GPU TECHNOLOGY CONFERENCE

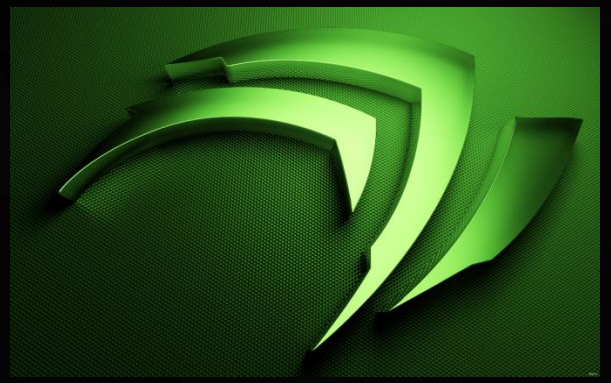
CUDA OPTIMIZATION WITH NVIDIA NSIGHT™ ECLIPSE EDITION

JAKOB PROGSCH, NVIDIA (jprogsch@nvidia.com)
CHRISTOPH ANGERER, NVIDIA
JULIEN DEMOUTH, NVIDIA

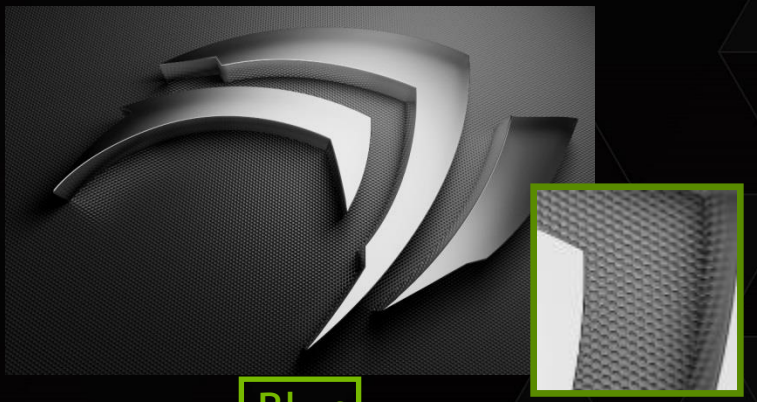
WHAT YOU WILL LEARN

- 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

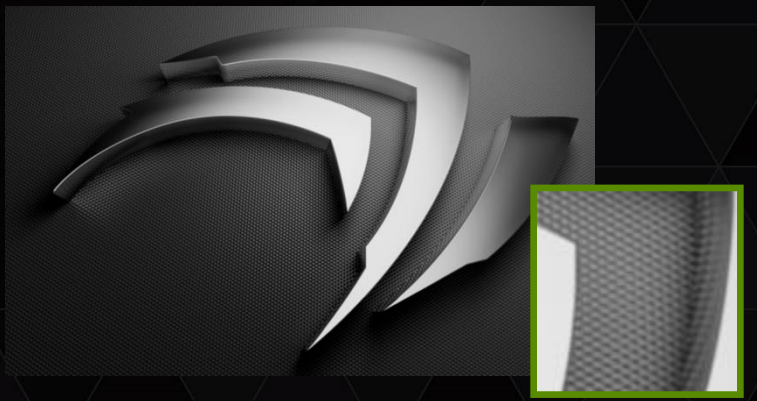
INTRODUCING THE APPLICATION



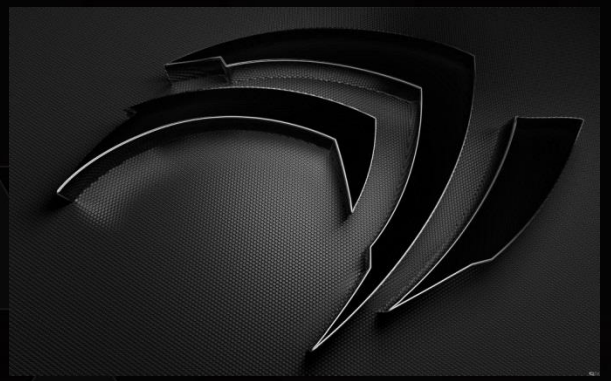
Grayscale



Blur



Edges



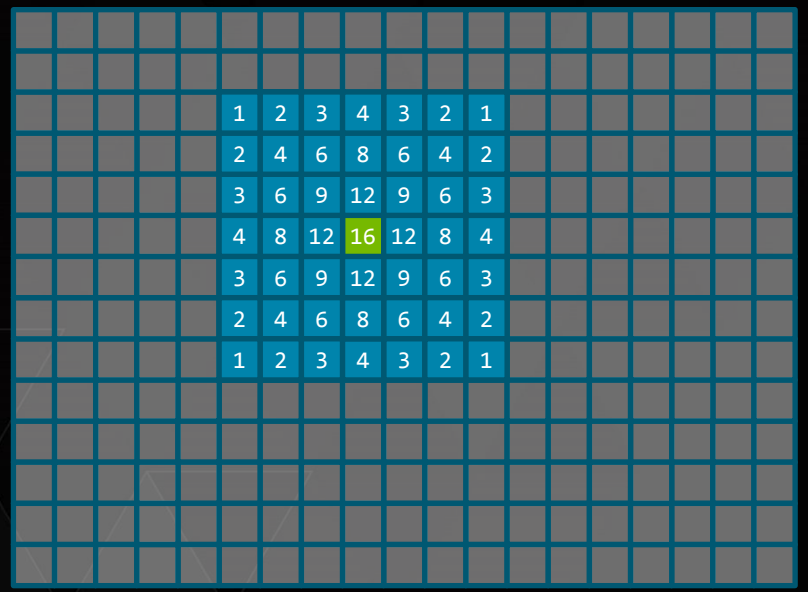
INTRODUCING THE APPLICATION

- 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
```

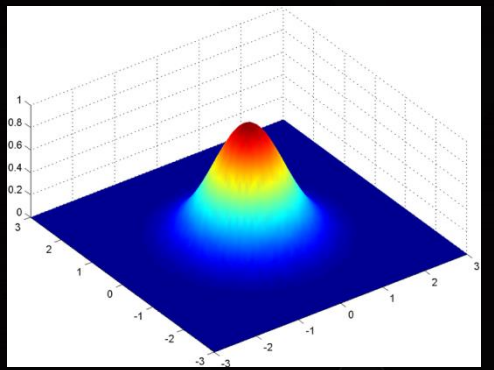
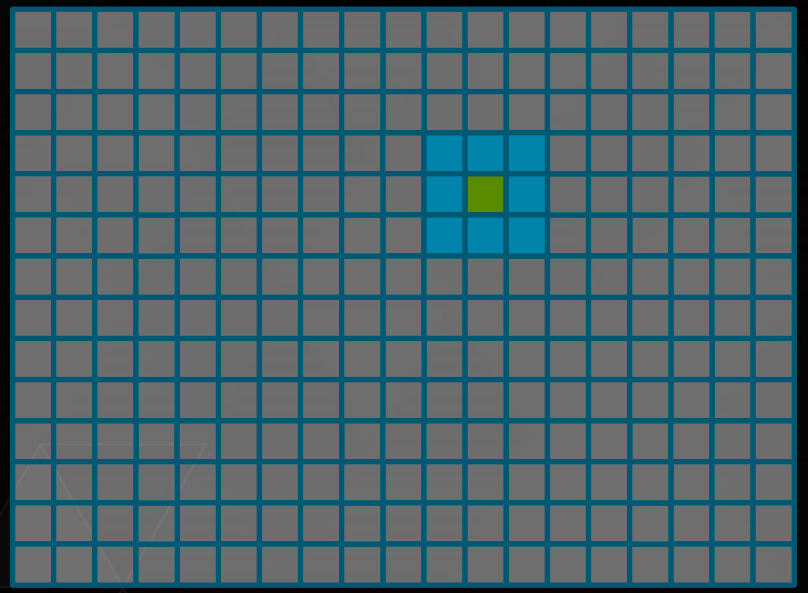


Image from Wikipedia

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

