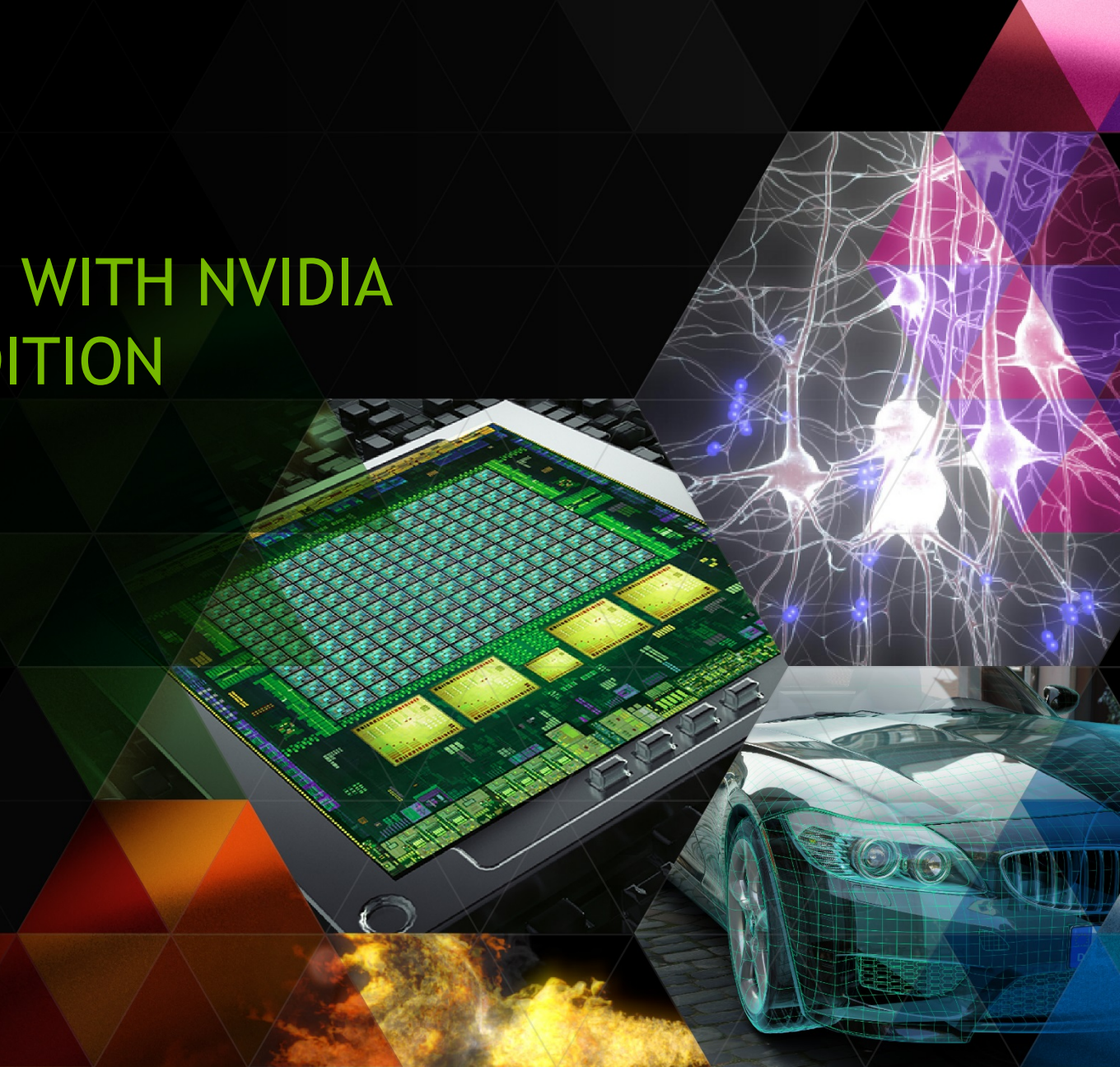
NVIDIA

CUDA OPTIMIZATION WITH NVIDIA NSIGHT™ ECLIPSE EDITION

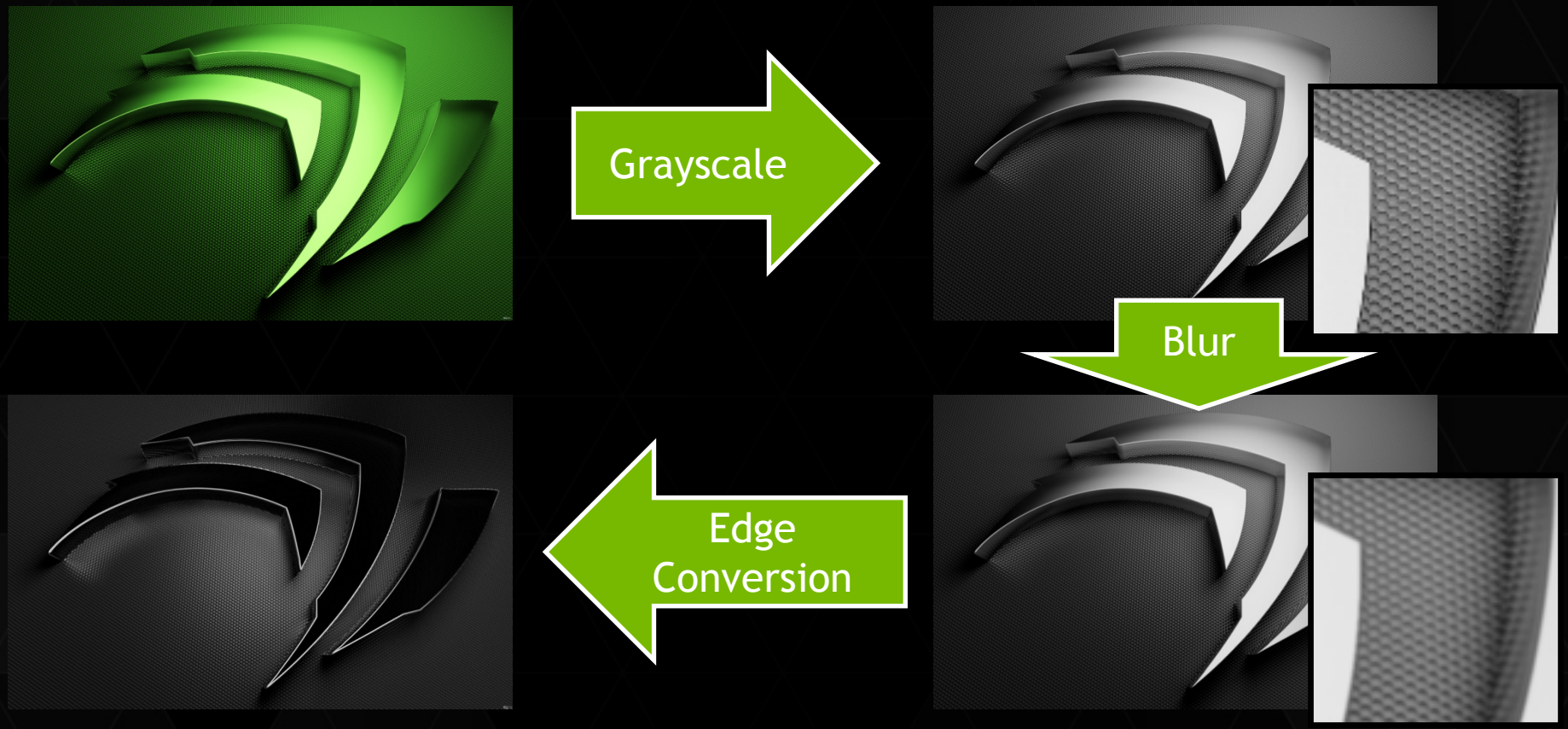


WHAT YOU WILL LEARN

- 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



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

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

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

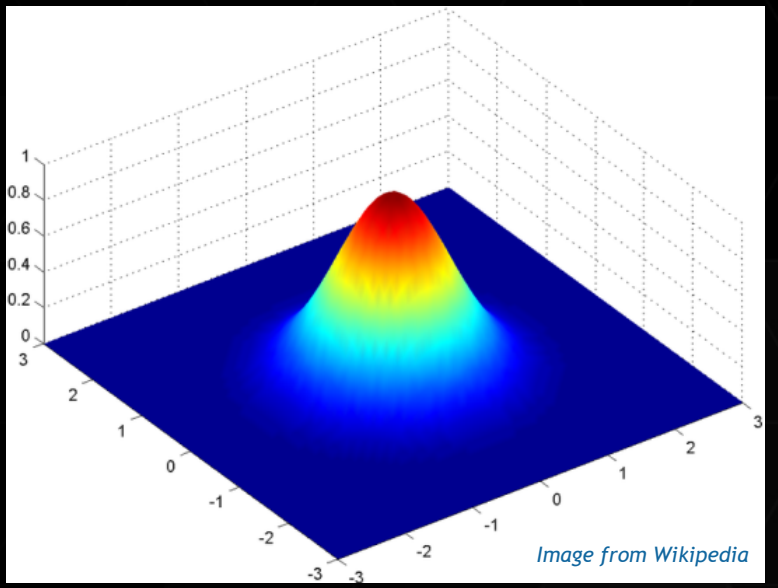
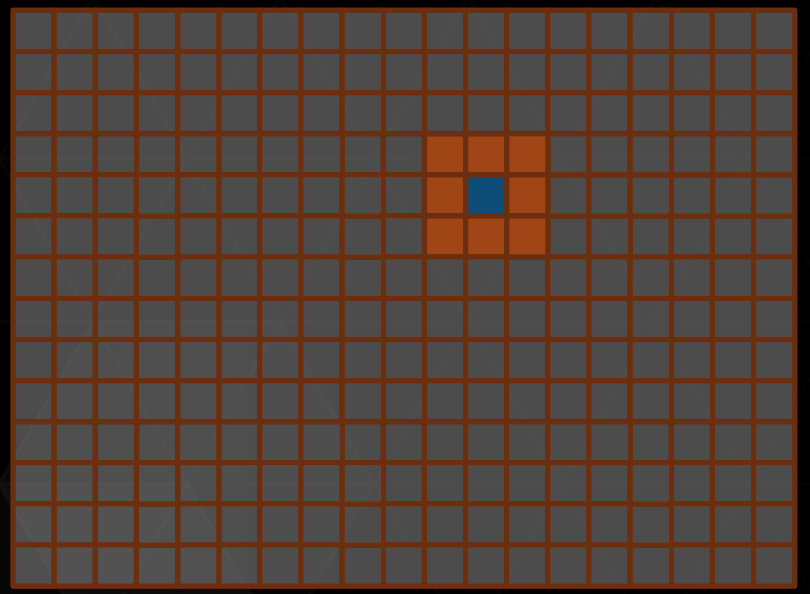


Image from Wikipedia

INTRODUCING THE APPLICATION

Edge conversion (Sobel Filter)

