

GPU TECHNOLOGY
CONFERENCE

CUDA OPTIMIZATION WITH NVIDIA NSIGHT™ ECLIPSE EDITION

CHRISTOPH ANGERER, NVIDIA

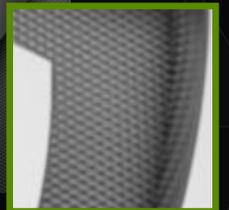
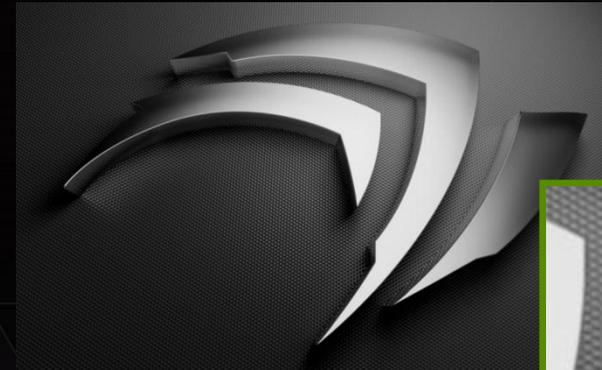
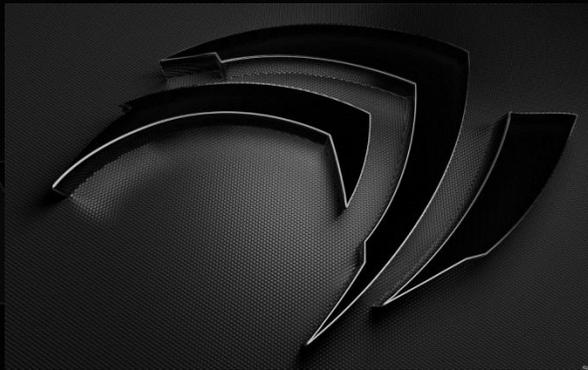
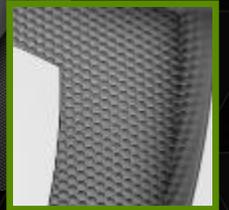
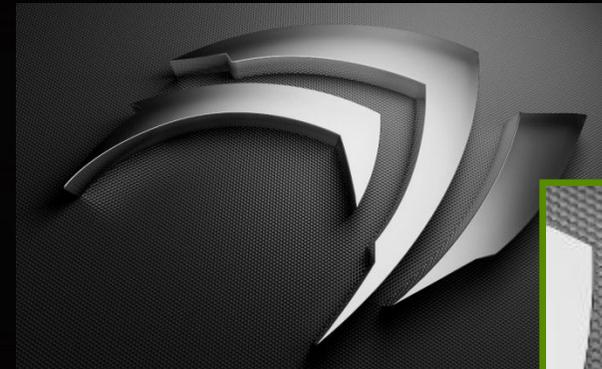
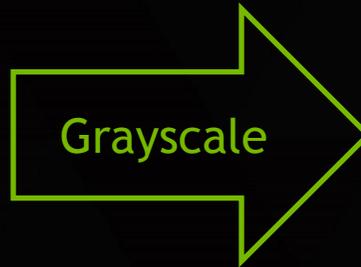
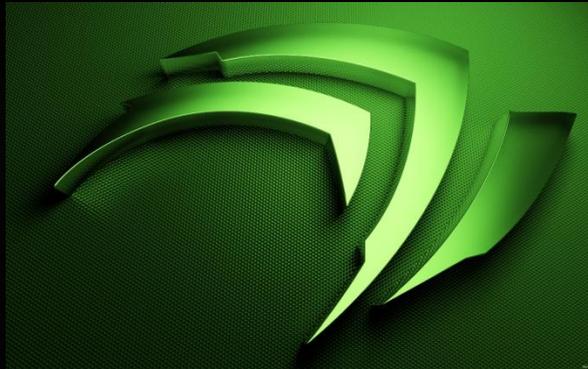
JULIEN DEMOUTH, NVIDIA

WHAT YOU WILL LEARN

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

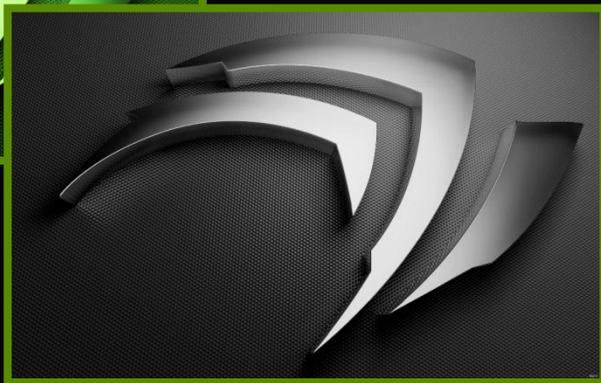
▶ Companion Code: <https://github.com/chmaruni/nsight-gtc2015>

INTRODUCING THE APPLICATION



INTRODUCING THE APPLICATION

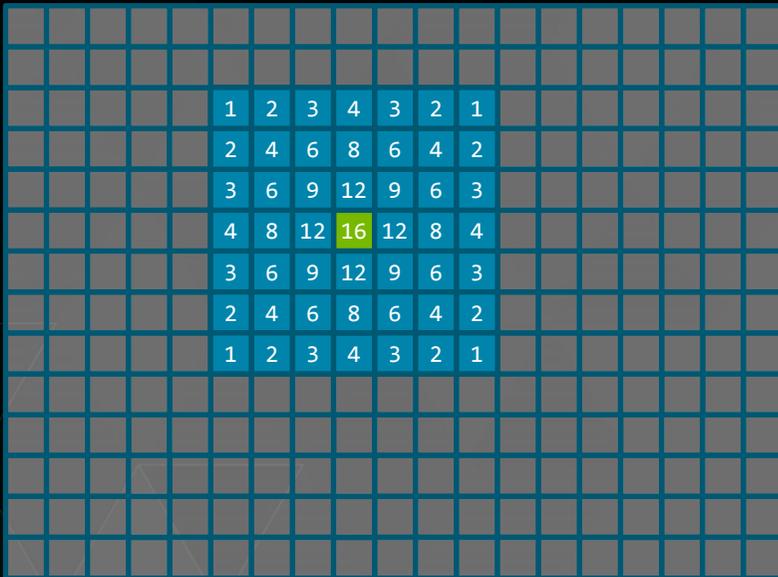
▶ Grayscale Conversion



```
// 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
```

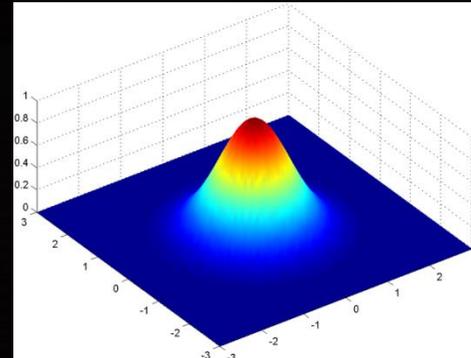
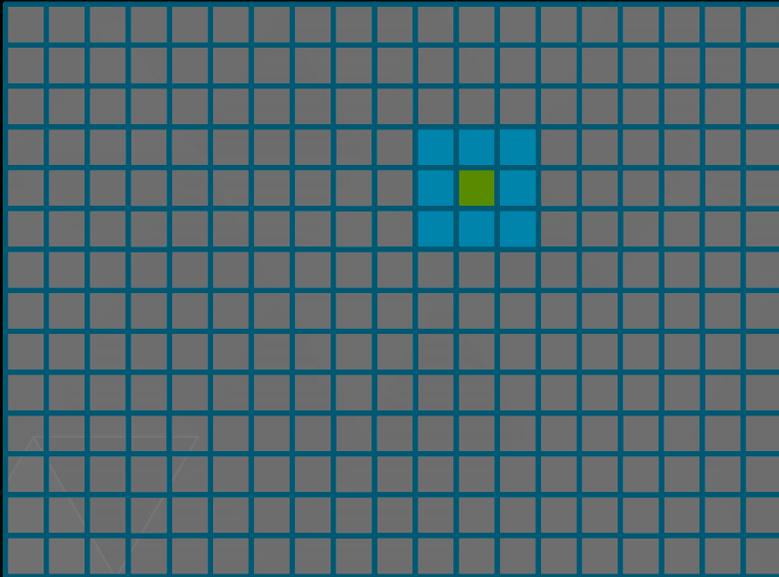


Image from Wikipedia

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 Tesla K40m
 - ▶ GK110B
 - ▶ SM3.5
 - ▶ ECC off
 - ▶ 3004 MHz memory clock, 875 MHz SM clock
- ▶ NVIDIA CUDA 7.0 release candidate
- ▶ Similar results are obtained on Windows

PERFORMANCE OPTIMIZATION CYCLE

1. Profile Application

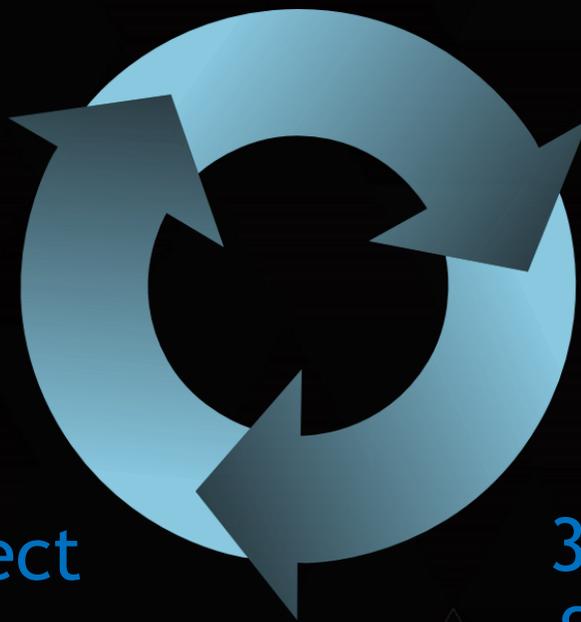
2. Identify Performance Limiter

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

CREATE A NEW NVVP SESSION

Create New Session

Executable Properties

Set executable properties

Connection: Local Manage...

Toolkit: CUDA Toolkit 7.0 (C:/Program Files/NVIDIA GPU Computing Toolkit, Manage...

File: C:\Users\cangerer\Projects\GTC2015\nsight-gtc2015-master\x64\Ste Browse...

Working directory: Enter working directory [optional] Browse...

Arguments: -no-opengl data\claw.ppm

Environment:

Name	Value

Add
Delete

< Back Next > Finish Cancel

The screenshot displays the NVIDIA Visual Profiler interface. The main window shows a timeline of GPU activity for a process named "claw.ppm". The timeline includes various events such as "cudaMemcpy", "cudaEventSynchronize", "MemCpy HtoD", and "gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned c...". The "Compute" section is highlighted, showing a list of kernels with their respective utilization percentages: "88.7% gaussian_filter_7x7...", "8.1% sobel_filter_3x3_v0(in...", and "3.2% rgba_to_grayscale_k...".

