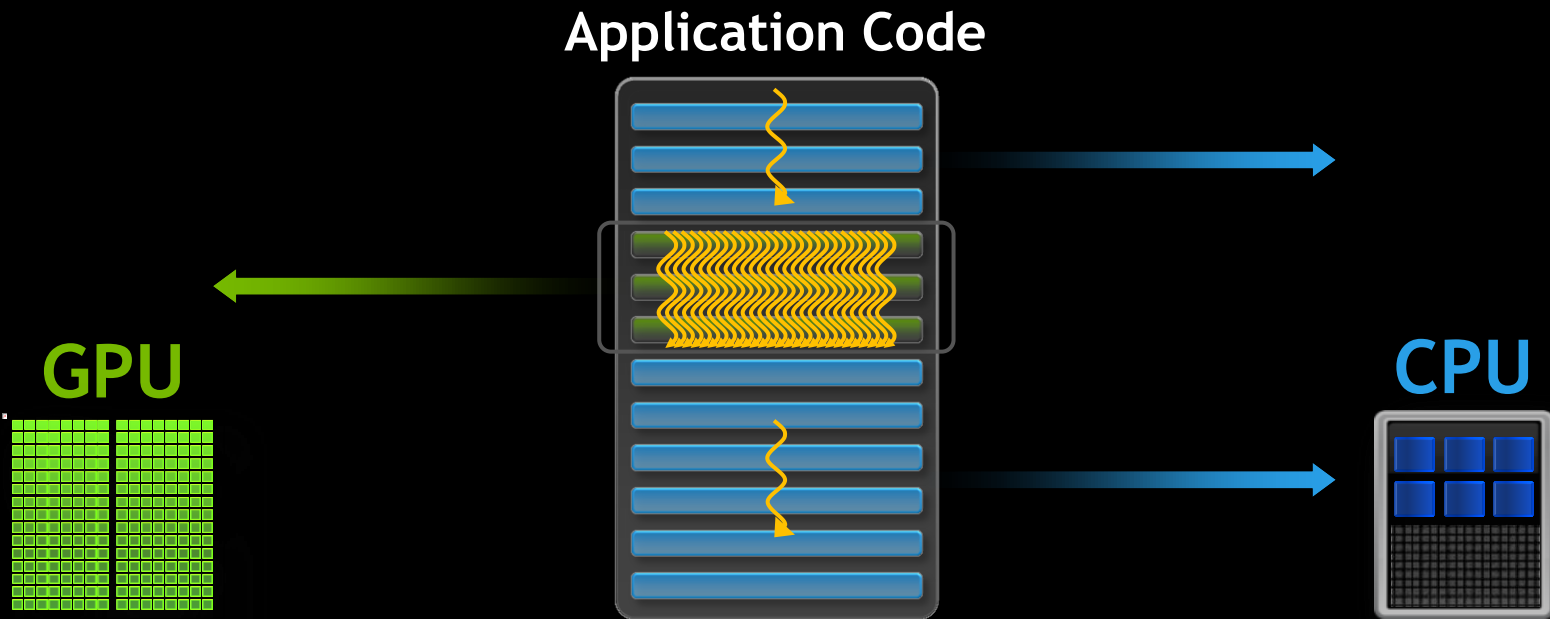


GPU TECHNOLOGY
CONFERENCE

Optimizing Application Performance with CUDA Profiling Tools



Why Profile?

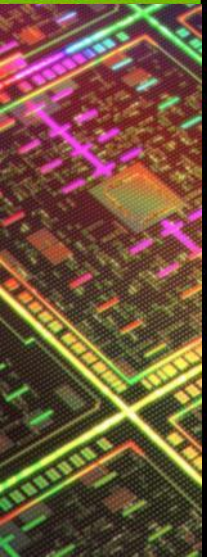


- 100's of cores
- 10,000's of threads
- Great memory bandwidth
- Best at parallel execution

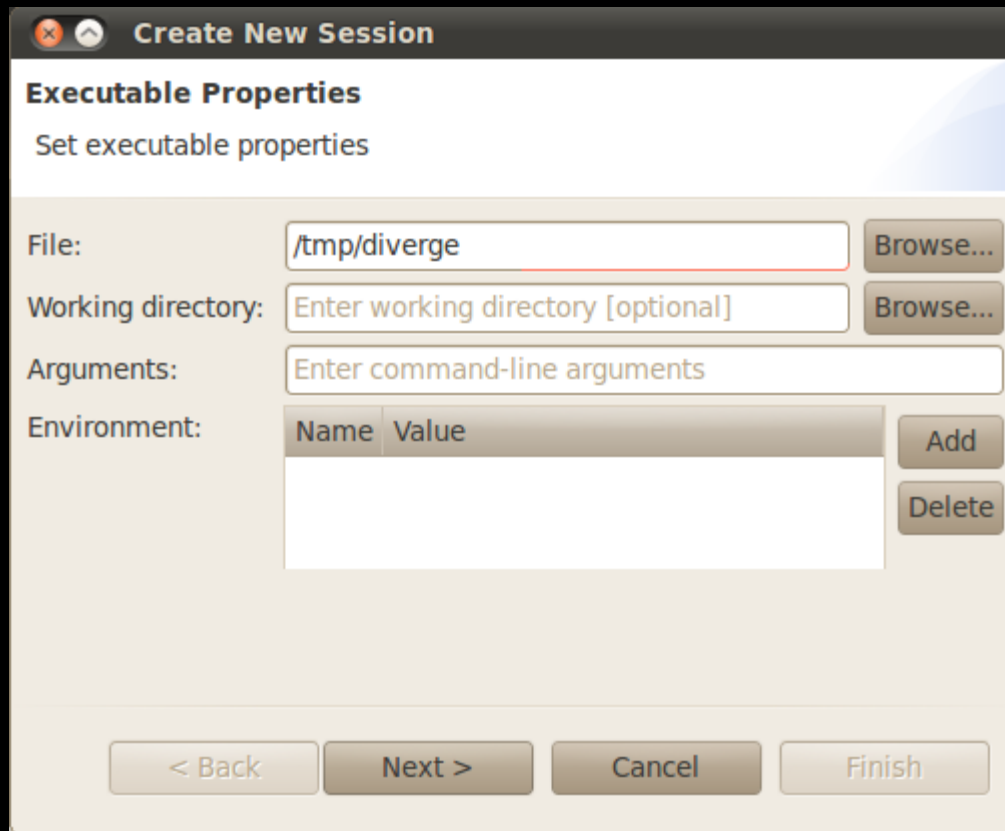
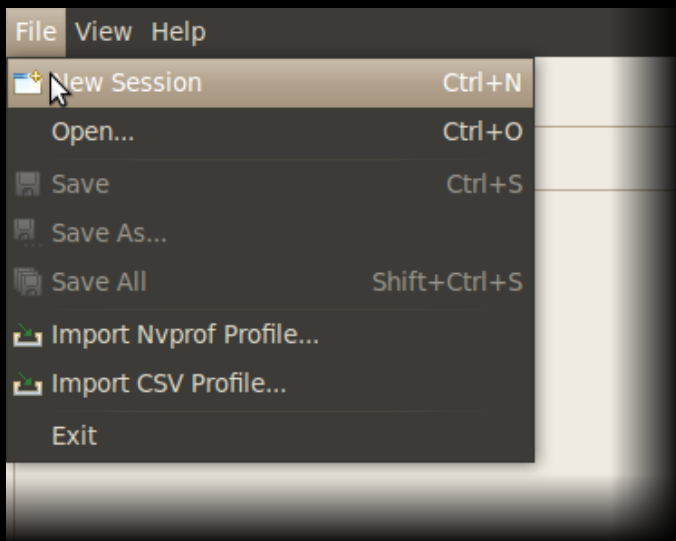
- A few cores
- 10's of threads
- Good memory bandwidth
- Best at serial execution

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



NVIDIA Visual Profiler

The screenshot displays the NVIDIA Visual Profiler interface for a session named 'diverge.vp'. The main window shows a GPU performance timeline for a GeForce GTX 480. A detailed view of a kernel, 'Vec32of32(int*, int*, int*, int)', is shown with the following properties:

Name	Value
Start	70.642 ms
End	72.064 ms
Duration	1.422 ms
Grid Size	[256,1,1]
Block Size	[256,1,1]
Registers/Thread	11
Shared Memory/Block	0 bytes
Occupancy	
Theoretical	100%
L1 Cache Configuration	
Shared Memory Requested	48 KB
Shared Memory Executed	48 KB

The bottom panel shows the session configuration for 'diverge.vp':

- Executable:** File: bin/diverge (Browse...)
- Working directory:** Enter working directory [optional] (Browse...)
- Arguments:** Enter command-line arguments
- Environment:** Name Value

Timeline

The screenshot displays a performance analysis tool interface. The main window shows a timeline for a process named '25290' on a 'GeForce GTX 480' GPU. A green box highlights a specific section of the timeline, including the 'Runtime API' (cudaMemcpyAsync), 'Driver API', and 'Compute' sections. The 'Compute' section shows several tasks: 'VecThe...', 'Vec50(in...', 'Vec1o...', and 'Vec1of32x(int*, int*, int*, int)'. The 'Streams' section shows 'Stream 1' with similar tasks. A 'Properties' panel on the right provides details for the 'Vec32of32(int*, int*, int*, int)' task.

Name	Value
Start	70.642 ms
End	72.064 ms
Duration	1.422 ms
Grid Size	[256,1,1]
Block Size	[256,1,1]
Registers/Thread	11
Shared Memory/Block	0 bytes
Occupancy	
Theoretical	100%
L1 Cache Configuration	
Shared Memory Requested	48 KB
Shared Memory Executed	48 KB

GPU/CPU Timeline

Session diverge.vp

Executable

File: bin/diverge [Browse...]

Working directory: Enter working directory [optional] [Browse...]

Arguments: Enter command-line arguments

Environment: Name Value

CPU Timeline

The screenshot displays the NVIDIA Visual Profiler interface. The main window shows a CPU timeline for process 25290, thread -1813960928. A green box highlights the 'Runtime API' section, which contains a single invocation of 'cudaMemcpyAsync' starting at approximately 0.055s and ending at 0.064s. A green callout box with the text 'CUDA API Invocations' points to this invocation. The GPU timeline below shows various compute kernels, including 'VecThe...', 'Vec50(in...', 'Vec1o...', and 'Vec1of32x(int*, int*, int*, int)'. The properties panel on the right shows details for the selected kernel 'Vec32of32(int*, int*, int*, int)', including start and end times, duration, and occupancy.

