NVIDIA PROFILING TOOLS

Jeff Larkin, December 04, 2018

UPDATES FOR CUDA 9.2

NVPROF

Many New Metrics:
- Tensor Core Metrics
- L2 Metrics
- Memory Instructions Per Load/Store

Display PCIe Topology

View Trace and Profile in combined output (`--trace`)

VISUAL PROFILER

Summary View for Memory Hierarchy

Improved Handling of Segments for UVM Data on the Timeline

UPDATES FOR CUDA 10.0

Added tracing support for Turing

New kernel profiler – Nsight Compute (supports Turing)

OpenMP profiling

Tracing support for CUDA kernels, memcpy and memset nodes launched by a CUDA Graph

Support for version 3 NVIDIA Tools Extension API (NVTX) (This is a header-only implementation)

CUDA VISUAL PROFILER

Overview of key features

Kernel profile – memory hierarchy view

Unified Memory

NVLink

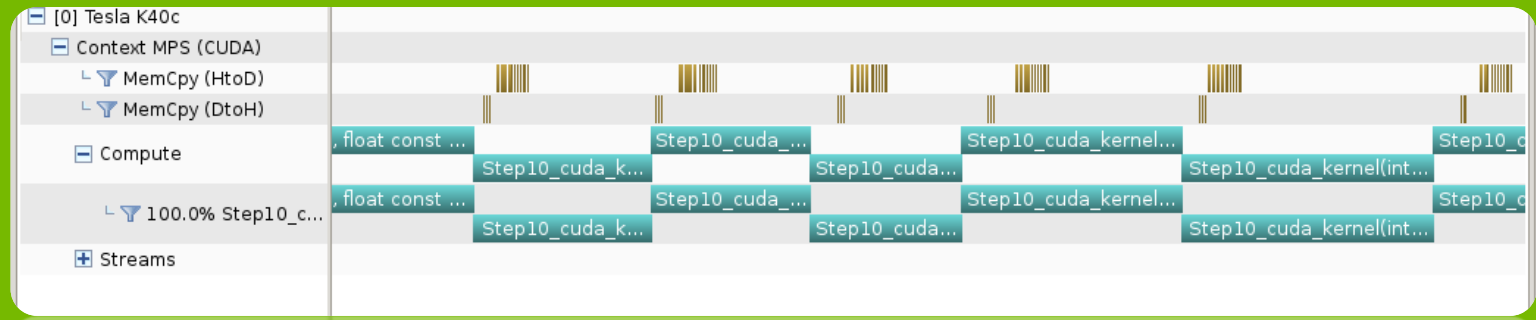
PC sampling

OpenACC/OpenMP Profiling

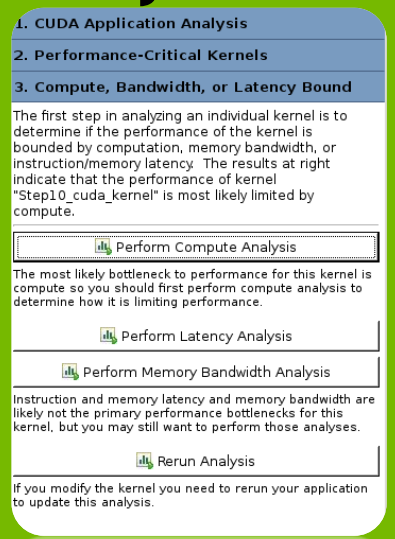
NVTX

NVIDIA'S VISUAL PROFILER (NVVP)