PERFORMANCE OPTIMIZATION CYCLE

1. Profile Application

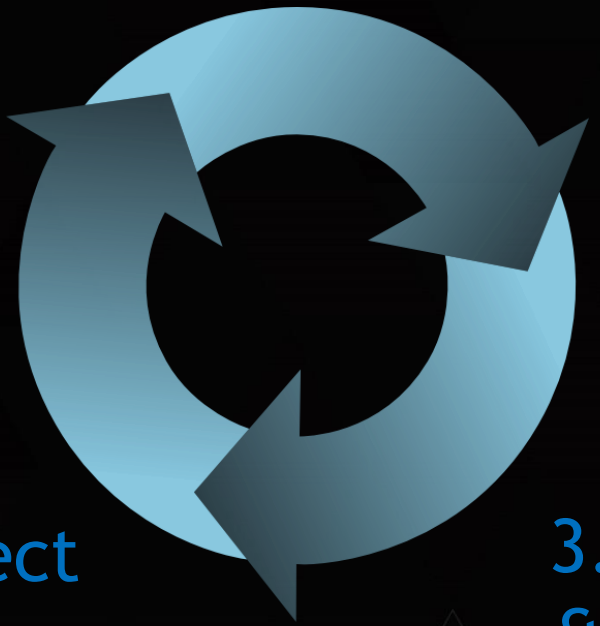
2. Identify Performance Limiter

3. Analyze Profile & Find Indicators

4. Reflect

4b. Build Knowledge

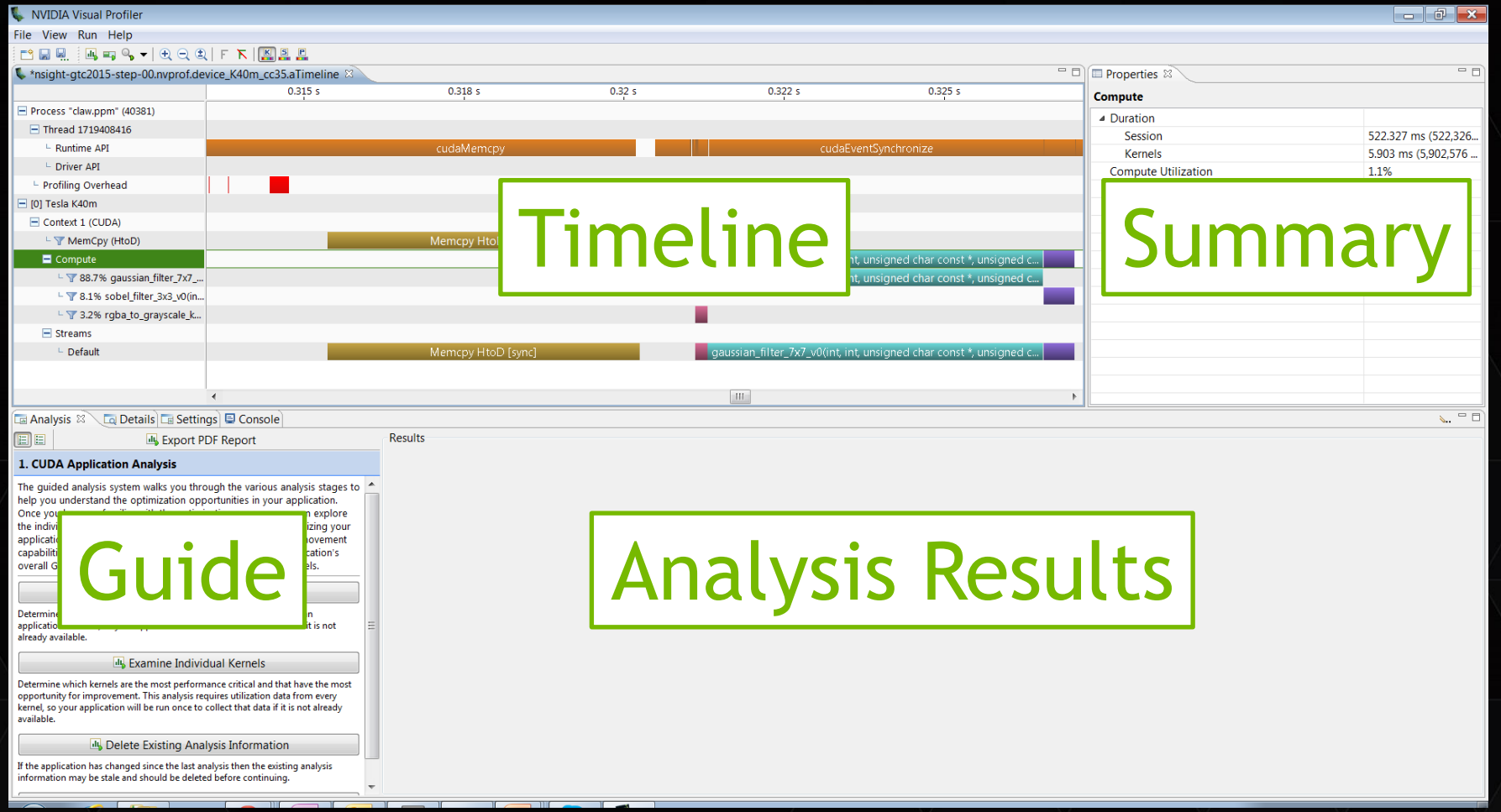
5. Change and Test Code



Chameleon from http://www.vectorportal.com, Creative Commons

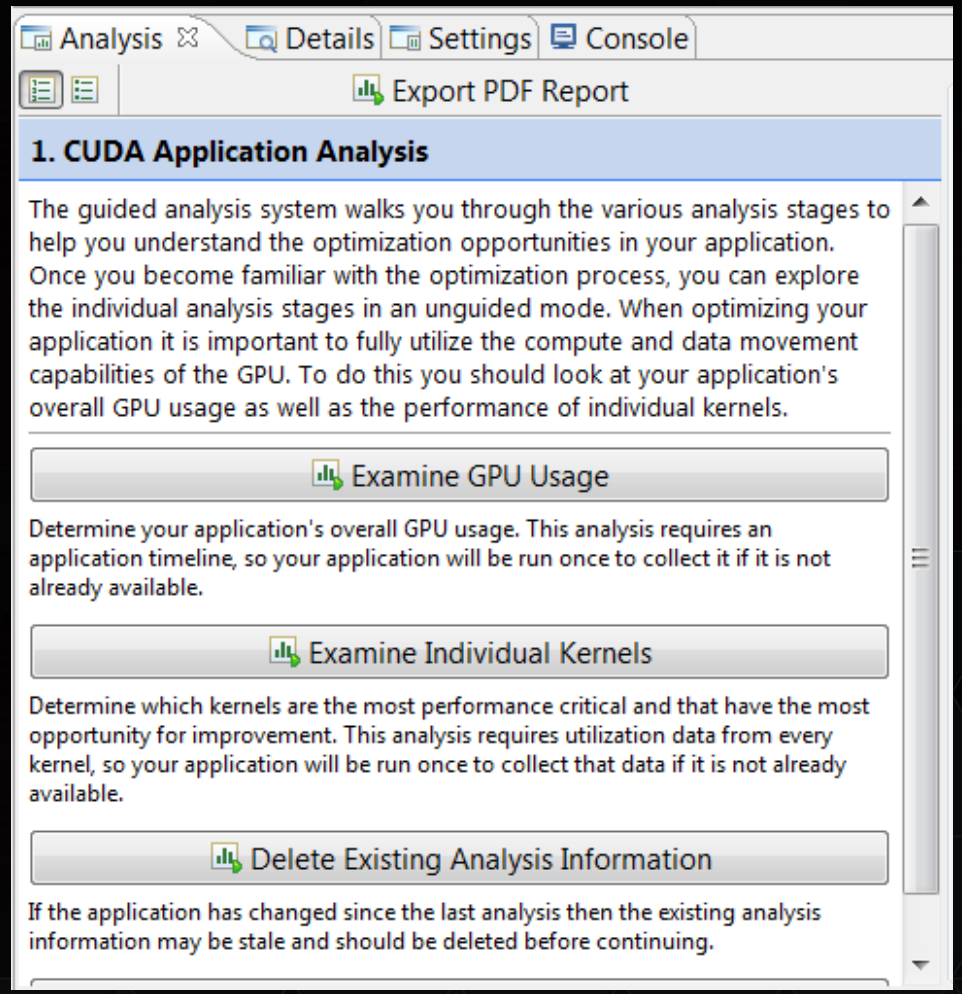
ITERATION 1

THE PROFILER WINDOW



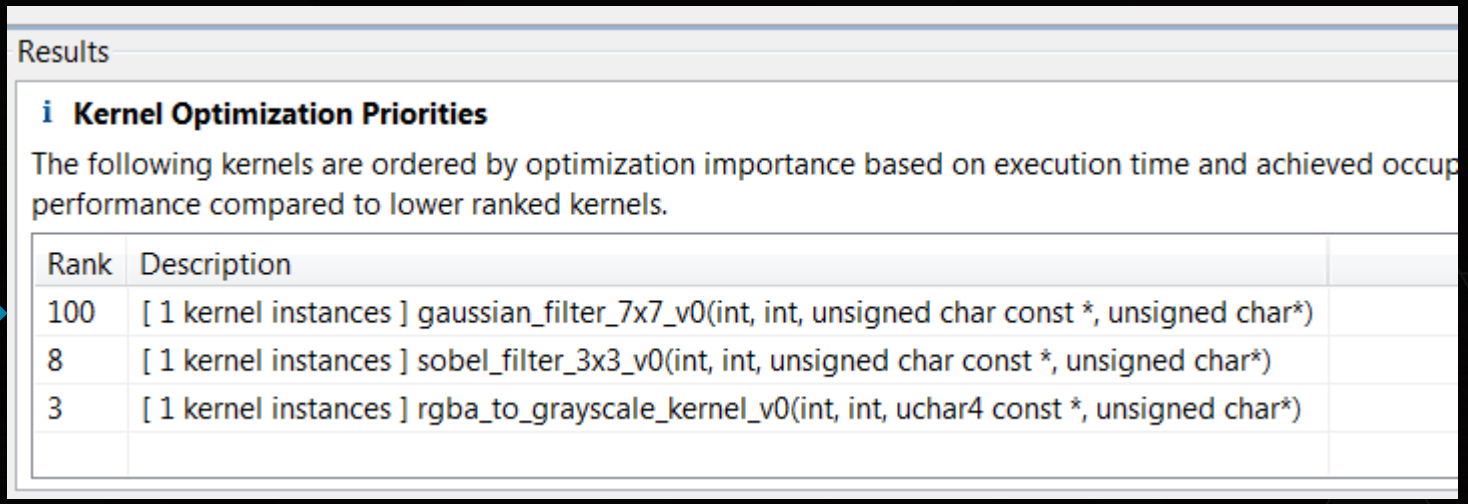
EXAMINE INDIVIDUAL KERNELS
(GUIDED ANALYSIS)

Launch



IDENTIFY HOTSPOT

Hotspot



Results

Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved occup
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()

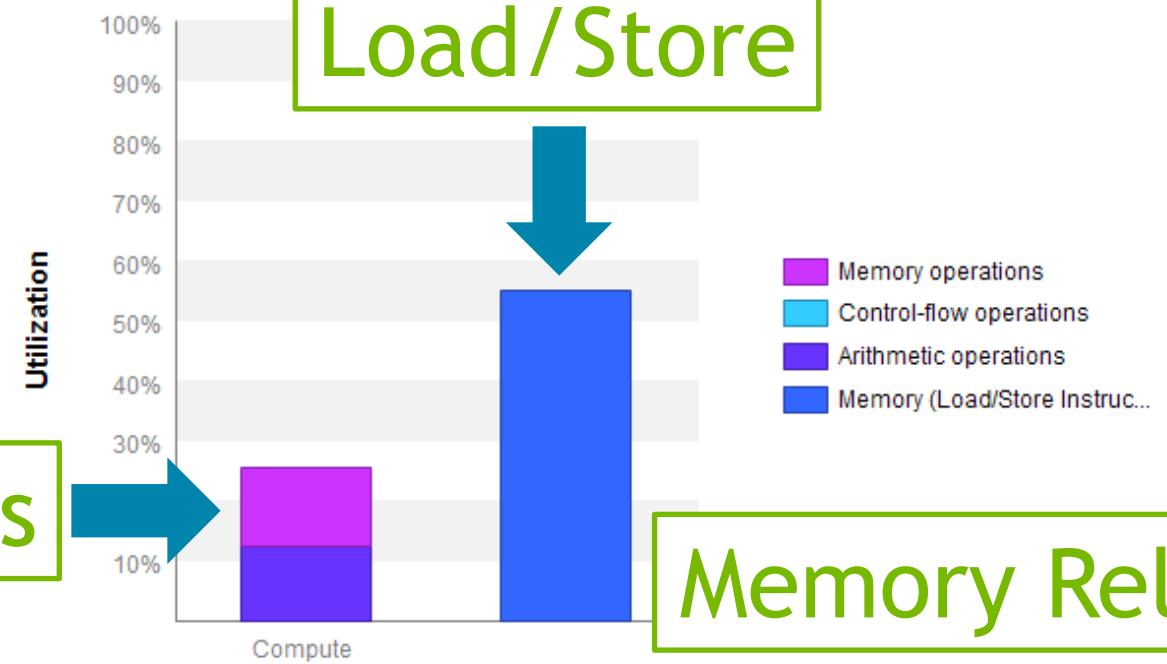
Kernel	Time	Speedup
Original Version	5.233ms	1.00x