3x3 Sobel Filter



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

Similar results are obtained on Windows

PERFORMANCE OPTIMIZATION CYCLE

1. Profile Application

2. Identify Performance Limiter

3. Analyze Profile & Find Indicators

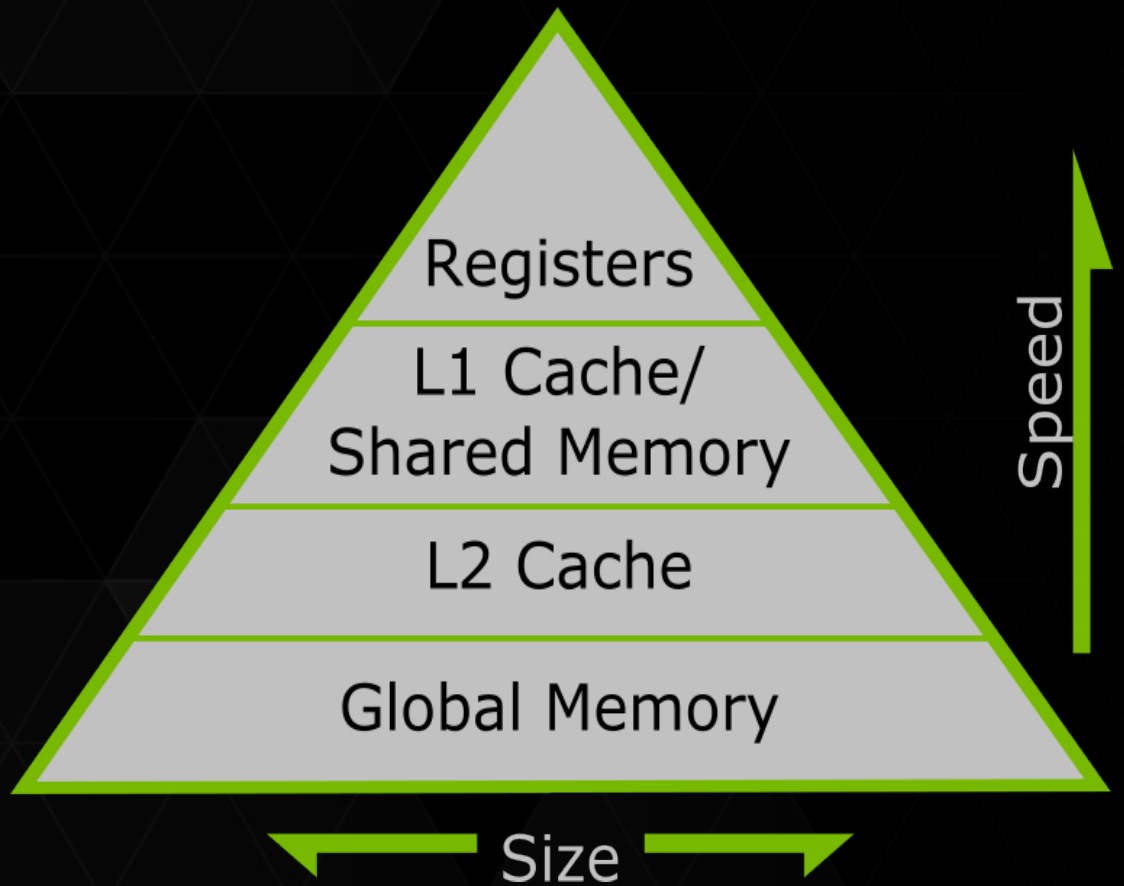
4. Reflect

4b. Build Knowledge

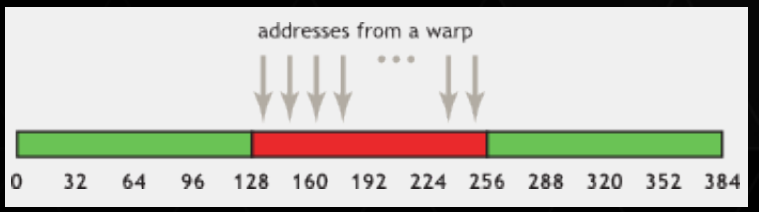
5. Change and Test Code

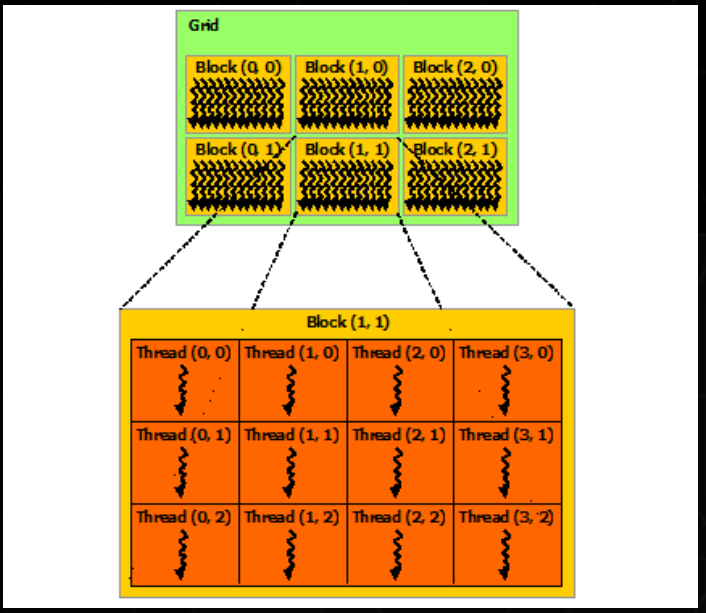
PREREQUISITES

GPU Memory Spaces



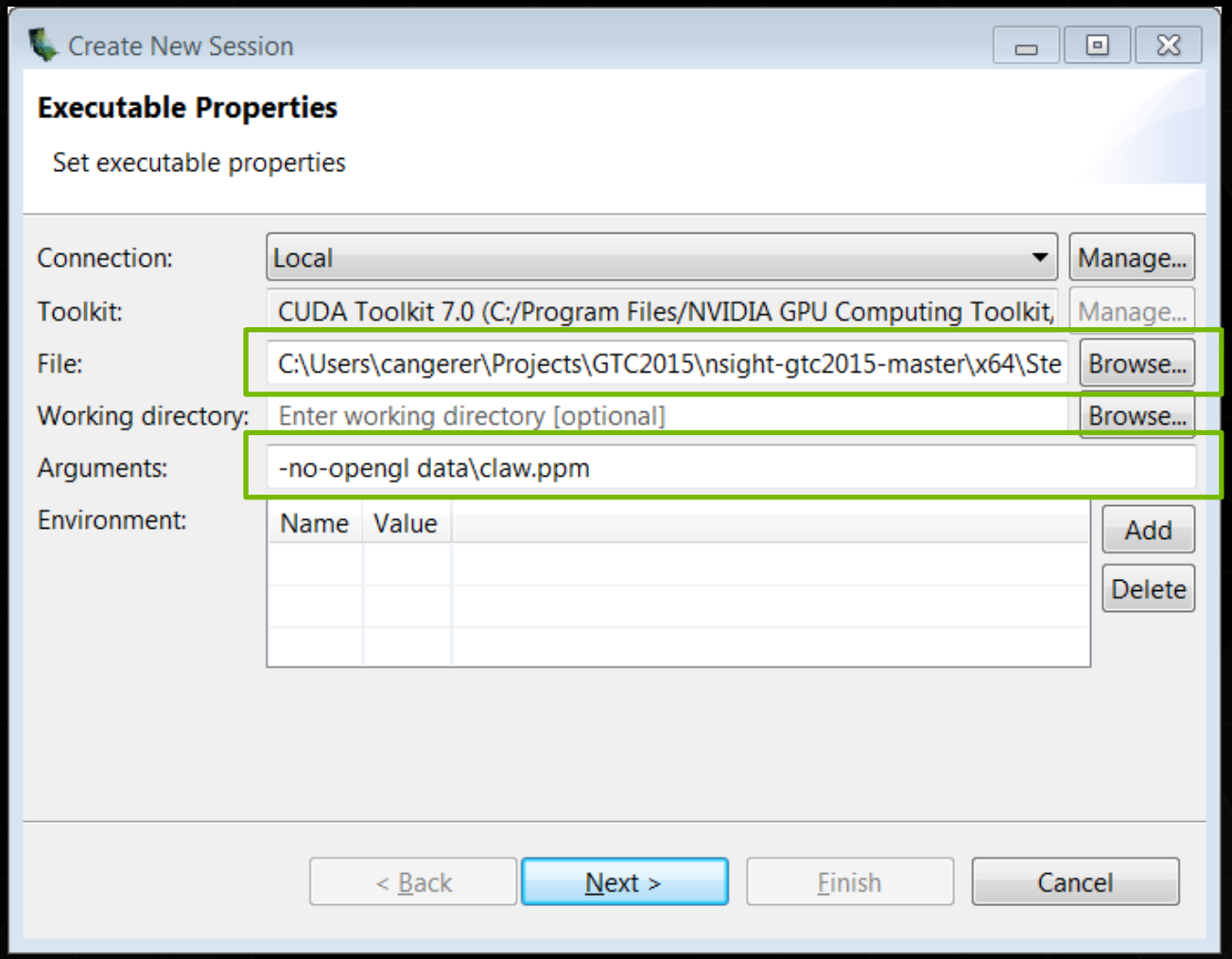
CUDA Execution Model



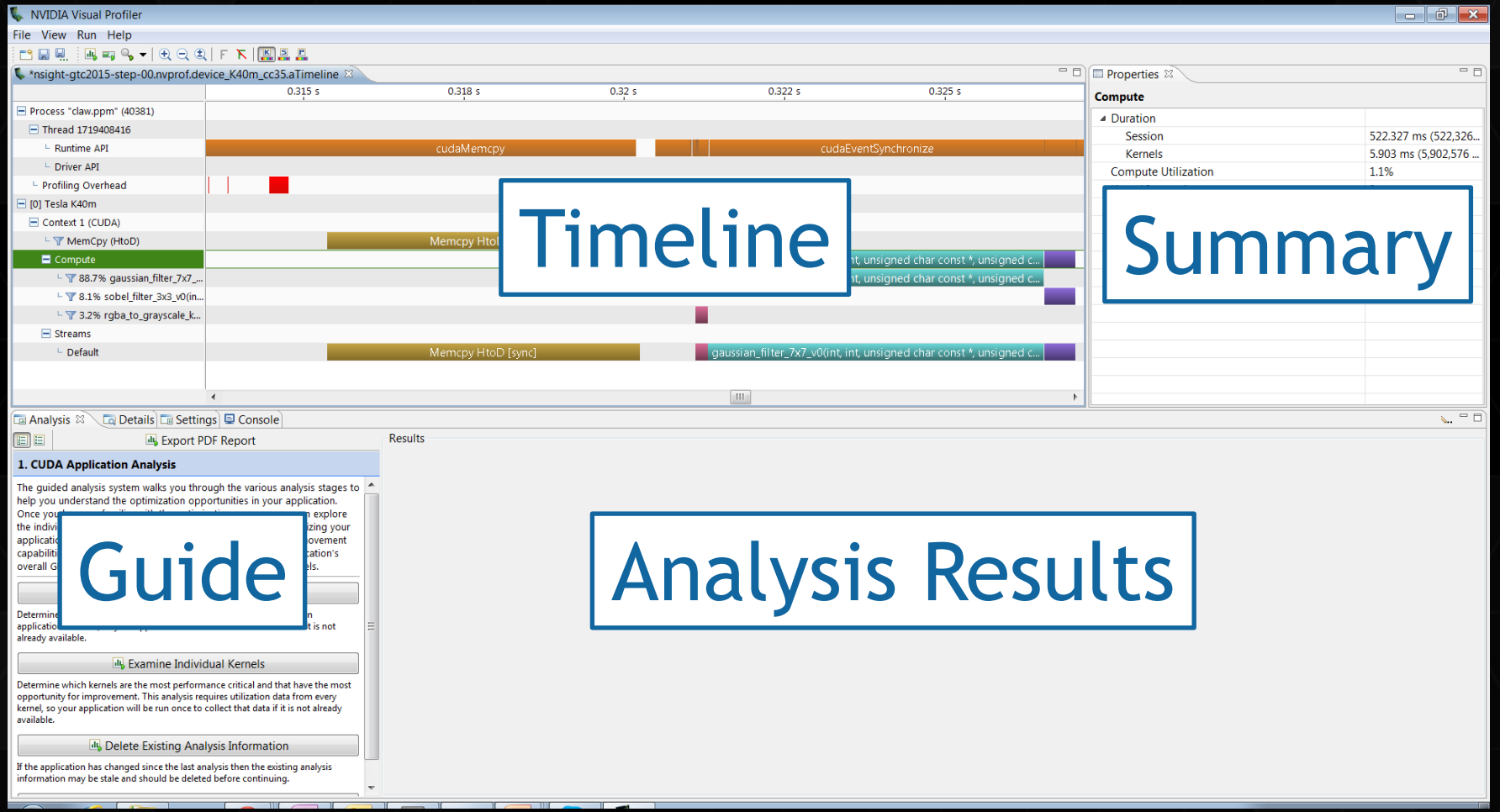


Iteration 1

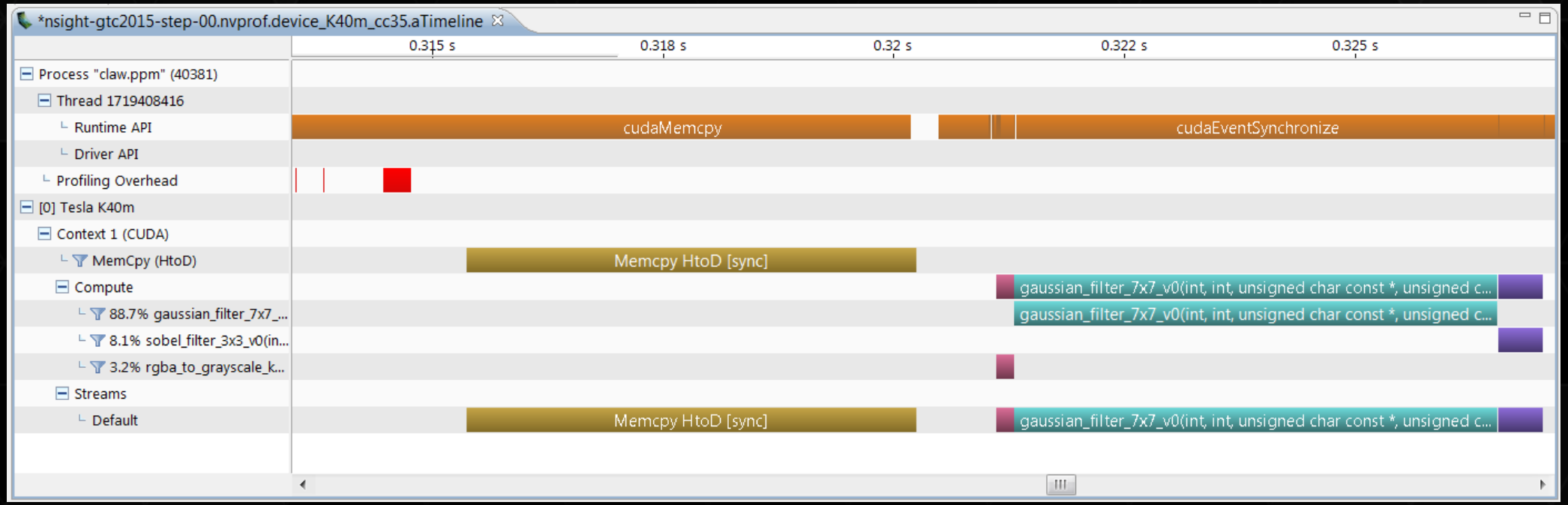
CREATE A NEW NVVP SESSION



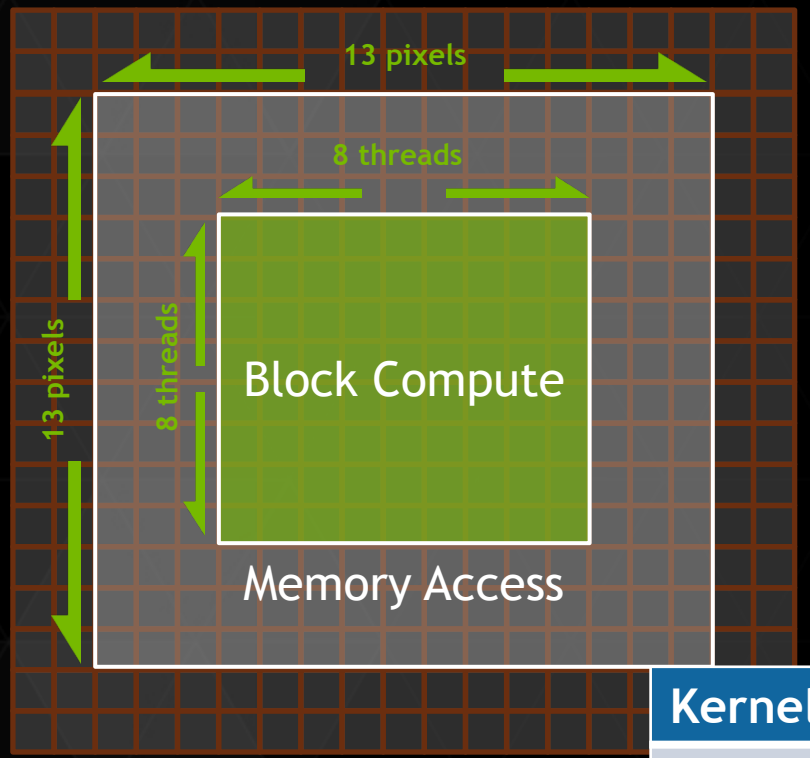
THE PROFILER WINDOW



TIMELINE



7X7 FILTER KERNEL

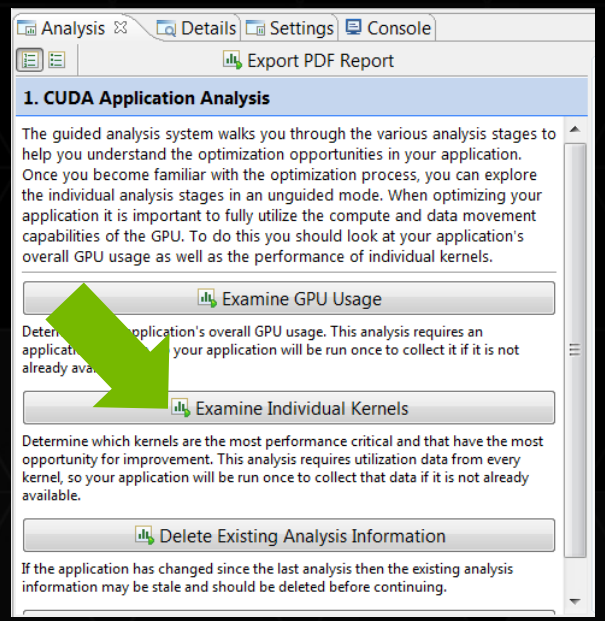


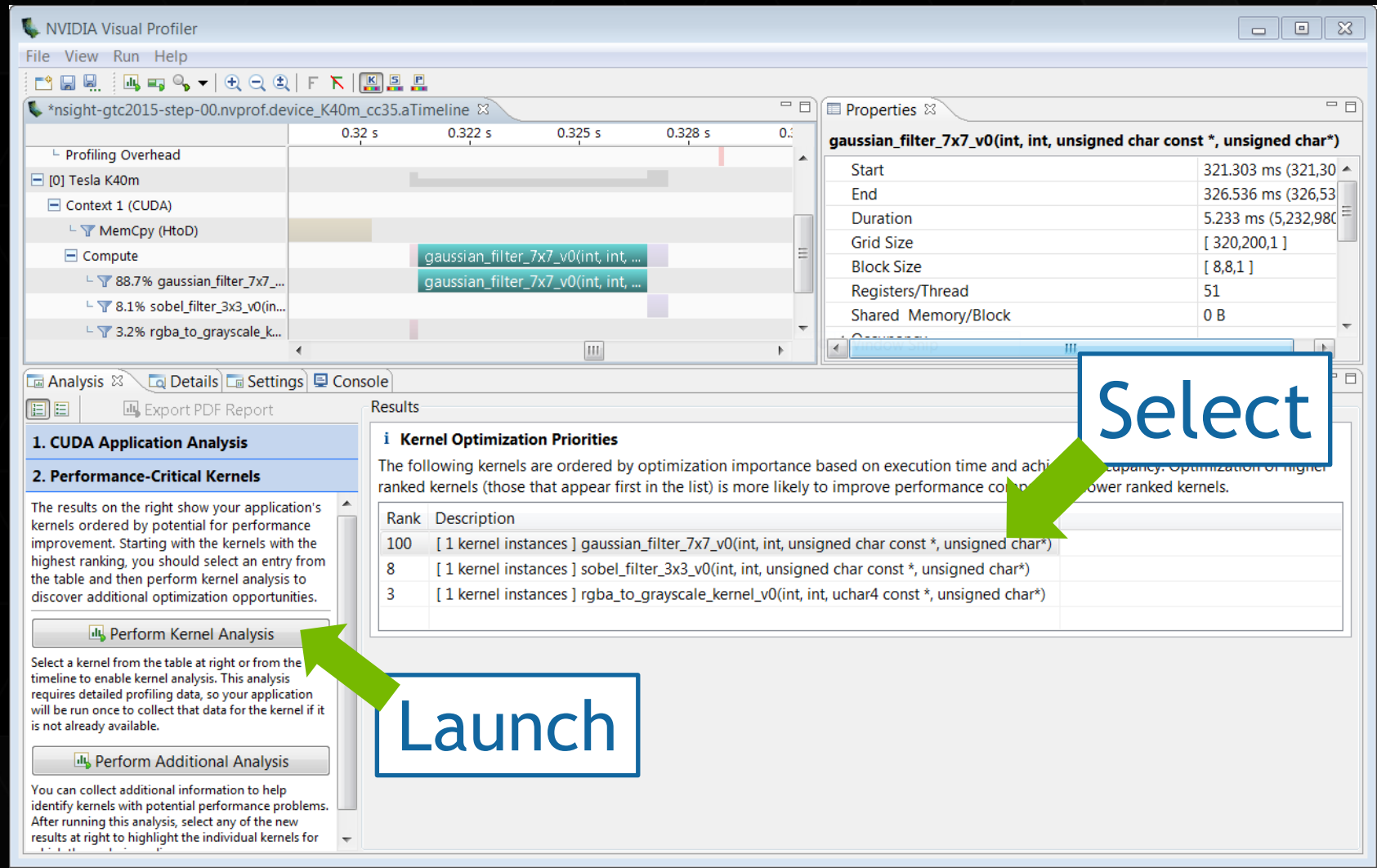
Each thread first loads 7x7 uchar, convert to int, compute convolution for a pixel, store as uchar.

