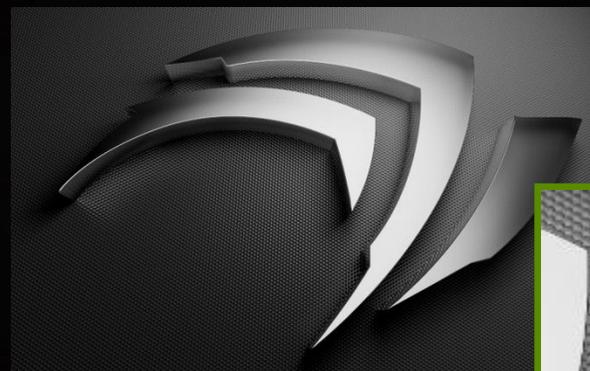
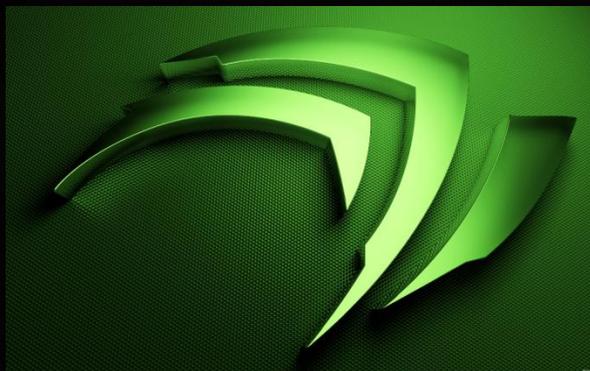
CHRISTOPH ANGERER, NVIDIA  
JULIEN DEMOUTH, NVIDIA

# WHAT YOU WILL LEARN

- ▶ An iterative method to optimize your GPU code
- ▶ A way to conduct that method with NVIDIA Nsight VSE

▶ Companion Code: <https://github.com/chmaruni/nsight-gtc2015>

# INTRODUCING THE APPLICATION



# INTRODUCING THE APPLICATION

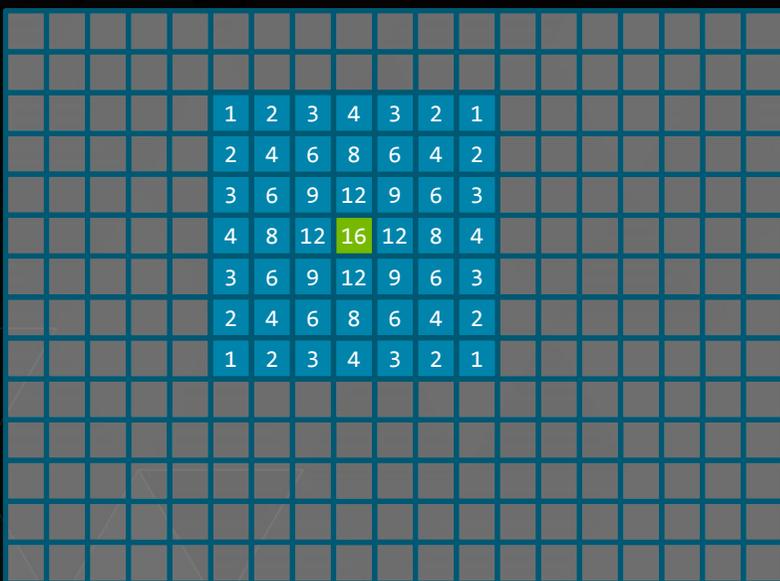
## ► Grayscale Conversion



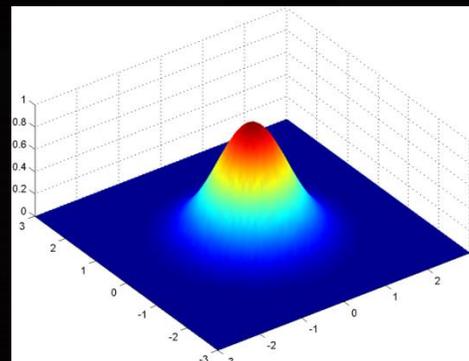
```
// r, g, b: Red, green, blue components of the pixel p  
foreach pixel p:  
    p = 0.298839f*r + 0.586811f*g + 0.114350f*b;
```

# INTRODUCING THE APPLICATION

## ► Blur: 7x7 Gaussian Filter



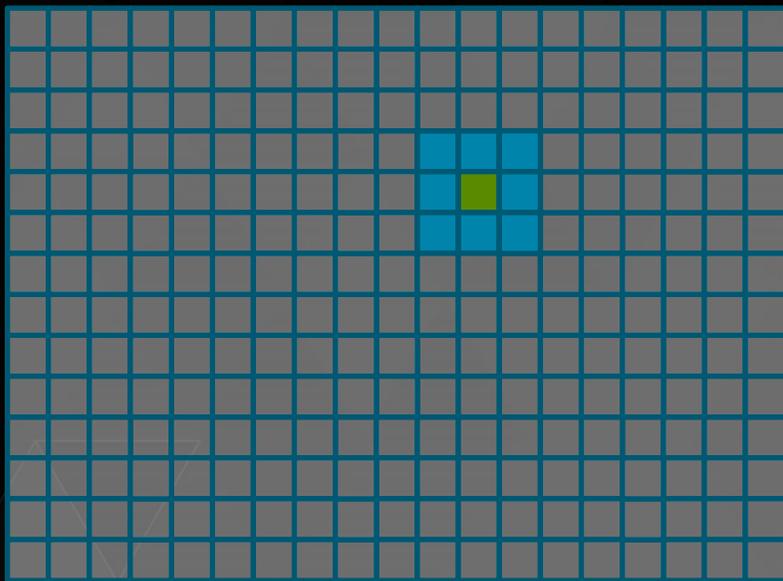
```
foreach pixel p:  
  p = weighted sum of p and its 48 neighbors
```



*Image from Wikipedia*

# INTRODUCING THE APPLICATION

## ► Edges: 3x3 Sobel Filters



`foreach` pixel `p`:

`Gx` = weighted sum of `p` and its 8 neighbors

`Gy` = weighted sum of `p` and its 8 neighbors

`p` = `sqrt(Gx + Gy)`

Weights for `Gx`:

-1	0	1
-2	0	2
-1	0	1

Weights for `Gy`:

1	2	1
0	0	0
-1	-2	-1

# ENVIRONMENT

- ▶ NVIDIA GTX Titan X
  - ▶ GM200
  - ▶ SM5.2
- ▶ Windows 7
- ▶ NVIDIA Nsight Visual Studio Edition 4.6

# PERFORMANCE OPTIMIZATION CYCLE

1. Profile Application

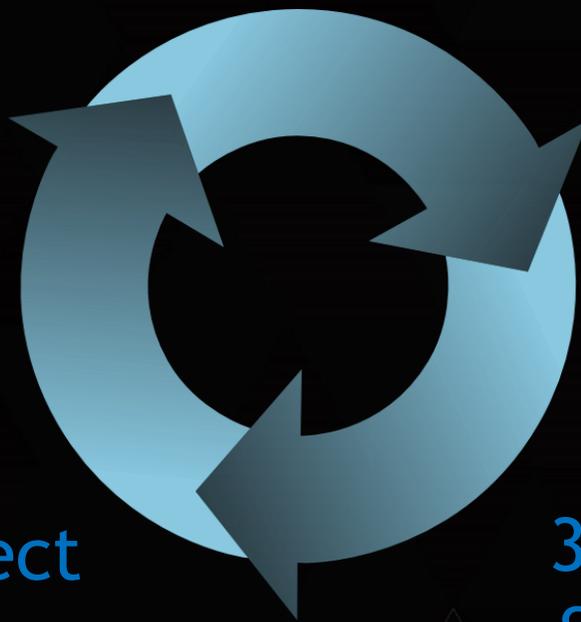
2. Identify Performance Limiter

3. Analyze Profile & Find Indicators

5. Change and Test Code

4. Reflect

4b. Build Knowledge



# PREREQUISITES

- ▶ Basic understanding of the GPU Memory Hierarchy
  - ▶ Global Memory (slow, generous)
  - ▶ Shared Memory (fast, limited)
  - ▶ Registers (very fast, very limited)
  - ▶ (Texture Cache)
- ▶ Basic understanding of the CUDA execution model
  - ▶ Grid 1D/2D/3D
  - ▶ Block 1D/2D/3D
  - ▶ Warp-synchronous execution (32 threads per warp)



**GPU** TECHNOLOGY  
CONFERENCE

# ITERATION 1

## TRACING THE APPLICATION

Verify  
Parameters

Select  
Trace  
Application

Activate  
CUDA

Launch

nsight-gtc2015-vs2...\_Capture\_000.nvact\* x nsight-gtc2015-vs2...pture\_000.nvreport nsight-gtc2015-vs2...pture\_000.nvreport nsight-gtc2015-vs2...pture\_000.nvreport

Application Settings Conn: localhost App: nsight-gtc2015-vs2013.exe Args: data\claw.ppm Sync: True Import From Project

Connection Name: localhost Disconnect

Application: C:\Users\jdemouth\Devtech\GTC\nsight-gtc2015-master\x64\Step-00\nsight-gtc2015-vs2013.exe

Arguments: data\claw.ppm

Working Directory: C:\Users\jdemouth\Devtech\GTC\nsight-gtc2015-master\

Remote Options

Triggers and Actions Start: On Process Launch Stop: Manually

Activity Type Trace Application

Trace Application (selected)  
Collects events from the target application. The analysis session and data collection are stopped when the launched application exits.

Trace Process Tree  
Collects events from the target application and all native child processes of the target application. The analysis session and data collection are not stopped when the launched application exits. The session and data collection must be stopped manually.

Trace Settings Domains: CUDA

- System (2/5) CPU Thread Trace, Module Trace
- Tools Extension (4/4) Markers, Push/Pop Ranges, Start/End Ranges, Resource Naming
- CUDA (4/4) Driver API Trace, Runtime API Trace, Software Counters, Kernel Launches and Memory Operations, Host Callback Trace
- OpenCL (3/3) API Trace, Resource Trace, Program Source Code, Program Build Callback Trace, Program Binary Code, Reference Counter, Command Trace
- DirectX (7/19) API Trace, CPU Frames, GPU Frames, Push Buffers, Shader Compiles, Performance Markers, Performance Ranges
- OpenGL (5/6) API Trace, CPU Frames, GPU Frames, Draw Calls, Transfers

Connection Status Application Control Capture Control

Available Devices Launch Kill Start Stop Cancel

Open Report on Stop  
Summary Report

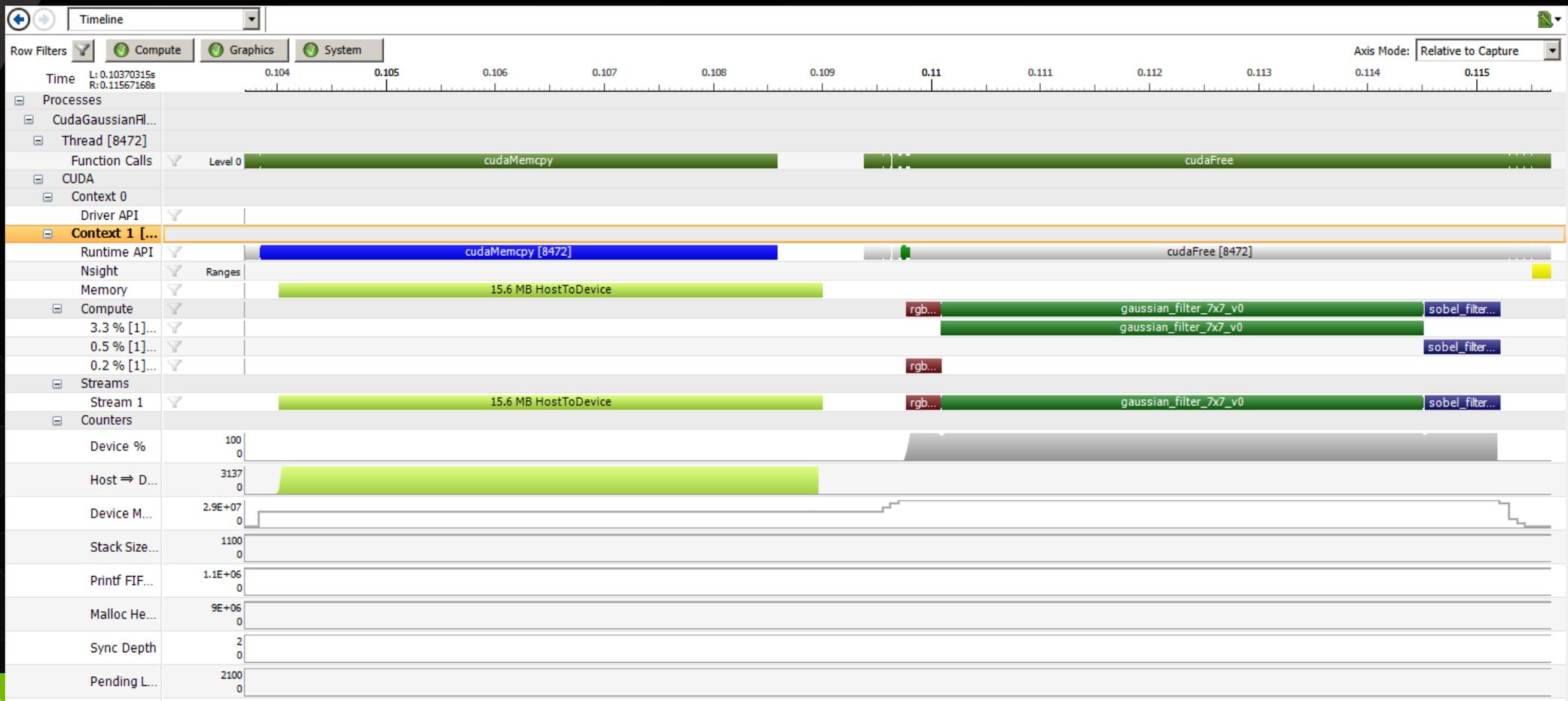
# NAVIGATING THE ANALYSIS REPORTS

The screenshot shows the NVIDIA Nsight Visual Studio Edition interface. The top window is titled "simpleStreams13072...pture\_000.nvreport" and "Activity2.nvact\*". A dropdown menu is open, showing the following items:

- Session Summary
- Common
  - Summary Report
  - Session Summary
  - Timeline
- CUDA
  - CUDA Summary
  - CUDA Devices
  - CUDA Driver API Call Summary
  - CUDA Driver API Calls
  - CUDA Launch Summary
  - CUDA Launches
  - CUDA Memory Copies
  - CUDA Memory Sets
  - CUDA Runtime API Call Summary
  - CUDA Runtime API Calls
- System
  - Disk IO Events
  - File IO Events
  - Function Calls
  - GPU Devices
  - Modules
  - Nsight Annotations
  - System Information

Three callout boxes on the left side of the image point to specific items in the menu:

- Timeline** points to the "Timeline" item under the "Common" category.
- CUDA Summary** points to the "CUDA Summary" item under the "CUDA" category.
- CUDA Launches** points to the "CUDA Launches" item under the "CUDA" category.