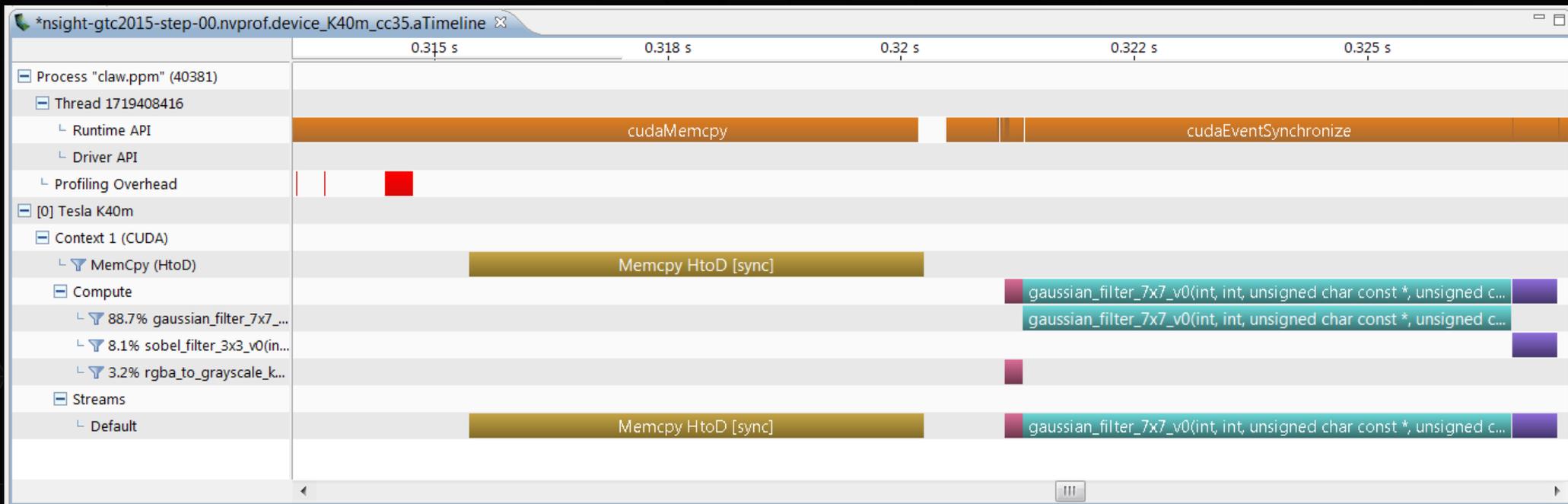
On the right side, the "Properties" panel shows a summary of the compute session:

Compute	
Duration	
Session	522.327 ms (522,326...)
Kernels	5.903 ms (5,902,576 ...)
Compute Utilization	1.1%

At the bottom left, the "Analysis" panel provides a guide for CUDA Application Analysis, including instructions on how to explore the application's performance and how to examine individual kernels.

At the bottom right, the "Results" panel displays the analysis results.

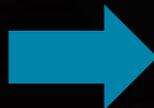
Four green boxes with white text are overlaid on the image, highlighting key features: "Timeline" (centered over the main timeline), "Summary" (centered over the Properties panel), "Guide" (centered over the Analysis panel), and "Analysis Results" (centered over the Results panel).



EXAMINE INDIVIDUAL KERNELS

(GUIDED ANALYSIS)

Launch



Analysis Details Settings Console

Export PDF Report

1. CUDA Application Analysis

The guided analysis system walks you through the various analysis stages to help you understand the optimization opportunities in your application. Once you become familiar with the optimization process, you can explore the individual analysis stages in an unguided mode. When optimizing your application it is important to fully utilize the compute and data movement capabilities of the GPU. To do this you should look at your application's overall GPU usage as well as the performance of individual kernels.

Examine GPU Usage

Determine your application's overall GPU usage. This analysis requires an application timeline, so your application will be run once to collect it if it is not already available.

Examine Individual Kernels

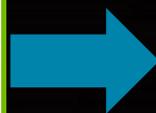
Determine which kernels are the most performance critical and that have the most opportunity for improvement. This analysis requires utilization data from every kernel, so your application will be run once to collect that data if it is not already available.

Delete Existing Analysis Information

If the application has changed since the last analysis then the existing analysis information may be stale and should be deleted before continuing.

IDENTIFY HOTSPOT

Hotspot



Results

i Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved occupancy performance compared to lower ranked kernels.

Rank	Description
100	[1 kernel instances] gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)
8	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
3	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

- Identify the hotspot: gaussian_filter_7x7_v0()

Kernel	Time	Speedup
Original Version	5.233ms	1.00x

PERFORM KERNEL ANALYSIS

NVIDIA Visual Profiler

File View Run Help

*nsight-gtc2015-step-00.nvprof.device_K40m_cc35.aTimeline

0.32 s 0.322 s 0.325 s 0.328 s 0.331 s

Profiling Overhead

[0] Tesla K40m

Context 1 (CUDA)

MemCpy (HtoD)

Compute

88.7% gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)

8.1% sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)

3.2% rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

Properties

gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)

Start	321.303 ms (321,303)
End	326.536 ms (326,536)
Duration	5.233 ms (5,232,980)
Grid Size	[320,200,1]
Block Size	[8,8,1]
Registers/Thread	51
Shared Memory/Block	0 B

Analysis Details Settings Console

Export PDF Report

1. CUDA Application Analysis

2. Performance-Critical Kernels

The results on the right show your application's kernels ordered by potential for performance improvement. Starting with the kernels with the highest ranking, you should select an entry from the table and then perform kernel analysis to discover additional optimization opportunities.

[Perform Kernel Analysis](#)

Select a kernel from the table at right or from the timeline to enable kernel analysis. This analysis requires detailed profiling data, so your application will be run once to collect that data for the kernel if it is not already available.

[Perform Additional Analysis](#)

You can collect additional information to help identify kernels with potential performance problems. After running this analysis, select any of the new results at right to highlight the individual kernels for further analysis.

Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved occupancy. Optimization of higher ranked kernels (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[1 kernel instances] gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)
8	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
3	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

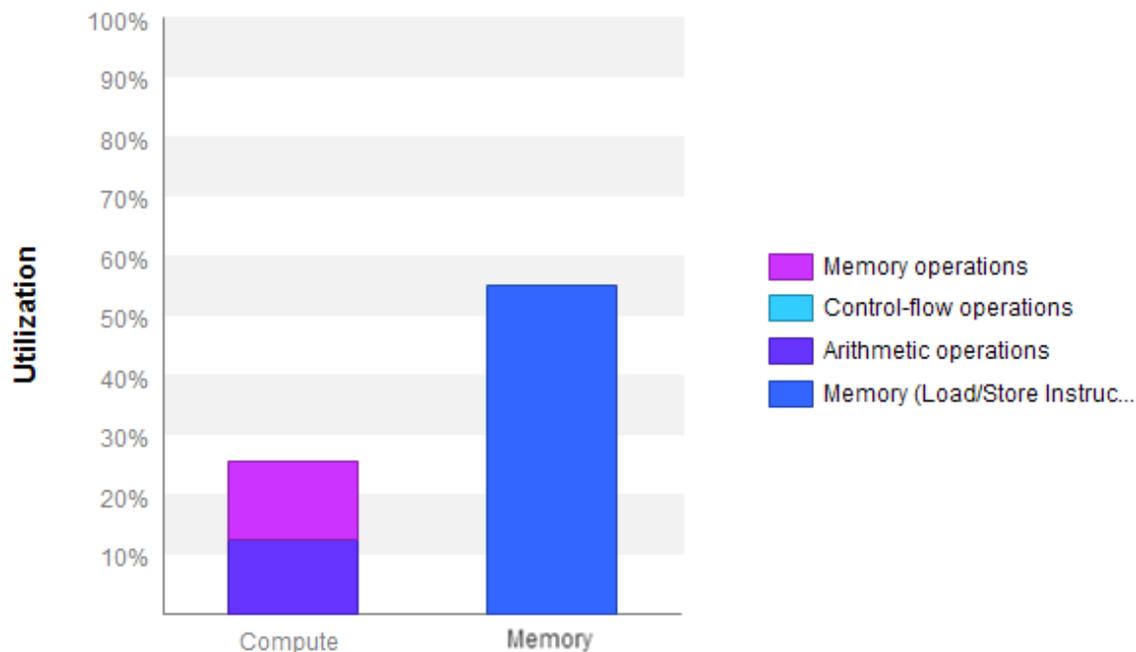
Launch

Select

Results

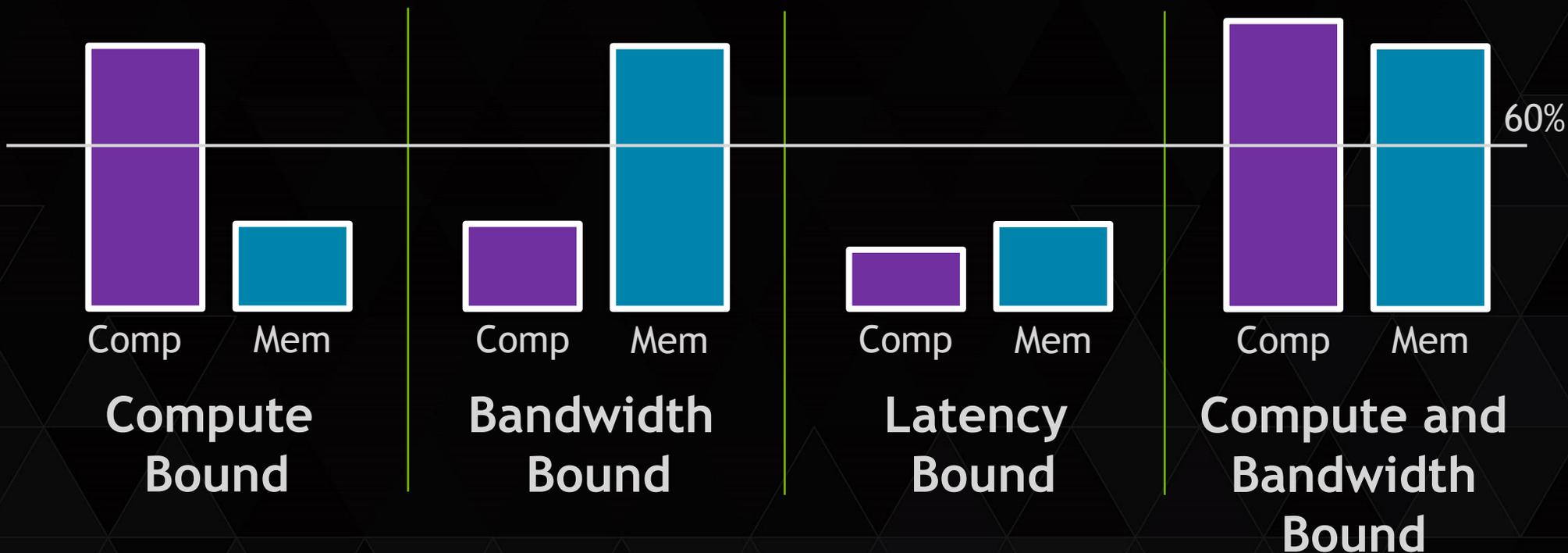
i Kernel Performance Is Bound By Instruction And Memory Latency

This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K40m". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.





