

NEW DEVELOPER TOOLS FEATURES IN CUDA 8.0

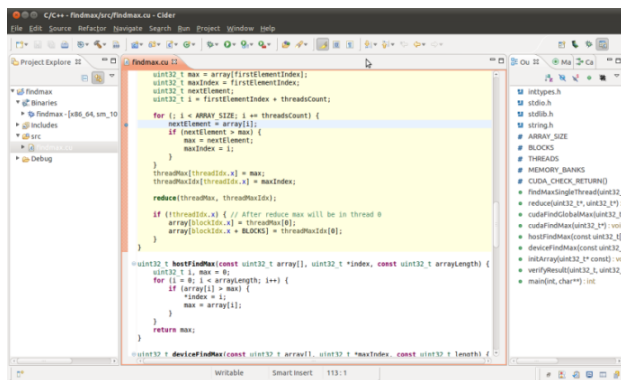
Sanjiv Satoor



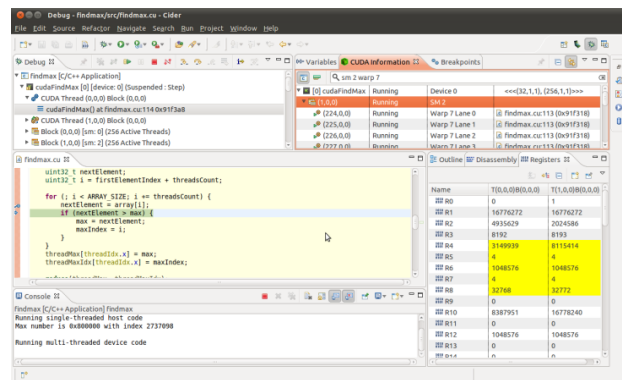
CUDA TOOLS

NVIDIA® NSIGHT™

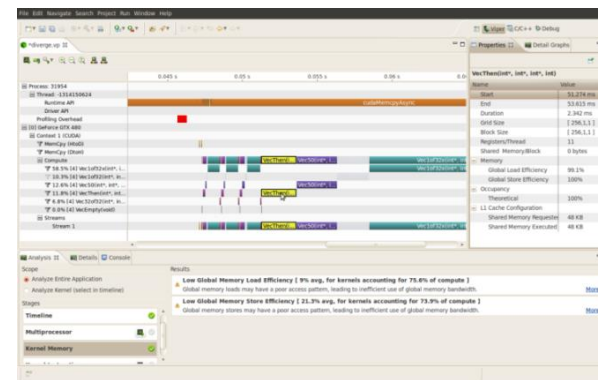
Homogeneous application development for CPU+GPU compute platforms



CUDA-Aware Editor



CUDA Debugger



CUDA Profiler

CPU+GPU



CUDA TOOLS

VISUAL PROFILER

- Trace CUDA activities
- Profile CUDA kernels
- Correlate performance instrumentation with source code
- Expert-guided performance analysis

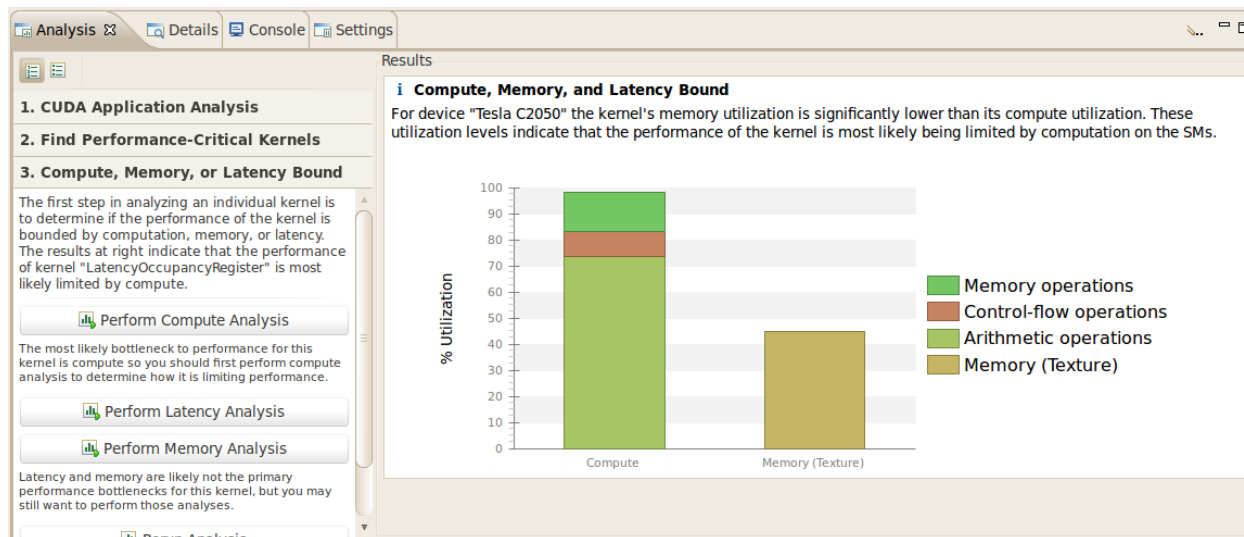
NVPROF

- Collect Performance events and metrics

GPU LIBRARY ADVISOR

- Detect CUDA library optimization opportunities

NVDISASM, CUOBJDUMP



CUDA-MEMCHECK

- Detect out-of-bounds and misaligned memory accesses
- Detect race condition in memory accesses
- Detect uninitialized global memory accesses
- Detect incorrect GPU thread synchronization

CUDA-GDB

- Debug CUDA kernels with CLI
- Debug CPU and GPU code
- CPU and GPU core dump support

NEW TOOLS FEATURES IN CUDA 8.0

SUPPORT FOR PASCAL ARCHITECTURE

ALL TOOLS

Support for GPUs with Compute Capability 6.0, 6.1 and 6.2

DEPENDENCY ANALYSIS

Visual Profiler, `nvprof`

Provide insight into application-level performance limiters

Expose dependencies between activities according to the programming model

Identify waiting time due to inter-stream dependencies

Highlight activities on the critical application runtime path

Supports CUDA (Linux/Mac/Windows) and POSIX threads (Linux/Mac)

DEPENDENCY ANALYSIS IN VISUAL PROFILER

The screenshot displays the NVIDIA Visual Profiler interface. The top section shows a timeline for a process named "matrixMul" (24278) on a Tesla K20c GPU. The timeline includes various API calls such as `cudaMalloc`, `cudaMemcpy`, `cudaDeviceSynchronize`, and `void matrixMulCUDA`. The bottom section shows the "Analysis" tab with the "Dependency Analysis" option selected. The "Results" panel displays a table of metrics for the `void matrixMulCUDA` kernel.

