CUDA OPTIMIZATION WITH NVIDIA NSIGHT™ ECLIPSE EDITION

CHRISTOPH ANGERER, NVIDIA
MATHIAS WAGNER, 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
Grayscale Conversion

// r, g, b: Red, green, blue components of the pixel p
foreach pixel p:
    p = 0.298839fr + 0.586811fg + 0.114350fb;
INTRODUCING THE APPLICATION

- **Blur: 7x7 Gaussian Filter**

  ```plaintext
  foreach pixel p:
  p = weighted sum of p and its 48 neighbors
  ```
**Edges: 3x3 Sobel Filters**

```plaintext
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

Create New Session

Executable Properties
Set executable properties

Connection:
- Local
- Add connection...

Toolkit:
- [Available tools]

File:
Enter executable file [required]

Working directory:
Enter working directory [optional]

Arguments:
Enter command-line arguments
Profile child processes

Environment:
Name Value

New Remote Connection

Remote Connections
Manage available connections

user@gpunode

Host name: gpunode
User name: user
Label: user@gpunode
System type: SSH
Port number: 22

THE PROFILER WINDOW

Timeline
Summary
Guide
Analysis Results
ITERATION 1
EXAMINE INDIVIDUAL KERNELS
(GUIDED ANALYSIS)

1. CUDA Application Analysis

The guided analysis system walks you through the various analysis stages to help you understand the optimization opportunities in your application. Once you become familiar with the optimization process, you can explore the individual analysis stages in an unguided mode. When optimizing your application it is important to fully utilize the compute and data movement capabilities of the GPU. To do this you should look at your application’s overall GPU usage as well as the performance of individual kernels.

Examine GPU Usage

Determine your application’s overall GPU usage. This analysis requires an application timeline, so your application will be run once to collect it if it is not already available.

Examine Individual Kernels

Determine which kernels are the most performance critical and that have the most opportunity for improvement. This analysis requires utilization data from every kernel, so your application will be run once to collect that data if it is not already available.

Delete Existing Analysis Information

If the application has changed since the last analysis then the existing analysis
### 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>5.141ms</td>
<td>1.00x</td>
</tr>
</tbody>
</table>
PERFORM KERNEL ANALYSIS

Launch

Select
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.
Memory Utilization vs Compute Utilization
Four possible combinations:

- Compute Bound
- Bandwidth Bound
- Latency Bound
- Compute and Bandwidth Bound
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.

- Memory Ops
- Load/Store
- Memory Related Issues?
LOOKING FOR INDICATORS

Large number of memory operations stalling LSU
LOOKING FOR MORE INDICATORS

Unguided Analysis

For line numbers use: nvcc -lineinfo

4-5 Global Load/Store Transactions per 1 Request
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
- 1x 128B L1 transaction per warp
- 4x 32B L2 transactions per warp

- 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

**Stride: 32x4B**

- 1x 128B load/store request per warp
- 1x 128B L1 transaction per thread
- 1x 32B L2 transaction per thread

- 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

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>5.141ms</td>
<td>1.00x</td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>1.572ms</td>
<td>3.27x</td>
</tr>
</tbody>
</table>
### 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**

<table>
<thead>
<tr>
<th>Category:</th>
<th>Bandwidth Bound - Coalescing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem:</td>
<td>Too much unused data clogging memory system</td>
</tr>
<tr>
<td>Goal:</td>
<td>Reduce traffic, move more <em>useful</em> data per request</td>
</tr>
<tr>
<td>Indicators:</td>
<td>Low global load/store efficiency,</td>
</tr>
<tr>
<td></td>
<td>High #transactions/#request compared to ideal</td>
</tr>
<tr>
<td>Strategy:</td>
<td>Improve memory coalescing by:</td>
</tr>
<tr>
<td></td>
<td>• Cooperative loading inside a block</td>
</tr>
<tr>
<td></td>
<td>• Change block layout</td>
</tr>
<tr>
<td></td>
<td>• Aligning data</td>
</tr>
<tr>
<td></td>
<td>• Changing data layout to improve locality</td>
</tr>
</tbody>
</table>
ITERATION 2
IDENTIFY HOTSPOT

- `gaussian_filter_7x7_v0()` still the hotspot

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Version</td>
<td>5.141ms</td>
<td>1.00x</td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>1.5724ms</td>
<td>3.27x</td>
</tr>
</tbody>
</table>
IDENTIFY PERFORMANCE LIMITER

Results

1. 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 50% of peak typically indicates latency issues.

Still Latency Bound
Looking for more indicators:

Is our working set mostly in L2?

Medium L2 Bandwidth Utilization

Very low device memory bandwidth utilization

Launch
CHECKING L2 HIT RATE: 98.9%

Our working set is mostly in L2. Can we move it even closer?
MEMORY BANDWIDTH

Global Memory (Framebuffer)
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“

<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>5.141ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>1.572ms</td>
<td>3.27x</td>
<td>3.27x</td>
</tr>
<tr>
<td>L1 cache</td>
<td>1.454ms</td>
<td>3.54x</td>
<td>1.08x</td>
</tr>
</tbody>
</table>
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, ~ 1ms

<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>5.141ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>1.572ms</td>
<td>3.27x</td>
<td>3.27x</td>
</tr>
<tr>
<td>(L1 cache)</td>
<td>(1.454ms)</td>
<td>(3.54x)</td>
<td>(1.08x)</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>1.021ms</td>
<td>5.04x</td>
<td>1.54x</td>
</tr>
<tr>
<td>Category:</td>
<td>Latency Bound - Shared Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem:</td>
<td>Long memory latencies are difficult to hide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal:</td>
<td><strong>Reduce</strong> latency, move data to faster memory</td>
<td></td>
<td></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>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy:</td>
<td>(Cooperatively) move data to: • Shared Memory • (or Registers) • (or Constant Memory) • (or Texture Cache)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category:</td>
<td>Memory Bound - Shared Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem:</td>
<td>Too much data movement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal:</td>
<td><strong>Reduce</strong> amount of data traffic to/from global mem</td>
<td></td>
<td></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 3
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>5.141ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>1.572ms</td>
<td>3.27x</td>
<td>3.27x</td>
</tr>
<tr>
<td><em>(L1 cache)</em></td>
<td>(1.454ms)</td>
<td>(3.54x)</td>
<td>(1.08x)</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>1.021ms</td>
<td>5.04x</td>
<td>1.54x</td>
</tr>
</tbody>
</table>
IDENTIFY PERFORMANCE LIMITER

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

- 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

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

