GPU TECHNOLOGY CONFERENCE

April 4-7, 2016 | Silicon Valley

CUDA OPTIMIZATION WITH NVIDIA NSIGHT™ ECLIPSE EDITION

CHRISTOPH ANGERER, NVIDIA

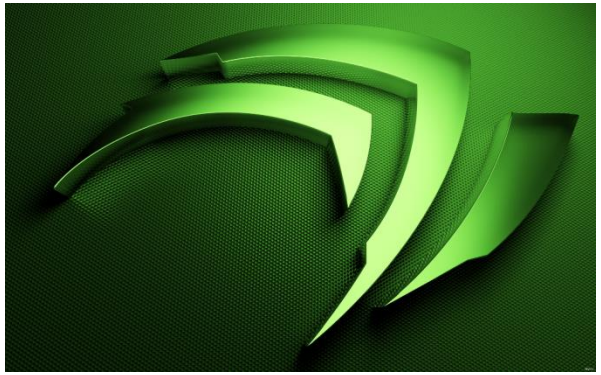
MATHIAS WAGNER, NVIDIA

PRESENTED BY NVIDIA.

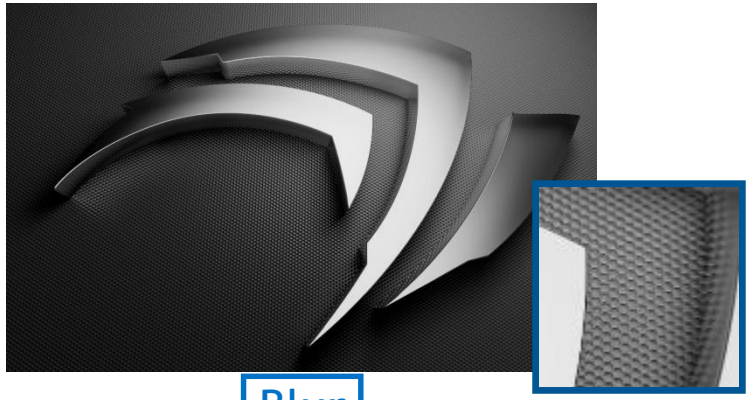
WHAT YOU WILL LEARN

- 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-gtc

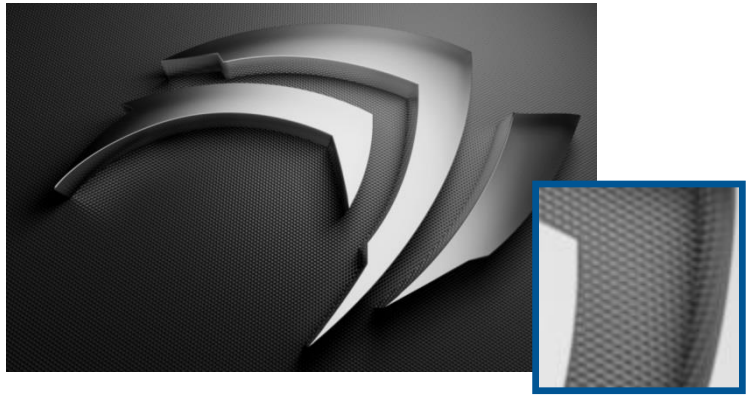
INTRODUCING THE APPLICATION



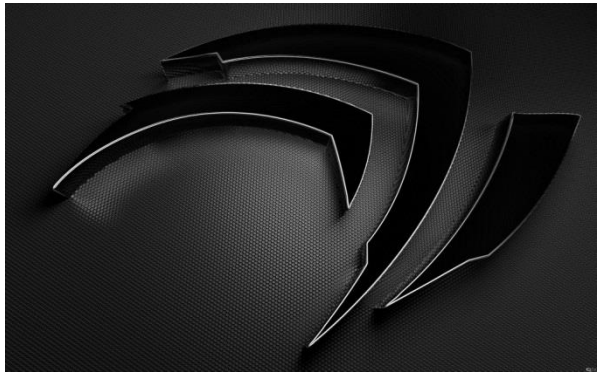
Grayscale



Blur



Edges



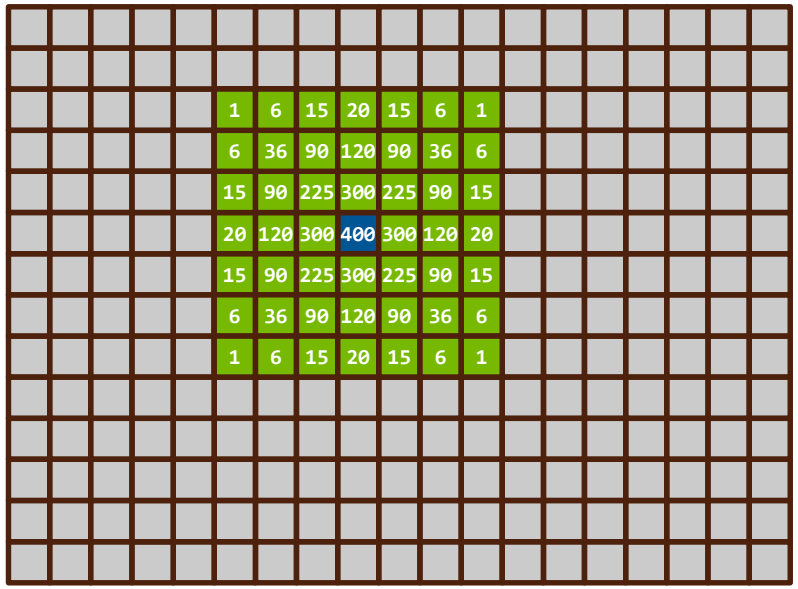
INTRODUCING THE APPLICATION

- 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
```

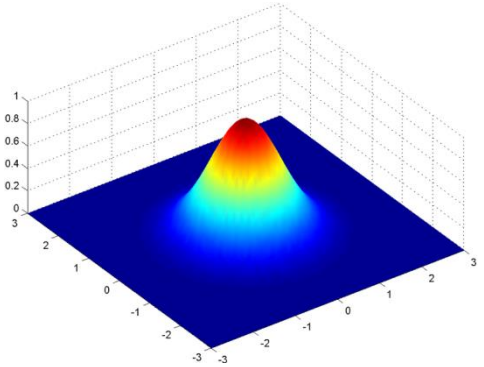
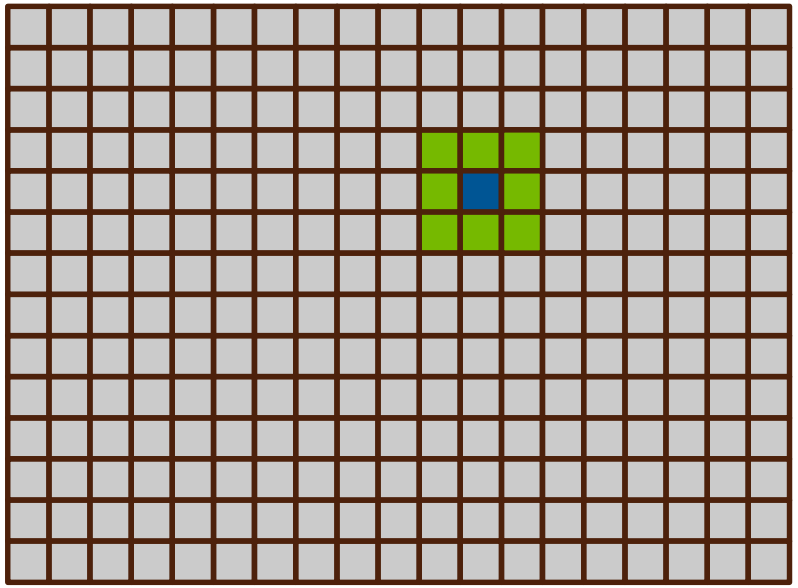


Image from Wikipedia

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

ENVIRONMENT

- NVIDIA Tesla K80
  - GK210
  - SM3.7
  - ECC on
  - 2505 MHz memory clock, 875 MHz SM clock
- CUDA 7.5, Linux

- Tools identical for Windows, Linux, and Mac

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)

THE APOD CYCLE

1. Assess

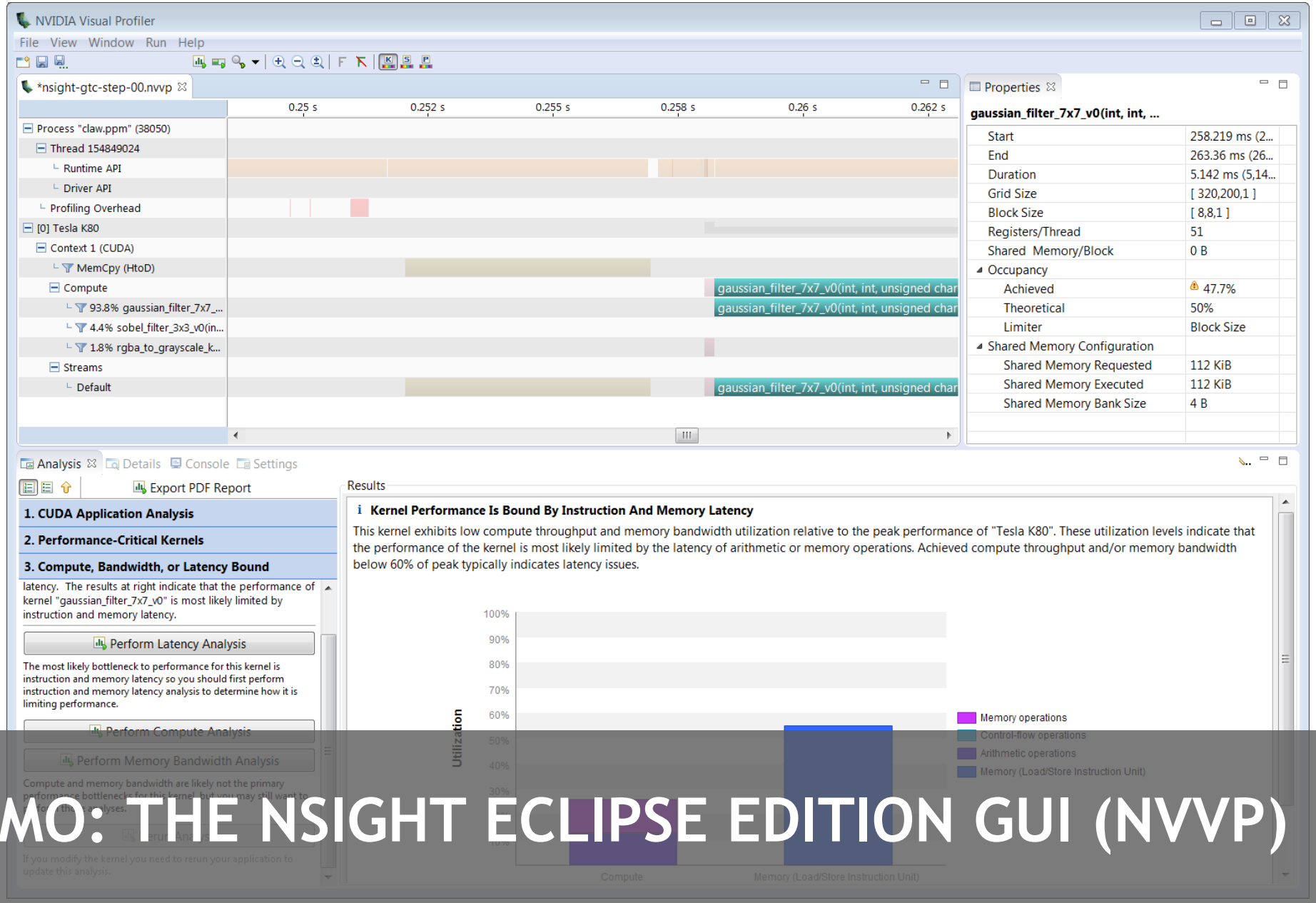
- Identify Performance Limiter
- Analyze Profile
- Find Indicators

2. Parallelize

3. Optimize

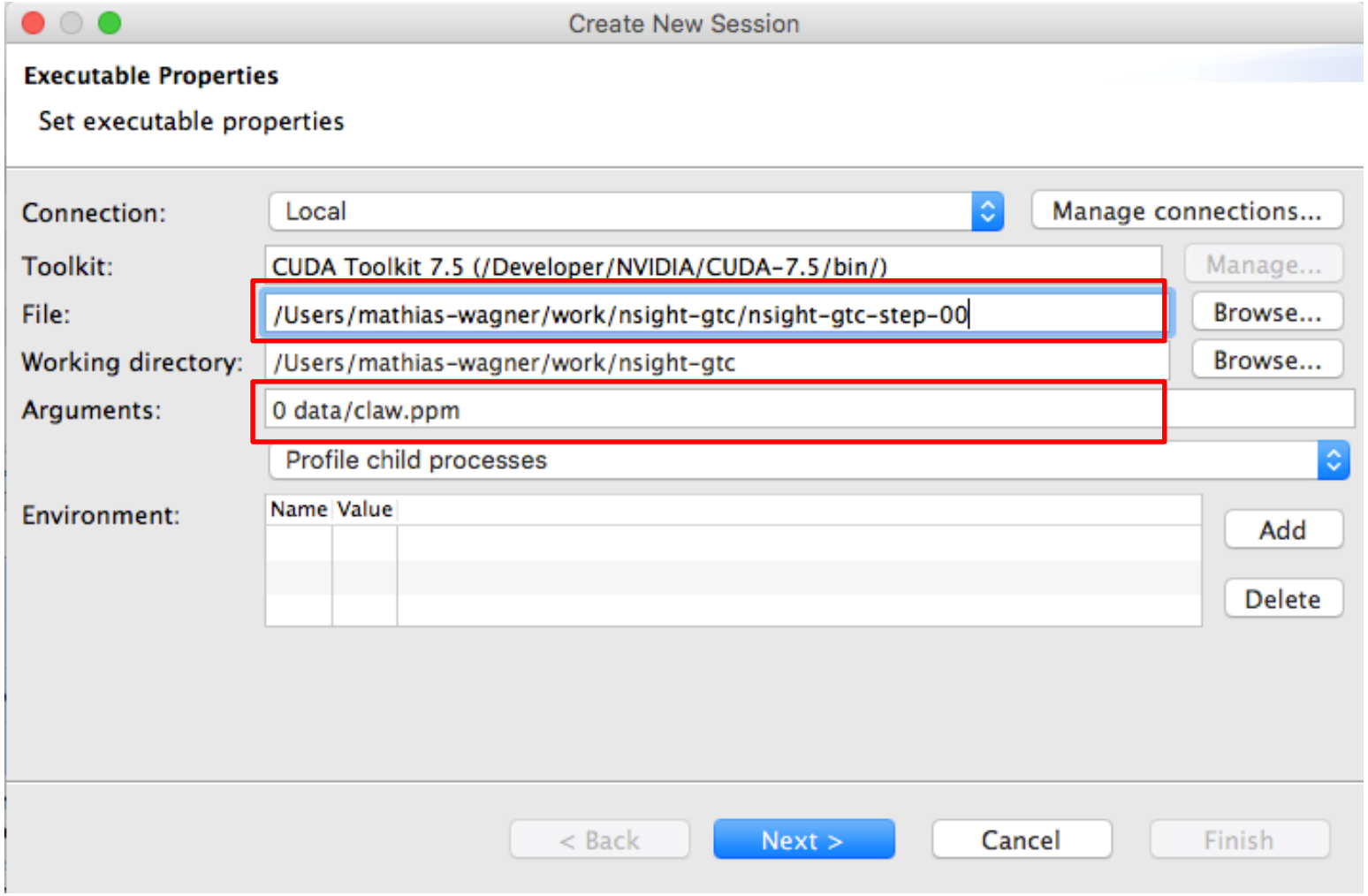
3b. Build Knowledge

4. Deploy and Test



DEMO: THE NSIGHT ECLIPSE EDITION GUI (NVVP)

CREATE A NEW NVVP SESSION



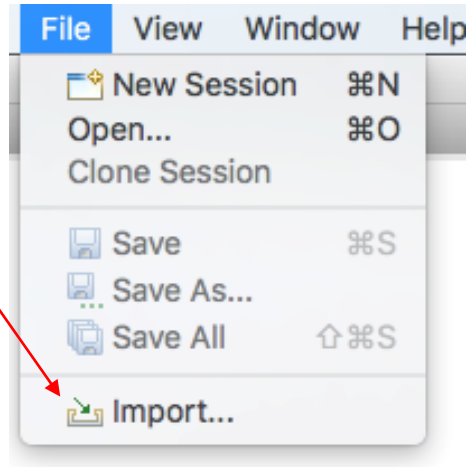
IF REQ'D: REMOTE PROFILING

- GPU you want to profile for might not be available in your local workstation

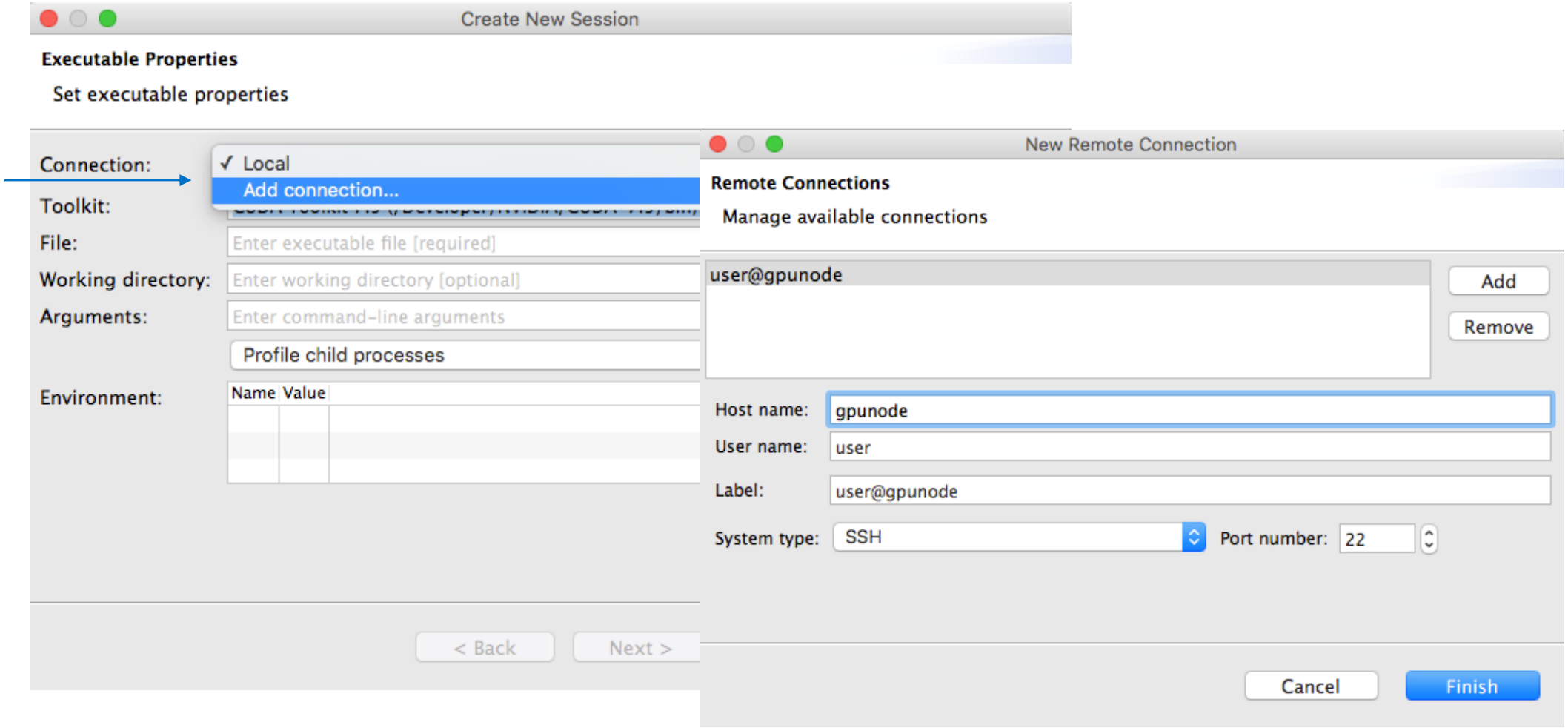
- various approaches to remote profiling
  A. Run profiler on remote machine and use remote desktop (X11 forwarding, NX, VNC, …)
  B. Collect data using command line profiler *nvprof* and view your local workstation

