

Debugging and Performance Tools

Peter Messmer



Debugging via printf



- printf supported on devices of sm_20 and higher
- Requires inclusion of “stdio.h”
- Caution:
 - Fixed buffer size
 - Flushed under certain circumstances
 - E.g. next time a kernel is launched
 - Not flushed by default at end of application
 - Forced eg. via cudaDeviceReset()

Debugging via cuda-gdb



- Compile with `-g -G` options
- Use `-gencode` option
`nvcc -g -G -gencode arch=compute_35,code=sm_35`
- run via `cuda-gdb myapp`
- Determining focus:
`(cuda-gdb) cuda device sm warp lane block thread`
- Breakpoint
- `(cuda-gdb) break my_file.cu:185`

Debugging via cuda-memcheck

- Useful in case of “unspecified launch failure”
 - Out-of-bound access, memory leaks
- Does not require recompilation
- More precise information if compiled with flags:
 - G -lineinfo -rdynamic
- racecheck to detect race conditions
 - cuda-memcheck -tool racecheck myapp.x

Performance Analysis with Cmd-Line Profiler



- Basic trace of kernel launches and data transfers
`export CUDA_PROFILE=1`
- Additional env variables for e.g visualization of parallel run in NVVP
`export COMPUTE_PROFILE_CSV=1`
`export COMPUTE_PROFILE_LOG="profile_%p.csv"`
`export COMPUTE_PROFILE_CONFIG=profile.cfg`
- Signals to collect specified in configuration file "profile.cfg"

Performance Analysis with Cmd-Line Profiler



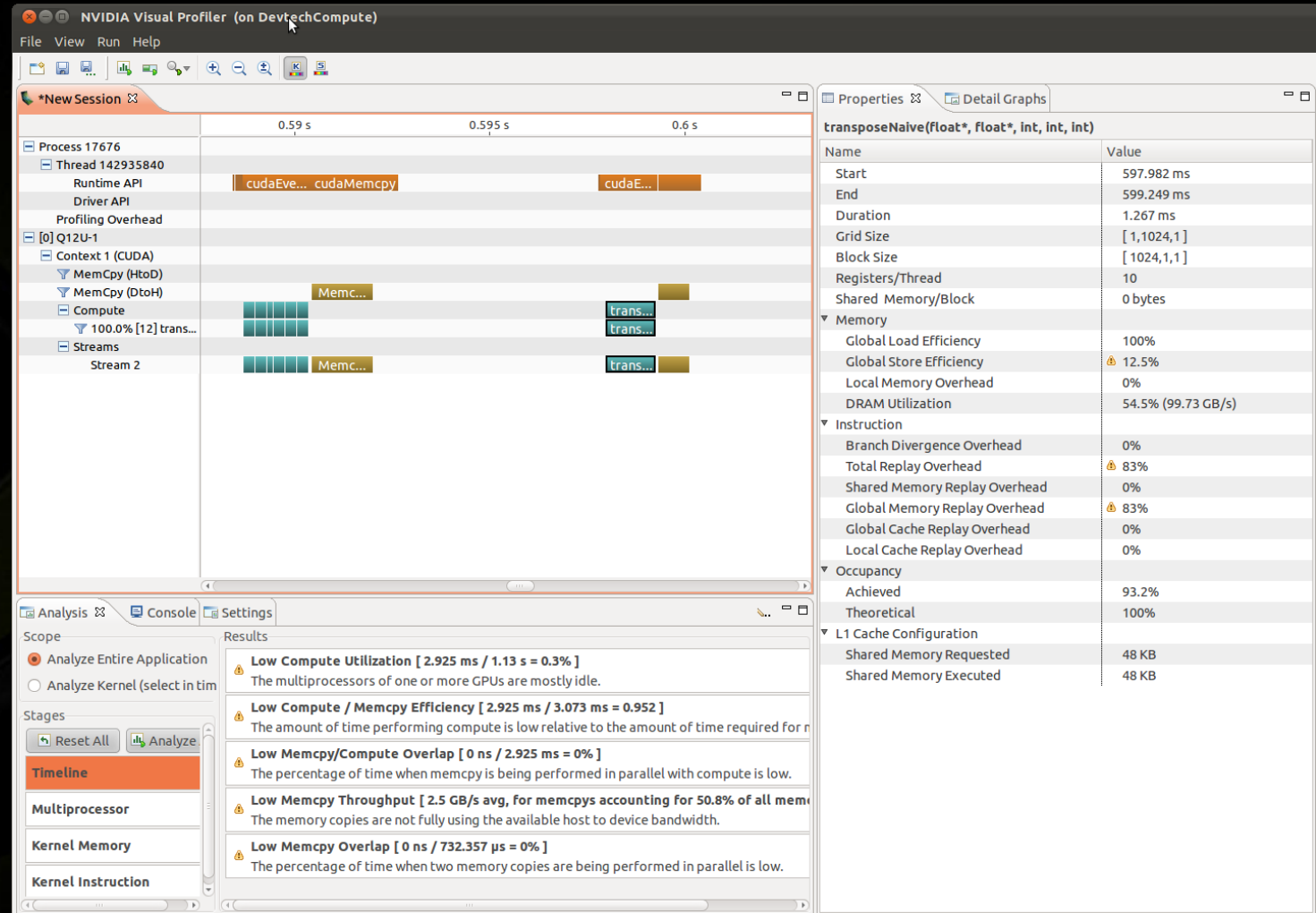
- Sample configuration file profile.cfg

```
gpustarttimestamp  
gridsize3d  
threadblocksize  
dynsmemperblock  
stasmemperblock  
regperthread  
memtransfersize  
memtransferdir  
streamid  
countermodeaggregate  
active_warps  
active_cycles
```

