CUDA OPTIMIZATION WITH NVIDIA NSIGHT™ VISUAL STUDIO EDITION

CHRISTOPH ANGERER, NVIDIA
JAKOB PROGSCH, NVIDIA
WHAT YOU WILL LEARN

- An iterative method to optimize your GPU code
- A way to conduct that method with NVIDIA Nsight VSE
- Companion Code: https://github.com/chmaruni/nsight-gtc
INTRODUCING THE APPLICATION

Grayscale

Blur

Edges
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`:

\[ p = \text{weighted sum of } p \text{ and its 48 neighbors} \]
Edges: 3x3 Sobel Filters

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
-1 -2 -1
ENVIRONMENT

NVIDIA GTX 980 Ti
  ▶ GM200
  ▶ SM5.2 (Compute Capability)
CUDA 7.5
Windows 7
NVIDIA Nsight Visual Studio Edition 5.0
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
ITERATION 1
DEMO: THE NSIGHT VISUAL STUDIO EDITION GUI
THE CUDA LAUNCHES VIEW

Select Kernel

Select Experiment

Accesses go directly to (slow, uncached) system memory instead of device memory!
THE PROBLEM

- The machine has a GTX 980 Ti for compute and a GTX 960 for the display
- If multiple GPUs are present and do not support peer access CUDA defaults to using zero copy memory for UVM

Solution:
- Select the one device to be used by cuda by setting the CUDA_VISIBLE_DEVICES env var.
- Restart the nsight monitor to make it pick up on the env var.
THE CUDA LAUNCHES VIEW

| Function Name               | Grid Dimensions | Block Dimensions | Start Time (ps) | Duration (ps) | Occupancy | Registers per Thread | Static Shared Memory per Block (bytes) | Dynamic Shared Memory per Block (bytes) | Cache Configuration Executed | Global Caching Requested | Global Caching Executed | Local Memory per Thread (bytes) | Device Name
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rgba_tr_grayscale_kernel_v0</td>
<td>(80, 200, 1)</td>
<td>(32, 8, 1)</td>
<td>737761.647</td>
<td>101.016</td>
<td>100.00%</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>PREFER_SHARED</td>
<td>OFF</td>
<td>OFF</td>
<td>0 GeForce GTX 980 Ti</td>
<td></td>
</tr>
<tr>
<td>gaussian_filter_7x7_v0</td>
<td>(128, 200, 1)</td>
<td>(16, 8, 1)</td>
<td>208517.371</td>
<td>2999.900</td>
<td>56.25%</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>PREFER_SHARED</td>
<td>OFF</td>
<td>OFF</td>
<td>0 GeForce GTX 980 Ti</td>
<td></td>
</tr>
<tr>
<td>sobel_filter_7x7_v0</td>
<td>(80, 200, 1)</td>
<td>(16, 8, 1)</td>
<td>634393.159</td>
<td>1244.108</td>
<td>100.00%</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>PREFER_SHARED</td>
<td>OFF</td>
<td>OFF</td>
<td>0 GeForce GTX 980 Ti</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**
- **Bandwidth**:
  - Occupancy: 0.00 B/s
  - 1.05 GB/s
  - Texture: 445.57 GB/s
  - Global: 0.00 B/s
  - Local: 0.00 B/s
  - Global Atomics: 0.00 B/s
  - Shared Atomics: 0.00 B/s
  - Shared: 0.00 B/s
  - L1/Cache: 445.57 GB/s
  - L2 Cache: 96.3%
  - Device Memory
  - System Memory

**Statistics:**
- Texture:
  - Fetches: 0.00
  - Loads: 0.00
  - Stores: 0.00
- Global:
  - Reads: 6400.000
  - Writes: 0.000
  - Loads: 6272.000
  - Stores: 128.000
- Local:
  - Reads: 31394.050
  - Writes: 245.27
  - Loads: 12023.840
  - Stores: 1.0231920
- Shared:
  - Reads: 9262.82
  - Writes: 674.81
  - Loads: 31.25
  - Stores: 256.00