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.



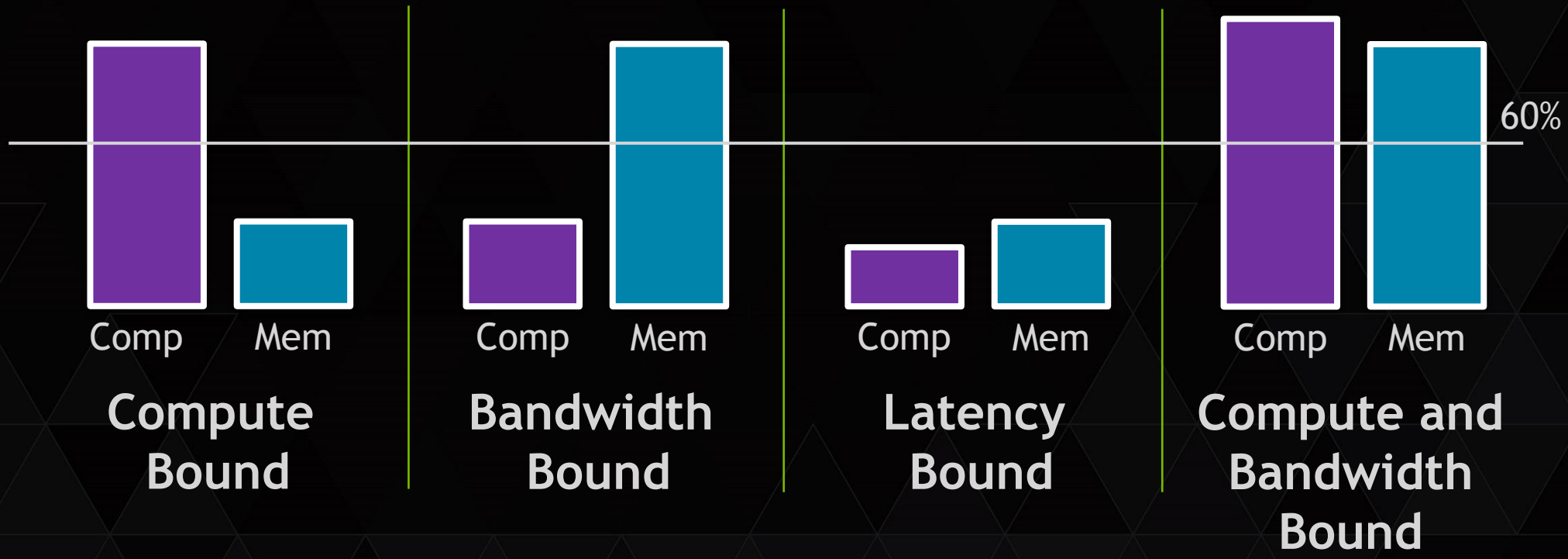
Load/Store

Memory Ops

Memory Related Issues?

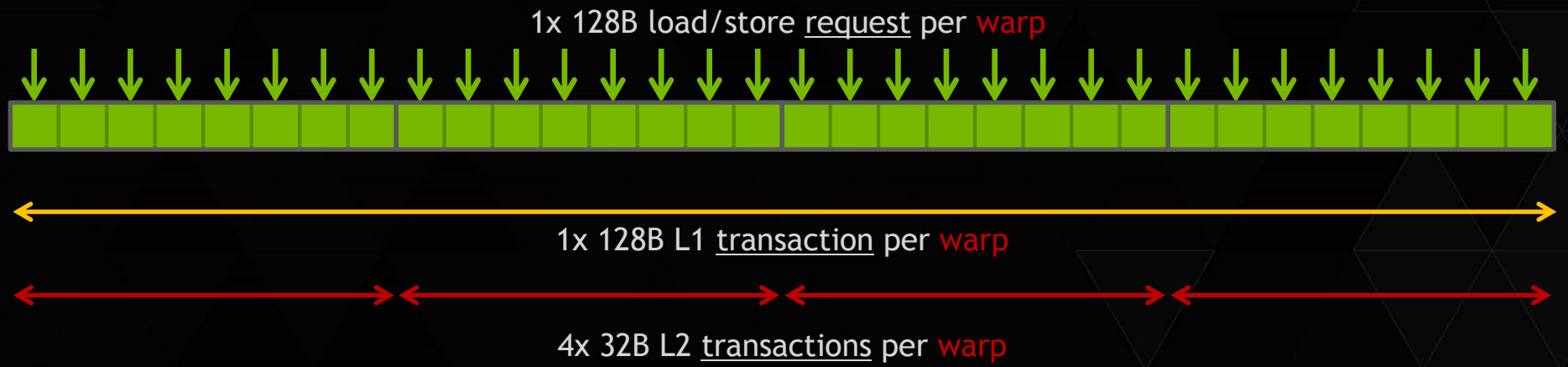
PERFORMANCE LIMITER CATEGORIES

- Memory Utilization vs Compute Utilization
- Four possible combinations:



MEMORY TRANSACTIONS: BEST CASE

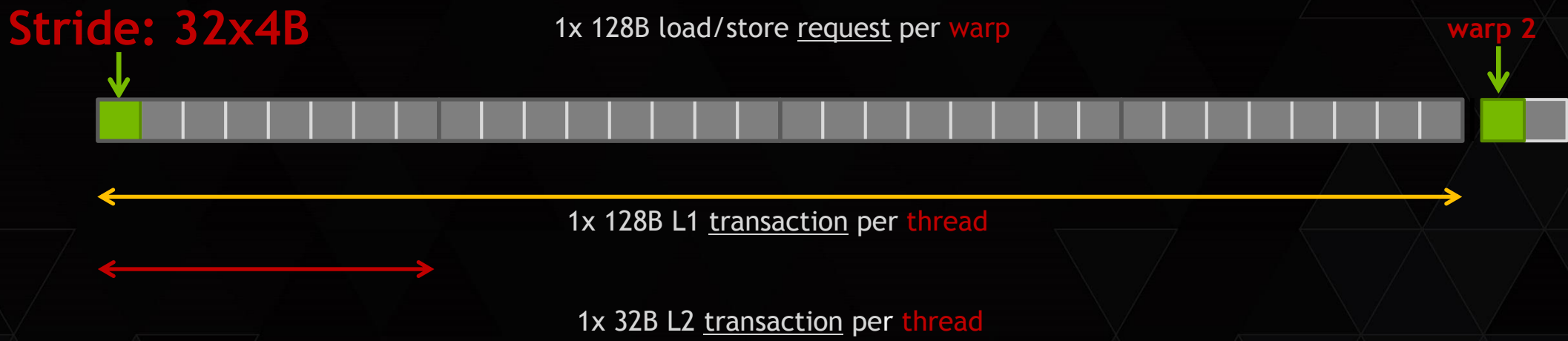
- A warp issues 32x4B aligned and consecutive load/store request
- Threads read different elements of the same 128B segment



- 1x L1 transaction: 128B needed / 128B transferred
- 4x L2 transactions: 128B needed / 128B transferred

MEMORY TRANSACTIONS: WORST CASE

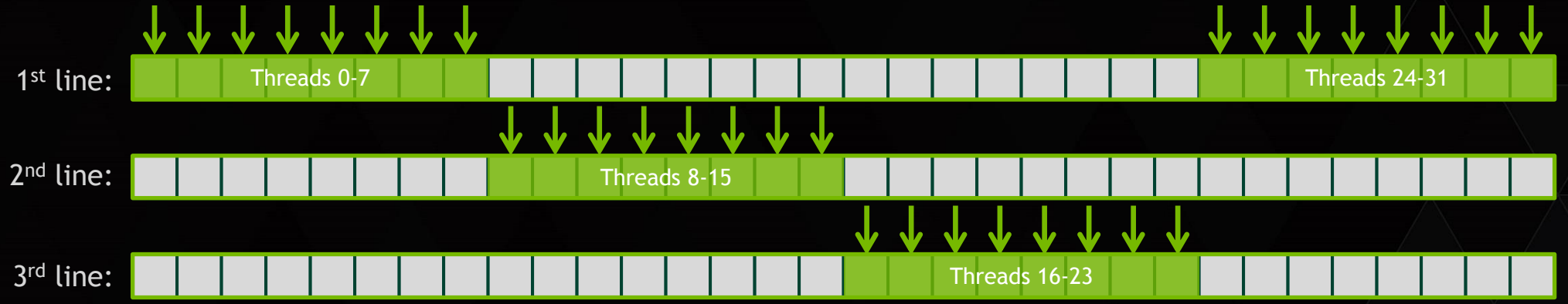
- Threads in a warp read/write 4B words, 128B between words
- Each thread reads the first 4B of a 128B segment