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

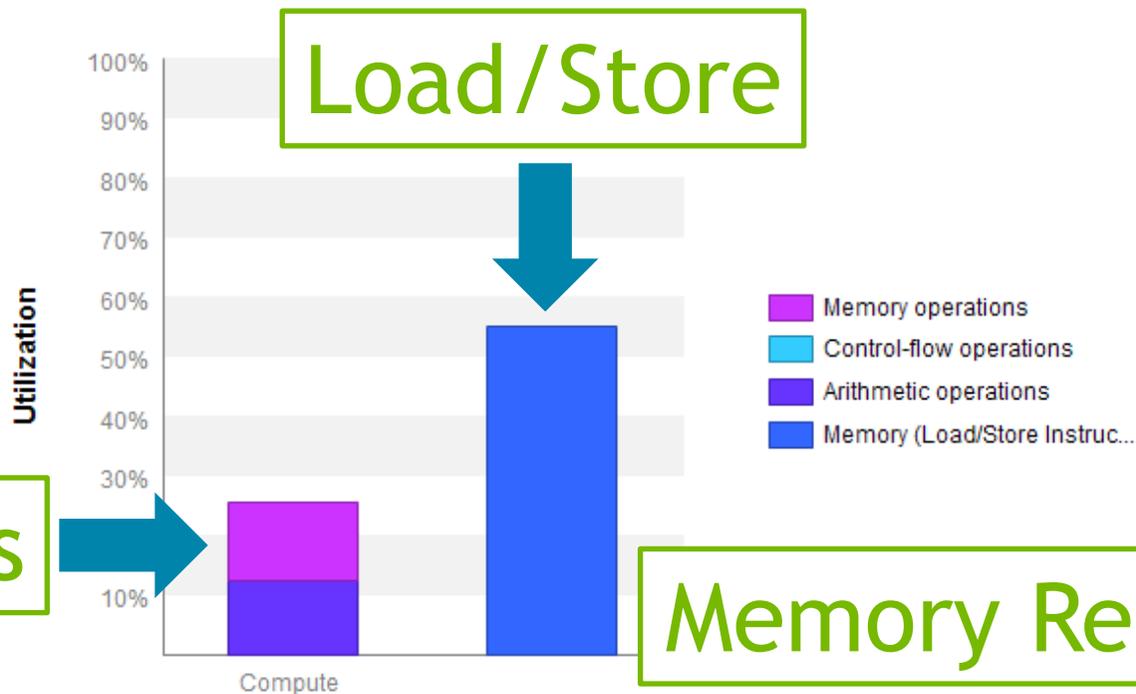


IDENTIFY PERFORMANCE LIMITER

Results

i Kernel Performance Is Bound By Instruction And Memory Latency

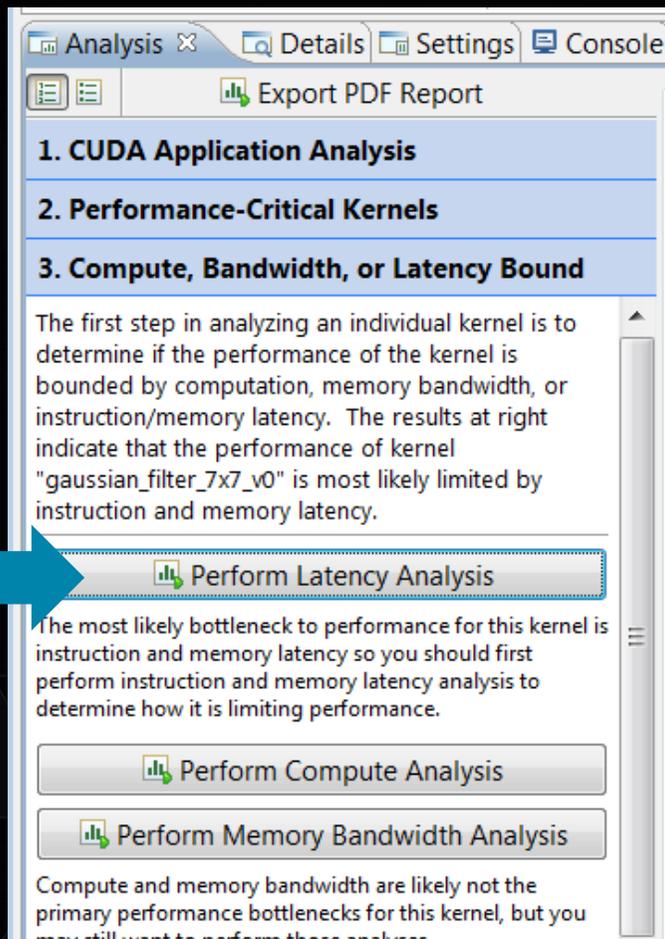
This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K40m". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.



Memory Ops

Memory Related Issues?

LOOKING FOR INDICATORS



Analysis x Details Settings Console

Export PDF Report

1. CUDA Application Analysis
2. Performance-Critical Kernels
3. Compute, Bandwidth, or Latency Bound

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "gaussian_filter_7x7_v0" is most likely limited by instruction and memory latency.

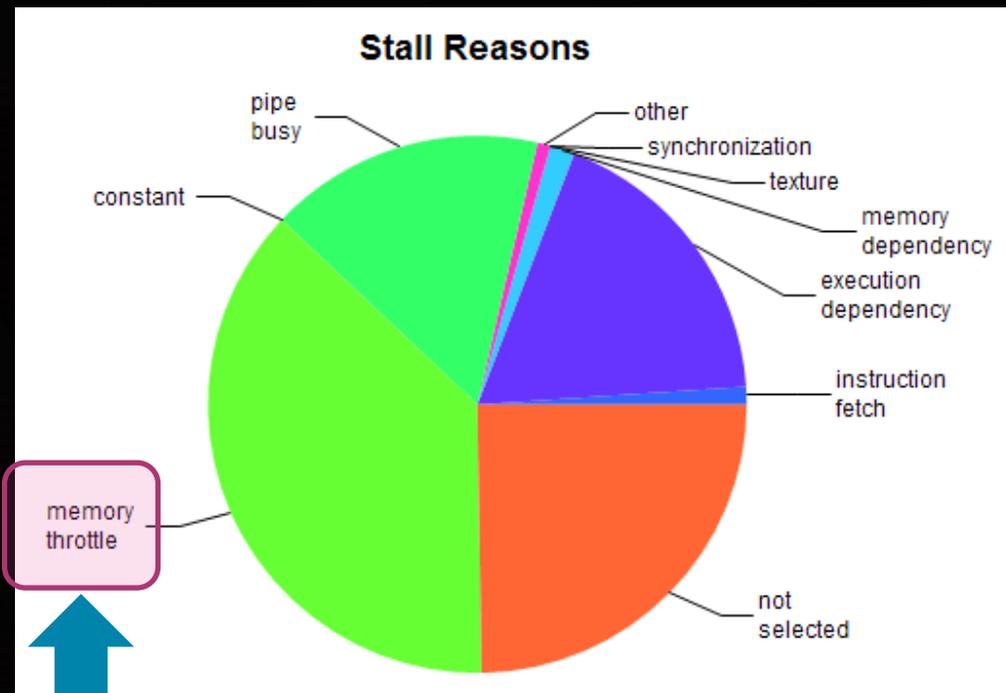
Launch → Perform Latency Analysis

The most likely bottleneck to performance for this kernel is instruction and memory latency so you should first perform instruction and memory latency analysis to determine how it is limiting performance.

Perform Compute Analysis

Perform Memory Bandwidth Analysis

Compute and memory bandwidth are likely not the primary performance bottlenecks for this kernel, but you may still want to perform these analyses.



Large number of memory operations stalling LSU

Unguided Analysis



LOOKING FOR MORE INDICATORS

```
*nsight-gtc2015-step-00.nvprof.device_K40m_cc35.aTimeline nsight-gtc2015.cu  
  
// Early exit if the thread is not in the image.  
if( !in_img(x, y, w, h) )  
    return;  
  
// Load the 48 neighbours and myself.  
int n[7][7];  
for( int j = -3 ; j <= 3 ; ++j )  
    for( int i = -3 ; i <= 3 ; ++i )  
        n[j+3][i+3] = in_img(x+i, y+j, w, h) ? (int) src[(y+j)*w + (x+i)] : 0;  
  
// Compute the convolution.
```

Global Memory Alignment and Access Pattern
Memory bandwidth is used most efficiently when each global memory load and store has proper alignment and access pattern.
Optimization: Select each entry below to open the source code to a global load or store within the kernel with an inefficient alignment access pattern. For each load or store improve the alignment and access pattern of the memory access.

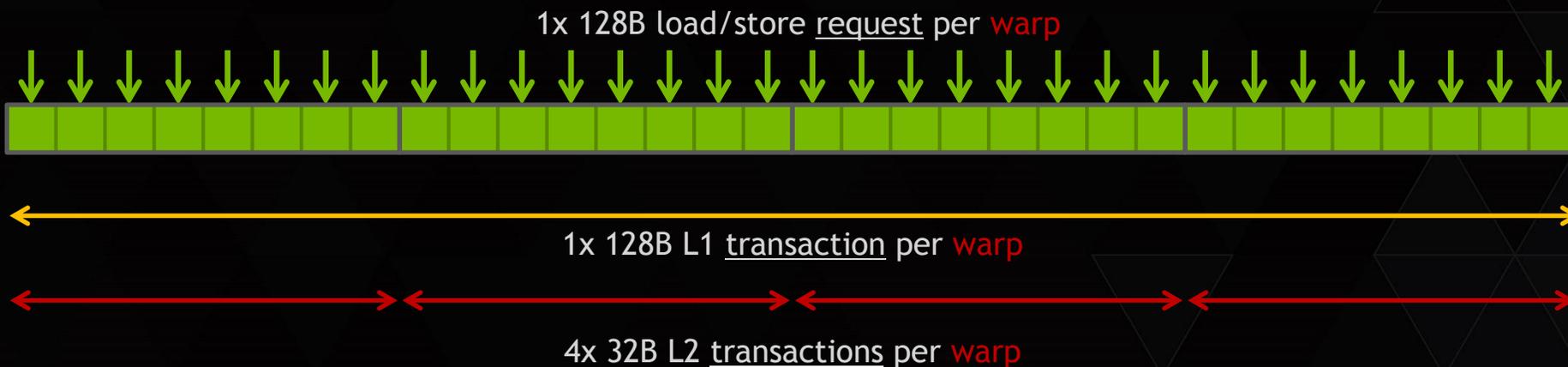
Line / File	nsight-gtc2015.cu - \home\cangerer\projects\GTC2015\nsight-gtc2015-master
243	Global Load L2 Transactions/Access = 5, Ideal Transactions/Access = 1 [637602 L2 transactions for 128000 total e
243	Global Load L2 Transactions/Access = 4, Ideal Transactions/Access = 1 [511040 L2 transactions for 128000 total e
243	Global Load L2 Transactions/Access = 4, Ideal Transactions/Access = 1 [511680 L2 transactions for 128000 total e
243	Global Load L2 Transactions/Access = 5, Ideal Transactions/Access = 1 [637203 L2 transactions for 128000 total e
243	Global Load L2 Transactions/Access = 4, Ideal Transactions/Access = 1 [511360 L2 transactions for 128000 total e
243	Global Load L2 Transactions/Access = 5, Ideal Transactions/Access = 1 [637602 L2 transactions for 128000 total e
243	Global Load L2 Transactions/Access = 5, Ideal Transactions/Access = 1 [638001 L2 transactions for 128000 total e



4-5 Global Load/Store Transactions per 1 Request



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



- ▶ 1x L1 transaction: 128B needed / 128B transferred
- ▶ 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