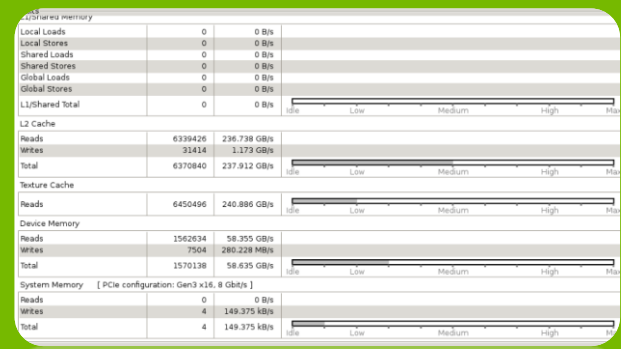
Timeline

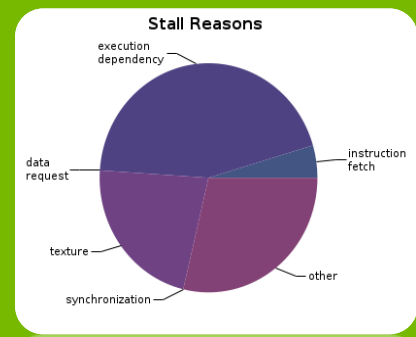


Guided System

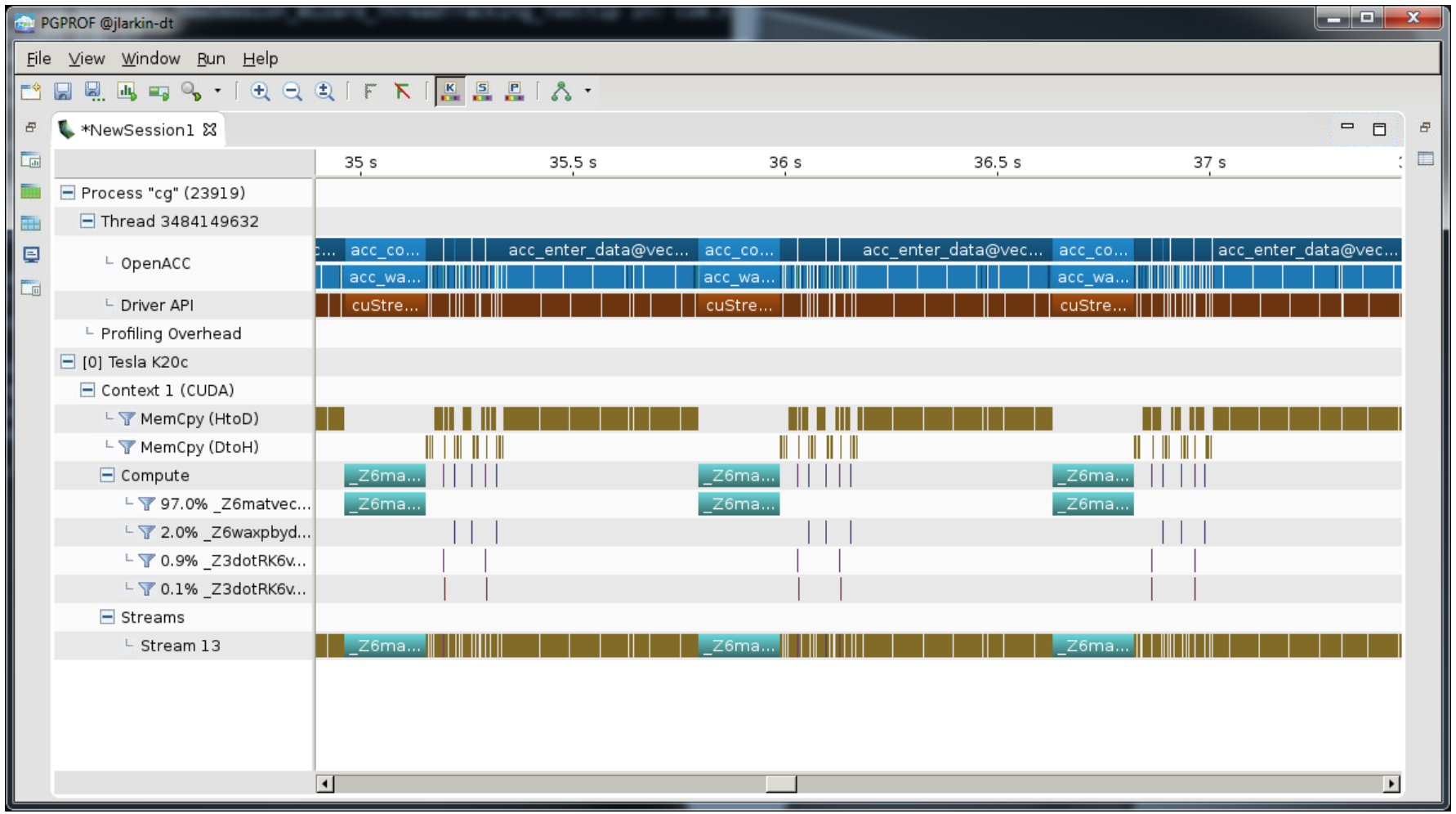


Analysis

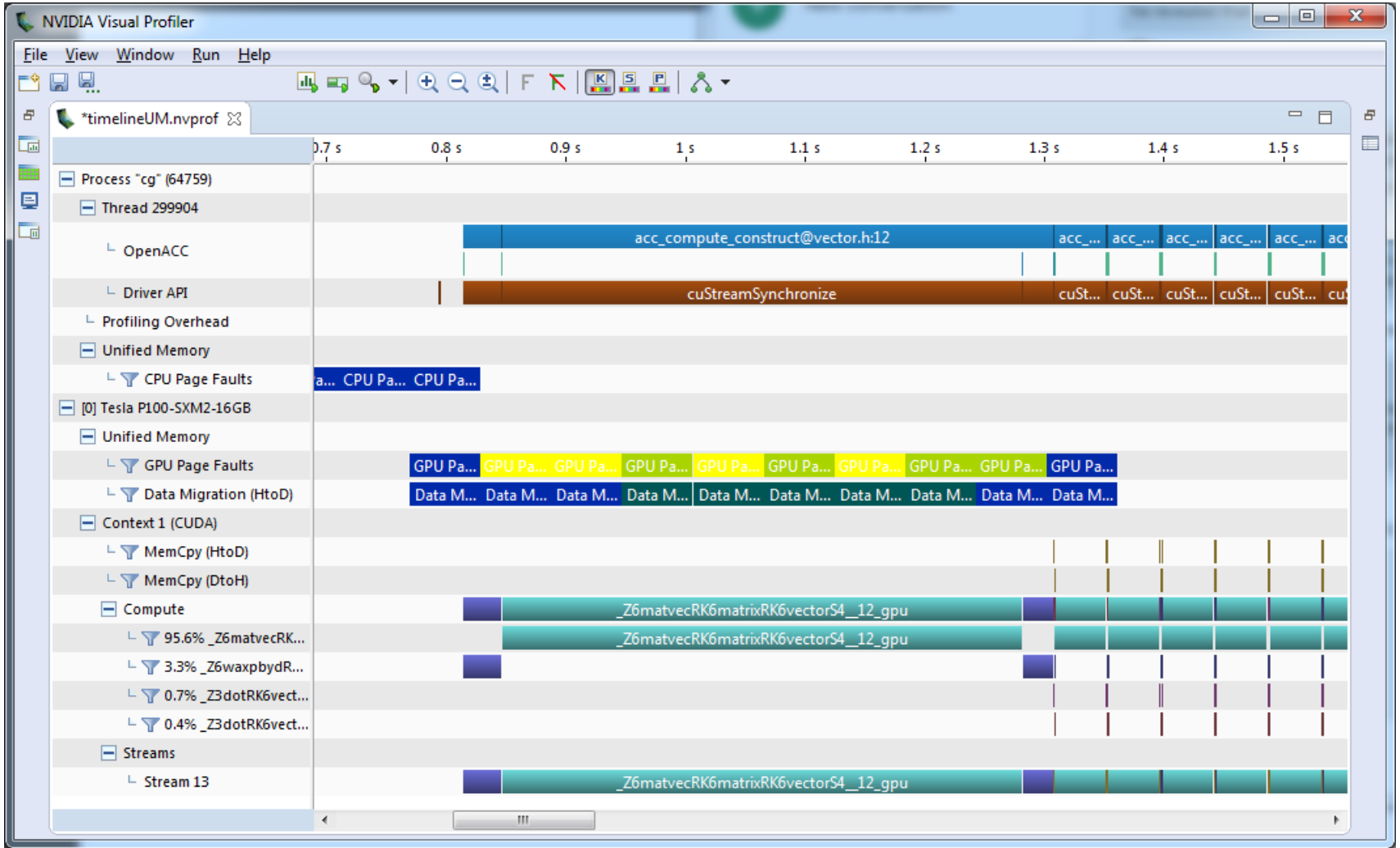




DATA MOVEMENT IN VISUAL PROFILER

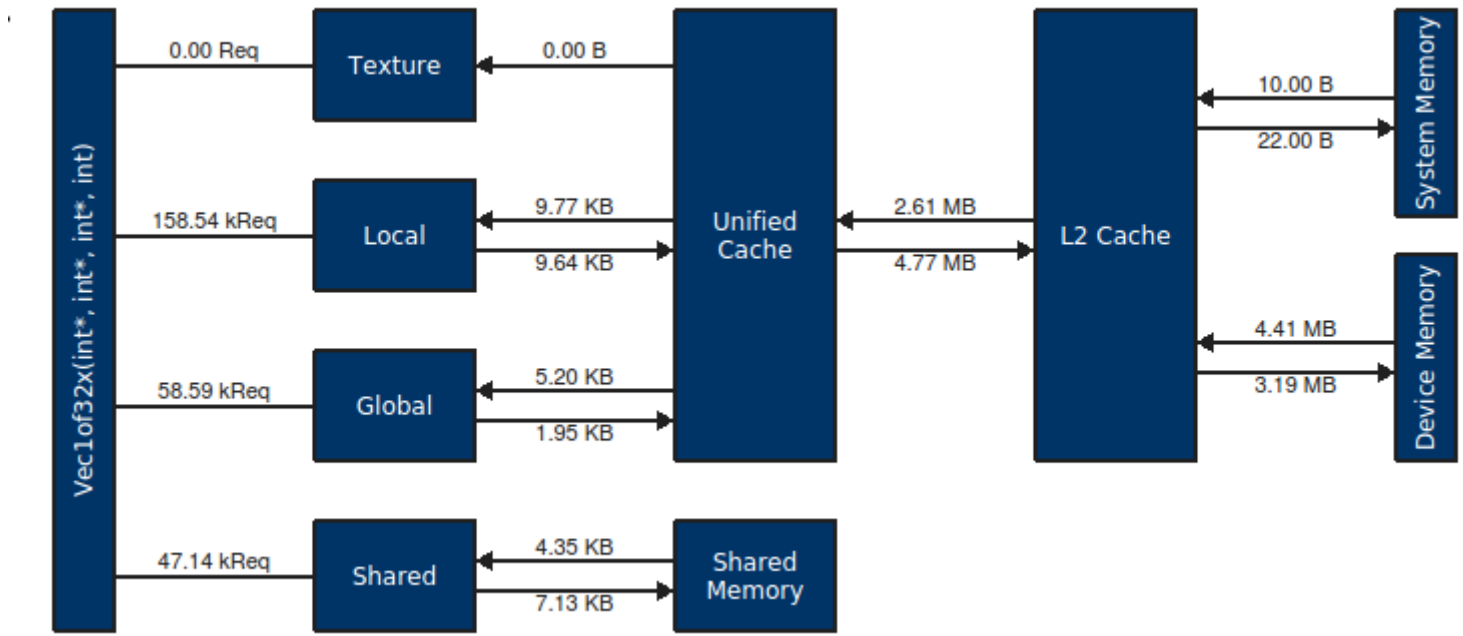


UVM IN VISUAL PROFILER



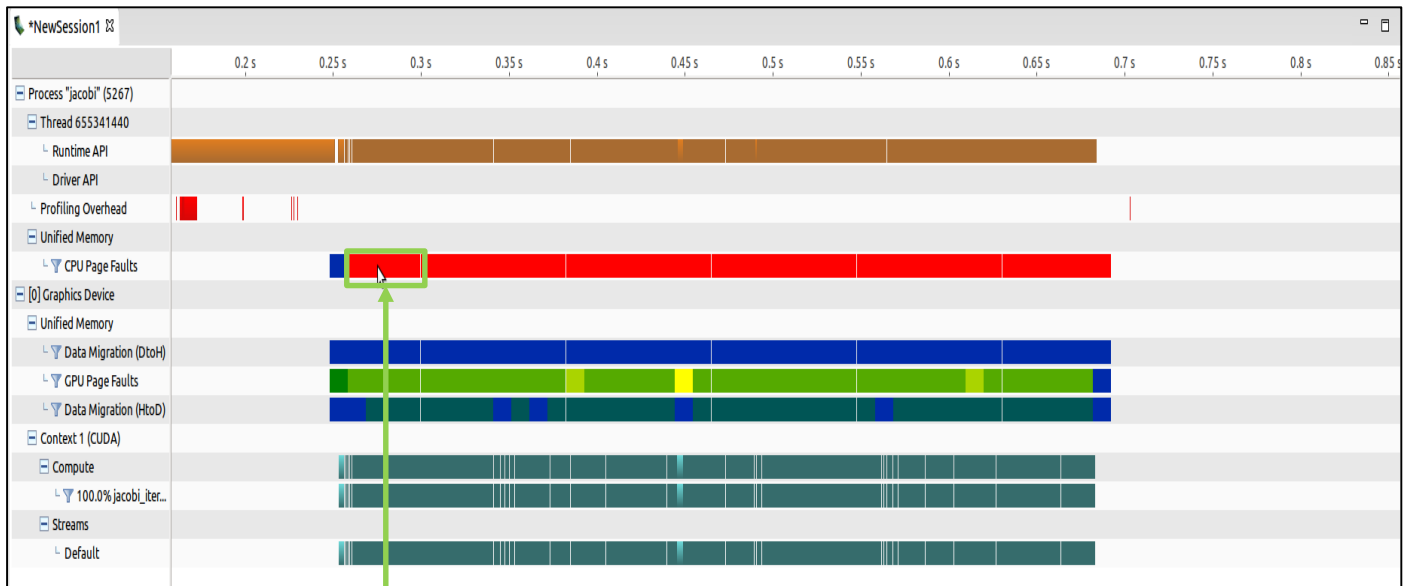