**Memory Usage:**
- 3.72 GB
- 445.57 GB

**Device:**
- GeForce GTX 980 Ti
STATUS ITERATION 1

- Baseline using UVM

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>UVM from Zero Copy Mem</td>
<td>454.2ms</td>
<td>0.0045x</td>
</tr>
<tr>
<td>UVM Version (Baseline)</td>
<td>2.065ms</td>
<td>1.00x</td>
</tr>
</tbody>
</table>
ITERATION 2
### IDENTIFY HOTSPOT (CUDA SUMMARY)

#### Top Device Functions By Total Time

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Launches</th>
<th>Device %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>gaussian_filter_7x7_v0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>sobel_filter_3x3_v0</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>3</td>
<td>rgba_to_grayscale_kernel_v0</td>
<td>1</td>
<td>0.00</td>
</tr>
</tbody>
</table>

- **Identify the hotspot:** `gaussian_filter_7x7_v0()`

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>2.065ms</td>
<td>1.00x</td>
</tr>
</tbody>
</table>
IDENTIFY MAIN PERFORMANCE LIMITER

Memory Utilization vs Compute Utilization

Four possible combinations:

- **Compute Bound**
  - Compute: High
  - Memory: Low

- **Bandwidth Bound**
  - Compute: Low
  - Memory: High

- **Latency Bound**
  - Compute: Low
  - Memory: Low

- **Compute and Bandwidth Bound**
  - Compute: High
  - Memory: High

60%
MEMORY BANDWIDTH

Global Memory (Framebuffer)

L2$

SM

Registers
SMEM/L1$

SM

Registers
SMEM/L1$

SMEM/L1$
Utilization of L2 Bandwidth (BW) limited and DRAM BW < 2%
Not limited by memory bandwidth
Each SM has 4 schedulers (Maxwell)

Schedulers issue instructions to pipes

Each scheduler schedules up to 2 instructions per cycle

A scheduler issues inst. from a single warp

Cannot issue to a pipe if its issue slot is full
INSTRUCTION THROUGHPUT

Schedulers saturated

- Utilization: 90%

Pipe saturated

- Utilization: 64%

Schedulers and pipe saturated

- Utilization: 92%
WARP ISSUE EFFICIENCY

Warp Issue Efficiency

- No Eligible
- One or More Eligible

Cycles 206,847,852

26.73% 73.27%

Percentage of issue slots used (blue)
Aggregated over all the schedulers
PIPE UTILIZATION

Percentages of issue slots used per pipe

Accounts for pipe throughputs

Four groups of pipes:
- Shared Memory
- Texture
- Control Flow
- Arithmetic (ALU)
Neither schedulers nor pipes are saturated

Not limited by the instruction throughput

⇒ Our Kernel is Latency Bound
LOOKING FOR INDICATORS

- 56% theoretical occupancy (36 out of 64 warps)
- 45.8% achieved occupancy (29.31 active warps per cycle)
- 1.25 warps eligible per cycle
- Let’s start with occupancy
Each SM has limited resources

64K Registers (32 bit) distributed between threads

Up to 48KB of shared memory per block (96KB per SMM)

32 Active Blocks per SMM

Full occupancy: 2048 threads per SM (64 warps)

Values vary with Compute Capability
GPUs cover latencies by having a lot of work in flight.