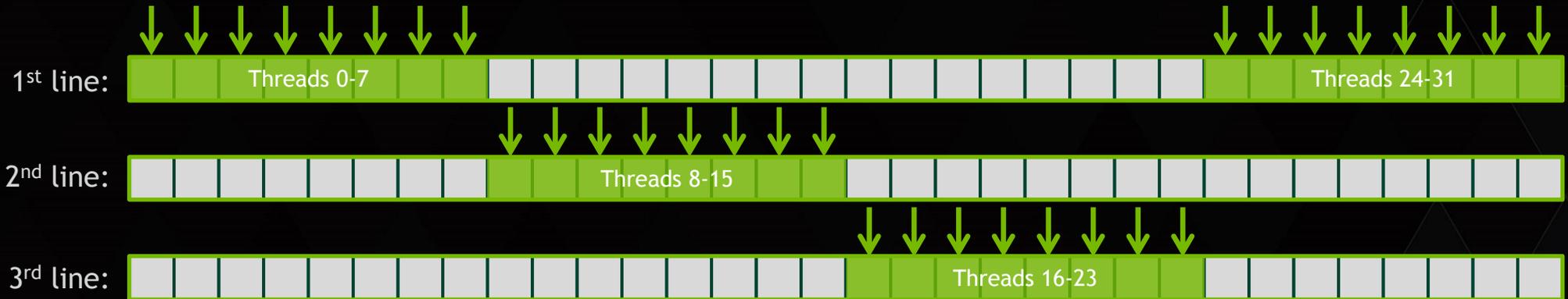
Stride: 32x4B



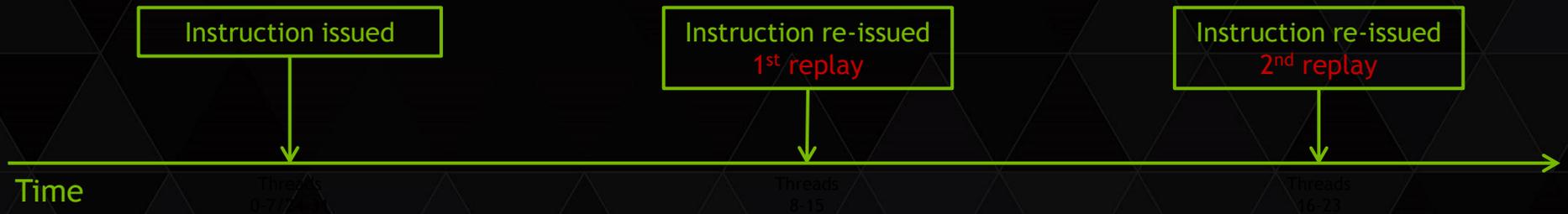
- 32x L1 transactions: 128B needed / 32x 128B transferred
- 32x L2 transactions: 128B needed / 32x 32B transferred



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

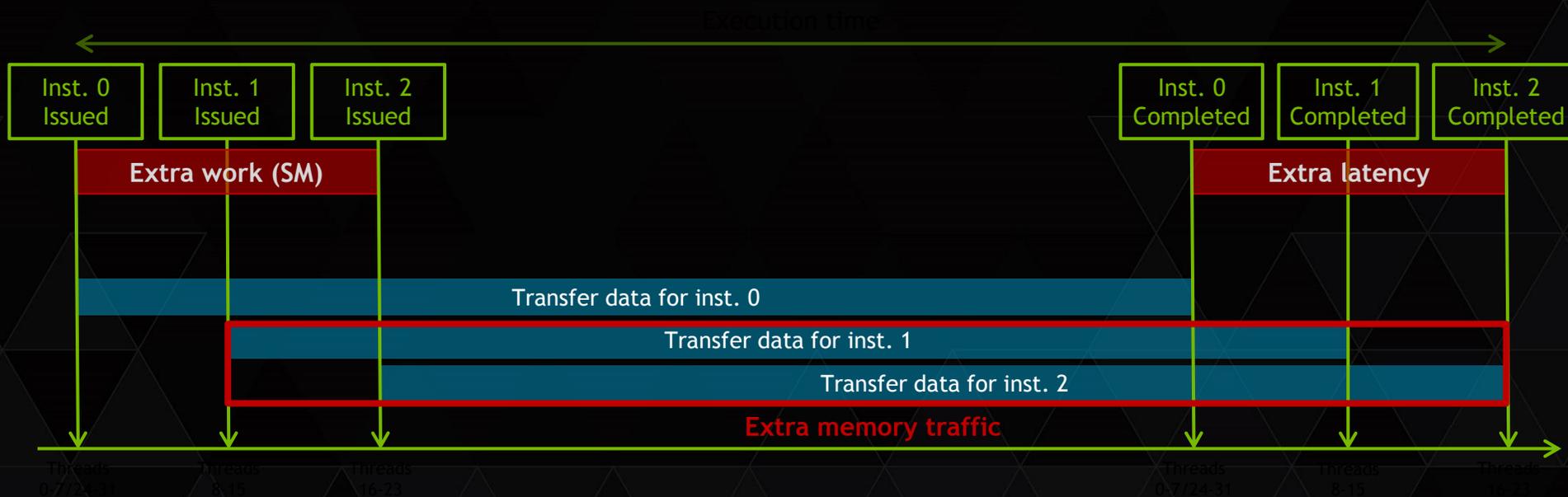


- ▶ 1 instr. executed and 2 replays = 1 request and 3 transactions





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



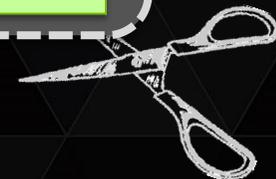
IMPROVED MEMORY ACCESS

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

Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x

PERF-OPT QUICK REFERENCE CARD

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



PERF-OPT QUICK REFERENCE CARD

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

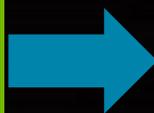


GPU TECHNOLOGY
CONFERENCE

ITERATION 2

IDENTIFY HOTSPOT

Hotspot



Results

i Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved speedup. The speedup of higher ranked kernels (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[1 kernel instances] gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)
28	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
11	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

- ▶ gaussian_filter_7x7_v0() still the hotspot

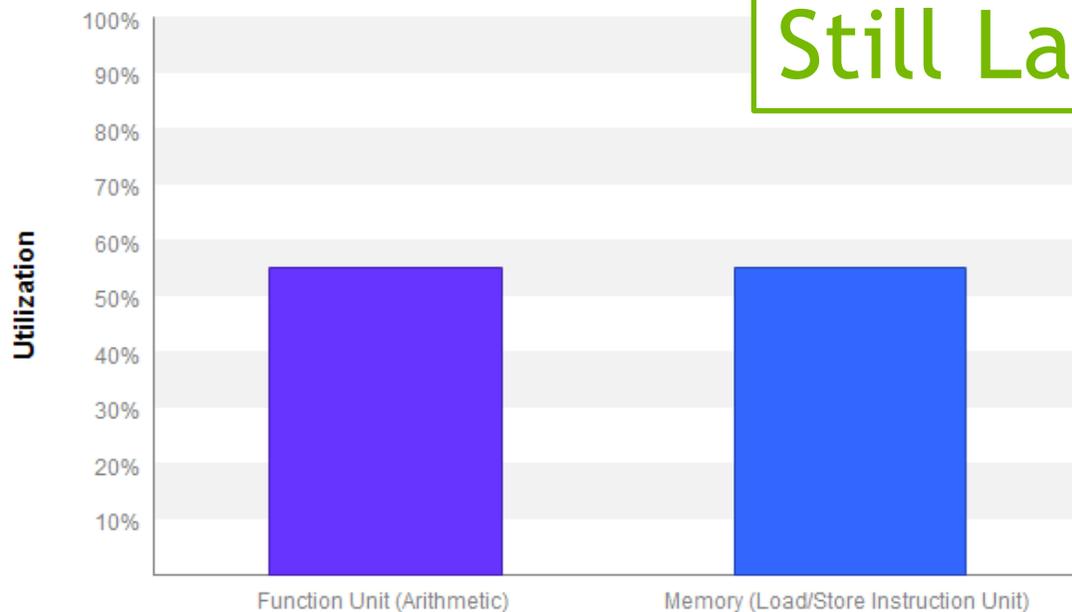
Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x

IDENTIFY PERFORMANCE LIMITER

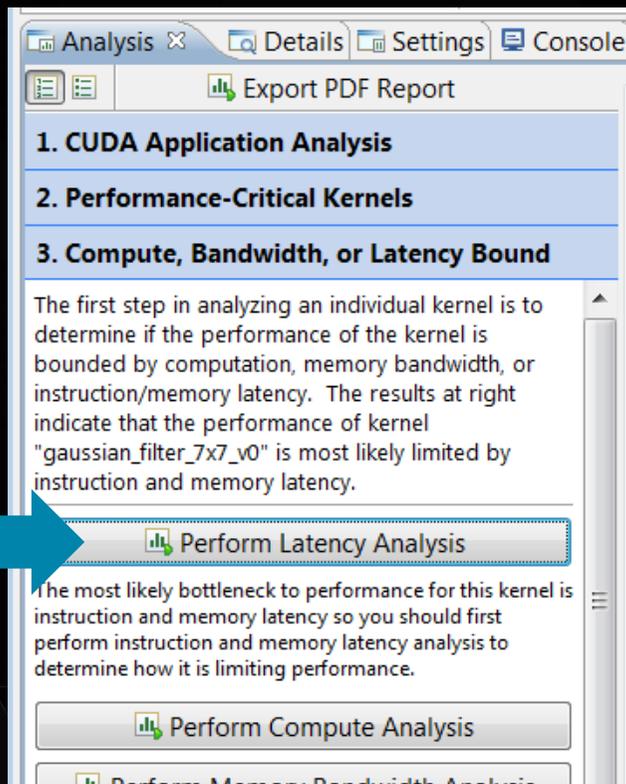
Results

i Kernel Performance Is Bound By Instruction And Memory Latency

This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K40m". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.



LOOKING FOR INDICATORS



Analysis Details Settings Console

Export PDF Report

1. CUDA Application Analysis
2. Performance-Critical Kernels
3. Compute, Bandwidth, or Latency Bound

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "gaussian_filter_7x7_v0" is most likely limited by instruction and memory latency.

Launch → Perform Latency Analysis

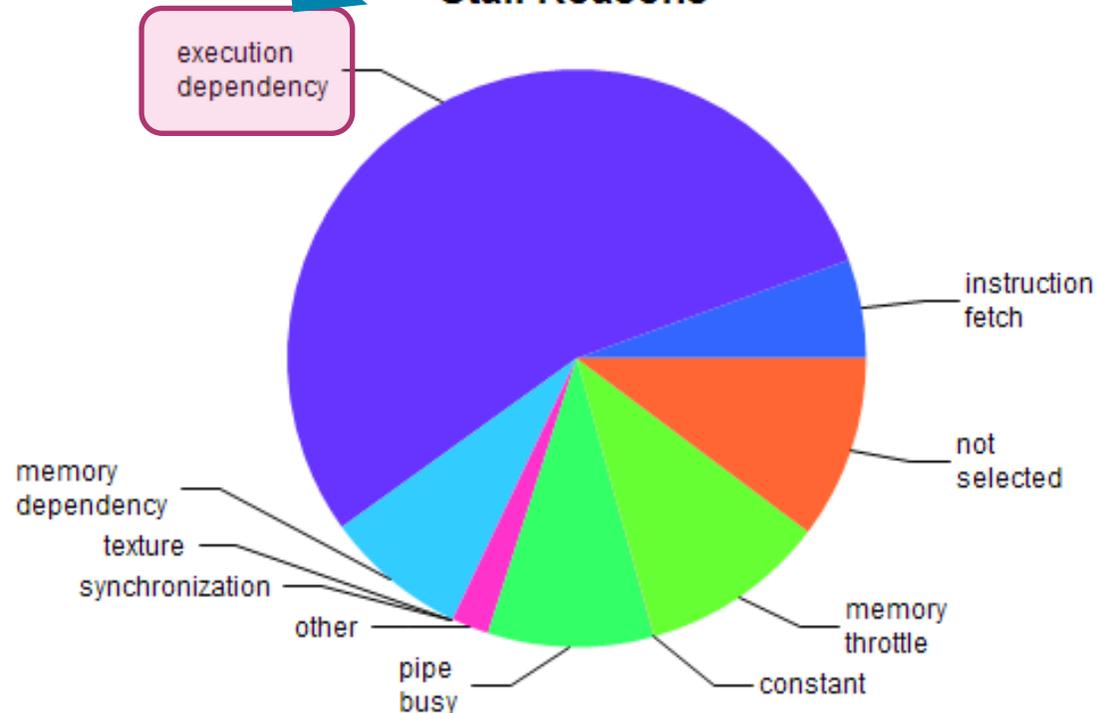
The most likely bottleneck to performance for this kernel is instruction and memory latency so you should first perform instruction and memory latency analysis to determine how it is limiting performance.