Name	Value
Start	70.642 ms
End	72.064 ms
Duration	1.422 ms
	[256,1,1]
	[256,1,1]
Registers/Thread	11
Shared Memory/Block	0 bytes
Occupancy	
Theoretical	100%
L1 Cache Configuration	
Shared Memory Requested	48 KB
Shared Memory Executed	48 KB

Session diverge.vp

Executable

File:

Working directory:

Arguments:

Environment:

Name	Value
------	-------

GPU Timeline

File View Run Help

*diverge.vp

0.05 s 0.055 s 0.06 s 0.065 s 0.

Process: 25290

- Thread: -1813960928
 - Runtime API: cudaMemcpyAsync
 - Driver API
- [0] GeForce GTX 480
 - Context 1 (CUDA)
 - MemCpy (HtoD)
 - MemCpy (DtoH)
 - Compute
 - 56.3% [4] Vec1of32x(in...)
 - 10.6% [4] Vec1of32(int...)
 - 13.3% [4] Vec50(int*, i...)
 - 12.5% [4] VecThen(int*...)
 - 7.3% [4] Vec32of32(int...)
 - 0.0% [4] VecEmpty(void)
 - Streams
 - Stream 1

Properties Detail Graphs

Vec32of32(int*, int*, int*, int)

Name	Value
Start	70.642 ms
End	72.064 ms
Duration	1.422 ms
Grid Size	[256,1,1]
Block Size	[256,1,1]
Registers/Thread	11
Shared Memory/Block	0 bytes
Occupancy	
Theoretical	100%
L1 Cache Configuration	
Shared Memory Requested	48 KB
Shared Memory Executed	48 KB

Analysis Details Console Settings

Session **diverge.vp**

Executable

File: bin/diverge

Working directory:

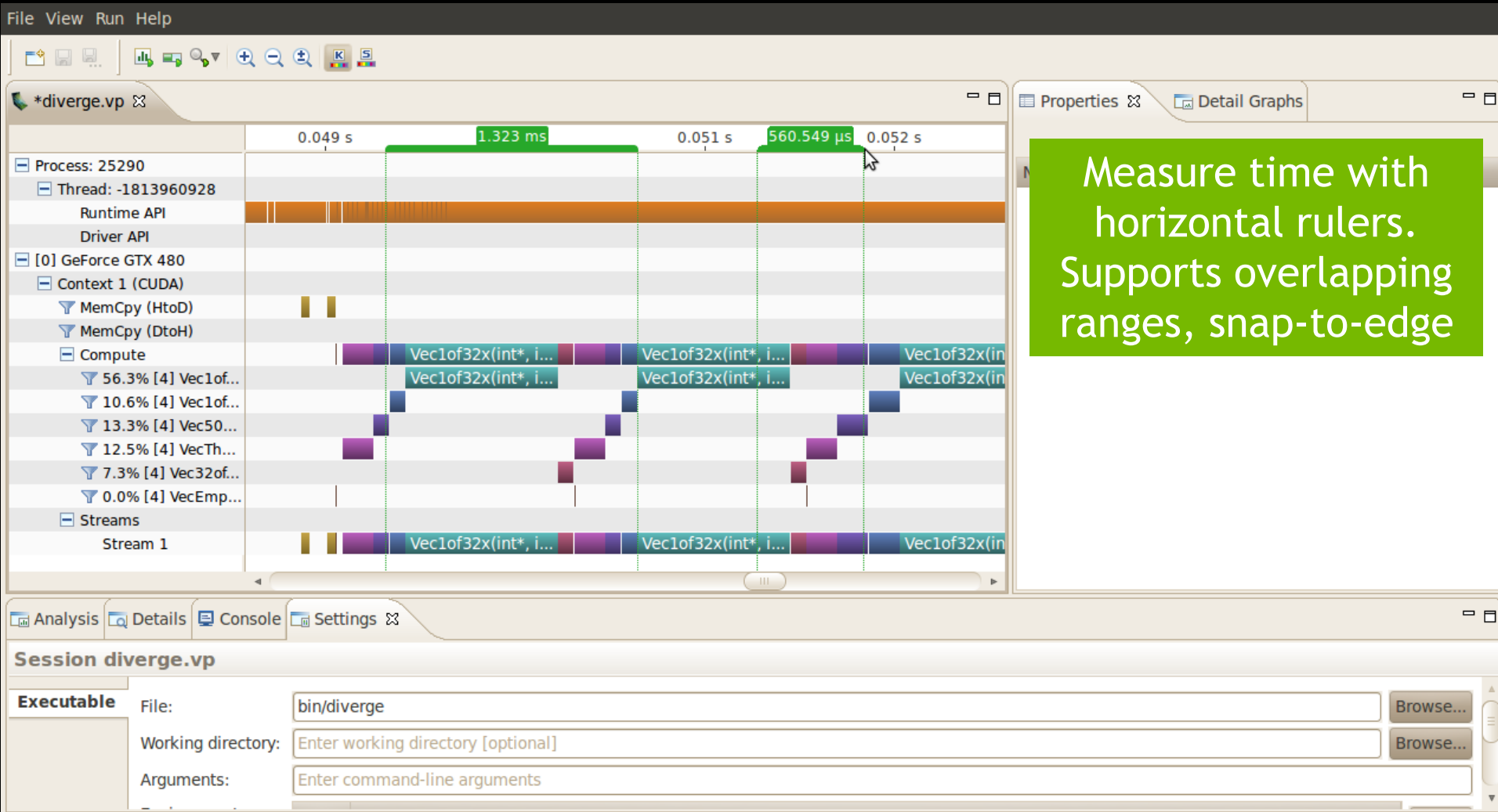
Arguments:

Environment:

Name	Value
------	-------

Device Activity

Measuring Time



Measure time with horizontal rulers. Supports overlapping ranges, snap-to-edge

Correlating CPU and GPU Activity

The screenshot displays the NVIDIA Visual Profiler interface. The top menu includes File, View, Run, and Help. The main window shows a timeline from 0.049 s to 0.052 s. On the left, the process tree shows 'Process: 25290' with threads for 'Runtime API', 'Driver API', and GPU context '[0] GeForce GTX 480'. The GPU context includes 'MemCpy (HtoD)', 'MemCpy (DtoH)', and 'Compute'. The 'Compute' section is expanded to show various kernels, with 'Vec1of32x(int*, i...)' highlighted in yellow. A green box labeled 'API Call' points to a small orange box on the CPU timeline. Another green box labeled 'Stream' points to a purple box on the GPU timeline. On the right, the 'Properties' pane shows details for the selected kernel:

Name	Value
Start	50.643 ms
End	51.448 ms
Duration	804.924 μ s
Grid Size	[2,1,1]
Block Size	[1,1,1]
Registers/Thread	12
Shared Memory/Block	0 bytes
Occupancy	
Theoretical	16.7%
L1 Cache Configuration	
Shared Memory Requested	48 KB
Shared Memory Executed	48 KB

At the bottom, the 'Session diverge.vp' configuration is shown with fields for 'Executable' (bin/diverge), 'Working directory', and 'Arguments'.

Properties - Kernel

The screenshot displays the NVIDIA Visual Profiler interface. The main window shows a timeline of execution from 0.049 s to 0.052 s. A tree view on the left shows the execution context: Process: 25290, Thread: -1813960928, Runtime API, Driver API, [0] GeForce GTX 480, Context 1 (CUDA), MemCpy (HtoD), MemCpy (DtoH), Compute, and Stream 1. The Compute section is expanded, showing several kernel calls, with 'Vec1of32x(int*, i...)' highlighted in yellow. A green box highlights the 'Kernel Properties' panel on the right, which displays the following data:

Name	Value
Start	50.643 ms
End	51.448 ms
Duration	804.924 μ s
Grid Size	[2,1,1]
Block Size	[1,1,1]
Registers/Thread	12
Shared Memory/Block	0 bytes
Occupancy	
Theoretical	16.7%
L1 Cache Configuration	
Shared Memory Requested	48 KB
Shared Memory Executed	48 KB

At the bottom of the interface, there is a 'Session diverge.vp' section with fields for 'Executable' (bin/diverge), 'Working directory', and 'Arguments'. A green box labeled 'Kernel Properties' is overlaid on the bottom right of the interface.

Properties - Memcpy

The screenshot displays the NVIDIA Nsight Visual Studio Edition interface. The main window shows a timeline for process 25290, thread -1813960928, on a GeForce GTX 480. A MemCpy (HtoD) operation is highlighted in the timeline. A properties window is open on the right, showing the following data:

Memcpy HtoD [async]	
Name	Value
Start	49.008 ms
End	49.054 ms
Duration	45.761 μ s
Size	256 KB
Throughput	5.34 GB/s

At the bottom of the interface, the 'Session diverge.vp' section shows the executable path: `/home/david/depot/davidg-linux-sw/sw/gpgpu/bin/x86_64_Linux_debug/diverge`.

Memcpy Properties

Analysis, Details, etc.

File View Run Help

*diverge.vp

0.05 s 0.055 s 0.06 s 0.065 s 0.