Function Name	Time on Critical Path (%)	Time on Critical Path	Waiting time
<code>cudaMalloc_v3020</code>	54.05 %	194.897 ms	0 ns
<Other>	45.54 %	164.223 ms	0 ns
<code>void matrixMulCUDA<int=32>(float*, float*, float*, int, int)</code>	0.29 %	1.031 ms	0 ns
<code>cudaMemcpy_v3020</code>	0.09 %	330.123 µs	124.352 µs
[CUDA memcpy DtoH]	0.03 %	124.352 µs	0 ns
<code>cudaEventRecord_v3020</code>	0.00 %	12.687 µs	0 ns
<code>pthread_enter</code>	0.00 %	0 ns	0 ns
<code>pthread_exit</code>	0.00 %	0 ns	0 ns
[CUDA memcpy HtoD]	0.00 %	0 ns	0 ns
<code>cudaLaunch_v3020</code>	0.00 %	0 ns	0 ns
<code>cudaDeviceSynchronize_v3020</code>	0.00 %	0 ns	514.496 µs
<code>cudaEventQuery_v3020</code>	0.00 %	0 ns	396.36 µs

The right sidebar shows the properties for the `void matrixMulCUDA` kernel, including Start (200.482 ms), End (200.996 ms), Duration (514.496 µs), and Occupancy (100%).

UNIFIED MEMORY PROFILING

Requirements: Pascal (GP100) and Linux-64, Windows TCC in the future

Data transfers to/from GPU with accurate timestamps (event-based)

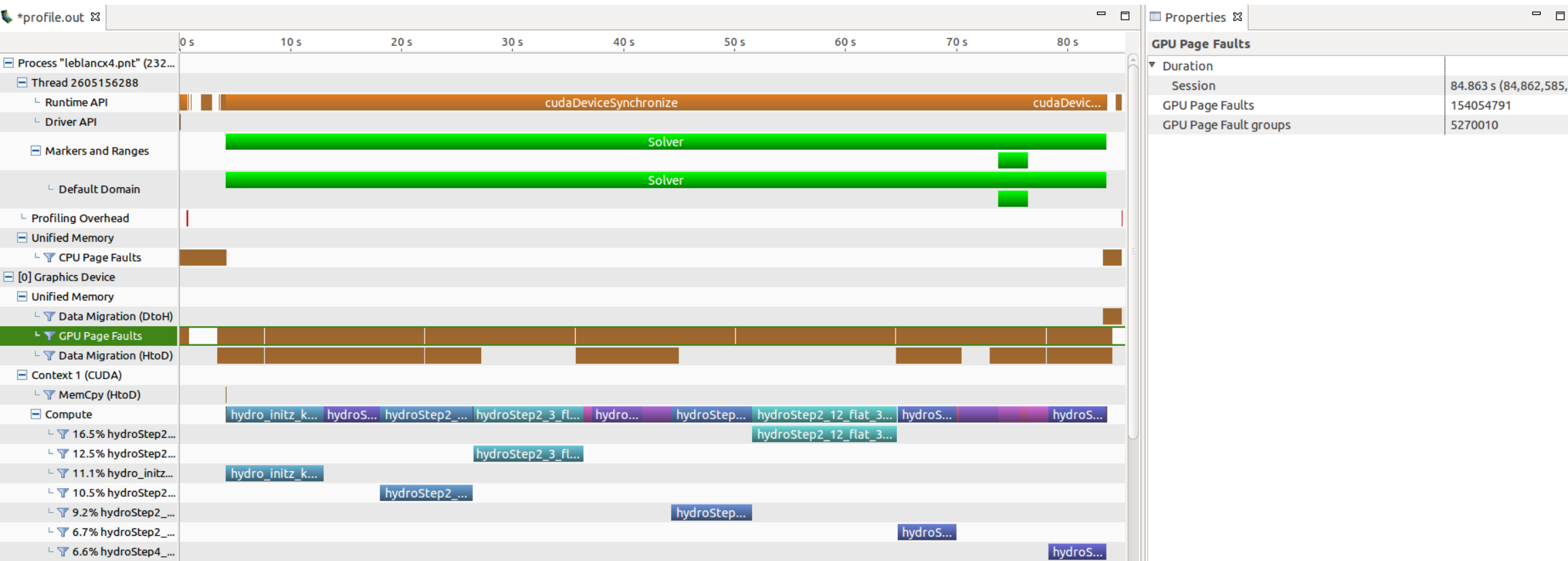
Supported counters:

CPU and GPU page faults, H2D data migration, D2H data migration

New activity record with timestamps, transfer size, virtual page base address, devices involved

UNIFIED MEMORY PROFILING

Visual profiler - 8.0 unified memory timeline



UNIFIED MEMORY PROFILING ANALYSIS

Results

i Unified Memory Analysis

The following table shows the top virtual addresses which have maximum data migration size

Virtual address	HtoD migration size	DtoH migration size	CPU page faults	GPU page faults	Migration throughput	Average faults per second
0x900709000	20.48 kB	1.012 MB	0	4	67.736 MB/s	37145.6631
0x900b08000	1.016 MB	4.096 kB	2	0	96.593 MB/s	0.00
0x900108000	1.016 MB	4.096 kB	2	1	78.726 MB/s	12180.5725
0x900907000	1.02 MB	0 B	0	6	81.543 MB/s	568367.9704
0x900309000	4.096 kB	1.012 MB	0	0	177.101 MB/s	0.00
0x902109000	4.096 kB	1.012 MB	0	4	215.42 MB/s	69839.1378
0x901909000	0 B	1.012 MB	0	26	5.444 GB/s	242854.0509
0x900909000	0 B	1.012 MB	0	0	5.251 GB/s	0.00
0x901b09000	0 B	1.012 MB	0	14	5.456 GB/s	68297.36
0x900b09000	0 B	1.012 MB	0	0	5.109 GB/s	0.00

The following table shows the summary of unified memory migrations and page faults for the application

Total HtoD migration size	Total DtoH migration size	Total CPU Page faults	Total GPU Page faults	Total different pages	The virtual address range
37.753 MB	33.559 MB	614	20868	1797	0x902c1d000-0x900887000

NVLINK VISUALIZATION

Visual Profiler

Unguided Analysis

The screenshot shows the Visual Profiler interface. The 'Analysis' tab is active, and the 'NVLink' option is selected in the 'Dependency Analysis' section. Other options like 'Data Movement And Concurrency', 'Compute Utilization', 'Kernel Performance', and 'Dependency Analysis' are also visible.