```
# generate time line
nvprof -o myprofile.timeline ./a.out
# collect data needed for guided analysis
nvprof -o myprofile.analysis --analysis-metrics ./a.out
# custom selection of metrics for detailed investigations
nvprof -o myprofile.metrics --metrics <...> ./a.out
```

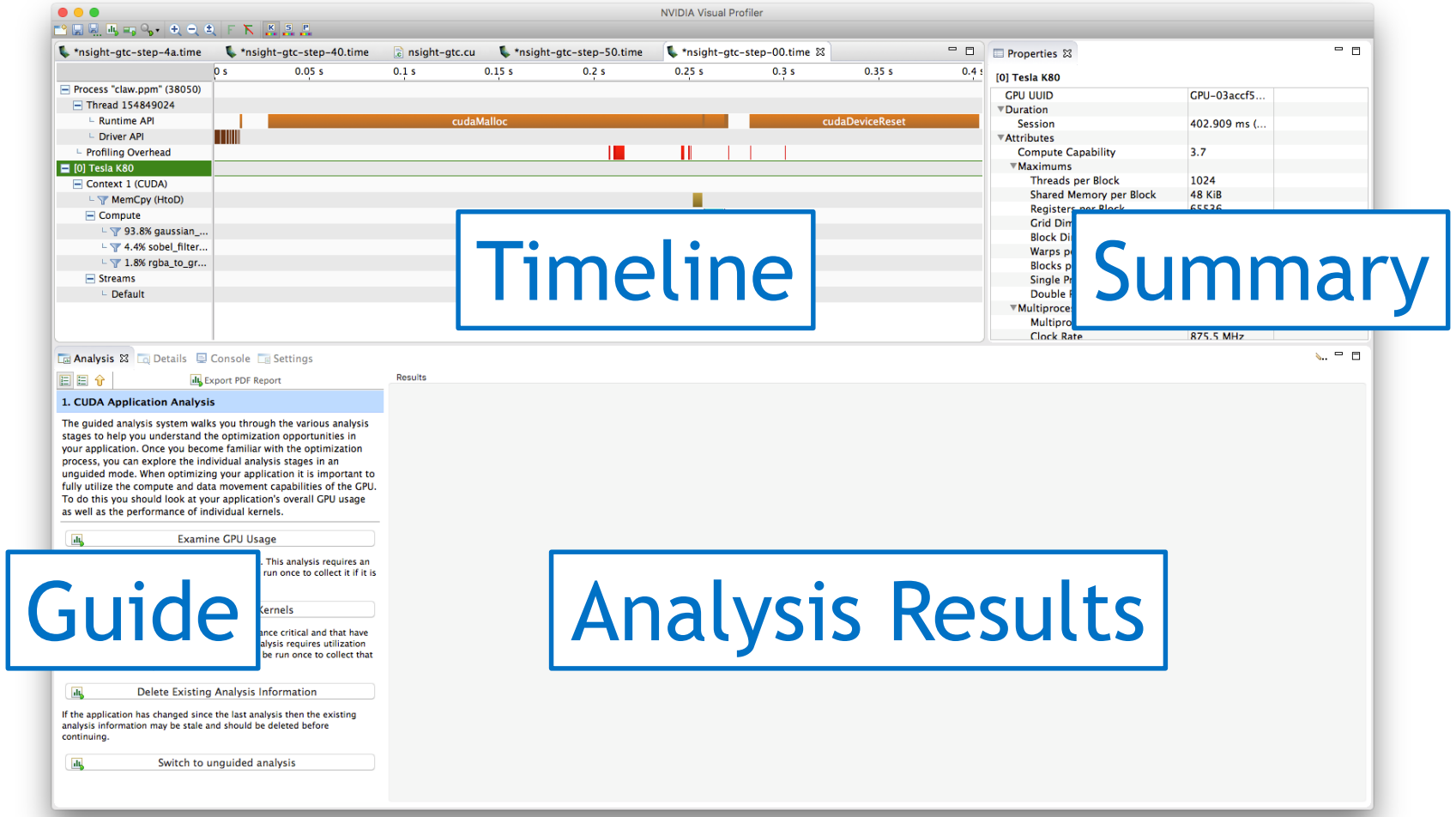
  C. Use remote connection feature to create a new session



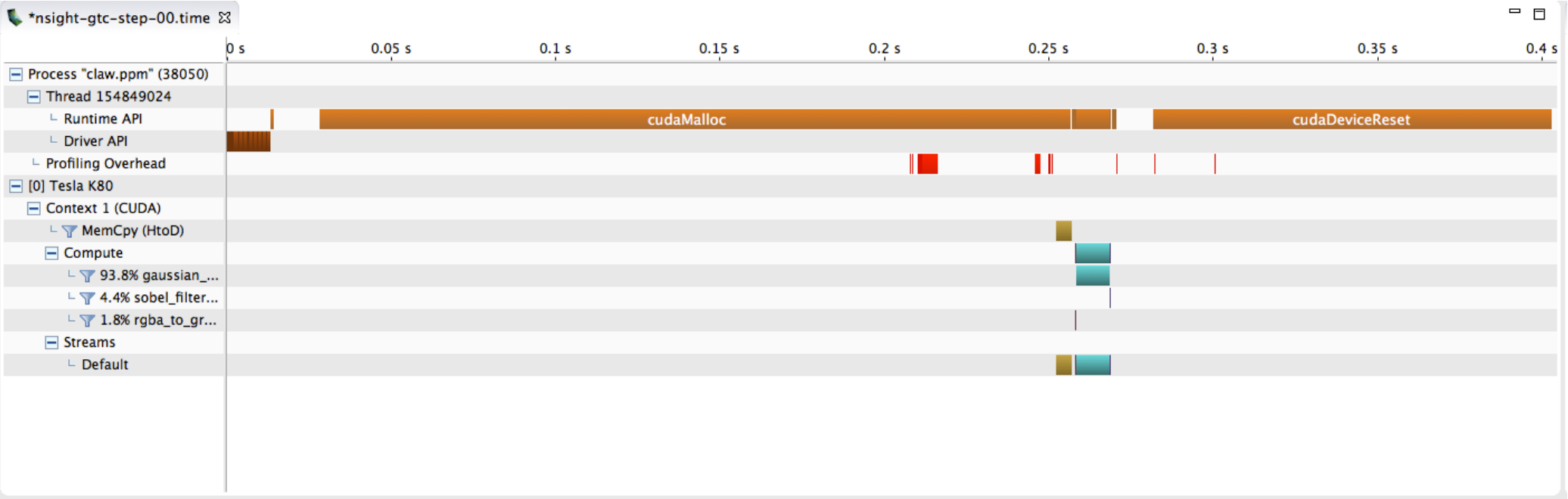
CREATE A REMOTE CONNECTION



THE PROFILER WINDOW



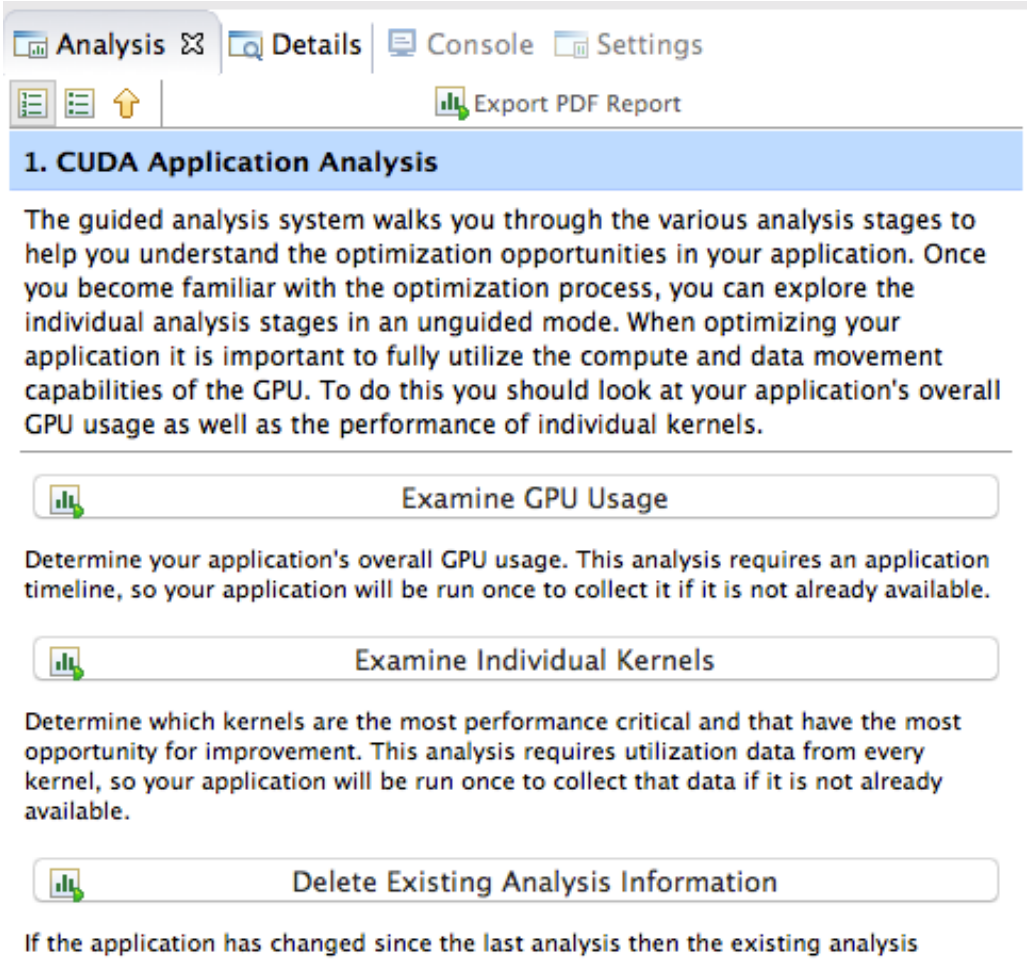
TIMELINE


*nsight-gtc-step-00.time
0 s
0.05 s
0.1 s
0.15 s
0.2 s
0.25 s
0.3 s
0.35 s
0.4 s
Process "claw.ppm" (38050)
Thread 154849024
Runtime API
cudaMalloc
cudaDeviceReset
Driver API
Profiling Overhead
[0] Tesla K80
Context 1 (CUDA)
MemCpy (HtoD)
Compute
93.8% gaussian_...
4.4% sobel_filter...
1.8% rgba_to_gr...
Streams
Default

ITERATION 1

EXAMINE INDIVIDUAL KERNELS
(GUIDED ANALYSIS)

Launch



IDENTIFY HOTSPOT

Hotspot

Kernel Optimization Priorities

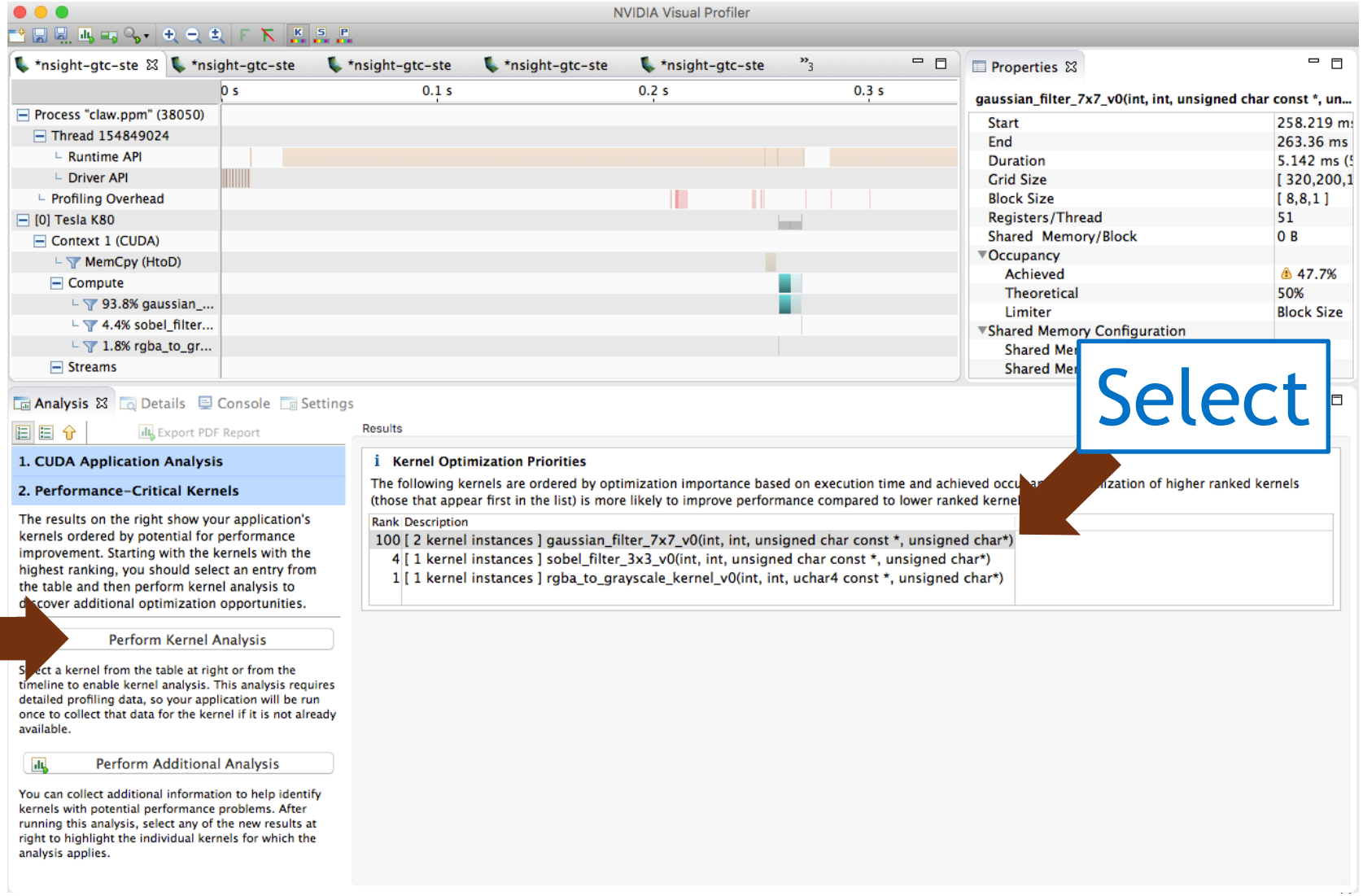
The following kernels are ordered by optimization importance based on execution time and achieved occupancy. Optimization of higher ranked kernels (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[2 kernel instances] gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)
4	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