Process: 25290

- Thread: -1813960928
 - Runtime API: cudaMemcpyAsync
 - Driver API
- [0] GeForce GTX 480
 - Context 1 (CUDA)
 - MemCpy (HtoD)
 - MemCpy (DtoH)
 - Compute
 - 56.3% [4] Vec1of32x(in...)
 - 10.6% [4] Vec1of32(int...)
 - 13.3% [4] Vec50(int*, i...)
 - 12.5% [4] VecThen(int*...)
 - 7.3% [4] Vec32of32(int...)
 - 0.0% [4] VecEmpty(void)
 - Streams
 - Stream 1

Properties Detail Graphs

Vec32of32(int*, int*, int*, int)

Name	Value
Start	70.642 ms
End	72.064 ms
Duration	1.422 ms
Grid Size	[256,1,1]
Block Size	[256,1,1]
Registers/Thread	11
Shared Memory/Block	0 bytes
Occupancy	
Theoretical	100%
L1 Cache Configuration	
Shared Memory Requested	48 KB
	48 KB

Additional Views

Analysis Details Console Settings

Session diverge.vp

Executable

File: bin/diverge

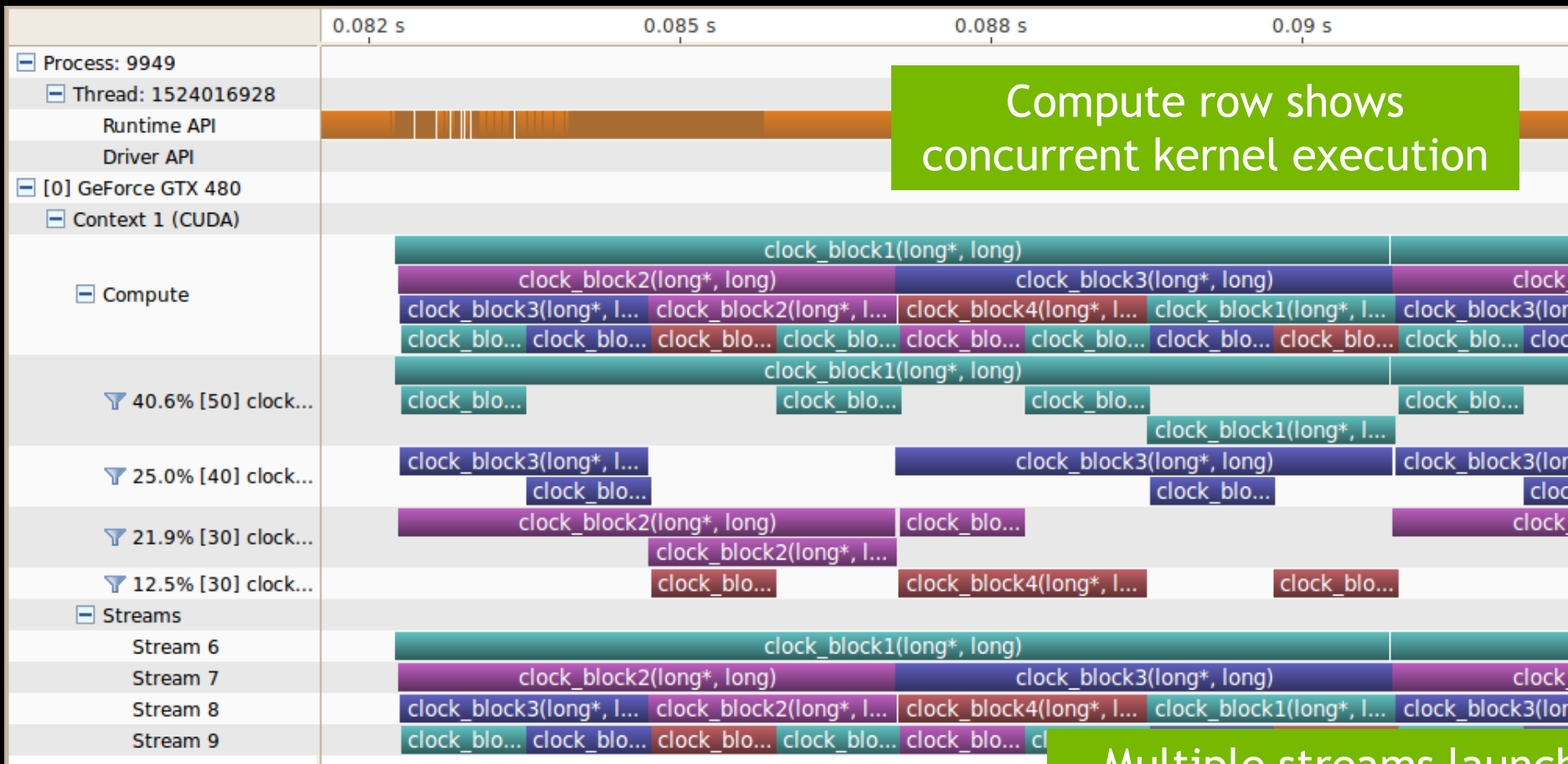
Working directory:

Arguments:

Environment:

Name	Value

Concurrent Kernels

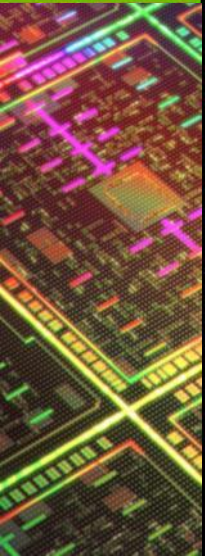


Compute row shows concurrent kernel execution

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

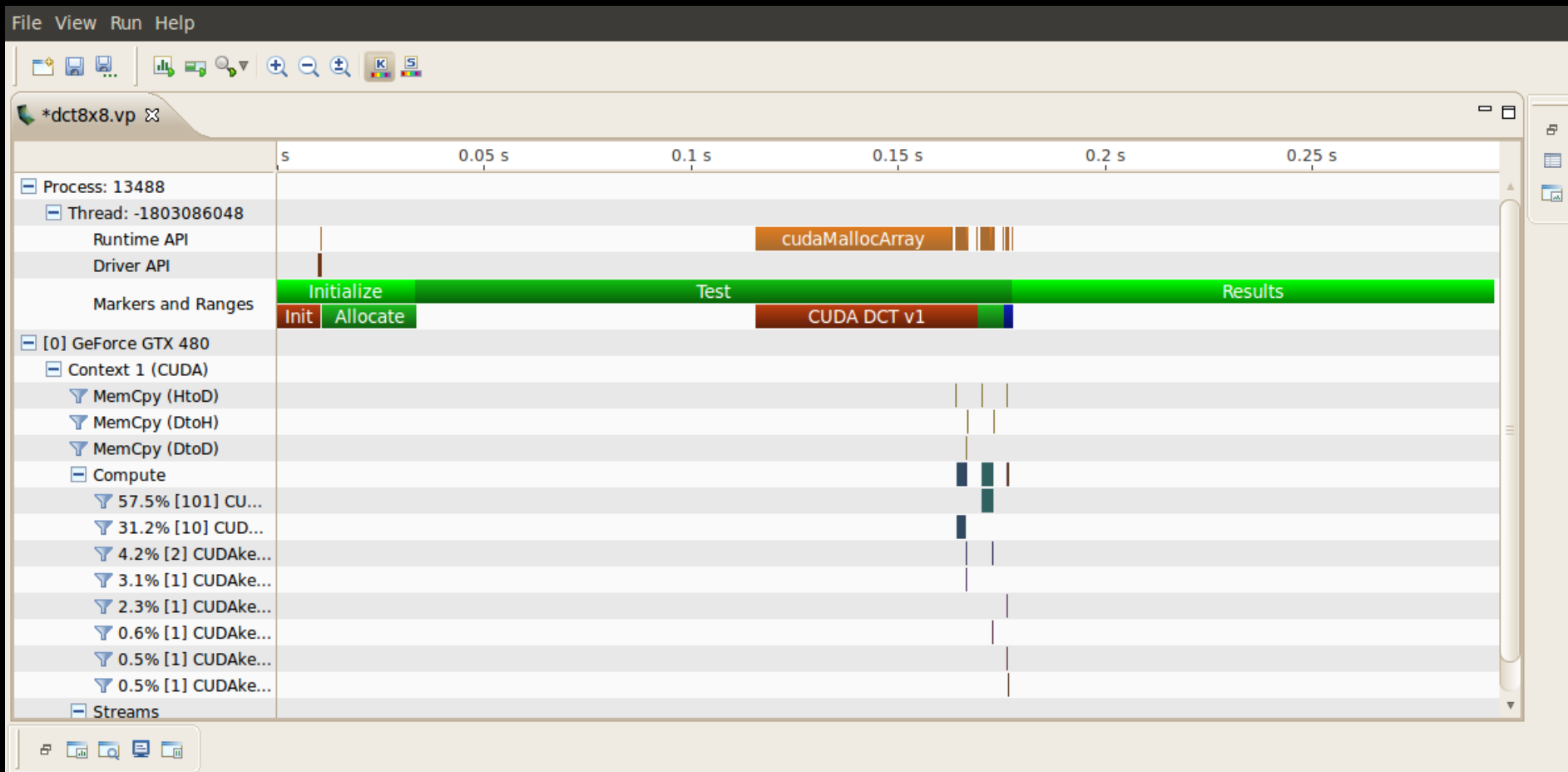
- Developer API for CPU code
- Installed with CUDA Toolkit (`libnvToolsExt.so`)
- Naming
 - Host OS threads: `nvtxNameOsThread()`
 - CUDA device, context, stream: `nvtxNameCudaStream()`
- Time Ranges and Markers
 - Range: `nvtxRangeStart()`, `nvtxRangeEnd()`
 - Instantaneous marker: `nvtxMark()`