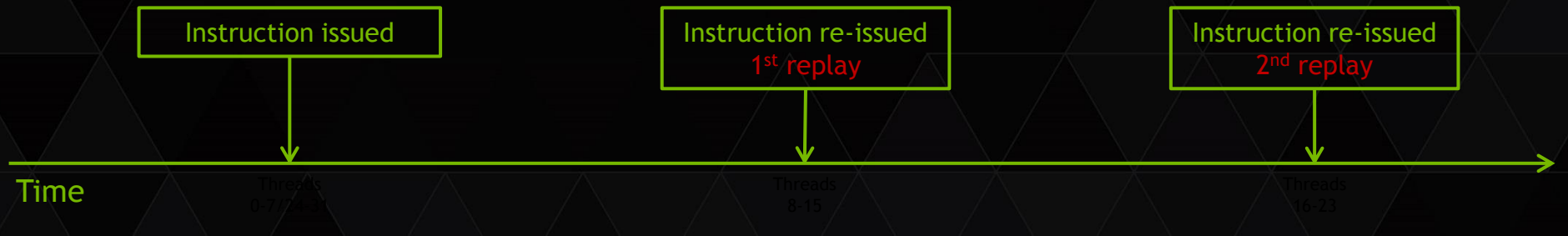
- 32x L1 transactions: 128B needed / 32x 128B transferred
- 32x L2 transactions: 128B needed / 32x 32B transferred

TRANSACTIONS AND REPLAYS

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

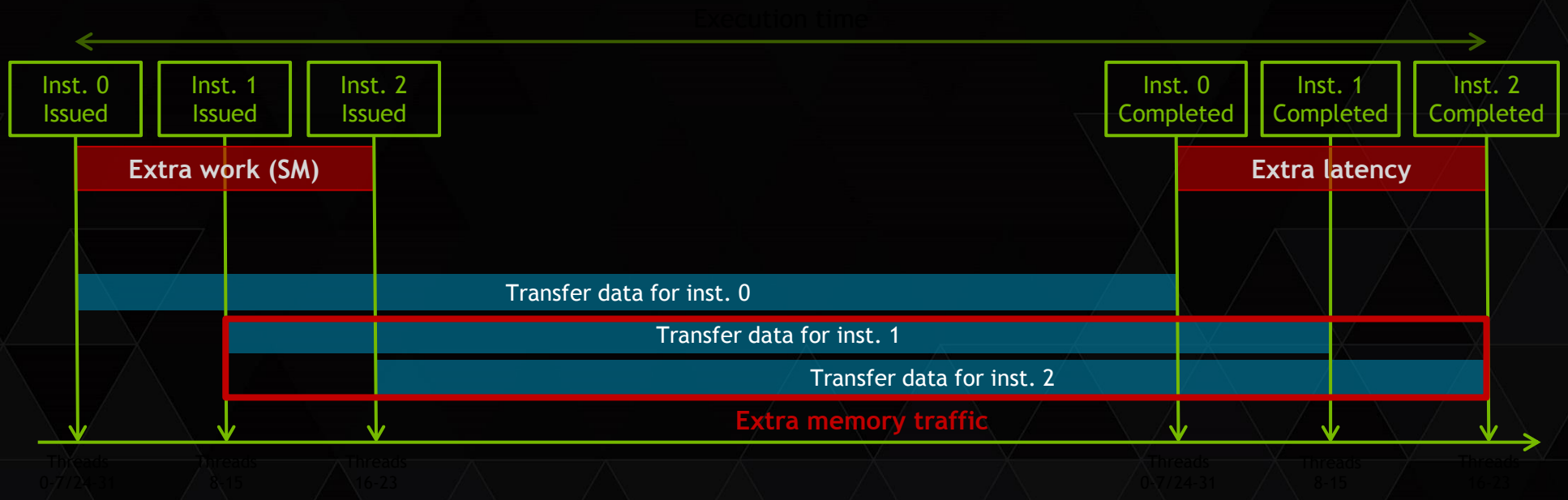


- 1 instr. executed and 2 replays = 1 request and 3 transactions



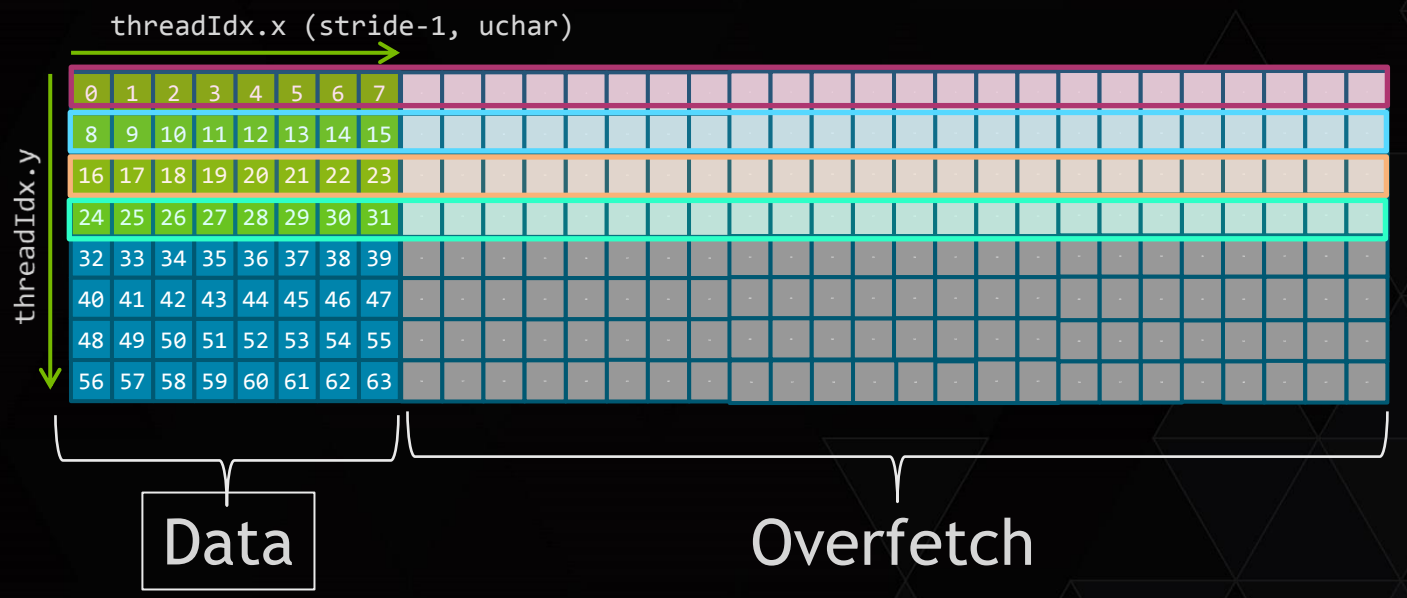
TRANSACTIONS AND REPLAYS

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

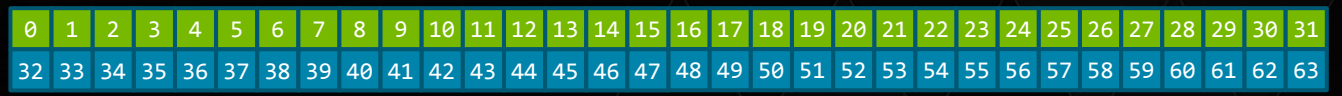


CHANGING THE BLOCK LAYOUT

- Our blocks are 8x8



- We should use blocks of size 32x2



IMPROVED MEMORY ACCESS

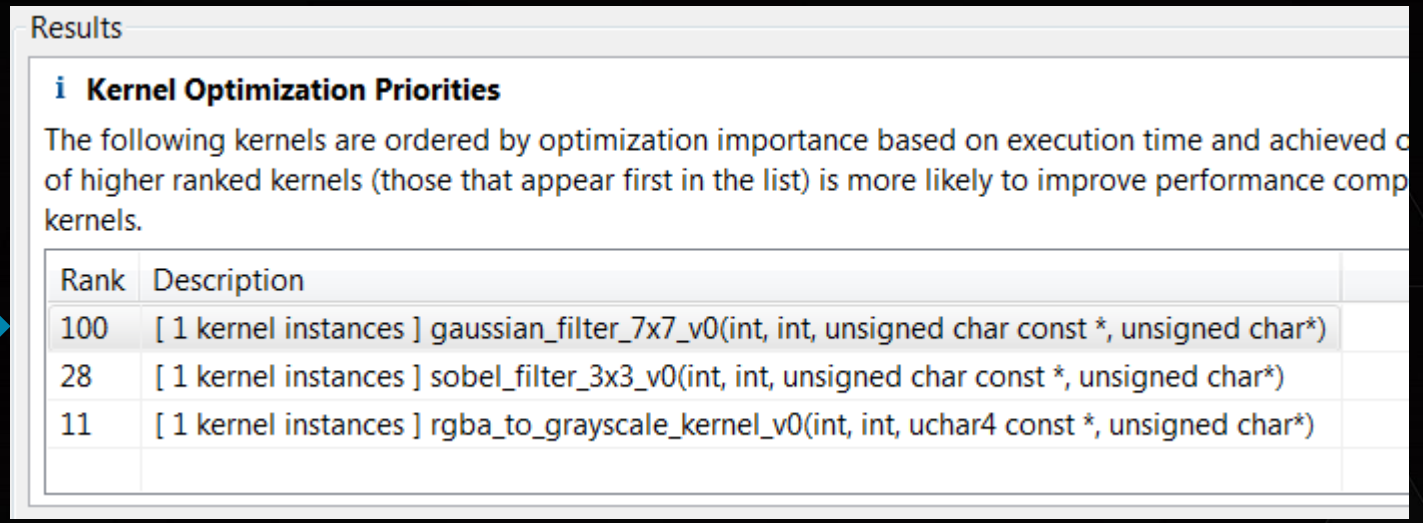
- Blocks of size 32x2
- Memory is used more efficiently