Perform Compute Analysis

Perform Memory Bandwidth Analysis

A lot of idle time

Stall Reasons



▶ Not enough work inside a thread to hide latency?

STALL REASONS: EXECUTION DEPENDENCY



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

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



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

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



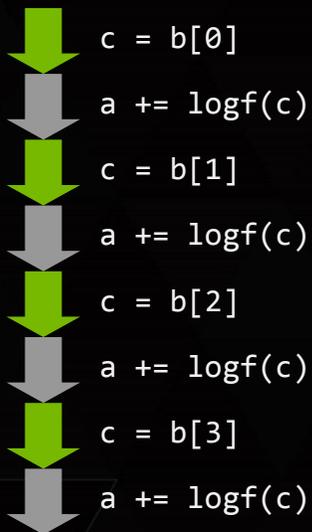
- ▶ Memory accesses may influence execution dependencies
 - ▶ Global accesses create longer dependencies than shared accesses
 - ▶ Read-only/texture dependencies are counted in Texture
- ▶ Instruction level parallelism can reduce dependencies

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



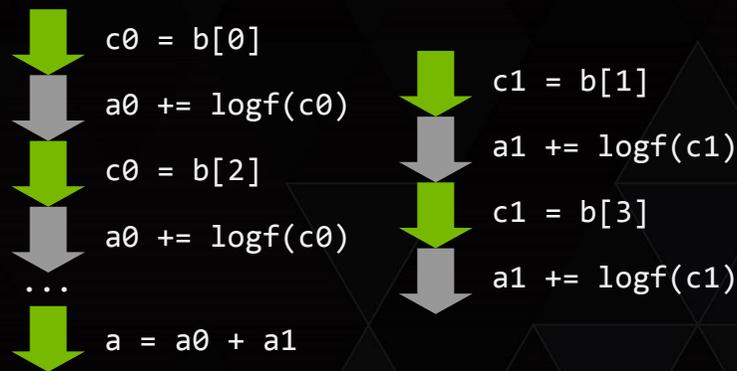
No ILP

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



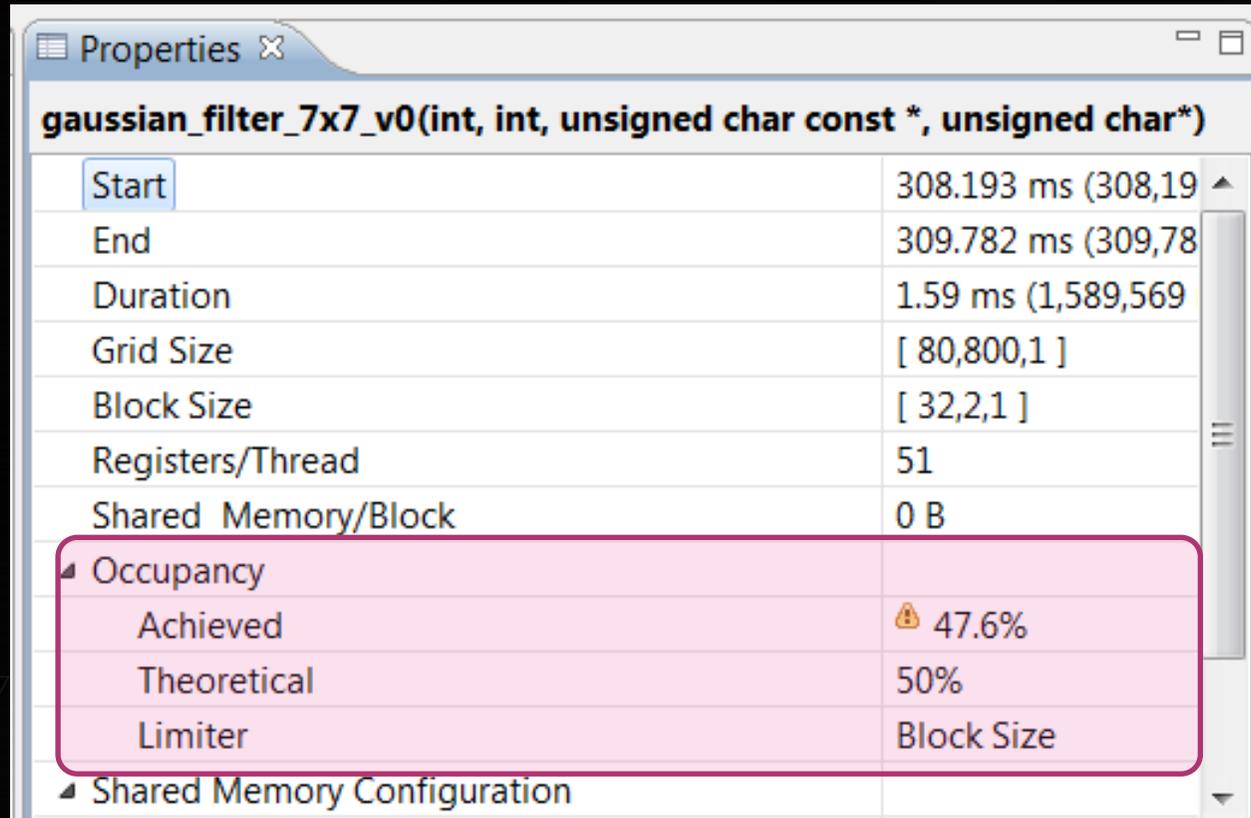
2-way ILP (with loop unrolling)

```
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



Properties ✕

gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)

Start	308.193 ms (308,19
End	309.782 ms (309,78
Duration	1.59 ms (1,589,569
Grid Size	[80,800,1]
Block Size	[32,2,1]
Registers/Thread	51
Shared Memory/Block	0 B
Occupancy	
Achieved	⚠ 47.6%
Theoretical	50%
Limiter	Block Size
Shared Memory Configuration	

LOOKING FOR MORE INDICATORS

Analysis Details Settings Console

Export PDF Report

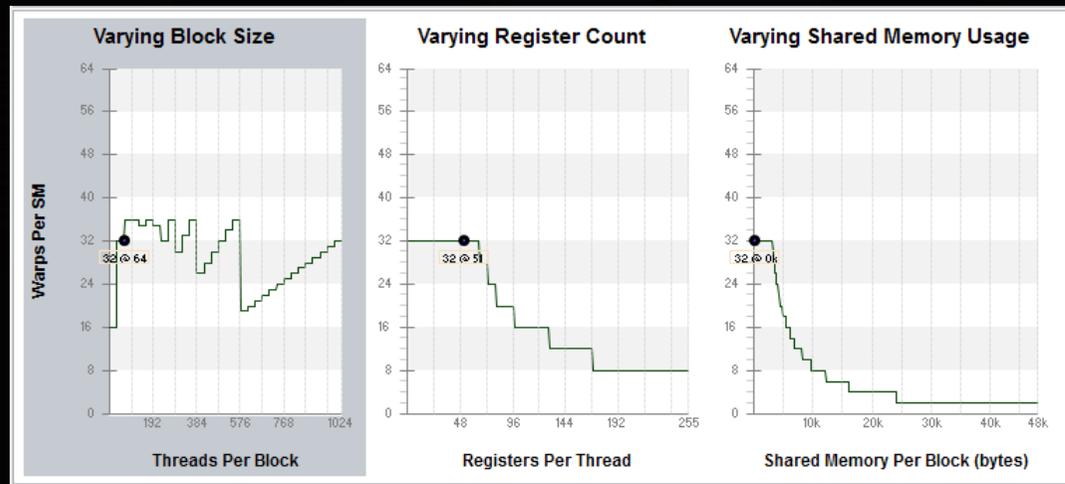
- CUDA Application Analysis
- Performance-Critical Kernels
- Compute, Bandwidth, or Latency Bound
- Instruction and Memory Latency

Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The results at right indicate that the GPU does not have enough work because instruction execution is stalling excessively.

Examine Occupancy

Occupancy is a measure of how many warps the kernel has active on the GPU, relative to the maximum number of warps supported by the GPU. Theoretical occupancy provides an upper bound while achieved occupancy

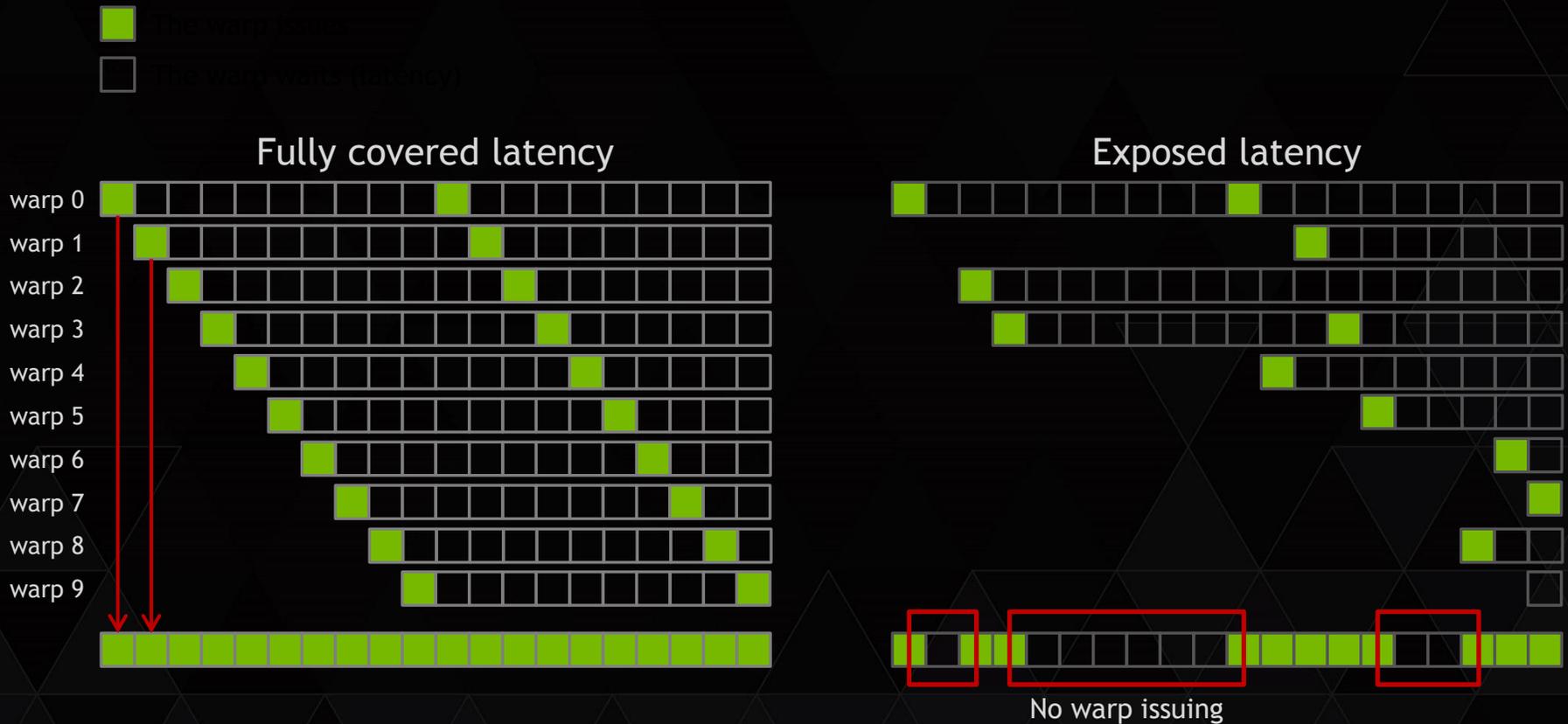
Warps				
Threads/Block		64	1024	
Warps/Block		2	32	
Block Limit		16	16	



▶ Not enough active warps to hide latencies?

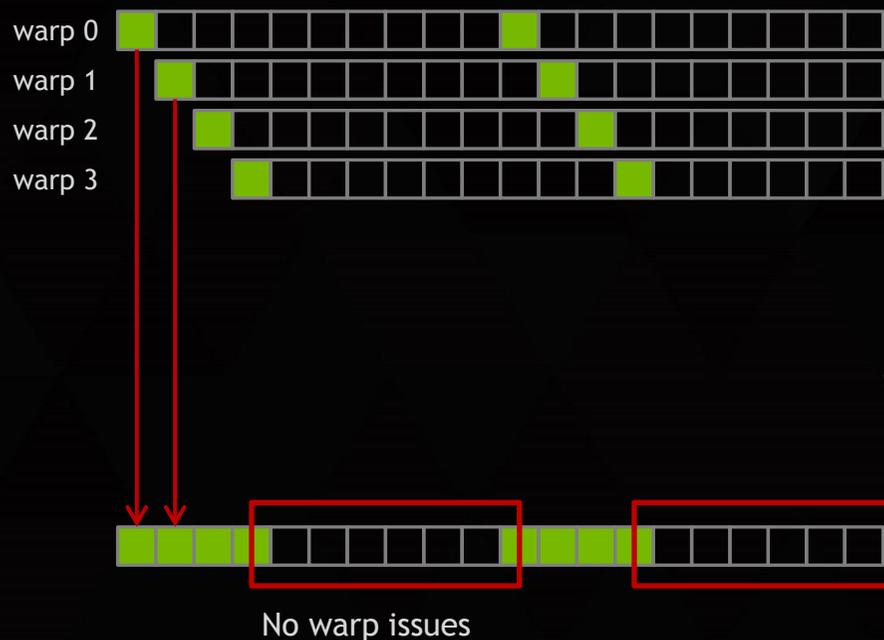


- GPU cover latencies by having a lot of work in flight





- ▶ Not enough active warps



- ▶ The schedulers cannot find eligible warps at every cycle

IMPROVED OCCUPANCY

- ▶ Bigger blocks of size 32x4
- ▶ Increases achieved occupancy slightly (from 47.6% to 52.4%)

Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x
Higher Occupancy	1.562ms	3.35x

PERF-OPT QUICK REFERENCE CARD