Option to collect NVLink information

The 'Results' section displays the NVLink Analysis. It includes a topology diagram showing the CPU connected to GPU 0 and GPU 1. The diagram shows data flow with throughput values: CPU to GPU 0 (18.48 GB/s), GPU 0 to CPU (18.89 GB/s), CPU to GPU 1 (17.95 GB/s), GPU 1 to CPU (17.98 GB/s), GPU 0 to GPU 1 (57.47 MB/s), and GPU 1 to GPU 0 (57.35 MB/s).

Below the diagram are two tables:

Logical NVLink Properties

Logical NVLink	Peak Bandwidth	Physical NVLinks	Peer Access	System Access	Peer Atomic	System Atomic	Utilization %	Idle time %
GPU0->GPU1	80 GB/s	2	Yes	No	Yes	No	0	0
GPU0<->CPU	80 GB/s	2	No	Yes	No	No	46	18
GPU1<->CPU	80 GB/s	2	No	Yes	No	No	44	9

Logical NVLink Throughput

Logical NVLink	Avg Throughput	Max Throughput	Min Throughput
GPU0->GPU1	57.35 MB/s	27.667 GB/s	50.872 kB/s
GPU0<-GPU1	57.466 MB/s	18.516 GB/s	33.914 kB/s
GPU0->CPU	18.886 GB/s	25.772 GB/s	29.488 kB/s
GPU0<-CPU	18.476 GB/s	28.897 GB/s	14.341 kB/s
GPU1->CPU	17.945 GB/s	36.183 GB/s	54.306 kB/s
GPU1<-CPU	17.982 GB/s	34.96 GB/s	24.437 kB/s

Static properties

Runtime values

Topology

Achieved throughput

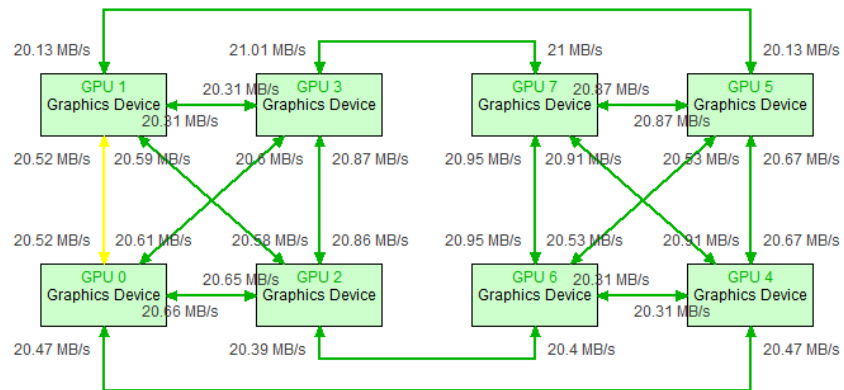
NVLINK VISUALIZATION

Visual Profiler on DGX1

i NVLink Analysis

The following NVLink topology diagram shows logical NVLink connections between GPUs and CPUs. A logical NVLink can contain one or more physical links. When two devices A and B are connected by an NVLink, the receive throughput of device A is same as the transmit throughput of device B. The tables on right hand side show the properties for each logical NVLink.

* NVLink utilization may vary in accuracy, because any activity within the sampling period is treated as active, even though most of that period could be idle.



Logical NVLink Properties

Logical NVLink	PeakBandwidth	PhysicalNVLinks	PeerAccess	SystemAccess	PeerAtomic	SystemAtomic	Utilization %	Idle time %
GPU0<-->GPU1	40 GB/s	1	Yes	No	Yes	No	0	53
GPU0<-->GPU2	40 GB/s	1	Yes	No	Yes	No	0	53
GPU0<-->GPU3	40 GB/s	1	Yes	No	Yes	No	0	53
GPU0<-->GPU4	40 GB/s	1	Yes	No	Yes	No	0	53
GPU1<-->GPU2	40 GB/s	1	Yes	No	Yes	No	0	53
GPU1<-->GPU3	40 GB/s	1	Yes	No	Yes	No	0	52
GPU1<-->GPU5	40 GB/s	1	Yes	No	Yes	No	0	52
GPU2<-->GPU3	40 GB/s	1	Yes	No	Yes	No	0	54
GPU2<-->GPU6	40 GB/s	1	Yes	No	Yes	No	0	52
GPU3<-->GPU7	40 GB/s	1	Yes	No	Yes	No	0	54
GPU4<-->GPU5	40 GB/s	1	Yes	No	Yes	No	0	53
GPU4<-->GPU6	40 GB/s	1	Yes	No	Yes	No	0	52
GPU4<-->GPU7	40 GB/s	1	Yes	No	Yes	No	0	54
GPU5<-->GPU6	40 GB/s	1	Yes	No	Yes	No	0	53
GPU5<-->GPU7	40 GB/s	1	Yes	No	Yes	No	0	54
GPU6<-->GPU7	40 GB/s	1	Yes	No	Yes	No	0	54

Logical NVLink Throughput

Logical NVLink	Avg Throughput	Max Throughput	Min Throughput
GPU0-->GPU1	20.518 MB/s	16.925 GB/s	2.568 kB/s
GPU0<--GPU1	20.524 MB/s	16.139 GB/s	2.536 kB/s
GPU0-->GPU2	20.652 MB/s	16.559 GB/s	3.804 kB/s
GPU0<--GPU2	20.659 MB/s	16.595 GB/s	2.536 kB/s
GPU0-->GPU3	20.602 MB/s	13.673 GB/s	5.088 kB/s
GPU0<--GPU3	20.607 MB/s	13.671 GB/s	7.633 kB/s
GPU0-->GPU4	20.469 MB/s	15.11 GB/s	3.015 kB/s
GPU0<--GPU4	20.467 MB/s	15.068 GB/s	2.905 kB/s
GPU1-->GPU2	20.583 MB/s	16.703 GB/s	55 B/s
GPU1<--GPU2	20.588 MB/s	16.533 GB/s	57 B/s
GPU1-->GPU3	20.314 MB/s	17.101 GB/s	55 B/s
GPU1<--GPU3	20.313 MB/s	17.101 GB/s	57 B/s
GPU1-->GPU5	20.127 MB/s	13.108 GB/s	55 B/s
GPU1<--GPU5	20.131 MB/s	9.136 GB/s	57 B/s
GPU2-->GPU3	20.865 MB/s	16.837 GB/s	1.861 kB/s
GPU2<--GPU3	20.865 MB/s	14.994 GB/s	1.241 kB/s
GPU2-->GPU6	20.397 MB/s	16.754 GB/s	3.33 kB/s
GPU2<--GPU6	20.392 MB/s	17.069 GB/s	4.995 kB/s