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

ITERATION 2

IDENTIFY HOTSPOT

Hotspot



- gaussian_filter_7x7_v0() still the hotspot

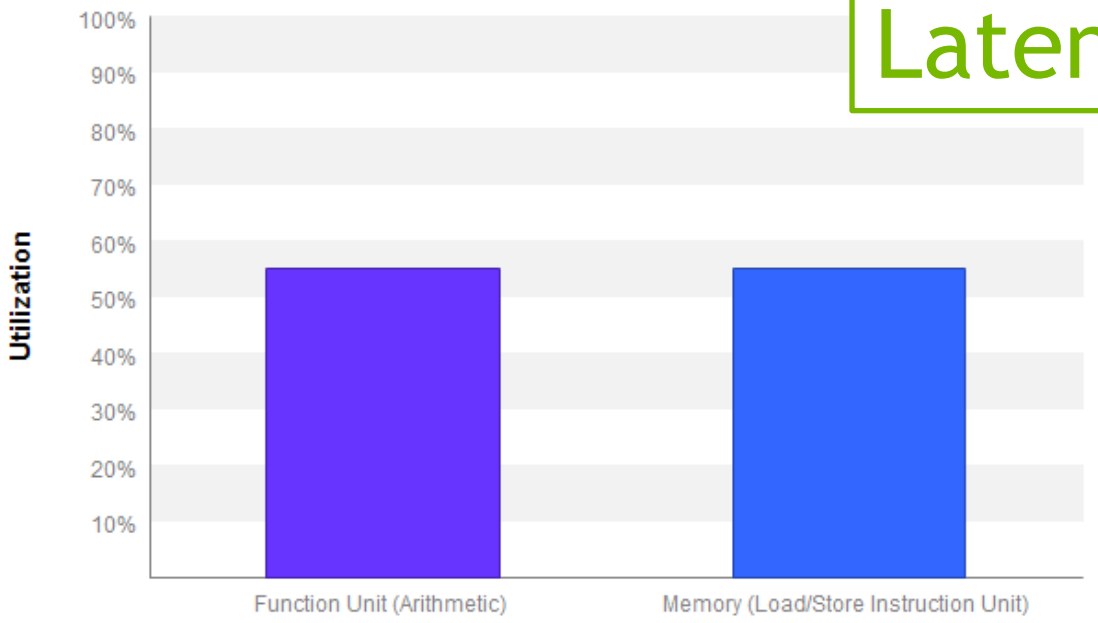
Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x

IDENTIFY PERFORMANCE LIMITER

Results

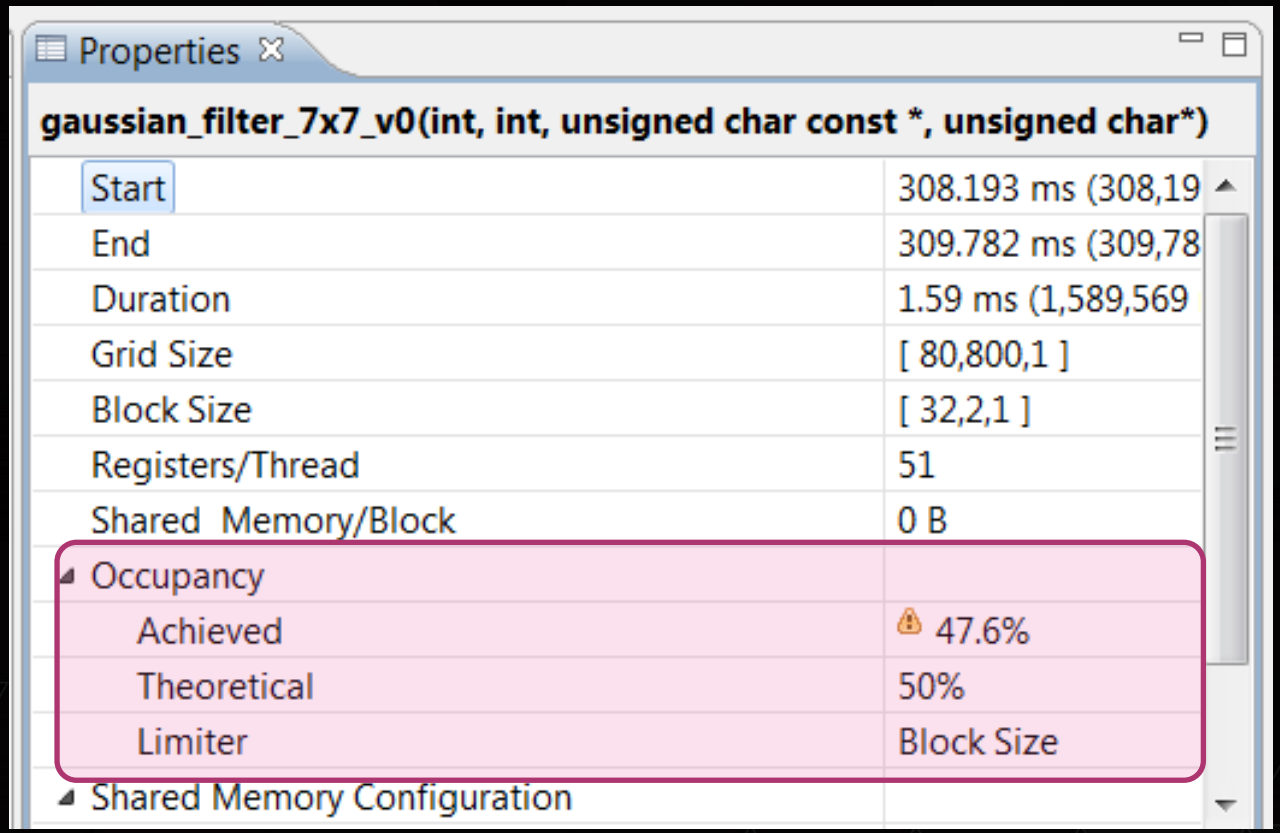
Kernel Performance Is Bound By Instruction And Memory Latency

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.



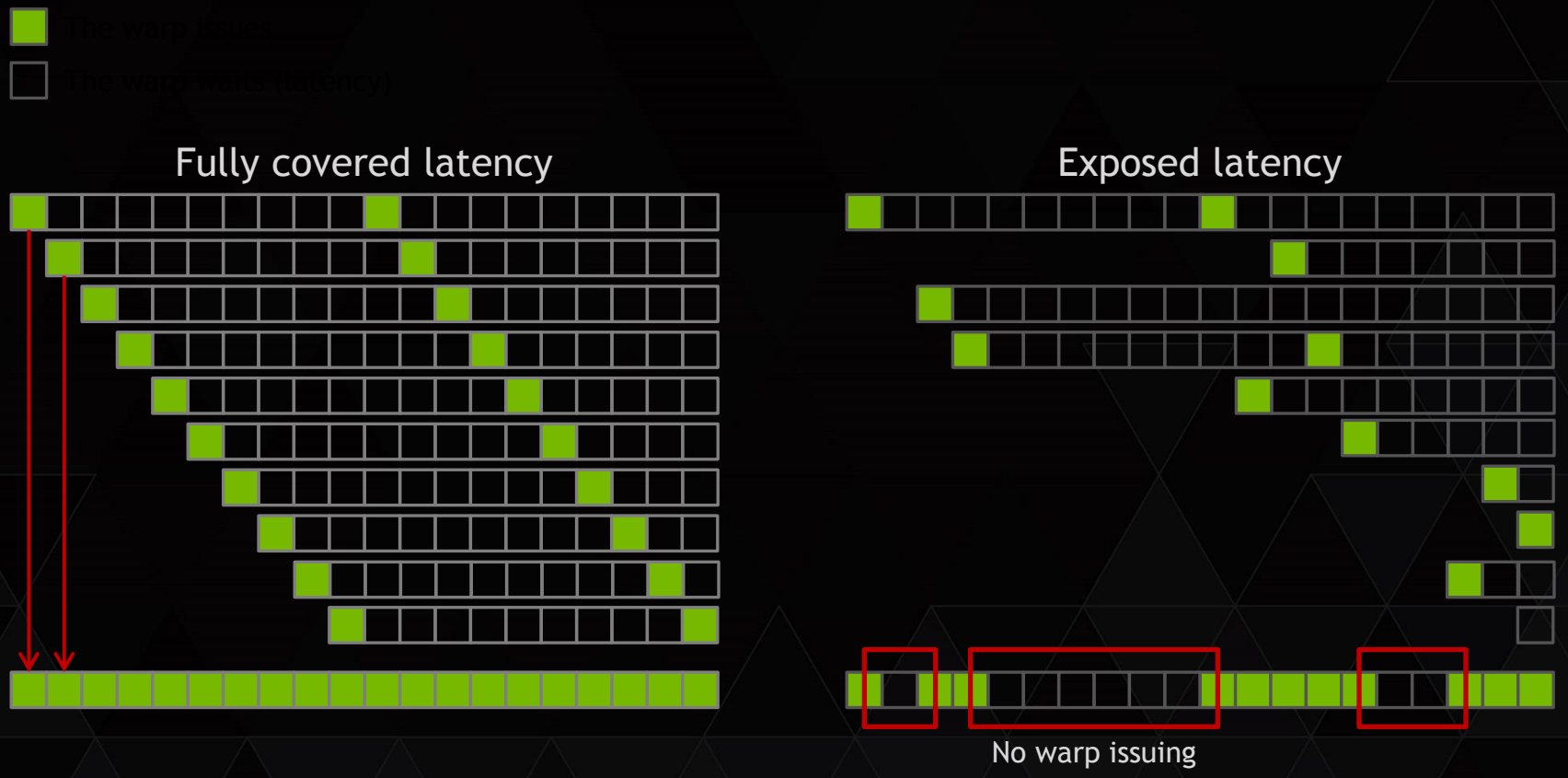
Latency Bound

LOOKING FOR MORE INDICATORS



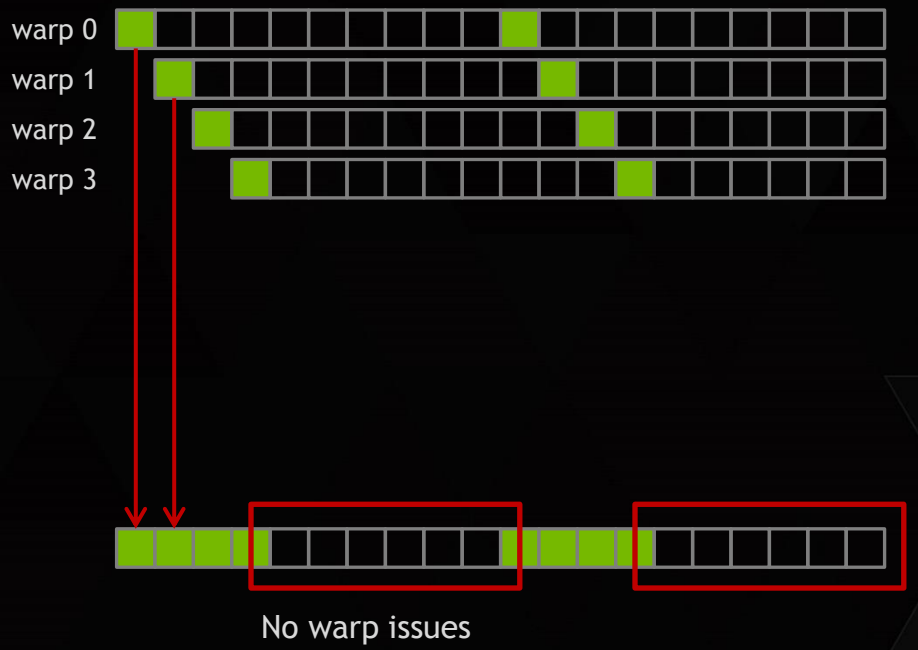
LATENCY

- 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

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]);
```

```
c = b[0]
a += logf(c)
c = b[1]
a += logf(c)
c = b[2]
a += logf(c)
c = b[3]
a += logf(c)
```