Image is 2560 x 1600 (~4M pixels)

Kernel	Time	Speedup
Original Version	5.233ms	1.00x

PERFORM KERNEL ANALYSIS



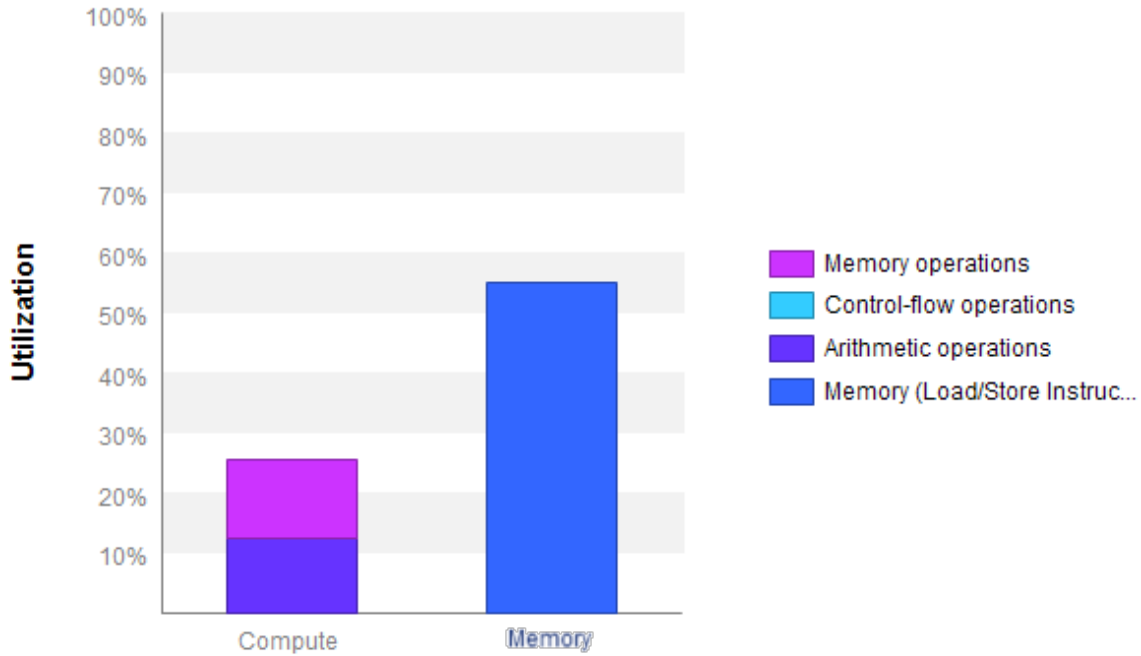


IDENTIFY PERFORMANCE LIMITER

Results

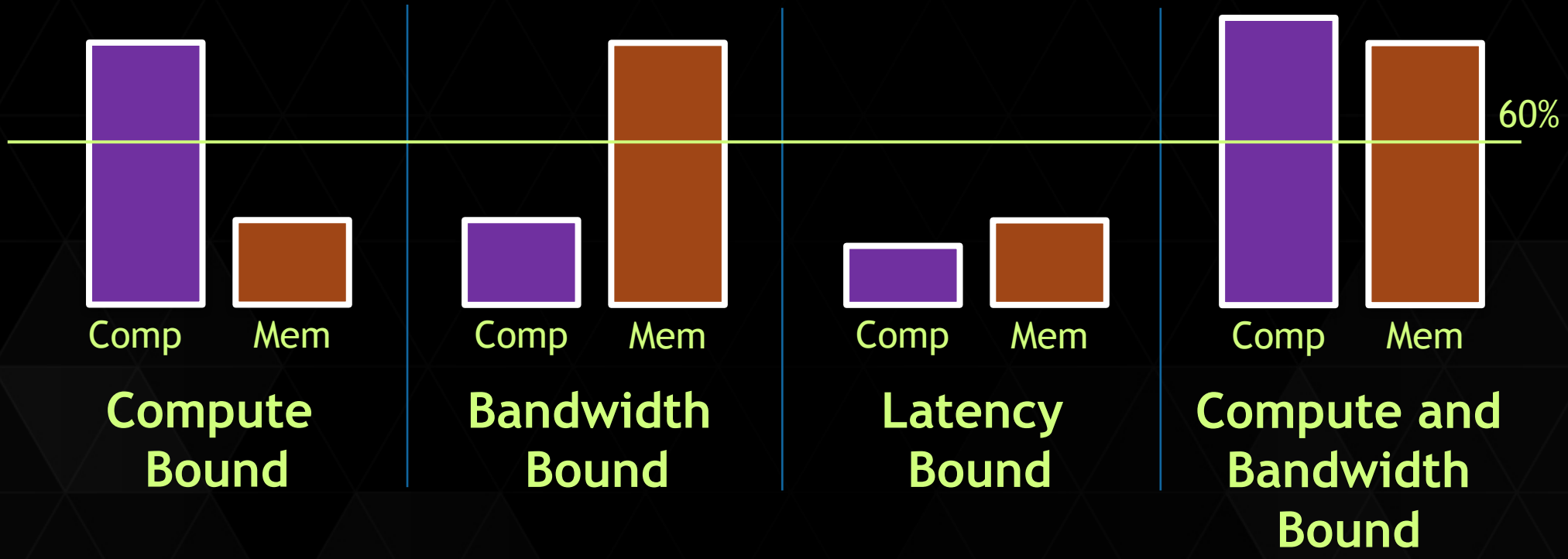
Kernel Performance Is Bound By Instruction And Memory Latency

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

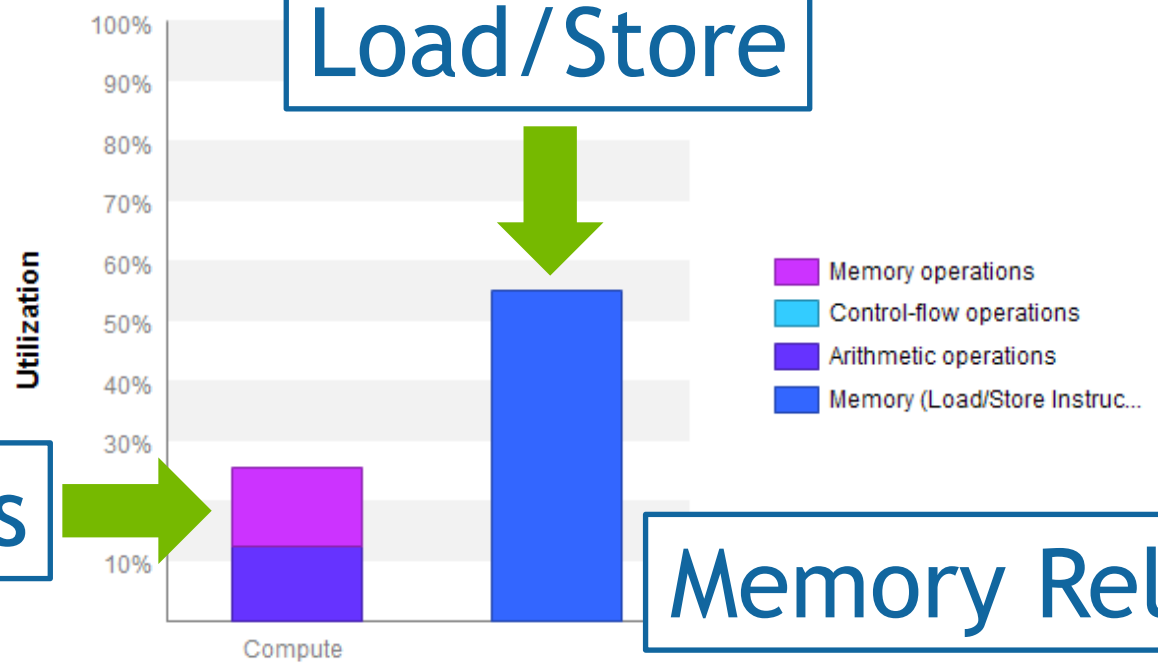


IDENTIFY PERFORMANCE LIMITER

Results

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.

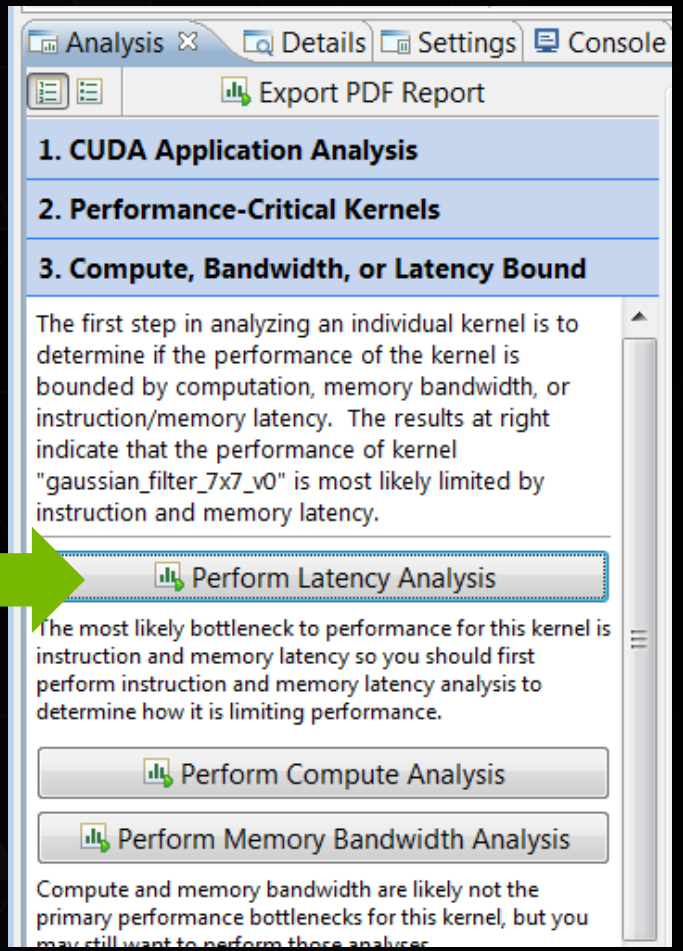


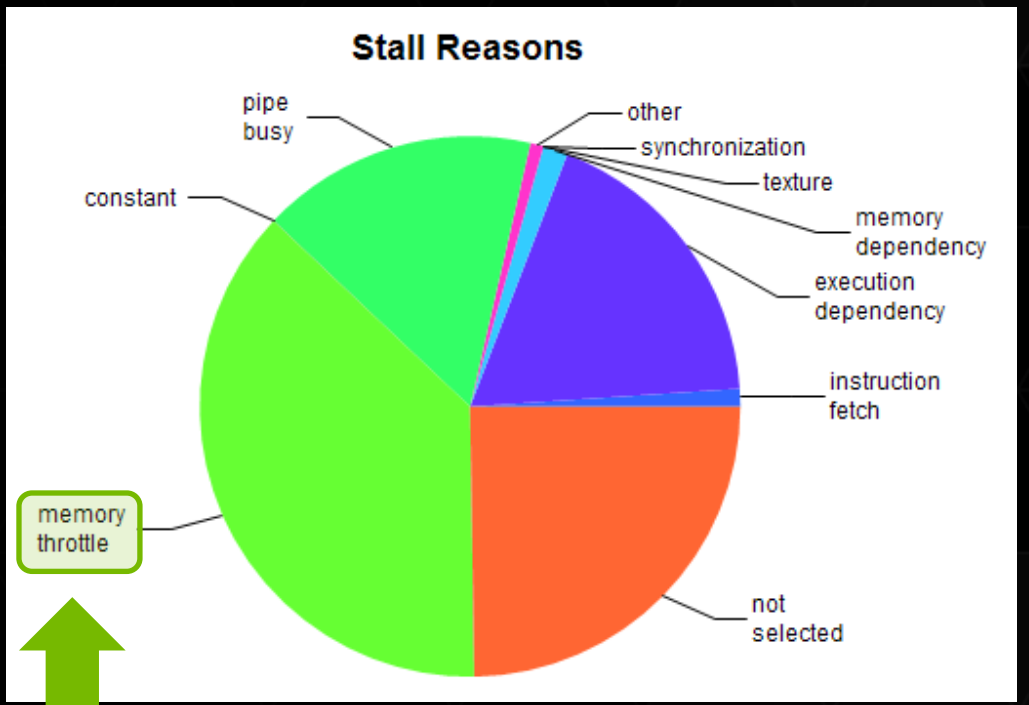
Memory Ops

Memory Related Issues?

