Optimizing Application Performance with CUDA Profiling Tools
Why Profile?

- 100’s of cores
- 10,000’s of threads
- Great memory bandwidth
- Best at parallel execution

- A few cores
- 10’s of threads
- Good memory bandwidth
- Best at serial execution
Graphical and Command-Line

- NVIDIA® Visual Profiler
  - Standalone (nvvp)
  - Integrated into NVIDIA® Nsight™ Eclipse Edition (nsight)

- nvprof
  - Command-line profiler

- Current command-line profiler still available
Profiling Session

Create New Session
Set executable properties

File: /tmp/diverge
Working directory: Enter working directory [optional]
Arguments: Enter command-line arguments
Environment: Name Value

< Back Next > Cancel Finish
NVIDIA Visual Profiler
Timeline

GPU/CPU Timeline
CPU Timeline

CUDA API Invocations
Measuring Time

Measure time with horizontal rulers. Supports overlapping ranges, snap-to-edge.
Correlating CPU and GPU Activity

API Call

Stream
Properties - Memcpy

Memcpy Properties

- Name: Start
  - Value: 49.008 ms
- Name: End
  - Value: 49.054 ms
- Name: Duration
  - Value: 45.761 µs
- Name: Size
  - Value: 256 KB
- Name: Throughput
  - Value: 5.34 GB/s
Analysis, Details, etc.

Additional Views
Concurrent Kernels

Compute row shows concurrent kernel execution

Multiple streams launch independent kernels
Profiling Flow

- Understand CPU behavior on timeline
  - Add profiling “annotations” to application
  - NVIDIA Tools Extension
    - Custom markers and time ranges
    - Custom naming

- Focus profiling on region of interest
  - Reduce volume of profile data
  - Improve usability of Visual Profiler
  - Improve accuracy of analysis

- Analyze for optimization opportunities
Annotations: NVIDIA Tools Extension

- Developer API for CPU code
- Installed with CUDA Toolkit (libnvToolsExt.so)
- Naming
  - Host OS threads: nvtxNameOsThread()
  - CUDA device, context, stream: nvtxNameCudaStream()
- Time Ranges and Markers
  - Range: nvtxRangeStart(), nvtxRangeEnd()
  - Instantaneous marker: nvtxMark()
Example: Time Ranges

- Testing algorithm in testbench
- Use time ranges API to mark initialization, test, and results

```c
... nvtxRangeId_t id0 = nvtxRangeStart("Initialize");
< init code >
nvtxRangeEnd(id0);
nvtxRangeId_t id1 = nvtxRangeStart("Test");
< compute code >
nvtxRangeEnd(id1);
...```

Example code snippet showing how to use the time ranges API to mark different stages of a computation.
Example: Time Ranges
Profile Region Of Interest

- cudaProfilerStart() / cudaProfilerStop() in CPU code
- Specify representative subset of app execution
  - Manual exploration and analysis simplified
  - Automated analysis focused on performance critical code

```c
for (i = 0; i < N; i++) {
    if (i == 12) cudaProfilerStart();
    <loop body>
    if (i == 15) cudaProfilerStop();
}
```
Enable Region Of Interest

- Insert `cudaProfilerStart()` / `cudaProfilerStop()`
- Disable profiling at start of application
Example: Without cudaProfilerStart/Stop

Region of Interest
Example: With cudaProfilerStart/Stop
Analysis

- Visual inspection of timeline
- Automated Analysis
- Metrics and Events
Visual Inspection

- Understand CPU/GPU interactions
  - Use nvToolsExt to mark time ranges on CPU
  - Is application taking advantage of both CPU and GPU?
  - Is CPU waiting on GPU? Is GPU waiting on CPU?

- Look for potential concurrency opportunities
  - Overlap memcpy and kernel
  - Concurrent kernels

- Automated analysis does some of this
Automated Analysis - Application

- Analyze entire application
  - Timeline
  - Hardware performance counters
LowMemcpy Throughput [ 997.19 MB/s avg, for memcpys accounting for 68.1% of all memcpy time ]
The memory copies are not fully using the available host to device bandwidth.

Pinned Memory

Page-locked or pinned memory transfers attain the highest bandwidth between the host and the device. On PCIe x16 Gen2 cards, for example, pinned memory can attain greater than 5 GBps transfer rates.

Pinned memory is allocated using the cudaMallocHost() or cudaMemcpyHostAlloc() functions in the Runtime API. The bandwidthTest.cu program in the CUDA SDK shows how to use these functions as well as how to measure memory transfer performance.

Pinned memory should not be overused. Excessive use can reduce overall system performance because pinned memory is a scarce resource. How much is too much is difficult to tell in advance, so as with all optimizations, test the applications and the systems they run on for optimal performance parameters.

Parent topic: Data Transfer Between Host and Device
Results Correlated With Timeline
Analysis Properties

- Highlight a kernel or memcpy in timeline
  - Properties shows analysis results for that specific kernel / memcpy
  - Optimization opportunities are flagged
Automated Analysis - Single Kernel

Analysis performed on single kernel instance
Uncoalesced Global Memory Accesses

- Access pattern determines number of memory transactions
  - Report loads/stores where access pattern is inefficient
Source Correlation
Divergent Branches

- Divergent control-flow for threads within a warp
  - Report branches that have high average divergence

![Divergent Branches](image)
Source Correlation