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



PERF-OPT QUICK REFERENCE CARD

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

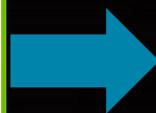


GPU TECHNOLOGY
CONFERENCE

ITERATION 3

IDENTIFY HOTSPOT

Hotspot



Results

i Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved occupancy. The kernel with the highest rank (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[1 kernel instances] gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)
30	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
12	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

- ▶ gaussian_filter_7x7_v0() still the hotspot

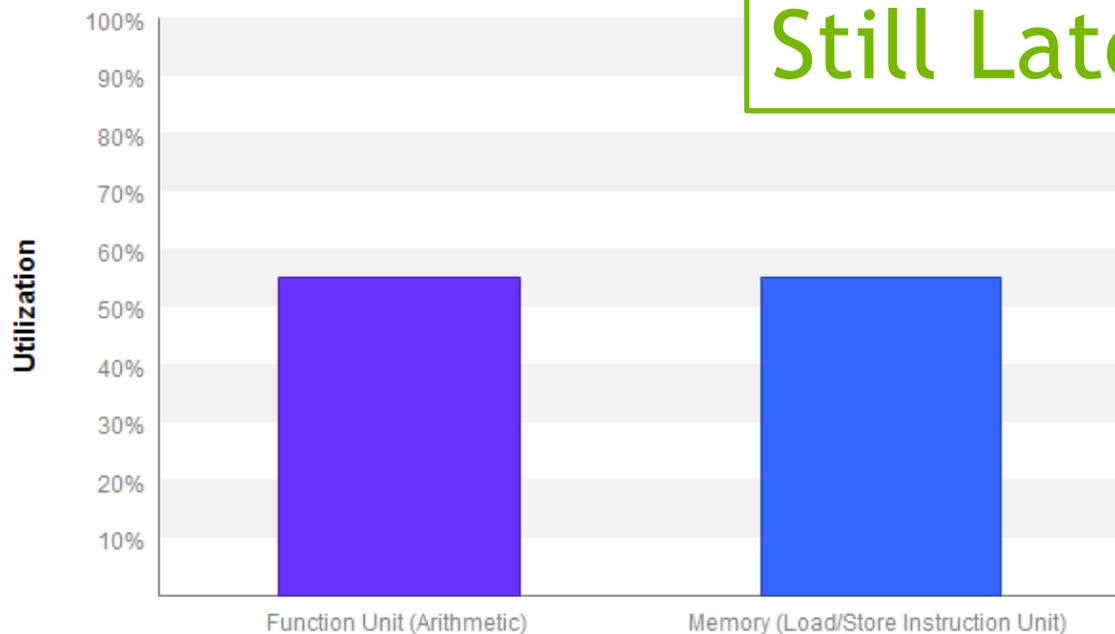
Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x
Higher Occupancy	1.562ms	3.35x

IDENTIFY PERFORMANCE LIMITER

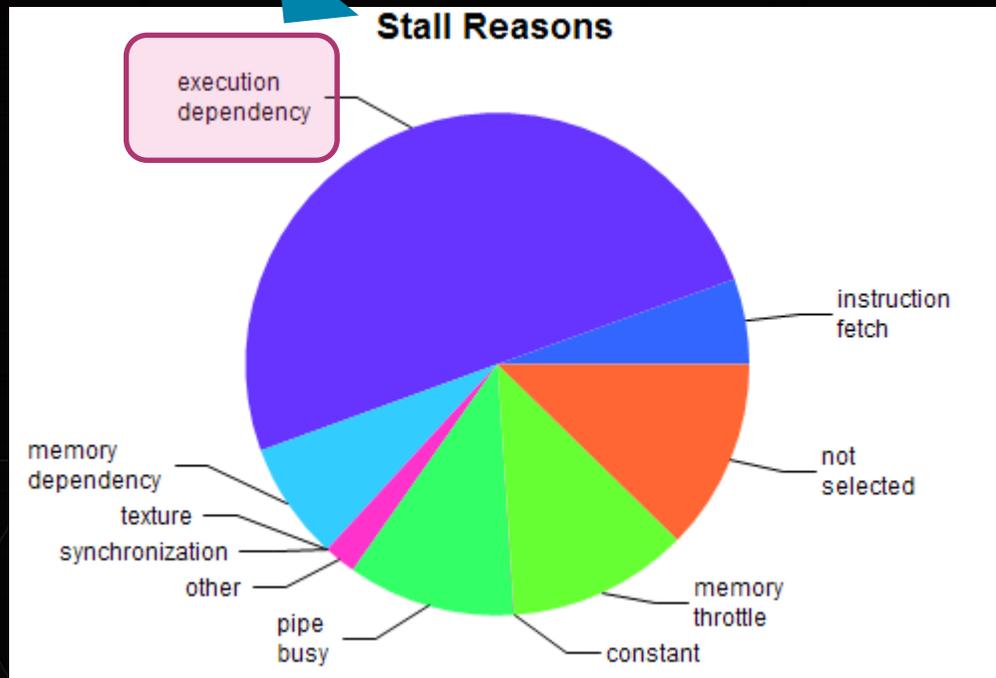
Results

i Kernel Performance Is Bound By Instruction And Memory Latency

This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K40m". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.



LOOKING FOR INDICATORS



Still high
execution
dependency,
but
occupancy OK

LOOKING FOR MORE INDICATORS

Medium L2
Bandwidth
Utilization

Analysis x Details Settings Console

Export PDF Report

1. CUDA Application Analysis
2. Performance-Critical Kernels
3. Compute, Bandwidth, or Latency Bound

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "gaussian_filter_7x7_v0" is most likely limited by instruction and memory latency.

Perform Latency Analysis

The most likely bottleneck to performance for this kernel is instruction and memory latency so you should first perform instruction and memory latency analysis to determine how it is limiting performance.

Perform Compute Analysis

Perform Memory Bandwidth Analysis

Compute and memory bandwidth are

L2 Cache		
L1 Reads	11568392	236.715 GB/s
L1 Writes	128000	2.619 GB/s
Texture Reads	0	0 B/s
Atomic	0	0 B/s
Noncoherent Reads	0	0 B/s
Total	11696392	239.333 GB/s

Texture Cache

Reads	0	0 B/s
-------	---	-------

Device Memory

Reads	128545	2.63 GB/s
Writes	128002	2.619 GB/s
Total	256547	5.249 GB/s

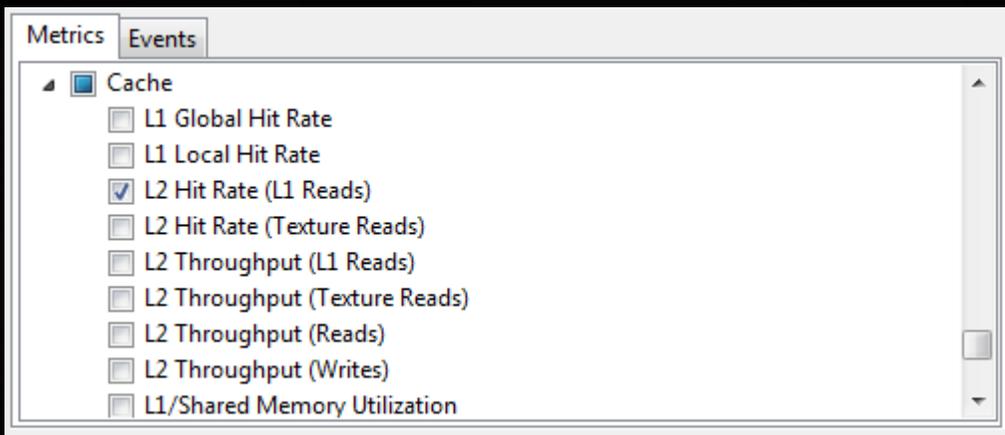
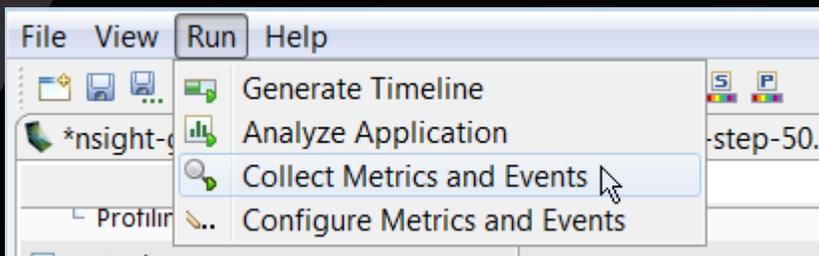


Launch

Very low device memory bandwidth utilization

Is our working set mostly in L2\$?

CHECKING L2 HIT RATE: 98.9%

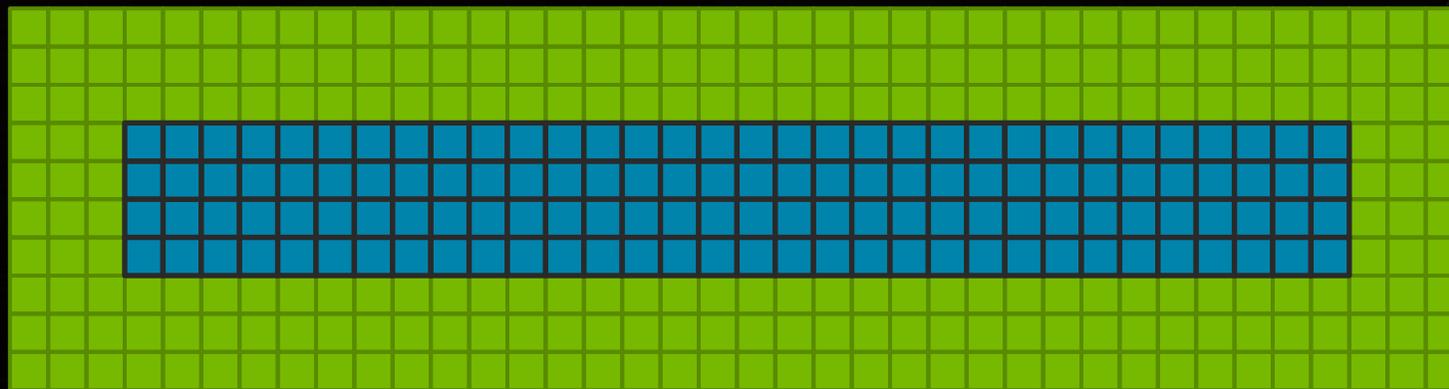


Name	Duration	L2 Hit Rate (L1 Reads)
gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)	3.606 ms	98.9%

**Our working set is mostly in L2\$
Can we move it even closer?**



- ▶ 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];
```