2-way ILP (with loop unrolling)

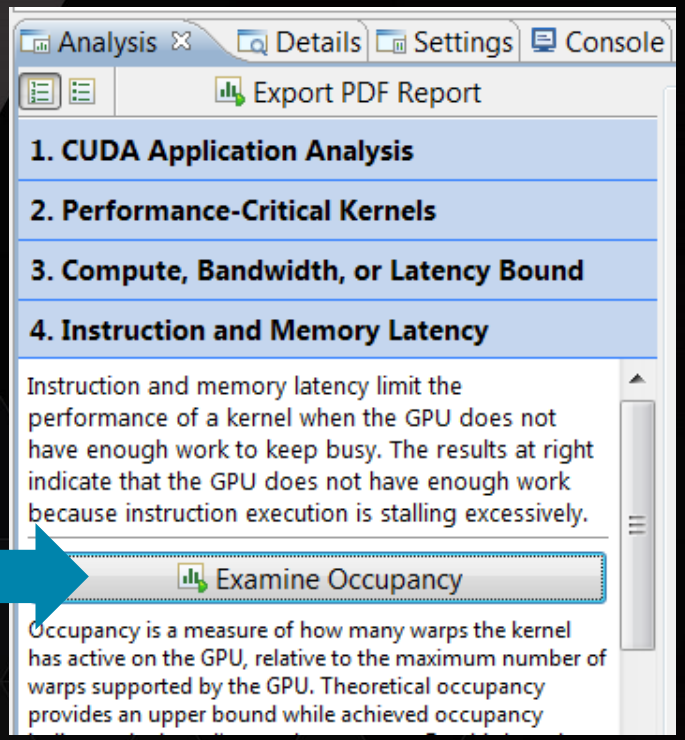
```
float a, a0 = 0.0f, a1 = 0.0f;
for( int i = 0 ; i < N ; i += 2 )
{
  a0 += logf(b[i]);
  a1 += logf(b[i+1]);
}
a = a0 + a1
```

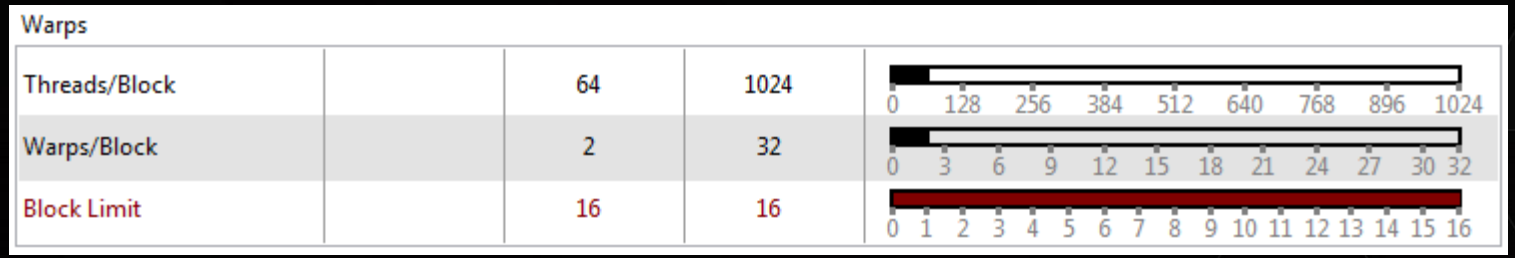
```
c0 = b[0]
a0 += logf(c0)
c0 = b[2]
a0 += logf(c0)
...
a = a0 + a1
```

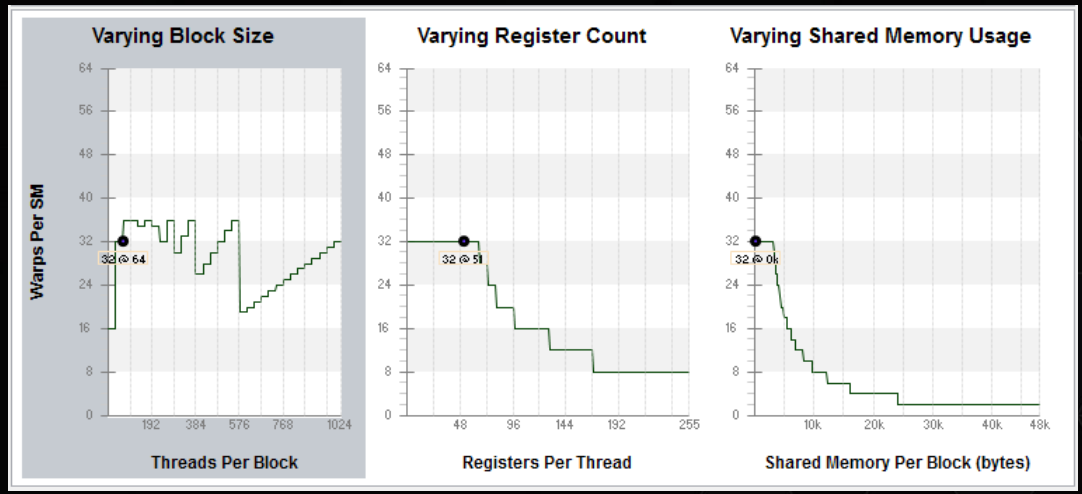
```
c1 = b[1]
a1 += logf(c1)
c1 = b[3]
a1 += logf(c1)
```

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

LOOKING FOR MORE INDICATORS







- Not enough active warps to hide latencies?