## IDENTIFY HOTSPOT (CUDA SUMMARY)

Top Device Functions By Total Time [Summary](#) | [All](#)

	Name	Launches	Device %
1	gaussian_filter_7x7_v6	1	0.01
2	sobel_filter_3x3_v1	1	0.01
3	rgba_to_grayscale_kernel_v1	1	0.00

Hotspot

- ▶ Identify the hotspot: gaussian\_filter\_7x7\_v0()

Kernel	Time	Speedup
Original Version	1.971ms	1.00x

# PERFORM KERNEL ANALYSIS

Select Profile  
CUDA Application

Select the Kernel

Select the  
Experiments (All)

Activity Type: Profile CUDA Application

- Trace Application
- Trace Process Tree
- Profile CUDA Application
- Profile CUDA Process Tree

Experiment Settings: Experiments: All Kernel

Kernel Selection

Kernels to Profile: gaussian\_filter\_7x7\_v0

After skipping: No kernels, profile All kernels.

Profile Options

- Print Progress Output to Console
- Non-Overlapping Input/Output Buffers
- Collect Information for CUDA Source View

Experiment Configuration

Experiments to Run: All (Kernel-Level)

All Kernel-Level Experiments

Select this experiment group to collect kernel-level experiments. Please note that this template adds significant overhead to the target application. When this group is selected, the following experiments will be run.

Experiment	Description
Achieved FLOPS	Calculates the achieved single/double floating point operations per second.
Achieved IOPS	Calculates the achieved integer operations per second.
Achieved Occupancy	Calculates the occupancy achieved at runtime of the kernel.
Branch Statistics	Collects efficiency metrics for the kernel's usage of flow control.
Instruction Statistics	Collects instructions per clock cycle (IPC), instructions per warp (IPW) and SM activity.
Issue Efficiency	Collects efficiency metrics for issuing the kernel's instructions.

Connection Status: Available Devices:

Application Control: Launch Kill

Capture Control: Start Stop Cancel

Open Report on Stop

Summary Report

Launch

# THE CUDA LAUNCHES VIEW

Select Kernel

Select Experiment

nsight-gtc2015-vs2...\_Capture\_000.nvact\*    nsight-gtc2015-vs2...pture\_000.nvreport    nsight-gtc2015-vs2...pture\_000.nvreport    nsight-gtc2015-vs2...pture\_000.nvreport

CUDA Launches    Hierarchy    Flat

Filter    Viewing: 3 / 3

	Function Name	Start Time (μs)	Duration (μs)	Occupancy	Registers per Thread	Grid Dimensions	Block Dimensions	Static Shared Memory per Block (bytes)	Dynamic Shared Memory per Block (bytes)	Cache Configuration Executed	Local Memory per Thread (bytes)	Device Name
1	gaussian_filter_7x7_v6	1,472,809.855	151.744	100.00 %	30	(80, 100, 1)	(32, 8, 1)	5120	0	PREFER_SHARED	0	Graphics
2	sobel_filter_3x3_v1	2,011,601.471	149.696	100.00 %	17	(80, 200, 1)	(32, 8, 1)	0	0	PREFER_SHARED	0	Graphics
3	rgba_to_grayscale_kernel_v1	922,330.399	71.200	100.00 %	10	(80, 100, 1)	(32, 8, 1)	0	0	PREFER_SHARED	0	Graphics

gaussian\_filter\_7x7\_v6<<<8000,256>>> [CUDA Launch]

- Device Launches
- Call Graph
- gaussian\_filter\_7x7\_v6 [CUDA Kernel]
  - Experiment Results
    - Occupancy
    - All Counters
    - Instruction Statistics
    - Branch Statistics
    - Issue Efficiency
    - Achieved FLOPS
    - Achieved IOPS
    - Pipe Utilization
    - Memory Statistics

View By: Size

Kernel

- Texture: 0.00 Req, 0.00 B
- Global: 384.00 kReq, 104.33 MB
- Local: 0.00 Req, 0.00 B
- L1/Tex Cache: 26.04 MB
- Global Atomics: 0.00 Req
- Shared Atomics: 0.00 Req
- Shared Memory: 2.82 MReq, 375.00 MB
- System Memory: 32.00 B
- Device Memory: 9.05 MB

Experiment Results

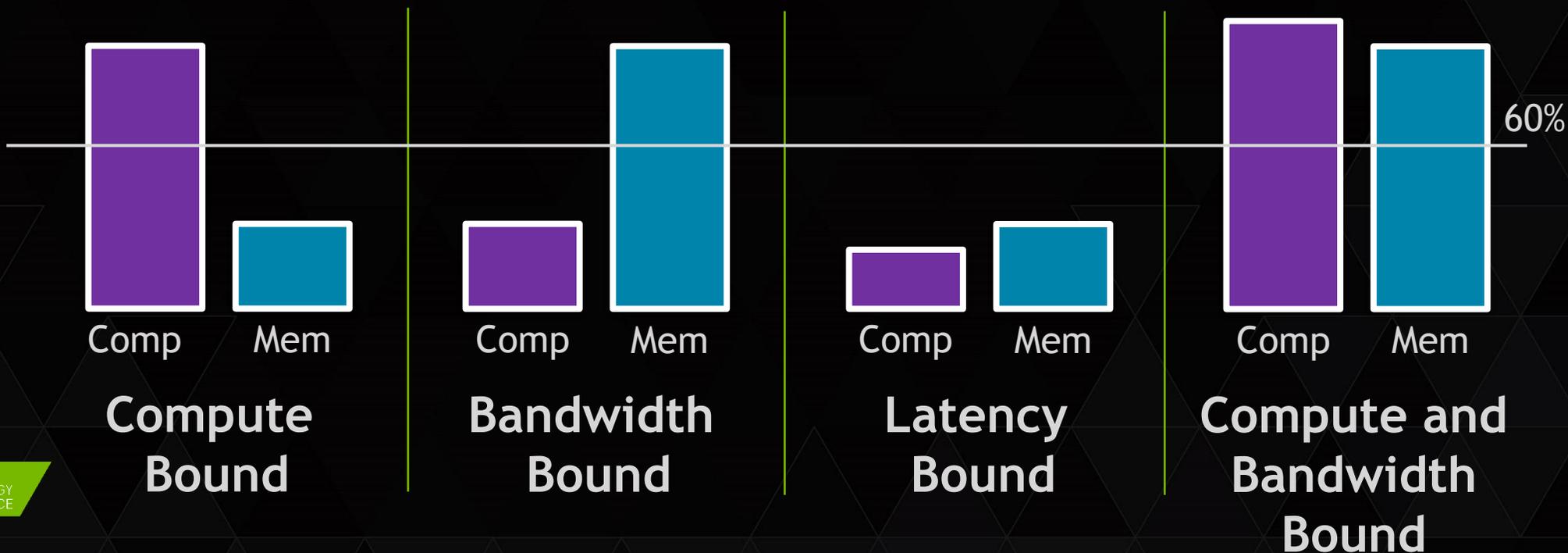
Name	Tota	Per'	Per'
Texture			
Fetches	0.00	0.00	0.00
L2 Tran	2.03	31.7	13.3
Size	0.00	0.00	0.00
Cache i	0.00	0.00	0.00
Cache i	39.0		
Global			
Reques	384,	6.00	2.53
Load	256,	4.00	1.68
Store	128,	2.00	843,
L2 Tran	1.23	19.3	8.16
Load	726,	11.3	4.78
Store	512,	8.00	3.37
Size	104,	1.67	671.
Load	88.7	1.42	570.
Store	15.6	256,	100.
Local			

Overview    Global    Local    Atomics    Shared    Texture    Caches    Buffers

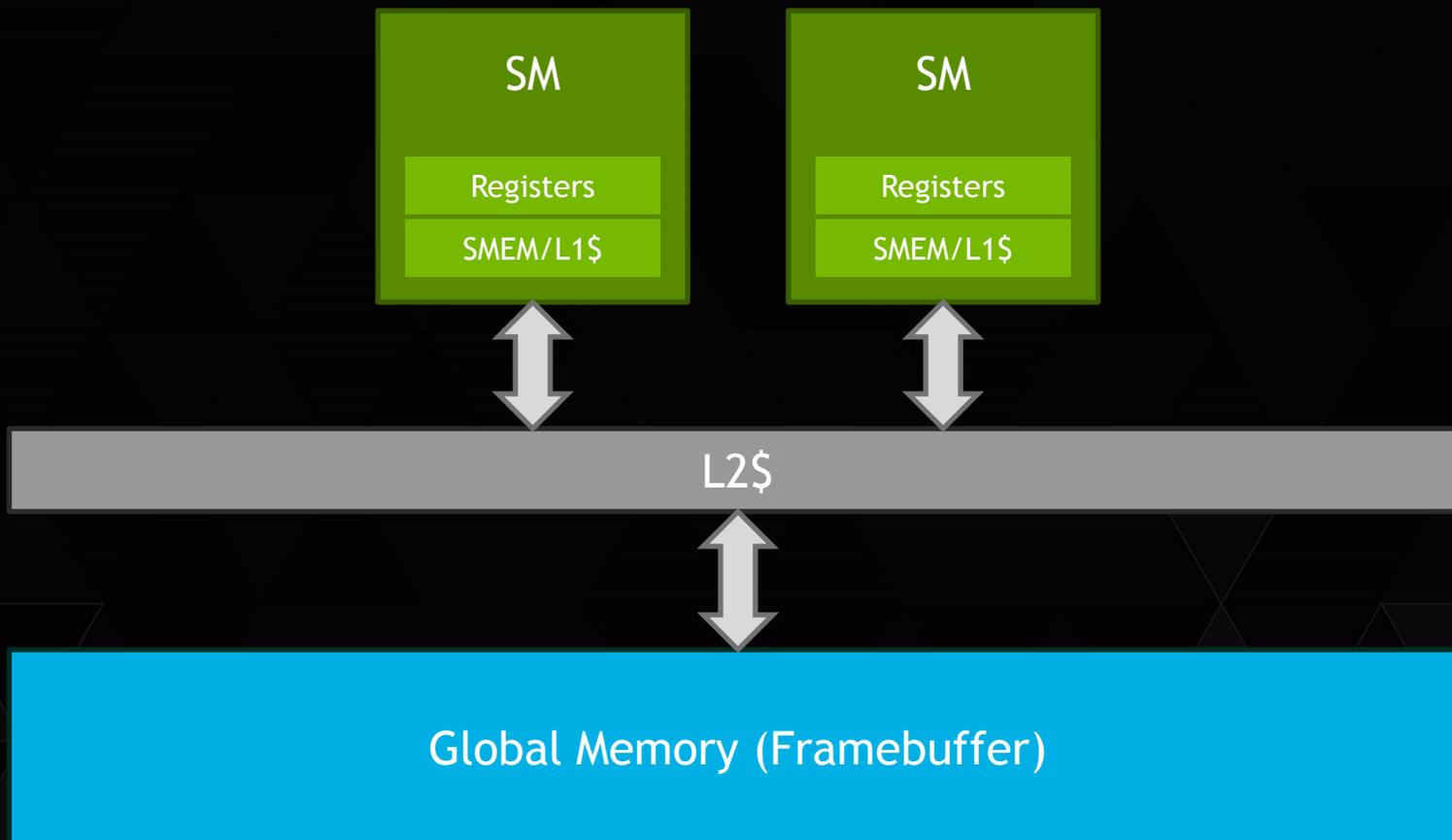
# IDENTIFY MAIN PERFORMANCE LIMITER



- ▶ Memory Utilization vs Compute Utilization
- ▶ Four possible combinations:

