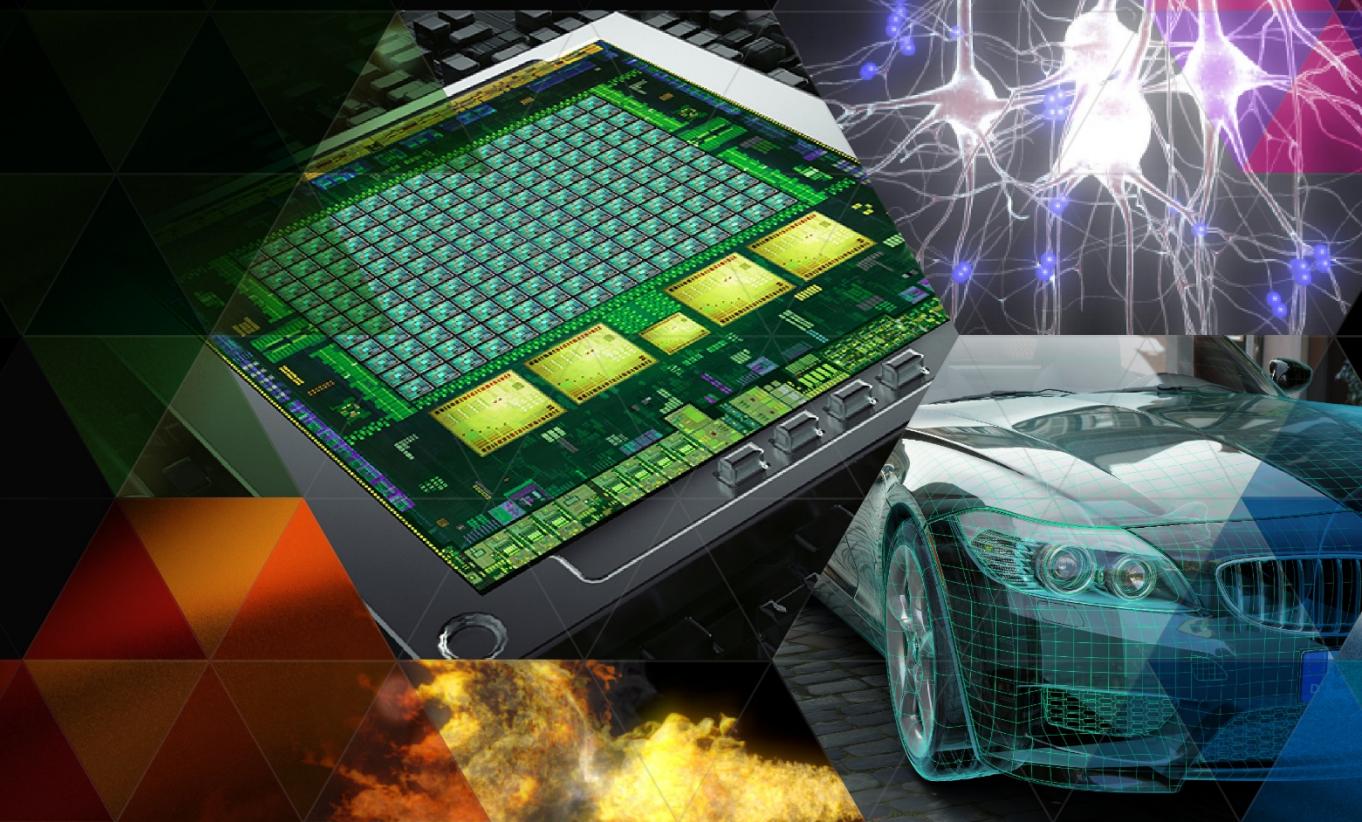




## CUDA OPTIMIZATION WITH NVIDIA NSIGHT™ ECLIPSE EDITION

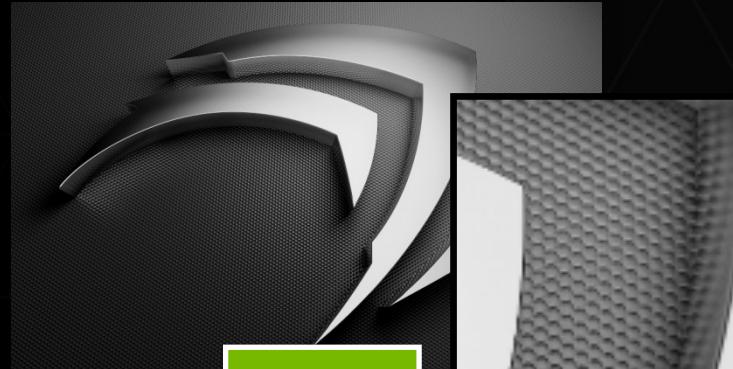
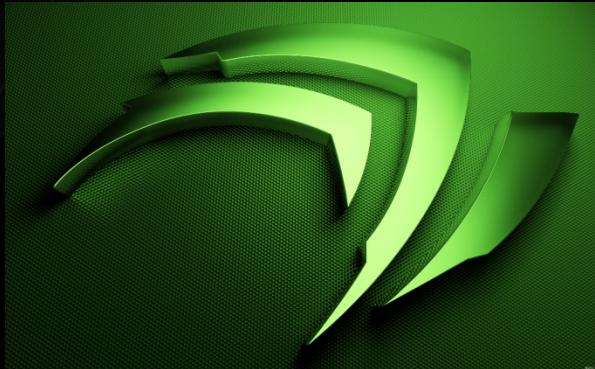


# WHAT YOU WILL LEARN

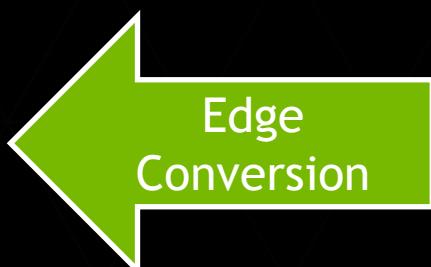
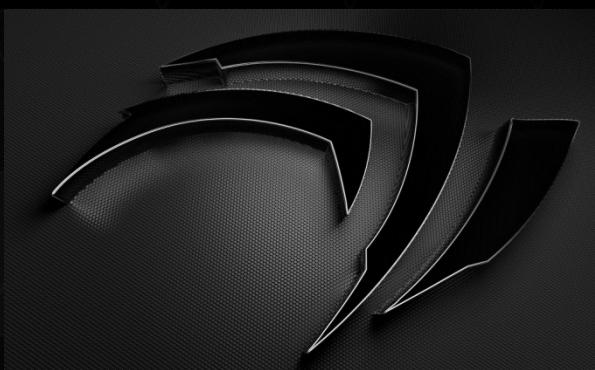
- ▶ An iterative method to optimize your GPU code
- ▶ Some common bottlenecks to look out for
- ▶ Performance diagnostics with NVIDIA Nsight EE
- ▶ Companion Code: <https://github.com/chmaruni/nsight-gtc2015>

# INTRODUCING THE APPLICATION

Series of common image processing filters

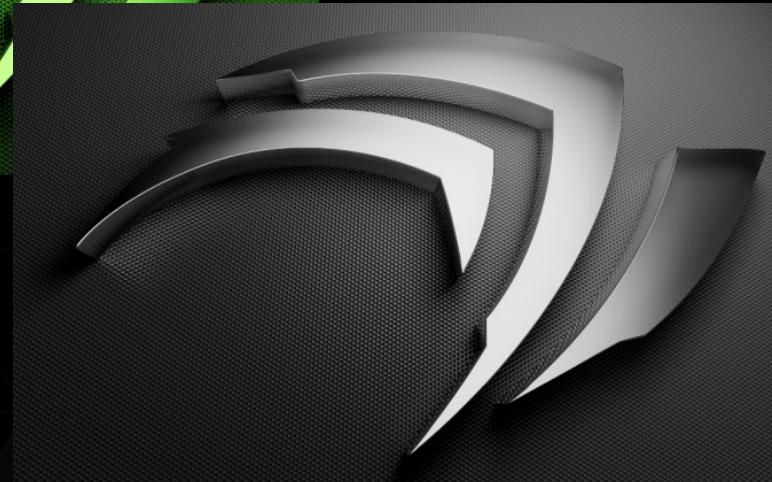


Blur



# INTRODUCING THE APPLICATION

## Grayscale Conversion



```
// r, g, b: Red, green, blue comp  
foreach pixel p:
```

$$p = 0.298839f * r + 0.586811f * g + 0.114350f * b$$

# INTRODUCING THE APPLICATION

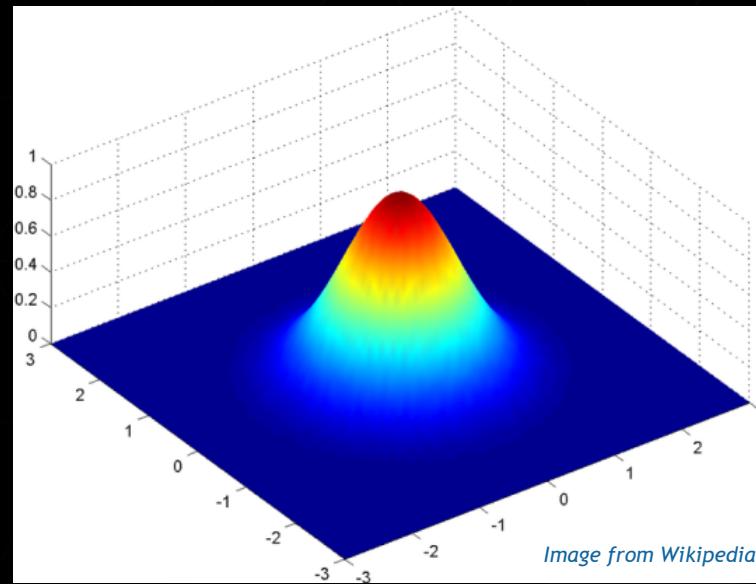
## Gaussian filter blur

7x7 Gaussian Filter



foreach pixel p:

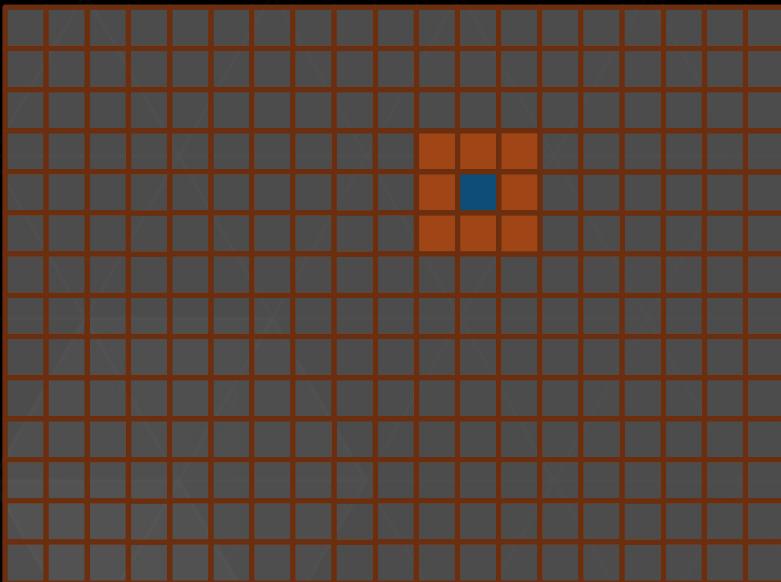
p = weighted sum of p and its 48 neighbors



# INTRODUCING THE APPLICATION

## Edge conversion (Sobel Filter)

3x3 Sobel Filter



**foreach pixel p:**

$G_x = \text{weighted sum of } p \text{ and its 8 neighbors}$

$G_y = \text{weighted sum of } p \text{ and its 8 neighbors}$

$$p = \sqrt{G_x + G_y}$$

Weights for  $G_x$ :

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

Weights for  $G_y$ :

|    |    |    |
|----|----|----|
| 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