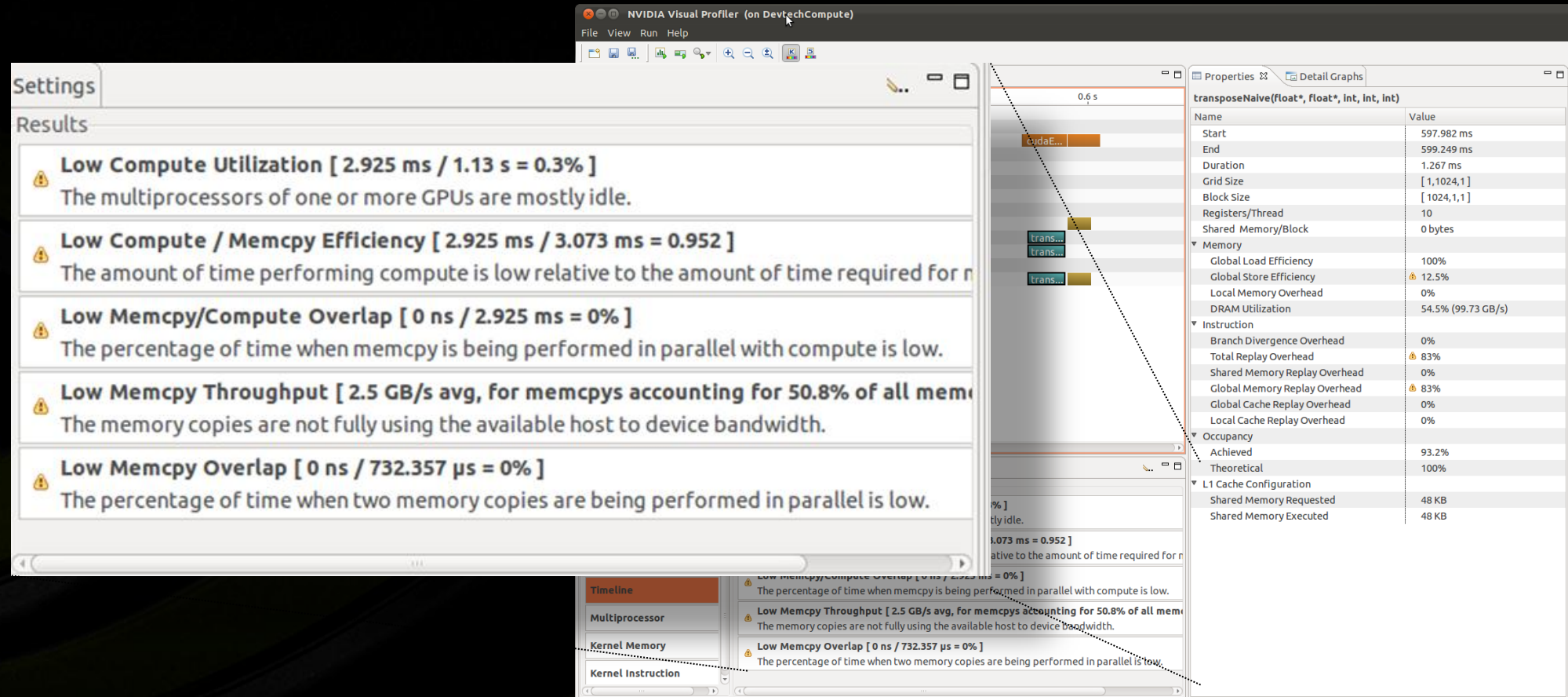
NVVP – NVIDIA Visual Profiler