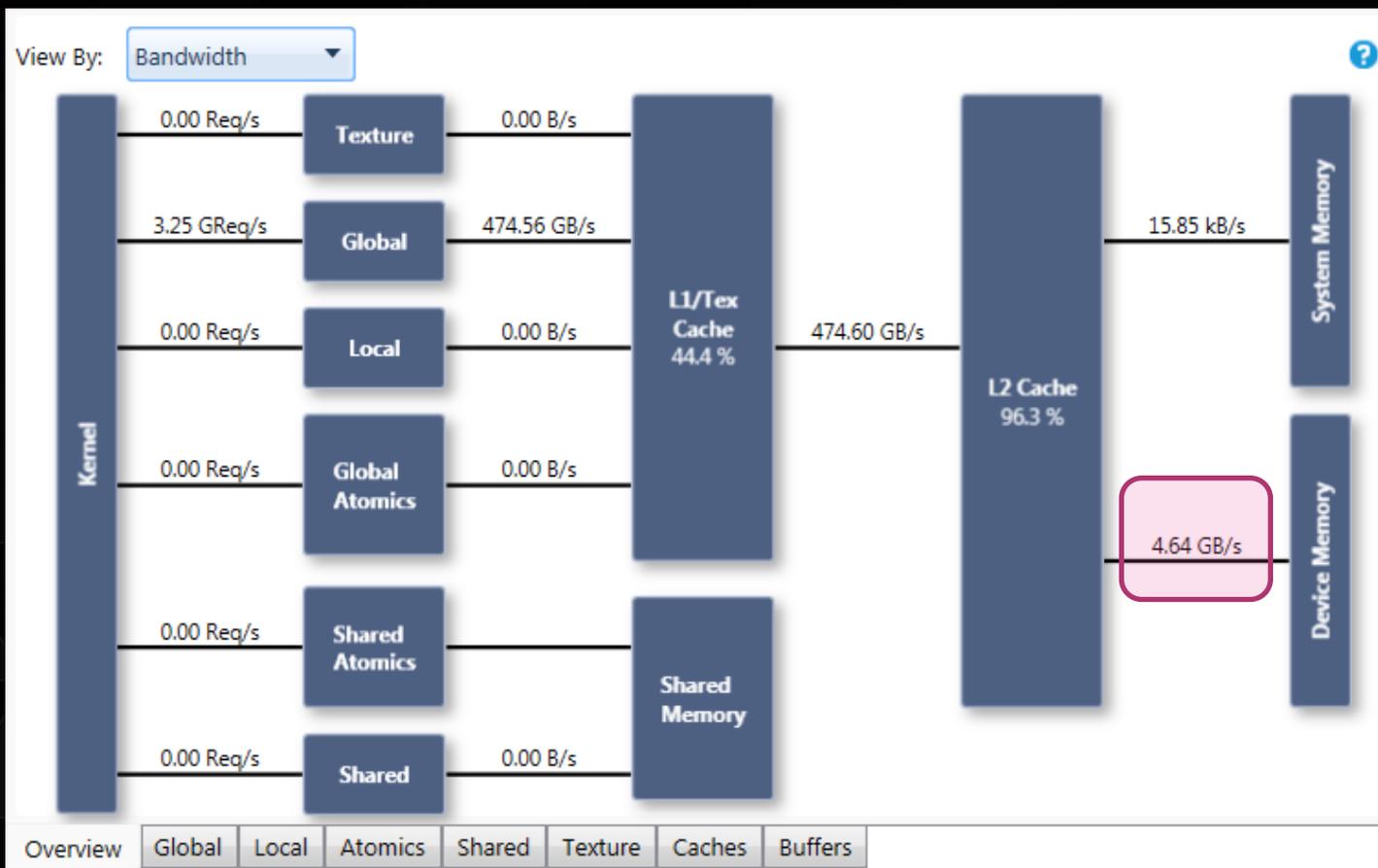


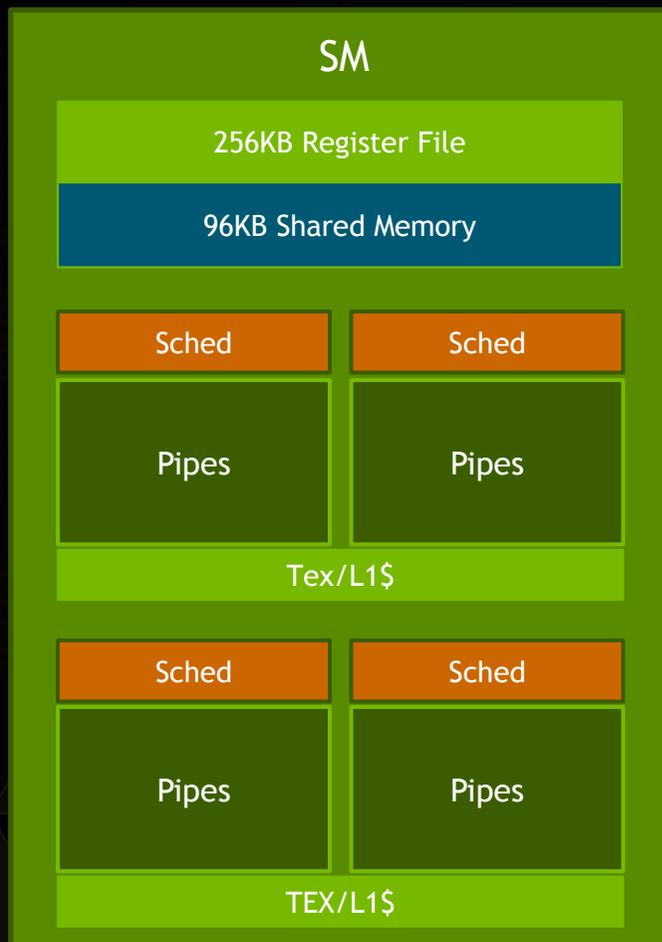
# MEMORY BANDWIDTH



# IDENTIFY PERFORMANCE LIMITER

- ▶ Utilization of L2\$ Bandwidth (BW) limited and DRAM BW < 2%
- ▶ Not limited by memory bandwidth



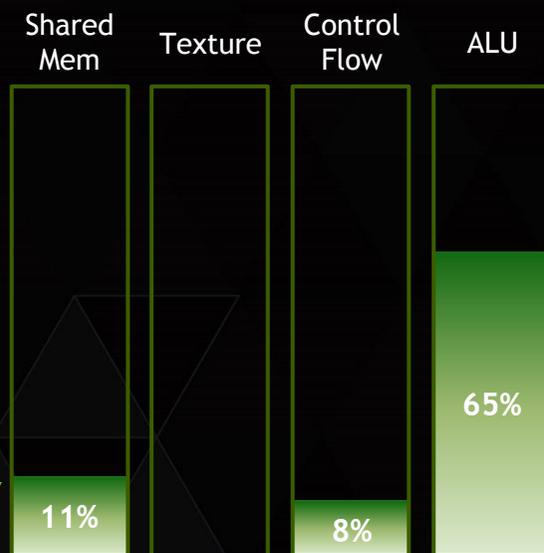


- ▶ Each SM has 4 schedulers (Maxwell)
- ▶ Schedulers issue instructions to pipes
- ▶ Each scheduler schedules up to 2 instructions per cycle
- ▶ A scheduler issues inst. from a single warp
- ▶ Cannot issue to a pipe if its issue slot is full

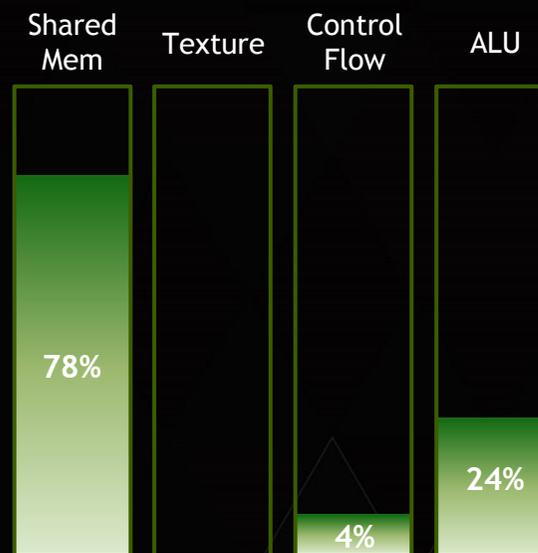
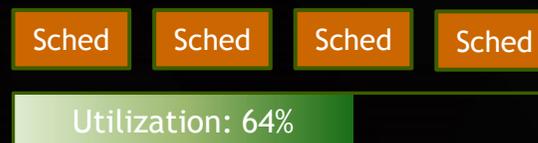
# INSTRUCTION THROUGHPUT



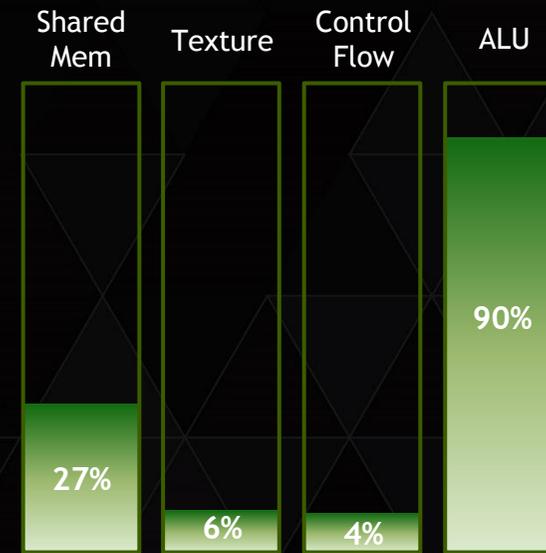
Schedulers saturated



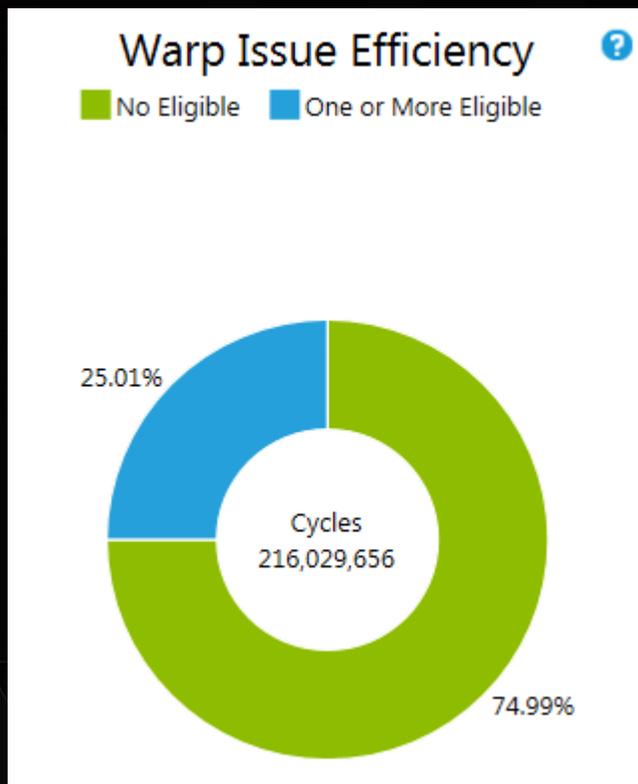
Pipe saturated



Schedulers and pipe saturated

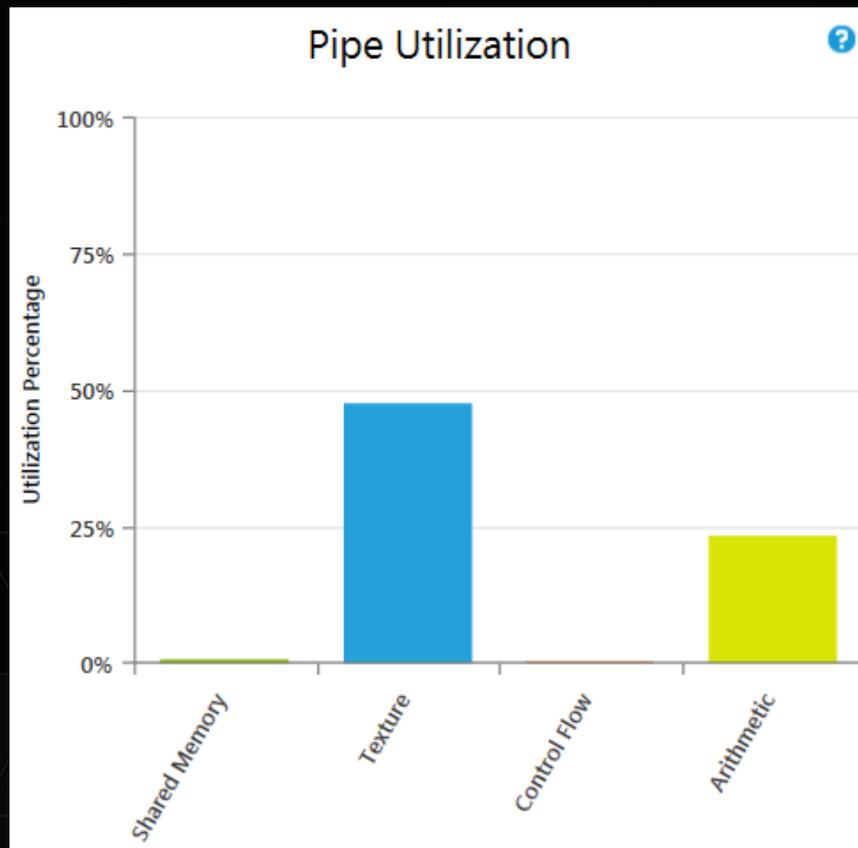


# WARP ISSUE EFFICIENCY



- ▶ Percentage of issue slots used (blue)
- ▶ Aggregated over all the schedulers

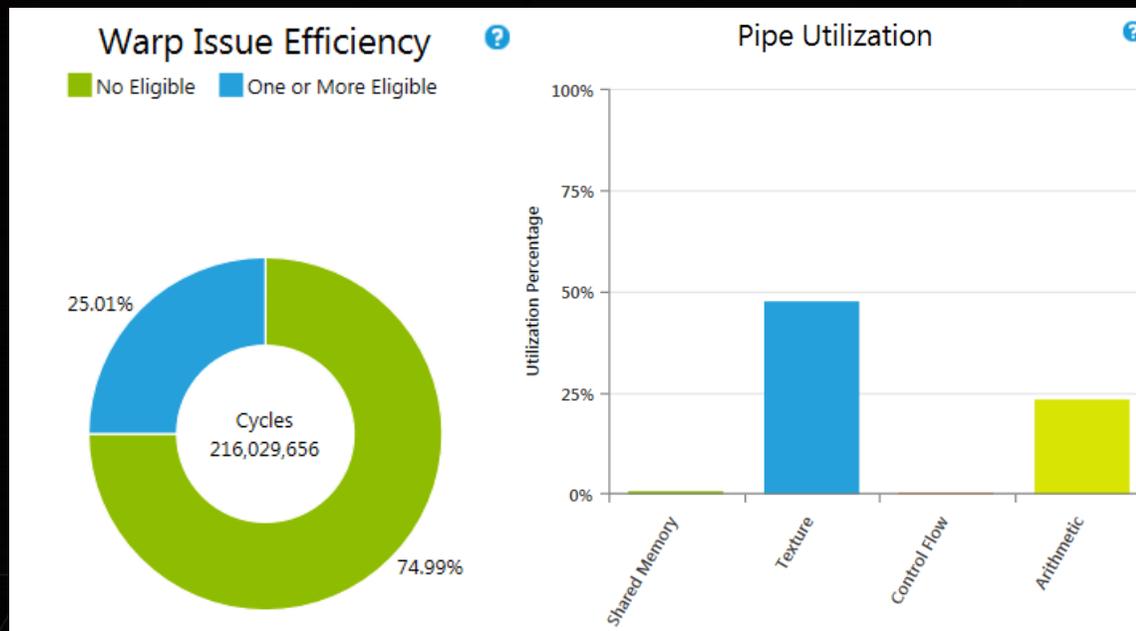
# PIPE UTILIZATION



- ▶ Percentages of issue slots used per pipe
- ▶ Accounts for pipe throughputs
- ▶ Four groups of pipes:
  - ▶ Shared Memory
  - ▶ Texture
  - ▶ Control Flow
  - ▶ Arithmetic (ALU)