Example: Time Ranges

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

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

Example: Time Ranges



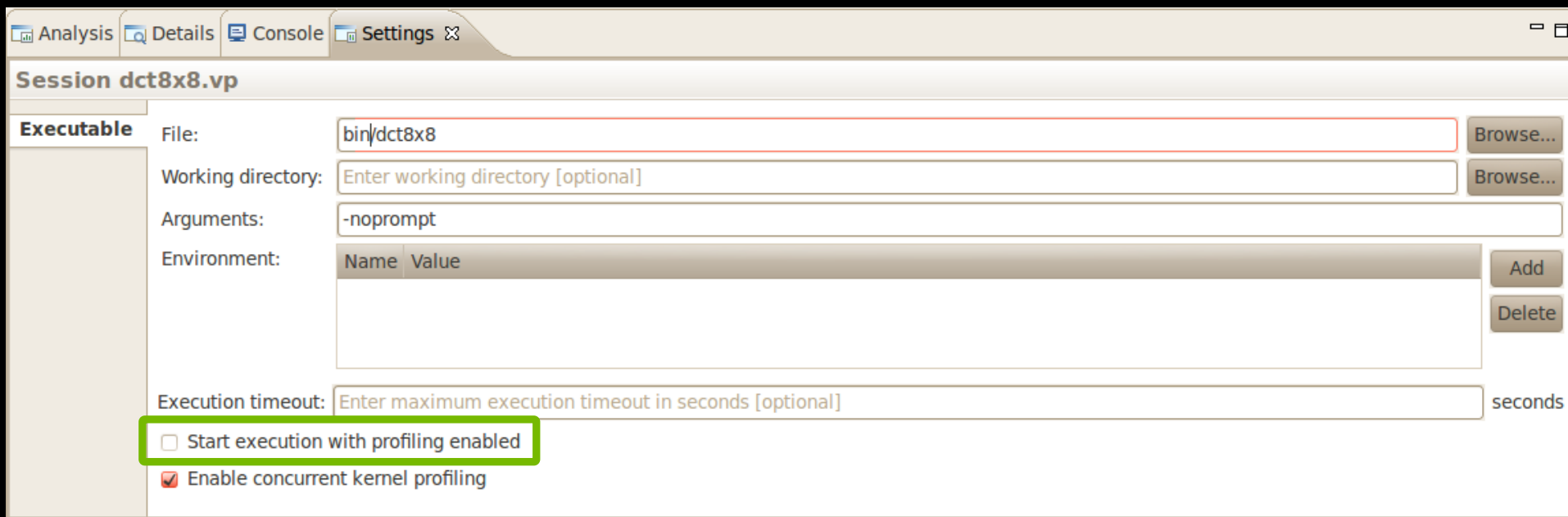
Profile Region Of Interest

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

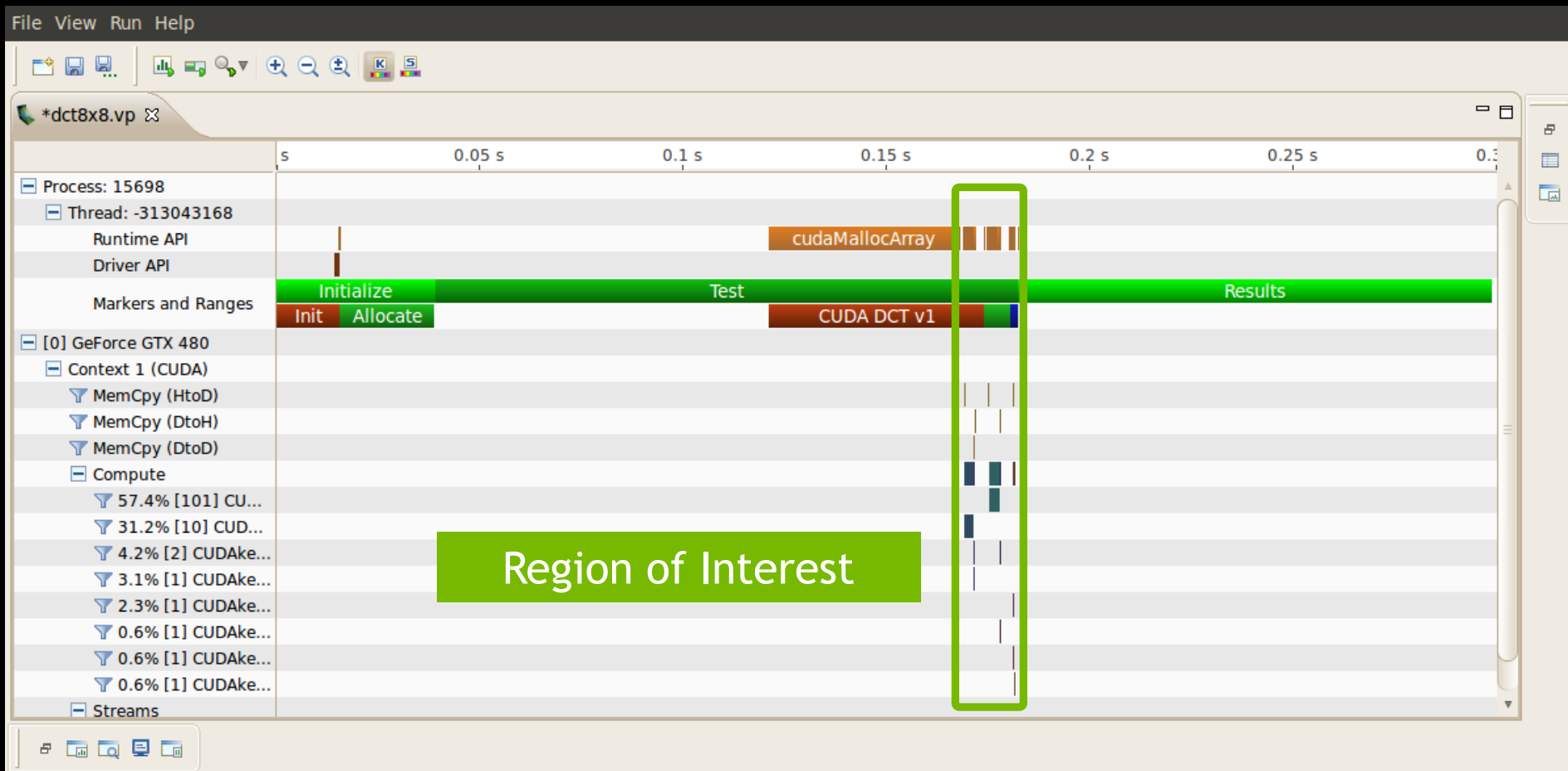
```
for (i = 0; i < N; i++) {  
    if (i == 12) cudaProfilerStart();  
    <loop body>  
    if (i == 15) cudaProfilerStop();  
}
```

Enable Region Of Interest

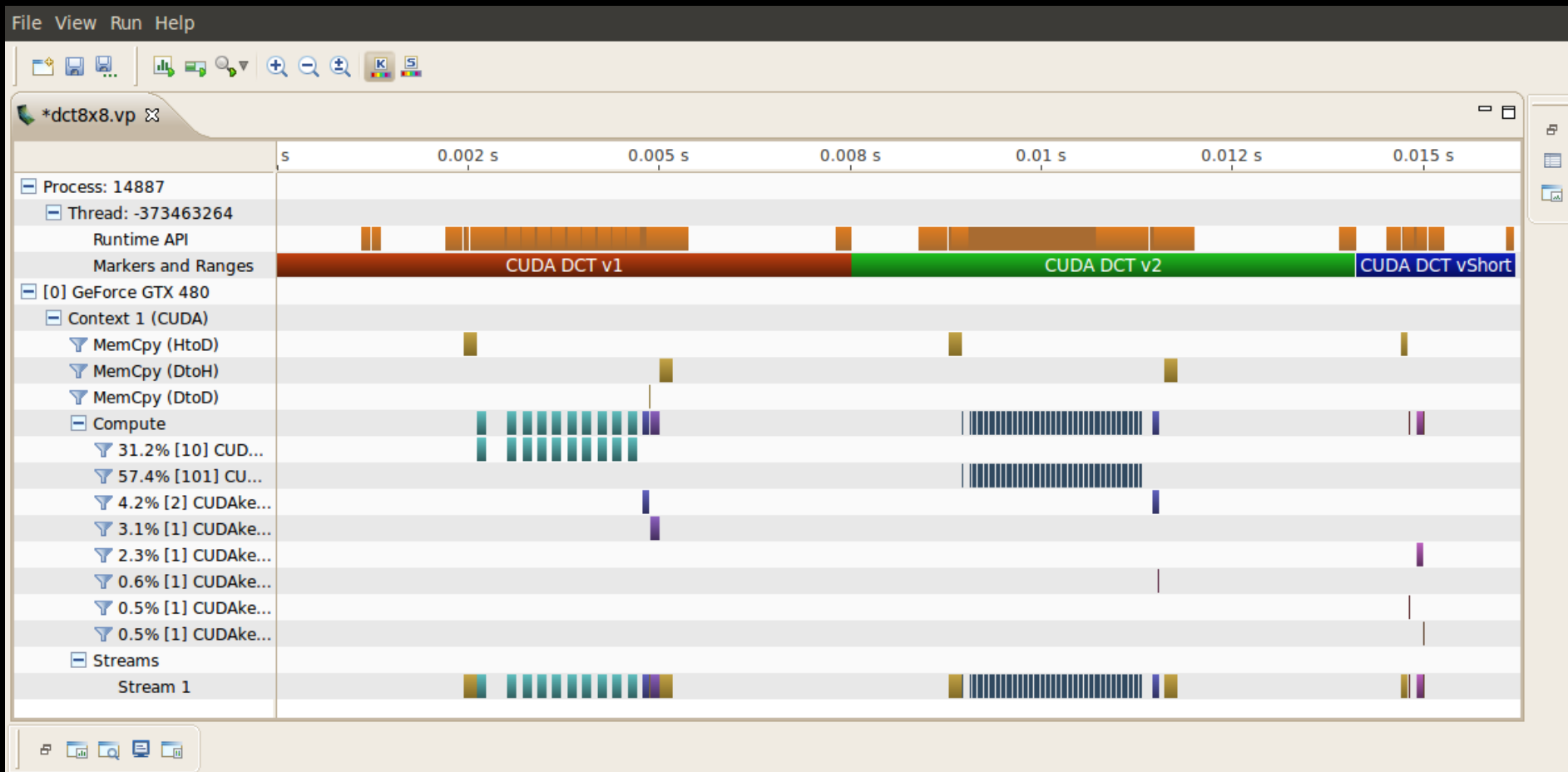
- Insert `cudaProfilerStart()` / `cudaProfilerStop()`
- Disable profiling at start of application