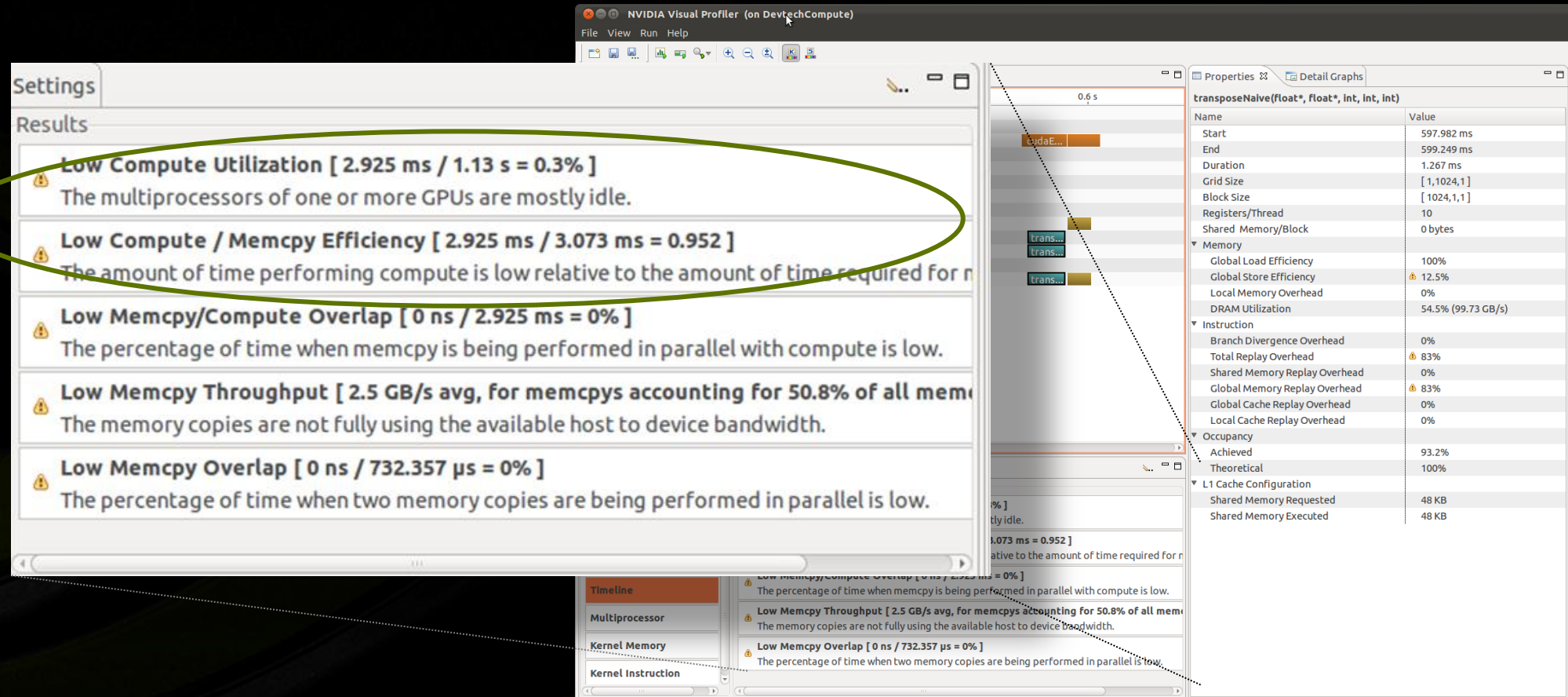
- Application analysis
- Kernel properties