# INSTRUCTION THROUGHPUT

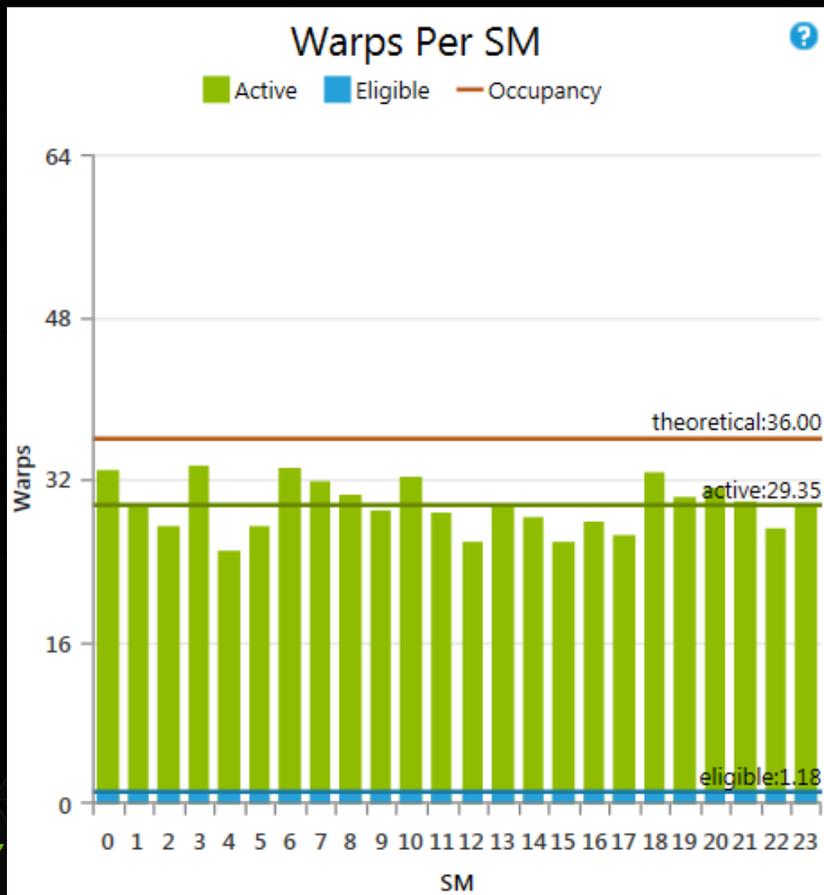
- ▶ Neither schedulers nor pipes are saturated



- ▶ Not limited by the instruction throughput

⇒ **Our Kernel is Latency Bound**

# LOOKING FOR INDICATORS



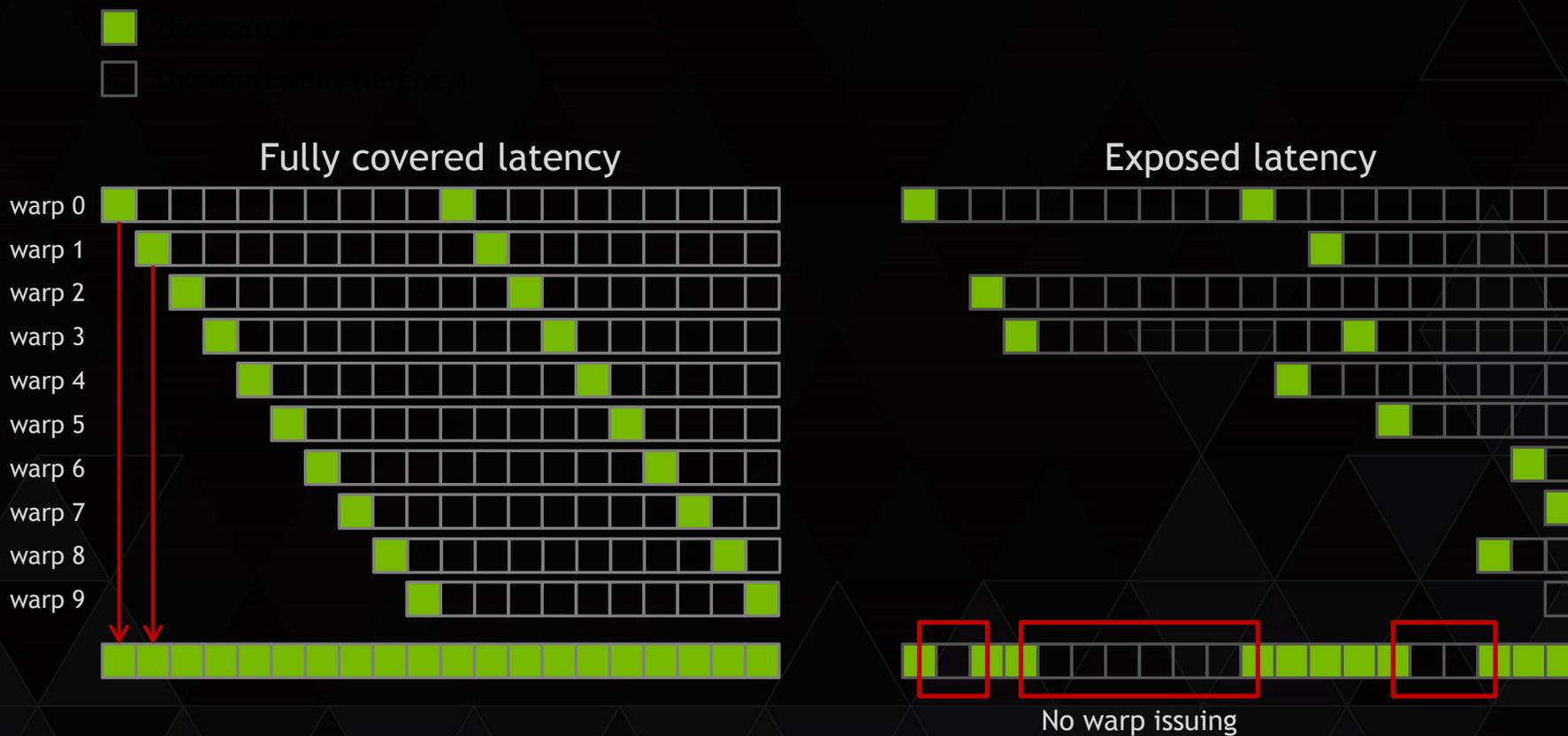
- ▶ 56% of theoretical occupancy
- ▶ 29.35 active warps per cycle
- ▶ 1.18 warps eligible per cycle
- ▶ Let's start with occupancy



- ▶ Each SM has limited resources
- ▶ 64K Registers (32 bit) shared by threads
- ▶ Up to 48KB of shared memory per block (96KB per SMM)
- ▶ 32 Active Blocks per SMM
- ▶ Full occupancy: 2048 threads per SM (64 warps)