1	[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	5.141ms	1.00x

PERFORM KERNEL ANALYSIS

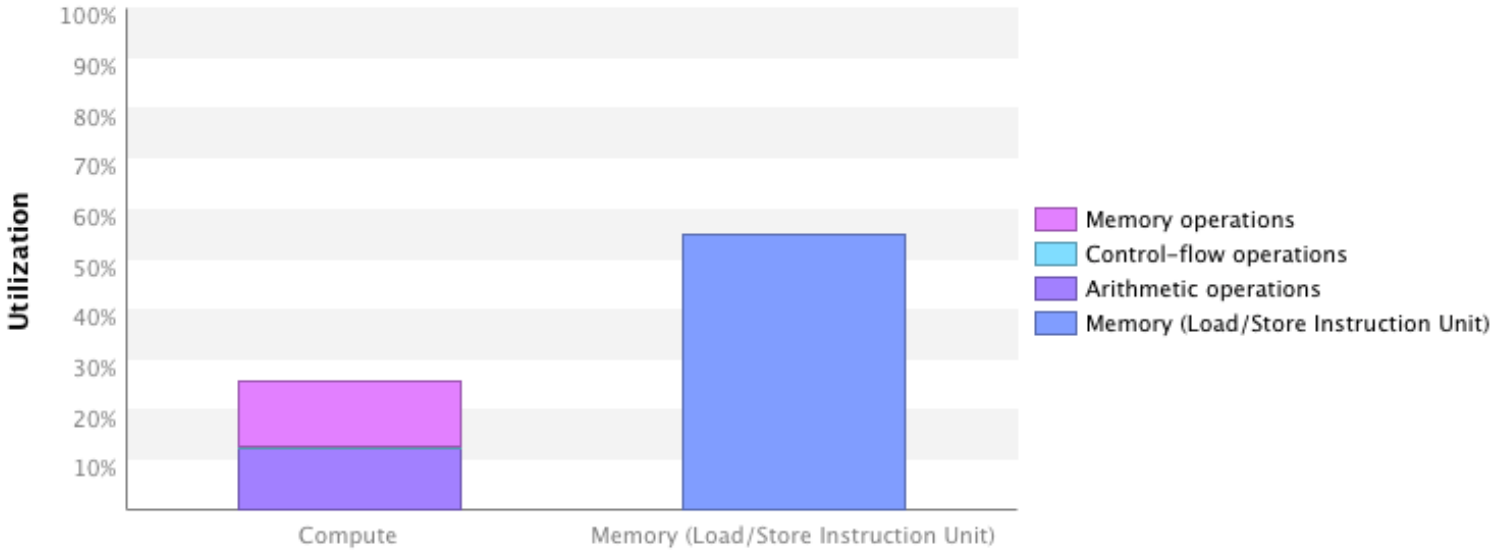


GPU TECHNOLOGY CONFERENCE
NVIDIA

IDENTIFY PERFORMANCE LIMITER

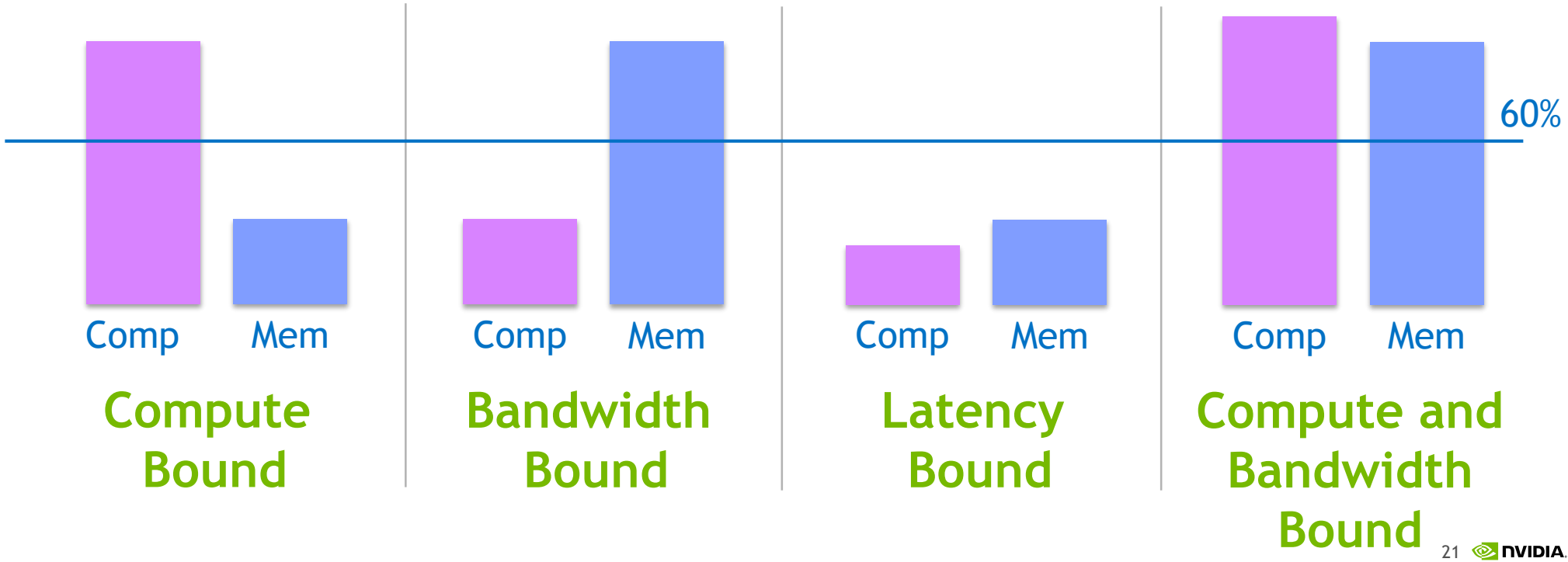
Kernel Performance Is Bound By Instruction And Memory Latency

This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Tesla K80". 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

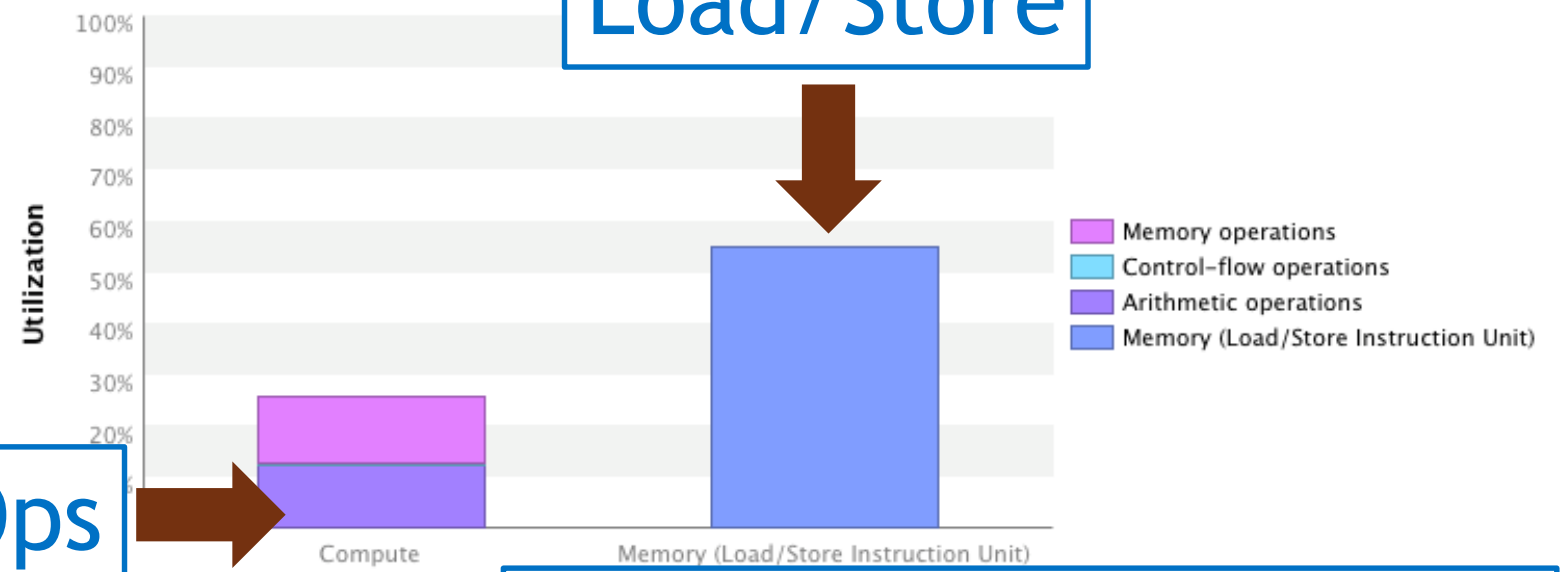
- Memory Utilization vs Compute Utilization
- Four possible combinations:



IDENTIFY PERFORMANCE LIMITER

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 K80". 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?

LOOKING FOR INDICATORS

Analysis | Details | Console | Settir

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

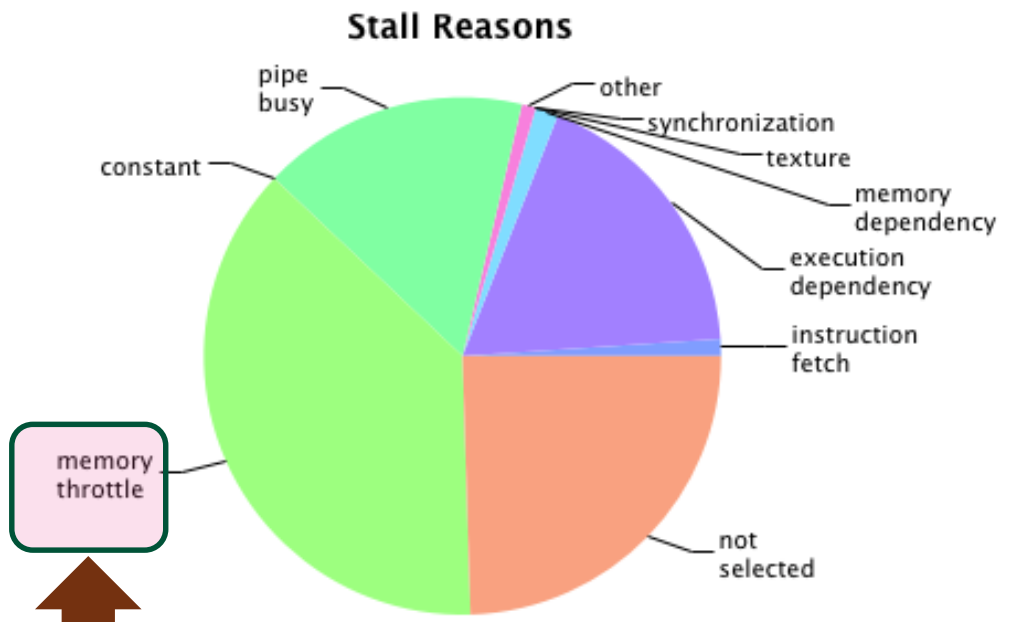
The most likely bottleneck to performance for this kernel is instruction and memory latency so you should first perform instruction and memory latency analysis to determine how it is limiting performance.

Perform Compute Analysis

Perform Memory Bandwidth Analysis

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

Launch



Large number of memory operations stalling LSU

LOOKING FOR MORE INDICATORS

Unguided Analysis

nsight-gtc.cu | *nsight-gtc-ste | *nsight-gtc-ste | *nsight-gtc-ste