- **Fully covered latency**
  - Warp 0:
  - Warp 1:
  - Warp 2:
  - Warp 3:
  - Warp 4:
  - Warp 5:
  - Warp 6:
  - Warp 7:
  - Warp 8:
  - Warp 9:

- **Exposed latency, not enough warps**
  - Warp 0:
  - Warp 1:
  - Warp 2:
  - Warp 3:

No warp issues
LATENCY AT HIGH OCCUPANCY

Many active warps but with high latency instructions

The schedulers cannot find eligible warps at every cycle
LOOKING FOR MORE INDICATORS

Block Size seems OK

We don’t want to change the register count yet
STALL REASONS

- Stall reasons show the source of the latencies we can not cover
STALL REASONS: EXECUTION/MEMORY DEPENDENCY

Instruction level parallelism can reduce dependencies

a = b + c;  // ADD

\[
\text{d} = a + e; \quad // \text{ADD}
\]

a = b[i];  // LOAD

\[
\text{d} = a + e; \quad // \text{ADD}
\]

a = b + c;  // Independent ADDs

\[
\text{d} = e + f;
\]
ILP AND MEMORY ACCESSES

No ILP

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

// Remaining code...
```

2-way ILP (with loop unrolling)

```c
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

Sadly not directly applicable to our current issue
CONTINUE LOOKING FOR INDICATORS

4-8 L2 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 128B load/store request per warp

4x 32B L2 transactions per warp

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

Stride: 32x4B

1x 128B load/store request per warp

1x 32B L2 transaction per thread

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

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
<th>Rel. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>2.065ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>0.799ms</td>
<td>2.58x</td>
<td>2.58x</td>
</tr>
<tr>
<td>Category:</td>
<td>Latency Bound - Occupancy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem:</td>
<td>Latency is exposed due to low occupancy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal:</td>
<td>Hide latency behind more parallel work</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indicators:</td>
<td>Occupancy low (&lt; 60%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Execution Dependency High</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy:</td>
<td>Increase occupancy by:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Varying block size</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Varying shared memory usage</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Varying register count (use __launch_bounds)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category:</td>
<td>Latency Bound - Instruction Level Parallelism</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem:</td>
<td>Not enough independent work per thread</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal:</td>
<td>Do more parallel work inside single threads</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indicators:</td>
<td>High execution dependency, increasing occupancy has no/little positive effect, still registers available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy:</td>
<td>• Unroll loops (#pragma unroll)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Refactor threads to compute n output values at the same time (code duplication)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category:</td>
<td>Latency Bound - Coalescing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem:</td>
<td>Memory is accessed inefficiently =&gt; high latency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal:</td>
<td>Reduce #transactions/request to reduce latency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indicators:</td>
<td>Low global load/store efficiency, High #transactions/#request compared to ideal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy:</td>
<td>Improve memory coalescing by:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Cooperative loading inside a block</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Change block layout</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Aligning data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Changing data layout to improve locality</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 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
IDENTIFY HOTSPOT

- gaussian_filter_7x7_v0() still the hotspot

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
<th>Rel. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>2.065ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>0.766ms</td>
<td>2.70x</td>
<td>2.70x</td>
</tr>
</tbody>
</table>
IDENTIFY PERFORMANCE LIMITER

- Utilization of L2$ Bandwidth (BW) limited and DRAM BW < 4%
- Not limited by memory bandwidth
Scheduler is starting to be busy
but Tex pipe is clearly the limiter

⇒ Texture pipeline getting busy
LOOKING FOR MORE INDICATORS

- Kernel Transfers 8MB to/from Device Memory but 360MB to/from L2 Cache

Can we move the data closer to the SM?
SHARED MEMORY

Adjacent pixels access similar neighbors in Gaussian Filter

We should use shared memory to store those common pixels

```c
__shared__ unsigned char smem_pixels[10][64];
```
Using shared memory for the Gaussian Filter

Significant speedup, < 0.5ms

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
<th>Rel. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>2.065ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>0.766ms</td>
<td>2.70x</td>
<td>2.70x</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>0.370ms</td>
<td>5.58x</td>
<td>2.07x</td>
</tr>
</tbody>
</table>
### Category: Latency Bound - Shared Memory