Application Assessment with NVVP



Application Assessment with NVVP



Application Assessment with NVVP



Start	597.982 ms
End	599.249 ms
Duration	1.267 ms
Grid Size	[1,1024,1]
Block Size	[1024,1,1]
Registers/Thread	10
Shared Memory/Block	0 bytes
Memory	
Global Load Efficiency	100%
Global Store Efficiency	⚠ 12.5%
Local Memory Overhead	0%
DRAM Utilization	54.5% (99.73 GB/s)
Instruction	
Branch Divergence Overhead	0%
Total Replay Overhead	⚠ 83%
Shared Memory Replay Overhead	0%
Global Memory Replay Overhead	⚠ 83%
Global Cache Replay Overhead	0%
Local Cache Replay Overhead	0%
Occupancy	
Achieved	93.2%
Theoretical	100%

The screenshot shows the NVVP application assessment interface. On the left, a timeline graph displays the execution of the 'transposeNaive' kernel, with a duration of 0.6 s. On the right, the 'Properties' window is open, showing the following details:

Name	Value
Start	597.982 ms
End	599.249 ms
Duration	1.267 ms
Grid Size	[1,1024,1]
Block Size	[1024,1,1]
Registers/Thread	10
Shared Memory/Block	0 bytes
Memory	
Global Load Efficiency	100%
Global Store Efficiency	⚠ 12.5%
Local Memory Overhead	0%
DRAM Utilization	54.5% (99.73 GB/s)
Instruction	
Branch Divergence Overhead	0%
Total Replay Overhead	⚠ 83%
Shared Memory Replay Overhead	0%
Global Memory Replay Overhead	⚠ 83%
Global Cache Replay Overhead	0%
Local Cache Replay Overhead	0%
Occupancy	
Achieved	93.2%
Theoretical	100%
L1 Cache Configuration	
Shared Memory Requested	48 KB
Shared Memory Executed	48 KB

Source-Level Hot-spot Analysis in NVVP



NVIDIA Visual Profiler (on DevtechCompute)

File View Help

*New Session transpose.cu

```
//  
  
__global__ void transposeNaive(float *odata, float *idata, int width, int height, int nreps)  
{  
    int xIndex = blockIdx.x * TILE_DIM_X + threadIdx.x;  
    int yIndex = blockIdx.y * TILE_DIM_Y + threadIdx.y;  
  
    int index_in = xIndex + width * yIndex;  
    int index_out = yIndex + height * xIndex;  
  
    for (int r=0; r < nreps; r++)  
    {  
        for (int i=0; i<TILE_DIM_Y; i+=BLOCK_ROWS)  
        {  
            odata[index_out+i] = idata[index_in+i*width];  
        }  
    }  
}  
  
// coalesced transpose (with bank conflicts)
```

Analysis Console Settings

Scope

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

Stages

Reset All Analyze All

Uncoalesced Global Memory ☒

Divergent Branch ☒

Results

Uncoalesced Global Memory Accesses

Global memory loads and stores have poor access patterns, leading to inefficient use of global memory bandwidth. [More...](#)

Select from the table below to see the source code which generates the inefficient global loads and stores.

Location	Description
File: transpose.cu	
Line: 142	Global Store L2 Transactions/Access = 32.0 [5242880 L2 transactions for 163840 total executions]
Line: 142	Global Store L2 Transactions/Access = 32.0 [5242880 L2 transactions for 163840 total executions]

Source-Level Hot-spot Analysis in NVVP



NVIDIA Visual Profiler (on DevtechCompute)