Similar results are obtained on Windows

# PERFORMANCE OPTIMIZATION CYCLE

5. Change and  
Test Code



4. Reflect



4b. Build Knowledge



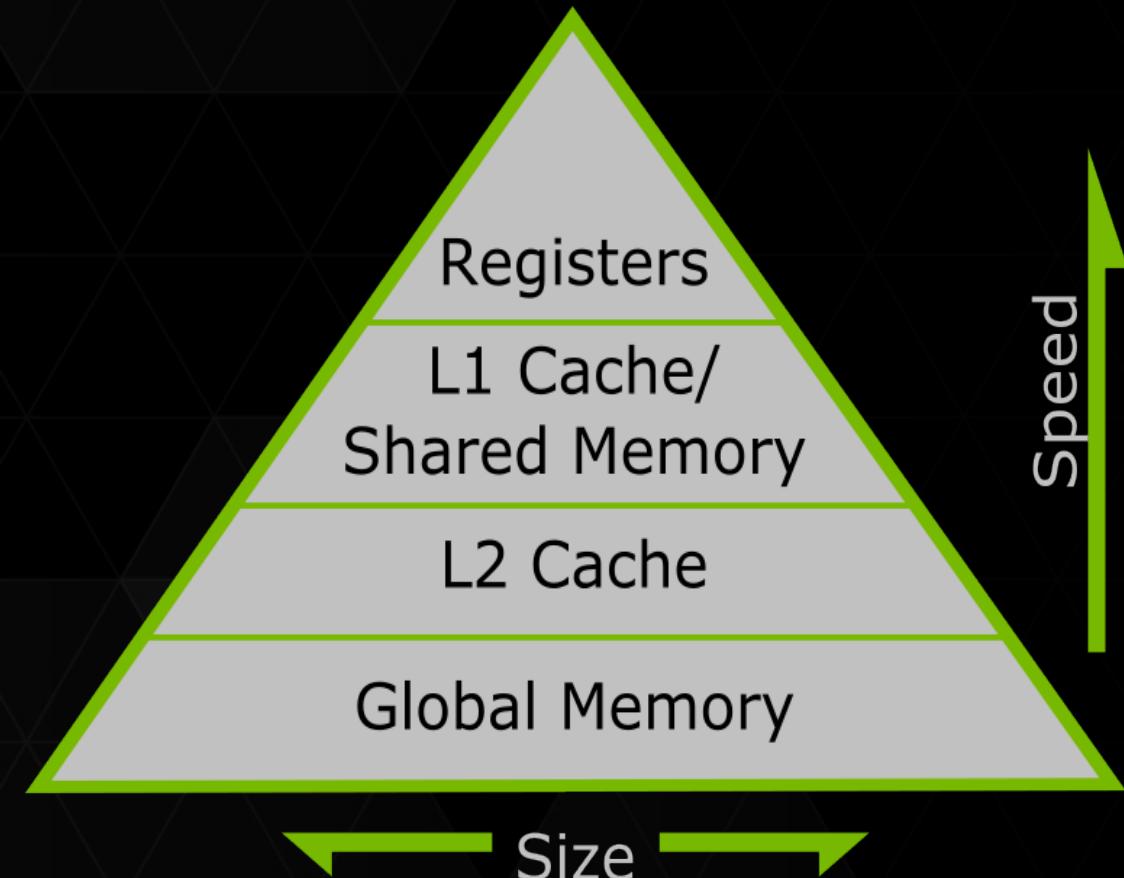
1. Profile  
Application

2. Identify  
Performance  
Limiter

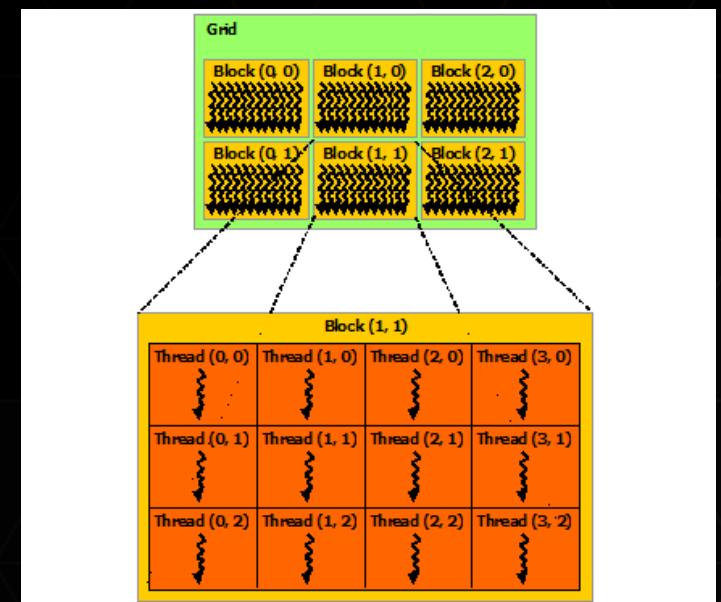
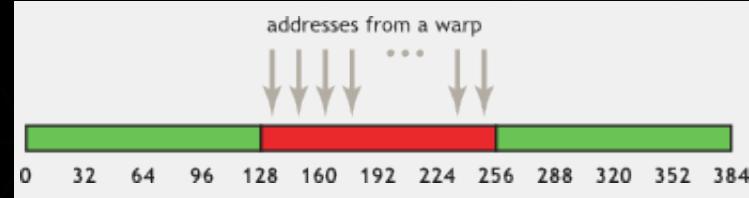
3. Analyze Profile  
& Find Indicators

# PREREQUISITES

## GPU Memory Spaces

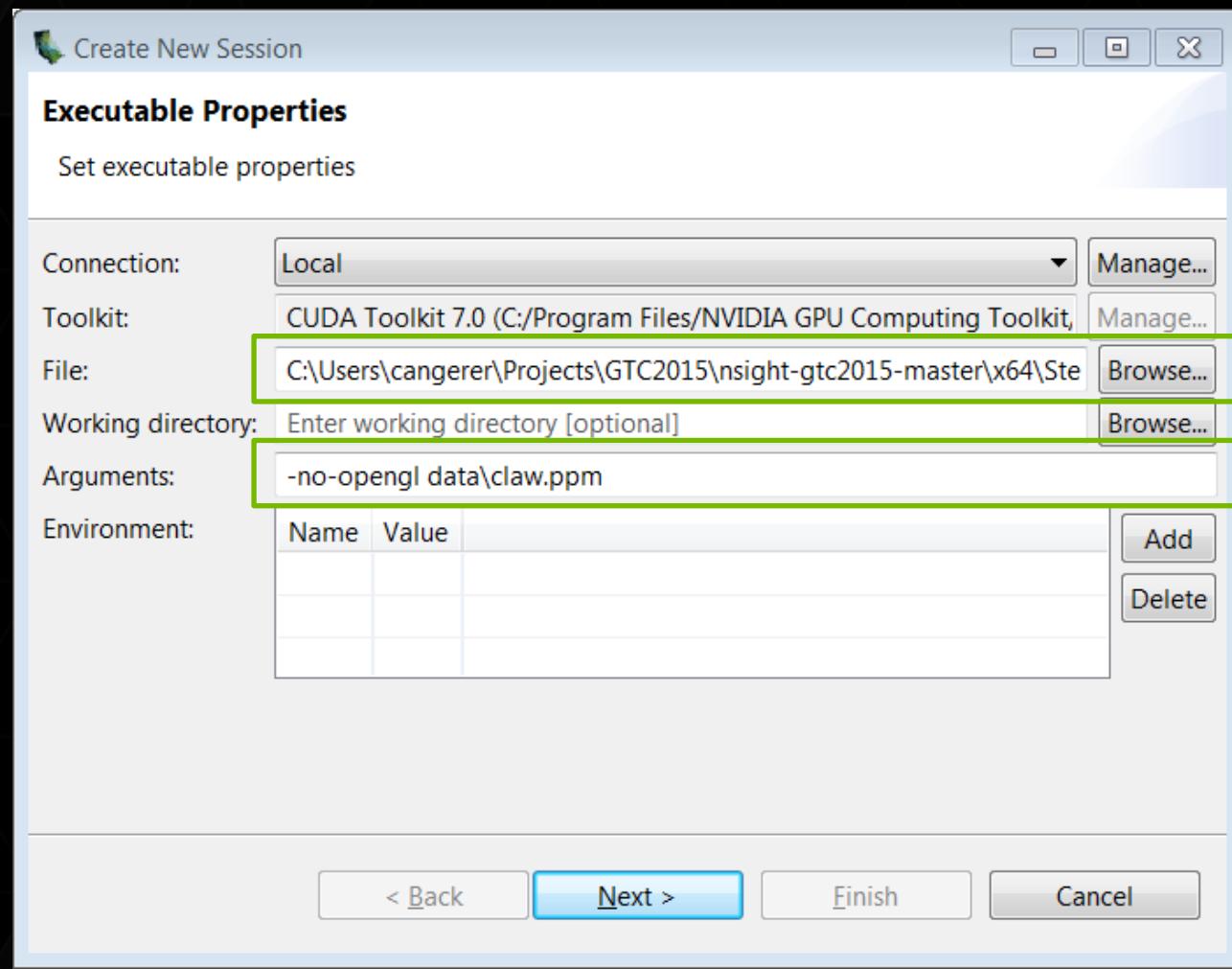


## CUDA Execution Model

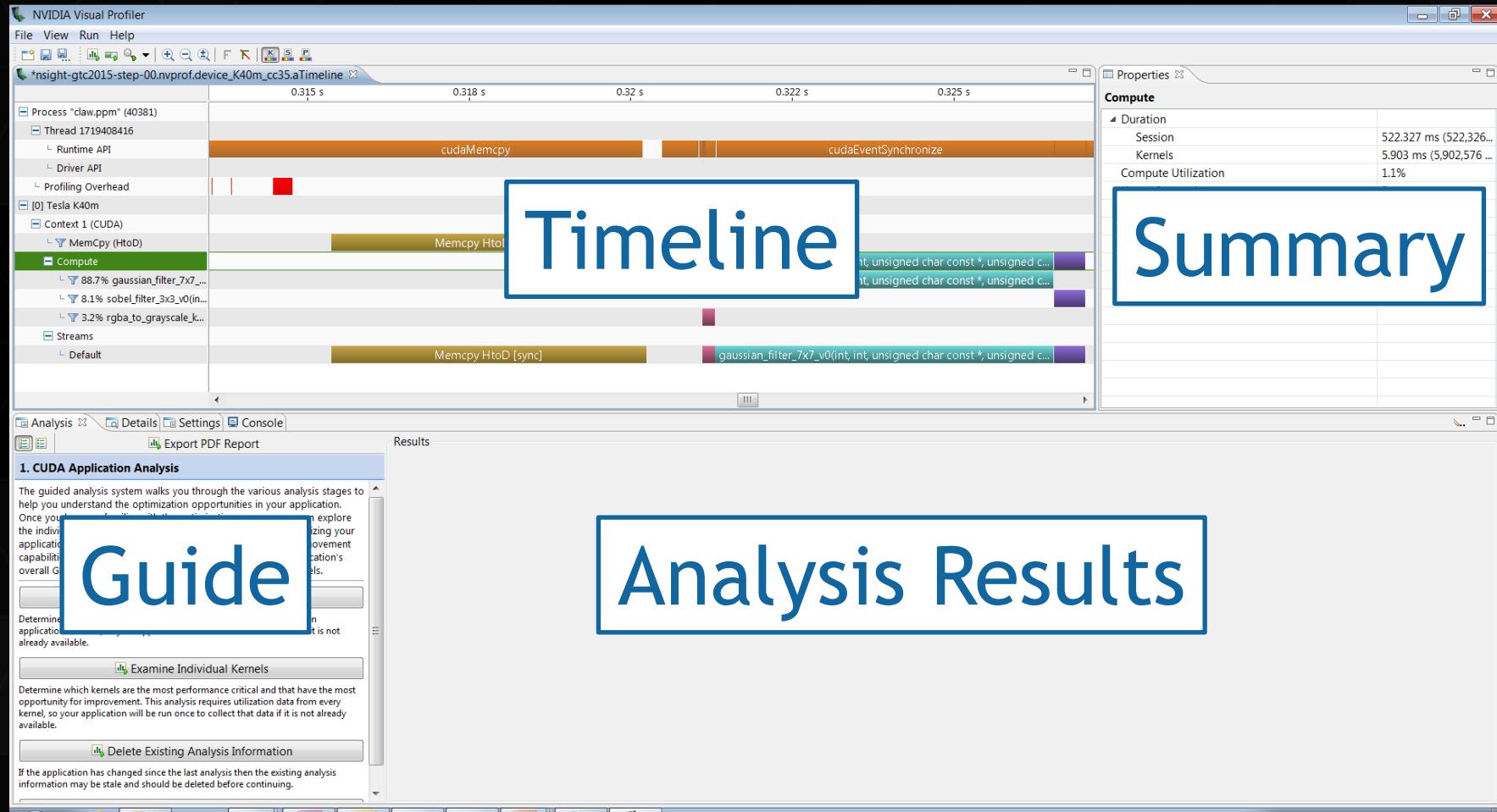


*Iteration 1*

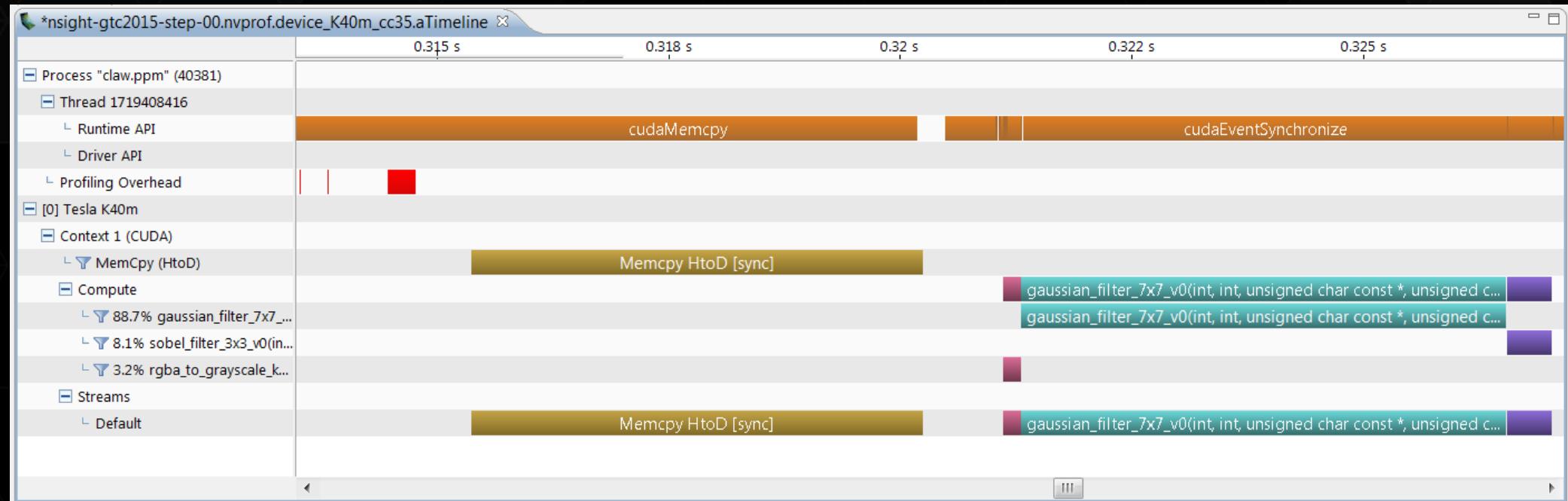
# CREATE A NEW NVVP SESSION



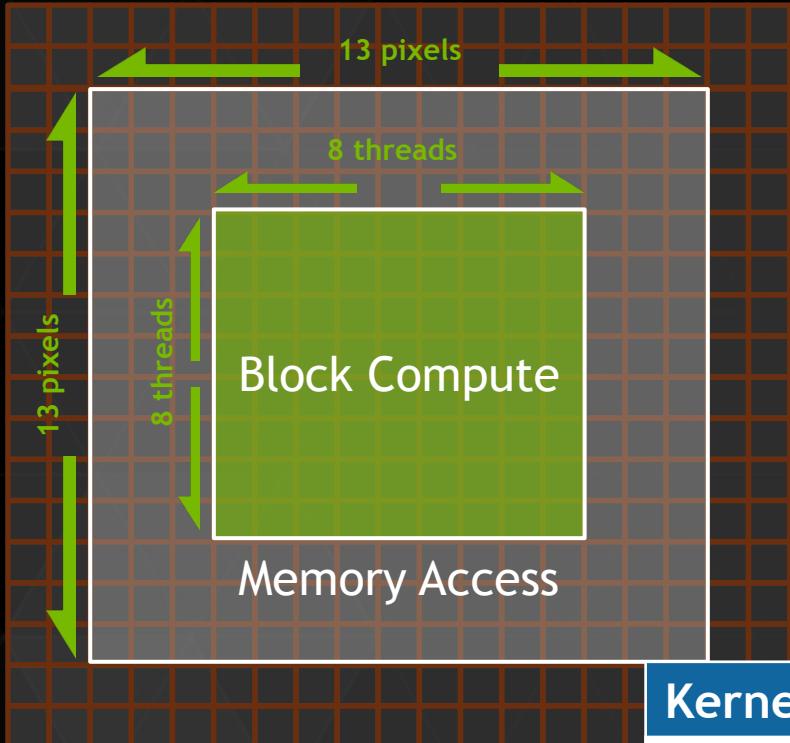
# THE PROFILER WINDOW



# TIMELINE



# 7X7 FILTER KERNEL



Each thread first loads 7x7 uchar, convert to int, compute convolution for a pixel, store as uchar.