```
// 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.
```

Analysis | Details | Console | Settings

Reset All | Analyze All

gaussian_filter_7x7_v0(int, int, unsigned char

- Kernel Performance Limiter
- Kernel Latency
- Kernel Compute
- Kernel Memory
- Global Memor...cess Pattern
- Shared Memor...cess Pattern

Results

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 or access pattern. For each load or store improve the alignment and access pattern of the memory access.

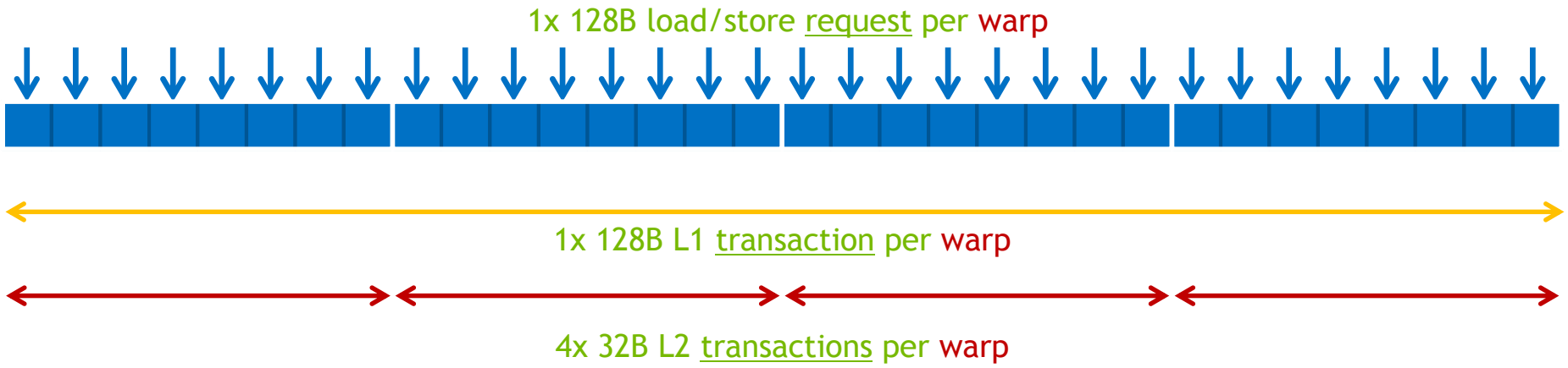
Line / File	nsight-gtc.cu – /home-2/mathiasw/nsight-gtc
245	Global Load L2 Transactions/Access = 5, Ideal Transactions/Access = 1 [637203 L2 transactions for 128000 total executio
245	Global Load L2 Transactions/Access = 5, Ideal Transactions/Access = 1 [637203 L2 transactions for 128000 total executio
245	Global Load L2 Transactions/Access = 5, Ideal Transactions/Access = 1 [637602 L2 transactions for 128000 total executio
245	Global Load L2 Transactions/Access = 4, Ideal Transactions/Access = 1 [511360 L2 transactions for 128000 total executio
245	Global Load L2 Transactions/Access = 5, Ideal Transactions/Access = 1 [637602 L2 transactions for 128000 total executio
245	Global Load L2 Transactions/Access = Ideal Transactions/Access = 1 [637602 L2 transactions for 128000 total executio
245	Global Load L2 Transactions/Acces al Transactions/Access = 1 [637602 L2 transactions for 128000 total executio
245	Global Load L2 Transactions/Access deal Transactions/Access = 1 [638400 L2 transactions for 128000 total executio
245	Global Load L2 Transactions/Access deal Transactions/Access = 1 [638400 L2 transactions for 128000 total executio

For line numbers use:
`nvcc -lineinfo`

4-5 Global Load/Store Transactions per 1 Request

MEMORY TRANSACTIONS: BEST CASE

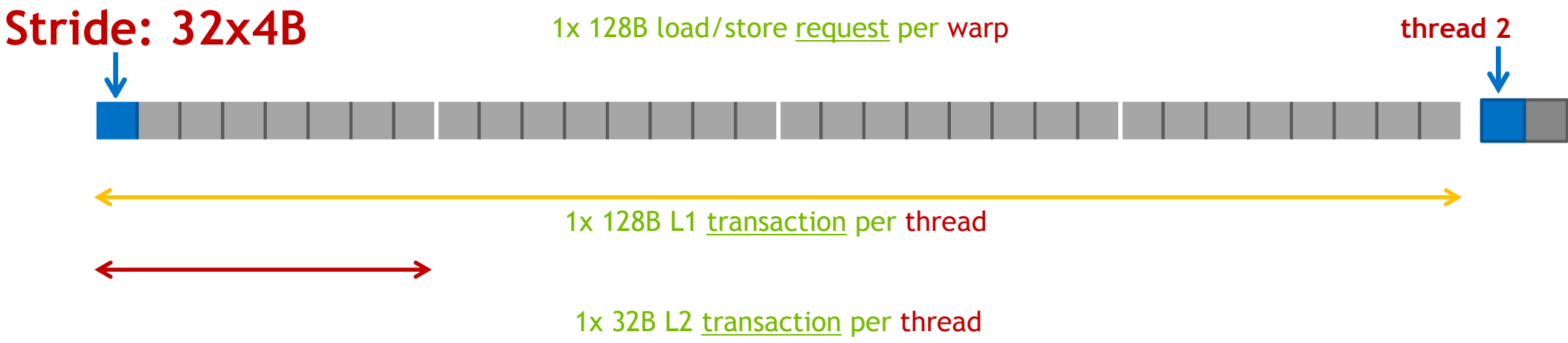
- A warp issues 32x4B aligned and consecutive load/store request
- Threads read different elements of the same 128B segment



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

MEMORY TRANSACTIONS: WORST CASE

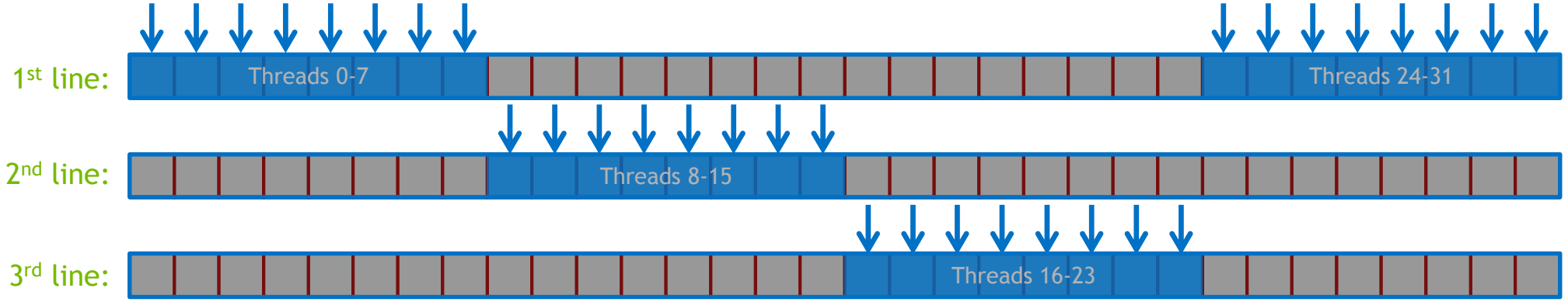
- Threads in a warp read/write 4B words, 128B between words
- Each thread reads the first 4B of a 128B segment



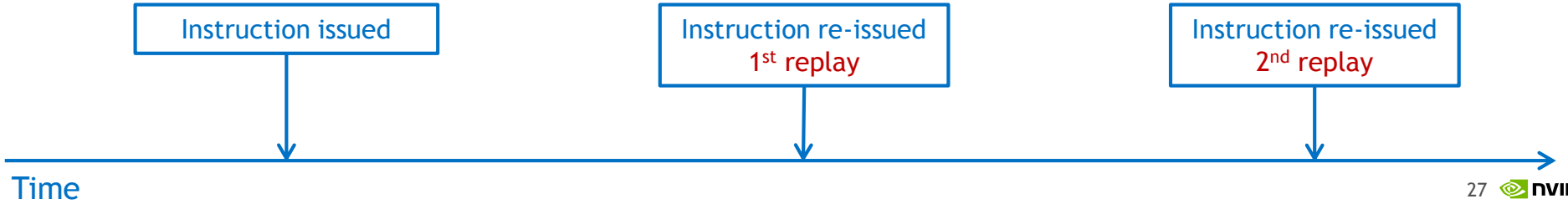
- 32x L1 transactions: 128B needed / 32x 128B transferred
- 32x L2 transactions: 128B needed / 32x 32B transferred

TRANSACTIONS AND REPLAYS

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

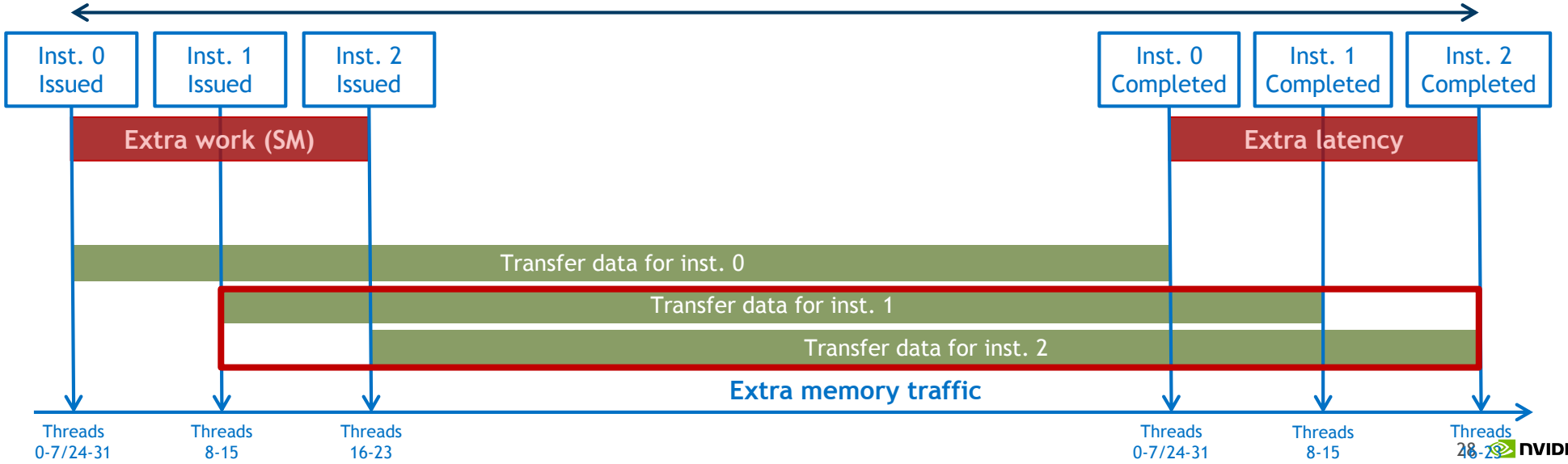


- 1 instr. executed and 2 replays = 1 request and 3 transactions



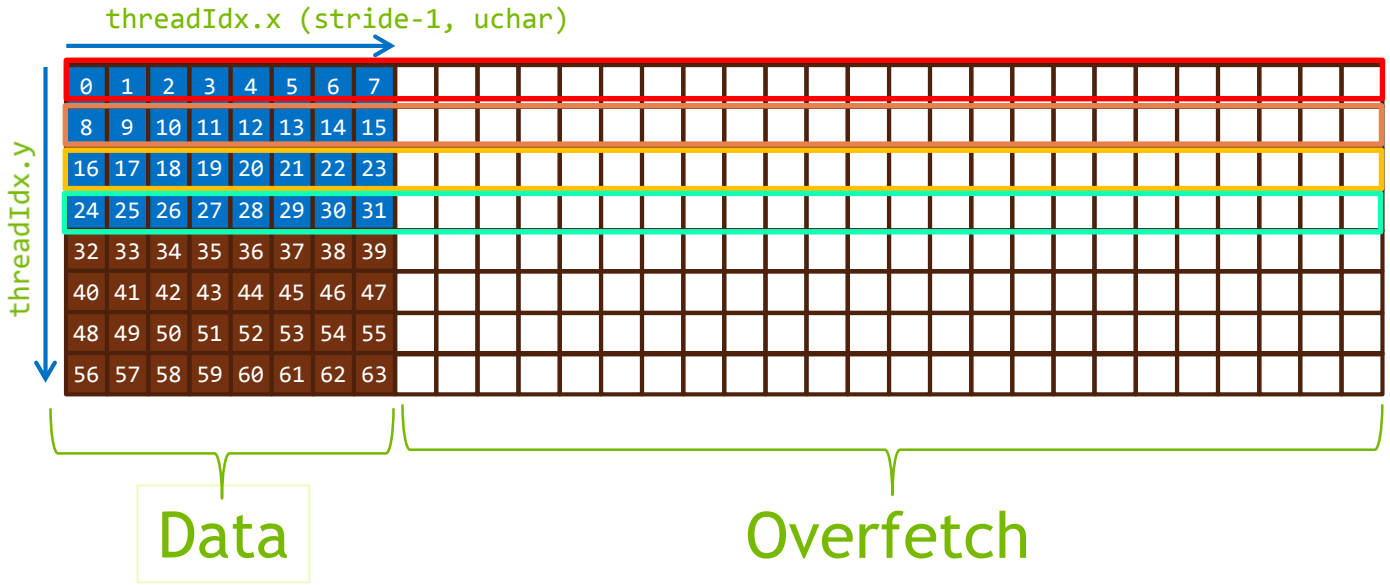
TRANSACTIONS AND REPLAYS

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

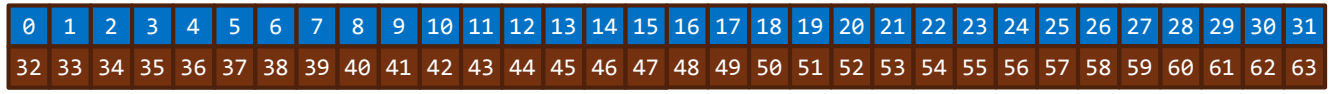


CHANGING THE BLOCK LAYOUT

- 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	5.141ms	1.00x
Better Memory Accesses	1.572ms	3.27x