```c
SrcDst = IMAD( IMAD(blockIdx.y, KERS_BLOCK_HEIGHT, OffsThreadInCol), ImgStride, IMAD(blockIdx.x, KERS_BLOCK_WIDTH_HALF, OffsThreadInRow) * short *bl_ptr = block + IMAD(OffsThreadInCol, KERS_SMEMBLOCK_STRIDE, OffsThreadInRow) * 2)

// Load data to shared memory (only first half of threads in each row performs data movin
if(OffsThreadInRow < KERS_BLOCK_WIDTH_HALF){
    #pragma unroll
    for(int i = 0; i < BLOCK_SIZE; i++)
        ((int *)bl_ptr)[i * KERS_SMEMBLOCK_STRIDE / 2] = ((int *)SrcDst)[i * (ImgStride

    __syncthreads();
    CUDAKernelShortDCT(block + OffsThreadInCol * KERS_SMEMBLOCK_STRIDE, OffsThreadInRow, KERS_SMEMBLOCK_STRIDE)
    __syncthreads();
    CUDAKernelShortDCT((unsigned int *)block + OffsThreadInRow * KERS_SMEMBLOCK_STRIDE + 0)
    __syncthreads();
```
Enabling Source Correlation

- Source correlation requires that source/line information be embedded in executable
  - Available in debug executables: `nvcc -G`
  - New flag for optimized executables: `nvcc -lineinfo`
Detailed Profile Data
Detailed Summary Profile Data
Filtering
Metrics and Events

Select metrics and events to be collected on individual devices

Device: GeForce GTX 480

Metrics
- Memory
- Instruction
  - Branch Efficiency
  - IPC
  - Per Multiprocessor IPC
  - Instruction Replay Overhead
  - Shared Memory Replay Overhead
  - Global Memory Cache Replay Overhead
  - Warp Execution Efficiency
  - Local Memory Cache Replay Overhead

Multiprocessor
- Multiprocessor Efficiency

Achieved Occupancy
- Per Multiprocessor Efficiency

Cache
- Texture
  - Texture Cache Hit Rate
  - Texture Cache Throughput
# Metrics and Events

## Detailed Analysis

<table>
<thead>
<tr>
<th>Name</th>
<th>Start Time</th>
<th>Duration</th>
<th>Warp Execution Efficiency</th>
<th>Achieved Occupancy</th>
<th>Grid Size</th>
<th>Block Size</th>
<th>Regs</th>
<th>Static SMem</th>
<th>Dynamic SMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory HtoA [sync]</td>
<td>3.929 ms</td>
<td>176.773 µs</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>4.108 ms</td>
<td>708.262 µs</td>
<td>100%</td>
<td>0.328</td>
<td>[64,64,1]</td>
<td>[8,8,1]</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>5.122 ms</td>
<td>708.49 µs</td>
<td>100%</td>
<td>0.328</td>
<td>[64,64,1]</td>
<td>[8,8,1]</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>5.945 ms</td>
<td>708.394 µs</td>
<td>100%</td>
<td>0.327</td>
<td>[64,64,1]</td>
<td>[8,8,1]</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>6.763 ms</td>
<td>708.418 µs</td>
<td>100%</td>
<td>0.328</td>
<td>[64,64,1]</td>
<td>[8,8,1]</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>7.581 ms</td>
<td>708.534 µs</td>
<td>100%</td>
<td>0.327</td>
<td>[64,64,1]</td>
<td>[8,8,1]</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>8.4 ms</td>
<td>708.153 µs</td>
<td>100%</td>
<td>0.327</td>
<td>[64,64,1]</td>
<td>[8,8,1]</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>9.219 ms</td>
<td>708.221 µs</td>
<td>100%</td>
<td>0.327</td>
<td>[64,64,1]</td>
<td>[8,8,1]</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
</tbody>
</table>

## Summary Analysis

<table>
<thead>
<tr>
<th>Name</th>
<th>Warp Execution Efficiency</th>
<th>Achieved Occupancy</th>
<th>Avg. Duration</th>
<th>Regs</th>
<th>Static SMem</th>
<th>Avg. Dynamic SMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDAkernel2DCT(float*, float*, int)</td>
<td>100%</td>
<td>0.3</td>
<td>92.66 µs</td>