Image is 2560 x 1600 (~4M pixels)

| Kernel           | Time    | Speedup |
|------------------|---------|---------|
| Original Version | 5.233ms | 1.00x   |

# PERFORM KERNEL ANALYSIS

**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 run. Your application will be run once to collect data 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.

**NVIDIA Visual Profiler**

**\*nsight-gtc2015-step-00.nvprof.device\_K40m\_cc35.aTimeline**

**Properties**

**gaussian\_filter\_7x7\_v0(int, int, unsigned char const \*, unsigned char\*)**

|                     | Start              | End                | Duration            | Grid Size     | Block Size | Registers/Thread | Shared Memory/Block |
|---------------------|--------------------|--------------------|---------------------|---------------|------------|------------------|---------------------|
| Start               | 321.303 ms (321,30 |                    |                     |               |            |                  |                     |
| End                 |                    | 326.536 ms (326,53 |                     |               |            |                  |                     |
| 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                 |

**Select**

**Launch**

**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**

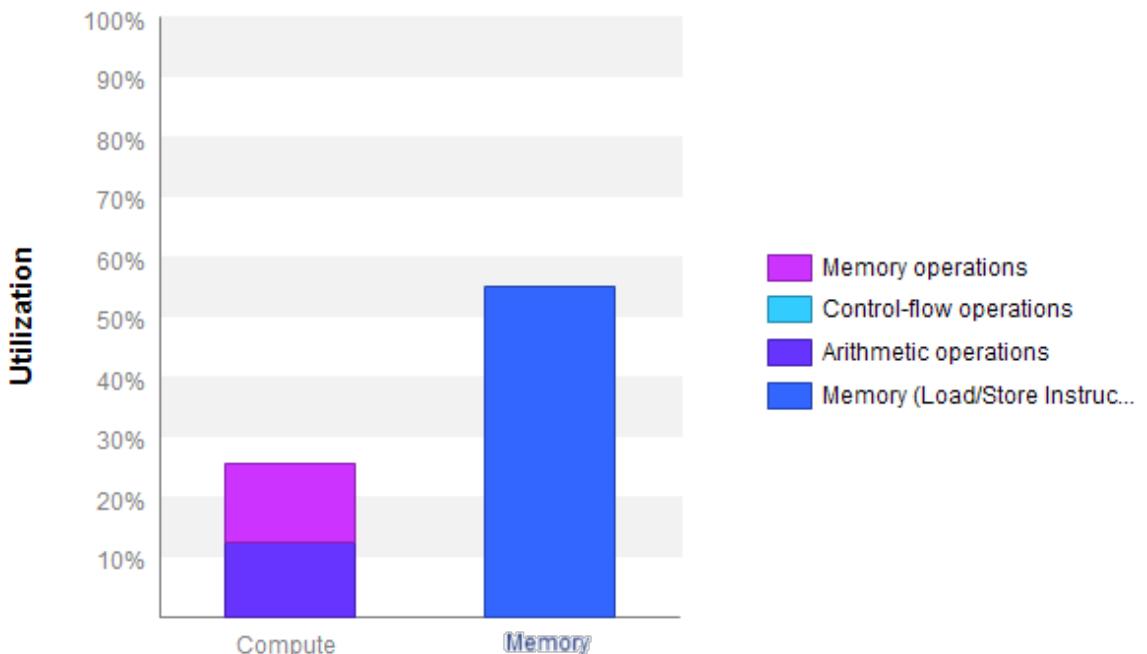
You can collect additional information to help identify kernels with potential performance problems. After running this analysis, select any of the new results at right to highlight the individual kernels for analysis.

# IDENTIFY PERFORMANCE LIMITER

## Results

### **i Kernel Performance Is Bound By Instruction And Memory Latency**

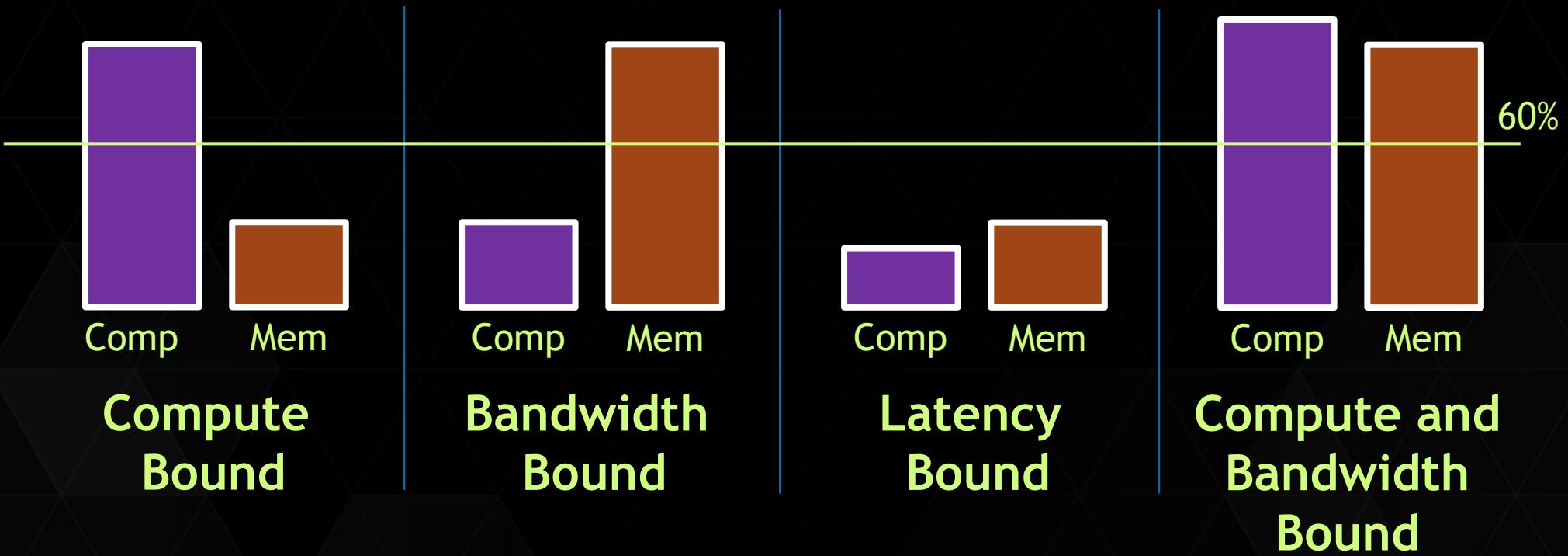
This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K40m". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or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



# PERFORMANCE LIMITER CATEGORIES



## Memory vs Compute Utilization

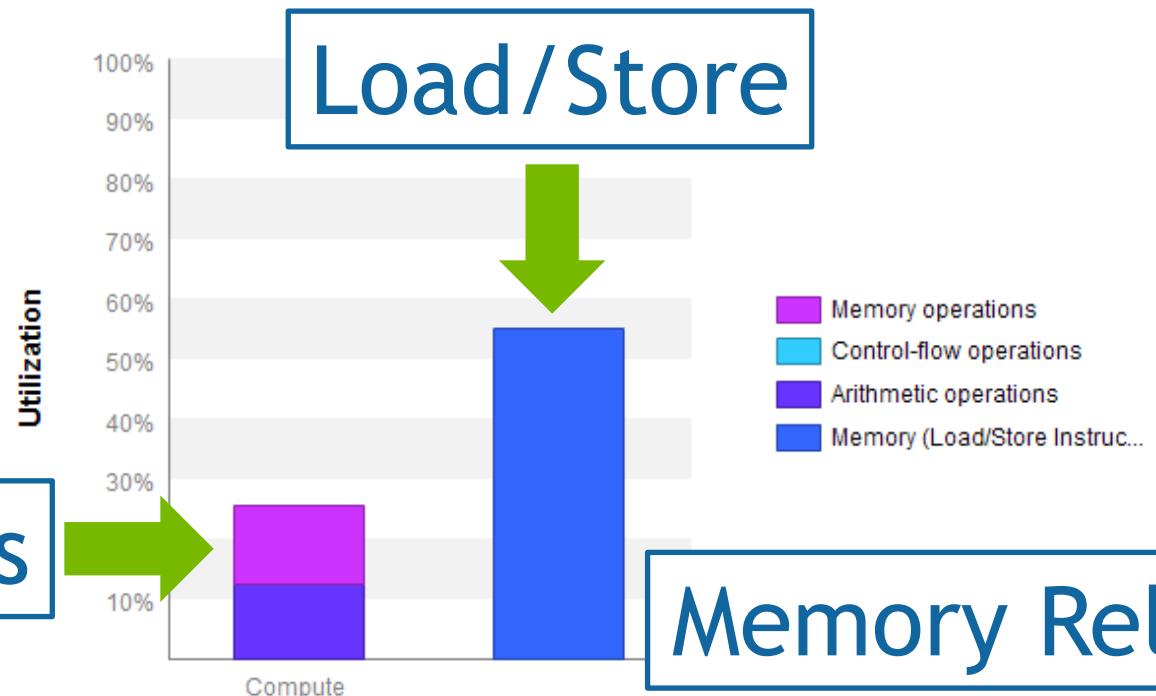


# IDENTIFY PERFORMANCE LIMITER

## Results

### i Kernel Performance Is Bound By Instruction And Memory Latency

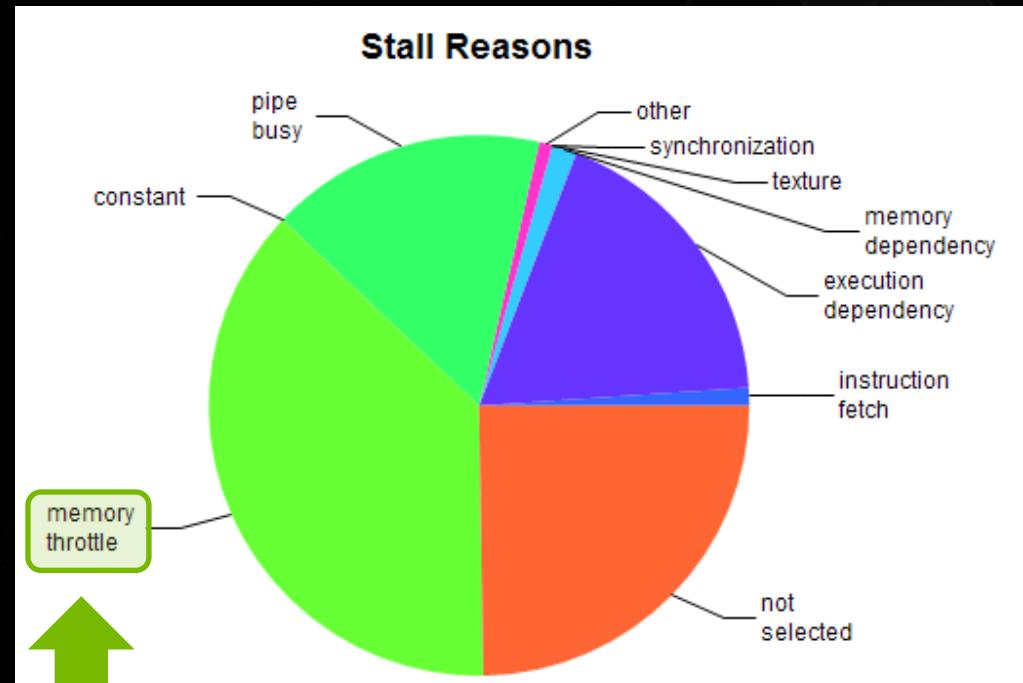
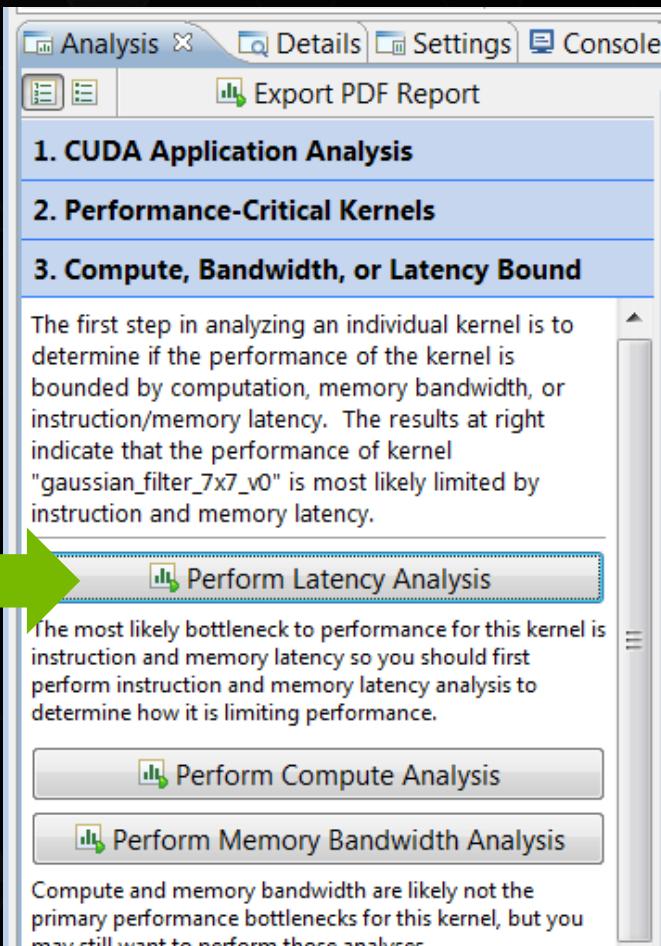
This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K40m". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or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



Memory Ops

Memory Related Issues?