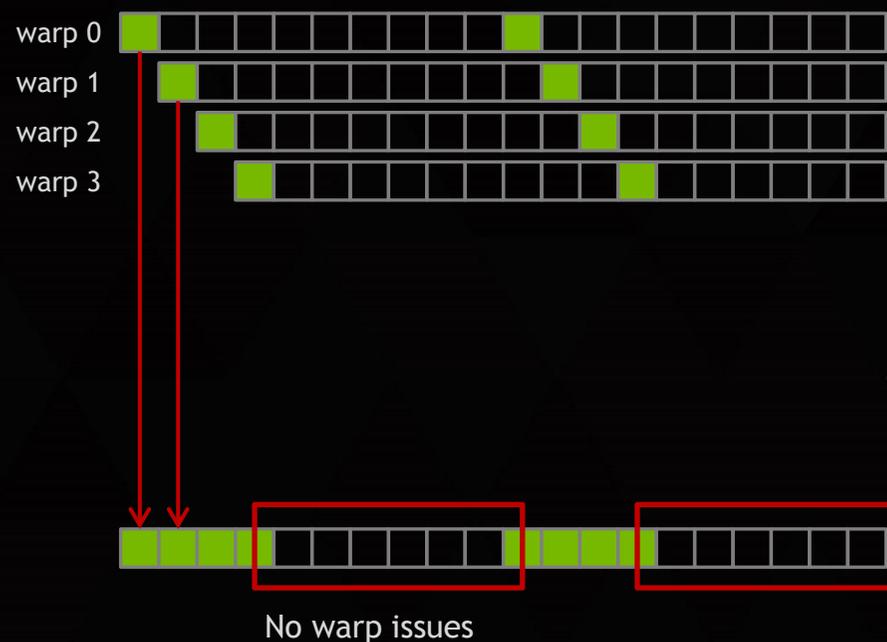


- GPU cover latencies by having a lot of work in flight





- ▶ Not enough active warps

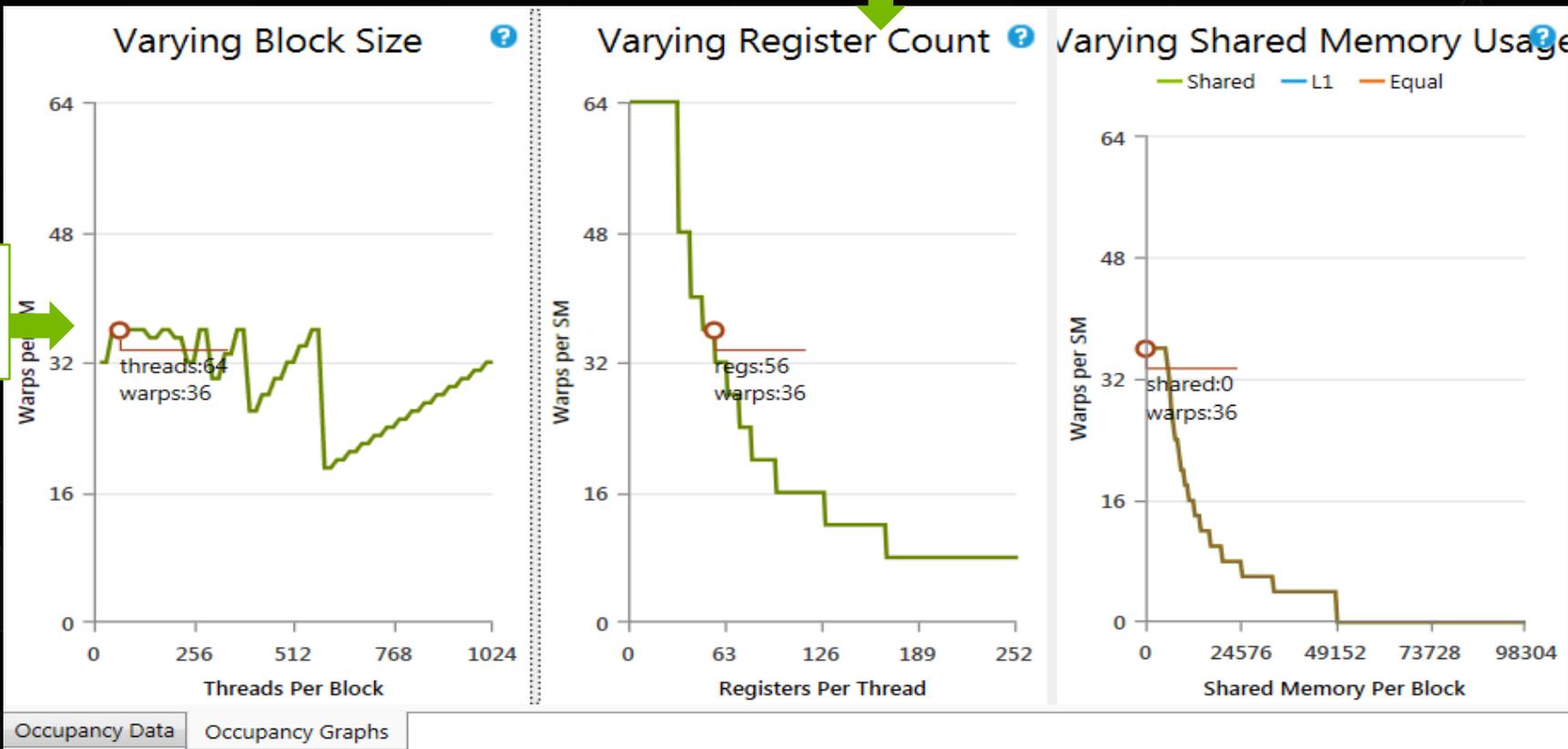


- ▶ The schedulers cannot find eligible warps at every cycle

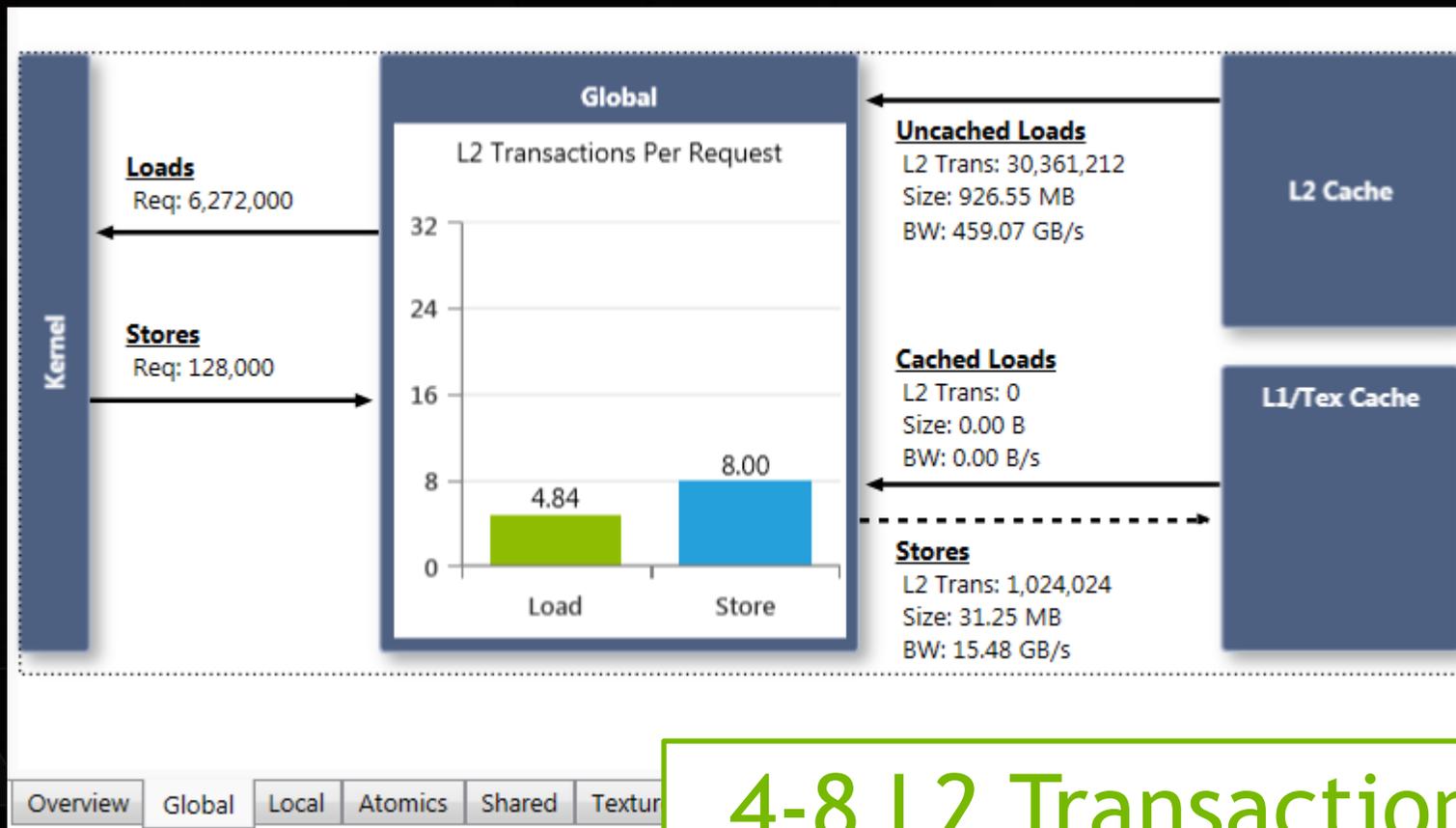
# LOOKING FOR MORE INDICATORS

We don't want to change the register count yet

Block Size seems OK



# CONTINUE LOOKING FOR INDICATORS

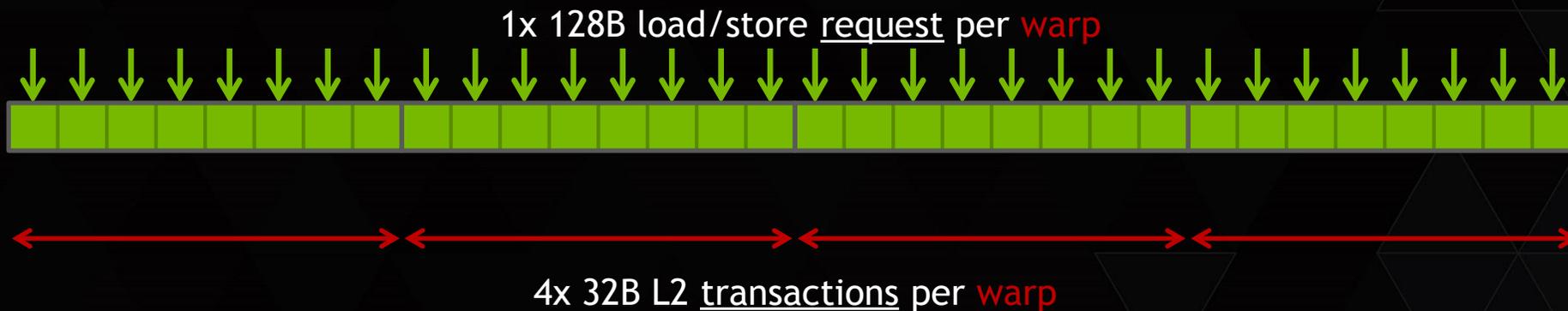


4-8 L2 Transactions  
per 1 Request

# MEMORY TRANSACTIONS: BEST CASE



- ▶ A warp issues 32x4B aligned and consecutive load/store request
- ▶ Threads read different elements of the same 128B segment



- ▶ 4x L2 transactions: 128B needed / 128B transferred



- ▶ Threads in a warp read/write 4B words, 128B between words
- ▶ Each thread reads the first 4B of a 128B segment

Stride: 32x4B



1x 128B load/store request per warp

warp 2

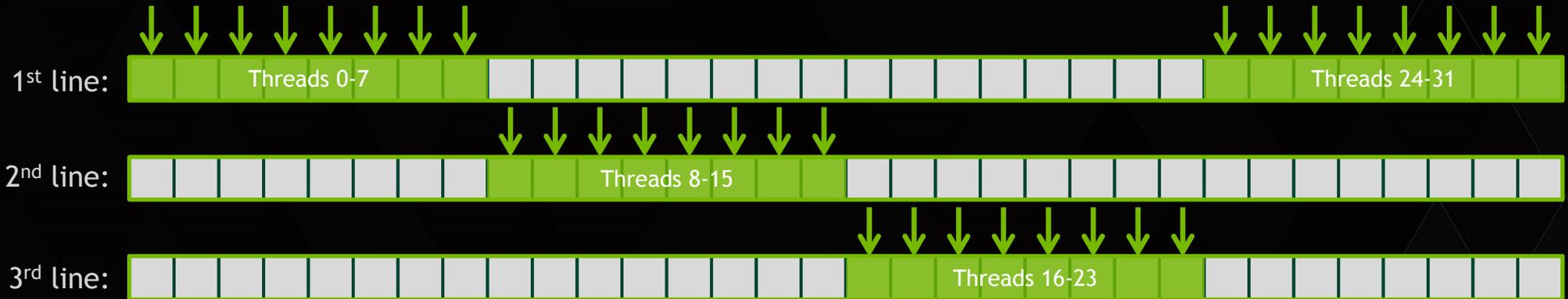


1x 32B L2 transaction per thread

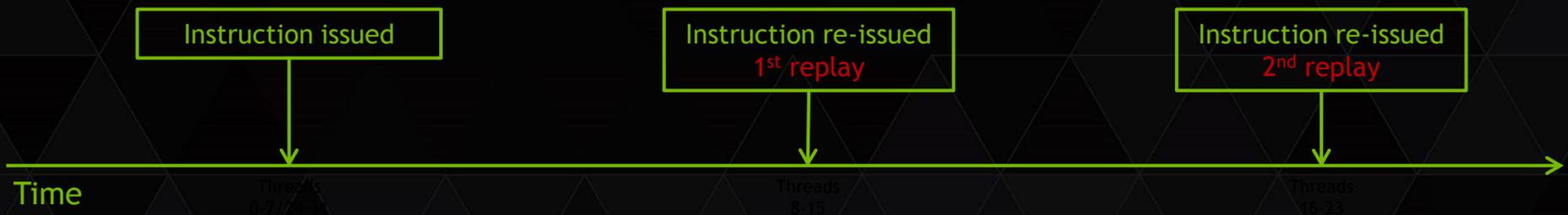
- ▶ 32x L2 transactions: 128B needed / 32x 32B transferred



- ▶ A warp reads from addresses spanning 3 lines of 128B

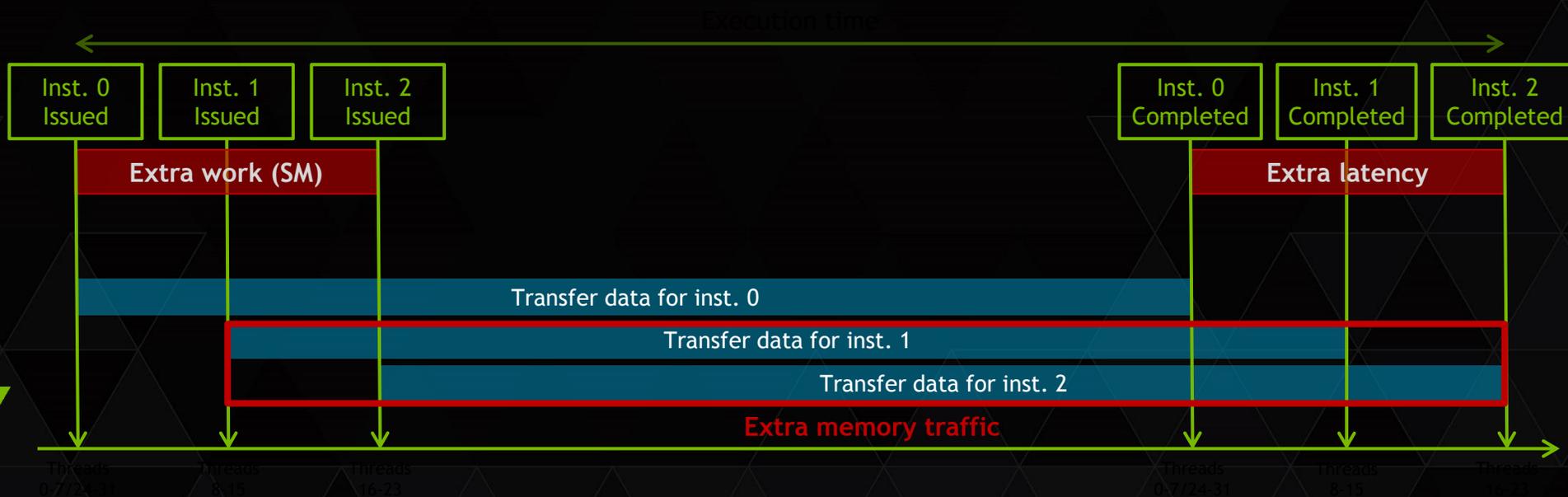


- ▶ 1 instr. executed and 2 replays = 1 request and 3 transactions





- ▶ With replays, requests take more time and use more resources
  - ▶ More instructions issued
  - ▶ More memory traffic
  - ▶ Increased execution time





# IMPROVED MEMORY ACCESS

- ▶ Blocks of size 32x2
- ▶ Memory is used more efficiently

Kernel	Time	Speedup
Original Version	1.971ms	1.00x
Better Memory Accesses	0.725ms	2.72x

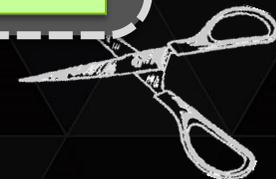
# PERF-OPT QUICK REFERENCE CARD

<b>Category:</b>	Latency Bound - Occupancy
<b>Problem:</b>	Latency is exposed due to low occupancy
<b>Goal:</b>	<b>Hide</b> latency behind more parallel work
<b>Indicators:</b>	Occupancy low (< 60%) Execution Dependency High
<b>Strategy:</b>	Increase occupancy by: <ul style="list-style-type: none"><li>• Varying block size</li><li>• Varying shared memory usage</li><li>• Varying register count</li></ul>



# PERF-OPT QUICK REFERENCE CARD

<b>Category:</b>	<b>Latency Bound - Coalescing</b>
<b>Problem:</b>	Memory is accessed inefficiently => high latency
<b>Goal:</b>	Reduce #transactions/request to reduce latency
<b>Indicators:</b>	Low global load/store efficiency, High #transactions/#request compared to ideal
<b>Strategy:</b>	Improve memory coalescing by: <ul style="list-style-type: none"><li>• Cooperative loading inside a block</li><li>• Change block layout</li><li>• Aligning data</li><li>• Changing data layout to improve locality</li></ul>



# PERF-OPT QUICK REFERENCE CARD

<b>Category:</b>	<b>Bandwidth Bound - Coalescing</b>
<b>Problem:</b>	Too much unused data clogging memory system
<b>Goal:</b>	Reduce traffic, move more <u>useful</u> data per request
<b>Indicators:</b>	Low global load/store efficiency, High #transactions/#request compared to ideal
<b>Strategy:</b>	Improve memory coalescing by: <ul style="list-style-type: none"><li>• Cooperative loading inside a block</li><li>• Change block layout</li><li>• Aligning data</li><li>• Changing data layout to improve locality</li></ul>



**GPU** TECHNOLOGY  
CONFERENCE

# ITERATION 2

## IDENTIFY HOTSPOT

Hotspot

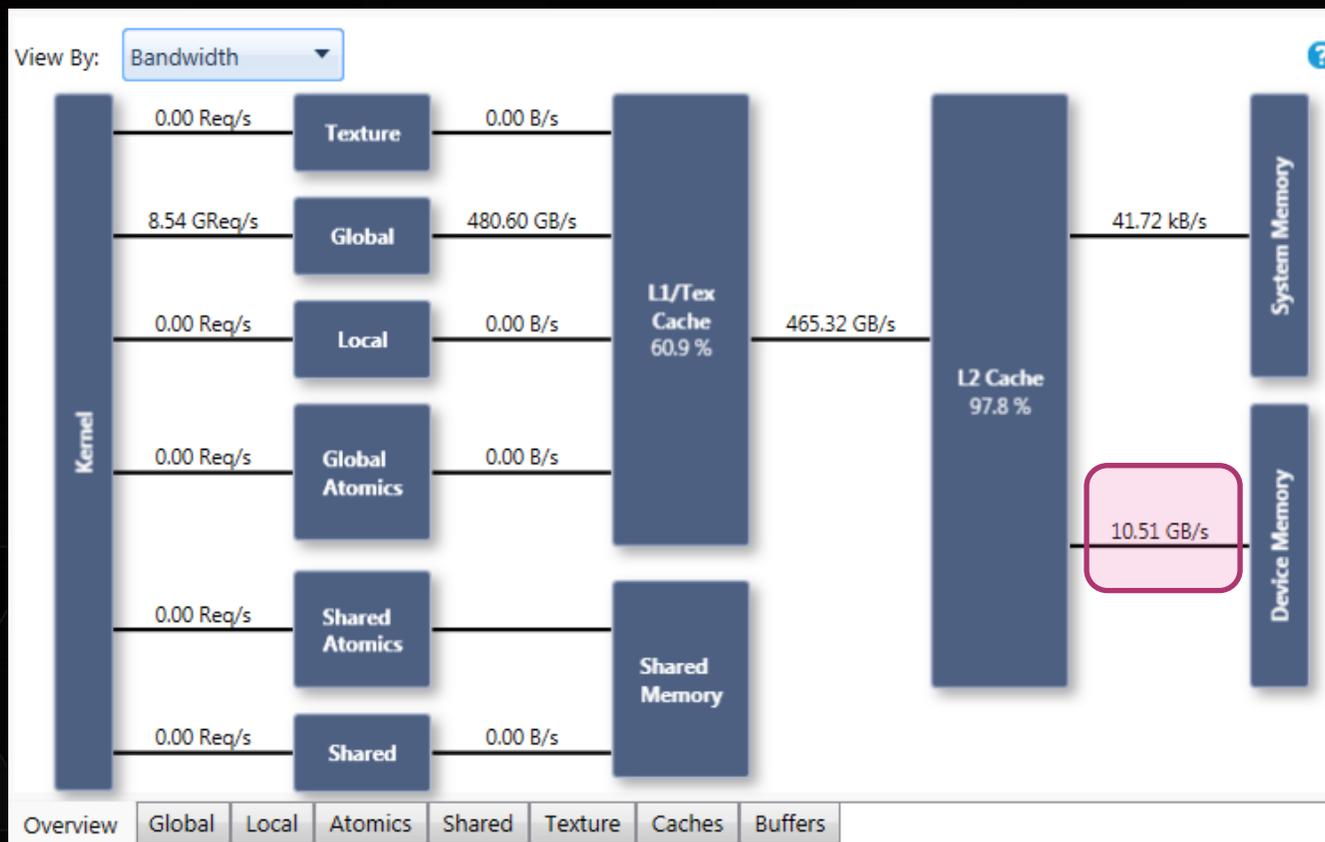
	Function Name	Start Time (μs)	Duration (μs)	Occupancy	Registers per Thread	Grid Dimensions	Block Dimensions	Static Shared Memory per Block (bytes)
1	gaussian_filter_7x7_v0	1,658,458.585	749.120	56.25 %	56	{80, 400, 1}	{32, 4, 1}	0
2	sobel_filter_3x3_v0	2,232,566.488	200.769	100.00 %	17	{80, 200, 1}	{32, 8, 1}	0
3	rgba_to_grayscale_kernel_v0	895,146.264	104.288	100.00 %	8	{80, 200, 1}	{32, 8, 1}	0