<td>43</td>
<td>2112</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel2DICT(float*, float*, int)</td>
<td>100%</td>
<td>0.302</td>
<td>97.655 µs</td>
<td>43</td>
<td>2112</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernelQuantizationShort(short*, int)</td>
<td>67.5%</td>
<td>0.317</td>
<td>143.288 µs</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernelQuantizationFloat(float*, int)</td>
<td>98.7%</td>
<td>0.318</td>
<td>173.964 µs</td>
<td>27</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernelShortIDCT(short*, int)</td>
<td>74.7%</td>
<td>0.468</td>
<td>174.399 µs</td>
<td>39</td>
<td>2176</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernelShortDCT(short*, int)</td>
<td>75%</td>
<td>0.376</td>
<td>189.663 µs</td>
<td>45</td>
<td>2176</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>100%</td>
<td>0.328</td>
<td>708.301 µs</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>CUDAkernel1DCT(float*, int, int, int)</td>
<td>100%</td>
<td>0.328</td>
<td>708.327 µs</td>
<td>28</td>
<td>512</td>
<td>0</td>
</tr>
</tbody>
</table>
nvprof

- Textual reports
  - Summary of GPU and CPU activity
  - Trace of GPU and CPU activity
  - Event collection

- Headless profile collection
  - Use nvprof on headless node to collect data
  - Visualize timeline with Visual Profiler
nvprof Usage

$ nvprof [nvprof_args] <app> [app_args]

- Argument help
  $ nvprof --help
nvprof - GPU Summary

$ nvprof dct8x8

<table>
<thead>
<tr>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>49.52</td>
<td>9.36ms</td>
<td>101</td>
<td>92.68us</td>
<td>92.31us</td>
<td>94.31us</td>
<td>CUDAkernel2DCT(float*, float*, int)</td>
</tr>
<tr>
<td>37.47</td>
<td>7.08ms</td>
<td>10</td>
<td>708.31us</td>
<td>707.99us</td>
<td>708.50us</td>
<td>CUDAkernel1DCT(float*, int, int,int)</td>
</tr>
<tr>
<td>3.75</td>
<td>708.42us</td>
<td>1</td>
<td>708.42us</td>
<td>708.42us</td>
<td>708.42us</td>
<td>CUDAkernel1IDCT(float*,int,int,int)</td>
</tr>
<tr>
<td>1.84</td>
<td>347.99us</td>
<td>2</td>
<td>173.99us</td>
<td>173.59us</td>
<td>174.40us</td>
<td>CUDAkernelQuantizationFloat()</td>
</tr>
<tr>
<td>1.75</td>
<td>331.37us</td>
<td>2</td>
<td>165.69us</td>
<td>165.67us</td>
<td>165.70us</td>
<td>[CUDA memcpyDtoH]</td>
</tr>
<tr>
<td>1.41</td>
<td>266.70us</td>
<td>2</td>
<td>89.70us</td>
<td>133.35us</td>
<td>177.00us</td>
<td>[CUDA memcpyHtoD]</td>
</tr>
<tr>
<td>1.00</td>
<td>189.64us</td>
<td>1</td>
<td>189.64us</td>
<td>189.64us</td>
<td>189.64us</td>
<td>CUDAkernelShortDCT(short*, int)</td>
</tr>
<tr>
<td>0.94</td>
<td>176.87us</td>
<td>1</td>
<td>176.87us</td>
<td>176.87us</td>
<td>176.87us</td>
<td>[CUDA memcpyHtoA]</td>
</tr>
<tr>
<td>0.92</td>
<td>174.16us</td>
<td>1</td>
<td>174.16us</td>
<td>174.16us</td>
<td>174.16us</td>
<td>CUDAkernelShortIDCT(short*, int)</td>
</tr>
<tr>
<td>0.76</td>
<td>143.31us</td>
<td>1</td>
<td>143.31us</td>
<td>143.31us</td>
<td>143.31us</td>
<td>CUDAkernelQuantizationShort(short*)</td>
</tr>
<tr>
<td>0.52</td>
<td>97.75us</td>
<td>1</td>
<td>97.75us</td>
<td>97.75us</td>
<td>97.75us</td>
<td>CUDAkernel2IDCT(float*, float*)</td>
</tr>
<tr>
<td>0.12</td>
<td>22.59us</td>
<td>1</td>
<td>22.59us</td>
<td>22.59us</td>
<td>22.59us</td>
<td>[CUDA memcpyDtoA]</td>
</tr>
</tbody>
</table>
nvprof - GPU Summary (csv)

$ nvprof --csv dct8x8

======= Profiling result:
Time(%) , Time, Calls, Avg, Min, Max, Name
, ms, us, us, us, us, us, us,