# LOOKING FOR INDICATORS



Large number of memory operations stalling LSU

# LOOKING FOR MORE INDICATORS

## Unguided Analysis



The screenshot shows the Nsight Compute interface with the following details:

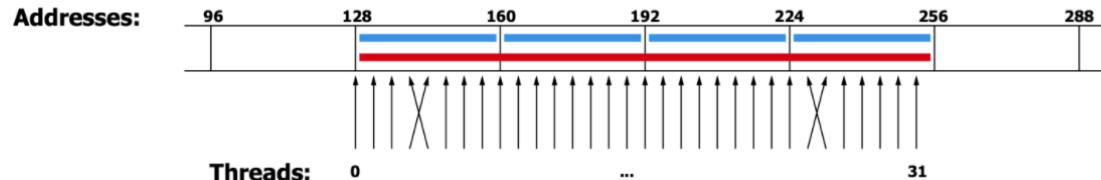
- Analysis Tab:** Shows the analysis progress for the `gaussian_filter_7x7_v0` kernel.
- Results Panel:** Displays a warning about **Global Memory Alignment and Access Pattern**. It states that memory bandwidth is used most efficiently when each global memory load and store has proper alignment and access pattern. An optimization note suggests selecting entries to open source code for inefficient alignment and access patterns.
- Source Code Editor:** Shows the C code for the `in_img` function, which handles neighborhood loading and stores.
- List View:** Shows a list of 243 transactions with details like L2 Transactions/Access and Ideal Transactions/Access ratios.

4-5 Global Load/Store Transactions per 1 Request

# GLOBAL MEMORY TRANSACTIONS: REVIEW

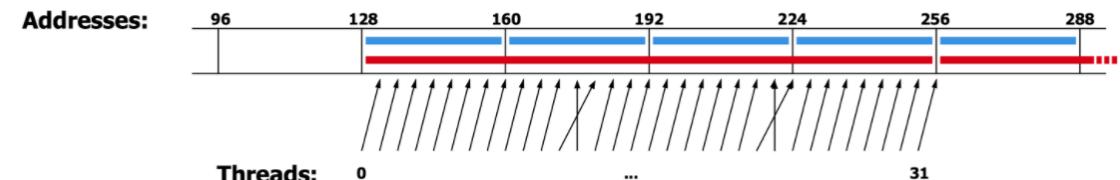


## Aligned accesses (sequential/non-sequential)



| Compute capability:  | 2.0 and later |                |
|----------------------|---------------|----------------|
|                      | Uncached      | Cached         |
| Memory transactions: | 1x 32B at 128 | 1x 128B at 128 |
|                      | 1x 32B at 160 |                |
|                      | 1x 32B at 192 |                |
|                      | 1x 32B at 224 |                |
|                      |               |                |

## Mis-aligned accesses (sequential/non-sequential)

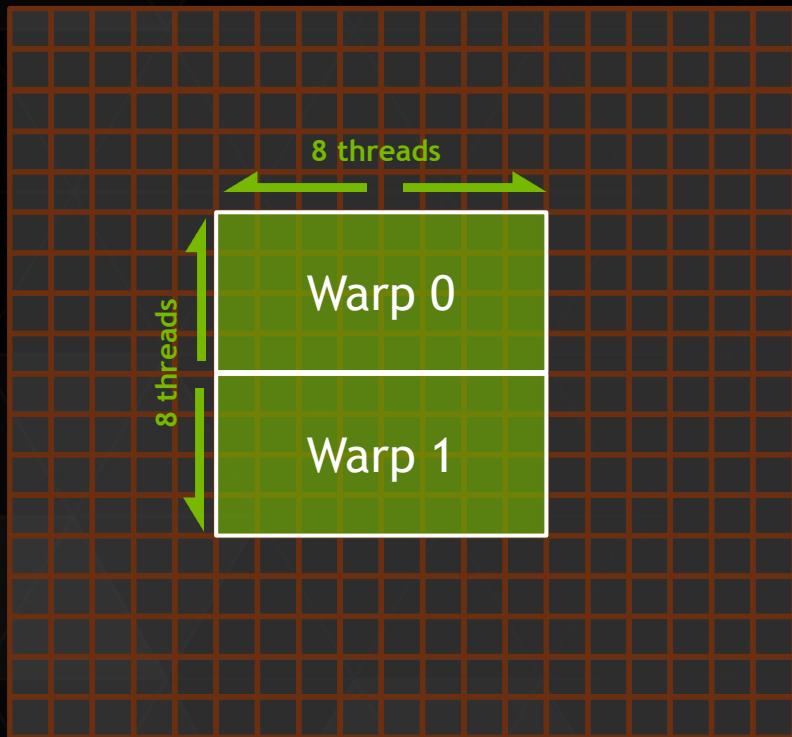


| Compute capability:  | 2.0 and later |                |
|----------------------|---------------|----------------|
|                      | Uncached      | Cached         |
| Memory transactions: | 1x 32B at 128 | 1x 128B at 128 |
|                      | 1x 32B at 160 | 1x 128B at 256 |
|                      | 1x 32B at 192 |                |
|                      | 1x 32B at 224 |                |
|                      | 1x 32B at 256 |                |
|                      |               |                |

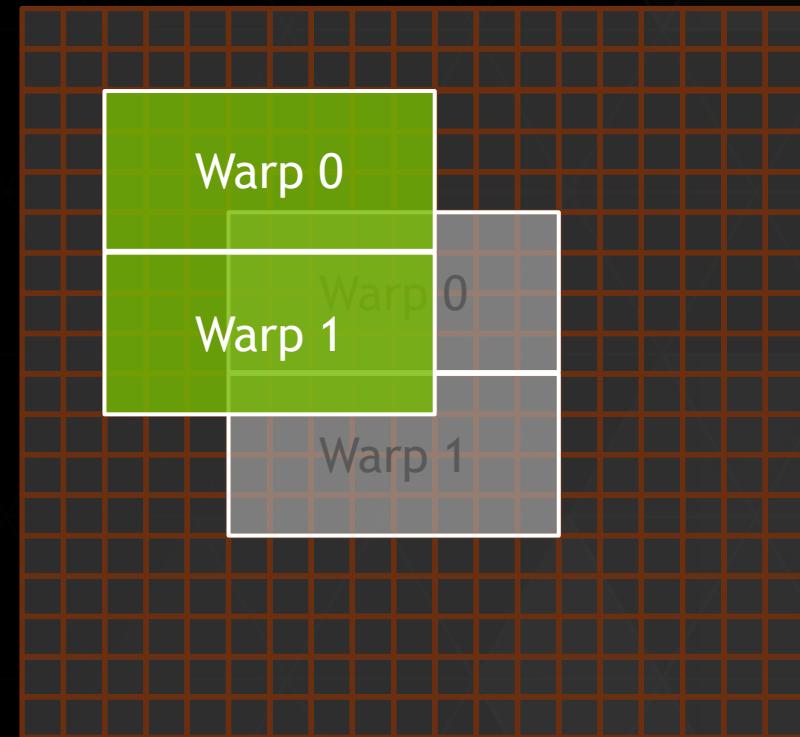
# 7X7 FILTER KERNEL A WARP VIEW



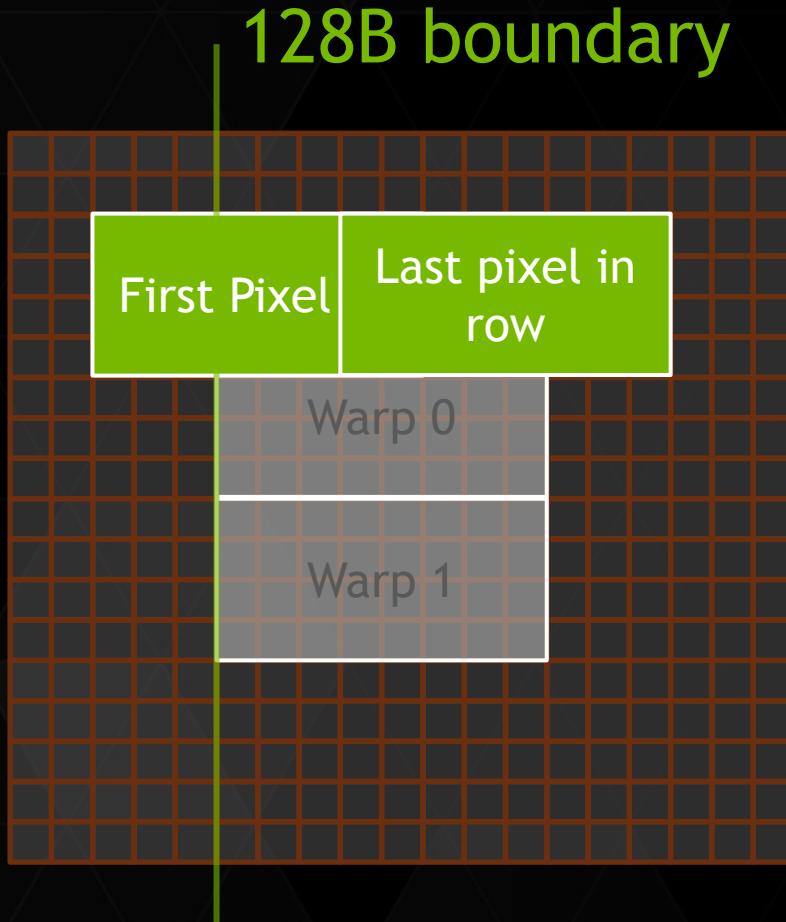
Computing



Loading the first  
pixel



# 7X7 FILTER KERNEL A WARP VIEW



Pixels are now in grayscale, so each pixel is a uchar.

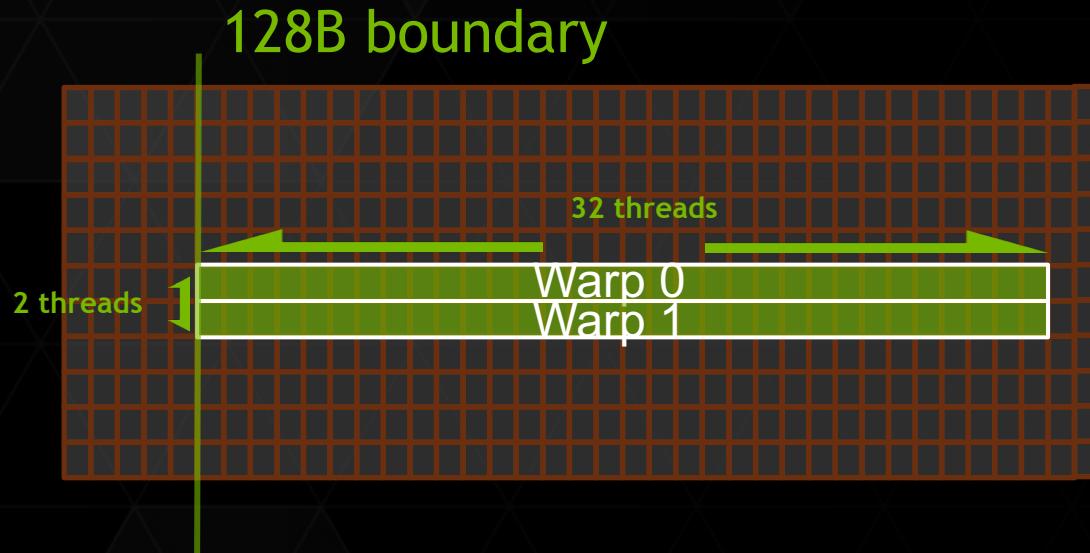
Warps will be loading 4 lines at a time.

In worst case, they could be going across a 128B boundary line causing 8 128B loads from global memory.

This would be loading 1024 B to only be using 32 B.

Typical case will not go across 128 B boundary but will still be loading 4 128B loads from global memory while still only using 32B.

# 7X7 FILTER KERNEL FLATTEN THE WARP



A warp loads a single line.