LOOKING FOR INDICATORS

Launch

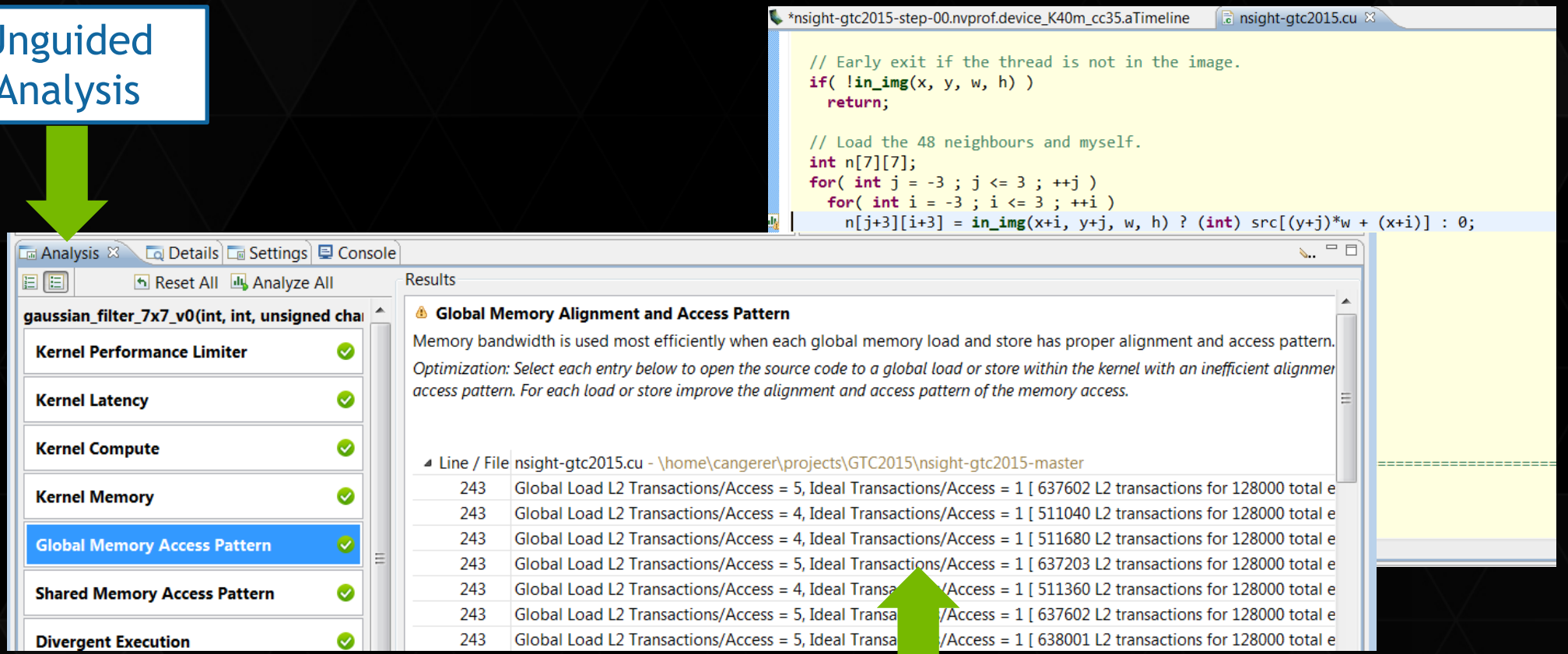




Large number of memory operations stalling LSU

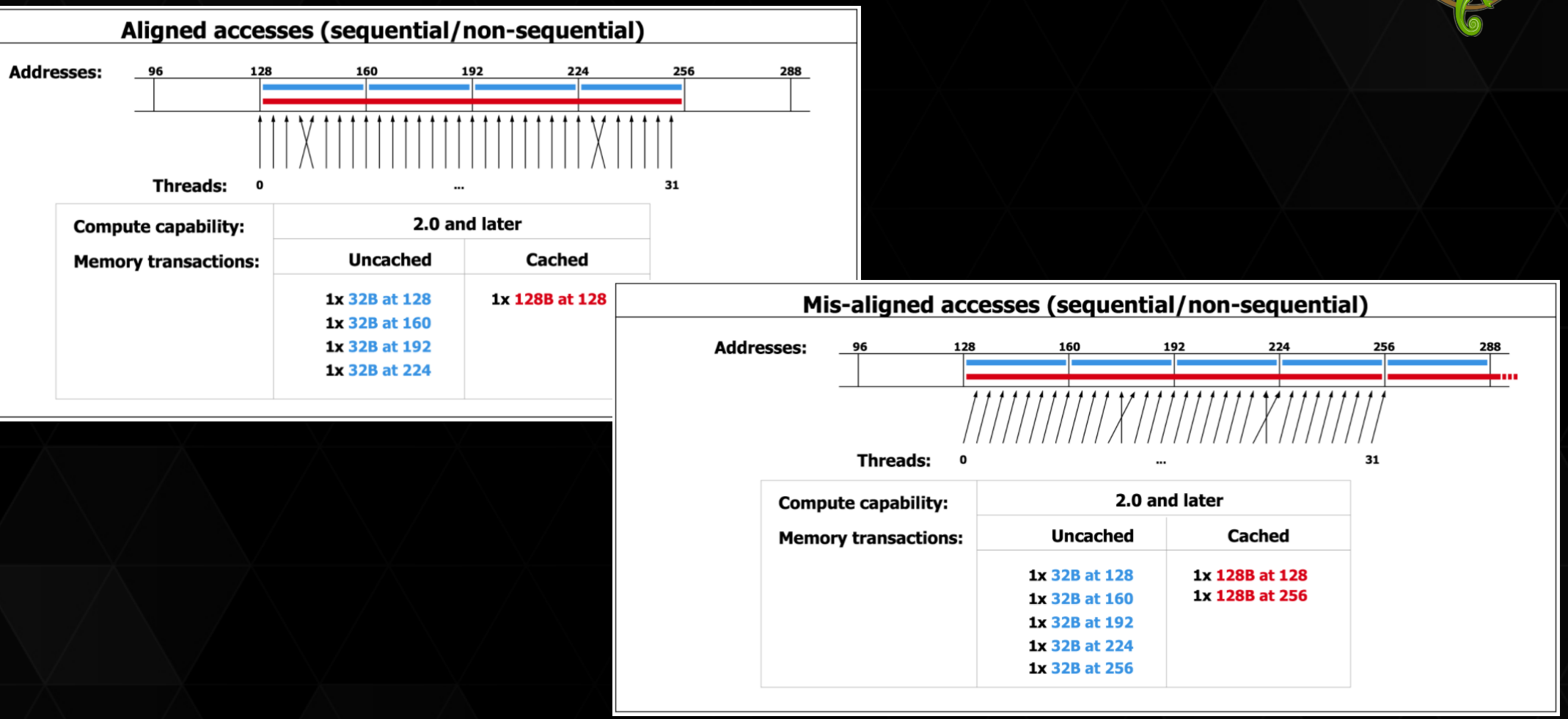
LOOKING FOR MORE INDICATORS

Unguided Analysis



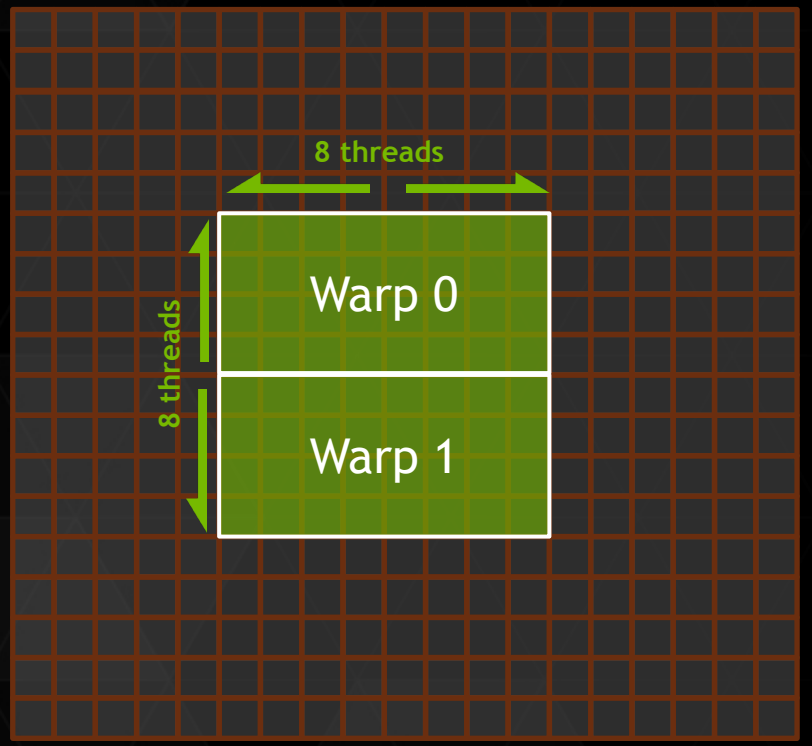
4-5 Global Load/Store Transactions per 1 Request

GLOBAL MEMORY TRANSACTIONS: REVIEW

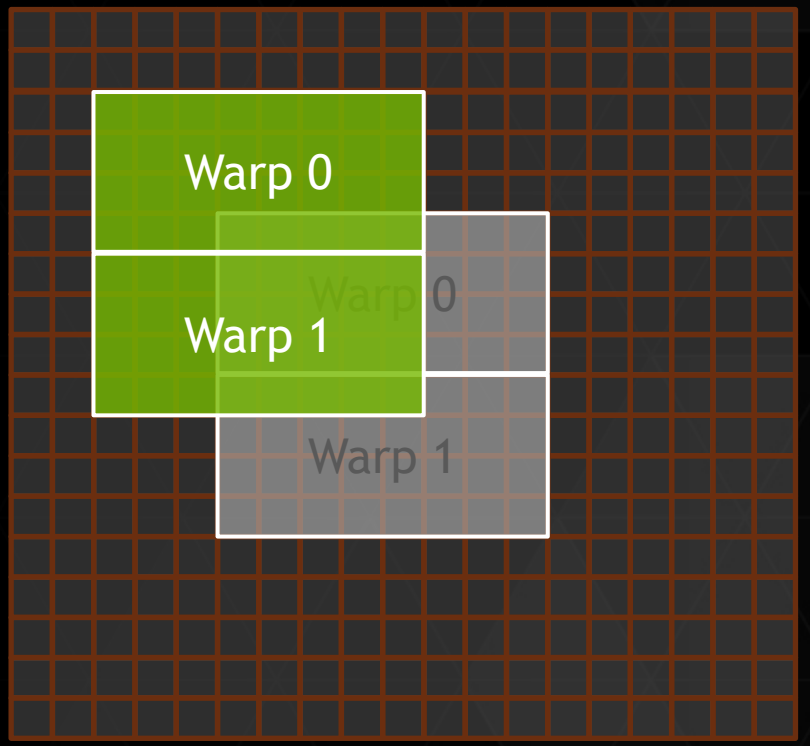


7X7 FILTER KERNEL A WARP VIEW

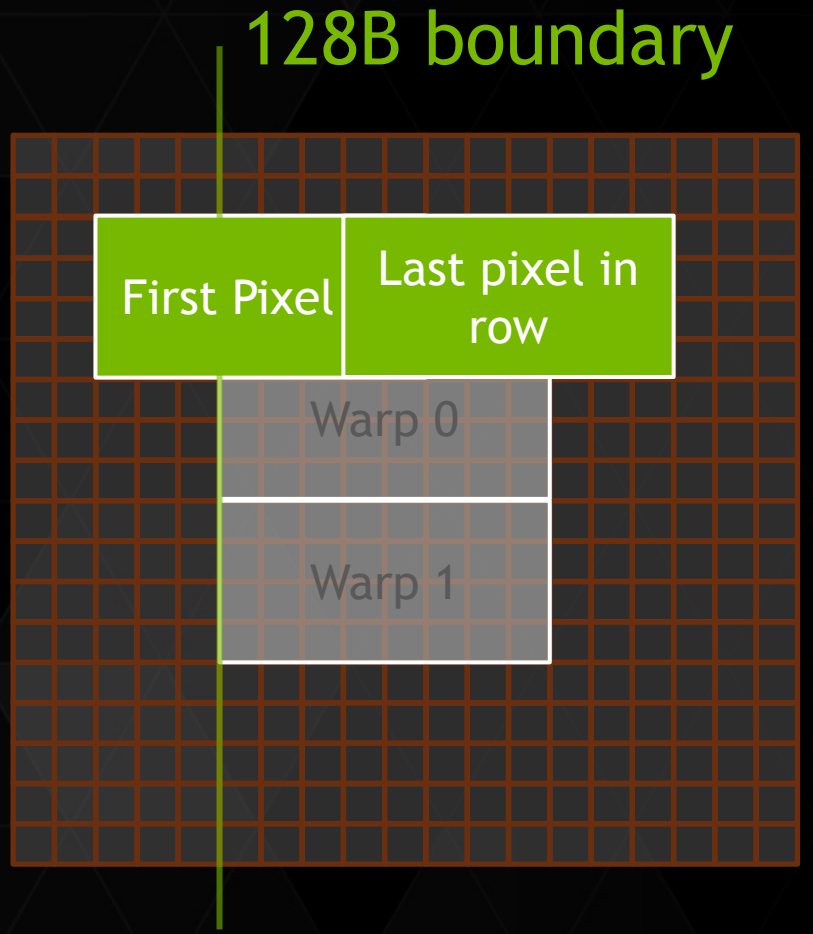
Computing



Loading the first pixel



7X7 FILTER KERNEL A WARP VIEW



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

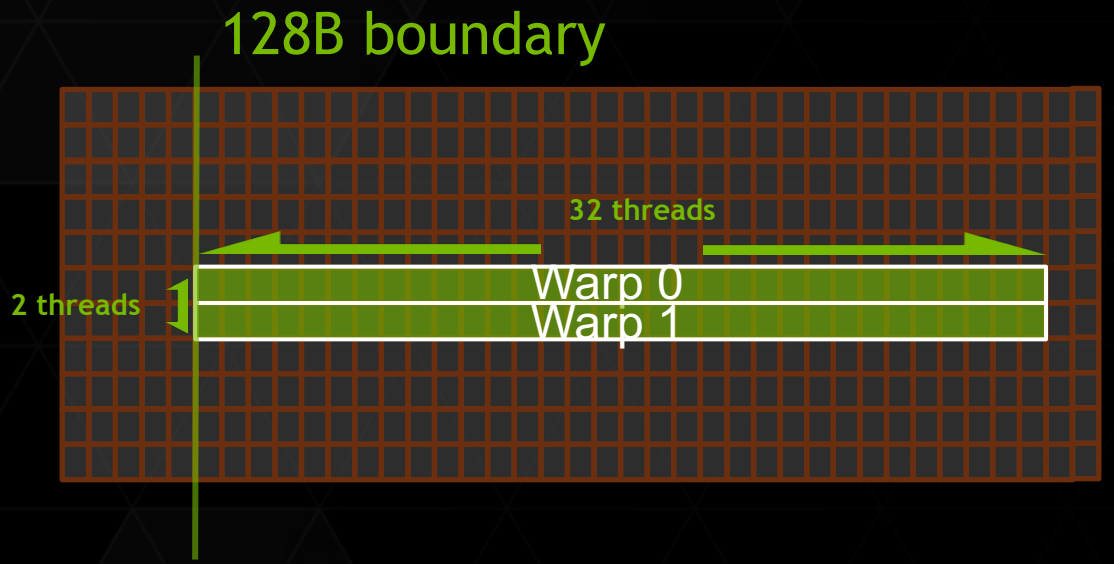
Warps will be loading 4 lines at a time.

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, High #transactions/#request compared to ideal
Strategy:	Improve memory coalescing by: • Cooperative loading inside a block • Change block layout • Aligning data • Changing data layout to improve locality

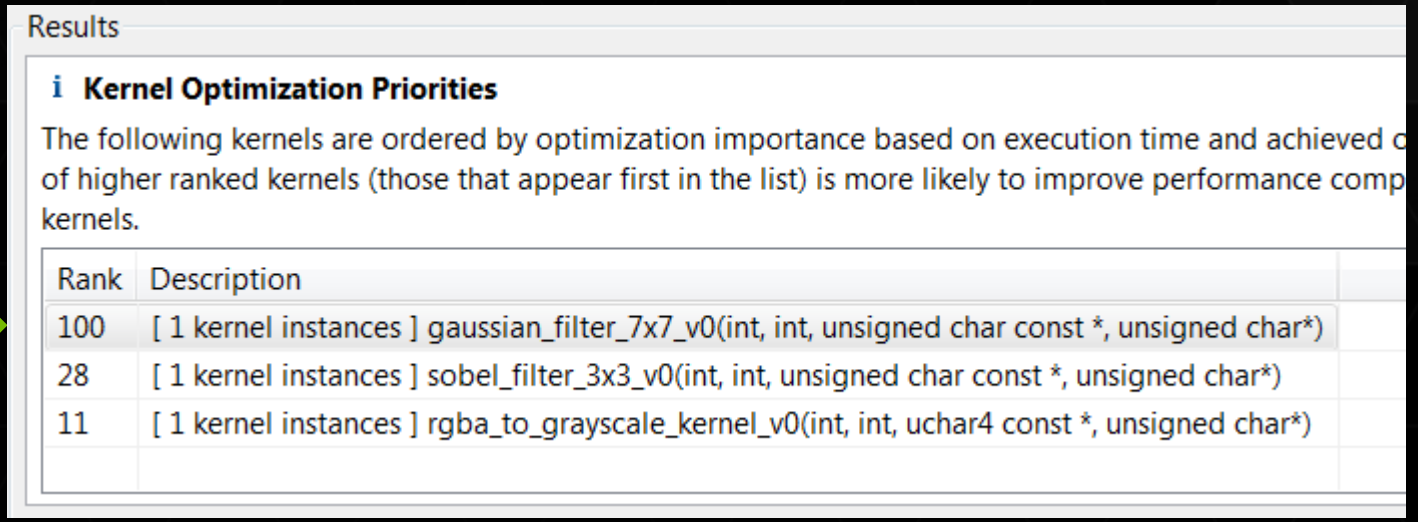
PERF-OPT QUICK REFERENCE CARD

Category:	Bandwidth Bound - Coalescing
Problem:	Too much unused data clogging memory system
Goal:	Reduce traffic, move more useful data per request
Indicators:	Low global load/store efficiency, High #transactions/#request compared to ideal
Strategy:	Improve memory coalescing by: • Cooperative loading inside a block • Change block layout • Aligning data • Changing data layout to improve locality