Example: Without cudaProfilerStart/Stop

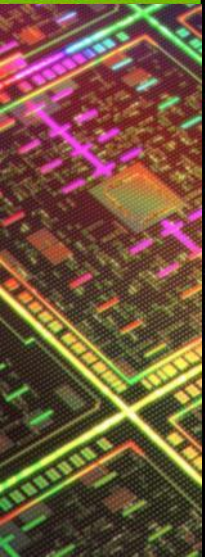


Example: With cudaProfilerStart/Stop



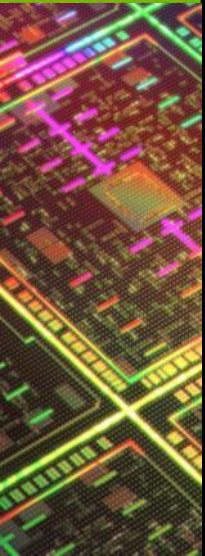
Analysis

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



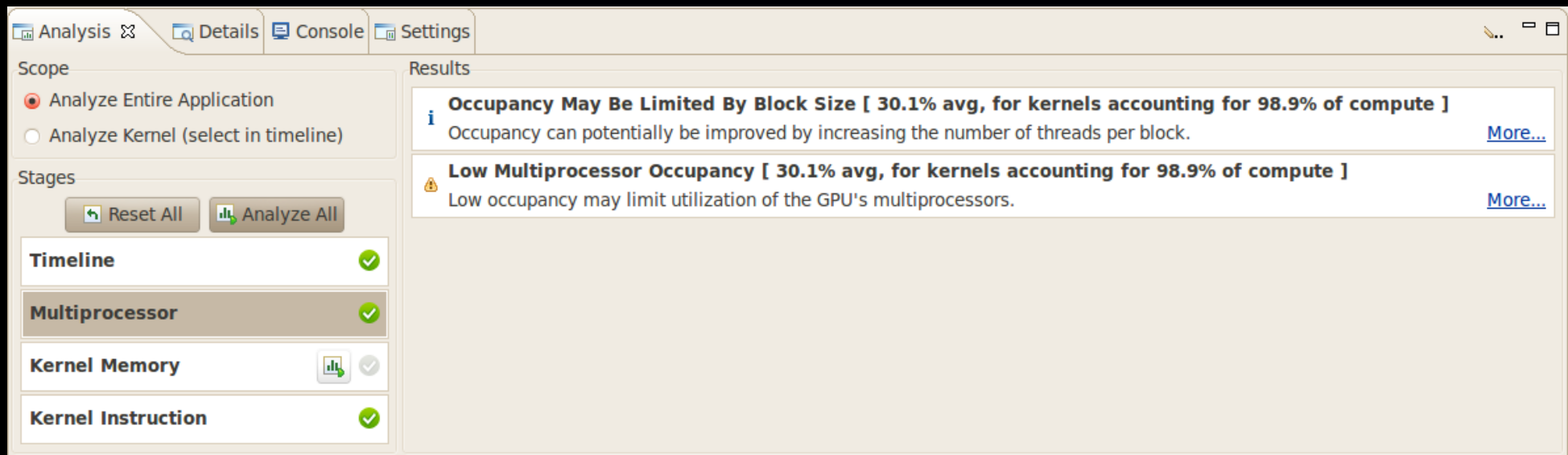
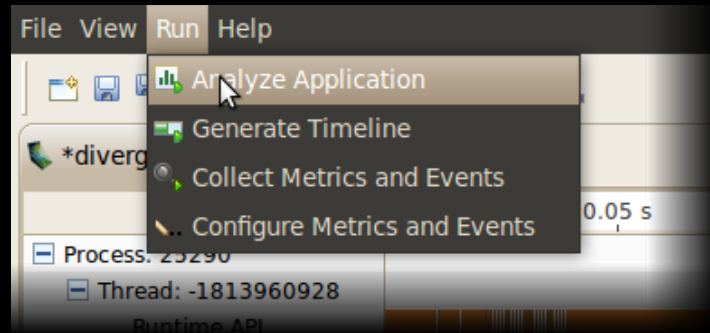
Visual Inspection

- 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




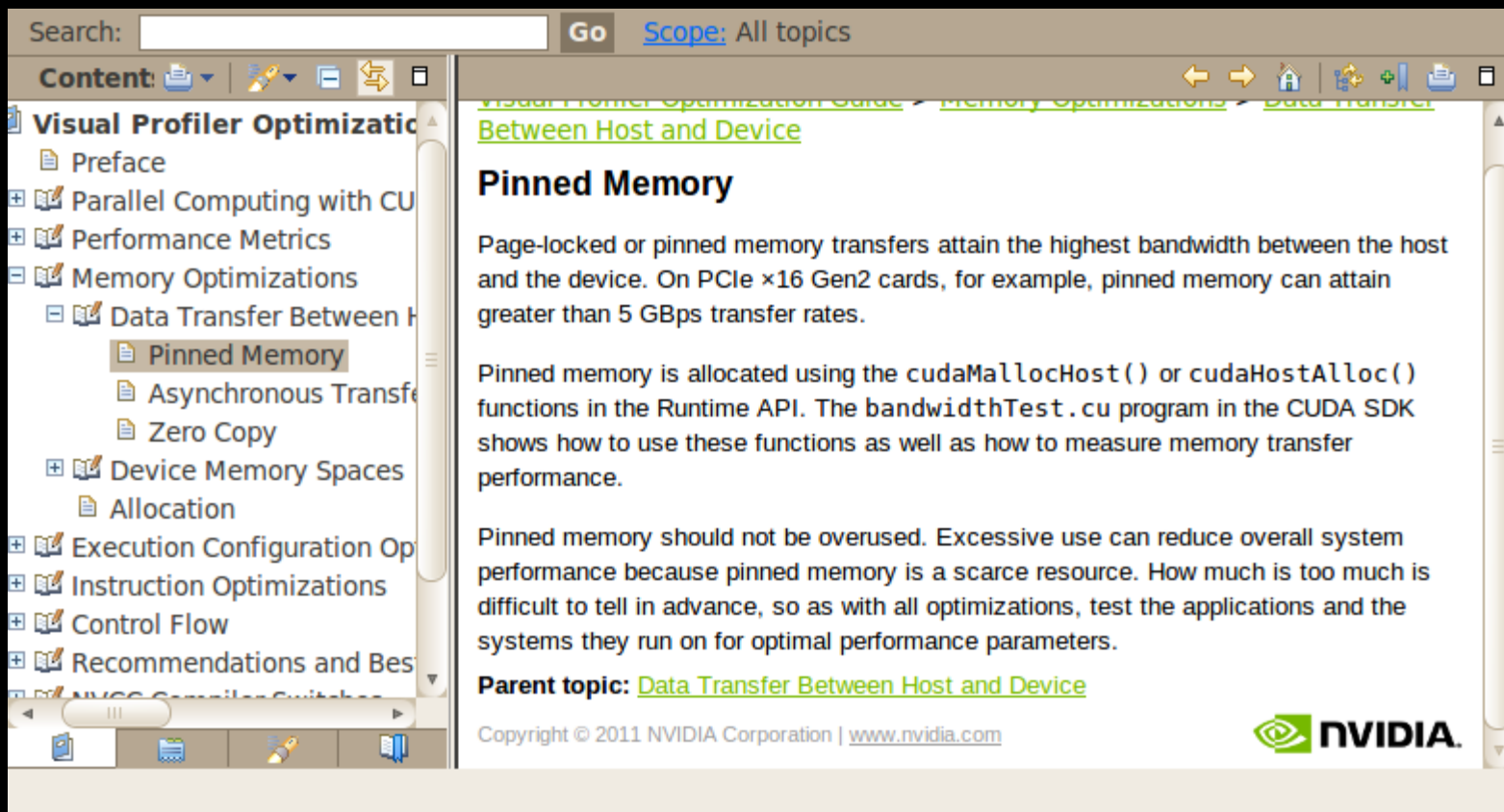
Automated Analysis - Application

- Analyze entire application
 - Timeline
 - Hardware performance counters







Analysis Documentation

 **Low Memcpy Throughput [997.19 MB/s avg, for memcpy accounting for 68.1% of all memcpy time]**
The memory copies are not fully using the available host to device bandwidth. [More..](#)



Search: Go [Scope: All topics](#)

Content:    

- Visual Profiler Optimization Guide
 - Preface
 - Parallel Computing with CUDA
 - Performance Metrics
 - Memory Optimizations
 - Data Transfer Between Host and Device
 - Pinned Memory**
 - Asynchronous Transfers
 - Zero Copy
 - Device Memory Spaces
 - Allocation
 - Execution Configuration Optimizations
 - Instruction Optimizations
 - Control Flow
 - Recommendations and Best Practices
 - NVCC Compiler Switches

[Visual Profiler Optimization Guide](#) > [Memory Optimizations](#) > [Data Transfer Between Host and Device](#)

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 `cudaHostAlloc()` 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.

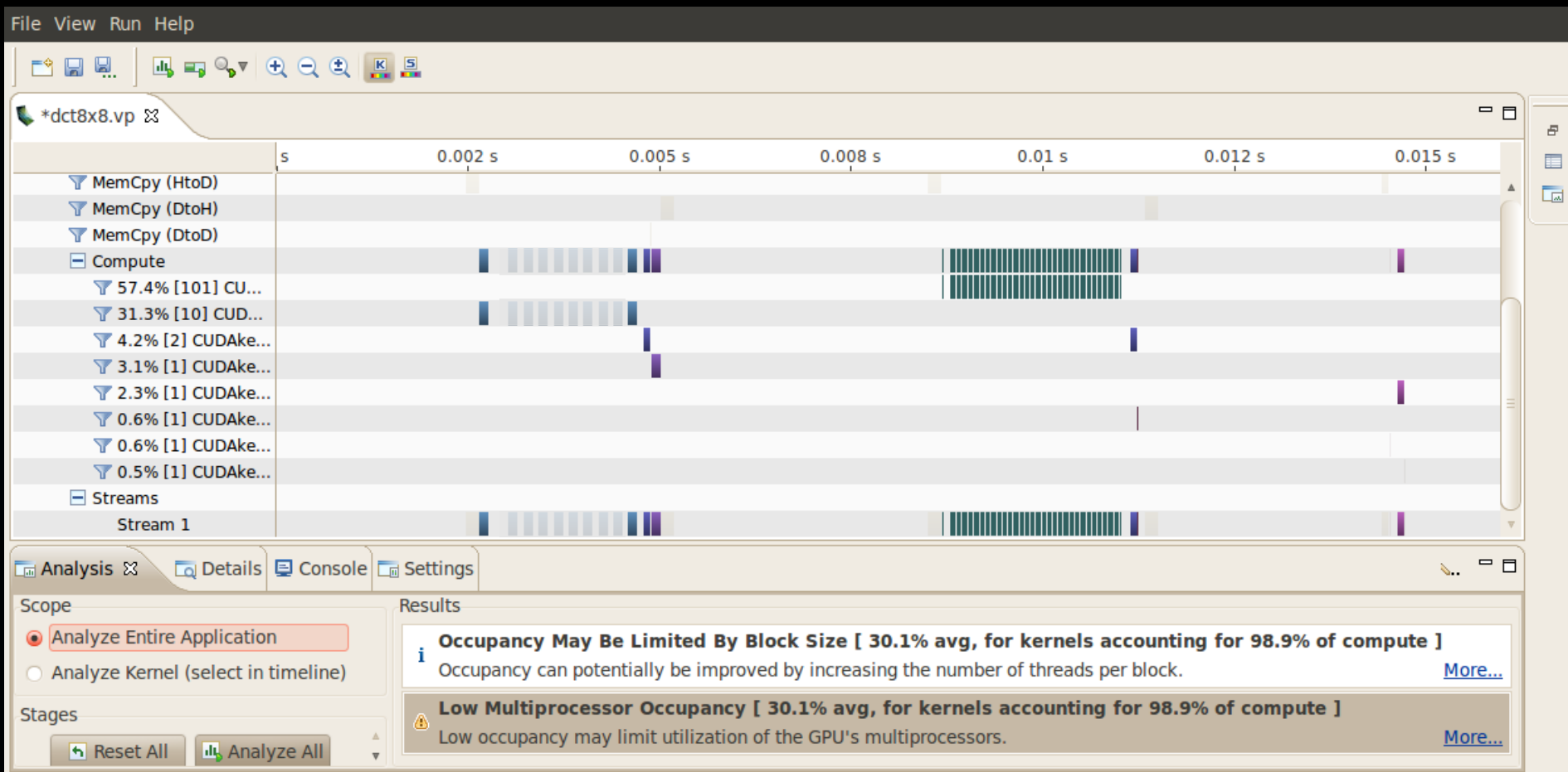
Pinned memory should not be overused. Excessive use can reduce overall system performance because pinned memory is a scarce resource. How much is too much is difficult to tell in advance, so as with all optimizations, test the applications and the systems they run on for optimal performance parameters.