Could still be going across 128B lines but more often will be within a line.

| 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,<br>High #transactions/#request compared to ideal  |
| Strategy:   | 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

## 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,<br>High #transactions/#request compared to ideal   |
| Strategy:   | <p>Improve memory coalescing by:</p> <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> |



*Iteration 2*

# IDENTIFY HOTSPOT

Hotspot →

Results

**i Kernel Optimization Priorities**

The following kernels are ordered by optimization importance based on execution time and achieved optimization potential. A higher rank means the kernel is more important. The first few kernels (those that appear first in the list) is more likely to improve performance compared to other 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

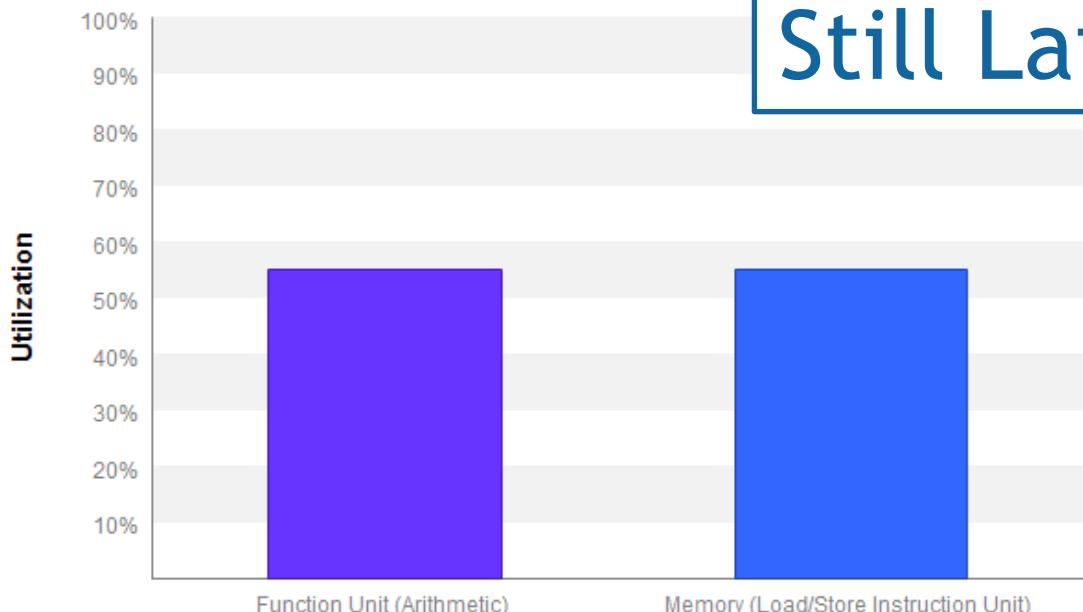
| Kernel                 | Time    | Speedup |
|------------------------|---------|---------|
| Original Version       | 5.233ms | 1.00x   |
| Better Memory Accesses | 1.589ms | 3.29x   |

# IDENTIFY PERFORMANCE LIMITER

## Results

### **i Kernel Performance Is Bound By Instruction And Memory Latency**

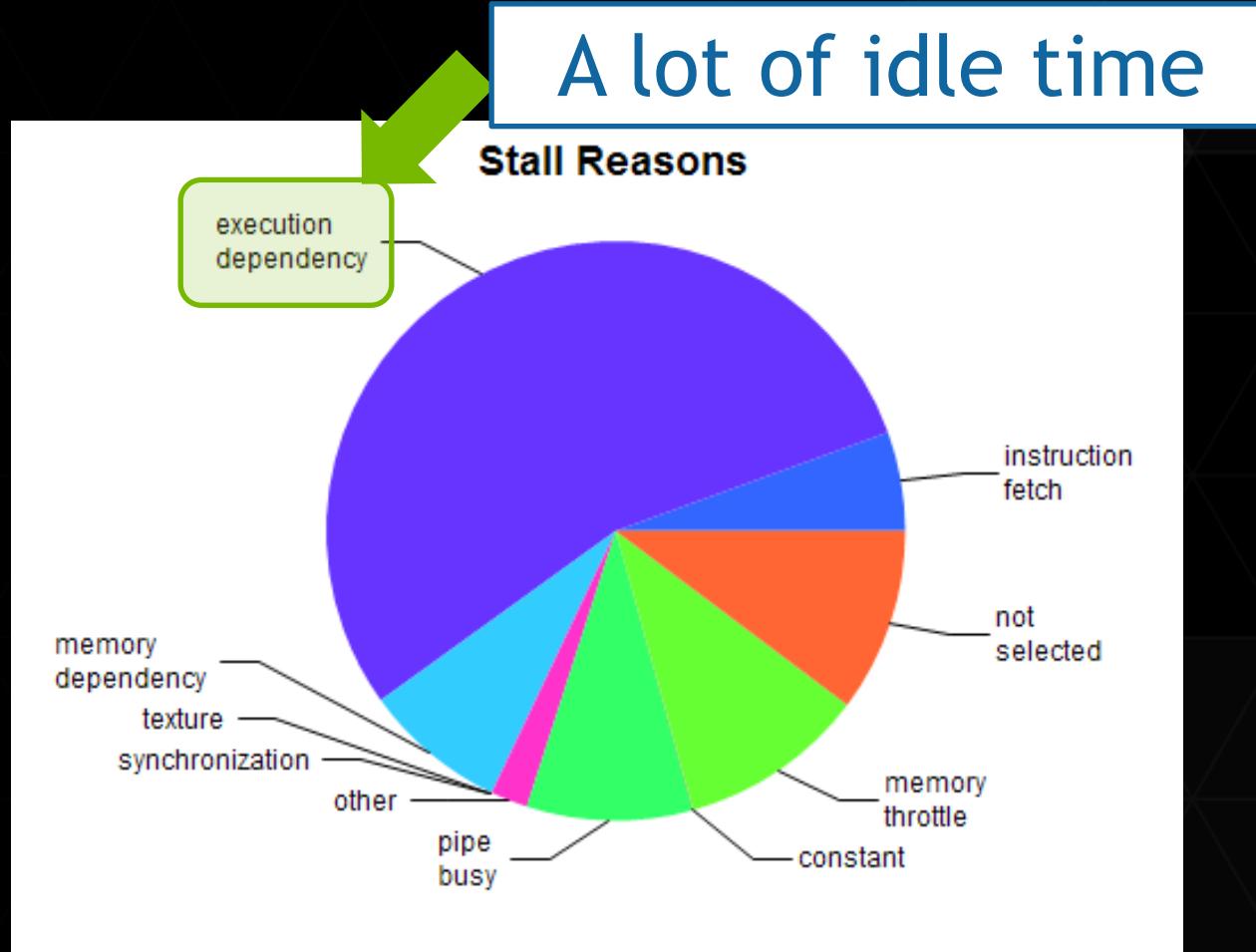
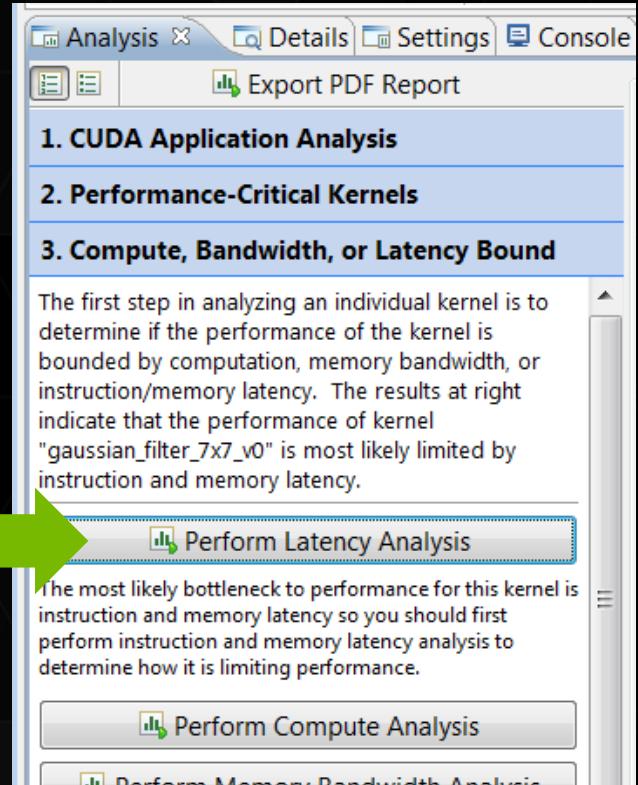
This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K40m". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or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



**Still Latency Bound**

# LOOKING FOR INDICATORS

A lot of idle time

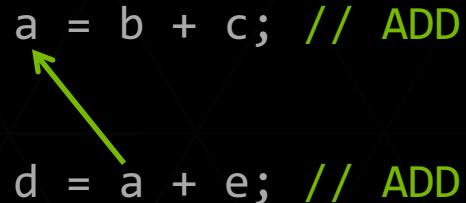


- Not enough work inside a thread to hide latency?



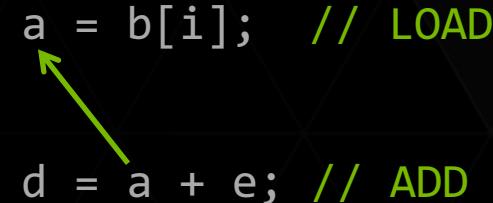
# STALL REASONS: EXECUTION DEPENDENCY

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



A diagram illustrating a data dependency between two addition instructions. The first instruction is `a = b + c; // ADD`. An arrow points from the result register of this instruction to the source register of the second instruction, `d = a + e; // ADD`.

```
a = b[i]; // LOAD  
d = a + e; // ADD
```



A diagram illustrating a data dependency between a load instruction and an addition instruction. The first instruction is `a = b[i]; // LOAD`. An arrow points from the result register of this instruction to the source register of the second instruction, `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

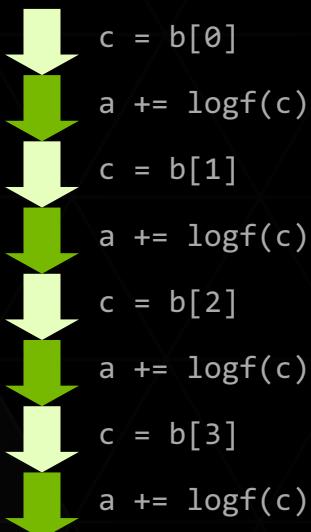
```
a = b + c; // Independent ADDs  
d = e + f;
```



# ILP AND MEMORY ACCESSES

No ILP

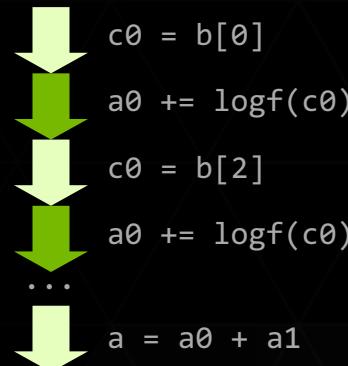
```
float a = 0.0f;  
for( int i = 0 ; i < N ; ++i )  
    a += logf(b[i]);
```



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 X   |                     |
|--|---------------------|
| <b>gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)</b> |                     |
| 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

Not enough active warps to hide latencies?

Analysis X Details Settings Console

Export PDF Report

**1. CUDA Application Analysis**

**2. Performance-Critical Kernels**

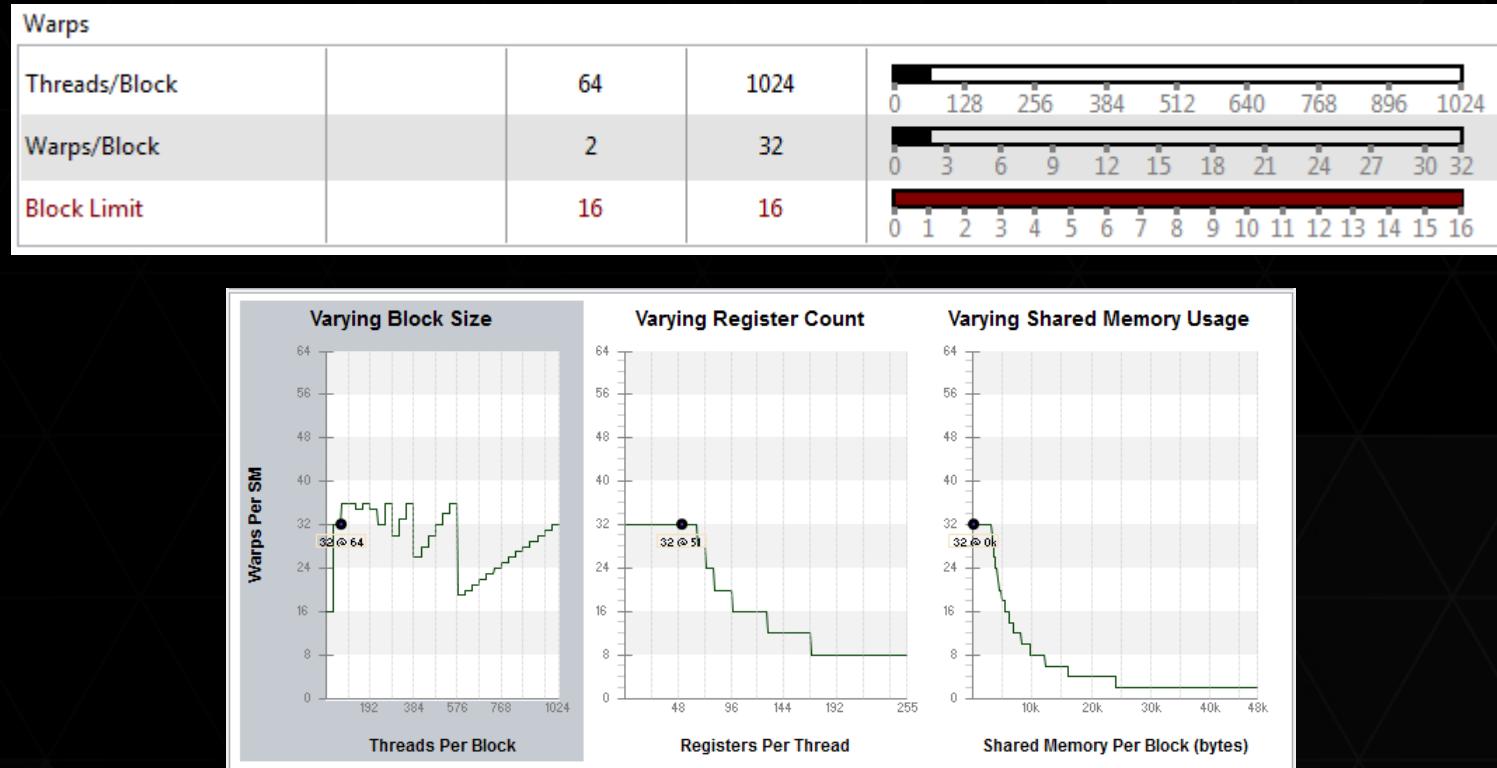
**3. Compute, Bandwidth, or Latency Bound**