Iteration 2

IDENTIFY HOTSPOT

Hotspot



- gaussian_filter_7x7_v0() still the hotspot

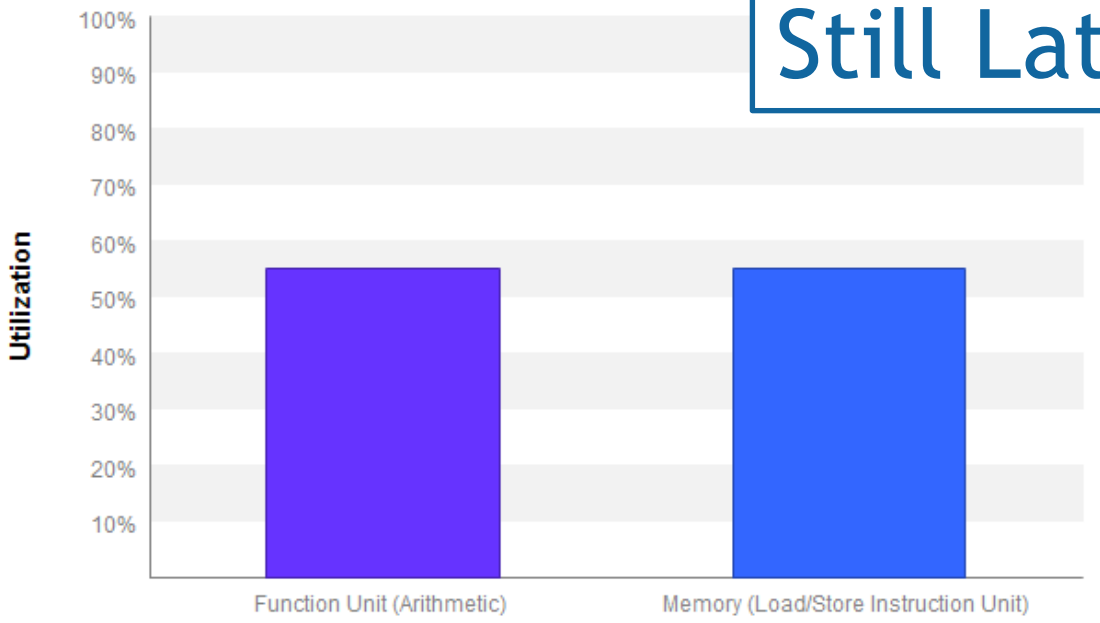
Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x

IDENTIFY PERFORMANCE LIMITER

Results

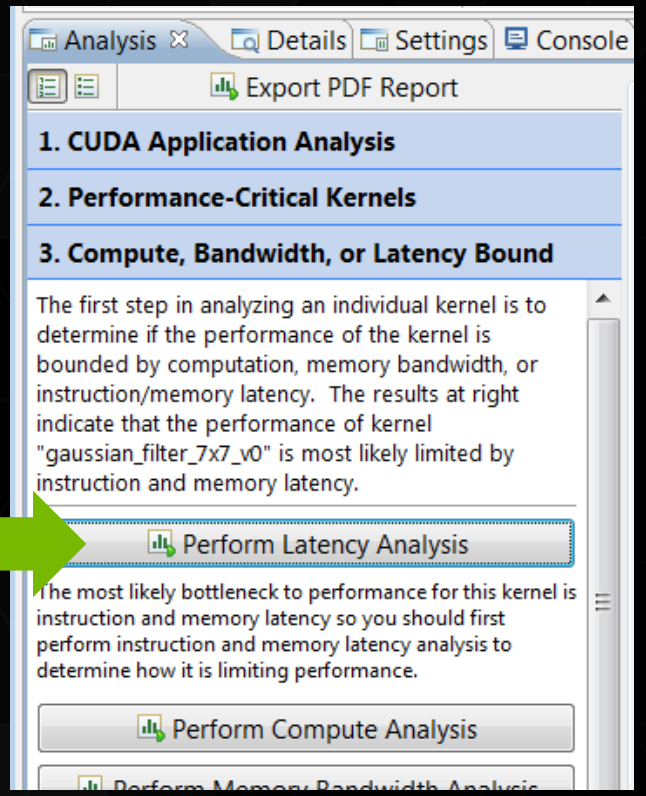
Kernel Performance Is Bound By Instruction And Memory Latency

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.

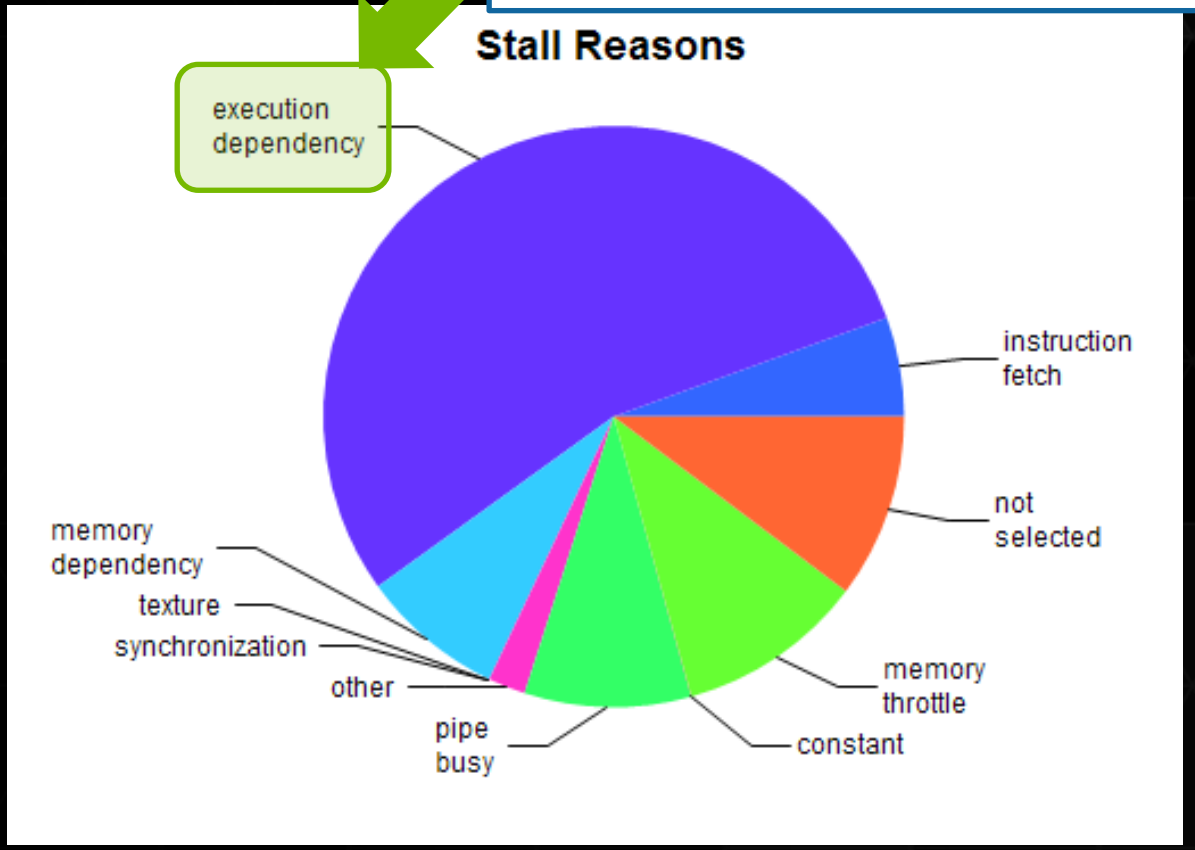


Still Latency Bound

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 = b[i];  // LOAD

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

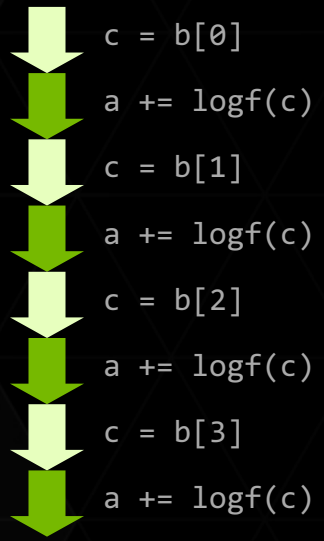
- Memory accesses may influence execution dependencies
  - Global accesses create longer dependencies than shared accesses
  - Read-only/texture dependencies are counted in Texture
- Instruction level parallelism can reduce dependencies

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

ILP AND MEMORY ACCESSES

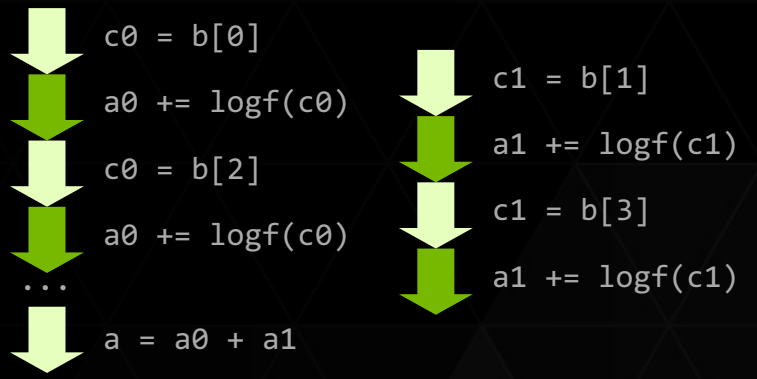
No ILP

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



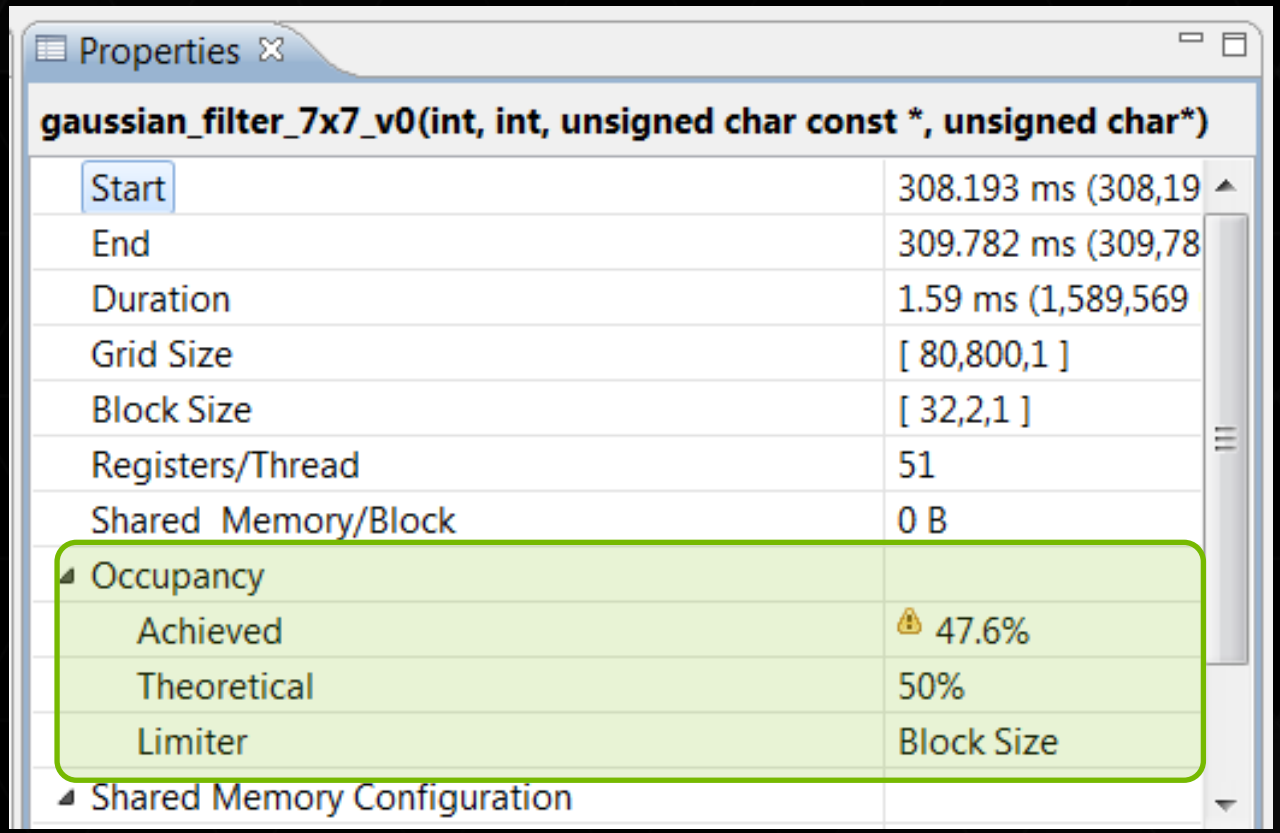
2-way ILP (with loop unrolling)

```
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
```



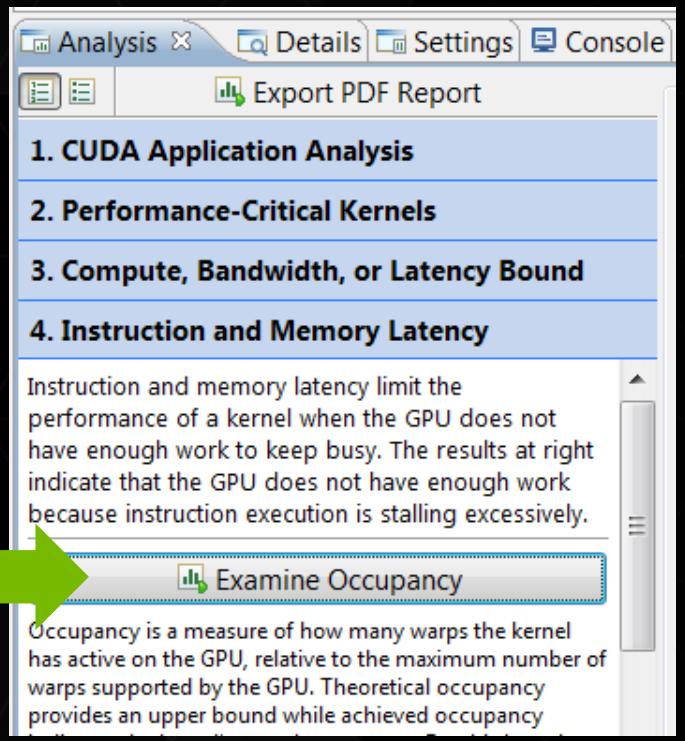
- #pragma unroll is useful to extract ILP
- Manually rewrite code if not a simple loop

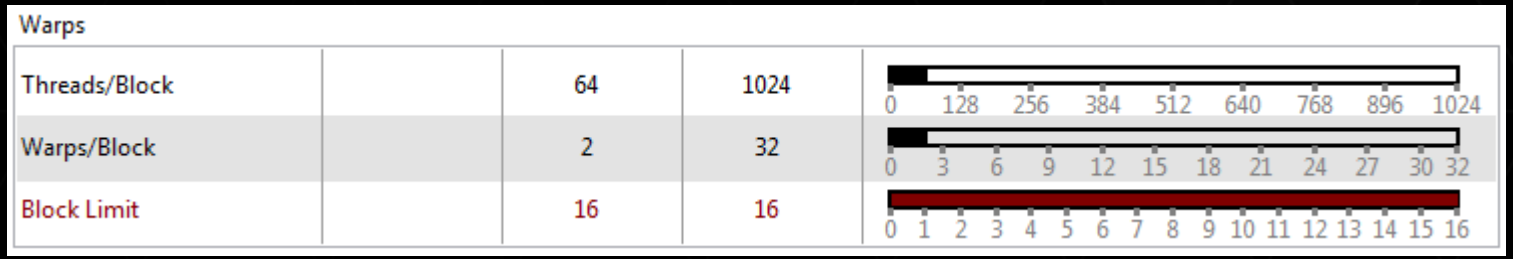
LOOKING FOR MORE INDICATORS

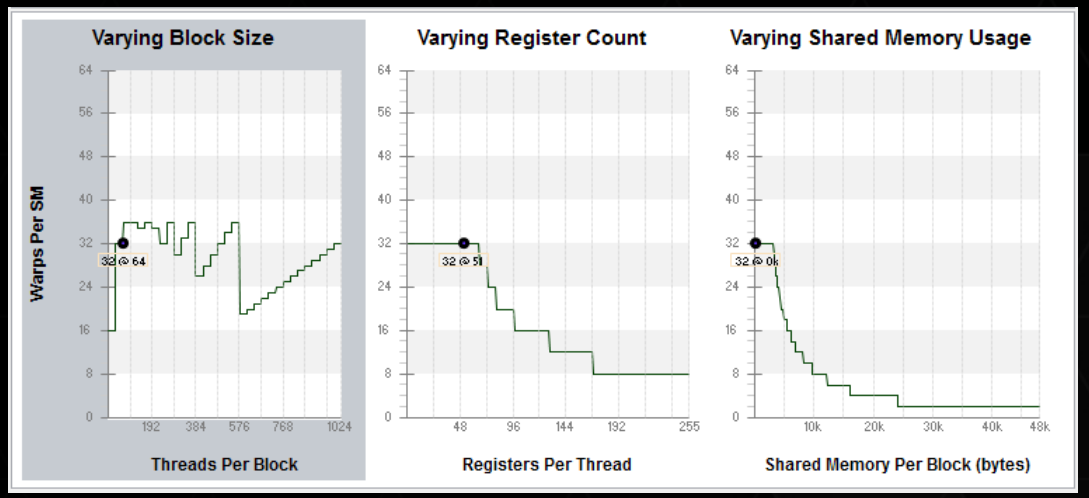


LOOKING FOR MORE INDICATORS

Not enough active warps to hide latencies?