- ▶ gaussian\_filter\_7x7\_v0() still the hotspot

Kernel	Time	Speedup
Original Version	1.971ms	1.00x
Better Memory Accesses	0.725ms	2.72x

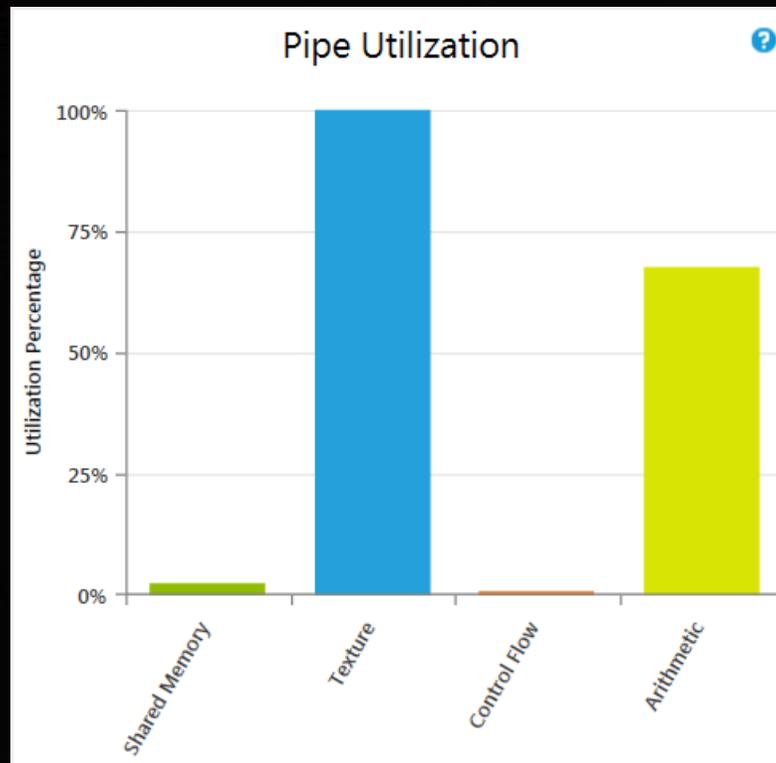
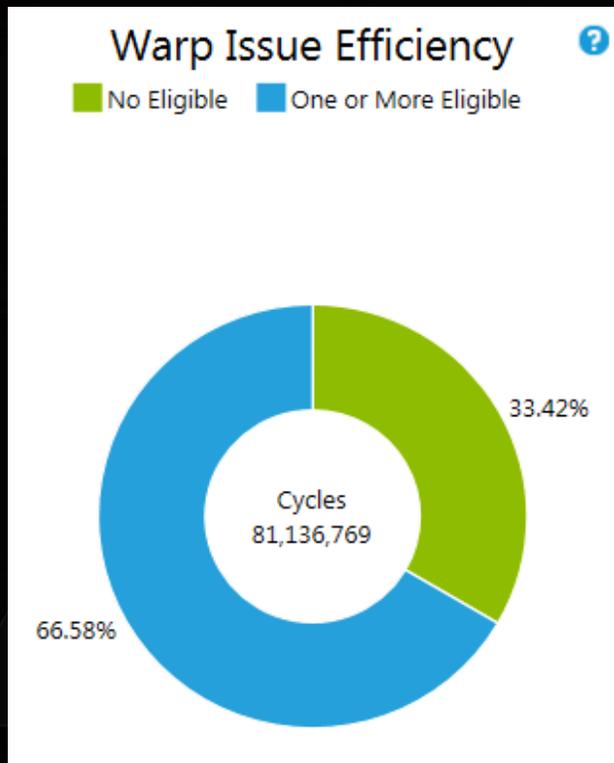
# IDENTIFY PERFORMANCE LIMITER

- ▶ Utilization of L2\$ Bandwidth (BW) limited and DRAM BW < 4%
- ▶ Not limited by memory bandwidth



# IDENTIFY PERFORMANCE LIMITER

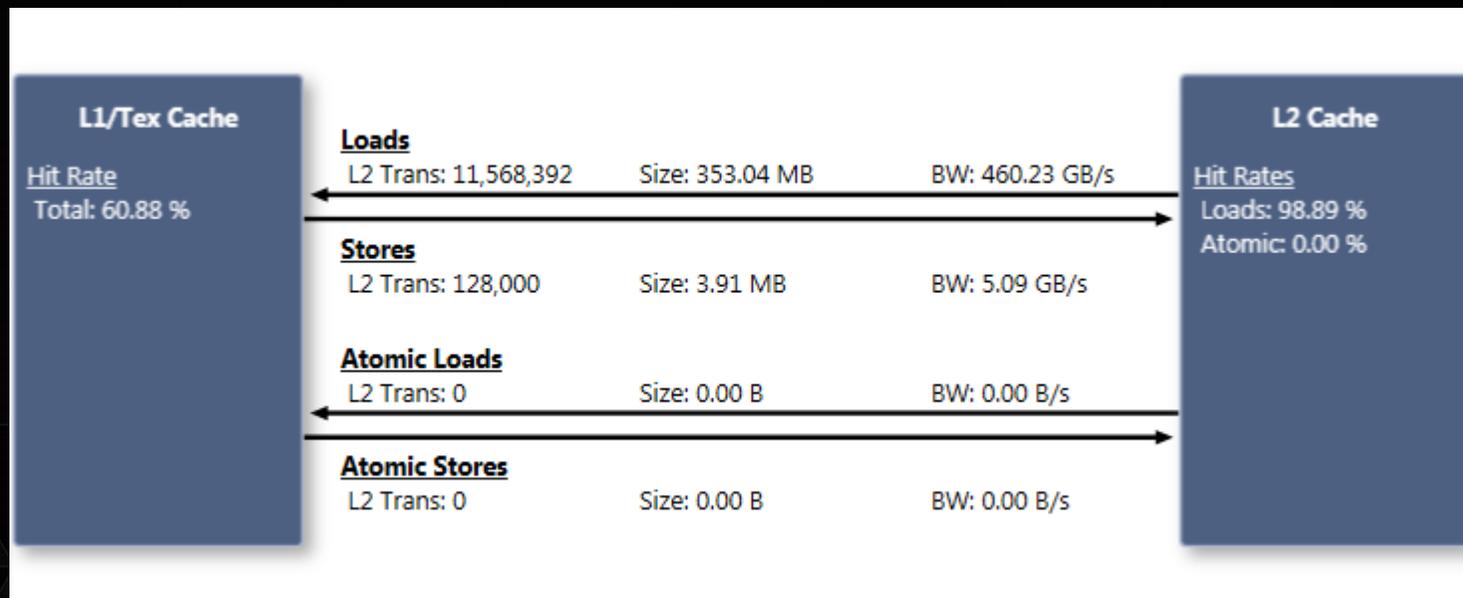
- ▶ Scheduler is starting to be busy
- ▶ but Tex pipe is clearly the limiter



⇒ Load/Store pipeline is saturated

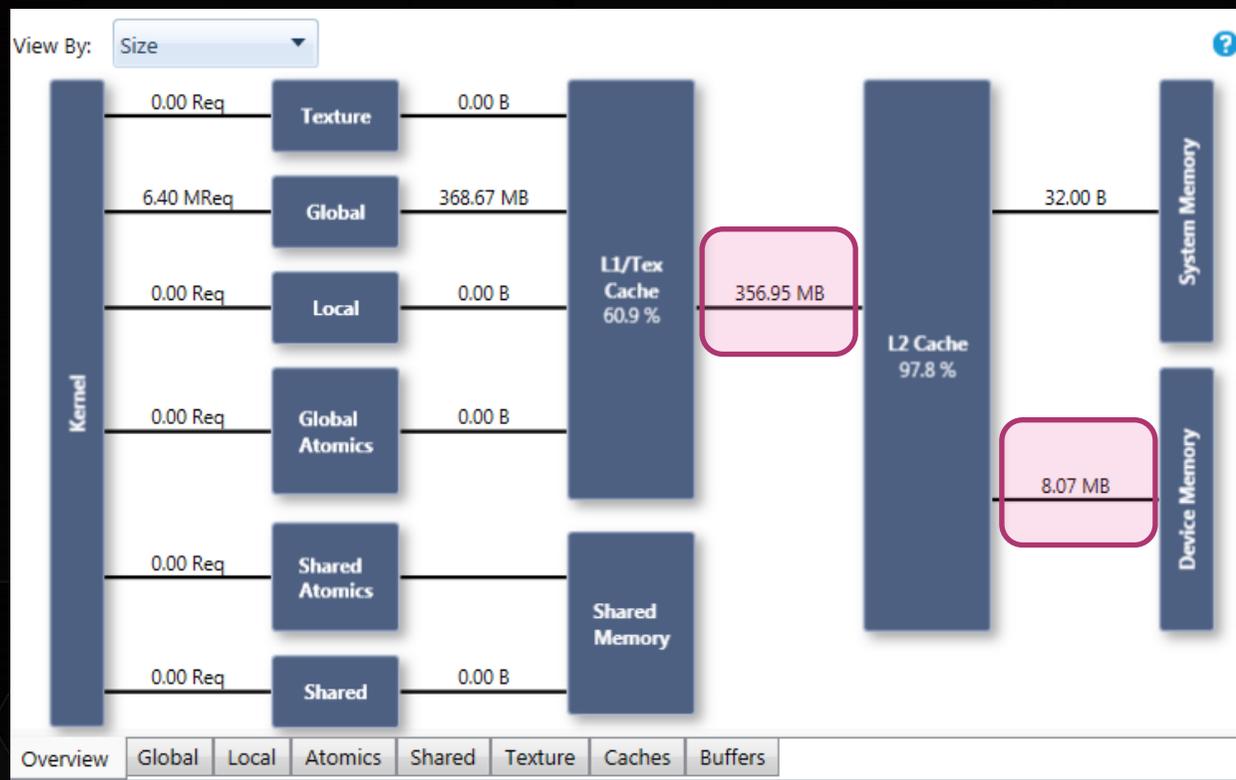
# LOOKING FOR INDICATORS

- ▶ 98.89% Hit Rate in L2 Cache
- ▶ The kernel is mostly working from the L2 cache



# LOOKING FOR MORE INDICATORS

- Kernel Transfers 8MB to/from Device Memory but 360MB to/from L2 Cache

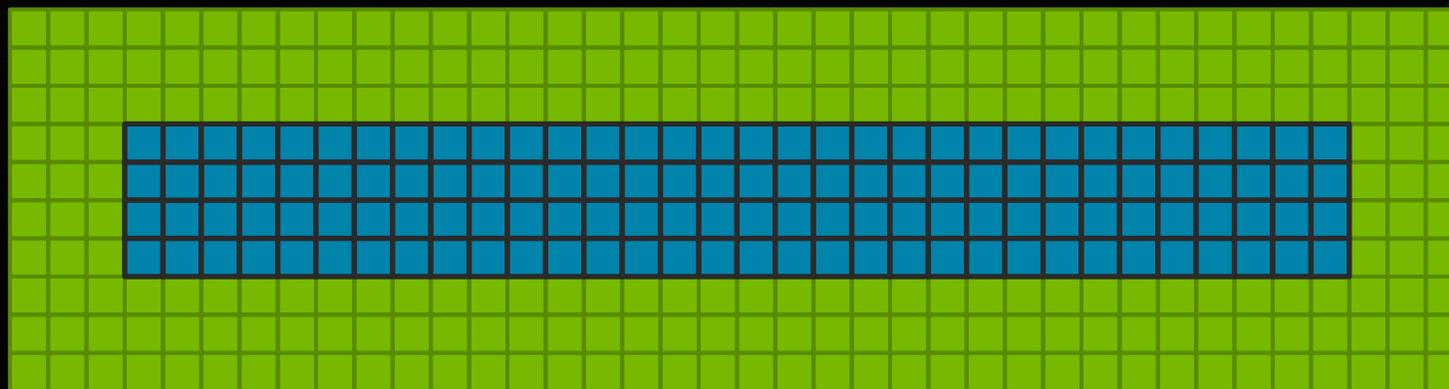


Can we move the data closer to the SM?