**4. 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**

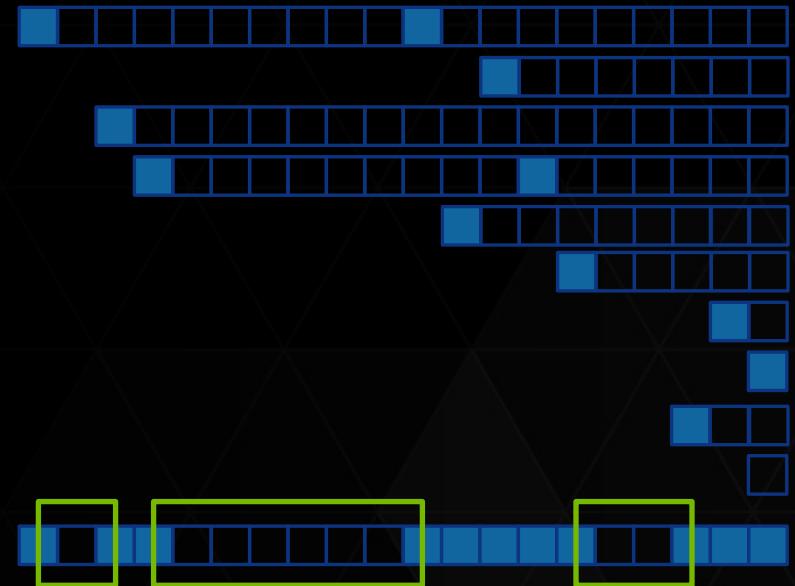
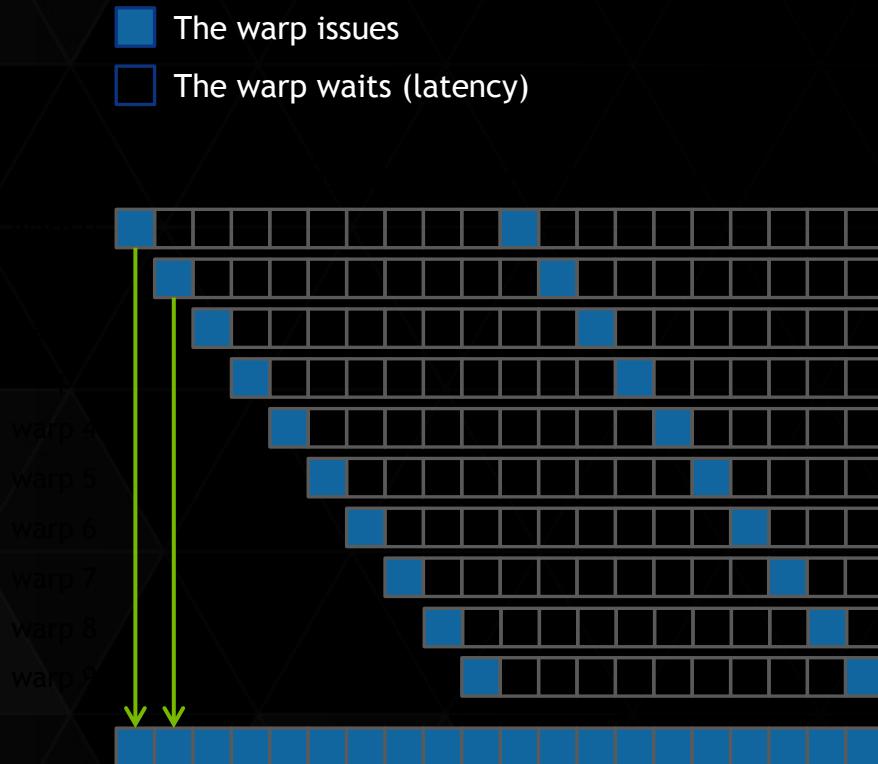
Occupancy is a measure of how many warps the kernel has active on the GPU, relative to the maximum number of warps supported by the GPU. Theoretical occupancy provides an upper bound while achieved occupancy





# LATENCY

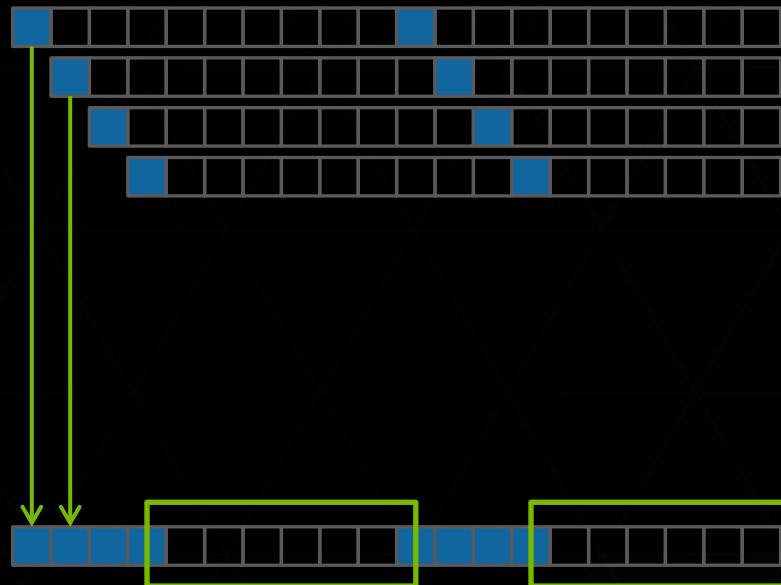
GPUs cover latencies by having a lot of work in flight





# LATENCY: LACK OF OCCUPANCY

- ▶ 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 | Incremental Speedup |
|------------------------|---------|---------|---------------------|
| Original Version       | 5.233ms | 1.00x   |                     |
| Better Memory Accesses | 1.589ms | 3.29x   | 3.29x               |
| Higher Occupancy       | 1.562ms | 3.35x   | 1.02x               |

# 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:

- 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"><li>• Unroll loops (#pragma unroll)</li><li>• Refactor threads to compute n output values at the same time (code duplication)</li></ul> |



*Iteration 3*

# IDENTIFY HOTSPOT

Hotspot →

| Results                                 |  |
|---|--|
| <b>i Kernel Optimization Priorities</b> |  |
| 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

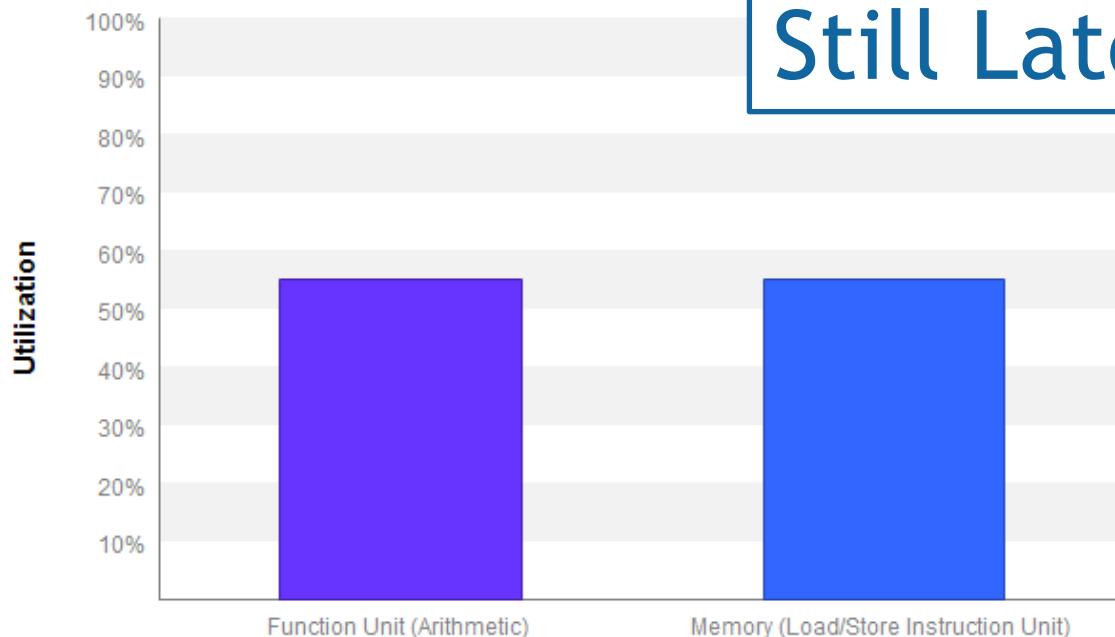
| Kernel                 | Time    | Speedup | Incremental Speedup |
|------------------------|---------|---------|---------------------|
| Original Version       | 5.233ms | 1.00x   |                     |
| Better Memory Accesses | 1.589ms | 3.29x   | 3.29x               |
| Higher Occupancy       | 1.562ms | 3.35x   | 1.02x               |

# IDENTIFY PERFORMANCE LIMITER

## Results

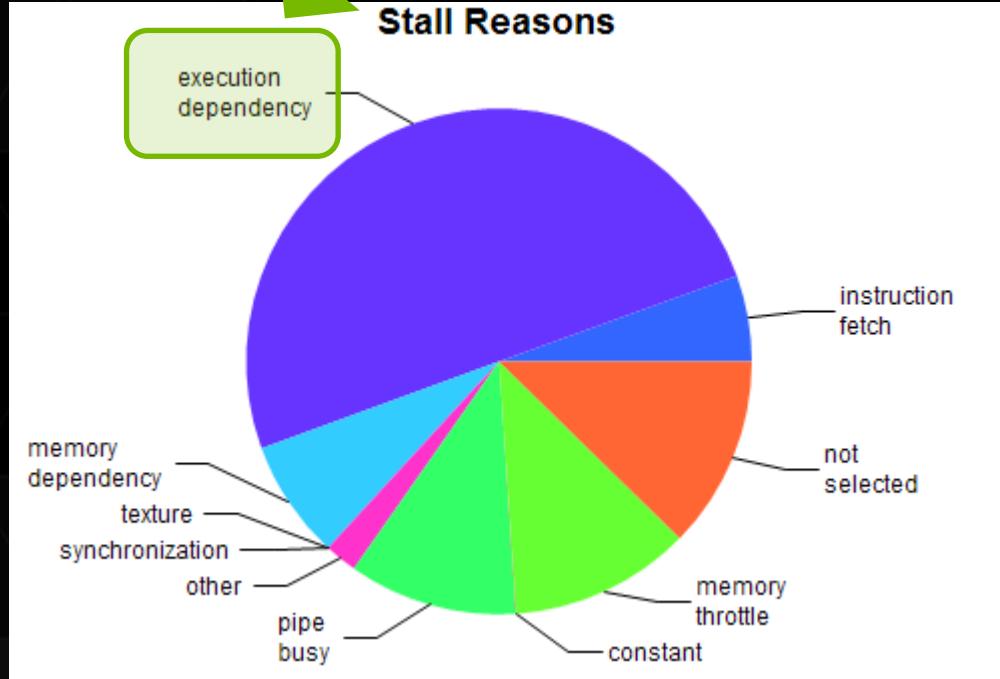
### **i Kernel Performance Is Bound By Instruction And Memory Latency**

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.



**Still Latency Bound**

# LOOKING FOR INDICATORS



Still high  
execution  
dependency,  
but  
occupancy OK

# LOOKING FOR MORE INDICATORS

The screenshot shows the NVIDIA Visual Profiler interface. On the left, under '1. CUDA Application Analysis', it says '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\_y0" is most likely limited by instruction and memory latency.' Below this are buttons for 'Perform Latency Analysis' and 'Perform Compute Analysis'. A green arrow points to the 'Perform Memory Bandwidth Analysis' button. At the bottom, it says 'Compute and memory bandwidth are likely not the'.

| L2 Cache          |          |              |  |
|-------------------|----------|--------------|--|
| L1 Reads          | 11568392 | 236.713 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 |  |

Is our working set mostly in L2?

# CHECKING L2 HIT RATE: 98.9%

The screenshot shows the NVIDIA Nsight Compute interface. On the left, the 'Run' tab is selected in the menu bar. Under the 'Run' tab, the 'Collect Metrics and Events' option is highlighted with a blue selection bar. On the right, the 'Metrics' tab is selected in a tab bar. A tree view under 'Metrics' shows various cache-related metrics: Cache, L1 Global Hit Rate, L1 Local Hit Rate, L2 Hit Rate (L1 Reads) (which is checked), L2 Hit Rate (Texture Reads), L2 Throughput (L1 Reads), L2 Throughput (Texture Reads), L2 Throughput (Reads), L2 Throughput (Writes), and L1/Shared Memory Utilization. Below this, a table displays performance data for a specific function:

| 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?