CPU PROFILING

Visual Profiler

Profiles the process of interest

Collect CPU sampling data alongside the GPU data you are used to

Sample at a specific frequency

Function & library symbols listed

Analyze the data per-thread from different perspectives (top-down, bottom-up or flat)

CPU PROFILING IN VISUAL PROFILER

The screenshot displays the Visual Profiler interface with the 'CPU Details' tab selected. A table lists the following events:

Event	%	Time
▼ bench_staggeredleapfrog2_	95.833%	689.695 ms
▼ CCTKi_BindingsFortranWrapperBenchADM	95.833%	689.695 ms
▶ CCTK_CallFunction	95.833%	689.695 ms
▶ __open_nocancel	1.389%	9.996 ms
▶ InitialFlat	1.389%	9.996 ms
▶ __c_mcopy8	1.389%	9.996 ms

Below the table, a code editor shows Fortran code with a callout box highlighting optimization details for a loop:

```
138 !$OMP PARALLEL DO
139   DO 100 J=1,N
140   Multiple markers at this line
141     - Generated 6 prefetch instructions for the loop
142     - Generated vector sse code for the loop
143     - Generated 5 alternate versions of the loop
144     - 2 loop-carried redundant expressions removed with 2 operations and 4 arrays
145     - Intensity = 1.93
146     
$$U(1,J+1) = U(1,J+1) + U(1,J) - U(1,J)$$

147   100 CONTINUE
148
149
150 C
151 C   PERIODIC CONTINUATION
152 C
153
154   DO 110 J=1,N
155     
$$CU(1,J) = CU(M+1,J)$$

```

OPENACC PROFILING

Visual Profiler, `nvprof`

`nvprof`:

- Summary of OpenACC activities with their inclusive and exclusive time
- Trace of activities along with CUDA API calls and GPU activities, including CUDA device/context/stream info, data transfer sizes, etc.

- Source file/line and kernel name correlation

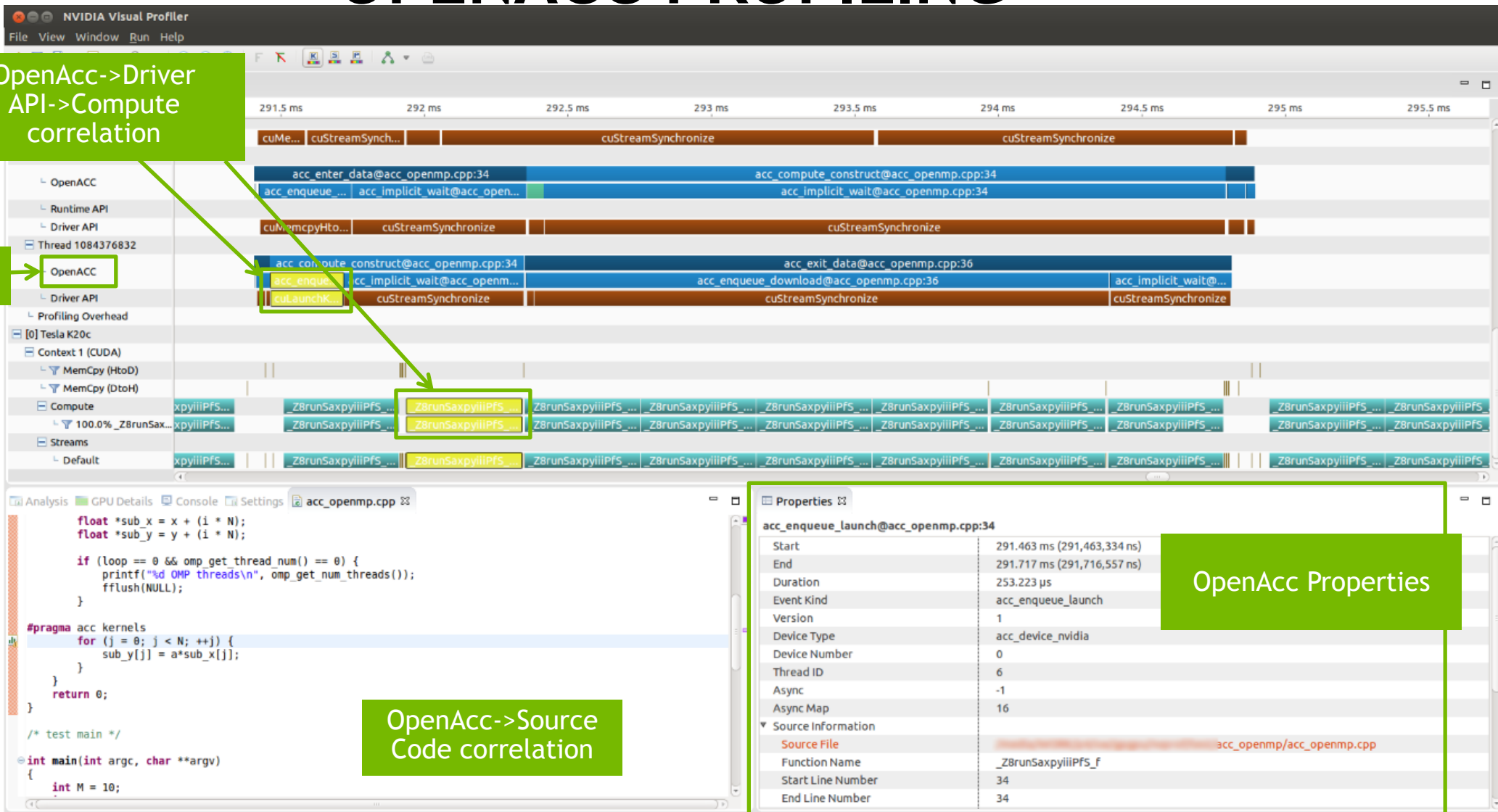
Visual Profiler:

- Visualization of OpenACC activities, coloring for different types of activities
- Correlation with their compiler-generated API calls and device activities
- Source code view for mapped OpenACC directives

OPENACC PROFILING

OpenAcc->Driver
API->Compute
correlation

OpenAcc
timeline



OpenAcc->Source
Code correlation

OpenAcc Properties

COMPUTE PREEMPTION SUPPORT

Debugger

New HW feature introduced with Pascal:

- Long running kernels can be preempted

- Processes using the GPU run without blocking each other (CUDA/Graphics)

Resulting debugger improvements:

- Support for fast single GPU debugging**

- Lower overhead for attaching to running process

Applications Places

Debug - matrixMul/src/matrixMul.cu - Nsight

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access C/C++ Debug

Debug

matrixMul [C/C++ Application]

- matrixMulCUDA<32> [0] [device 0 (GP100GL-A)] (Breakpoint)
 - CUDA Thread (0,22,0) Block (0,1,0)
 - matrixMulCUDA<32>() at matrixMul.cu:72 0x7fffe4e02a68
 - All Kernel Threads (200 Blocks of 1,024 Threads)

Name	Type	T(0,22,0)B(0,1,0)
C	@generic float * @p:	0x1001312c000
A	@generic float * @p:	0x10013000000
B	@generic float * @p:	0x10013064000

matrixMul.cu

```

65     int bBegin = BLOCK_SIZE * bx;
66
67     // Step size used to iterate through the sub-matrices of B
68     int bStep  = BLOCK_SIZE * wB;
69
70     // Csub is used to store the element of the block sub-matrix
71     // that is computed by the thread
72     float Csub = 0;
73
74     // Loop over all the sub-matrices of A and B
75     // required to compute the block sub-matrix
  
```

Outline

- stdio.h
- assert.h
- cuda_runtime.h
- cuda_profiler_api.h
- helper_functions.h

Console

matrixMul [C/C++ Application] matrixMul
Computing result using CUDA Kernel...
done



DEBUG ON DISPLAY GPU WITH NO RESTRICTIONS

MEMORY CHECKER

cuda-memcheck

Support for non-migratable system-scoped atomics checking on SM 6.x in Memcheck Tool.

Support for reporting fatal CPU-side faults when Unified Memory is enabled in Memcheck Tool.

Support for correctly determining the expected set of threads at a barrier in the presence of exited threads in Synccheck Tool.

NEW TOOLS FEATURES IN CUDA 8.0

PASCAL Architecture support

Profiler enhancements

Dependency Analysis

CPU Profiling

Unified Memory profiling

OpenACC profiling

NVLINK analysis

Debug on display GPU with no restrictions

Memcheck enhancements

FOR MORE INFORMATION ...

CUDA 8 Features Revealed Parallel Forall Blog Post :

<https://devblogs.nvidia.com/parallelforall/cuda-8-features-revealed/>

CUDA Documentation: <http://docs.nvidia.com/cuda/>

Download CUDA Toolkit 8: <https://developer.nvidia.com/cuda-downloads>



BACKUP SLIDES

AGENDA

CUDA Tools summary

New Tools Features in CUDA 8.0

CUDA TOOLS

IDE: Nsight Eclipse Edition, Nsight Visual Studio Edition

Debug: cuda-gdb, CUDA Debug API

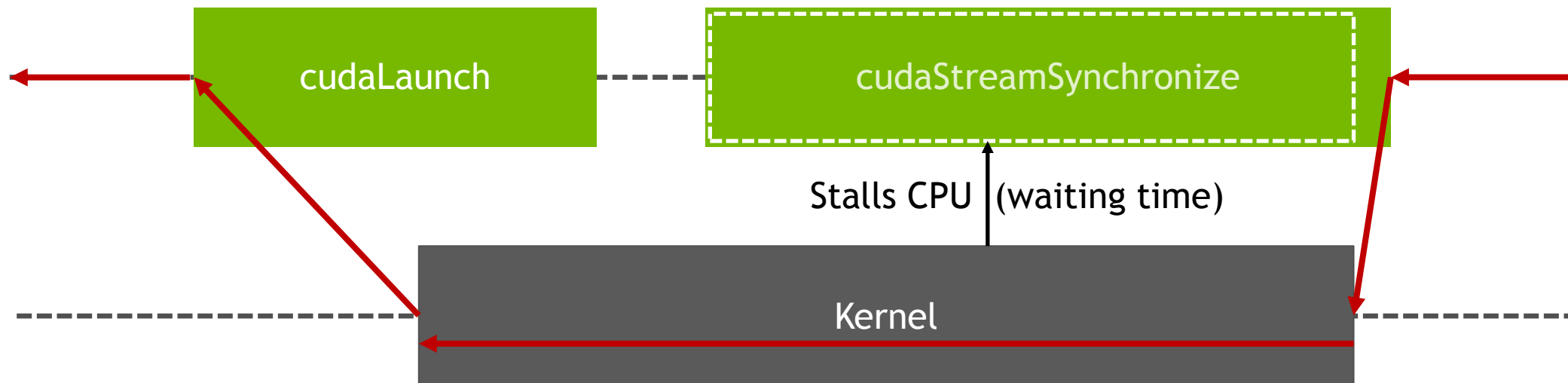
Memcheck: cuda-memcheck

Profile/Trace: CUDA Visual Profiler, nvprof, CUPTI

DEPENDENCY ANALYSIS EXAMPLE

Dependencies between events derived from programming model constraints

Allows to compute wait states and the critical path



DEPENDENCY ANALYSIS IN VISUAL PROFILER

New option in the Visual Profiler's expert system and new modes to display the critical path data on the timeline. Exec dependencies for a selected interval:

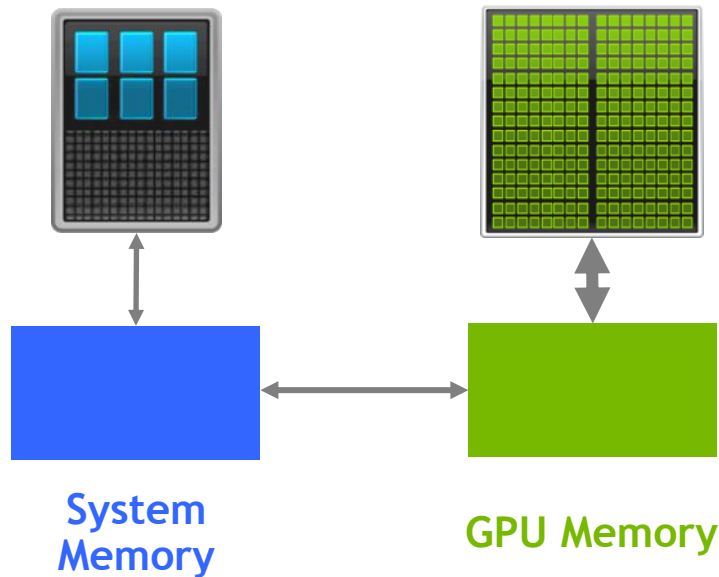
The screenshot displays the NVIDIA Visual Profiler interface. The top section shows a timeline with various execution events. The bottom section, titled 'Analysis', shows the 'Dependency Analysis' results. The results table lists function names, their percentage of time on the critical path, the time on the critical path, and the waiting time.