<table>
<thead>
<tr>
<th>Problem:</th>
<th>Long memory latencies are harder to hide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal:</td>
<td><strong>Reduce</strong> latency, move data to faster memory</td>
</tr>
<tr>
<td>Indicators:</td>
<td>Shared memory not occupancy limiter High L2 hit rate Data reuse between threads and small-ish working set</td>
</tr>
</tbody>
</table>
| Strategy:      | (Cooperatively) move data to:  
- Shared Memory  
- (or Registers)  
- (or Constant Memory)  
- (or Texture Cache) |
<table>
<thead>
<tr>
<th>Category:</th>
<th>Memory Bound - Shared Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem:</td>
<td>Too much data movement</td>
</tr>
<tr>
<td>Goal:</td>
<td>Reduce amount of data traffic to/from global mem</td>
</tr>
</tbody>
</table>
| 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
gaussian_filter_7x7_v0() still the hotspot

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Grid Dimensions</th>
<th>Block Dimensions</th>
<th>Start Time (µs)</th>
<th>Duration (µs)</th>
<th>Occupancy</th>
<th>Registers per Thread</th>
<th>Static Shared Memory per Block (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgb_to_grayscale_kernel_v0</td>
<td>(80, 200, 1)</td>
<td>(32, 8, 1)</td>
<td>682,572.777</td>
<td>100.800</td>
<td>100.00 %</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>gaussian_filter_7x7_v2</td>
<td>(80, 400, 1)</td>
<td>(32, 4, 1)</td>
<td>1,455,386.249</td>
<td>369.697</td>
<td>100.00 %</td>
<td>14</td>
<td>640</td>
</tr>
<tr>
<td>sobel_filter_3x3_v0</td>
<td>(80, 200, 1)</td>
<td>(32, 8, 1)</td>
<td>2,022,707.113</td>
<td>225.888</td>
<td>100.00 %</td>
<td>18</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
<th>Rel. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>2.065ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>0.766ms</td>
<td>2.70x</td>
<td>2.70x</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>0.370ms</td>
<td>5.58x</td>
<td>2.07x</td>
</tr>
</tbody>
</table>
IDENTIFY PERFORMANCE LIMITER

- Utilization of L2$ Bandwidth (BW) moderate and DRAM BW < 8%
- Not limited by memory bandwidth
IDENTIFY PERFORMANCE LIMITER

The Kernel is Compute Bound
LOOKING FOR INDICATORS

- No Divergence in our code

Branch Condition

- Not Taken: 55.55%
- Diverged: 44.45%
- Taken: 575,360
BRANCH DIVERGENCE

Threads of a warp take different branches of a conditional

```c
if( threadIdx.x < 12 ) {}
```

```
else {}
```

Execution time = “if” branch + “else” branch

<table>
<thead>
<tr>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads execute the “if” branch</td>
</tr>
</tbody>
</table>
LOOKING FOR MORE INDICATORS

- Execution dependency is largest (non other) block
- Not a clear indicator however
- Warp issue efficiency is already high, so there is little to be gained from reducing the remaining stalls
LOOKING FOR MORE INDICATORS

- >4TIOP/second

⇒ The Kernel is simply computing a lot
REDUCING COMPUTATIONAL COMPLEXITY

- Separable Filter:
  - Gaussian filters are circular and separable
  - Compute horizontal and vertical convolution separately
  - Gaussian approximated by binomial coefficients in our code

\[
\begin{bmatrix}
1 & 6 & 15 & 20 & 15 & 6 & 1 \\
6 & 36 & 90 & 120 & 90 & 36 & 6 \\
15 & 90 & 225 & 300 & 225 & 90 & 15 \\
20 & 120 & 300 & 400 & 300 & 120 & 20 \\
15 & 90 & 225 & 300 & 225 & 90 & 15 \\
6 & 36 & 90 & 120 & 90 & 36 & 6 \\
1 & 6 & 15 & 20 & 15 & 6 & 1
\end{bmatrix}
\times
\begin{bmatrix}
1 \\
6 \\
15 \\
20 \\
15 \\
6 \\
1
\end{bmatrix}
=
\begin{bmatrix}
1 & 6 & 15 & 20 & 15 & 6 & 1 \\
6 & 36 & 90 & 120 & 90 & 36 & 6 \\
15 & 90 & 225 & 300 & 225 & 90 & 15 \\
20 & 120 & 300 & 400 & 300 & 120 & 20 \\
15 & 90 & 225 & 300 & 225 & 90 & 15 \\
6 & 36 & 90 & 120 & 90 & 36 & 6 \\
1 & 6 & 15 & 20 & 15 & 6 & 1
\end{bmatrix}
\]
SEPARABLE FILTER + INCREASED ILP