# 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

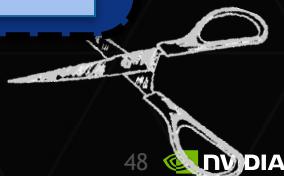
Significant speedup, < 1ms

| Kernel                 | Time    | Speedup | Incremental Speedup |
|------------------------|---------|---------|---------------------|
| Original Version       | 5.233ms | 1.00x   |                     |
| Better Memory Accesses | 1.589ms | 3.29x   | 3.29x               |
| Higher Occupancy       | 1.562ms | 3.35x   | 1.02x               |
| Shared Memory          | 0.911ms | 5.74x   | 1.7x                |

# PERF-OPT QUICK REFERENCE CARD

## Category: Latency Bound - Shared Memory

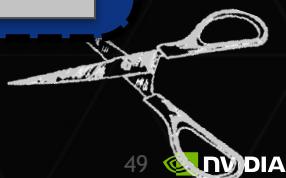
|             |  |
|-------------|--|
| Problem:    | Long memory latencies are difficult to hide  |
| Goal:       | <u>Reduce</u> latency, move data to <u>faster</u> memory   |
| Indicators: | Shared memory not occupancy limiter<br>High L2 hit rate<br>Data reuse between threads and small-ish working 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

## 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<br>Low arithmetic intensity of the kernel  |
| Strategy:   | (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> |



*Iteration 4*

# IDENTIFY HOTSPOT

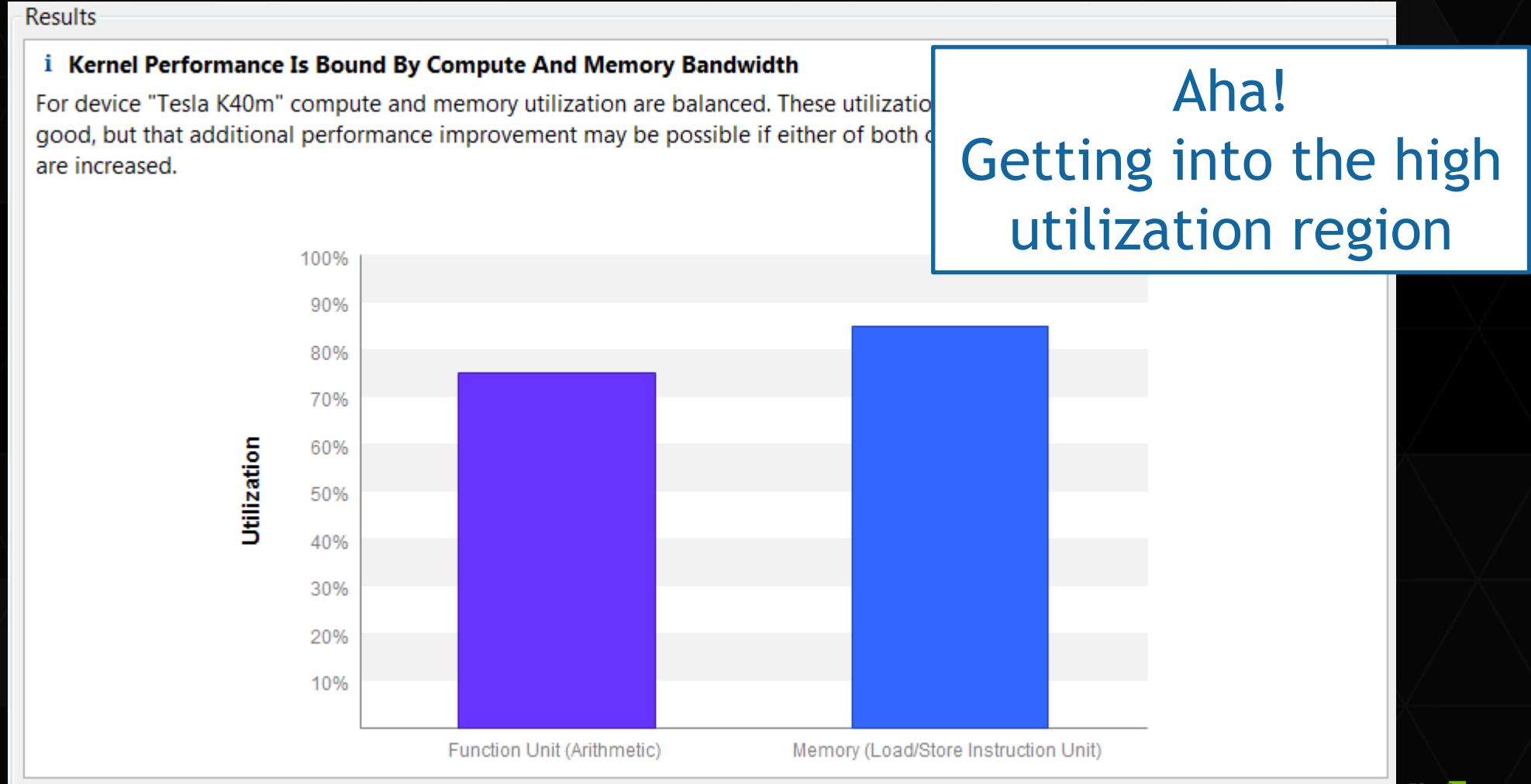
Hotspot →

| Results   |  |
|---|--|
| <b>i Kernel Optimization Priorities</b>   |  |
| The following kernels are ordered by optimization importance based on execution time and achieved optimization potential. 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

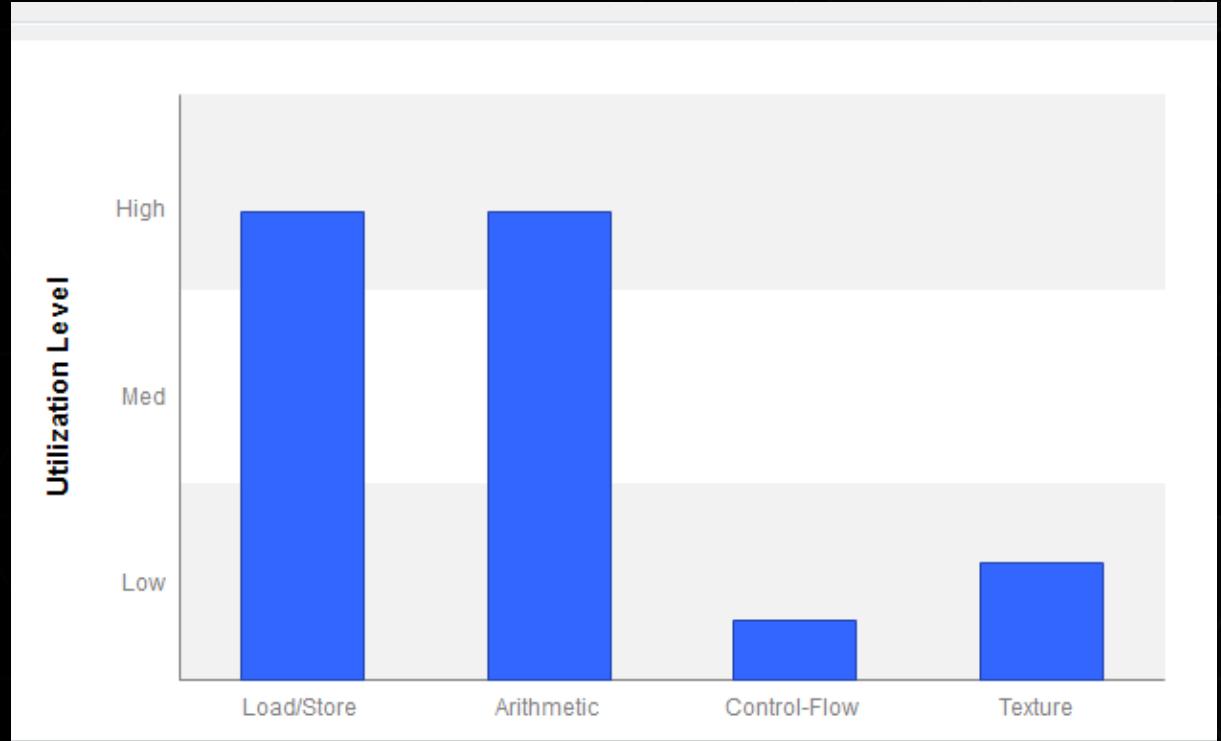
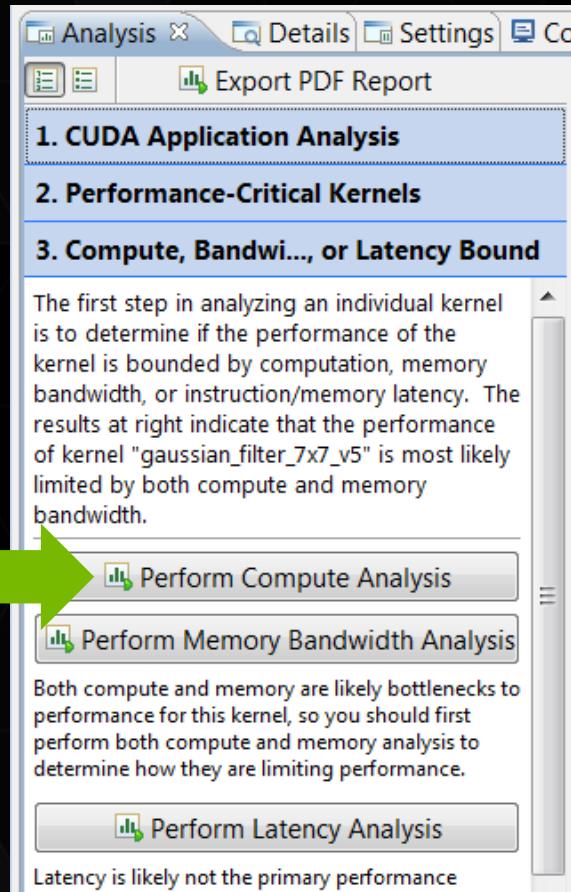
| Kernel                 | Time    | Speedup | Incremental Speedup |
|------------------------|---------|---------|---------------------|
| Original Version       | 5.233ms | 1.00x   |                     |
| Better Memory Accesses | 1.589ms | 3.29x   | 3.29x               |
| Higher Occupancy       | 1.562ms | 3.35x   | 1.02x               |
| Shared Memory          | 0.911ms | 5.74x   | 1.7x                |

# IDENTIFY PERFORMANCE LIMITER

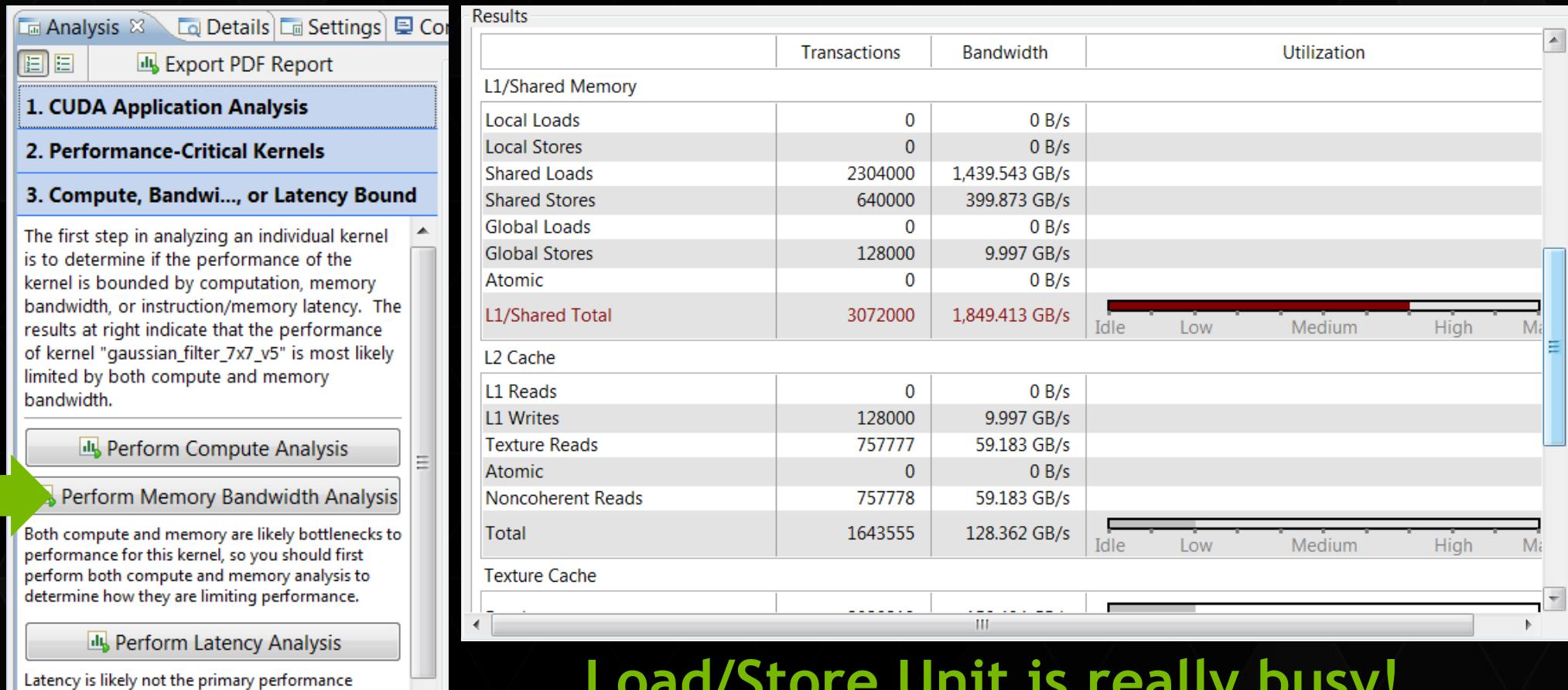


# LOOKING FOR INDICATORS

Launch



# LOOKING FOR MORE INDICATORS

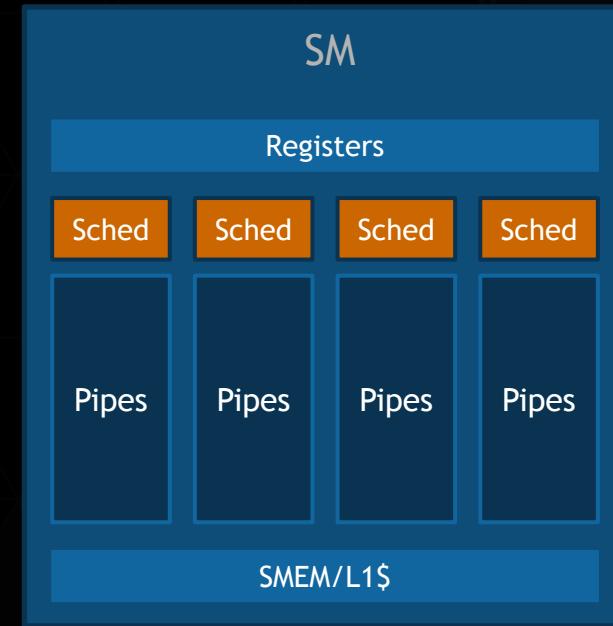


Load/Store Unit is really busy!  
Can we reduce the load?