Function Name	Time on Critical Path (%)	Time on Critical Path	Waiting
[CUDA memcpy HtoD]	41.29 %	8.134 ms	
cudaMalloc_v3020	34.96 %	6.887 ms	
[CUDA memcpy DtoH]	12.23 %	2.41 ms	
<Other>	3.79 %	746.529 µs	
GPUBlackScholesCallPut(float*, float*, float*, float*, float*, int)	3.52 %	692.799 µs	
GPUBlackScholesCallPut_Stream(float*, float*, float*, float*, float*, int)	1.60 %	314.336 µs	
cudaEventRecord_v3020	1.02 %	200.724 µs	
GPUBlackScholesCallPut_CPUOverlap(float*, float*, float*, float*, float*, int)	1.02 %	200.032 µs	
cudaMemcpyAsync_v3020	0.42 %	83.559 µs	
cudaEventSynchronize_v3020	0.15 %	29.021 µs	937.7

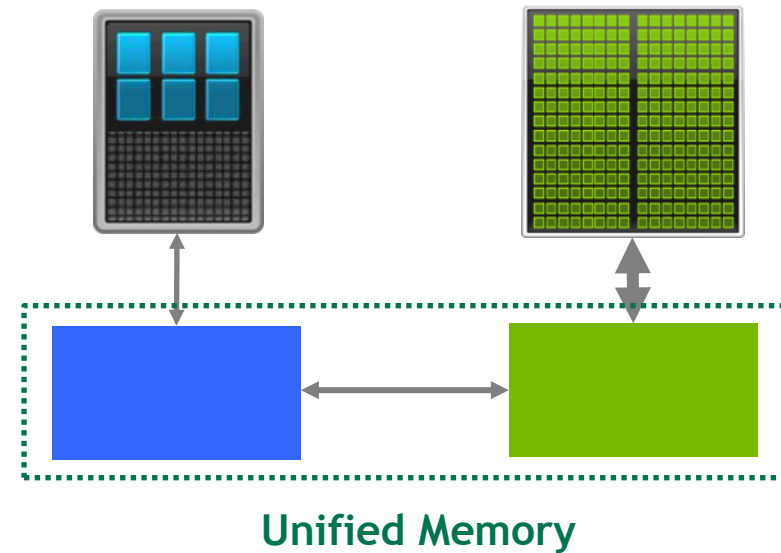
UNIFIED VIRTUAL MEMORY

Starting with Kepler and CUDA 6

Custom Data Management

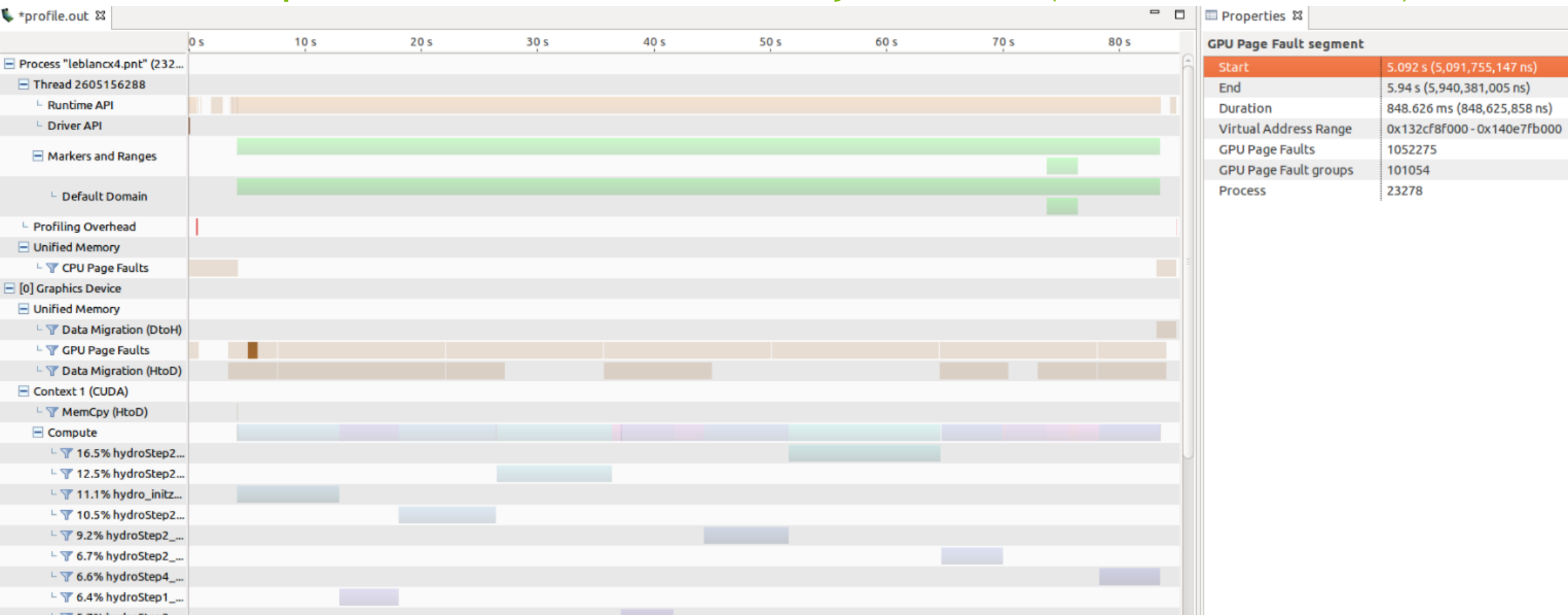


Developer View With Unified Memory



UNIFIED MEMORY PROFILING

Visual profiler - 8.0 unified memory timeline (interval selection)



Debug - matrixMul/src/matrixMul.cu - Nsight

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access C/C++ Debug

Debug

matrixMul [C/C++ Application]

- matrixMulCUDA<32> [0] [device 0 (GP100GL-A)] (Breakpoint)
- CUDA Thread (0,22,0) Block (0,1,0)
 - matrixMulCUDA<32>() at matrixMul.cu:72 0x7ffe4e02a68
 - All Kernel Threads (200 Blocks of 1,024 Threads)

matrixMul.cu

```
65 int bBegin = BLOCK_SIZE * bx;
66
67 // Step size used to iterate through the sub-matrices of B
68 int bStep = BLOCK_SIZE * wB;
69
70 // Csub is used to store the element of the block sub-matrix
71 // that is computed by the thread
72 float Csub = 0;
73
74 // Loop over all the sub-matrices of A and B
75 // required to compute the block sub-matrix
```