49.51, 9.35808, 101, 92.65400, 92.38200, 94.19000, "CUDAkernel2DCT(float*, float*, int)"
37.47, 7.08288, 10, 708.2870, 707.9360, 708.7070, "CUDAkernel1DCT(float*, int, int, int)"
3.75, 0.70847, 1, 708.4710, 708.4710, 708.4710, "CUDAkernel1IDCT(float*, int, int, int)"
1.84, 0.34802, 2, 174.0090, 173.8130, 174.2060, "CUDAkernelQuantizationFloat(float*, int)"
1.75, 0.33137, 2, 165.6850, 165.6690, 165.7020, "[CUDA memcpyDtoH]"
1.42, 0.26759, 2, 133.7970, 89.89100, 177.7030, "[CUDA memcpyHtoD]"
1.00, 0.18874, 1, 188.7360, 188.7360, 188.7360, "CUDAkernelShortDCT(short*, int)"
0.94, 0.17687, 1, 176.8690, 176.8690, 176.8690, "[CUDA memcpyHtoA]"
0.93, 0.17594, 1, 175.9390, 175.9390, 175.9390, "CUDAkernelShortIDCT(short*, int)"
0.76, 0.14281, 1, 142.8130, 142.8130, 142.8130, "CUDAkernelQuantizationShort(short*, int)"
0.52, 0.09758, 1, 97.57800, 97.57800, 97.57800, "CUDAkernel2IDCT(float*, float*, int)"
0.12, 0.02259, 1, 22.59300, 22.59300, 22.59300, "[CUDA memcpyDtoA]"
$ nvprof --print-gpu-trace dct8x8

======== Profiling result:

<table>
<thead>
<tr>
<th>Duration</th>
<th>Grid Size</th>
<th>Block Size</th>
<th>Regs</th>
<th>SSMem</th>
<th>DSMem</th>
<th>Size</th>
<th>Throughput</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>167.82ms</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.05MB</td>
<td>5.93GB/s</td>
<td>[CUDA memcpy HtoA]</td>
</tr>
<tr>
<td>168.00ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>168.95ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>169.74ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>170.53ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>171.32ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>172.11ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>172.89ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>173.68ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>174.47ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>175.26ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>176.05ms</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>27</td>
<td>0B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernelQuantization(...)</td>
</tr>
<tr>
<td>176.23ms</td>
<td>22.82us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.05MB</td>
<td>45.96GB/s</td>
<td>[CUDA memcpyDtoA]</td>
</tr>
</tbody>
</table>
nvprof - CPU/GPU Trace

$ nvprof --print-gpu-trace --print-api-trace dct8x8

======== Profiling result:

<table>
<thead>
<tr>
<th>Start</th>
<th>Duration</th>
<th>Grid Size</th>
<th>Block Size</th>
<th>Regs</th>
<th>SSMem</th>
<th>DSMem</th>
<th>Size</th>
<th>Throughput</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>167.82ms</td>
<td>176.84us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.05MB</td>
<td>5.93GB/s</td>
<td>[CUDA memcpy HtoA]</td>
</tr>
<tr>
<td>167.81ms</td>
<td>2.00us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaSetupArgument</td>
</tr>
<tr>
<td>167.81ms</td>
<td>38.00us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaLaunch</td>
</tr>
<tr>
<td>167.85ms</td>
<td>1.00ms</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaDeviceSynchronize</td>
</tr>
<tr>
<td>168.00ms</td>
<td>708.51us</td>
<td>(64 64 1)</td>
<td>(8 8 1)</td>
<td>28</td>
<td>512B</td>
<td>0B</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
<tr>
<td>168.86ms</td>
<td>2.00us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaConfigureCall</td>
</tr>
<tr>
<td>168.86ms</td>