File View Help

*New Session transpose.cu

```
//  
_global_ void transposeNaive(float *odata, float *idata  
{  
    int xIndex = blockIdx.x * TILE_DIM_X + threadIdx.x;  
    int yIndex = blockIdx.y * TILE_DIM_Y + threadIdx.y;  
  
    int index_in = xIndex + width * yIndex;  
    int index_out = yIndex + height * xIndex;  
  
    for (int r=0; r < nreps; r++)  
    {  
        for (int i=0; i<TILE_DIM_Y; i+=BLOCK_ROWS)  
        {  
            odata[index_out+i] = idata[index_in+i*width];  
        }  
    }  
}
```

Reset All Analyze All

Uncoalesced Global Memory ✓

Divergent Branch ✓

Stages

Reset All Analyze All

Uncoalesced Global Memory

Divergent Branch

Uncoalesced Global Memory Accesses

Global memory loads and stores have poor access patterns, leading to inefficient use of global memory bandwidth. [More...](#)

Select from the table below to see the source code which generates the inefficient global loads and stores.

Location	Description
File: transpose.cu	
Line: 142	Global Store L2 Transactions/Access = 32.0 [5242880 L2 transactions for 163840 total executions]
Line: 142	Global Store L2 Transactions/Access = 32.0 [5242880 L2 transactions for 163840 total executions]

Additional Metrics



Start	588.755 ms
End	588.808 ms
Duration	53.344 μ s
Grid Size	[64,64,1]
Block Size	[16,8,1]
Registers/Thread	21
Shared Memory/Block	1.062 KB
Memory	
Global Load Efficiency	100%
Global Store Efficiency	100%
Local Memory Overhead	0%
DRAM Utilization	92.7% (169.74 GB/s)
Instruction	
Branch Divergence Overhead	0%
Total Replay Overhead	17.6%
Shared Memory Replay Overhead	0%
Global Memory Replay Overhead	17.6%
Global Cache Replay Overhead	0%
Local Cache Replay Overhead	0%
Occupancy	
Achieved	91.3%
Theoretical	100%
Theoretical	100%

Alternatives to NVVP: nvprof



```
%nvprof --print-gpu-trace ./transpose
```

Profiling result:

Start	Duration	Grid Size	Block Size	Regs*	Size	Throughput	Name
577.11ms	874.57us	-	-	-	4.19MB	4.80GB/s	[CUDA memcpy HtoD]
598.45ms	1.67ms	(1 1 1)	(1024 1 1)	22	-	-	transposeNaive(float*,
600.12ms	1.67ms	(1 1 1)	(1024 1 1)	22	-	-	transposeNaive(float*,
601.79ms	1.67ms	(1 1 1)	(1024 1 1)	22	-	-	transposeNaive(float*,

```
nvprof --print-gpu-trace --aggregate-mode-off --events sm_cta_launched ./transpose
```

Profiling result:

Device	Event Name,	Kernel,	Values
0	sm_cta_launched,	transposeNaive(float*, ..),	76 73 72 72 73 74 75 73 73 72 73 73 72 73

- Command-Line Profiler
- Access to hardware counters
- List of supported counters: `--query-events`

Alternatives to NVVP: nvprof



docs.nvidia.com/cuda/profiler-users-guide/index.html

docs.nvidia.com/cuda/profiler-users-guide/index.html

DEVELOPER ZONE **CUDA TOOLKIT DOCUMENTATION**

▼ Profiler User's Guide

- Profiling Overview
- What's New
- Preparing An Application For Profiling
- Focused Profiling
- Marking Regions of CPU Activity
- Naming CPU and CUDA Resources
- Flush Profile Data
- Dynamic Parallelism
- Visual Profiler
- Getting Started
 - Modify Your Application For Profiling
 - Creating a Session

	cache misses	
gld_efficiency	Ratio of requested global memory load throughput to actual global memory load throughput	$100 * \text{gld_requested_throughput} / \text{gld_throughput}$
gst_efficiency	Ratio of requested global memory store throughput to actual global memory store throughput	$100 * \text{gst_requested_throughput} / \text{gst_throughput}$

Alternatives to NVVP: Instrumentation



```
cudaEventRecord(start, 0);  
  
transpose<<<grid, threads>>>(..);  
  
cudaEventRecord(stop, 0);  
  
cudaEventSynchronize(stop);  
  
cudaEventElapsedTime(&time, start, stop);
```