KERNEL PROFILE

Memory hierarchy view



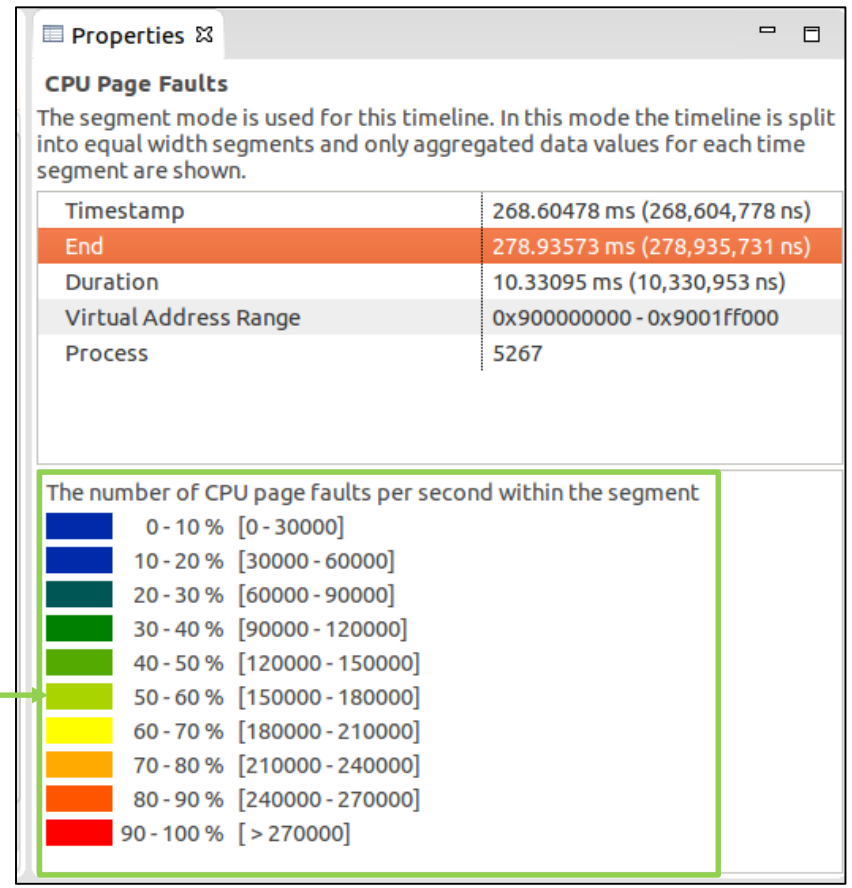
VISUAL PROFILER

Segment mode timeline



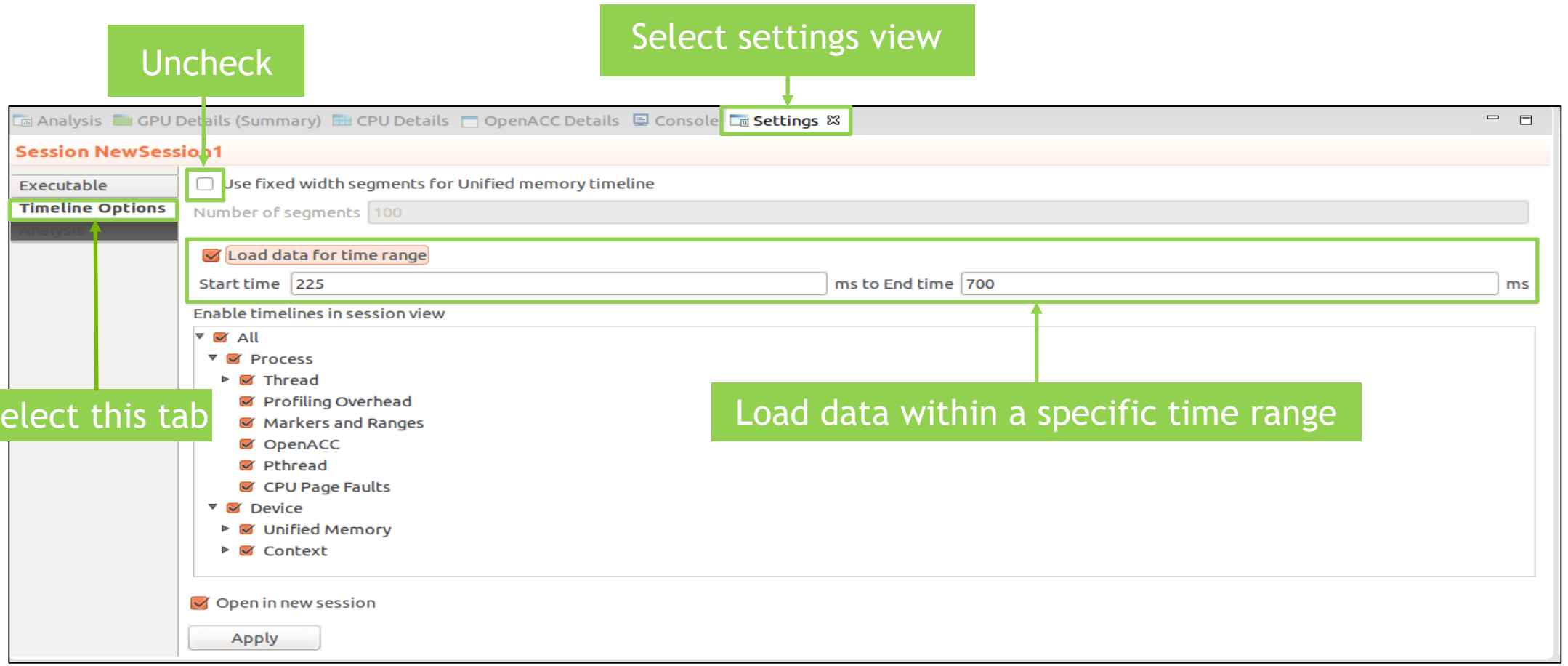
Segment mode interval

Heat map for CPU page faults



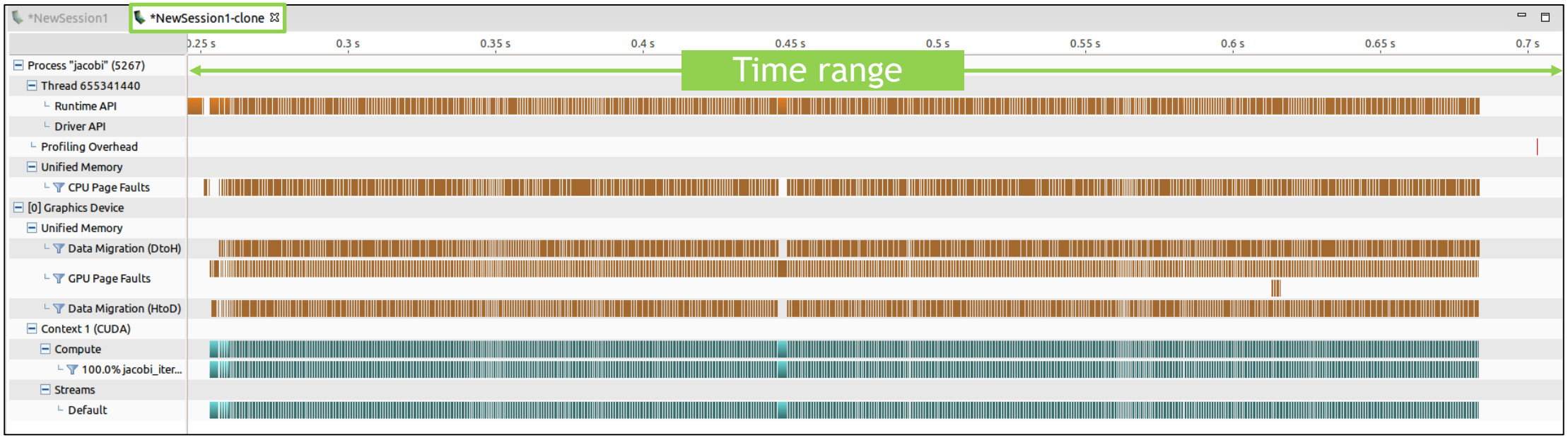
VISUAL PROFILER

Switch to non-segment view



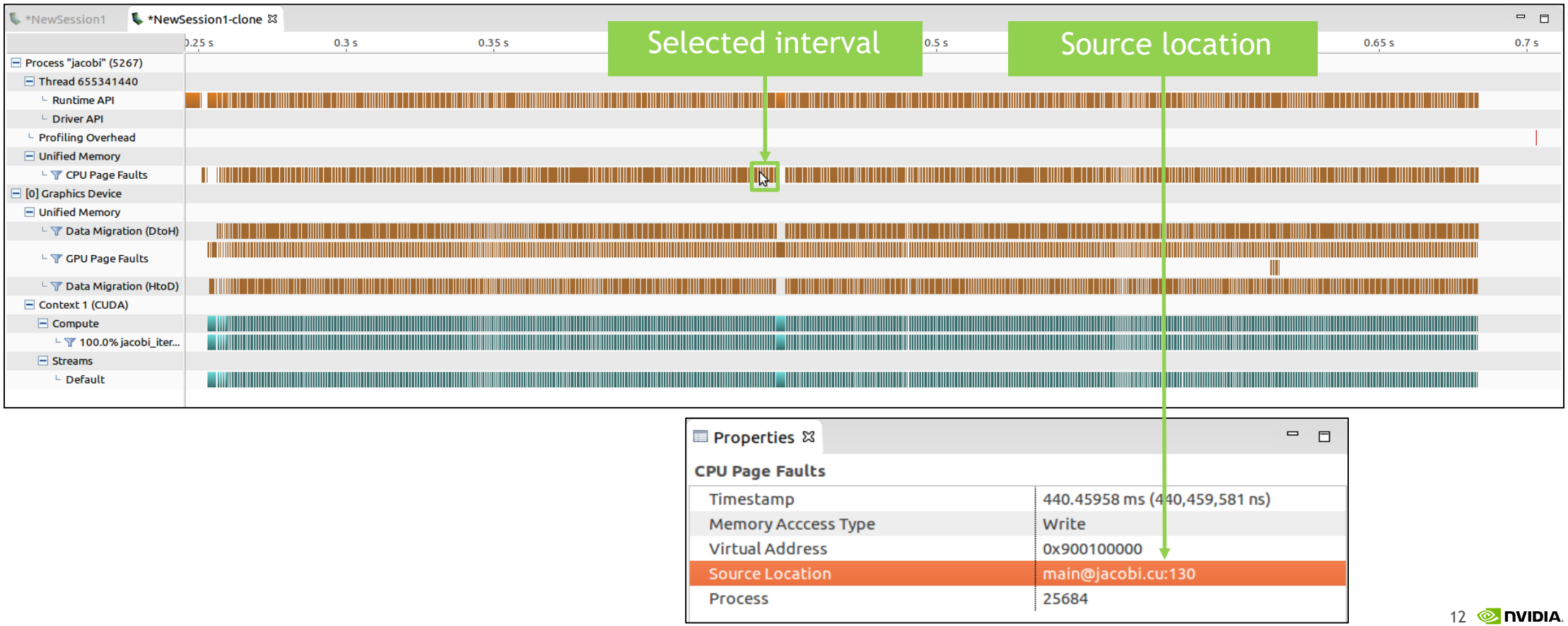
VISUAL PROFILER

Non-segmented mode timeline



VISUAL PROFILER

CPU Page Fault Source Correlation



VISUAL PROFILER

CPU Page Fault Source Correlation

Properties

CPU Page Faults

Timestamp	440.45958 ms (440,459,581 ns)
Memory Acccess Type	Write
Virtual Address	0x900100000
Source Location	main@jacobi.cu:130
Process	25684