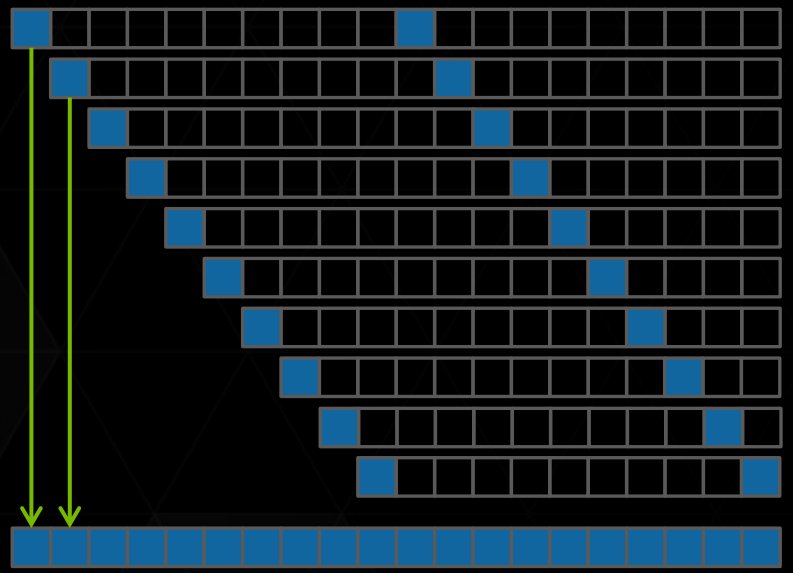


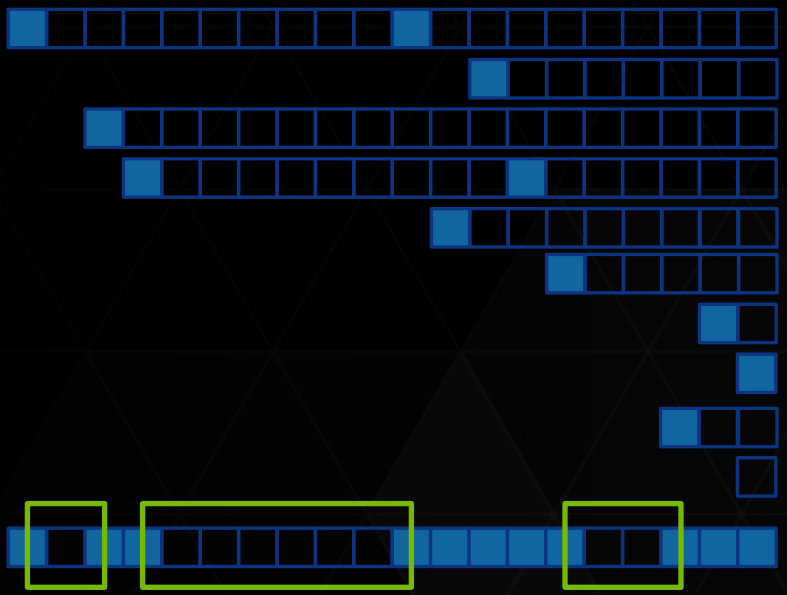
LATENCY

GPUs cover latencies by having a lot of work in flight

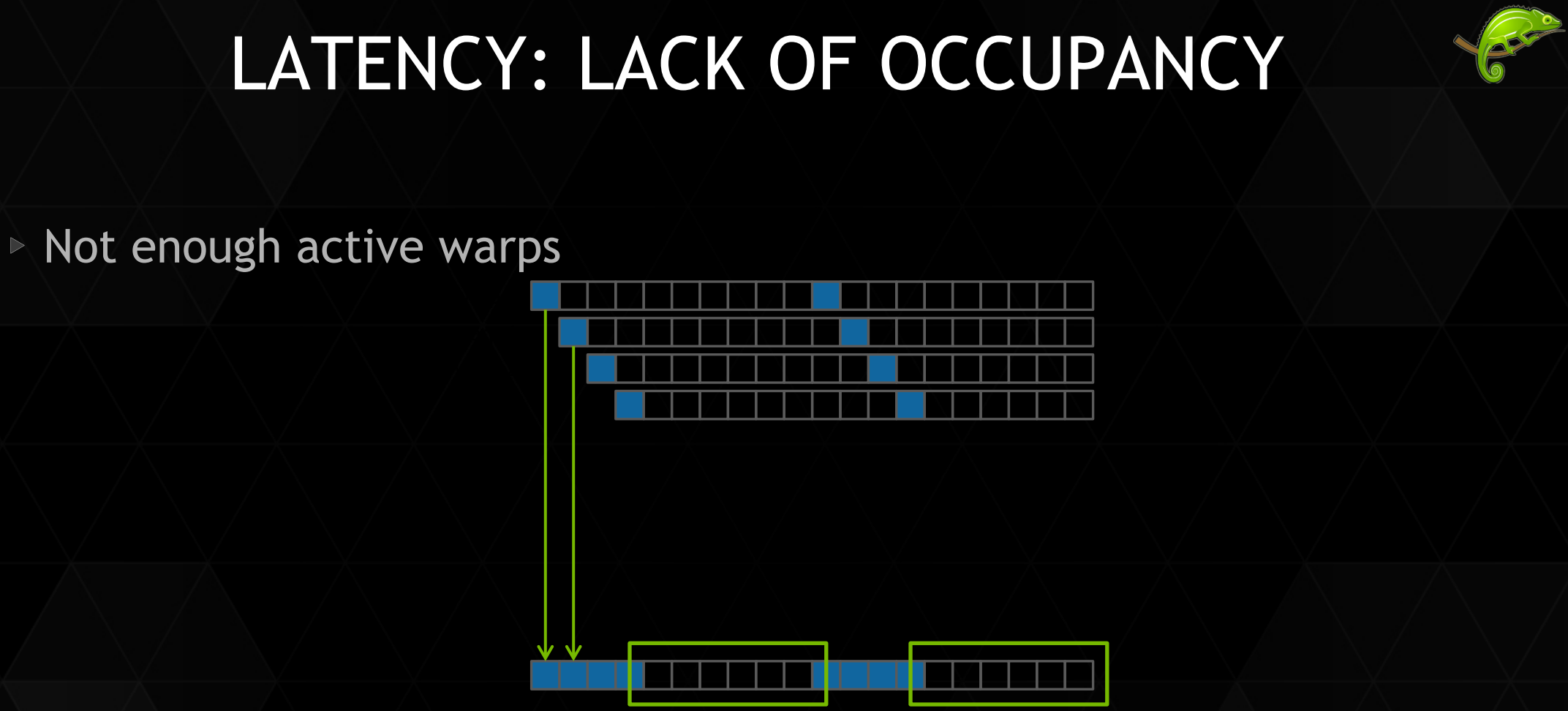
The warp issues

The warp waits (latency)





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

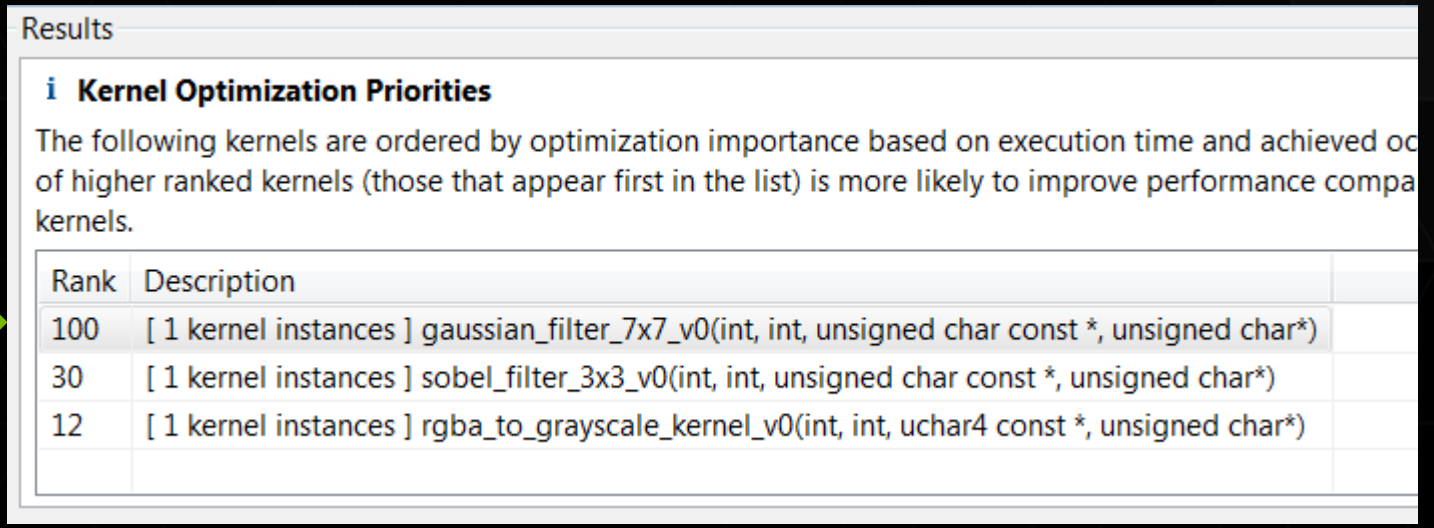
PERF-OPT QUICK REFERENCE CARD

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:	• Unroll loops (#pragma unroll) • Refactor threads to compute n output values at the same time (code duplication)

Iteration 3

IDENTIFY HOTSPOT

Hotspot



- gaussian_filter_7x7_v0() still the hotspot

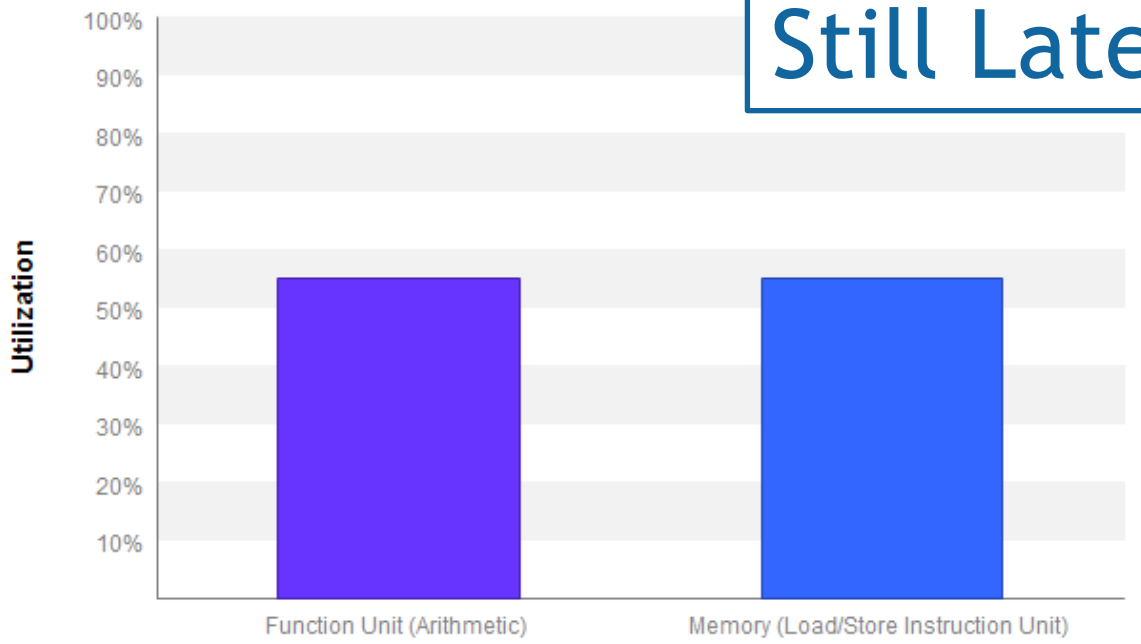
Kernel	Time	Speedup	Incremental Speedup
Original Version	5.233ms	1.00x	
Better Memory Accesses	1.589ms	3.29x	3.29x
Higher Occupancy	1.562ms	3.35x	1.02x

IDENTIFY PERFORMANCE LIMITER

Results

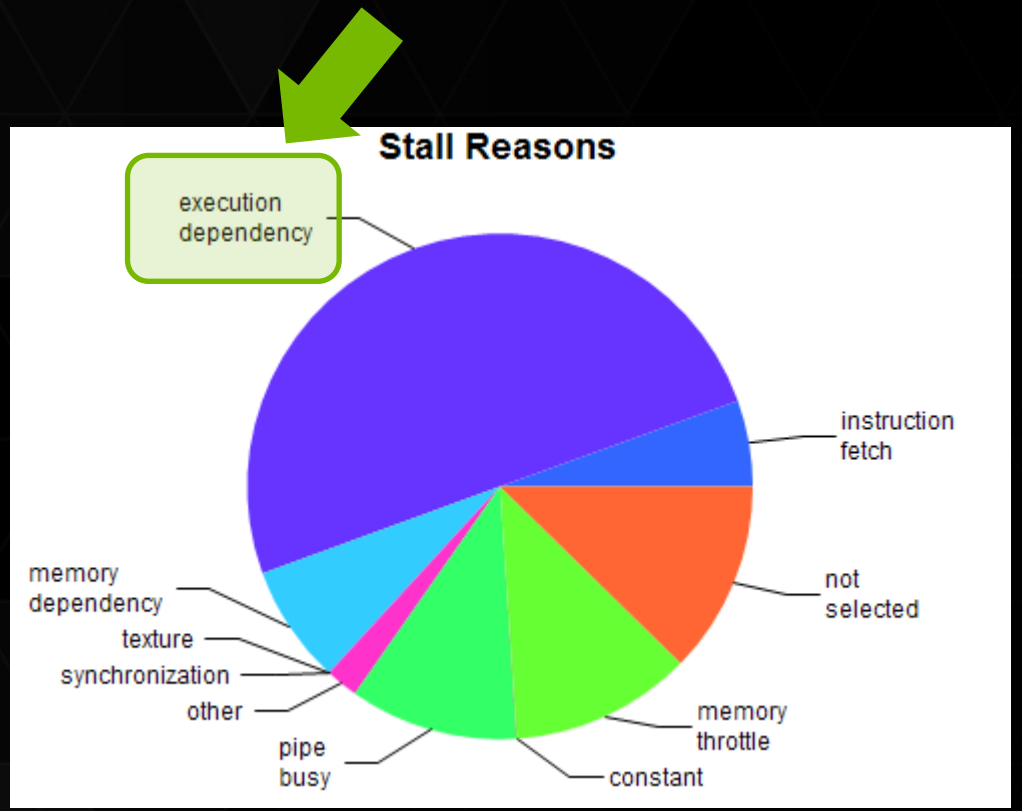
Kernel Performance Is Bound By Instruction And Memory Latency

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.