IMPROVED OCCUPANCY

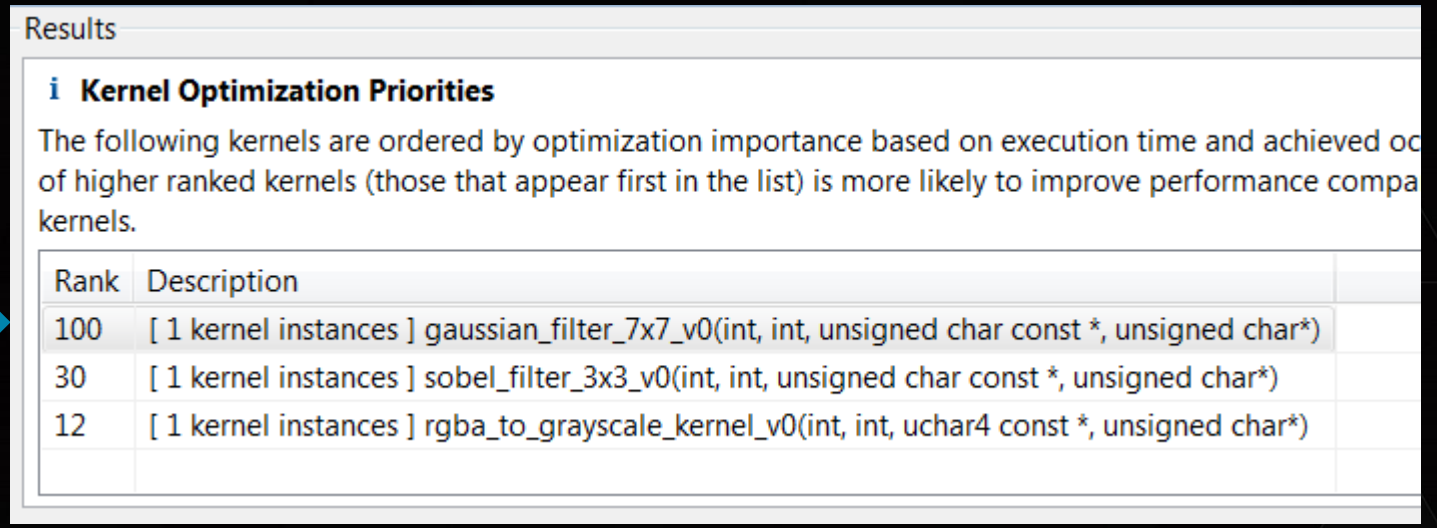
- Bigger blocks of size 32x4
- Increases achieved occupancy slightly (from 47.6% to 52.4%)

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

ITERATION 3

IDENTIFY HOTSPOT

Hotspot



- gaussian_filter_7x7_v0() still the hotspot

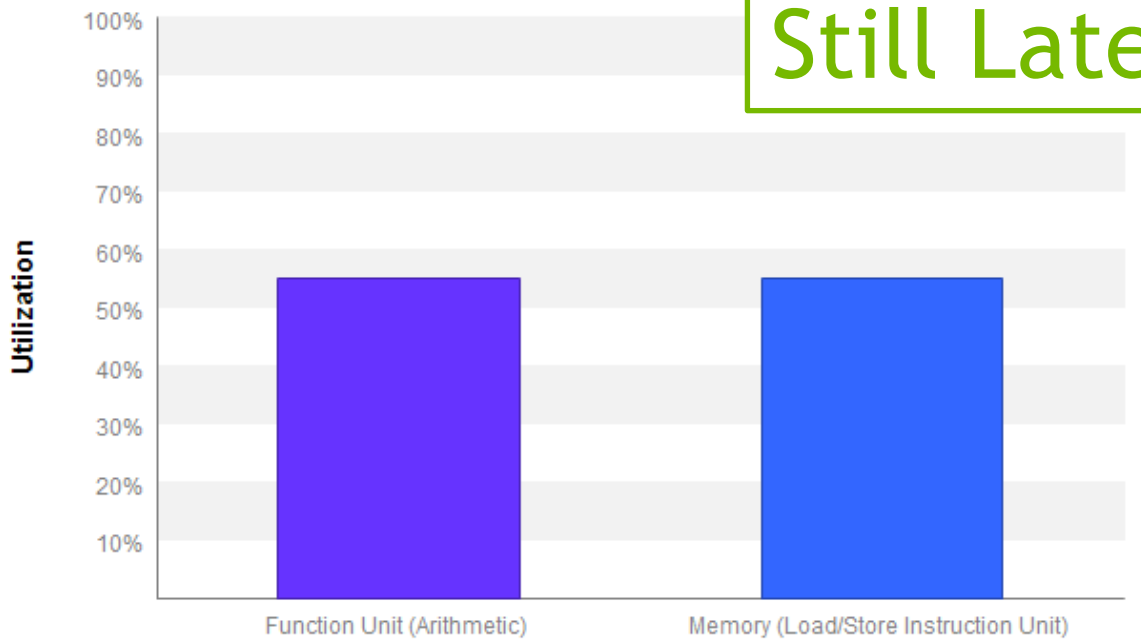
Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x
Higher Occupancy	1.562ms	3.35x

IDENTIFY PERFORMANCE LIMITER

Results

Kernel Performance Is Bound By Instruction And Memory Latency

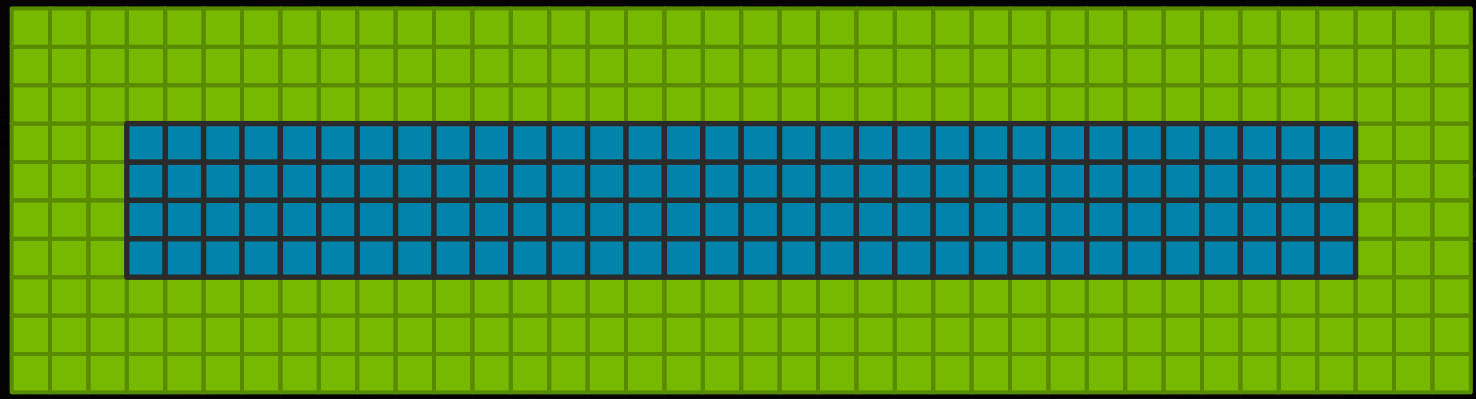
This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K40m". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or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