Parent topic: [Data Transfer Between Host and Device](#)

Copyright © 2011 NVIDIA Corporation | www.nvidia.com



Results Correlated With Timeline



Analysis Properties

- Highlight a kernel or memcopy in timeline
 - Properties shows analysis results for that specific kernel / memcopy
 - Optimization opportunities are flagged

Name	Value
Duration	21.117 μ s
Grid Size	[16,32,1]
Block Size	[8,4,2]
Registers/Thread	35
Shared Memory/Block	2.062 KB
Memory	
Global Load Efficiency	100%
Global Store Efficiency	100%
Instruction	
Branch Divergence Overhead	0%
Occupancy	
Achieved	⚠ 29.4%
Theoretical	33.3%
Limiter	Block Size
L1 Cache Configuration	
Shared Memory Requested	48 KB
Shared Memory Executed	48 KB

Automated Analysis - Single Kernel

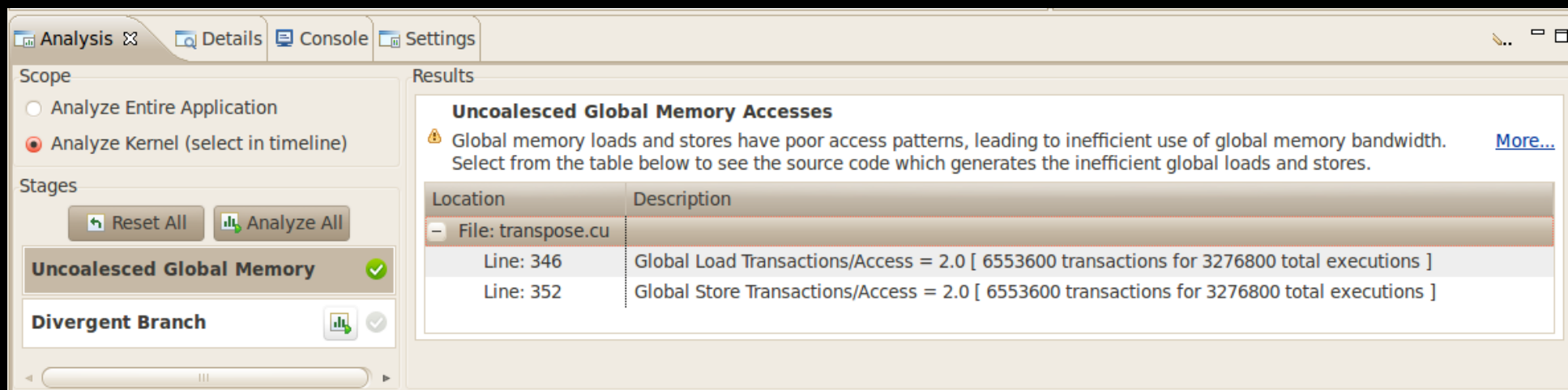
The screenshot displays a GPU analysis tool interface. The top section shows a timeline of execution for two files: *dct8x8.vp and *transpose.vp. The timeline is divided into segments representing different stages of execution, such as MemCpy (HtoD), MemCpy (DtoH), and Compute. The Compute stage is further broken down into sub-steps like coprocessor and transpose operations. The bottom section shows the 'Analysis' settings, where 'Analyze Kernel (select in timeline)' is selected. The 'Results' panel displays the following data for the kernel `transposeCoalesced(float*, float*, int, int, int)`:

Name	Value
Start	288.547 ms
End	318.947 ms
Duration	30.4 ms
Grid Size	[64,64,1]
Block Size	[16,16,1]
Registers/Thread	20
Shared Memory/Block	1 KB
Occupancy	
Theoretical	100%
L1 Cache Configuration	

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



The screenshot displays the GPU Visual Profiler interface. The left sidebar shows the 'Scope' section with 'Analyze Kernel (select in timeline)' selected, and the 'Stages' section with 'Uncoalesced Global Memory' checked. The main 'Results' pane is titled 'Uncoalesced Global Memory Accesses' and contains a warning icon and text: 'Global memory loads and stores have poor access patterns, leading to inefficient use of global memory bandwidth. Select from the table below to see the source code which generates the inefficient global loads and stores. [More...](#)'

Location	Description
File: transpose.cu	
Line: 346	Global Load Transactions/Access = 2.0 [6553600 transactions for 3276800 total executions]
Line: 352	Global Store Transactions/Access = 2.0 [6553600 transactions for 3276800 total executions]

Source Correlation

The screenshot displays a GPU profiler interface with the following components:

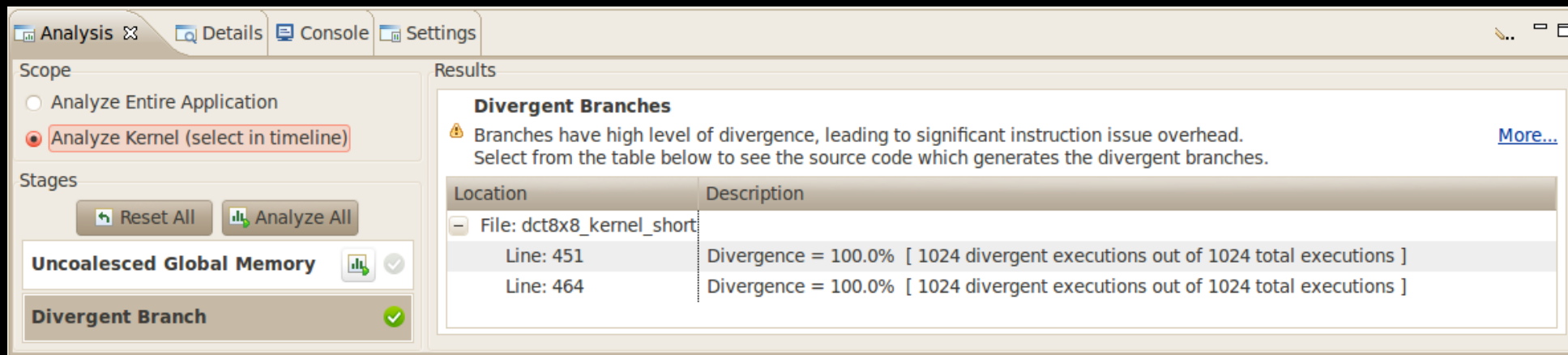
- Code Editor:** Shows C++ code for a transpose kernel. The line `tile[threadIdx.y+i][threadIdx.x] = idata[index_in+i*width];` is highlighted in blue.
- Properties Panel:** Shows details for the `transposeCoalesced(float*, float*, int, int, int)` kernel.

Name	Value
Start	288.547 ms
End	318.947 ms
Duration	30.4 ms
Grid Size	[64,64,1]
Block Size	[16,16,1]
Registers/Thread	20
Shared Memory/Block	1 KB
Occupancy	
Theoretical	100%
L1 Cache Configuration	
- Analysis Panel:** Shows the analysis scope set to "Analyze Kernel (select in timeline)" and the "Uncoalesced Global Memory" analysis stage selected with a green checkmark.
- Results Panel:** Displays "Uncoalesced Global Memory Accesses" with a warning icon. It includes a message: "Global memory loads and stores have poor access patterns, leading to inefficient use of global memory bandwidth. [More...](#)" and a table of results.

Location	Description
File: transpose	
Line: 268	Global Load Transactions/Access = 2.0 [6553600 transactions for 3276800 total executions]
Line: 274	Global Store Transactions/Access = 2.0 [6553600 transactions for 3276800 total executions]

Divergent Branches

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



The screenshot shows a software interface for GPU analysis. The left sidebar contains a 'Scope' section with 'Analyze Kernel (select in timeline)' selected, and a 'Stages' section with 'Uncoalesced Global Memory' and 'Divergent Branch' checked. The main 'Results' pane displays a warning about divergent branches and a table of specific locations.

Scope

- Analyze Entire Application
- Analyze Kernel (select in timeline)

Stages

-
-
- Uncoalesced Global Memory**
- Divergent Branch**

Results

Divergent Branches

⚠ Branches have high level of divergence, leading to significant instruction issue overhead. [More...](#)
Select from the table below to see the source code which generates the divergent branches.

Location	Description
- File: dct8x8_kernel_short	
Line: 451	Divergence = 100.0% [1024 divergent executions out of 1024 total executions]
Line: 464	Divergence = 100.0% [1024 divergent executions out of 1024 total executions]

Source Correlation

The screenshot displays the NVIDIA Visual Profiler interface. The main window shows the source code for a CUDA kernel named `dct8x8_kernel_short.cu`. A specific section of the code is highlighted, showing a loop with a `#pragma unroll` and a `for` loop that iterates over `blockSize`. The code includes `__syncthreads()` and `CUDAshortInplaceDCT` calls.