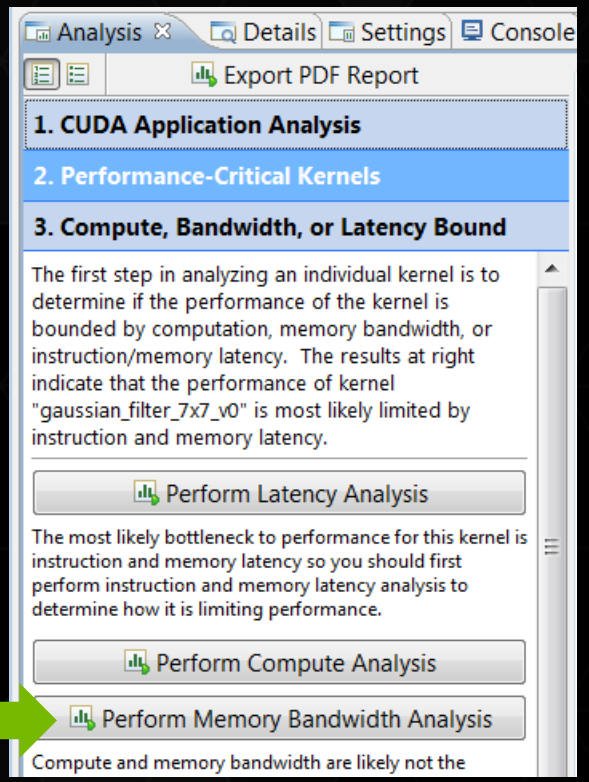
Still Latency Bound

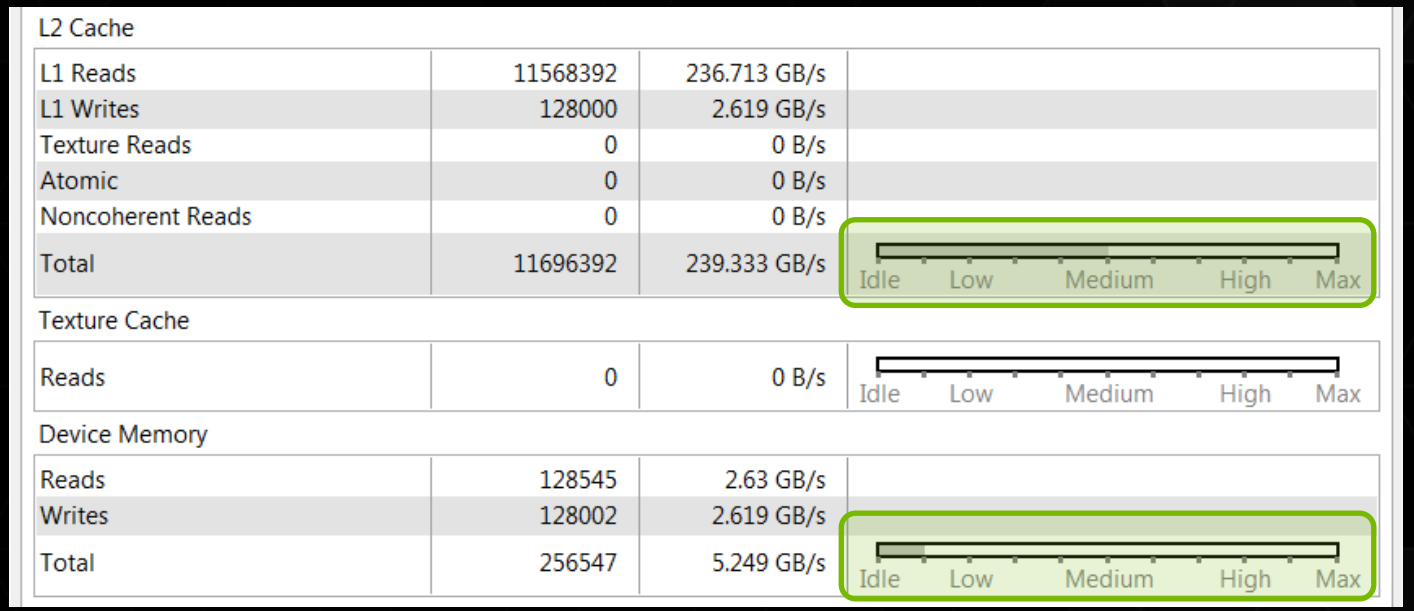
LOOKING FOR INDICATORS



Still high execution dependency, but occupancy OK

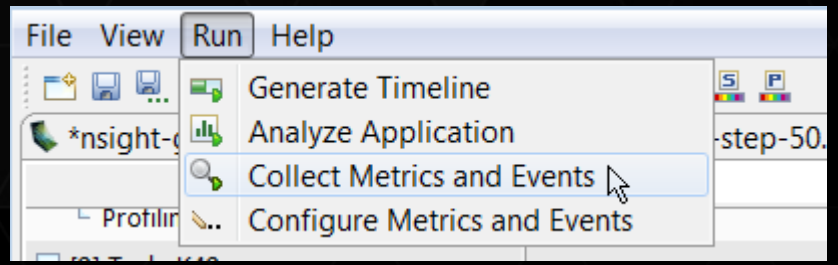
LOOKING FOR MORE INDICATORS

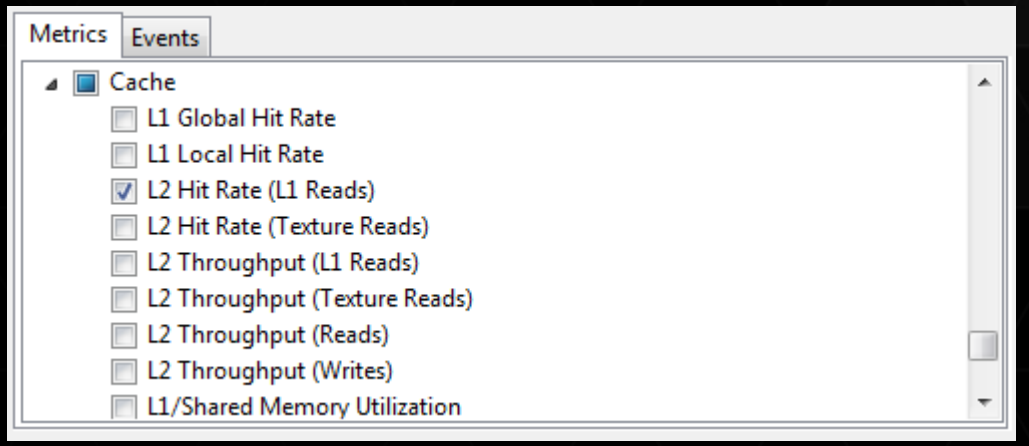




Is our working set mostly in L2?

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?

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:	**Reduce** latency, move data to faster 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: • Shared Memory • (or Registers) • (or Constant Memory) • (or Texture Cache)

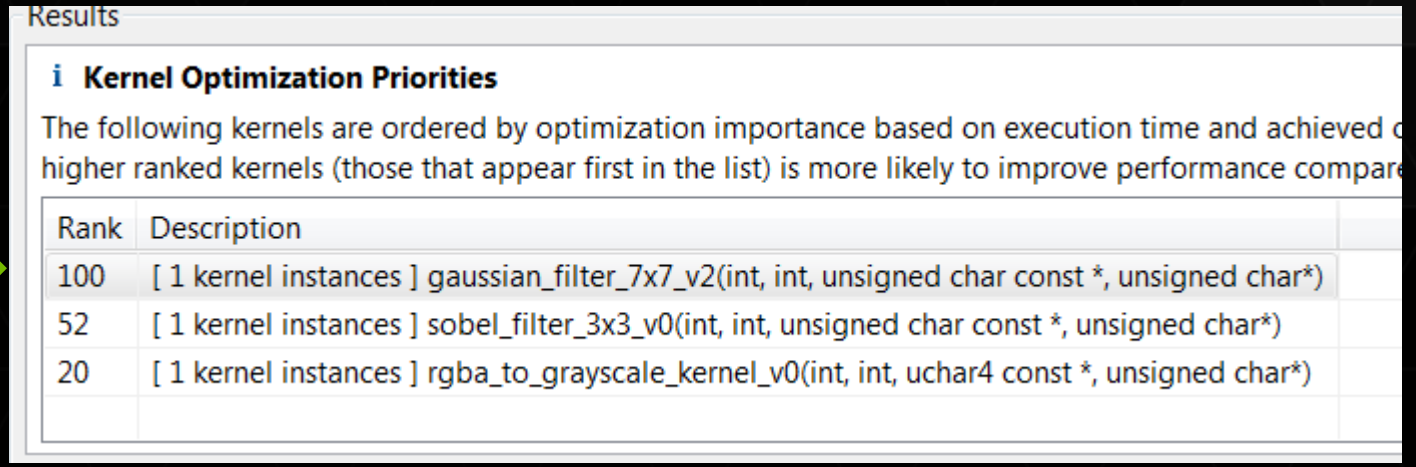
PERF-OPT QUICK REFERENCE CARD

Category:	Memory Bound – Shared Memory
Problem:	Too much data movement
Goal:	Reduce 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: • Shared Memory • (or Registers) • (or Constant Memory) • (or Texture Cache)