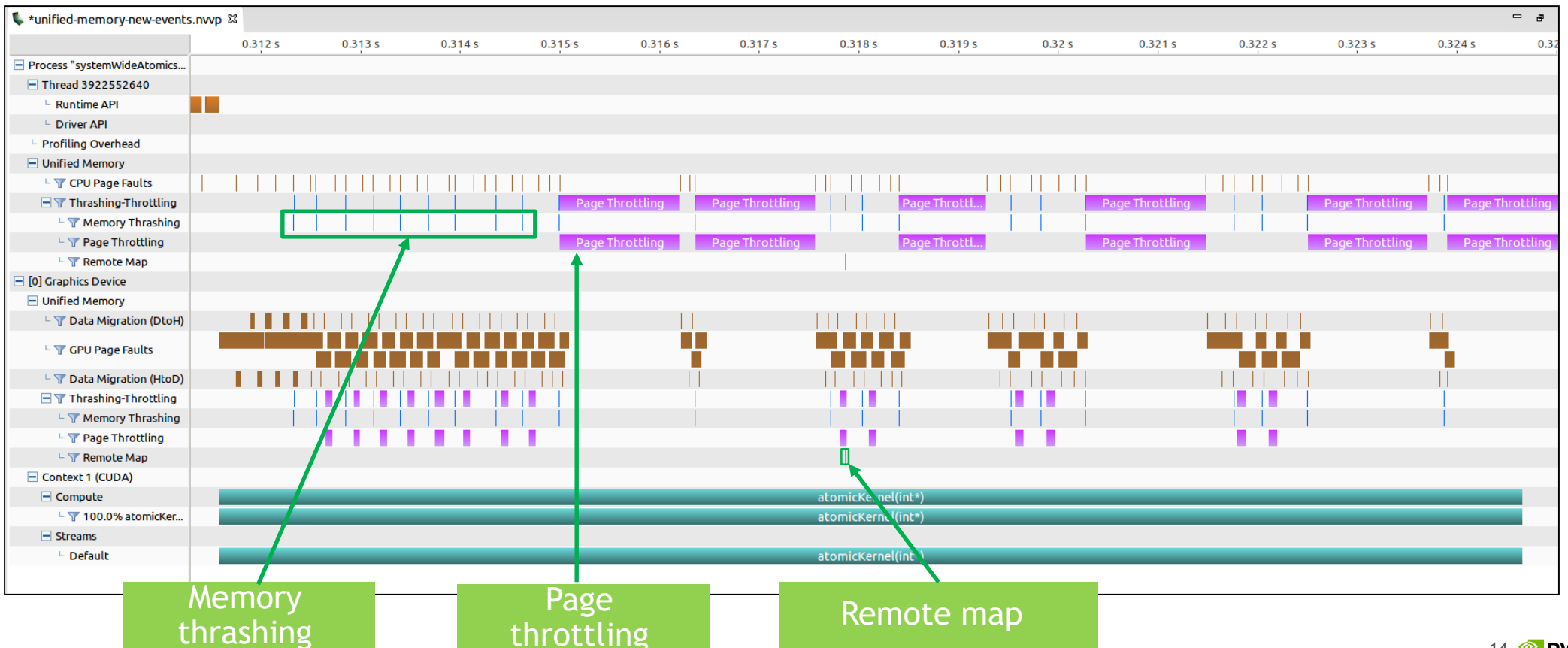
Source line causing CPU page fault

*NewSession1 jacobi.cu

```
    float * a;
    float * a_new;
    float * weights;

    CUDA_CALL(cudaMallocManaged(&a,     nx*ny*sizeof(float)));
    CUDA_CALL(cudaMallocManaged(&a_new, nx*ny*sizeof(float)));
    CUDA_CALL(cudaMallocManaged(&weights, n_weights*sizeof(float)));

    init(a,a_new,nx,ny,weights,n_weights);

    cudaEvent_t start,stop;
    CUDA_CALL(cudaEventCreate(&start));
    CUDA_CALL(cudaEventCreate(&stop));

    CUDA_CALL(cudaDeviceSynchronize());
    CUDA_CALL(cudaEventRecord(start));

    PUSH_RANGE("while loop",0)
    int iter = 0;

    while ( iter <= iter_max )
    {
        PUSH_RANGE("jacobi step",1)
        jacobi_iteration<<<dim3(nx/32,ny/4),dim3(32,4)>>>(a_new,a,nx,ny,weights[0]);
        CUDA_CALL(cudaGetLastError());
        CUDA_CALL(cudaDeviceSynchronize());
        POP_RANGE

        std::swap(a,a_new);

        PUSH_RANGE("periodic boundary conditions",2)
        //Apply periodic boundary conditions
        for (int ix = 0; ix < nx; ++ix)
        {
            a[     0*nx+ix]=a[(ny-2)*nx+ix];
            a[(ny-1)*nx+ix]=a[     1*nx+ix];
        }
        POP_RANGE

        if ( 0 == iter%100 )
            std::cout<<iter<<std::endl;
        iter++;
    }

    CUDA_CALL(cudaEventRecord(stop));
    CUDA_CALL(cudaDeviceSynchronize());
    POP_RANGE
```