Variable

Name	Type	T(0,22,0)B(0,1,0)
C	@generic float * @p:	0x1001312c000
A	@generic float * @p:	0x10013000000
B	@generic float * @p:	0x10013064000

Outline

- stdio.h
- assert.h
- cuda_runtime.h
- cuda_profiler_api.h
- helper_functions.h

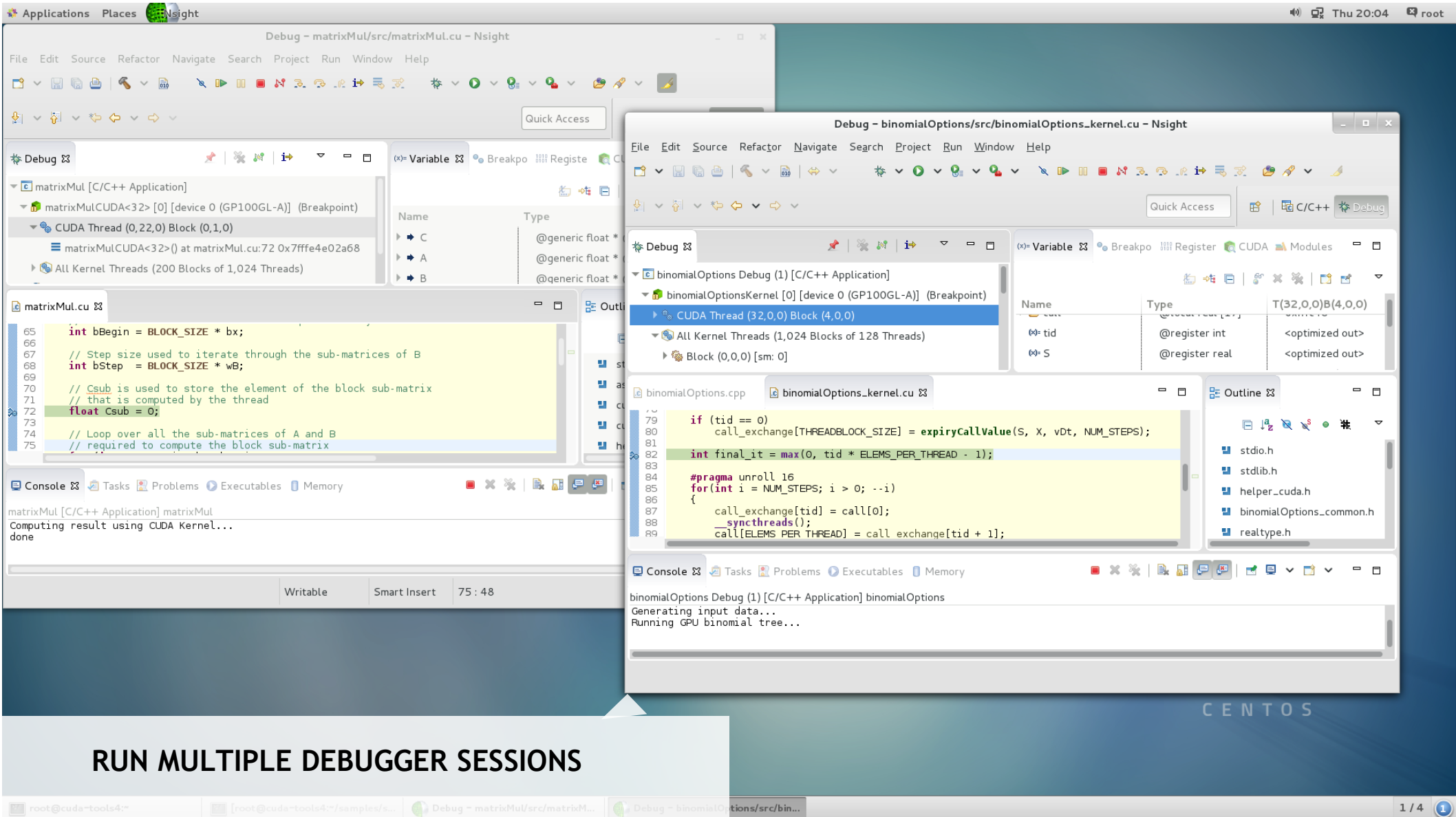
Console

```
matrixMul [C/C++ Application] matrixMul
Computing result using CUDA Kernel...
done
```

CUDA N-Body (57344 bodies): 59.1 fps | 194.3 BIPS | 3885.2 GFLOP/s | single pre...

Point Size: 0.030
Velocity Damping: 1.000
Softening Factor: 0.145
Time Step: 0.002
Cluster Scale: 0.330
Velocity Scale: 272.000

RUN GRAPHICS APPLICATIONS WHILE DEBUGGING CUDA APPLICATIONS



RUN MULTIPLE DEBUGGER SESSIONS

DEPENDENCY ANALYSIS IN NVPROF

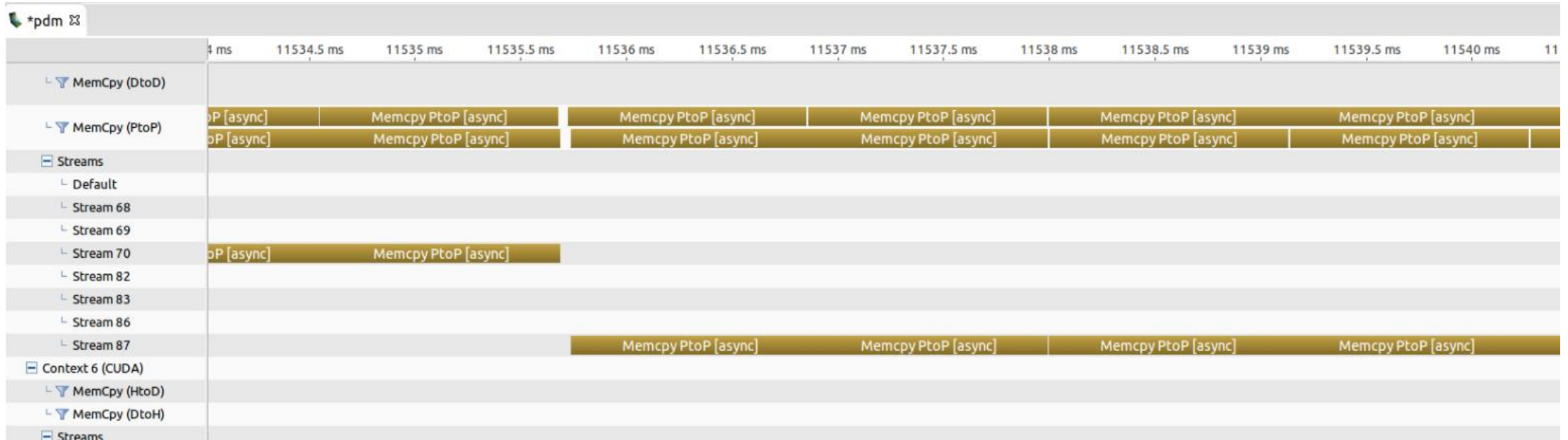
New option to run post-mortem dependency analysis