On the right side, the **Properties** window shows the following details for the `CUDAKernelShortDCT(short*, int)` kernel:

Name	Value
Start	30.872 ms
End	31.062 ms
Duration	189.663 μ s
Grid Size	[16,16,1]
Block Size	[8,4,4]
Registers/Thread	45
Shared Memory/Block	2.125 KB
Occupancy	
Theoretical	41.7%
L1 Cache Configuration	

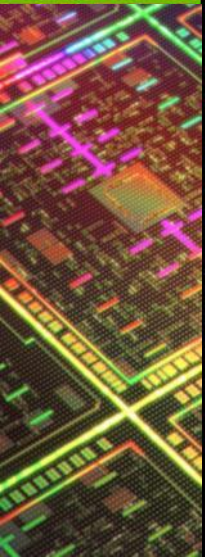
At the bottom, the **Results** panel shows **Divergent Branches**. A warning icon indicates that branches have a high level of divergence, leading to significant instruction issue overhead. The table below provides details on the divergent branches:

Location	Description
File: <code>dct8x8_kernel_short</code>	
Line: 451	Divergence = 100.0% [1024 divergent executions out of 1024 total executions]
Line: 464	Divergence = 100.0% [1024 divergent executions out of 1024 total executions]

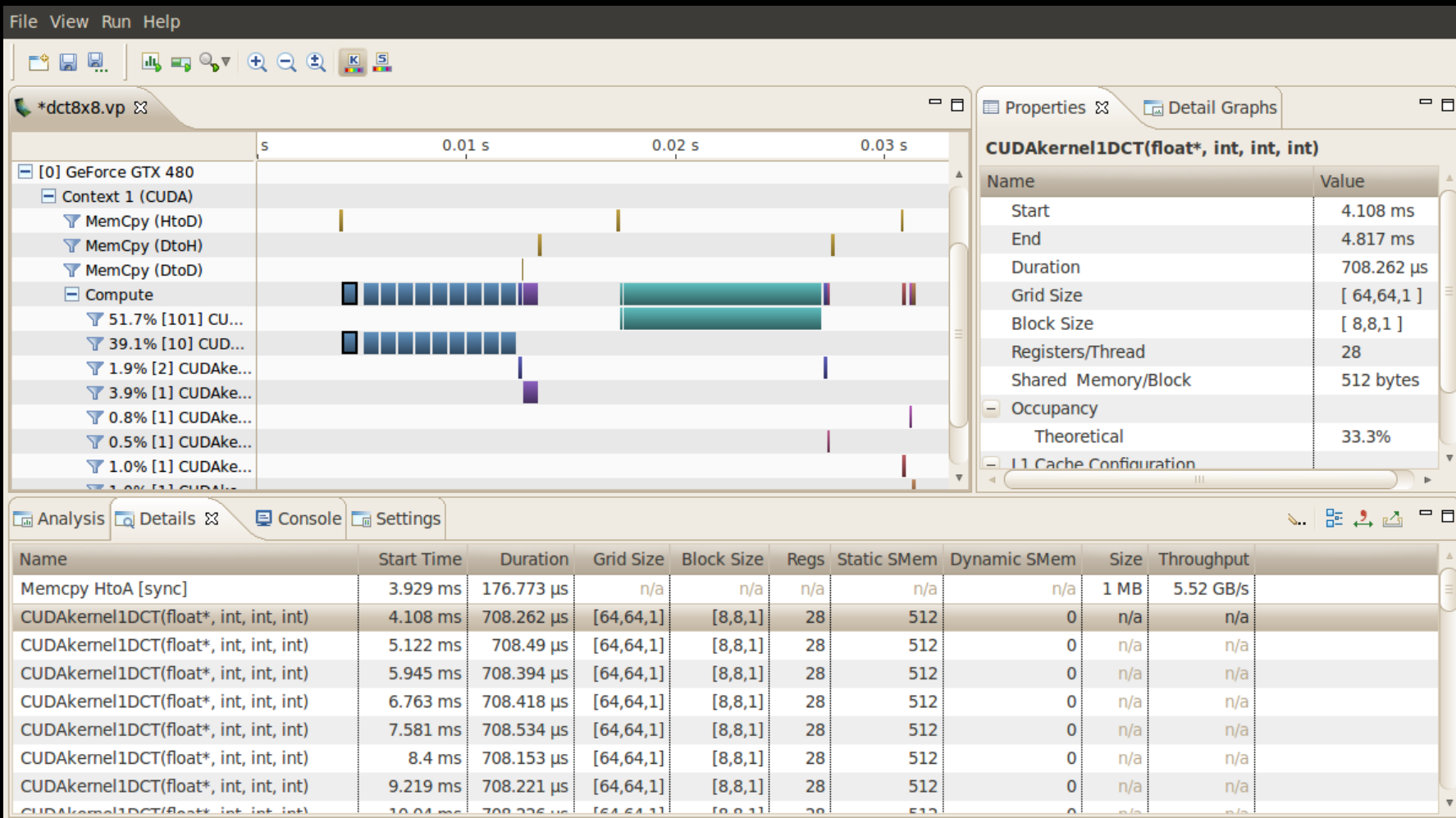
The left sidebar shows the analysis scope set to **Analyze Kernel (select in timeline)** and the **Divergent Branch** analysis is enabled.

Enabling Source Correlation

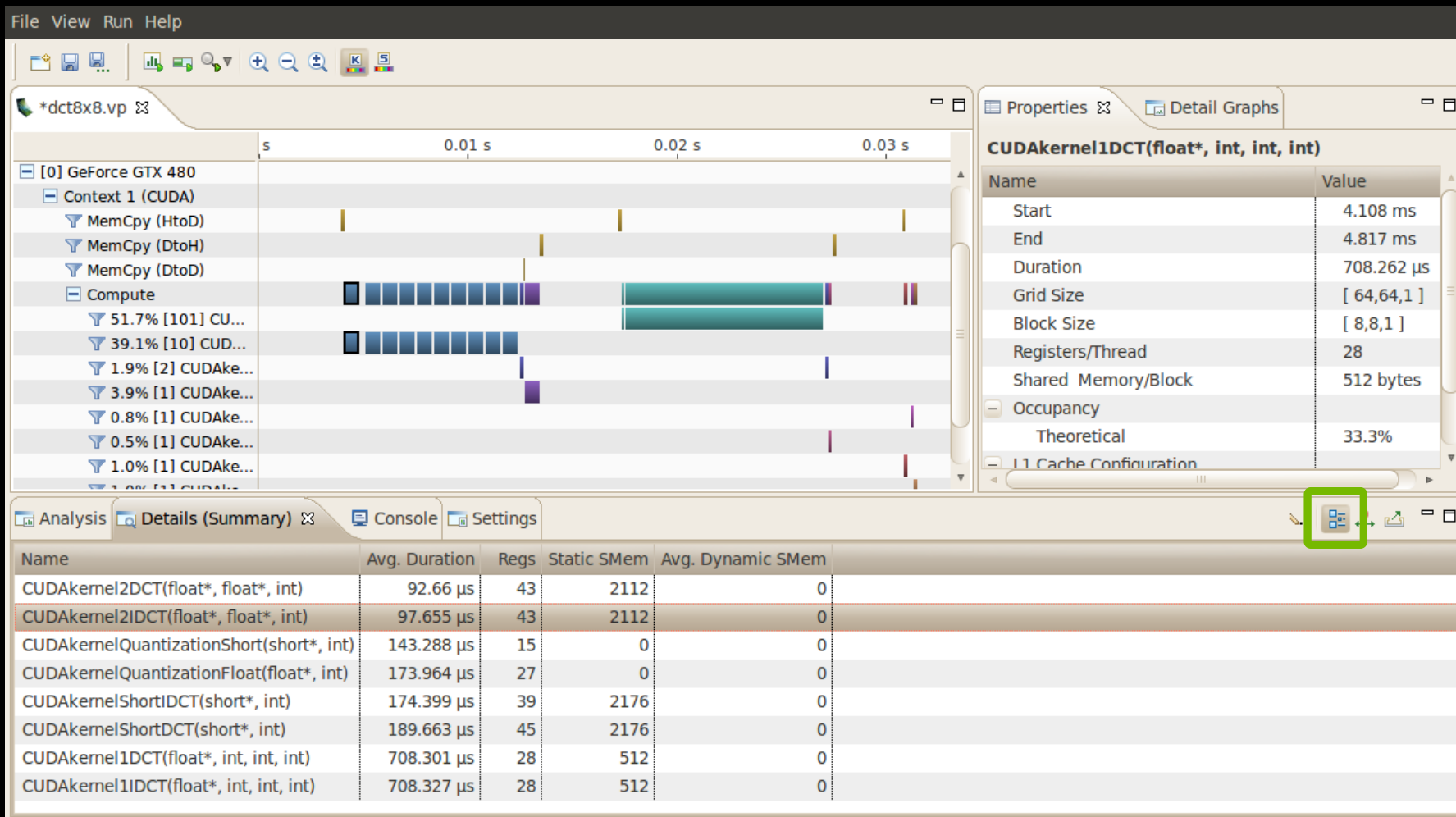
- Source correlation requires that source/line information be embedded in executable
 - Available in debug executables: `nvcc -G`
 - New flag for optimized executables: `nvcc -lineinfo`



Detailed Profile Data



Detailed Summary Profile Data

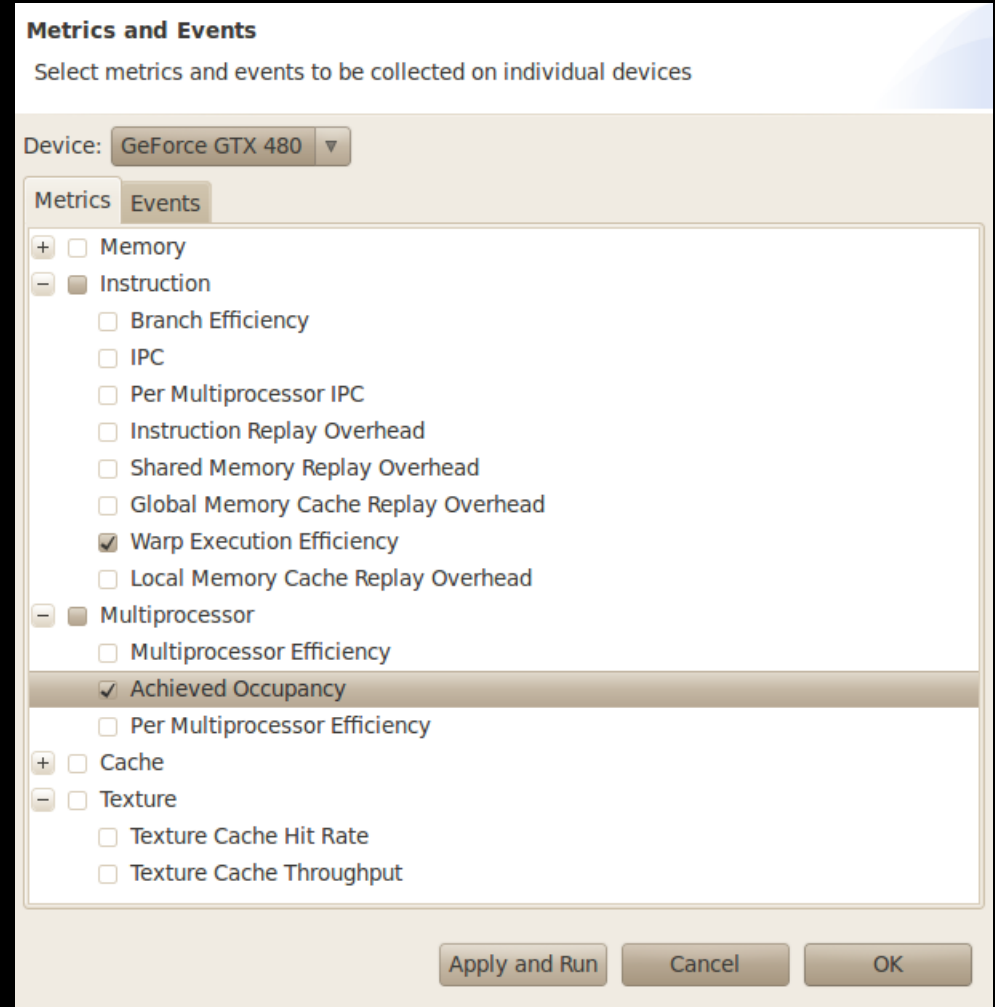
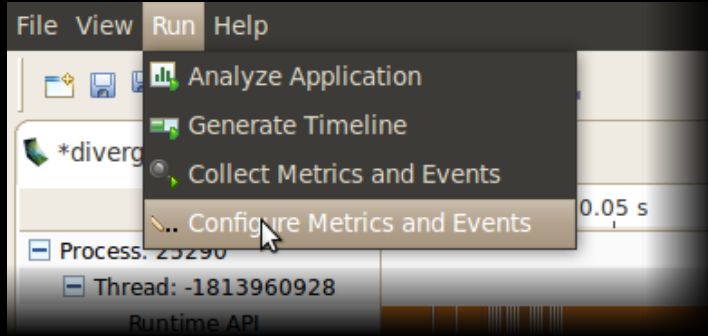