- ▶ Using shared memory for the Gaussian Filter
- ▶ Significant speedup, < 1ms

Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x
Higher Occupancy	1.562ms	3.35x
Shared Memory	0.911ms	5.74x

PERF-OPT QUICK REFERENCE CARD

Category:	Latency Bound - Shared Memory
Problem:	Long memory latencies are difficult to hide
Goal:	<u>Reduce</u> 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">• Shared Memory• (or Registers)• (or Constant Memory)• (or Texture Cache)



PERF-OPT QUICK REFERENCE CARD

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



GPU TECHNOLOGY
CONFERENCE

ITERATION 4

IDENTIFY HOTSPOT

Hotspot



Results

i Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved speedup. Higher ranked kernels (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[1 kernel instances] gaussian_filter_7x7_v2(int, int, unsigned char const *, unsigned char*)
52	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
20	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

- ▶ gaussian_filter_7x7_v0() still the hotspot

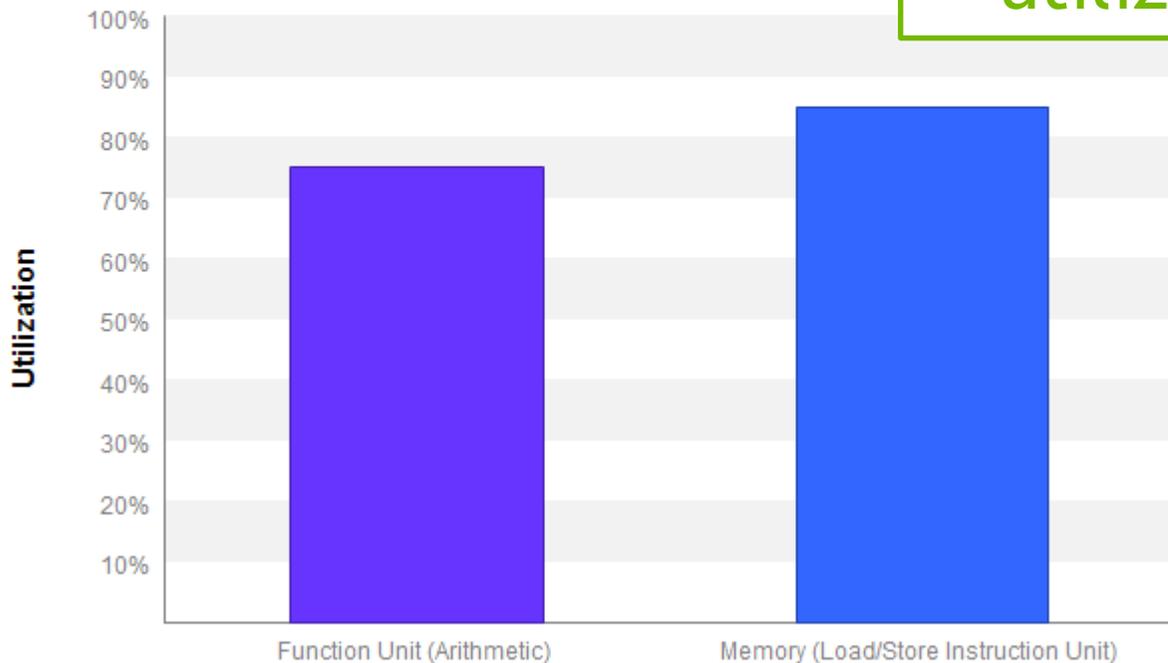
Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x
Higher Occupancy	1.562ms	3.35x
Shared Memory	0.911ms	5.74x

IDENTIFY PERFORMANCE LIMITER

Results

i Kernel Performance Is Bound By Compute And Memory Bandwidth

For device "Tesla K40m" compute and memory utilization are balanced. These utilization are good, but that additional performance improvement may be possible if either of both are increased.



Aha!
Getting into the high
utilization region

Analysis Details Settings Con

Export PDF Report

1. CUDA Application Analysis
2. Performance-Critical Kernels
3. Compute, Bandwi..., or Latency Bound

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "gaussian_filter_7x7_v5" is most likely limited by both compute and memory bandwidth.

Perform Compute Analysis

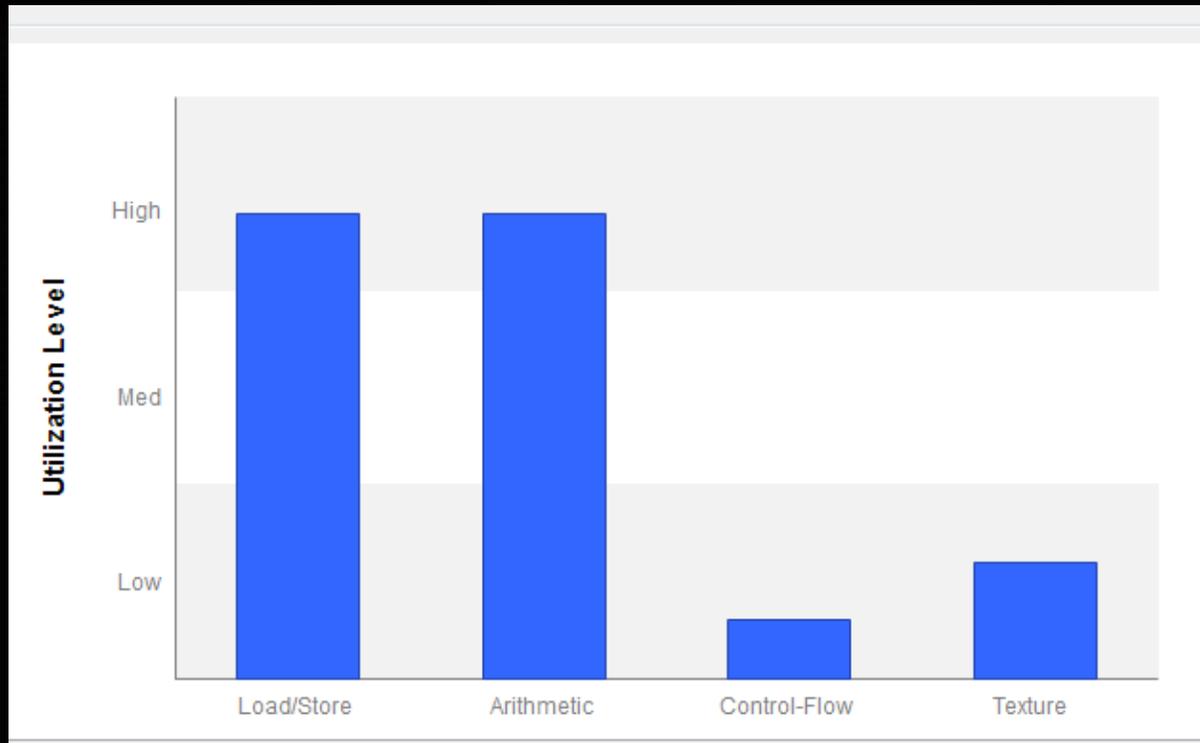
Perform Memory Bandwidth Analysis

Both compute and memory are likely bottlenecks to performance for this kernel, so you should first perform both compute and memory analysis to determine how they are limiting performance.

Perform Latency Analysis

Latency is likely not the primary performance

Launch



LOOKING FOR MORE INDICATORS

Analysis Details Settings Con

Export PDF Report

1. CUDA Application Analysis

2. Performance-Critical Kernels

3. Compute, Bandwi..., or Latency Bound

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "gaussian_filter_7x7_v5" is most likely limited by both compute and memory bandwidth.

Perform Compute Analysis

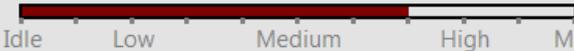
Perform Memory Bandwidth Analysis

Both compute and memory are likely bottlenecks to performance for this kernel, so you should first perform both compute and memory analysis to determine how they are limiting performance.

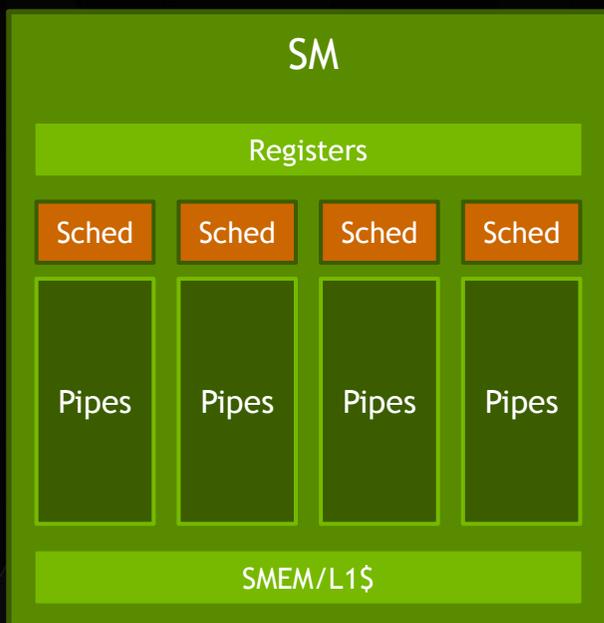
Perform Latency Analysis

Latency is likely not the primary performance

Results

	Transactions	Bandwidth	Utilization
L1/Shared Memory			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	2304000	1,439.543 GB/s	
Shared Stores	640000	399.873 GB/s	
Global Loads	0	0 B/s	
Global Stores	128000	9.997 GB/s	
Atomic	0	0 B/s	
L1/Shared Total	3072000	1,849.413 GB/s	
L2 Cache			
L1 Reads	0	0 B/s	
L1 Writes	128000	9.997 GB/s	
Texture Reads	757777	59.183 GB/s	
Atomic	0	0 B/s	
Noncoherent Reads	757778	59.183 GB/s	
Total	1643555	128.362 GB/s	
Texture Cache			

Load/Store Unit is really busy!
Can we reduce the load?