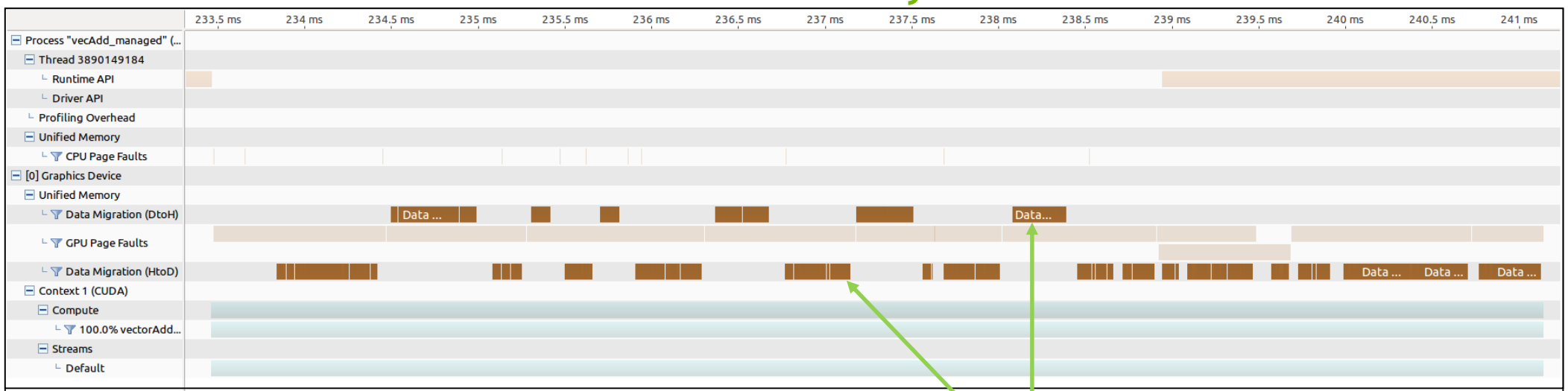
VISUAL PROFILER - NEW UNIFIED MEMORY EVENTS

Page throttling, Memory thrashing, Remote map



VISUAL PROFILER

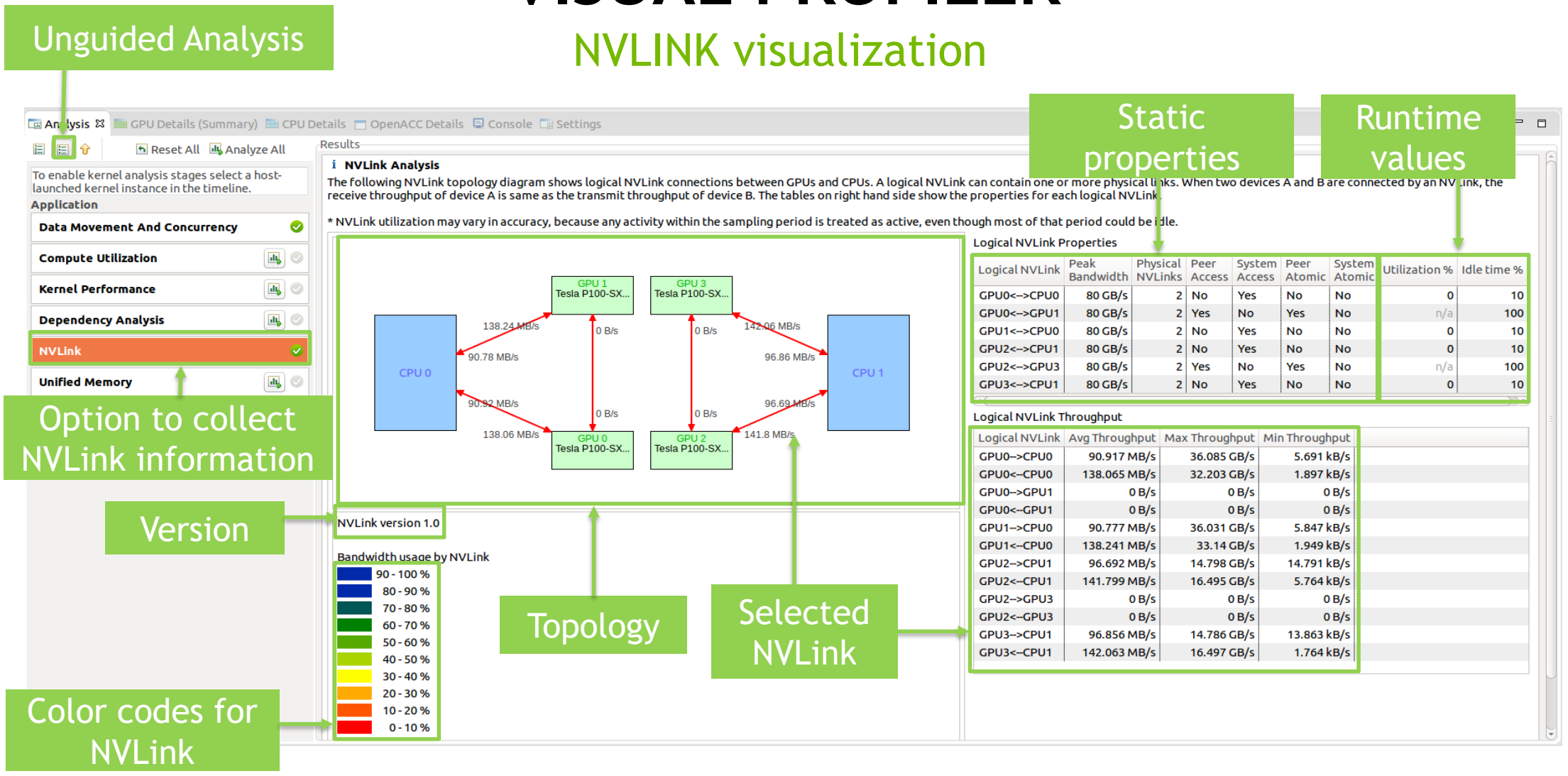
Filter and Analyze



Filtered intervals

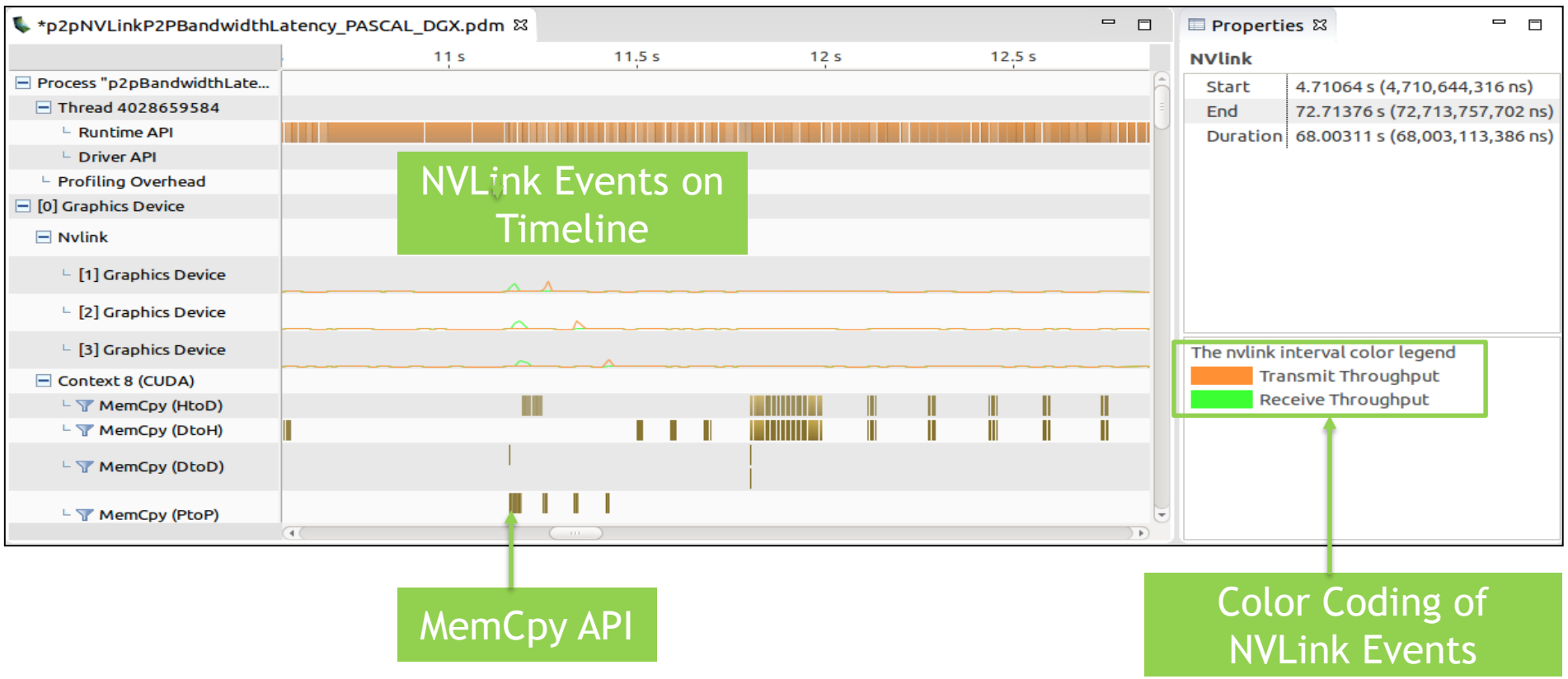
VISUAL PROFILER

NVLINK visualization



VISUAL PROFILER

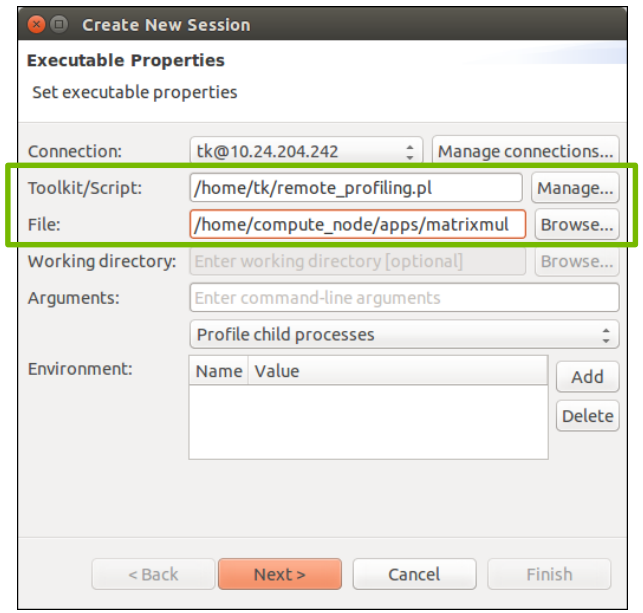
NVLink events on timeline



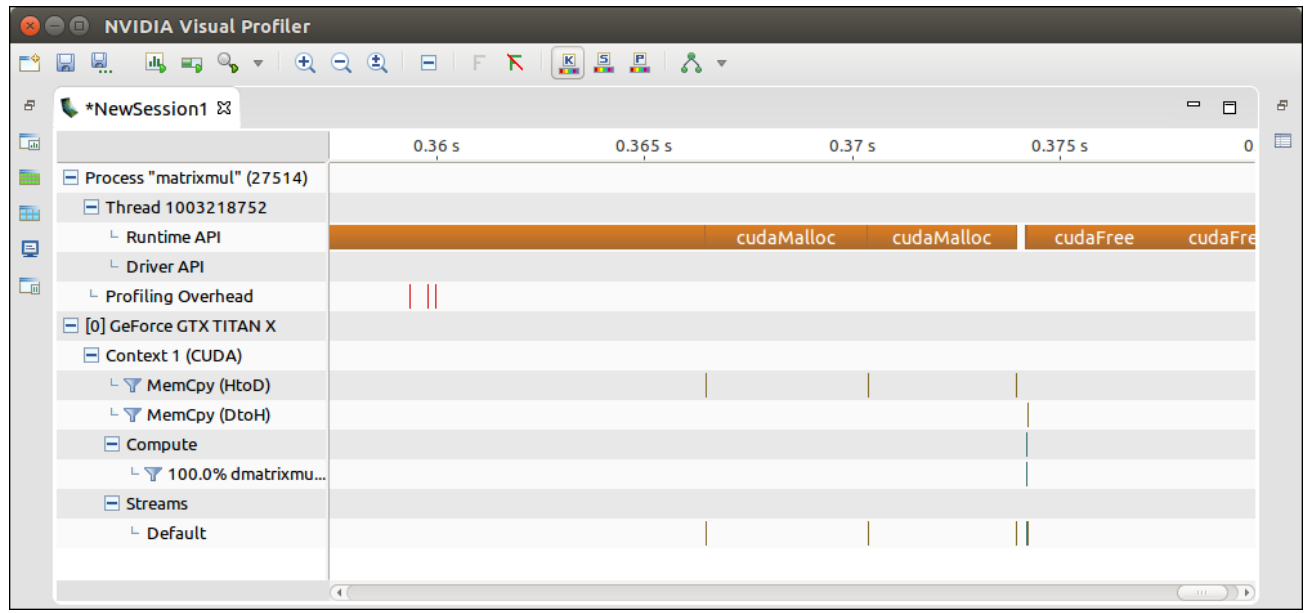
VISUAL PROFILER

Multi-hop remote profiling - Application Profiling

1. Select custom script, then create a remote session as usual



2. Application transparently runs on compute node and profiling data is displayed in the Visual Profiler



CPU SAMPLING

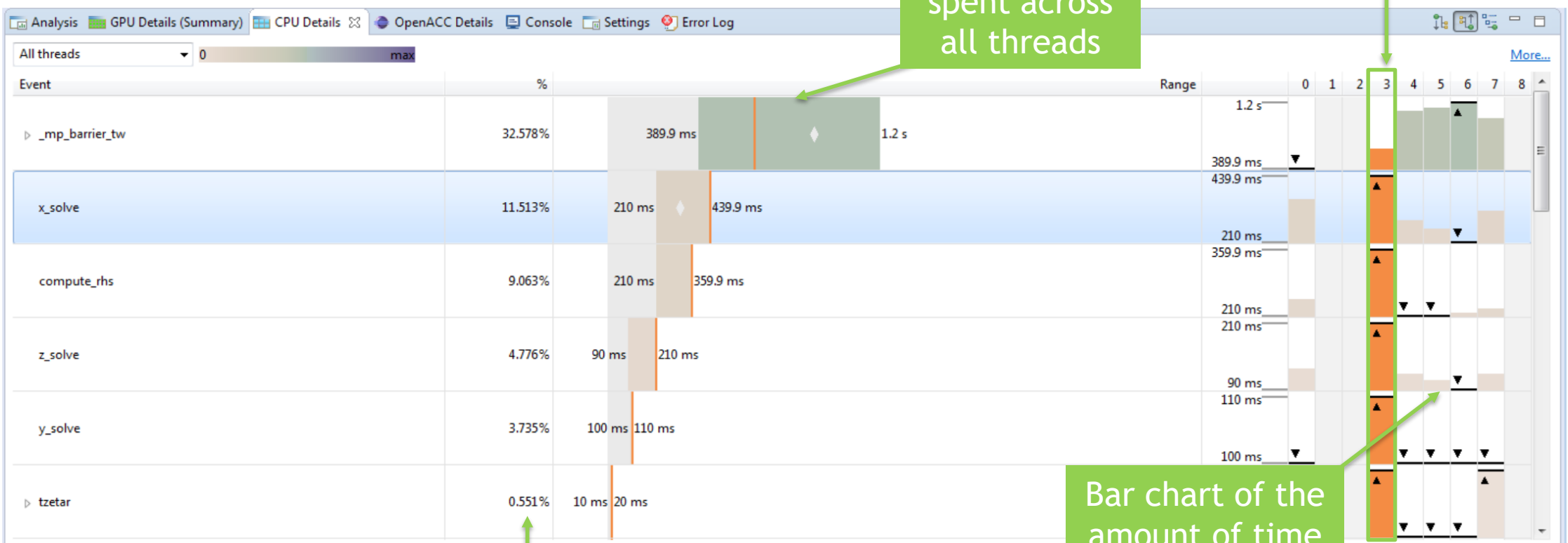