```c
a = b + c; // Independent ADDs  
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]);

    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)

```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
    c0 = b[0]
    a0 += logf(c0)
    c0 = b[2]
    a0 += logf(c0)
    c1 = b[1]
    a1 += logf(c1)
    c1 = b[3]
    a1 += logf(c1)
    a = a0 + a1
...
```

- `#pragma unroll` is useful to extract ILP
- Manually rewrite code if not a simple loop
Looking for more indicators

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>273.749 ms (100%)</td>
</tr>
<tr>
<td>End</td>
<td>274.77 ms (200%)</td>
</tr>
<tr>
<td>Duration</td>
<td>1.021 ms (10,000%)</td>
</tr>
<tr>
<td>Grid Size</td>
<td>[80,800,1]</td>
</tr>
<tr>
<td>Block Size</td>
<td>[32,2,1]</td>
</tr>
<tr>
<td>Registers/Thread</td>
<td>12</td>
</tr>
<tr>
<td>Shared Memory/Block</td>
<td>640 B</td>
</tr>
<tr>
<td>Occupancy</td>
<td></td>
</tr>
<tr>
<td>Achieved</td>
<td>47.4%</td>
</tr>
<tr>
<td>Theoretical</td>
<td>50%</td>
</tr>
<tr>
<td>Limiter</td>
<td>Block Size</td>
</tr>
</tbody>
</table>
LOOKING FOR MORE INDICATORS

Not enough active warps to hide latencies?
GPUs cover latencies by having a lot of work in flight.