Still Latency Bound

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];
```

- Apart from higher bandwidth shared memory also has lower latency!

SHARED MEMORY

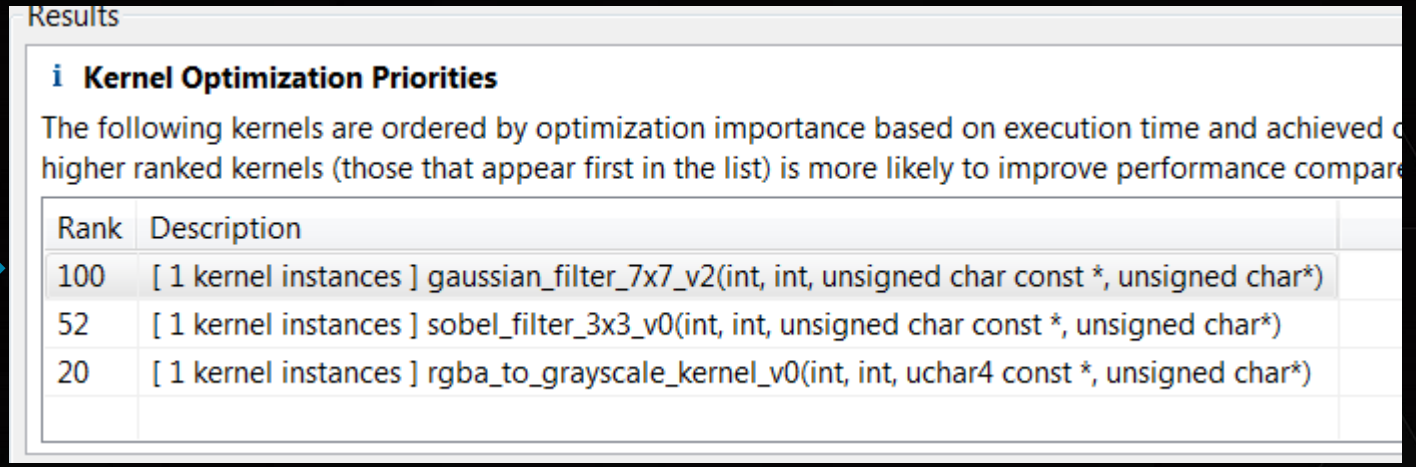
- Using shared memory for the Gaussian Filter
- Significant speedup, < 1ms

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

ITERATION 4

IDENTIFY HOTSPOT

Hotspot



- gaussian_filter_7x7_v0() still the hotspot

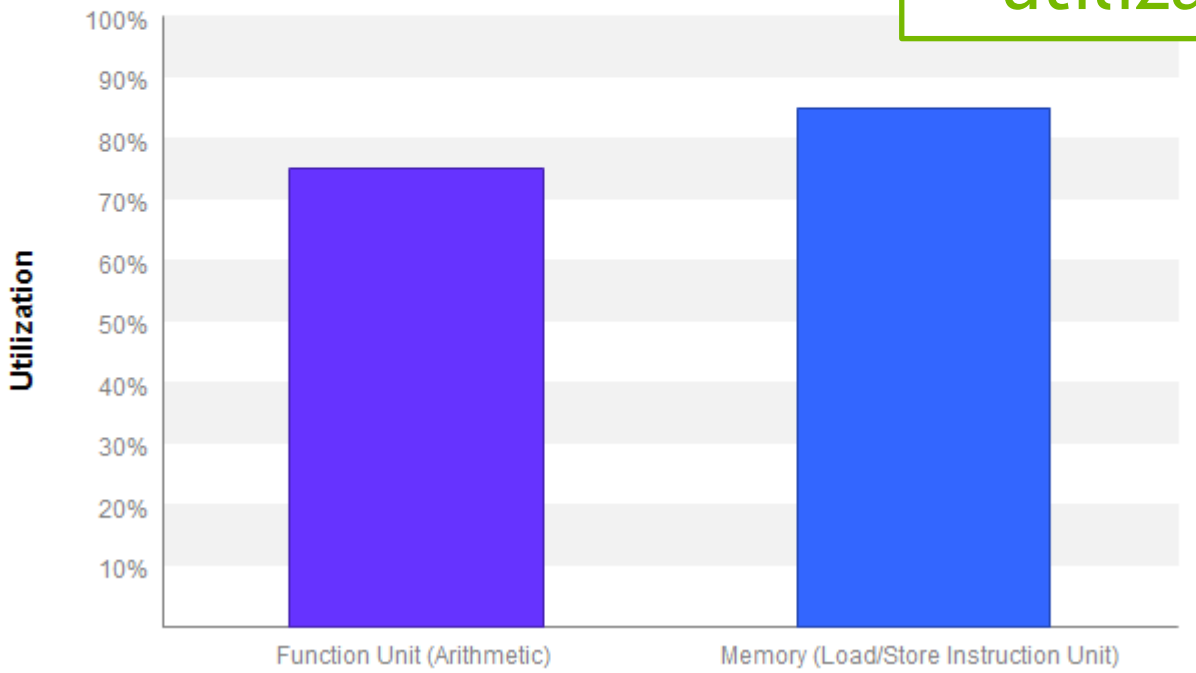
Kernel	Time	Speedup
Original Version	5.233ms	1.00x
Better Memory Accesses	1.589ms	3.29x
Higher Occupancy	1.562ms	3.35x
Shared Memory	0.911ms	5.74x

IDENTIFY PERFORMANCE LIMITER

Results

Kernel Performance Is Bound By Compute And Memory Bandwidth

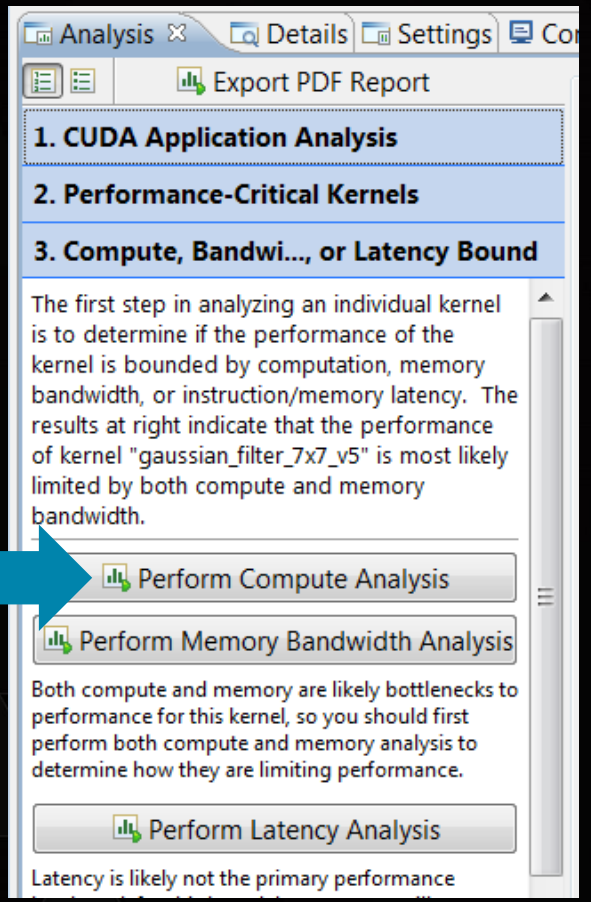
For device "Tesla K40m" compute and memory utilization are balanced. These utilizatio
good, but that additional performance improvement may be possible if either of both c
are increased.



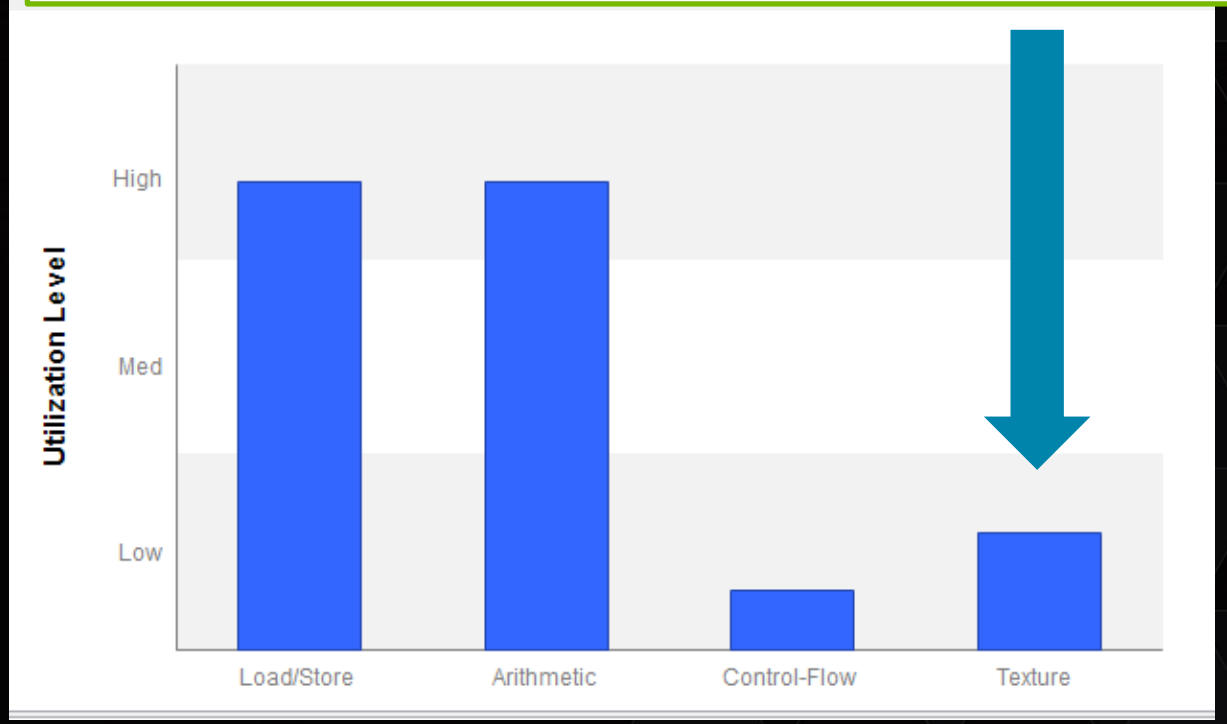
Aha!
Getting into the high utilization region

LOOKING FOR INDICATORS

Launch



Can we move LD/ST work here?



READ-ONLY PATH

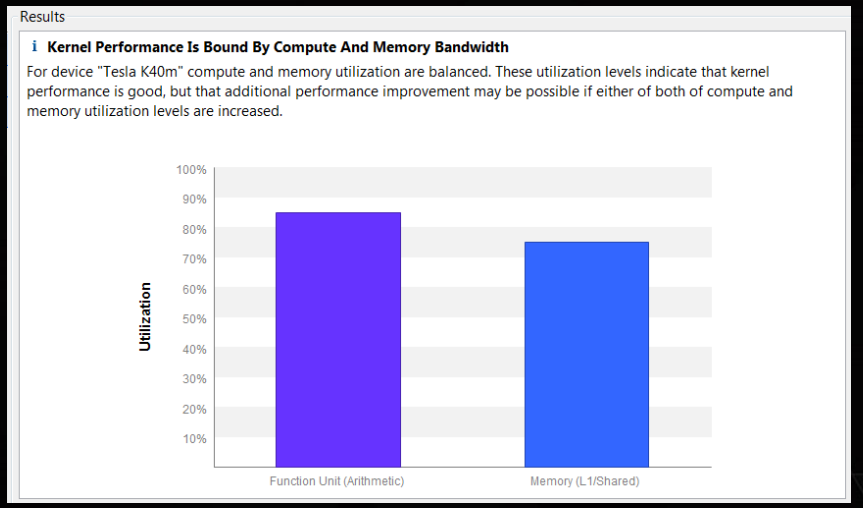
- Annotate read-only parameters with const __restrict (or use the __ldg intrinsic)

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

- The compiler generates LDG instructions that load through TEX instead of Load/Store

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

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*)

THANK YOU