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

Results

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*)
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

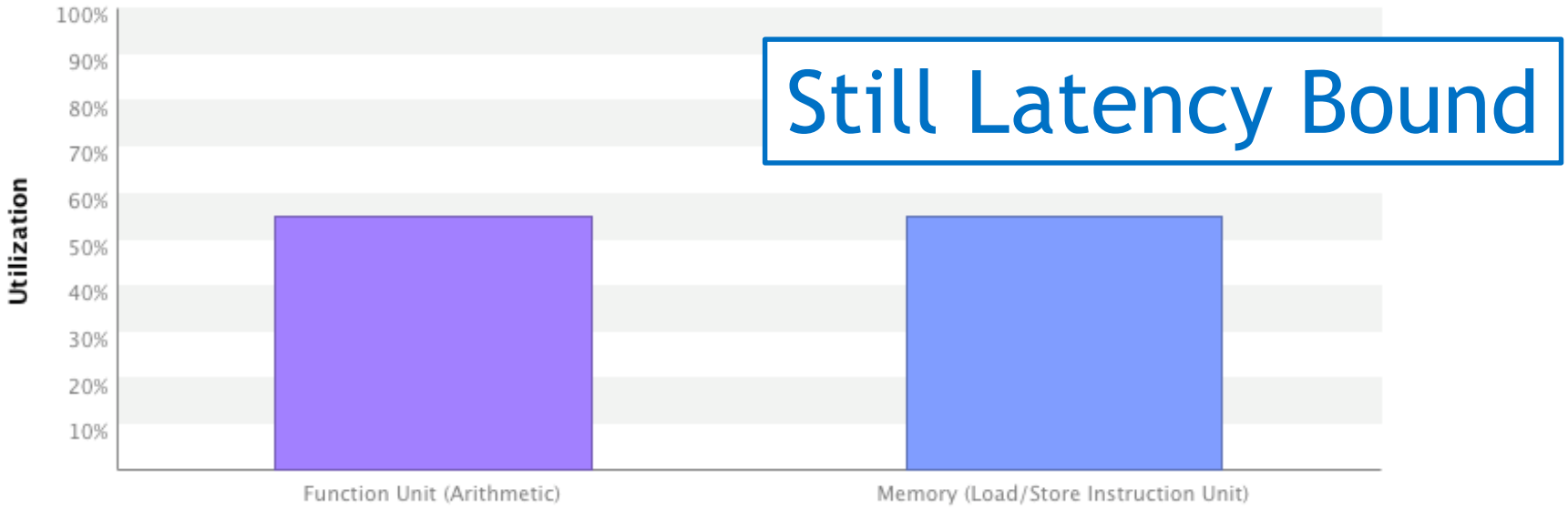
Kernel	Time	Speedup
Original Version	5.141ms	1.00x
Better Memory Accesses	1.5724ms	3.27x

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 K80". 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 MORE INDICATORS

Medium L2 Bandwidth Utilization

Analysis | Details | Console | Settings

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 Ba

Compute and memory bandwidt
primary performance bottlenecks
may still want to perform those a

Rerun Analysis

If you modify the kernel you need to rerun your application to update this analysis.

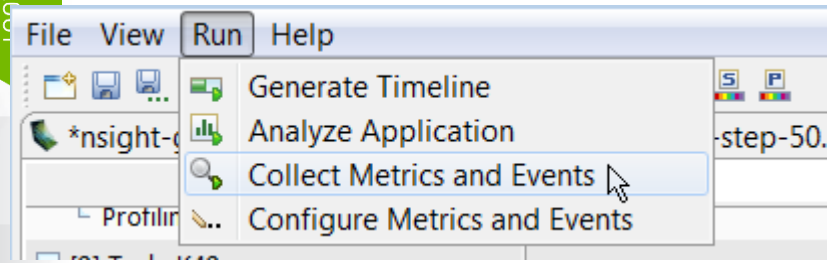
L2 Cache			
L1 Reads	11568392	236.316 GB/s	
L1 Writes	128000	2.615 GB/s	
Texture Reads	0	0 B/s	
Noncoherent Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	11696392	238.931 GB/s	Idle Low Medium High Max
Texture Cache			
Reads	0	0 B/s	Idle Low Medium High Max
Device Memory			
Reads	221370	4.522 GB/s	
Writes	223302	4.562 GB/s	
Total	444672	9.084 GB/s	Idle Low Medium High Max
ECC Overhead	188446	3.85 GB/s	

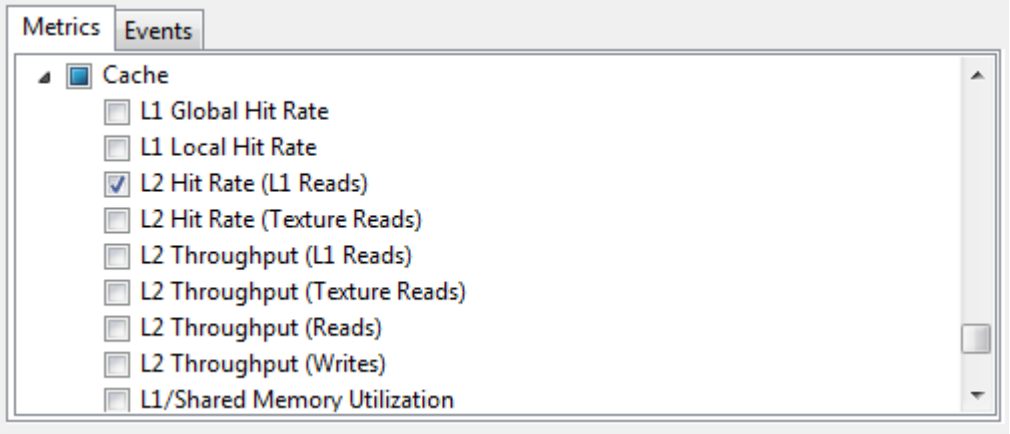
Launch

Very low device memory bandwidth utilization

Is our working set mostly in L2$?

CHECKING L2 HIT RATE: 98.9%



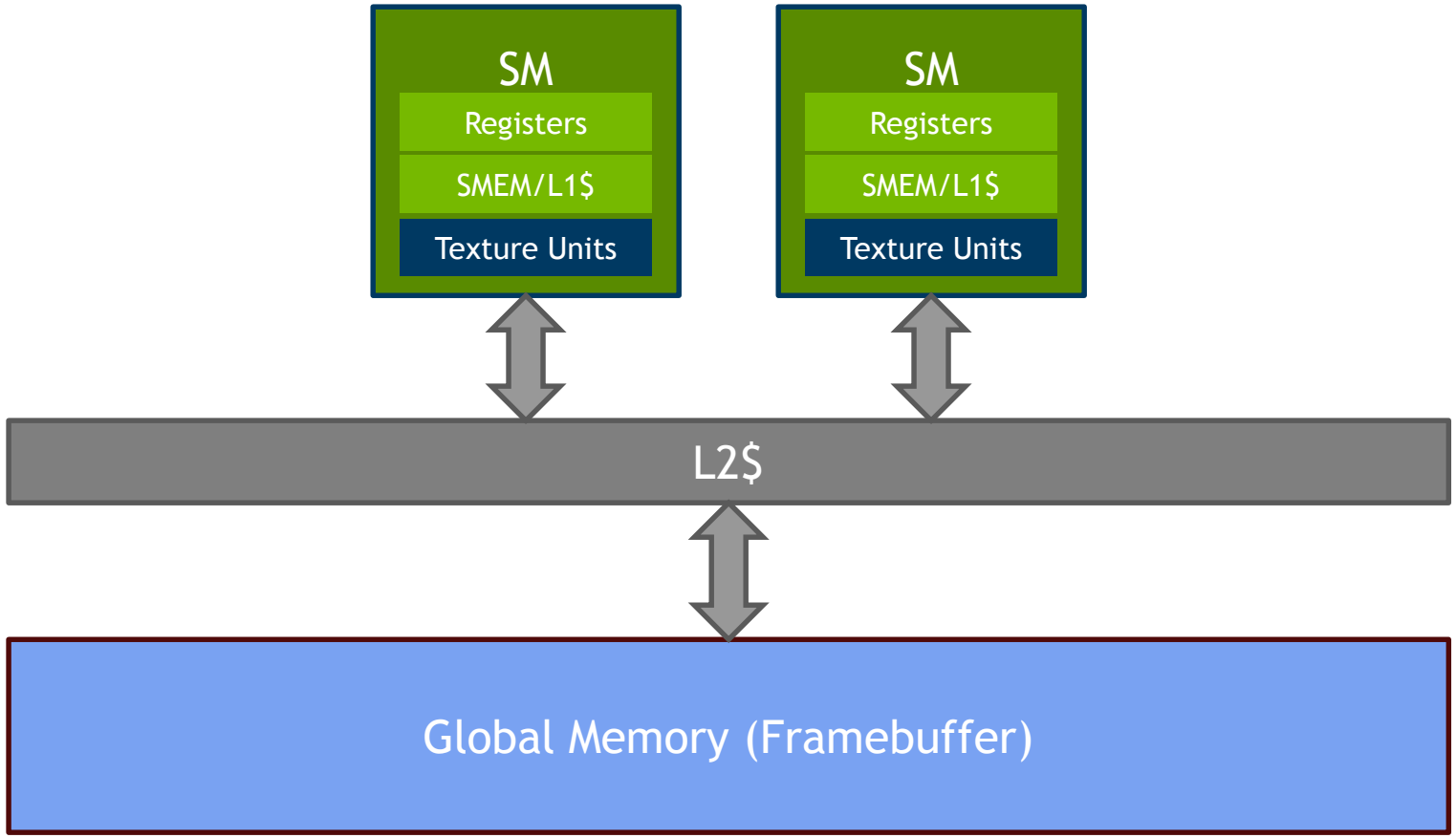


Analysis | Details | Console | Settings

Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem	L2 Hit Rate (L1 Reads)
gaussian_filter_7x7_v0(int, int, unsigned char const *, unsigned char*)	259.079 ms	1.572 ms	[80,800,1]	[32,2,1]	51	0	0	98.9%

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

MEMORY BANDWIDTH



OPT-IN L1 CACHING

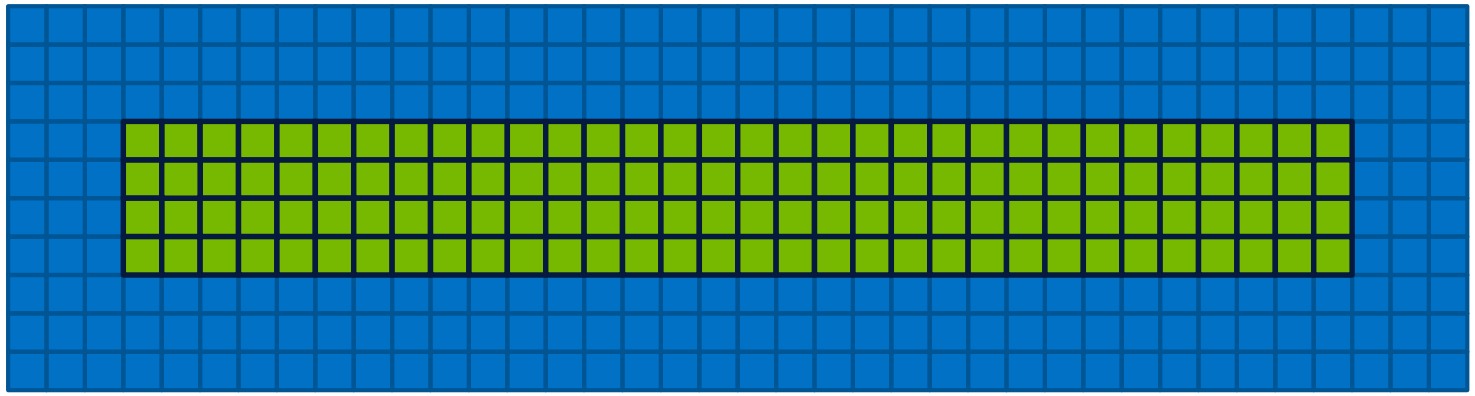
Compiler Option

"Global memory accesses for devices of compute capability 3.x are cached in L2 […]; they are normally not cached in L1. Some devices of compute capability 3.5 and devices of compute capability 3.7 allow opt-in to caching of global memory accesses in L1 via the -Xptxas -dlcm=ca option to nvcc"

Kernel	Time	Speedup	Rel. Speedup
Original Version	5.141ms	1.00x	
Better Memory Accesses	1.572ms	3.27x	3.27x
L1 cache	1.454ms	3.54x	1.08x

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	Rel. Speedup
Original Version	5.141ms	1.00x	
Better Memory Accesses	1.572ms	3.27x	3.27x
(L1 cache)	(1.454ms)	(3.54x)	(1.08x)
Shared Memory	1.021ms	5.04x	1.54x

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 3

IDENTIFY HOTSPOT

Hotspot

Results

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_v2(int, int, unsigned char const *, unsigned char*)
44	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
18	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

- gaussian_filter_7x7_v0() still the hotspot

Kernel	Time	Speedup	Rel. Speedup
Original Version	5.141ms	1.00x	