- CPU profile is gathered by periodically sampling the state of each thread in the running application.
- The CPU details view summarizes the samples collected into a call-tree, listing the number of samples (or amount of time) that was recorded in each function.

VISUAL PROFILER

CPU Sampling

Selected thread is highlighted in Orange

Range of time spent across all threads



Bar chart of the amount of time spent by thread

Percentage of time spent collectively by all threads

PC SAMPLING

PC sampling feature is available for device with CC >= 5.2

Provides CPU PC sampling parity + additional information for warp states/stalls reasons for GPU kernels

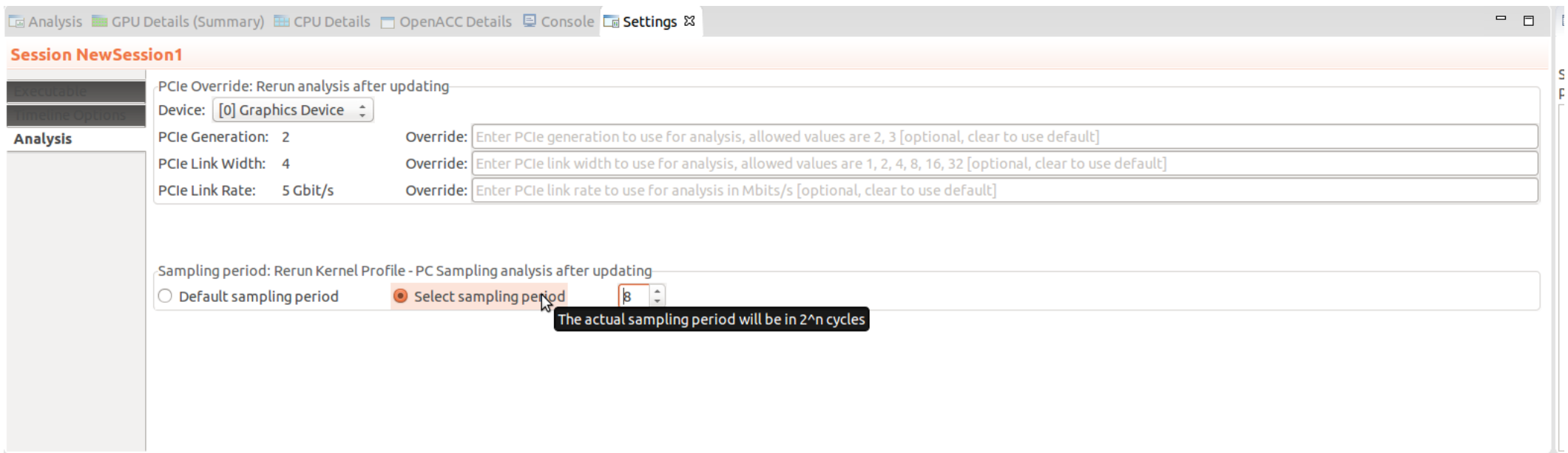
Effective in optimizing large kernels, pinpoints performance bottlenecks at specific lines in source code or assembly instructions