# INSTRUCTION THROUGHPUT

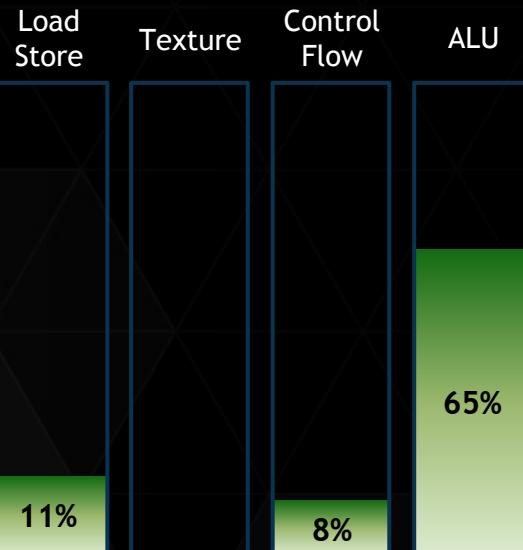
- ▶ Each SM has 4 schedulers (Kepler)
- ▶ Schedulers issue instructions to pipes
- ▶ A scheduler issues up to 2 instructions/cycle
  - ▶ Sustainable peak is 7 instructions/cycle per SM (not  $4 \times 2 = 8$ )
- ▶ A scheduler issues inst. from a single warp
- ▶ Cannot issue to a pipe if its issue slot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



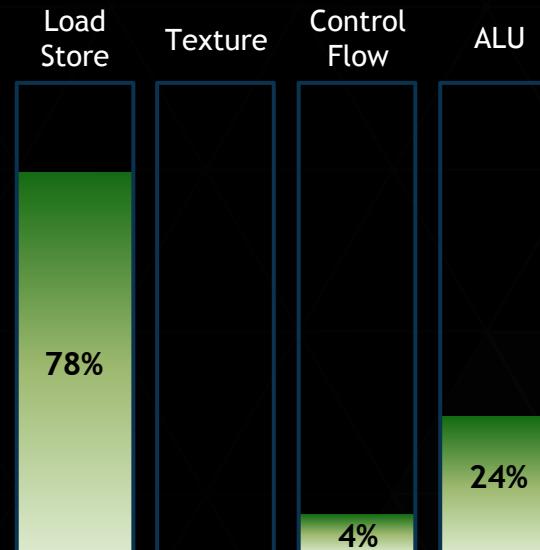
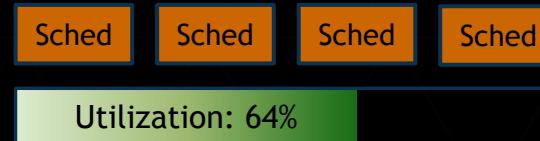


# INSTRUCTION THROUGHPUT

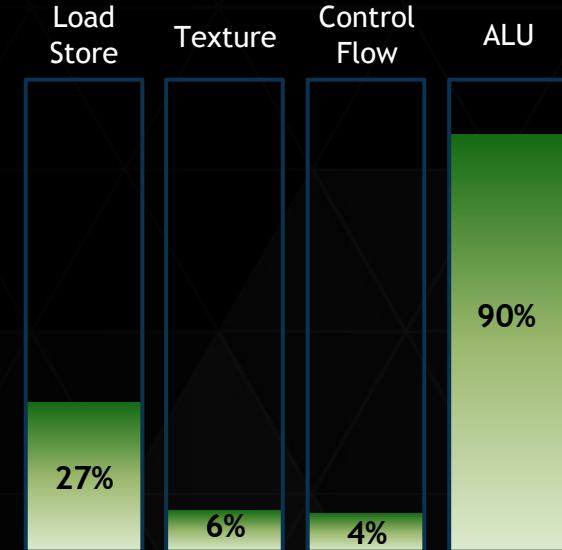
Schedulers saturated



Pipe saturated

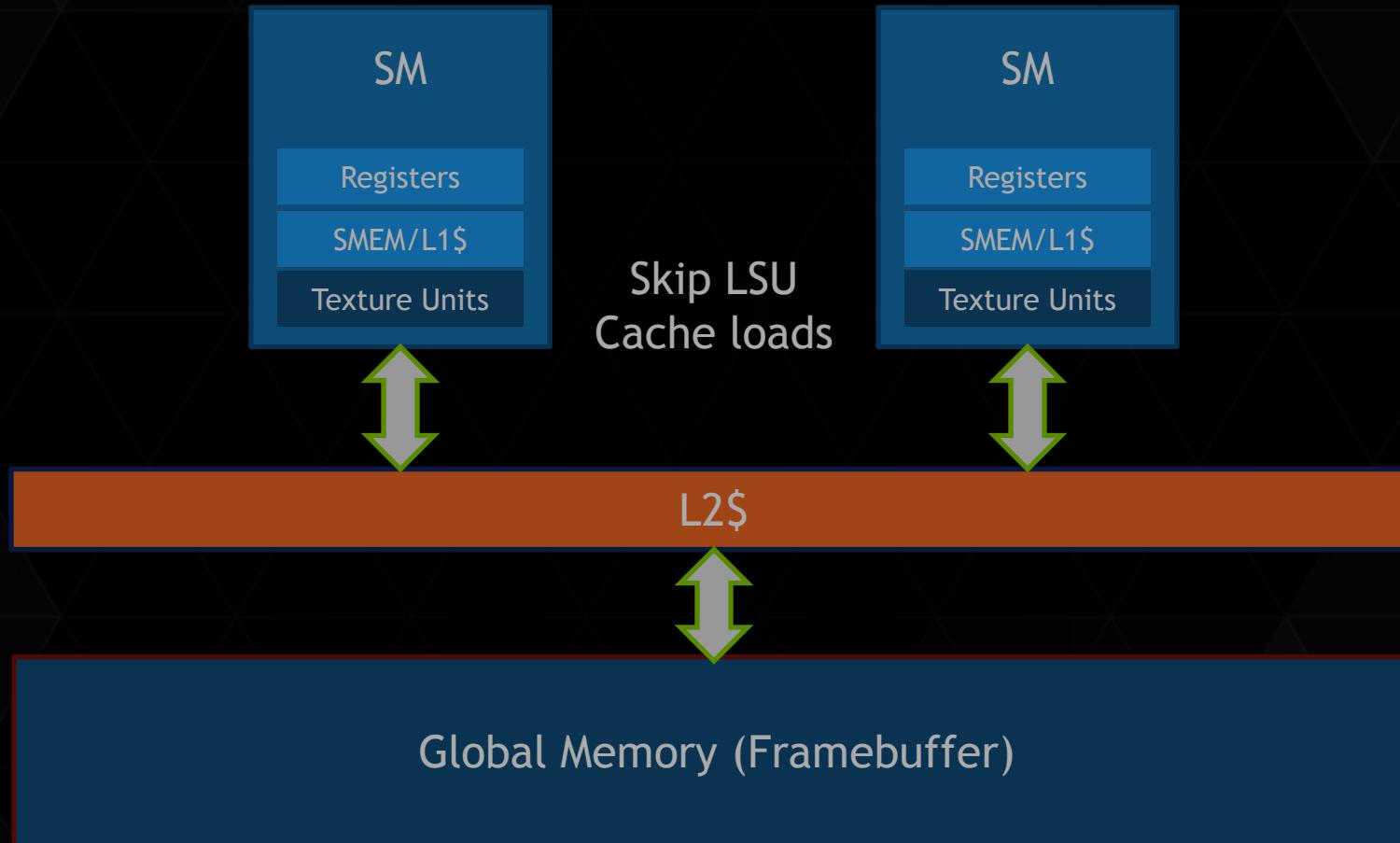


Schedulers and pipe saturated





# 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 | Incremental Speedup |
|------------------------|---------|---------|---------------------|
| Original version       | 5.233ms | 1.00x   |                     |
| Better memory accesses | 1.589ms | 3.29x   | 3.29x               |
| Higher Occupancy       | 1.562ms | 3.35x   | 1.02x               |
| Shared memory          | 0.911ms | 5.74x   | 1.7x                |
| Read-Only path         | 0.808ms | 6.48x   | 1.13x               |

# PERF-OPT QUICK REFERENCE CARD

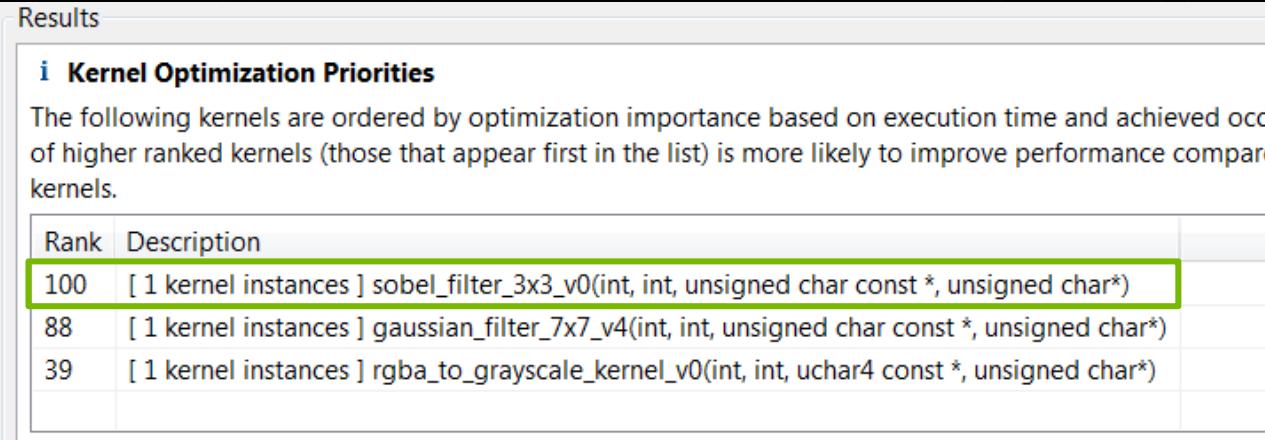
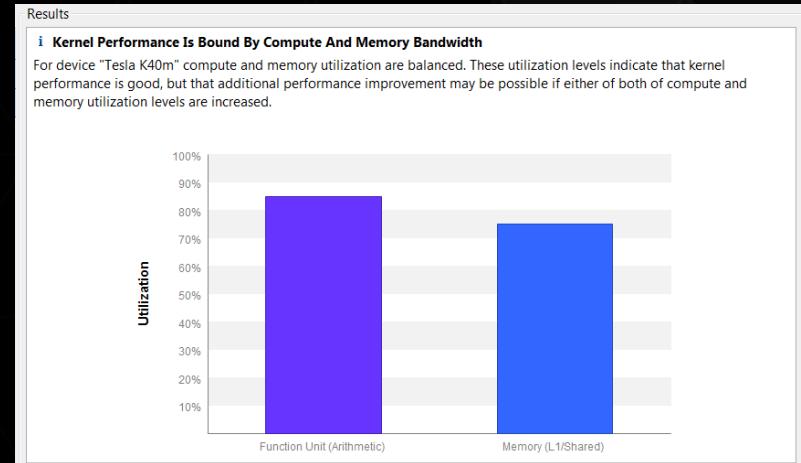
## Category: Latency Bound - Texture Cache

|             |   |
|-------------|---|
| 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:   | <p>Load read-only data through Texture Units:</p> <ul style="list-style-type: none"><li>• Annotate read-only pointers with <code>const __restrict__</code></li><li>• Use <code>__ldg()</code> intrinsic</li></ul> |



# 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



The figure is a screenshot of the NVIDIA Visual Profiler's "Results" window. It displays a section titled "Kernel Optimization Priorities". A note states: "The following kernels are ordered by optimization importance based on execution time and achieved occupancy. Higher ranked kernels (those that appear first in the list) are more likely to improve performance compared to lower ranked kernels." Below this, a table lists three 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>

# *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 Programming Guides
  - ▶ <http://docs.nvidia.com/cuda/>
    - ▶ Cuda Programming Guide
    - ▶ Best Practices Guide
    - ▶ Fermi/Kerpler/Maxwell/Pascal Tuning Guides
- ▶ Parallel Forall devblog
  - ▶ <http://devblogs.nvidia.com/parallelforall/>
- ▶ Vasily Volkov better performance at lower occupancy
  - ▶ [http://www.nvidia.com/content/gtc-2010/pdfs/2238\\_gtc2010.pdf/](http://www.nvidia.com/content/gtc-2010/pdfs/2238_gtc2010.pdf/)

*Thank YOU*