Better Memory Accesses	1.572ms	3.27x	3.27x
(L1 cache)	*(1.454ms)*	*(3.54x)*	*(1.08x)*
Shared Memory	1.021ms	5.04x	1.54x

GPU TECHNOLOGY CONFERENCE
NVIDIA.

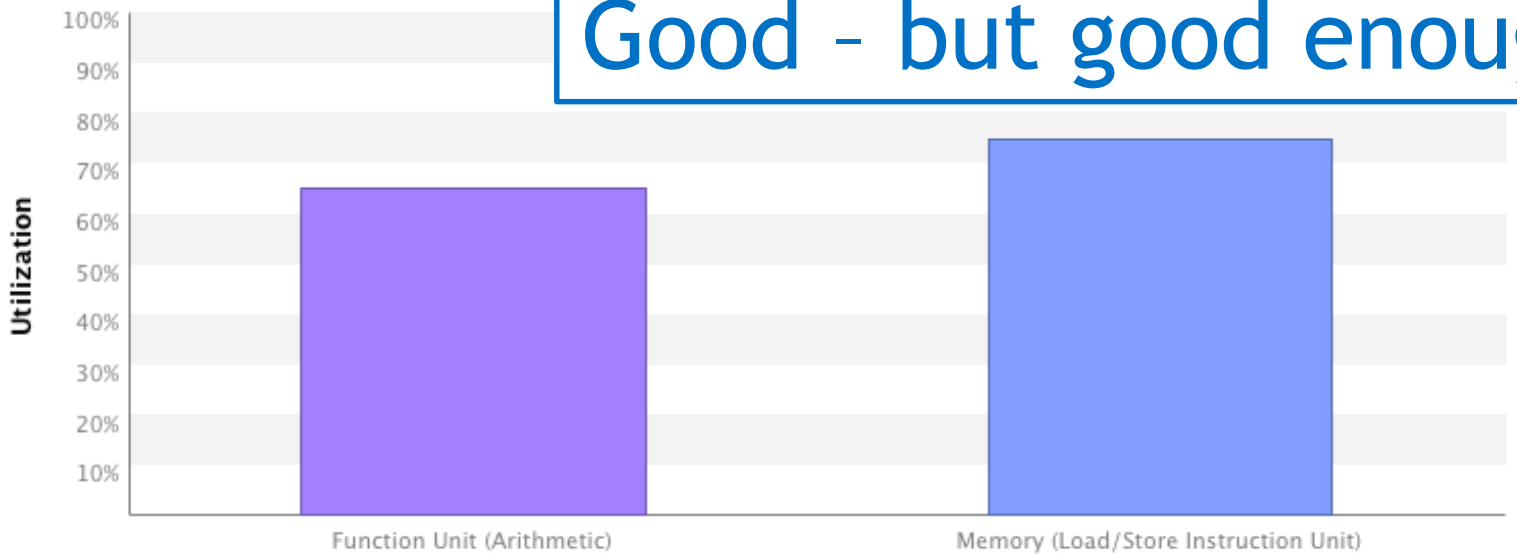
IDENTIFY PERFORMANCE LIMITER

Results

Kernel Performance Is Bound By Compute And Memory Bandwidth

For device "Tesla K80" compute and memory utilization are balanced. These utilization levels indicate that kernel performance is good, but that additional performance improvement may be possible if either of both of compute and memory utilization levels are increased.

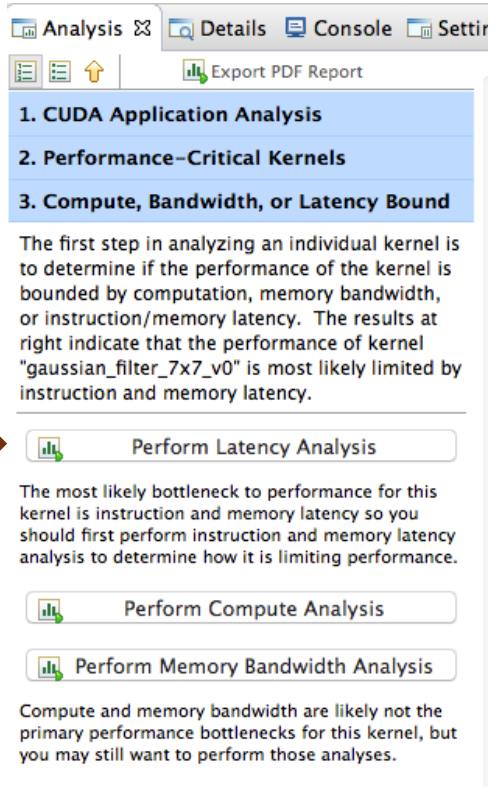
Good – but good enough?

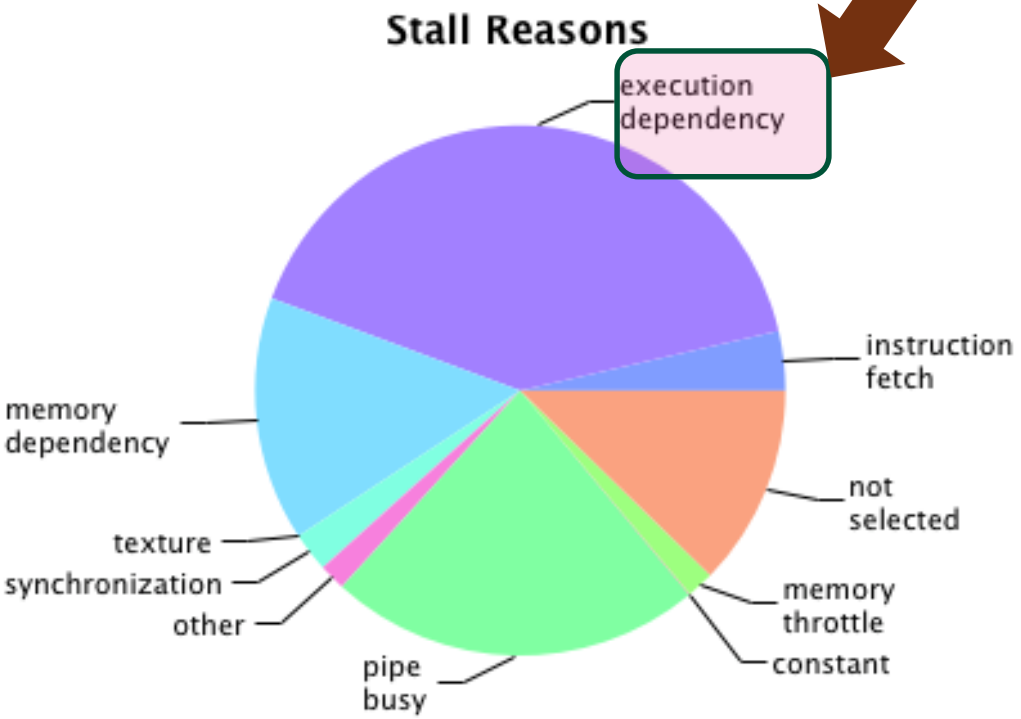


LOOKING FOR INDICATORS

A lot of idle time

Launch





- 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]);
```

```
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

NVIDIA.
GPU TECHNOLOGY CONFERENCE

LOOKING FOR MORE INDICATORS

Properties

gaussian_filter_7x7_v2(int, int, unsigned char const *, u...

Start	273.749 ms (...
End	274.77 ms (2...
Duration	1.021 ms (1,0...
Grid Size	[80,800,1]
Block Size	[32,2,1]
Registers/Thread	12
Shared Memory/Block	640 B
Occupancy	
Achieved	47.4%
Theoretical	50%
Limiter	Block Size

LOOKING FOR MORE INDICATORS

Analysis | Details | Console | Settings

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 indicates the kernel's actual occupancy. For this kernel, examining occupancy may not be useful since both achieved and theoretical occupancy are likely high enough to hide compute and memory latency.

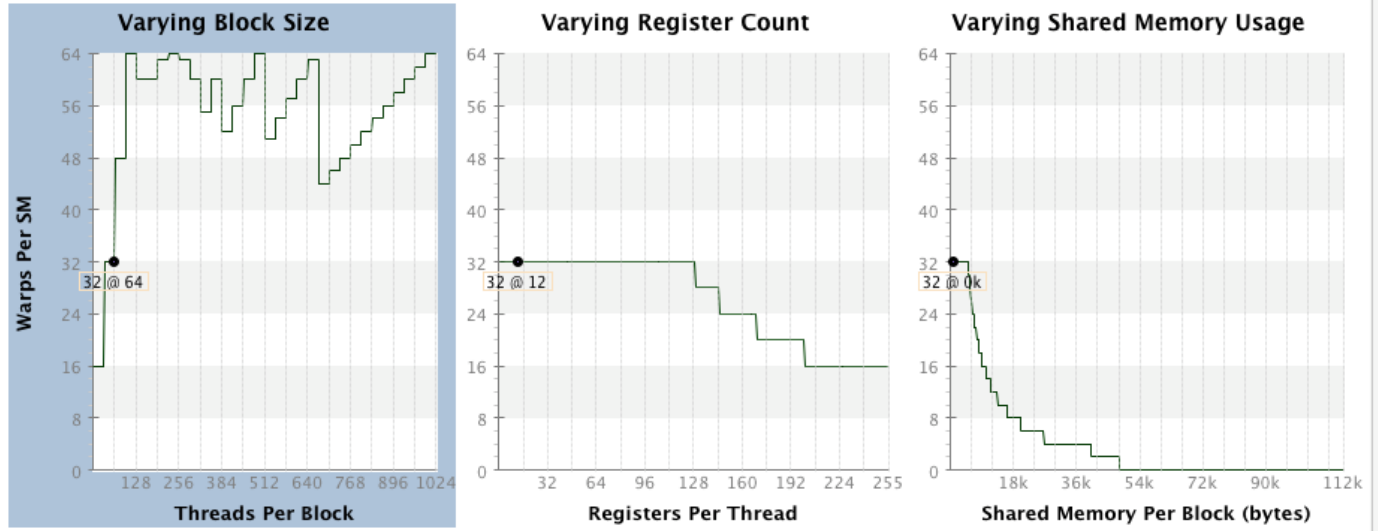
GPU Utilization May Be Limited By Block Size

Theoretical occupancy is less than 100% but is large enough that increasing occupancy may not improve performance. You can attempt the following optimization to increase the number of warps on each SM but it may not lead to increased performance.

The kernel has a block size of 64 threads. This block size is likely preventing the kernel from fully utilizing the GPU. Device "Tesla K80" can simultaneously execute up to 16 blocks on each SM. Because each block uses 2 warps to execute the block's 64 threads, the kernel is using only 32 warps on each SM. Chart "Varying Block Size" below shows how changing the block size will change the number of warps that can execute on each SM.

Optimization: Increase the number of threads in each block to increase the number of warps that can execute on each SM. More...

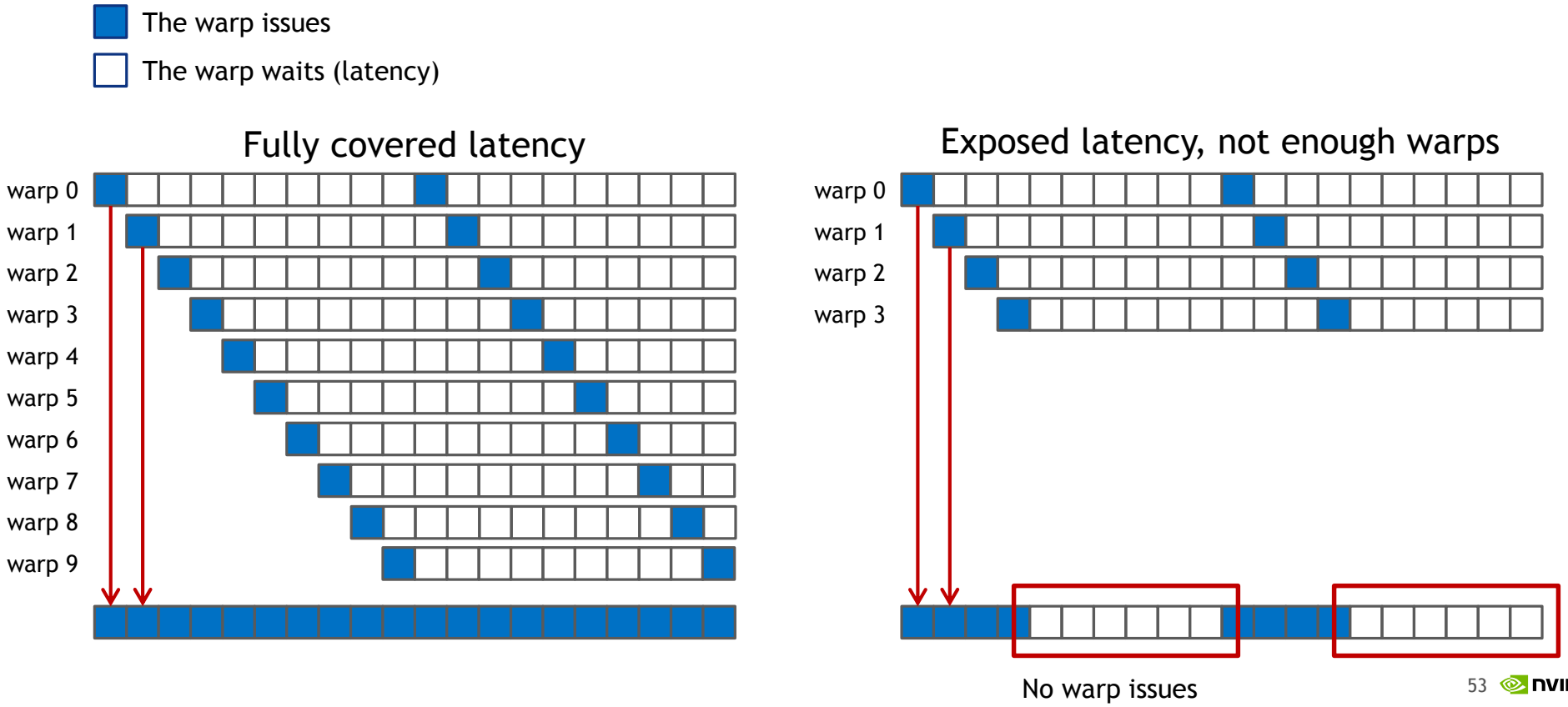
Variable	Achieved	Theoretical	Device Limit	Grid Size: [80,800,1] (64000 blocks)Block Size: [32,2,1] (64 thi
Occupancy Per SM				
Active Blocks		16	16	



▸ Not enough active warps to hide latencies?

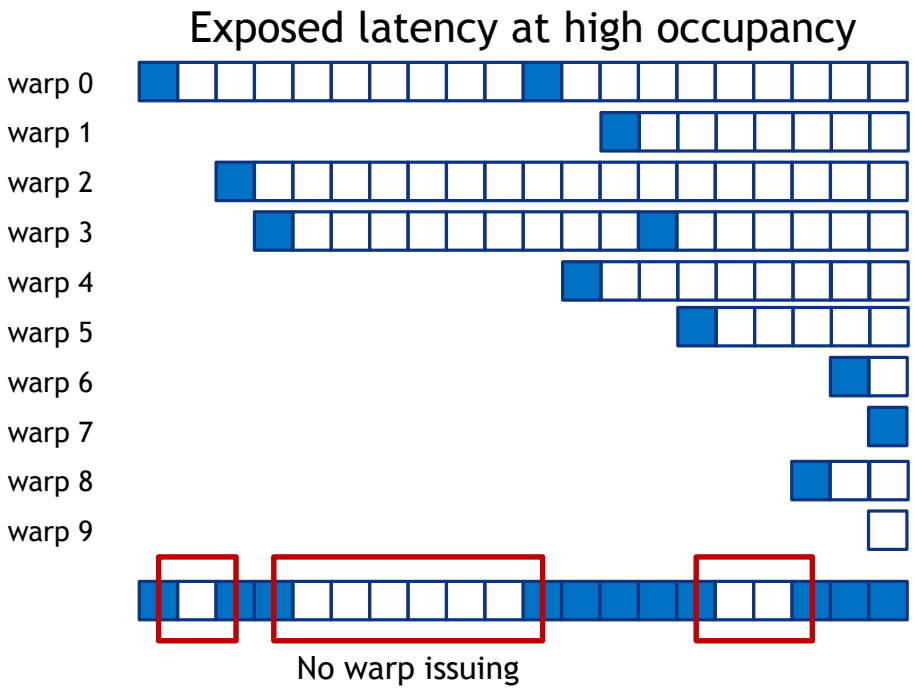
LATENCY

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



LATENCY AT HIGH OCCUPANCY

- Many active warps but with high latency instructions