Filtering

The screenshot shows the NVIDIA Visual Profiler interface. The left-hand tree view is expanded to show the 'Compute' section, which is highlighted with a green box. The main timeline displays GPU activity for a GeForce GTX 480, with two teal bars indicating compute activity between approximately 0.015s and 0.025s. The bottom panel shows a table of kernel execution details.

Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem
CUDAkernel2DCT(float*, float*, int)	17.343 ms	94.467 μs	[16,32,1]	[8,4,2]	43	2112	0
CUDAkernel2DCT(float*, float*, int)	17.548 ms	93.809 μs	[16,32,1]	[8,4,2]	43	2112	0
CUDAkernel2DCT(float*, float*, int)	17.643 ms	92.553 μs	[16,32,1]	[8,4,2]	43	2112	0
CUDAkernel2DCT(float*, float*, int)	17.737 ms	92.614 μs	[16,32,1]	[8,4,2]	43	2112	0
CUDAkernel2DCT(float*, float*, int)	17.831 ms	92.647 μs	[16,32,1]	[8,4,2]	43	2112	0
CUDAkernel2DCT(float*, float*, int)	17.925 ms	92.994 μs	[16,32,1]	[8,4,2]	43	2112	0
CUDAkernel2DCT(float*, float*, int)	18.019 ms	92.805 μs	[16,32,1]	[8,4,2]	43	2112	0
CUDAkernel2DCT(float*, float*, int)	18.113 ms	92.582 μs	[16,32,1]	[8,4,2]	43	2112	0
CUDAkernel2DCT(float*, float*, int)	18.207 ms	92.510 μs	[16,32,1]	[8,4,2]	43	2112	0

Metrics and Events



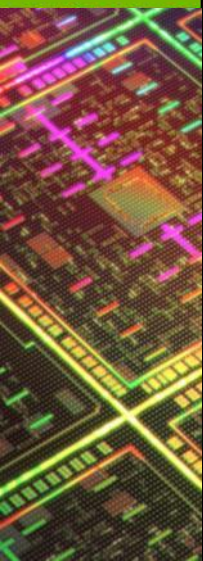
Metrics and Events

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

Name	Warp Execution Efficiency	Achieved Occupancy	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem
CUDAkernel2DCT(float*, float*, int)	100%	0.3	92.66 µs	43	2112	0
CUDAkernel2IDCT(float*, float*, int)	100%	0.302	97.655 µs	43	2112	0
CUDAkernelQuantizationShort(short*, int)	67.5%	0.317	143.288 µs	15	0	0
CUDAkernelQuantizationFloat(float*, int)	98.7%	0.318	173.964 µs	27	0	0
CUDAkernelShortIDCT(short*, int)	74.7%	0.468	174.399 µs	39	2176	0
CUDAkernelShortDCT(short*, int)	75%	0.376	189.663 µs	45	2176	0
CUDAkernel1DCT(float*, int, int, int)	100%	0.328	708.301 µs	28	512	0
CUDAkernel1IDCT(float*, int, int, int)	100%	0.328	708.327 µs	28	512	0

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
```

```
==== Profiling result:
```

Time(%)	Time	Calls	Avg	Min	Max	Name
49.52	9.36ms	101	92.68us	92.31us	94.31us	CUDAkernel2DCT(float*, float*, int)
37.47	7.08ms	10	708.31us	707.99us	708.50us	CUDAkernel1DCT(float*,int, int,int)
3.75	708.42us	1	708.42us	708.42us	708.42us	CUDAkernel1IDCT(float*,int,int,int)
1.84	347.99us	2	173.99us	173.59us	174.40us	CUDAkernelQuantizationFloat()
1.75	331.37us	2	165.69us	165.67us	165.70us	[CUDA memcpy DtoH]
1.41	266.70us	2	133.35us	89.70us	177.00us	[CUDA memcpy HtoD]
1.00	189.64us	1	189.64us	189.64us	189.64us	CUDAkernelShortDCT(short*, int)
0.94	176.87us	1	176.87us	176.87us	176.87us	[CUDA memcpy HtoA]
0.92	174.16us	1	174.16us	174.16us	174.16us	CUDAkernelShortIDCT(short*, int)
0.76	143.31us	1	143.31us	143.31us	143.31us	CUDAkernelQuantizationShort(short*)
0.52	97.75us	1	97.75us	97.75us	97.75us	CUDAkernel2IDCT(float*, float*)
0.12	22.59us	1	22.59us	22.59us	22.59us	[CUDA memcpy DtoA]

nvprof - GPU Summary (csv)

```
$ nvprof --csv dct8x8
```

```
==== Profiling result:
```

```
Time(%),Time,Calls,Avg,Min,Max,Name
```

```
,ms,,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 memcpy DtoH]"
1.42,0.26759,2,133.7970,89.89100,177.7030,"[CUdA memcpy HtoD]"
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 memcpy HtoA]"
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 memcpy DtoA]"
```

nvprof - GPU Trace

```
$ nvprof --print-gpu-trace dct8x8
```

```
===== Profiling result:
```

Start	Duration	Grid Size	Block Size	Regs	SSMem	DSMem	Size	Throughput	Name
167.82ms	176.84us	-	-	-	-	-	1.05MB	5.93GB/s	[CUDA memcpy HtoA]
168.00ms	708.51us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
168.95ms	708.51us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
169.74ms	708.26us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
170.53ms	707.89us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
171.32ms	708.12us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
172.11ms	708.05us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
172.89ms	708.38us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
173.68ms	708.31us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
174.47ms	708.15us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
175.26ms	707.95us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAKernel1DCT(float*, ...)
176.05ms	173.87us	(64 64 1)	(8 8 1)	27	0B	0B	-	-	CUDAKernelQuantization (...)
176.23ms	22.82us	-	-	-	-	-	1.05MB	45.96GB/s	[CUDA memcpy DtoA]

nvprof - CPU/GPU Trace

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

```
===== Profiling result:
```

Start	Duration	Grid Size	Block Size	Regs	SSMem	DSMem	Size	Throughput	Name
167.82ms	176.84us	-	-	-	-	-	1.05MB	5.93GB/s	[CUDA memcpy HtoA]
167.81ms	2.00us	-	-	-	-	-	-	-	cudaSetupArgument
167.81ms	38.00us	-	-	-	-	-	-	-	cudaLaunch
167.85ms	1.00ms	-	-	-	-	-	-	-	cudaDeviceSynchronize
168.00ms	708.51us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAkernel1DCT(float*, ...)
168.86ms	2.00us	-	-	-	-	-	-	-	cudaConfigureCall
168.86ms	1.00us	-	-	-	-	-	-	-	cudaSetupArgument
168.86ms	1.00us	-	-	-	-	-	-	-	cudaSetupArgument
168.86ms	1.00us	-	-	-	-	-	-	-	cudaSetupArgument
168.87ms	0ns	-	-	-	-	-	-	-	cudaSetupArgument
168.87ms	24.00us	-	-	-	-	-	-	-	cudaLaunch
168.89ms	761.00us	-	-	-	-	-	-	-	cudaDeviceSynchronize
168.95ms	708.51us	(64 64 1)	(8 8 1)	28	512B	0B	-	-	CUDAkernel1DCT(float*, ...)

nvprof - Event Query

```
$ nvprof --devices 0 --query-events
```

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

	Name	Description
Device 0:		
Domain domain_a:		
	sm_cta_launched:	Number of thread blocks launched on a multiprocessor.
	l1_local_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_local_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_local_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
```

```
==== Profiling result:
```

Device	Invocations	Avg	Min	Max	Event Name
Device 0					
Kernel: CUDAkernel1IDCT(float*, int, int, int)					
	1	475136	475136	475136	branch
	1	0	0	0	divergent_branch
Kernel: CUDAkernelQuantizationFloat(float*, int)					
	2	180809	180440	181178	branch
	2	6065	6024	6106	divergent_branch
Kernel: CUDAkernel1DCT(float*, int, int, int)					
	10	475136	475136	475136	branch
	10	0	0	0	divergent_branch
Kernel: CUDAkernelShortIDCT(short*, int)					
	1	186368	186368	186368	branch
	1	2048	2048	2048	divergent_branch
Kernel: CUDAkernel2IDCT(float*, float*, int)					
	1	61440	61440	61440	branch
	1	0	0	0	divergent_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

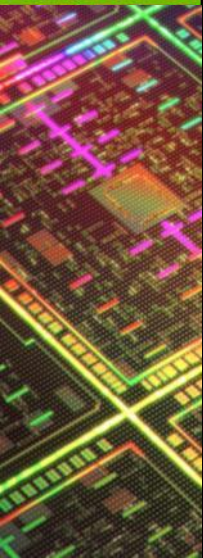
```
$ nvprof -i profile.out
```

```
$ 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?