- Separable filter reduces computational load
- Processing two elements per thread increases instruction level parallelism

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
<th>Rel. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>2.065ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>0.766ms</td>
<td>2.70x</td>
<td>2.70x</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>0.370ms</td>
<td>5.58x</td>
<td>2.07x</td>
</tr>
<tr>
<td>Separable Filter + incr. ILP</td>
<td>0.217ms</td>
<td>9.52x</td>
<td>1.71x</td>
</tr>
</tbody>
</table>
## Category: Compute Bound - Branch Divergence

<table>
<thead>
<tr>
<th>Problem:</th>
<th>Diverging threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal:</td>
<td>Reduce divergence <strong>within</strong> warps</td>
</tr>
<tr>
<td>Indicators:</td>
<td>Low warp execution efficiency, high control flow utilization</td>
</tr>
</tbody>
</table>
| Strategy: | • Refactor code to avoid intra-warp divergence  
        • Restructure data (sorting?) to avoid data-dependent branch divergence |
## Compute Bound - Algorithmic Changes

<table>
<thead>
<tr>
<th>Category:</th>
<th>Compute Bound - Algorithmic Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem:</td>
<td>GPU is computing as fast as possible</td>
</tr>
<tr>
<td>Goal:</td>
<td>Reduce computation if possible</td>
</tr>
<tr>
<td>Indicators:</td>
<td>Clearly compute bound problem, speedup only with less computation</td>
</tr>
</tbody>
</table>
| Strategy:        | • Pre-compute or store (intermediate) results  
                    • Trade memory for compute time  
                    • Use a computationally less expensive algorithm |
THE RESULT: 9.52X

Much better utilization

The sobel filter is starting to become the bottleneck
### More in Our Companion Code

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
<th>Rel. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original version</td>
<td>2.065ms</td>
<td>1.00x</td>
<td>1.00x</td>
</tr>
<tr>
<td>Better memory accesses</td>
<td>0.799ms</td>
<td>2.58x</td>
<td>2.58x</td>
</tr>
<tr>
<td>Shared memory</td>
<td>0.370ms</td>
<td>5.58x</td>
<td>2.07x</td>
</tr>
<tr>
<td>Separable Filter + incr. ILP</td>
<td>0.217ms</td>
<td>9.52x</td>
<td>1.71x</td>
</tr>
</tbody>
</table>

**9.52x speedup achieved with steps shown today**

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
<th>Rel. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floats+Intrinsics+fast_math</td>
<td>0.169ms</td>
<td>12.21x</td>
<td>1.28x</td>
</tr>
<tr>
<td>Floats instead of int ops</td>
<td>0.165ms</td>
<td>12.51x</td>
<td>1.02x</td>
</tr>
</tbody>
</table>

**12.51x speedup in companion code**

Companion Code: [https://github.com/chmaruni/nsight-gtc](https://github.com/chmaruni/nsight-gtc)
SUMMARY
ITERATIVE OPTIMIZATION WITH NSIGHT VSE

- 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

![Graph showing speedup across kernel versions for Tesla K80 (Kepler) and Quadro M6000 (Maxwell).]
REFERENCES

CUDA Documentation


Parallel Forall devblog


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
THANK YOU

JOIN THE CONVERSATION

#GTC16  

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