- The schedulers cannot find eligible warps at every cycle

IMPROVED OCCUPANCY

- Bigger blocks of size 32x4
- Increase achieved occupancy (from 47.4% to 81.3%)
  - Theoretical occupancy from 50% to 100%

Kernel	Time	Speedup	Rel. Speedup
Original Version	5.141ms	1.00x	
Better Memory Accesses	1.572ms	3.27x	3.27x
Shared Memory	1.021ms	5.04x	1.54x
Occupancy	0.924ms	5.56x	1.17x

GPU TECHNOLOGY CONFERENCE
NVIDIA.

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 (use __launch_bounds)

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 4

IDENTIFY HOTSPOT

Hotspot

Results

Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved occupancy. Optimization of hig
(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*)
51	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
21	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

- gaussian_filter_7x7_v0() still the hotspot

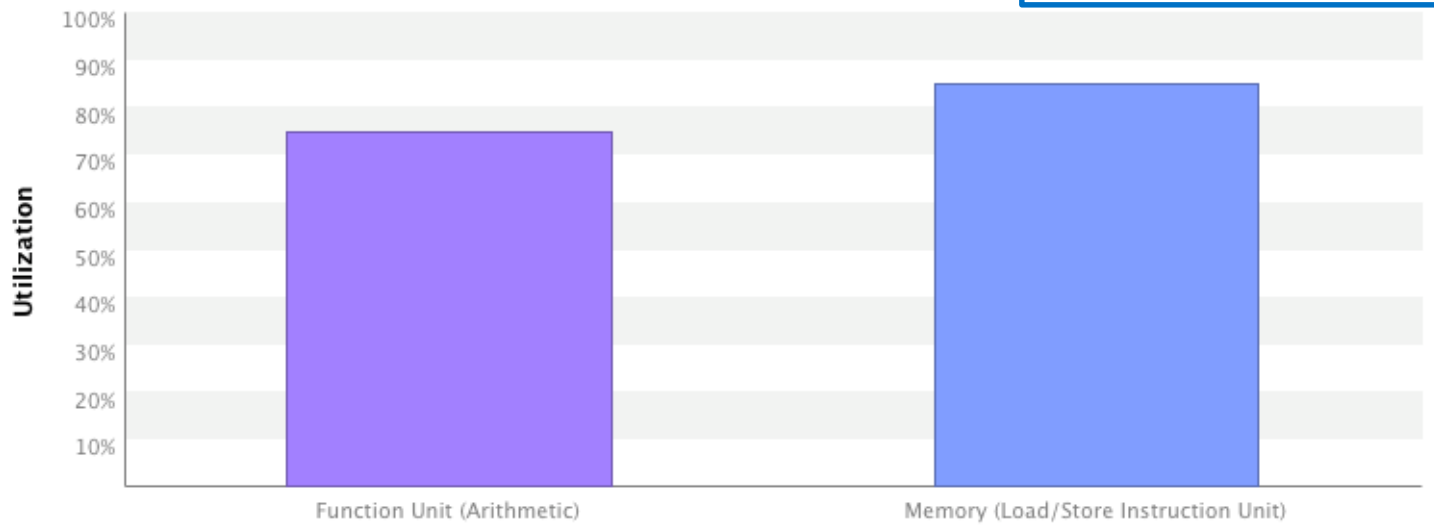
Kernel	Time	Speedup	Rel. Speedup
Original Version	5.141ms	1.00x	
Better Memory Accesses	1.572ms	3.27x	3.27x
Shared Memory	1.021ms	5.04x	1.54x
Higher Occupancy	0.924ms	5.56x	1.17x

IDENTIFY PERFORMANCE LIMITER

Results

Kernel Performance Is Bound By Compute And Memory Bandwidth

For device "Tesla K80" compute and memory utilization are balanced. These utilization levels indicate th
performance improvement may be possible if either of both of compute and memory utilization levels a



Aha!
Getting into the high utilization region

LOOKING FOR INDICATORS

Analysis | Details | Console | Setting

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_v2" 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 bottleneck for this kernel, but you may still want to perform that analysis.

Results

L1/Shared Memory

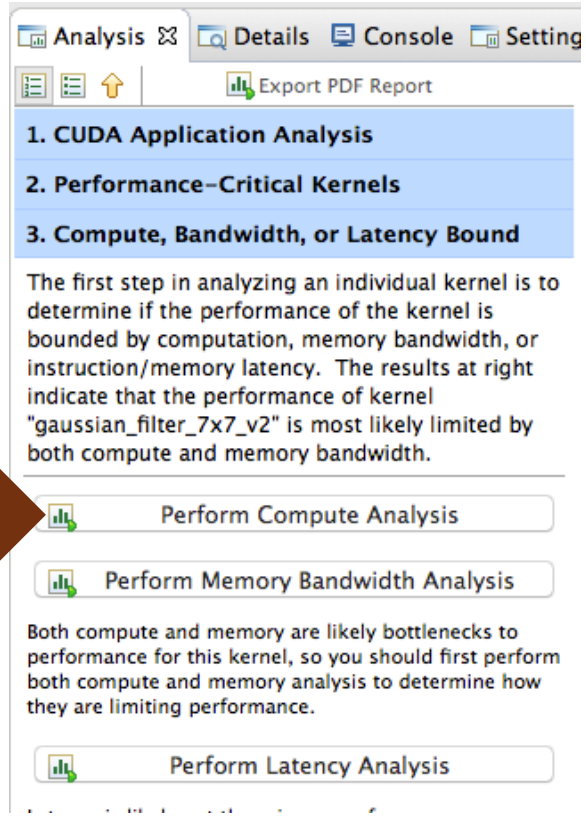
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	6272000	1,740.677 GB/s	
Shared Stores	638725	177.266 GB/s	
Global Loads	789822	44.006 GB/s	
Global Stores	128000	4.441 GB/s	
Atomic	0	0 B/s	
L1/Shared Total	7828547	1,966.39 GB/s	Idle Low Medium High Max

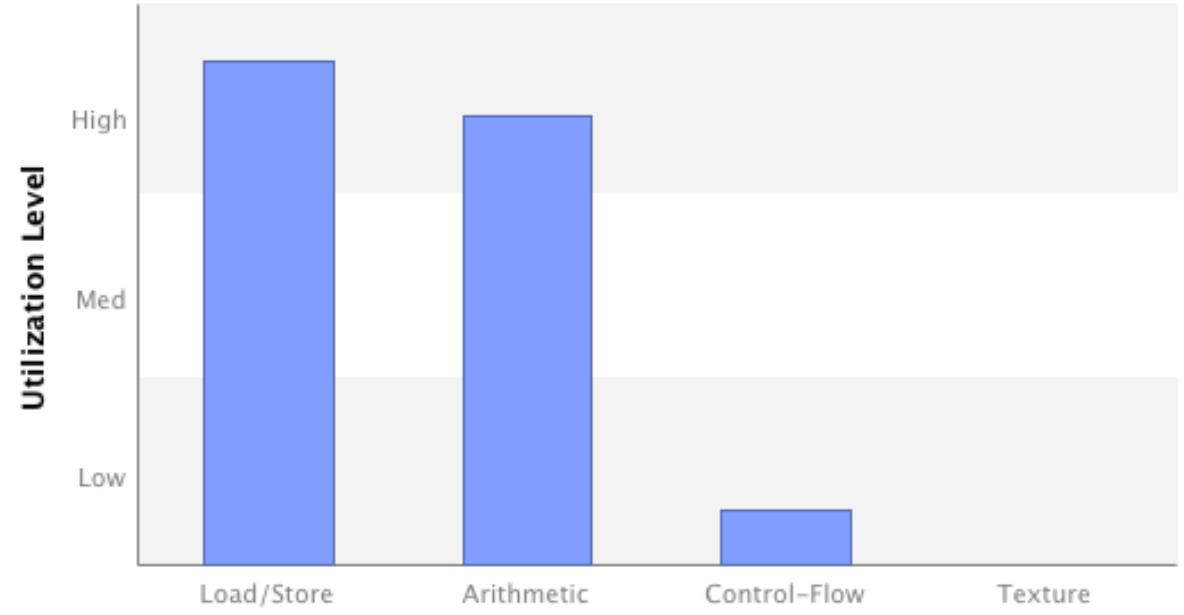
L2 Cache

L1 Reads	1268502	44.006 GB/s	
L1 Writes	128140	4.445 GB/s	
Texture Reads	0	0 B/s	
Noncoherent Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	1396642	48.452 GB/s	Idle Low Medium High Max

LOOKING FOR MORE INDICATORS

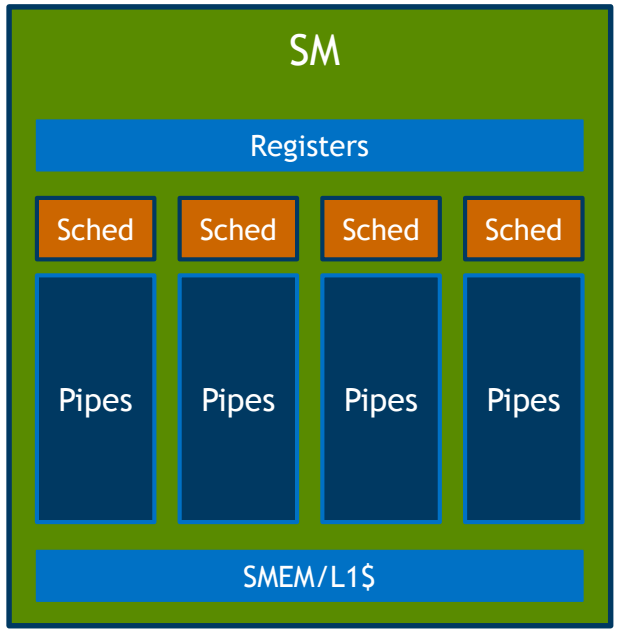
Launch





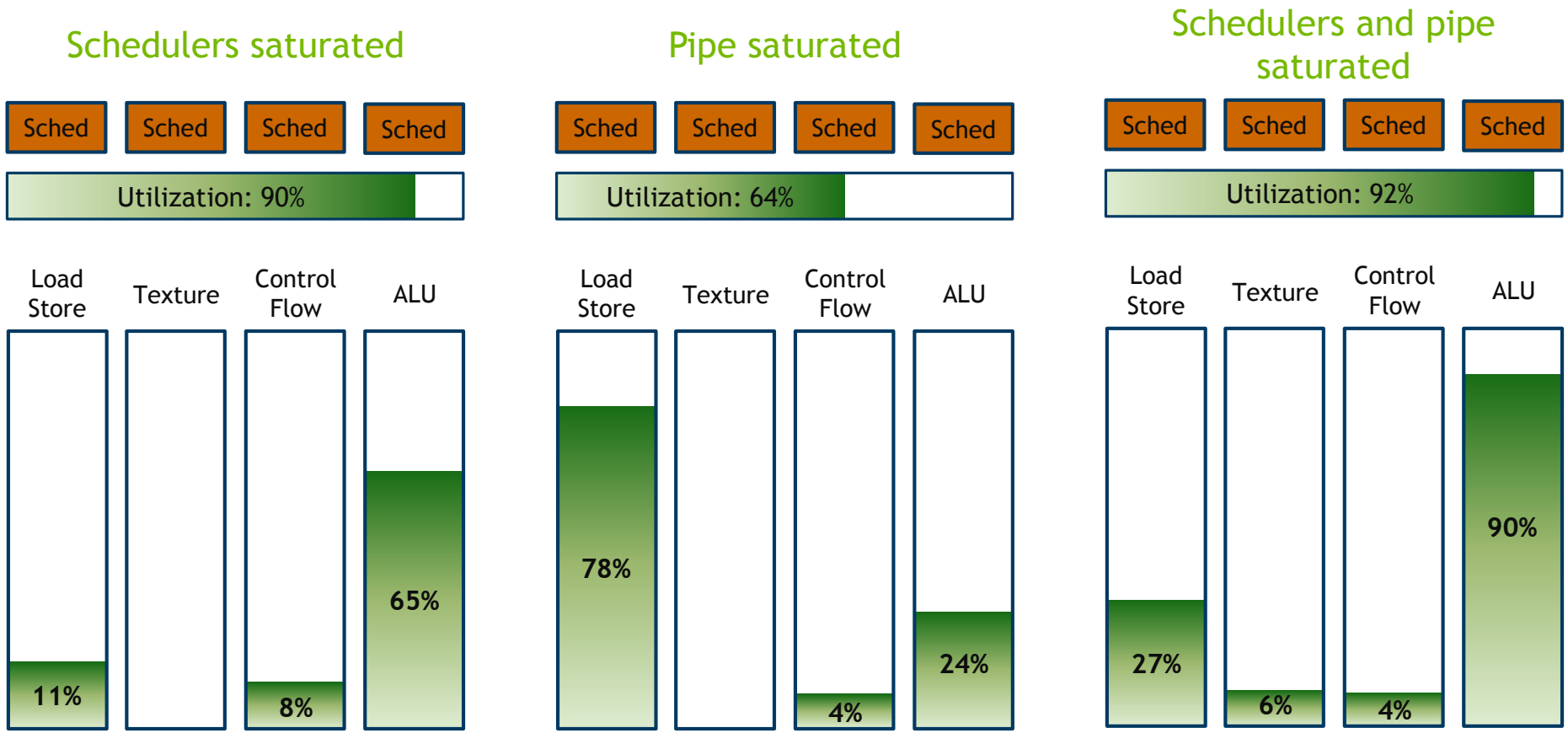
Load/Store Unit is really busy!
Can we reduce the load?

INSTRUCTION THROUGHPUT

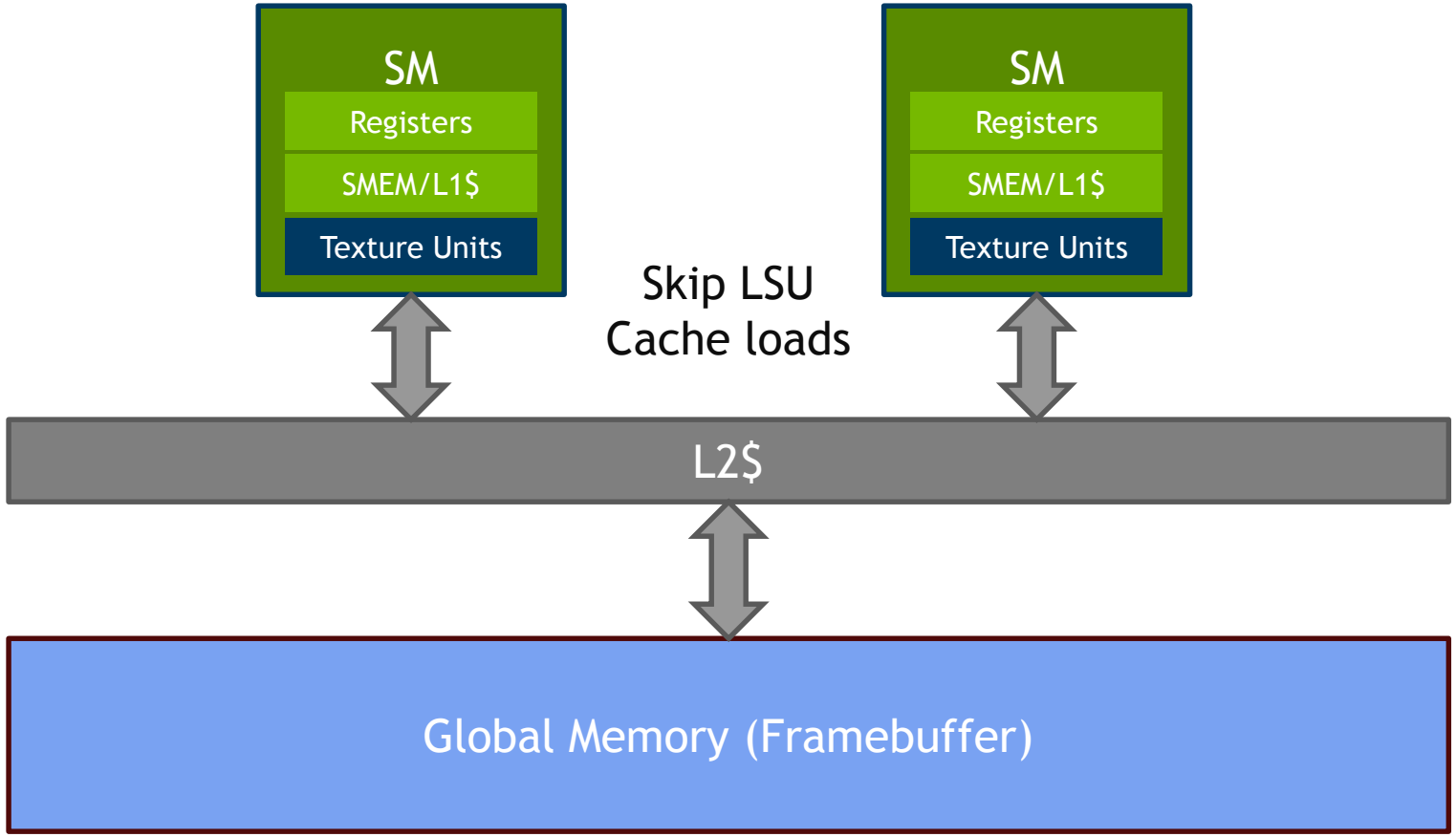


- Each SM has 4 schedulers (Kepler)
- Schedulers issue instructions to pipes
- A scheduler issues up to 2 instructions/cycle
- A scheduler issues inst. from a single warp
- Cannot issue to a pipe if its issue slot is full

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: 0.829ms

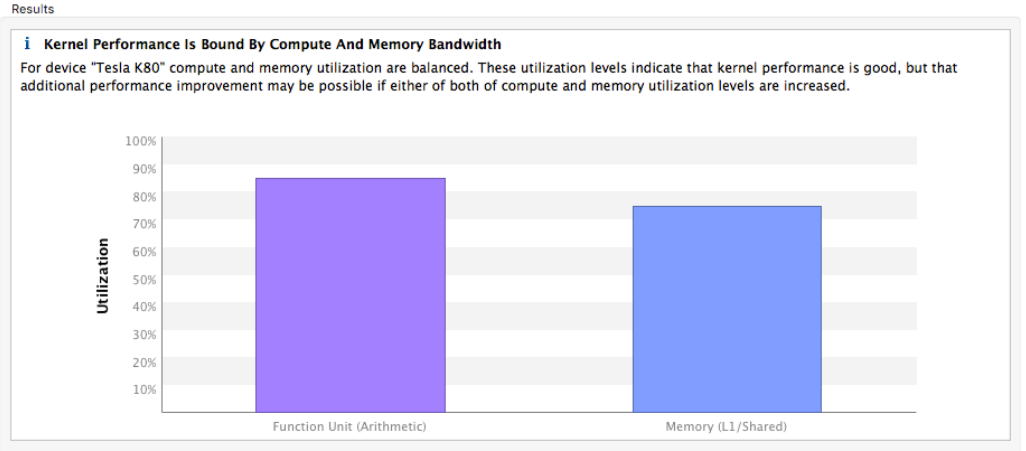
Kernel	Time	Speedup	Rel. Speedup