# SHARED MEMORY

- ▶ Adjacent pixels access similar neighbors in Gaussian Filter



- ▶ We should use shared memory to store those common pixels

```
__shared__ unsigned char smem_pixels[10][64];
```

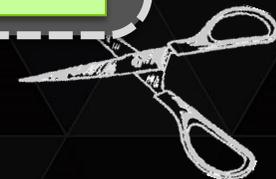
# SHARED MEMORY

- ▶ Using shared memory for the Gaussian Filter
- ▶ Significant speedup, < 0.5ms

Kernel	Time	Speedup
Original Version	1.971ms	1.00x
Better Memory Accesses	0.725ms	2.72x
Shared Memory	0.334ms	5.90x

# PERF-OPT QUICK REFERENCE CARD

Category:	Latency Bound - Shared Memory
Problem:	Long memory latencies are harder to hide
Goal:	<b><u>Reduce</u></b> latency, move data to <u>faster</u> memory
Indicators:	Shared memory not occupancy limiter High L2 hit rate Data reuse between threads and small-ish working set
Strategy:	(Cooperatively) move data to: <ul style="list-style-type: none"><li>• Shared Memory</li><li>• (or Registers)</li><li>• (or Constant Memory)</li><li>• (or Texture Cache)</li></ul>



# PERF-OPT QUICK REFERENCE CARD

<b>Category:</b>	<b>Memory Bound - Shared Memory</b>
<b>Problem:</b>	Too much data movement
<b>Goal:</b>	<u>Reduce</u> amount of data traffic to/from global mem
<b>Indicators:</b>	Higher than expected memory traffic to/from global memory Low arithmetic intensity of the kernel
<b>Strategy:</b>	(Cooperatively) move data closer to SM: <ul style="list-style-type: none"><li>• Shared Memory</li><li>• (or Registers)</li><li>• (or Constant Memory)</li><li>• (or Texture Cache)</li></ul>



**GPU** TECHNOLOGY  
CONFERENCE

# ITERATION 3

## IDENTIFY HOTSPOT

Hotspot

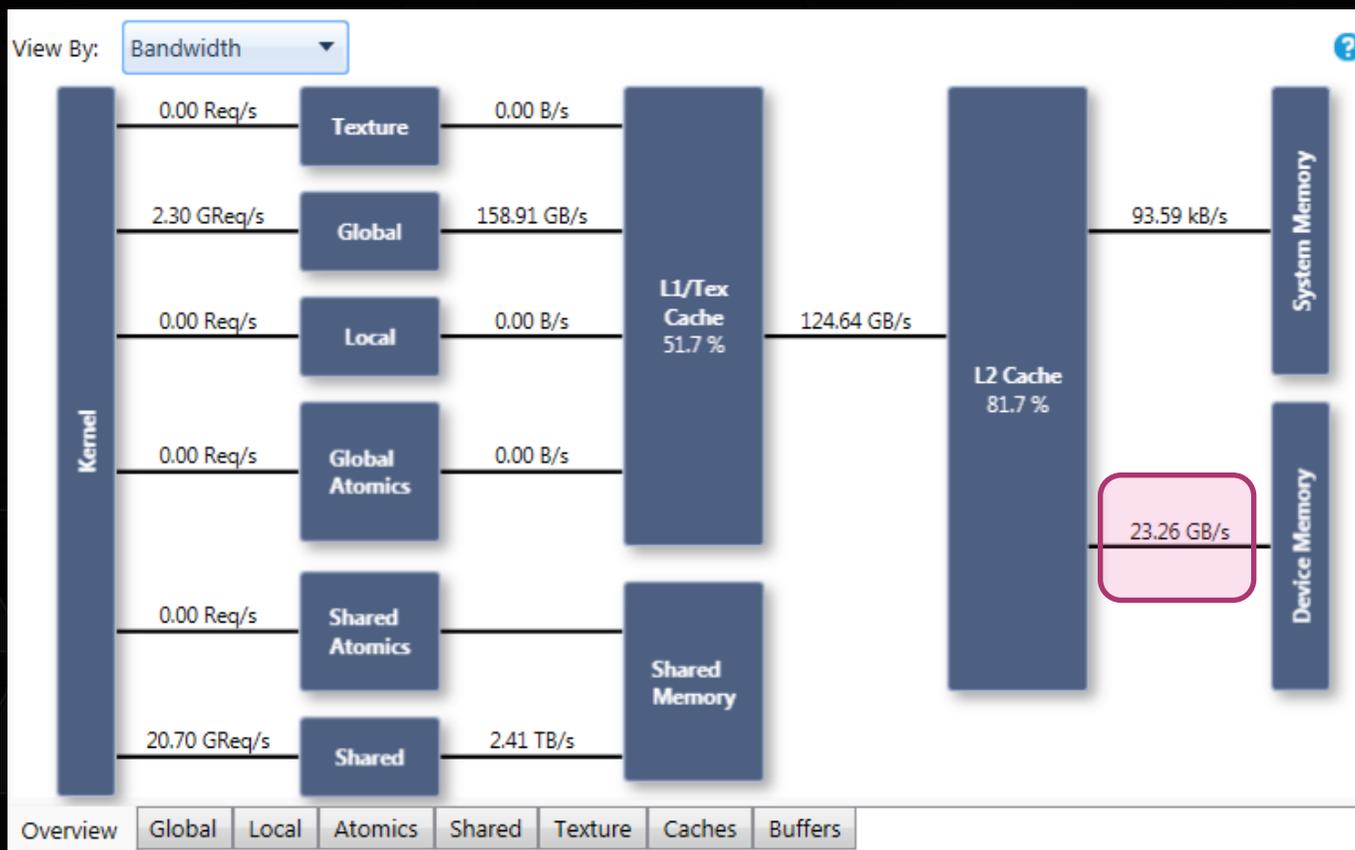
	Function Name	Start Time (μs)	Duration (μs)	Occupancy	Registers per Thread	Grid Dimensions	Block Dimensions	Static Shared Memory per Block (bytes)
1	gaussian_filter_7x7_v2	1,616,699.115	333.920	100.00 %	13	{80, 400, 1}	{32, 4, 1}	640
2	sobel_filter_3x3_v0	2,196,024.459	200.896	100.00 %	17	{80, 200, 1}	{32, 8, 1}	0
3	rgba_to_grayscale_kernel_v0	949,355.531	104.928	100.00 %	8	{80, 200, 1}	{32, 8, 1}	0

- ▶ gaussian\_filter\_7x7\_v0() still the hotspot

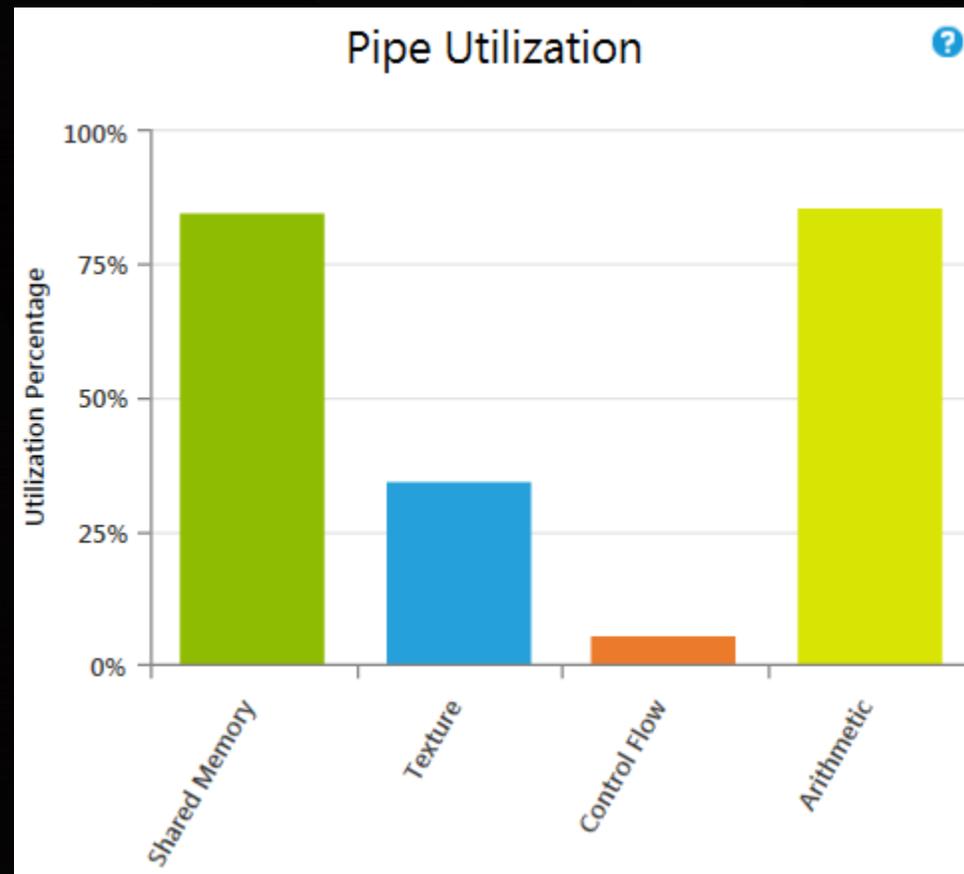
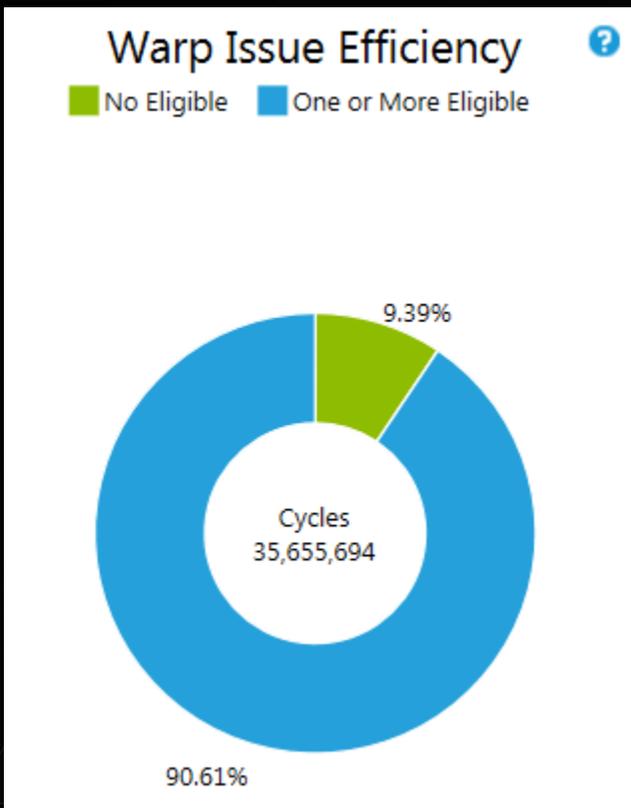
Kernel	Time	Speedup
Original Version	1.971ms	1.00x
Better Memory Accesses	0.725ms	2.72x
Shared Memory	0.334ms	5.90x

# IDENTIFY PERFORMANCE LIMITER

- ▶ Utilization of L2\$ Bandwidth (BW) moderate and DRAM BW < 8%
- ▶ Not limited by memory bandwidth



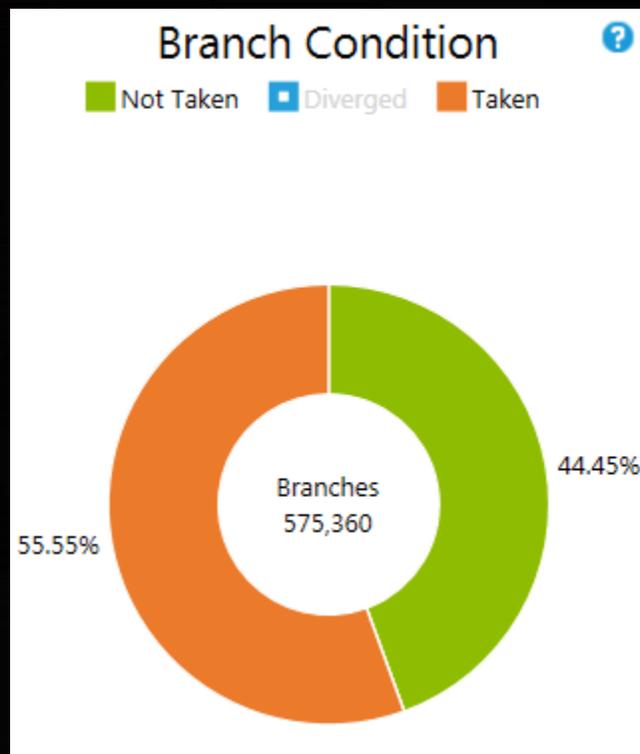
# IDENTIFY PERFORMANCE LIMITER



⇒ The Kernel is Compute Bound

# LOOKING FOR INDICATORS

- ▶ No Divergence in our code





- Threads of a warp take different branches of a conditional

```
if( threadIdx.x < 12 ) {}
```



```
else {}
```



Execution time = “if” branch + “else” branch

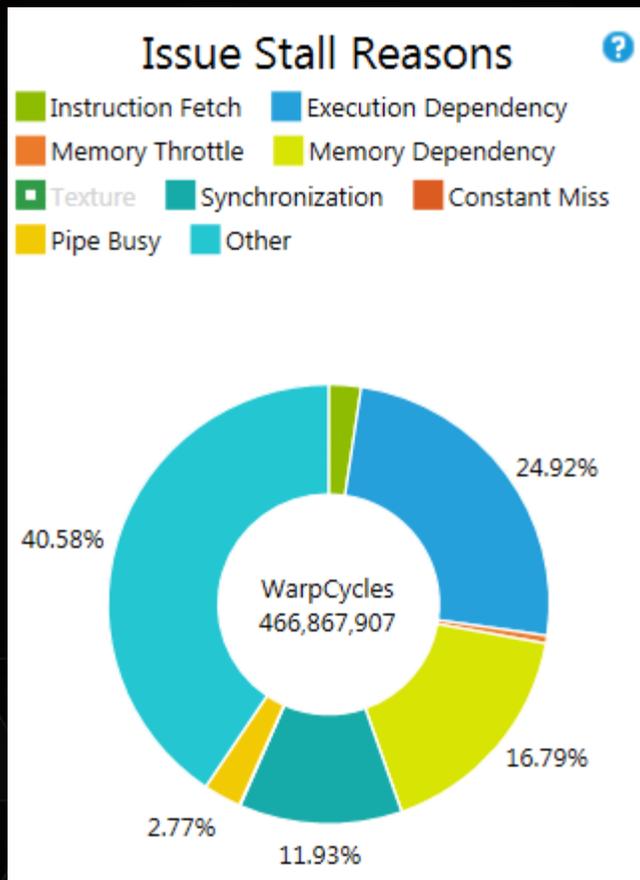


Threads execute the “if” branch

Threads execute the “else” branch

Time

# LOOKING FOR MORE INDICATORS



- ▶ Execution dependency is largest block
- ▶ Not a clear indicator however

# STALL REASONS: EXECUTION DEPENDENCY



```
a = b + c; // ADD
```

```
d = a + e; // ADD
```



```
a = b[i]; // LOAD
```

```
d = a + e; // ADD
```