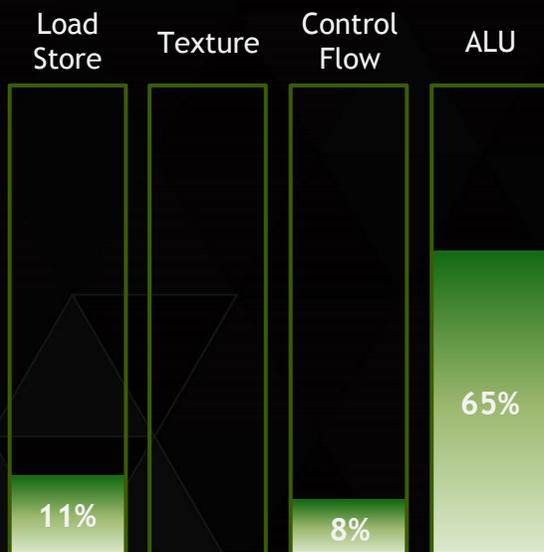
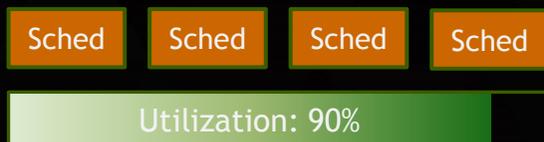


- ▶ Each SM has 4 schedulers (Kepler)
- ▶ Schedulers issue instructions to pipes
- ▶ A scheduler issues up to 2 instructions/cycle
 - ▶ Sustainable peak is 7 instructions/cycle per SM (not $4 \times 2 = 8$)
- ▶ A scheduler issues inst. from a single warp
- ▶ Cannot issue to a pipe if its issue slot is full

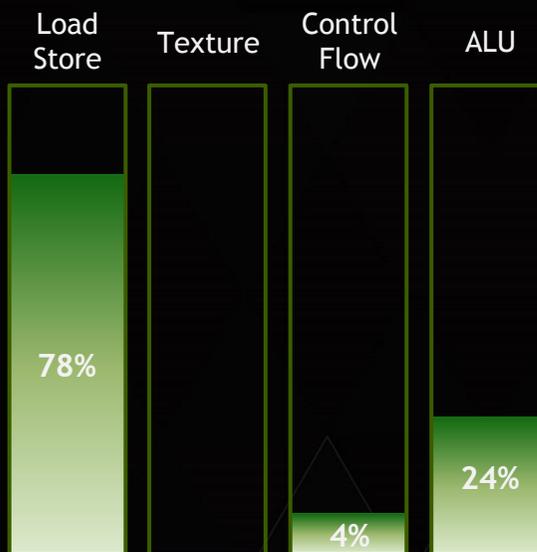
INSTRUCTION THROUGHPUT



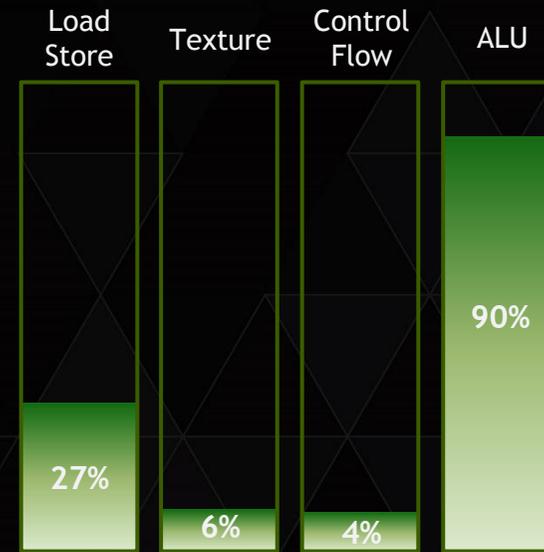
Schedulers saturated



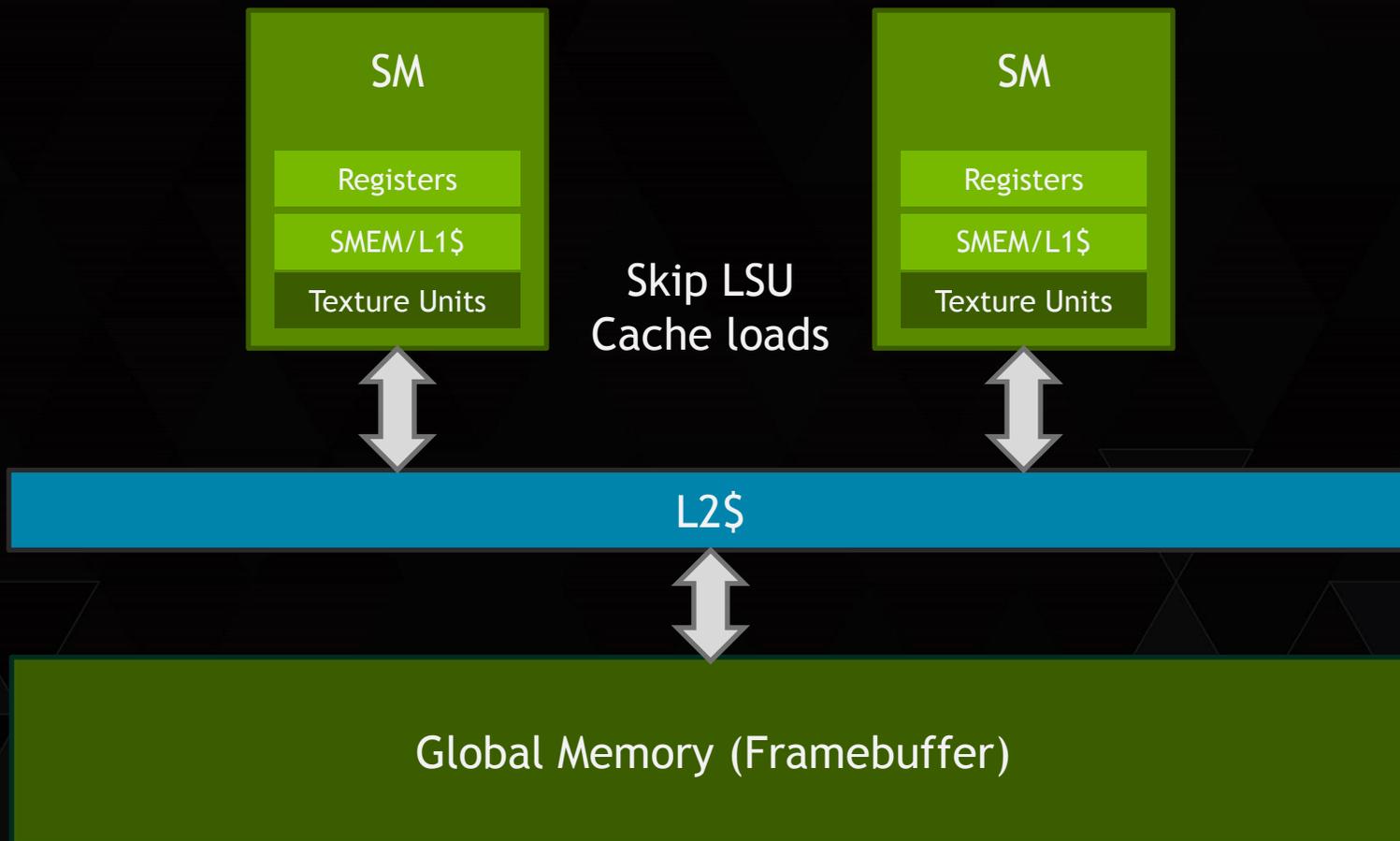
Pipe saturated



Schedulers and pipe saturated



READ-ONLY CACHE (TEXTURE UNITS)



READ-ONLY PATH

- ▶ Annotate read-only parameters with `const __restrict`

```
__global__ void gaussian_filter_7x7_v2(int w, int h, const uchar * __restrict src, uchar *dst)
```

- ▶ The compiler generates LDG instructions: 0.808ms

Kernel	Time	Speedup
Original version	5.233ms	1.00x
Better memory accesses	1.589ms	3.29x
Higher Occupancy	1.562ms	3.35x
Shared memory	0.911ms	5.74x
Read-Only path	0.808ms	6.48x

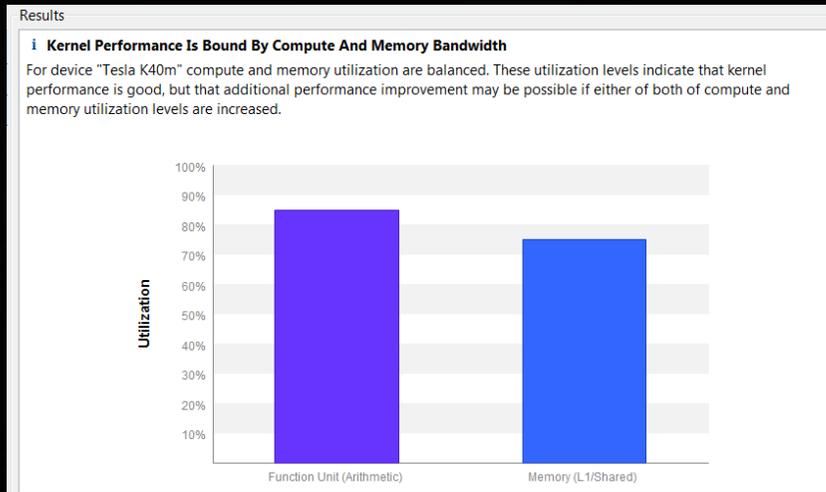
PERF-OPT QUICK REFERENCE CARD

Category:	Latency Bound - Texture Cache
Problem:	Load/Store Unit becomes bottleneck
Goal:	Relieve Load/Store Unit from read-only data
Indicators:	High utilization of Load/Store Unit, pipe-busy stall reason, significant amount of read-only data
Strategy:	Load read-only data through Texture Units: <ul style="list-style-type: none">• Annotate read-only pointers with <code>const __restrict__</code>• Use <code>__ldg()</code> intrinsic



THE RESULT: 6.5X

- ▶ Looking much better
- ▶ Things to investigate next
 - ▶ Reduce computational intensity (separable filter)
 - ▶ Increase Instruction Level Parallelism (process two elements per thread)
- ▶ The sobel filter is starting to become the bottleneck



Results

i Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved occupancy. The kernel with a higher rank (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
88	[1 kernel instances] gaussian_filter_7x7_v4(int, int, unsigned char const *, unsigned char*)
39	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

MORE IN OUR COMPANION CODE

Kernel	Time	Speedup
Original version	5.233ms	1.00x
Better memory accesses	1.589ms	3.29x
Higher Occupancy	1.562ms	3.35x
Shared memory	0.911ms	5.74x
Read-Only path	0.808ms	6.48x
Separable filter	0.481ms	10.88x
Process two pixels per thread (memory efficiency + ILP)	0.415ms	12.61x
Use 64-bit shared memory (remove bank conflicts)	0.403ms	12.99x
Use float instead of int (increase instruction throughput)	0.363ms	14.42x

Your next idea!!!

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

GPU TECHNOLOGY
CONFERENCE

SUMMARY

ITERATIVE OPTIMIZATION WITH NSIGHT EE

- ▶ Trace the Application
- ▶ Identify the Hotspot and Profile it
- ▶ Identify the Performance Limiter
 - ▶ Memory Bandwidth
 - ▶ Instruction Throughput
 - ▶ Latency
- ▶ Look for indicators
 - ▶ Take nvvp guided analysis as a starting point
 - ▶ But don't follow it too closely
- ▶ 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

GPU TECHNOLOGY
CONFERENCE

THANK YOU

JOIN THE CONVERSATION

#GTC15   