Original version	5.141ms	1.00x	
Better memory accesses	1.572ms	3.27x	3.27x
Shared memory	1.021ms	5.04x	1.54x
Higher Occupancy	0.924ms	5.56x	1.10x
Read-Only path	0.829ms	6.20x	1.11x

GPU TECHNOLOGY CONFERENCE
NVIDIA.

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: 6.2X



- Looking much better
- Things to investigate next
  - Function Unit is highly utilized
  - Reduce computational intensity (separable filter)
  - Increase Instruction Level Parallelism (process two elements per thread)
- Gaussian filer remains bottleneck

Results

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_v3(int, int, unsigned char const *, unsigned char*)
57	[1 kernel instances] sobel_filter_3x3_v0(int, int, unsigned char const *, unsigned char*)
23	[1 kernel instances] rgba_to_grayscale_kernel_v0(int, int, uchar4 const *, unsigned char*)

MORE IN OUR COMPANION CODE

Kernel	Time	Speedup	Rel. Speedup
Original version	5.141ms	1.00x	
Better memory accesses	1.572ms	3.27x	3.27x
Shared memory	1.021ms	5.04x	1.54x
Higher Occupancy	0.924ms	5.56x	1.10x
Read-Only path	**0.829ms**	**6.20x**	**1.11x**
6.20x speedup achieved with steps shown today			
Separable filter *(use a better algorithm)*	0.431ms	11.94x	1.93x
Process two pixels per thread (memory efficiency + ILP)	0.421ms	12.21x	1.02x
Use 64-bit shared memory (remove bank conflicts)	0.409ms	12.58x	1.03x
Use float instead of int (increase instruction throughput)	0.361ms	14.25x	1.13x
14.25x speedup in companion code (includes algorithmic improvements)			

Companion Code: https://github.com/chmaruni/nsight-gtc

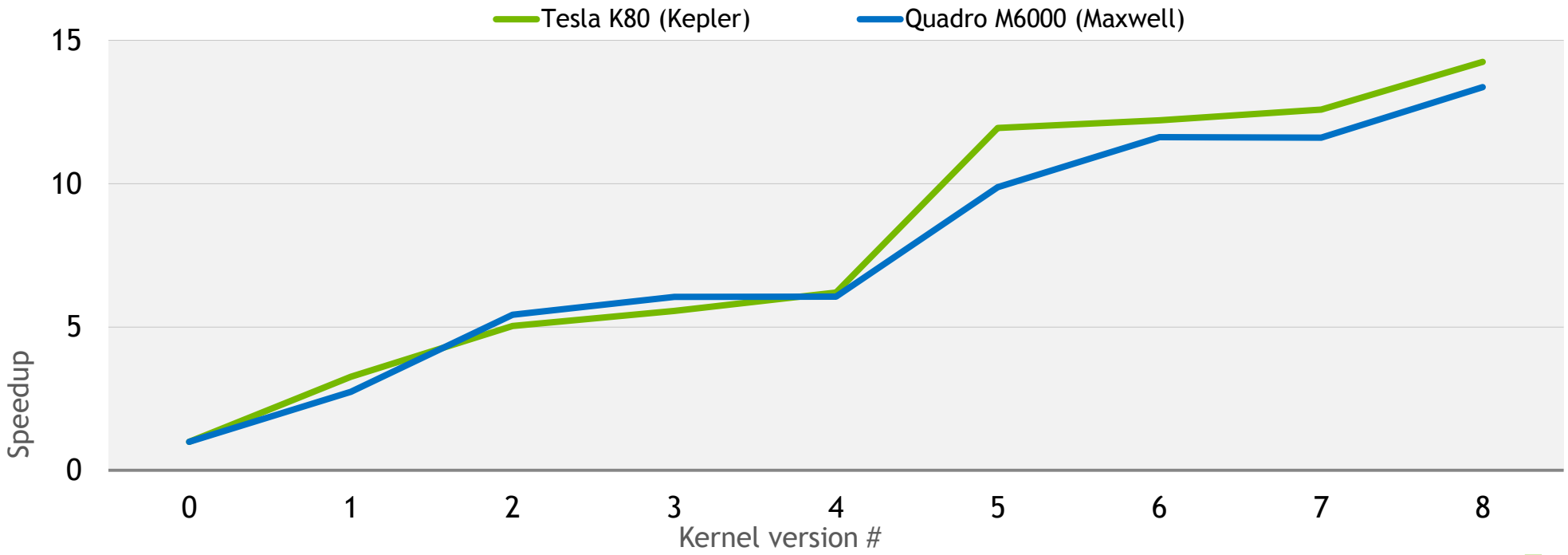
SUMMARY

ITERATIVE OPTIMIZATION WITH NSIGHT EE

- Assess the Performance
  - Identify the Hotspot
  - Classify the Performance Limiter
  - Look for indicators
- Parallelize the Application
- Optimize the Code
- Deploy and Test

ACROSS ARCHITECTURES

- Optimization concepts are general
  - Mileage of individual steps might vary



REFERENCES

- CUDA Documentation
  - Best Practices: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/
  - Kepler Tuning Guide: http://docs.nvidia.com/cuda/kepler-tuning-guide
  - Maxwell Tuning Guide: http://docs.nvidia.com/cuda/maxwell-tuning-guide
- Parallel Forall devblog
  - http://devblogs.nvidia.com/parallelforall/
- Upcoming GTC 2016 Sessions:
  - S6514 - CUDA Optimization Tips, Tricks and Techniques, Stephen Jones, SpaceX, Tuesday 13:00-13:50
  - S6810 - Optimizing Application Performance with CUDA® Profiling Tools, Swapna Matwankar, NVIDIA, Thursday 10:00-10:50
  - L6126 - Tips and Tricks for Unified Memory on NVIDIA Kepler and Maxwell Architectures, Nikolay Sakharnykh and Jiri Kraus, NVIDIA, Wednesday 09:30-11:00

GPU TECHNOLOGY CONFERENCE

April 4-7, 2016 | Silicon Valley

THANK YOU

JOIN THE CONVERSATION

#GTC16

JOIN THE NVIDIA DEVELOPER PROGRAM AT developer.nvidia.com/join

PRESENTED BY

NVIDIA.