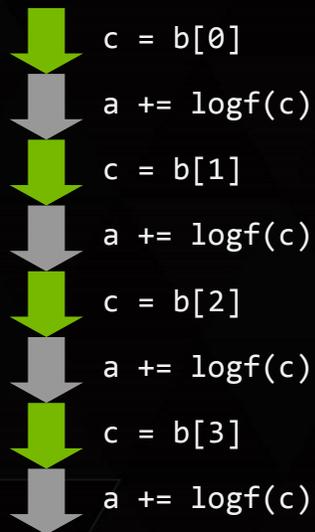
- ▶ Memory accesses may influence execution dependencies
  - ▶ Global accesses create longer dependencies than shared accesses
  - ▶ Read-only/texture dependencies are counted in Texture
- ▶ Instruction level parallelism can reduce dependencies

```
a = b + c; // Independent ADDs  
d = e + f;
```



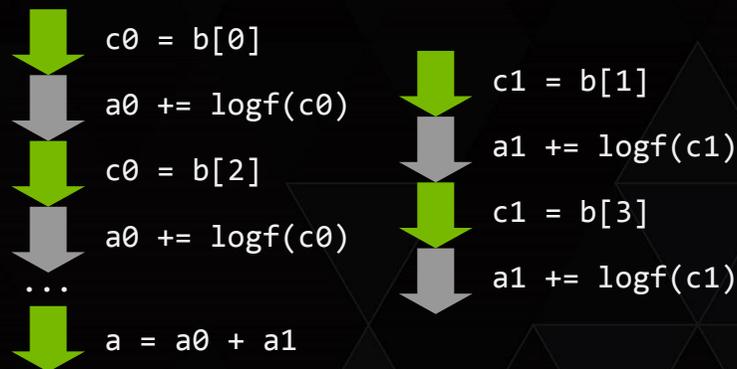
## No ILP

```
float a = 0.0f;
for( int i = 0 ; i < N ; ++i )
    a += logf(b[i]);
```



## 2-way ILP (with loop unrolling)

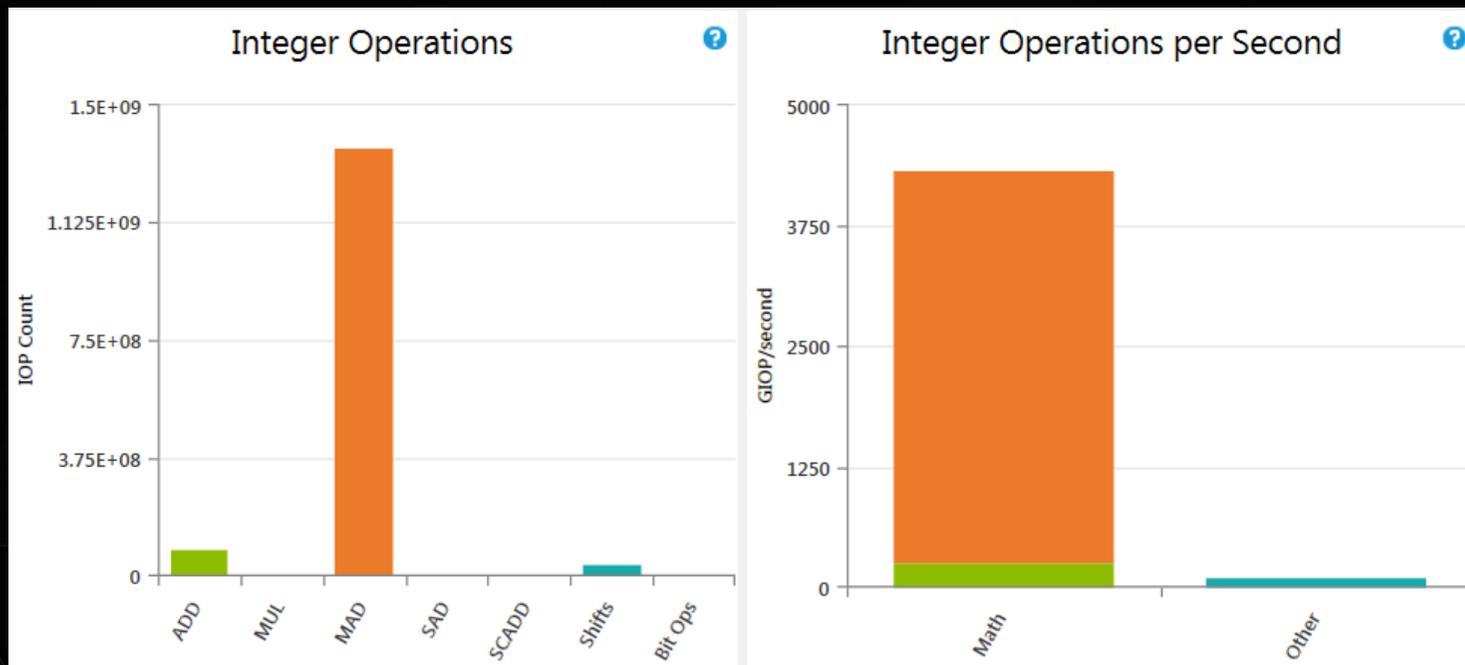
```
float a, a0 = 0.0f, a1 = 0.0f;
for( int i = 0 ; i < N ; i += 2 )
{
    a0 += logf(b[i]);
    a1 += logf(b[i+1]);
}
a = a0 + a1
```



- ▶ #pragma unroll is useful to extract ILP
- ▶ Manually rewrite code if not a simple loop

# LOOKING FOR MORE INDICATORS

▶ >4TIOP/second

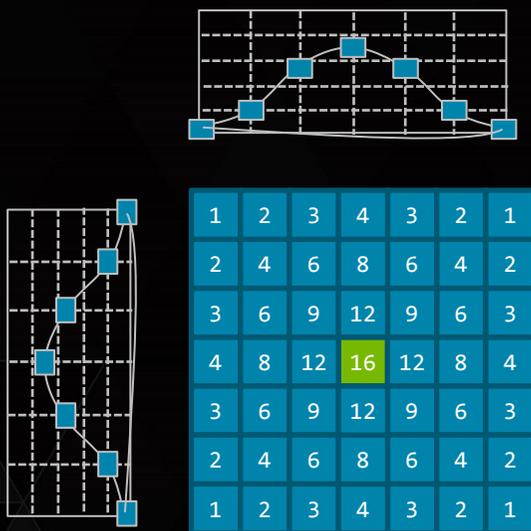


⇒ The Kernel is simply computing a lot

# REDUCING COMPUTATIONAL COMPLEXITY

## ▸ Separable Filter:

- Gaussian filters are circular and separable
- Compute horizontal and vertical convolution separately



$$[1 \ 2 \ 3 \ 4 \ 3 \ 2 \ 1] * \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 3 & 2 & 1 \\ 2 & 4 & 6 & 8 & 6 & 4 & 2 \\ 3 & 6 & 9 & 12 & 9 & 6 & 3 \\ 4 & 8 & 12 & 16 & 12 & 8 & 4 \\ 3 & 6 & 9 & 12 & 9 & 6 & 3 \\ 2 & 4 & 6 & 8 & 6 & 4 & 2 \\ 1 & 2 & 3 & 4 & 3 & 2 & 1 \end{bmatrix}$$

# SEPARABLE FILTER + INCREASED ILP

- ▶ Separable filter reduces computational load
- ▶ Processing two elements per thread increases instruction level parallelism

Kernel	Time	Speedup
Original Version	1.971ms	1.00x
Better Memory Accesses	0.725ms	2.72x
Shared Memory	0.334ms	5.90x
Separable Filter + incr. ILP	0.179ms	11.01x

# PERF-OPT QUICK REFERENCE CARD

Category:	Compute Bound - Branch Divergence
Problem:	Diverging threads
Goal:	Reduce divergence <u>within</u> warps
Indicators:	Low warp execution efficiency, high control flow utilization
Strategy:	<ul style="list-style-type: none"><li>• Refactor code to avoid intra-warp divergence</li><li>• Restructure data (sorting?) to avoid data-dependent branch divergence</li></ul>



# PERF-OPT QUICK REFERENCE CARD

<b>Category:</b>	Latency Bound - Instruction Level Parallelism
<b>Problem:</b>	Not enough independent work per thread
<b>Goal:</b>	Do more parallel work inside single threads
<b>Indicators:</b>	High execution dependency, increasing occupancy has no/little positive effect, still registers available
<b>Strategy:</b>	<ul style="list-style-type: none"><li>• Unroll loops (#pragma unroll)</li><li>• Refactor threads to compute n output values at the same time (code duplication)</li></ul>



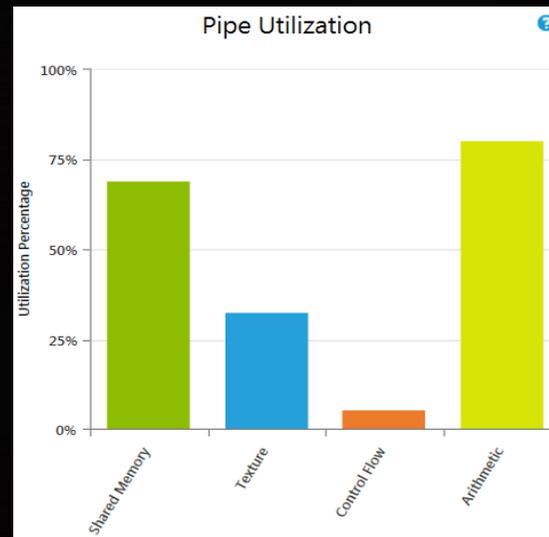
# PERF-OPT QUICK REFERENCE CARD

<b>Category:</b>	<b>Compute Bound - Algorithmic Changes</b>
<b>Problem:</b>	GPU is computing as fast as possible
<b>Goal:</b>	Reduce computation if possible
<b>Indicators:</b>	Clearly compute bound problem, speedup only with less computation
<b>Strategy:</b>	<ul style="list-style-type: none"><li>• Pre-compute or store (intermediate) results</li><li>• Trade memory for compute time</li><li>• Use a computationally less expensive algorithm</li></ul>



# THE RESULT: 11.01X

- ▶ Much better utilization



- ▶ The sobel filter is starting to become the bottleneck

	Function Name	Start Time (μs)	Duration (μs)	Occupancy	Registers per Thread	Grid Dimensions	Block Dimensions	Static Shared Memory per Block (bytes)
1	sobel_filter_3x3_v0	2,123,269.262	200.512	100.00 %	17	{80, 200, 1}	{32, 8, 1}	0
2	gaussian_filter_7x7_v5	1,537,822.349	179.361	100.00 %	30	{80, 100, 1}	{32, 8, 1}	5120
3	rgba_to_grayscale_kernel_v0	932,662.510	105.056	100.00 %	8	{80, 200, 1}	{32, 8, 1}	0

## MORE IN OUR COMPANION CODE

Kernel	Time	Speedup
Gaussian Original Version	1.971ms	1.00x
Better Memory Accesses	0.725ms	2.72x
Shared Memory	0.334ms	5.90x
Separable Filter + incr. ILP	0.179ms	11.01x
Floats instead of int ops	0.153ms	12.88x
Sobel Filter Baseline	0.200ms	1.00x
Floats+Intrinsics+fast_math	0.152ms	1.32x
Your Next Idea!		

**GPU** TECHNOLOGY  
CONFERENCE

# SUMMARY

# ITERATIVE OPTIMIZATION WITH NSIGHT VSE

- ▶ Trace the Application
- ▶ Identify the Hotspot and Profile It
- ▶ Identify the Performance Limiter
  - ▶ Memory Bandwidth
  - ▶ Instruction Throughput
  - ▶ Latency
- ▶ Look for indicators
- ▶ Reflect and Optimize the Code
- ▶ Iterate



# REFERENCES

- ▶ Performance Optimization: Programming Guidelines and GPU Architecture Details Behind Them, GTC 2013
  - ▶ <http://on-demand.gputechconf.com/gtc/2013/video/S3466-Performance-Optimization-Guidelines-GPU-Architecture-Details.mp4>
  - ▶ <http://on-demand.gputechconf.com/gtc/2013/presentations/S3466-Programming-Guidelines-GPU-Architecture.pdf>
- ▶ CUDA Best Practices Guide
  - ▶ <http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/>
- ▶ Parallel Forall devblog
  - ▶ <http://devblogs.nvidia.com/parallelforall/>
- ▶ Upcoming GTC 2015 Sessions:
  - ▶ S5655 CUDA Application Development Life Cycle with Nsight Eclipse Edition (Hands-on lab), Nikita Shulga, Thursday 2pm
  - ▶ S5353+S5376 Memory Bandwidth Bootcamp (and Beyond), Tony Scudiero, Thursday 3:30pm and 5pm

# NVIDIA REGISTERED DEVELOPER PROGRAMS

- ▶ Everything you need to develop with NVIDIA products
- ▶ Membership is your first step in establishing a working relationship with NVIDIA Engineering
  - ▶ Exclusive access to pre-releases
  - ▶ Submit bugs and features requests
  - ▶ Stay informed about latest releases and training opportunities
  - ▶ Access to exclusive downloads
  - ▶ Exclusive activities and special offers
  - ▶ Interact with other developers in the NVIDIA Developer Forums

**REGISTER FOR FREE AT: [developer.nvidia.com](https://developer.nvidia.com)**

**GPU** TECHNOLOGY  
CONFERENCE

# THANK YOU

JOIN THE CONVERSATION

#GTC15   