<td>1.00us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaSetupArgument</td>
</tr>
<tr>
<td>168.86ms</td>
<td>1.00us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaSetupArgument</td>
</tr>
<tr>
<td>168.86ms</td>
<td>1.00us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaSetupArgument</td>
</tr>
<tr>
<td>168.87ms</td>
<td>0ns</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaLaunch</td>
</tr>
<tr>
<td>168.87ms</td>
<td>24.00us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>cudaDeviceSynchronize</td>
</tr>
<tr>
<td>168.89ms</td>
<td>761.00us</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>CUDAkernel1DCT(float*, ...)</td>
</tr>
</tbody>
</table>
nvprof - Event Query

$ nvprof --devices 0 --query-events

========= Available Events:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device 0:</td>
<td></td>
</tr>
<tr>
<td>Domain domain_a:</td>
<td></td>
</tr>
<tr>
<td>sm_cta_launched:</td>
<td>Number of thread blocks launched on a multiprocessor.</td>
</tr>
<tr>
<td>l1_local_load_hit:</td>
<td>Number of cache lines that hit in L1 cache for local memory load accesses.</td>
</tr>
<tr>
<td>l1_local_load_miss:</td>
<td>Number of cache lines that miss in L1 cache for local memory load accesses.</td>
</tr>
<tr>
<td>l1_local_store_hit:</td>
<td>Number of cache lines that hit in L1 cache for local memory store accesses.</td>
</tr>
</tbody>
</table>

  - l1_load_hit: Number of cache lines that hit in L1 cache for local memory load accesses. In case of perfect coalescing this increments by 1, 2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.
  - l1_load_miss: Number of cache lines that miss in L1 cache for local memory load accesses. In case of perfect coalescing this increments by 1, 2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.
  - l1_store_hit: Number of cache lines that hit in L1 cache for local memory store accesses. In case of perfect coalescing this increments by 1, 2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.
### nvprof - Event Collection

$ nvprof --devices 0 --events branch,divergent_branch

| Device 0 |
|------------------|---------|---------|---------|---------|-----------------|
| Invocations | Avg     | Min     | Max     | Event Name |
| Kernel: CUDAkernel1DCT(float*, int, int, int) | 1 | 475136 | 475136 | 475136 | branch |
| Kernel: CUDAkernelQuantizationFloat(float*, int) | 2 | 180809 | 180440 | 181178 | branch |
| Kernel: CUDAkernel1DCT(float*, int, int, int) | 10 | 475136 | 475136 | 475136 | branch |
| Kernel: CUDAkernelShortIDCT(short*, int) | 1 | 186368 | 186368 | 186368 | branch |
| Kernel: CUDAkernel2IDCT(float*, float*, int) | 1 | 61440 | 61440 | 61440 | branch |
nvprof - Profile Data Import

- Produce profile into a file using -o
  
  $ nvprof -o profile.out <app> <app args>

- Import into Visual Profiler
  
  File menu -> Import nvprof Profile...

- Import into nvprof to generate textual outputs
  
  $ nvprof -i profile.out
  $ nvprof -i profile.out --print-gpu-trace
  $ nvprof -i profile.out --print-api-trace
Get Started

- Download free CUDA Toolkit: www.nvidia.com/getcuda
- Join the community: developer.nvidia.com/join
- Visit Experts Table, Developer Demo Stations
- Optimize your application with CUDA Profiling Tools

- **S0420** - Nsight Eclipse Edition for Linux and Mac
  - Wed. 5/16, 9am, Room A5
- **S0514** - GPU Performance Analysis and Optimization
  - Wed. 5/16, 3:30pm, Hall 1
Questions?