Iteration 4

IDENTIFY HOTSPOT

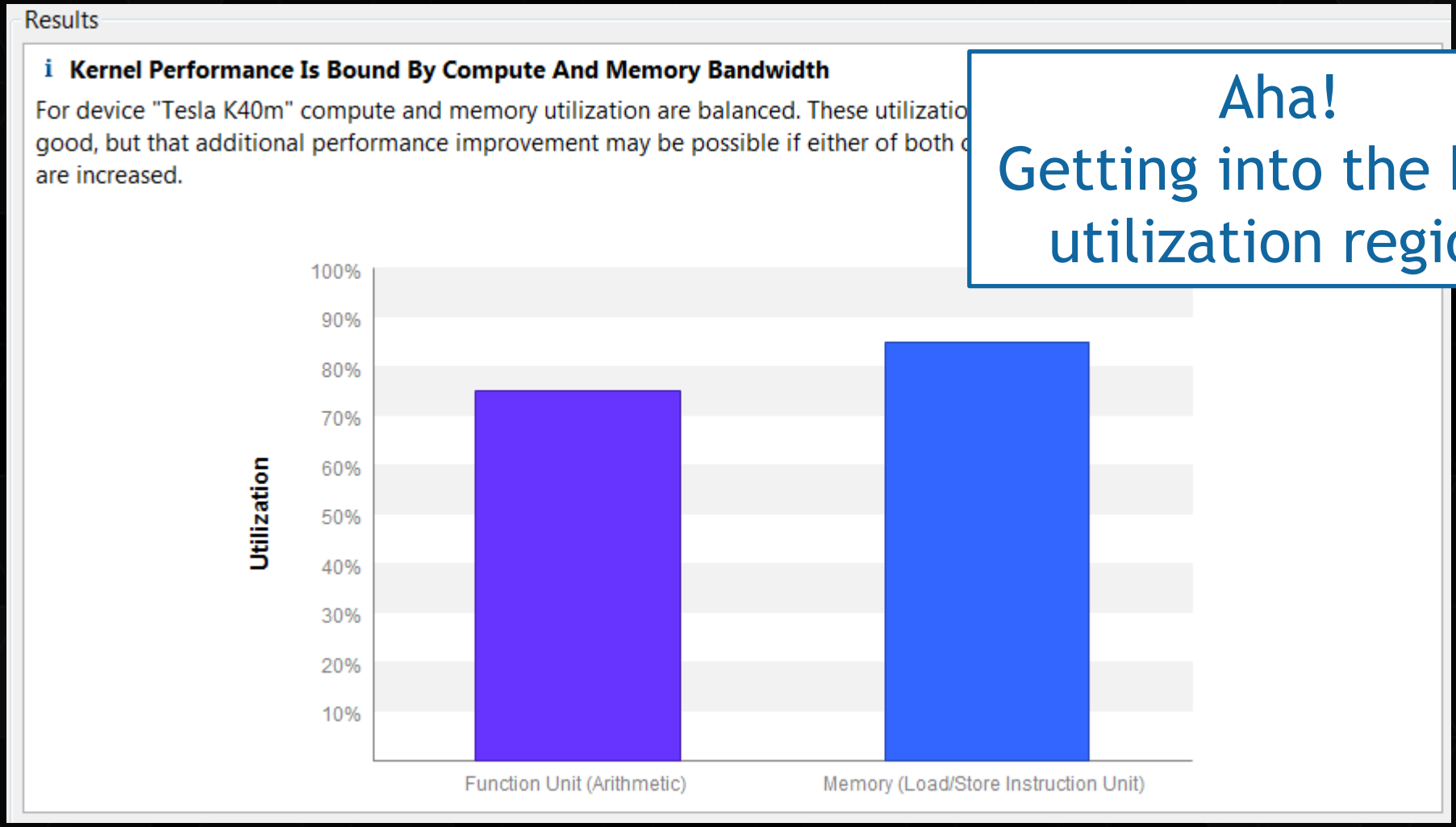
Hotspot



- 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

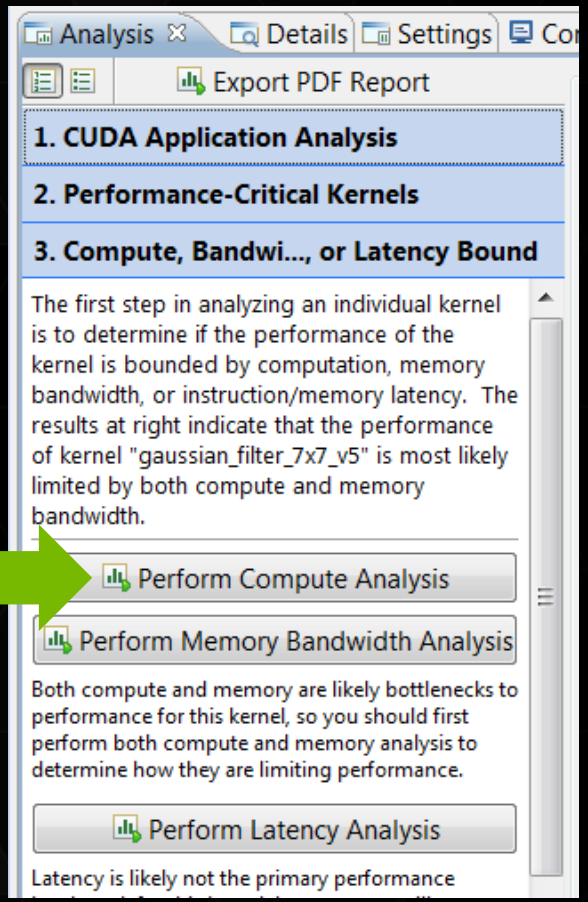
IDENTIFY PERFORMANCE LIMITER

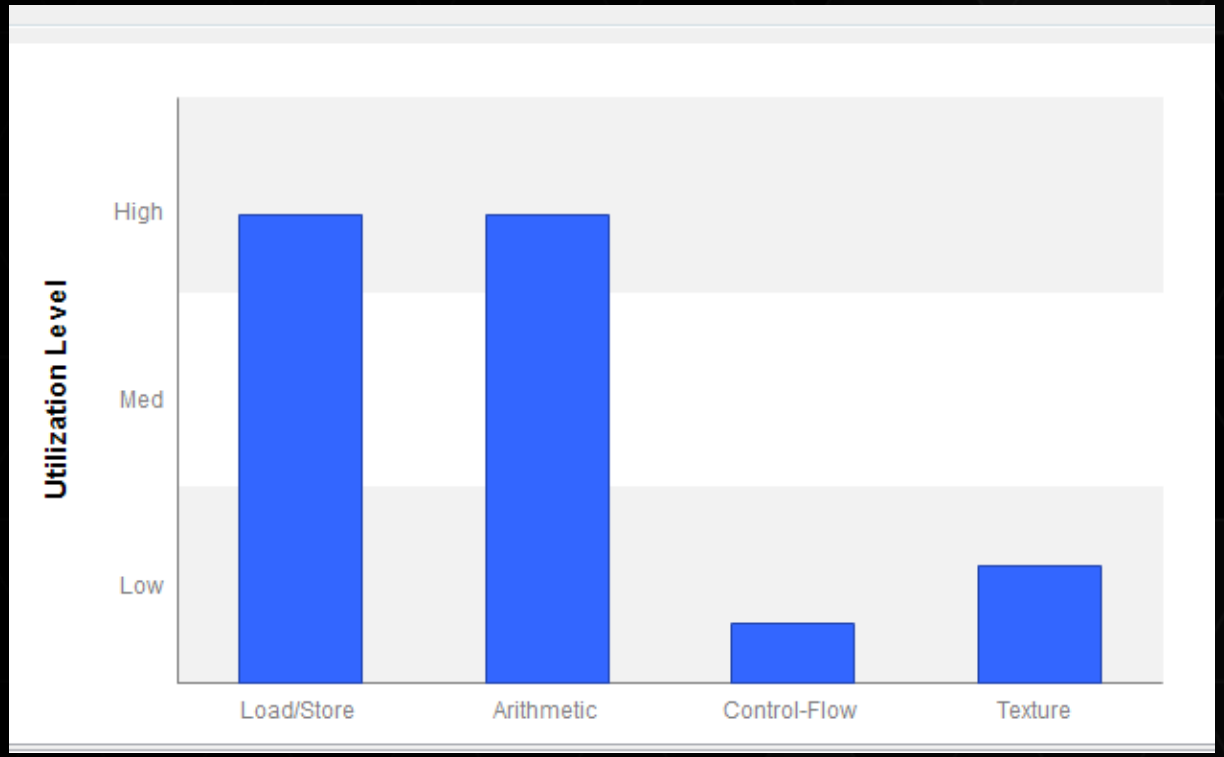


Aha!
Getting into the high utilization region

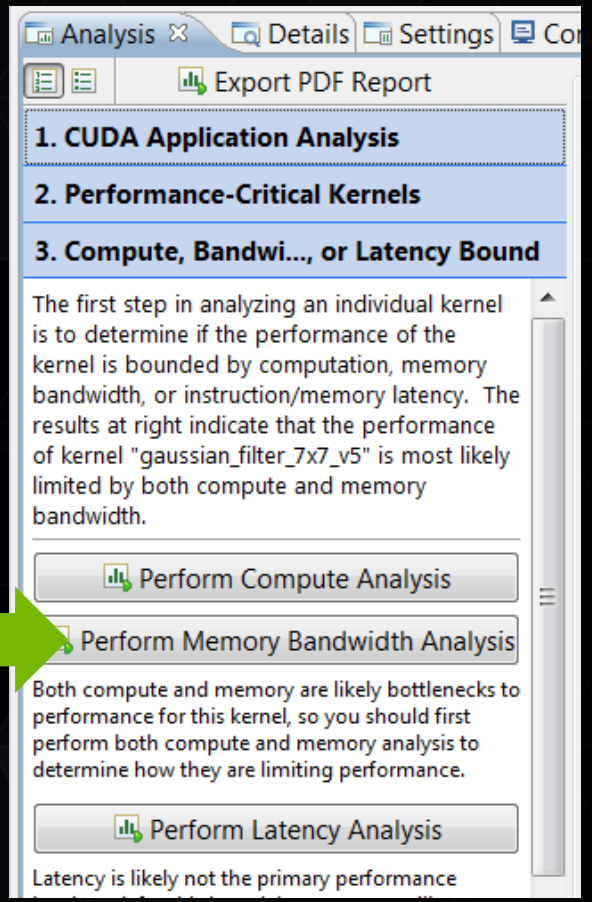
LOOKING FOR INDICATORS

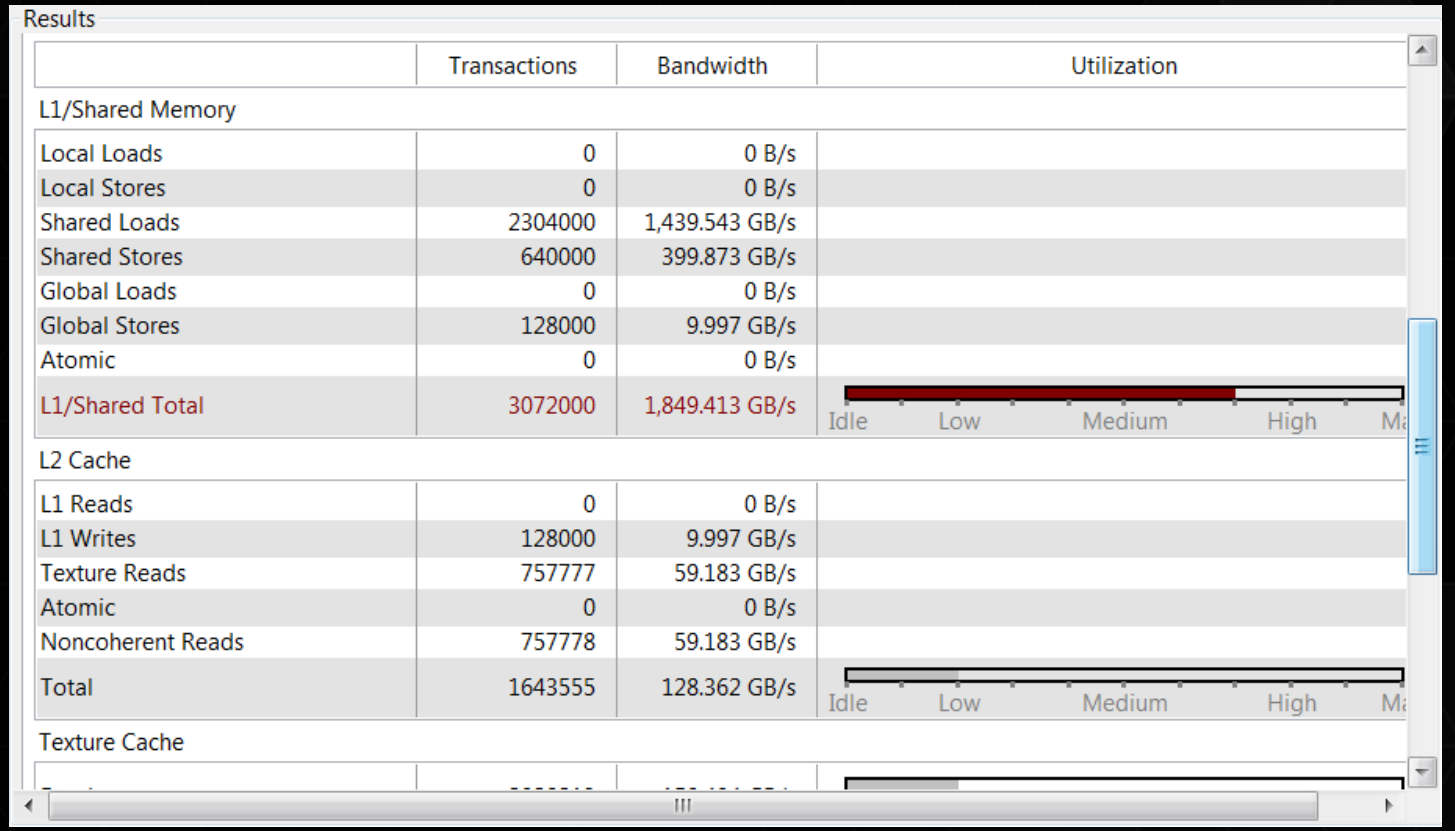
Launch





LOOKING FOR MORE INDICATORS

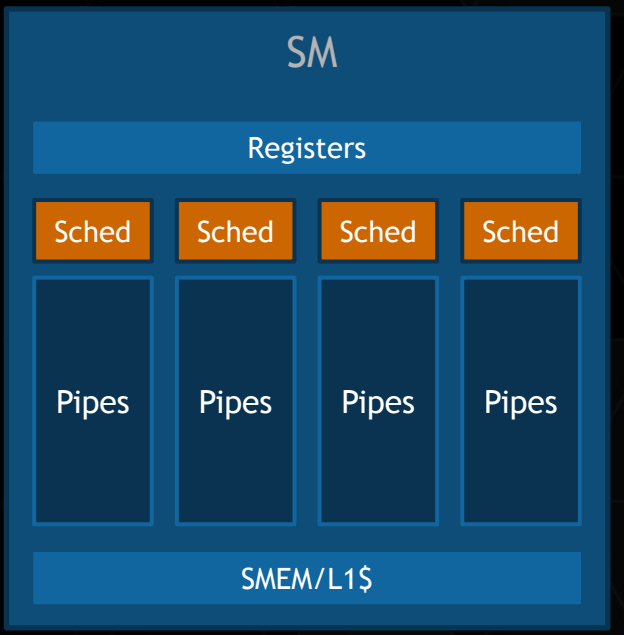




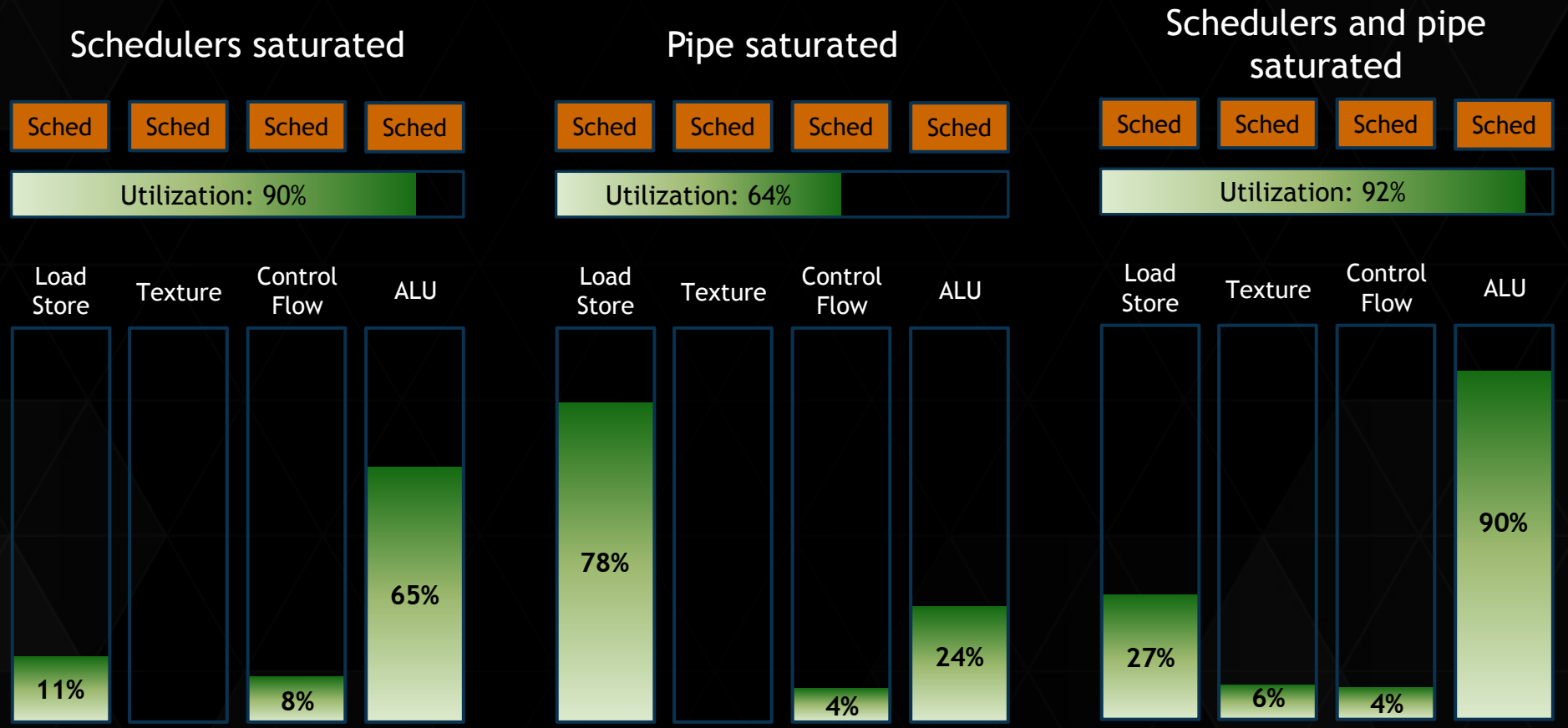
Load/Store Unit is really busy!
Can we reduce the load?

INSTRUCTION THROUGHPUT

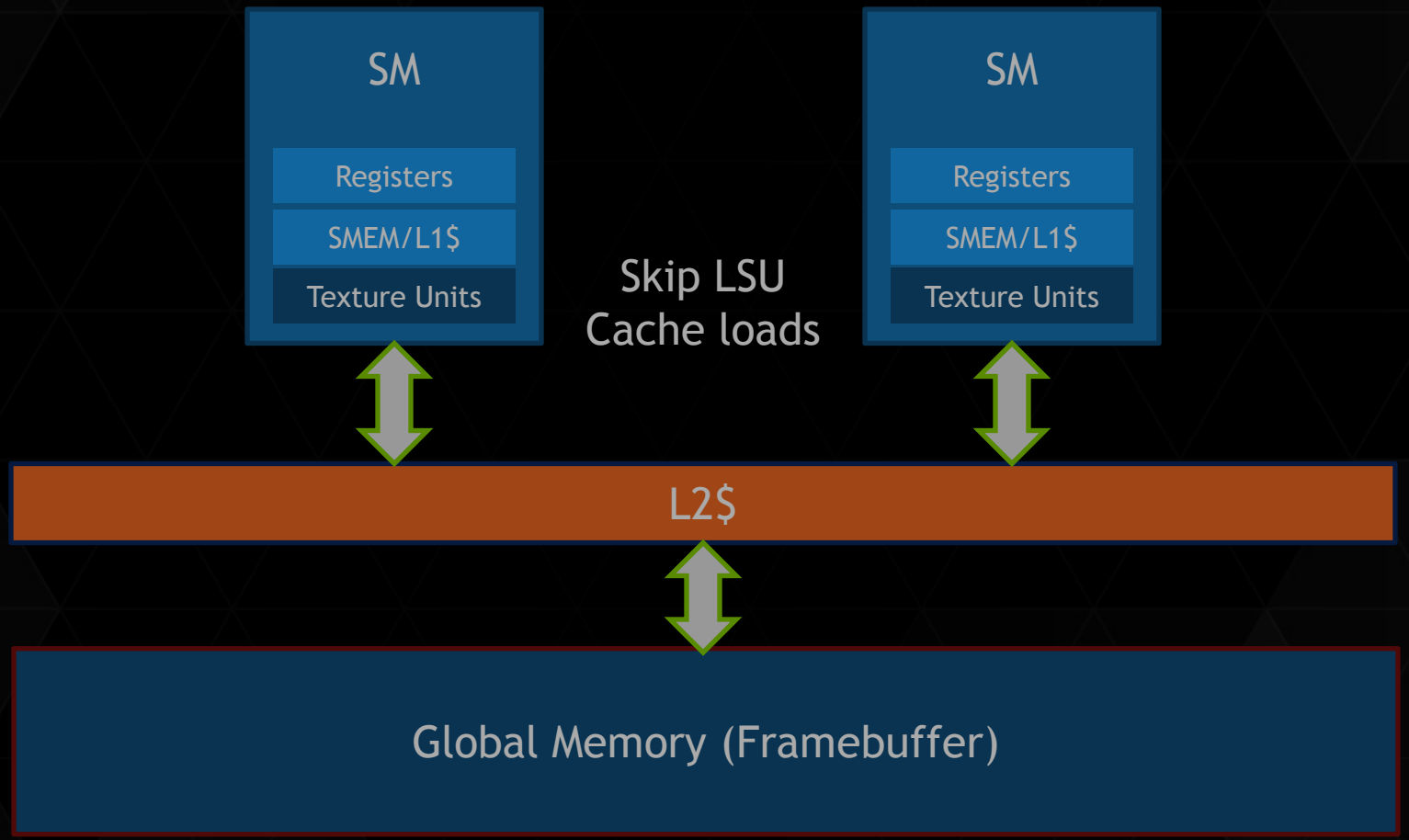
- Each SM has 4 schedulers (Kepler)
- Schedulers issue instructions to pipes
- A scheduler issues up to 2 instructions/cycle
  - Sustainable peak is 7 instructions/cycle per SM (not 4x2 = 8)
- A scheduler issues inst. from a single warp
- Cannot issue to a pipe if its issue slot is full



INSTRUCTION THROUGHPUT



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

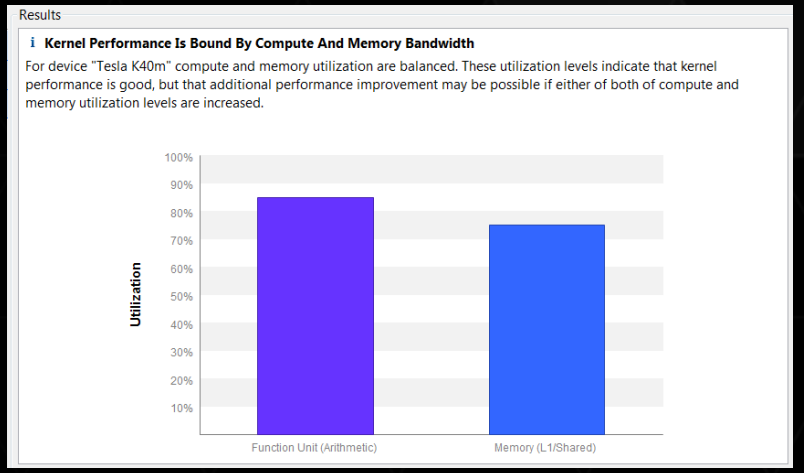
Kernel	Time	Speedup	Incremental Speedup
Original version	5.233ms	1.00x	
Better memory accesses	1.589ms	3.29x	3.29x
Higher Occupancy	1.562ms	3.35x	1.02x
Shared memory	0.911ms	5.74x	1.7x
Read-Only path	0.808ms	6.48x	1.13x

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:	Load read-only data through Texture Units: • Annotate read-only pointers with const __restrict__ • Use __ldg() 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

Kernel Optimization Priorities

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

Thank YOU