New option to trace POSIX threads for multi-threaded applications

```
==22557== Dependency Analysis:
Critical path(%)  Critical path  Waiting time  Name
94.61%          3.942181s    0ns          clock_block(long*, long)
5.20%          216.857718ms 0ns          cudaMalloc
0.16%          6.617667ms   0ns          <Other>
0.01%          293.028000us 0ns          cuDeviceGetAttribute
0.01%          235.154000us 0ns          cudaGetDeviceProperties
0.01%          221.116000us 0ns          cudaFree
0.00%          158.703000us 0ns          cudaStreamCreate
0.00%          35.252000us  0ns          cudaConfigureCall
0.00%          35.248000us  0ns          cuDeviceGetName
0.00%          33.139000us  0ns          cuDeviceTotalMem_v2
0.00%          20.298000us  0ns          cudaSetupArgument
0.00%          19.433000us  0ns          cudaGetDevice
0.00%           0ns      3.942147s    pthread_join
0.00%           0ns      3.942136s    cudaStreamSynchronize
0.00%           0ns      1.001459s    pthread_mutex_lock
0.00%           0ns     980.464357ms pthread_cond_wait
```

NVLINK ANALYSIS

Visual Profiler



Memcpy P2P

NVLINK CUPTI SUPPORT

New record for Nvlink topology

Information about connected devices and connected ports

Version, peak BW, properties (e.g. peer/system accesses, atomic accesses, ...)

Performance given using metrics: data transmitted/throughput, overhead

Metrics available split according to type of operation
(atom/reduction/write/response) as well as accumulated

SOURCE-DISASSEMBLY VIEW ENHANCEMENTS

Visual Profiler

Single integrated view for the different source level analysis results

The Source-Disassembly view contains:

- High level source

- Assembly instructions

- Hotspots at the source level

- Hotspots at the assembly instruction level

- Columns for profiling data aggregated to the source level

- Columns for profiling data collected at the assembly instruction level

SOURCE-DISASSEMBLY VIEW ENHANCEMENTS

The screenshot displays the NVIDIA Visual Profiler interface, showing the source code and disassembly views for a kernel named `combinedKernel`. The source code is on the left, and the disassembly is on the right. The bottom panel shows the **Kernel Profile - PC Sampling** results.

Source Code:

```
File - /C:/demo/session3/estimate_combined3
Line Execution Count Warp State
86 [ ] [ ] int wid = tid / warpSize;
87 [ ] [ ] val = warpReduce(val);
88 [ ] [ ] if(lane == 0)
89 [ ] [ ] {
90 [ ] [ ] shared[wid] = val;
91 [ ] [ ] }
92 [ ] [ ] __syncthreads();
93 [ ] [ ] val = (tid < CTA_SIZE / warpSize) ? shared[
94 [ ] [ ] if(wid == 0)
95 [ ] [ ] {
96 [ ] [ ] val = warpReduce(val);
97 [ ] [ ] }
98 [ ] [ ] return val;
99 [ ] [ ] }
100 [ ] [ ]
101 [ ] [ ] Mat33 Rcurr;
102 [ ] [ ] float3 tcurr;
103 [ ] [ ]
104 [ ] [ ] PtrStep<float> vmap_curr;
105 [ ] [ ] PtrStep<float> nmap_curr;
106 [ ] [ ]
107 [ ] [ ] Mat33 Rprev_inv;
```

Disassembly:

```
Execution Count Warp State Disassembly
[ ] [ ] { ISETP.GT.AND P2, PT, R4.reuse, 0x7, PT;
[ ] [ ] SSY `(.L_3); }
[ ] [ ] IADD32I R5, R4, 0x1f;
[ ] [ ] ISETP.GT.U32.AND P1, PT, R5, 0x3e, PT;
[ ] [ ] SHL R5, R8, 0x2;
[ ] [ ] MOV R8, RZ;
[ ] [ ] @IP2 LDS.U.32 R8, [R5];
[ ] [ ] @P1 SYNC;
[ ] [ ] SHFL.DOWN PT, R6, R8, 0x10, 0x1f;
[ ] [ ] FADD.FTZ R6, R8, R6;
[ ] [ ] SHFL.DOWN PT, R7, R6, 0x8, 0x1f;
[ ] [ ] FADD.FTZ R6, R6, R7;
[ ] [ ] SHFL.DOWN PT, R7, R6, 0x4, 0x1f;
[ ] [ ] FADD.FTZ R6, R6, R7;
[ ] [ ] SHFL.DOWN PT, R7, R6, 0x2, 0x1f;
[ ] [ ] FADD.FTZ R6, R6, R7;
[ ] [ ] SHFL.DOWN PT, R7, R6, 0x1, 0x1f;
[ ] [ ] { FADD.FTZ R8, R6, R7;
[ ] [ ] SYNC; }
[ ] [ ] .L_3:
[ ] [ ] { ISETP.NE.AND P3, PT, R4, RZ, PT;
[ ] [ ] SSY `(.L_4); }
```

Kernel Profile - PC Sampling Results:

Cuda Functions	Sample Count	% of Kernel Samples
combinedKernel(Combined)	19482	100.00 %

Analysis Panel:

- Kernel Memory: ✓
- Global Memory Access Pattern: ✓
- Shared Memor...cess Pattern: ✓

MISC IMPROVEMENTS

Profilers/CUPTI

Added support to establish correlation between an external API (such as OpenACC, OpenMP) and CUPTI API activity records

New activity records for CUDA synchronization constructs (ctx sync, CUDA event, ...)

UVM-lite counter profiling support on Mac

Support for FP16 data

Added containers to store the information of events and metrics in the form of activity records