- The warp issues
- The warp waits (latency)

Fully covered latency:

Exposed latency, not enough warps:

No warp issues:
LATENCY AT HIGH OCCUPANCY

- Many active warps but with high latency instructions

![Exposed latency at high occupancy](image)

- 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%

<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>5.141ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>1.572ms</td>
<td>3.27x</td>
<td>3.27x</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>1.021ms</td>
<td>5.04x</td>
<td>1.54x</td>
</tr>
<tr>
<td>Occupancy</td>
<td>0.924ms</td>
<td>5.56x</td>
<td>1.17x</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>
</tbody>
</table>
| 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**

<table>
<thead>
<tr>
<th>Category:</th>
<th>Latency Bound - Instruction Level Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem:</td>
<td>Not enough independent work per thread</td>
</tr>
<tr>
<td>Goal:</td>
<td>Do more parallel work inside single threads</td>
</tr>
<tr>
<td>Indicators:</td>
<td>High execution dependency, increasing occupancy has no/little positive effect, still registers available</td>
</tr>
<tr>
<td>Strategy:</td>
<td>• Unroll loops (#pragma unroll)</td>
</tr>
<tr>
<td></td>
<td>• Refactor threads to compute n output values at the same time (code duplication)</td>
</tr>
</tbody>
</table>
ITERATION 4
### 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>5.141ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better Memory Accesses</td>
<td>1.572ms</td>
<td>3.27x</td>
<td>3.27x</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>1.021ms</td>
<td>5.04x</td>
<td>1.54x</td>
</tr>
<tr>
<td>Higher Occupancy</td>
<td>0.924ms</td>
<td>5.56x</td>
<td>1.17x</td>
</tr>
</tbody>
</table>
IDENTIFY PERFORMANCE LIMITER

Aha!
Getting into the high utilization region
LOOKING FOR INDICATORS

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.

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.

Latency is likely not the primary performance bottleneck for this kernel, but you may still want to perform that analysis.
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

Schedulers saturated
- Utilization: 90%

Pipe saturated
- Utilization: 64%

Schedulers and pipe saturated
- Utilization: 92%
READ-ONLY CACHE (TEXTURE UNITS)

- SM
  - Registers
  - SMEM/L1$
  - Texture Units

- SM
  - Registers
  - SMEM/L1$
  - Texture Units

Skip LSU
Cache loads

L2$

Global Memory (Framebuffer)
READ-ONLY PATH

- Annotate read-only parameters with const __restrict

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

- The compiler generates LDG instructions: 0.829ms
### 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
## 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>5.141ms</td>
<td>1.00x</td>
<td></td>
</tr>
<tr>
<td>Better memory accesses</td>
<td>1.572ms</td>
<td>3.27x</td>
<td>3.27x</td>
</tr>
<tr>
<td>Shared memory</td>
<td>1.021ms</td>
<td>5.04x</td>
<td>1.54x</td>
</tr>
<tr>
<td>Higher Occupancy</td>
<td>0.924ms</td>
<td>5.56x</td>
<td>1.10x</td>
</tr>
<tr>
<td>Read-Only path</td>
<td>0.829ms</td>
<td>6.20x</td>
<td>1.11x</td>
</tr>
</tbody>
</table>

6.20x 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>Separable filter <em>(use a better algorithm)</em></td>
<td>0.431ms</td>
<td>11.94x</td>
<td>1.93x</td>
</tr>
<tr>
<td>Process two pixels per thread (memory efficiency + ILP)</td>
<td>0.421ms</td>
<td>12.21x</td>
<td>1.02x</td>
</tr>
<tr>
<td>Use 64-bit shared memory (remove bank conflicts)</td>
<td>0.409ms</td>
<td>12.58x</td>
<td>1.03x</td>
</tr>
<tr>
<td>Use float instead of int (increase instruction throughput)</td>
<td>0.361ms</td>
<td>14.25x</td>
<td>1.13x</td>
</tr>
</tbody>
</table>

14.25x speedup in companion code (includes algorithmic improvements)

**Companion Code:** [https://github.com/chmaruni/nsight-gtc](https://github.com/chmaruni/nsight-gtc)
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

![Graph showing speedup of Tesla K80 (Kepler) and Quadro M6000 (Maxwell) over kernel versions.](image-url)
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 Sakharynykh 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