Samples warp states periodically in round robin order over all active warps

No overheads in kernel runtime, CPU overheads to parse the records

VISUAL PROFILER - PC SAMPLING

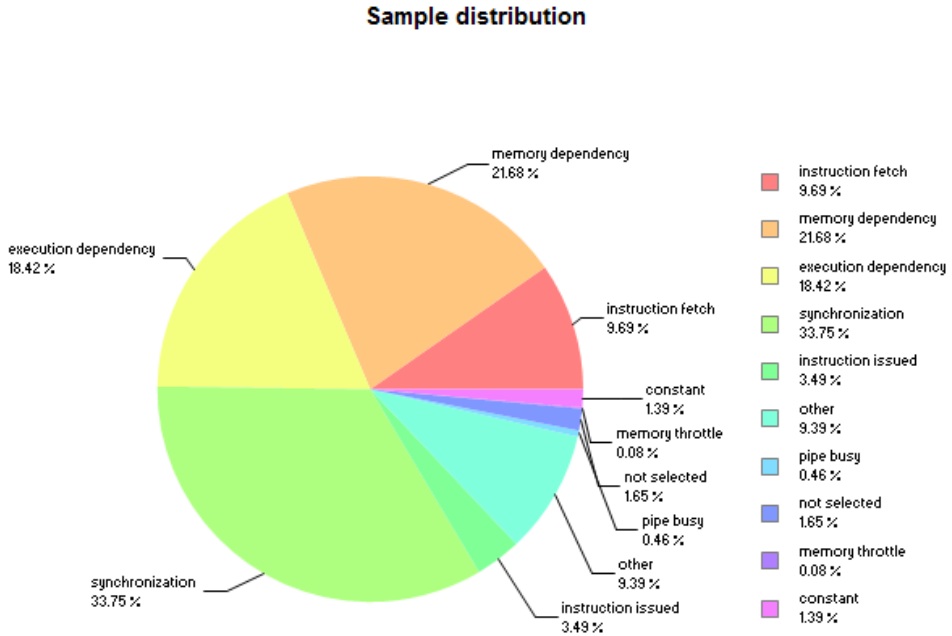
Option to select sampling period

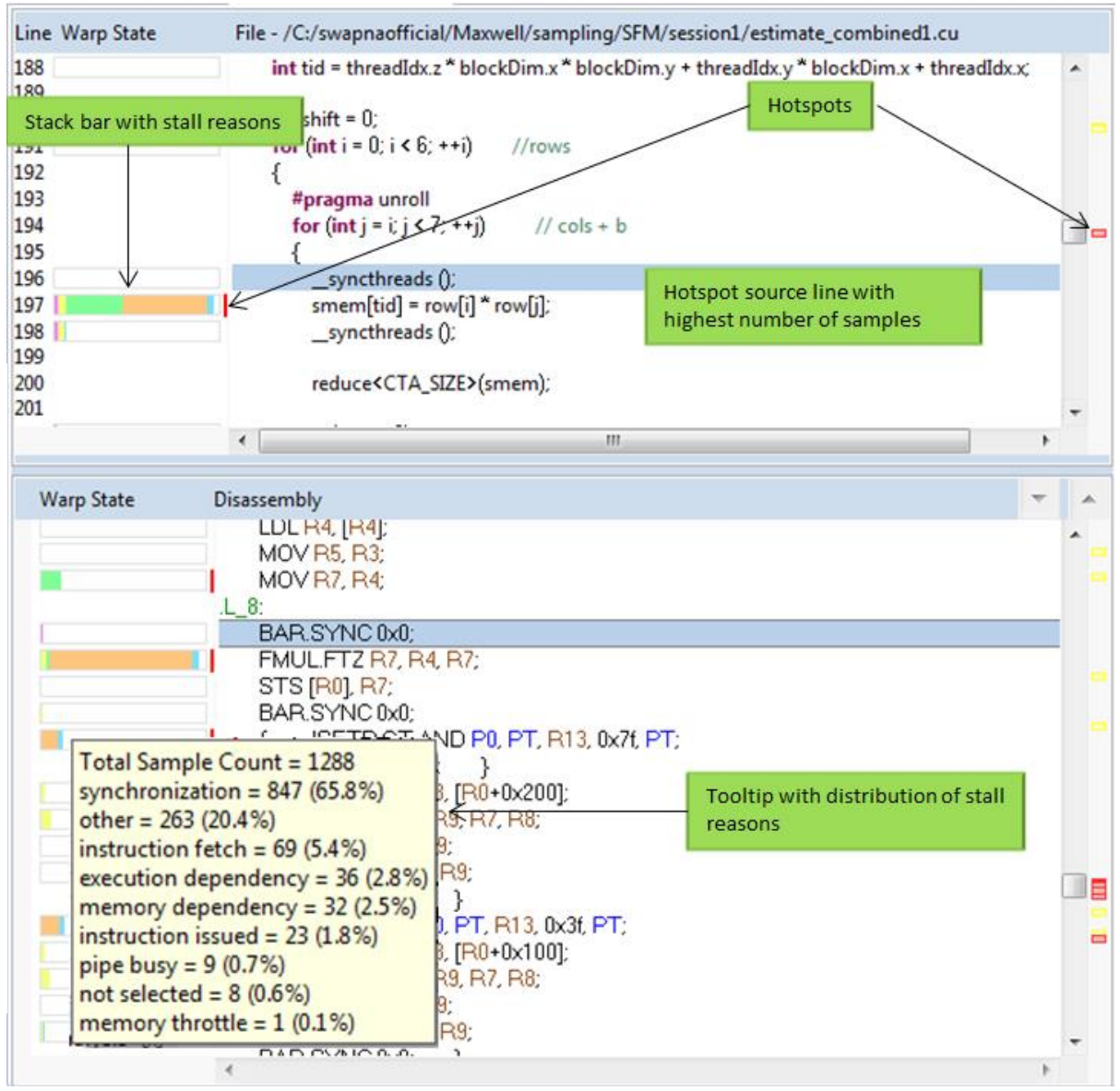


VISUAL PROFILER

PC SAMPLING UI

Pie chart for sample distribution for a CUDA function





Source-Assembly view

MULTI-PROCESS PROFILING

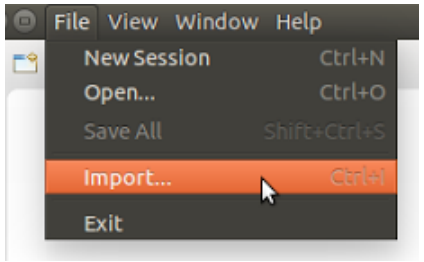
When running nvprof with multiple processes, it's useful to label each process:

```
$ nvprof -o timeline_rank%q{OMPI_COMM_WORLD_RANK} \
         --context-name "MPI Rank %q{OMPI_COMM_WORLD_RANK} \
         --process-name "MPI Rank %q{OMPI_COMM_WORLD_RANK} \
         --annotate-mpi openmpi …
```

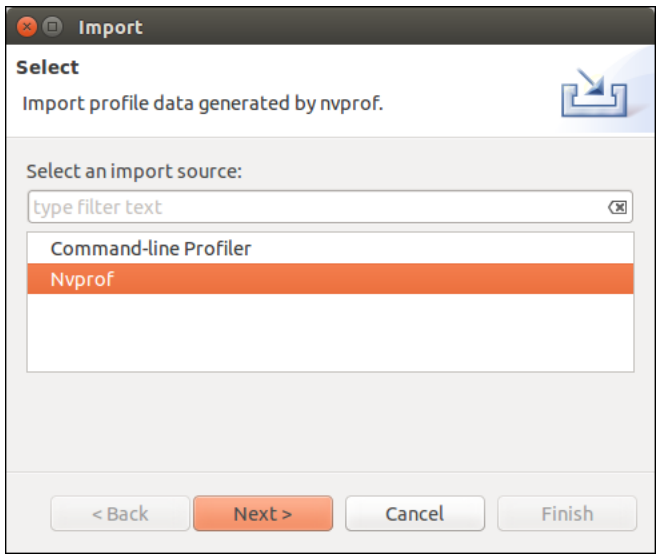
MPI PROFILING

Importing into the Visual Profiler

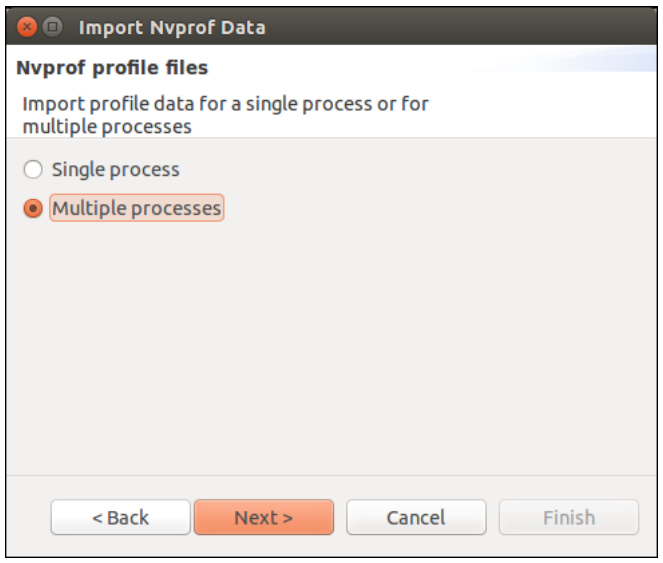
1



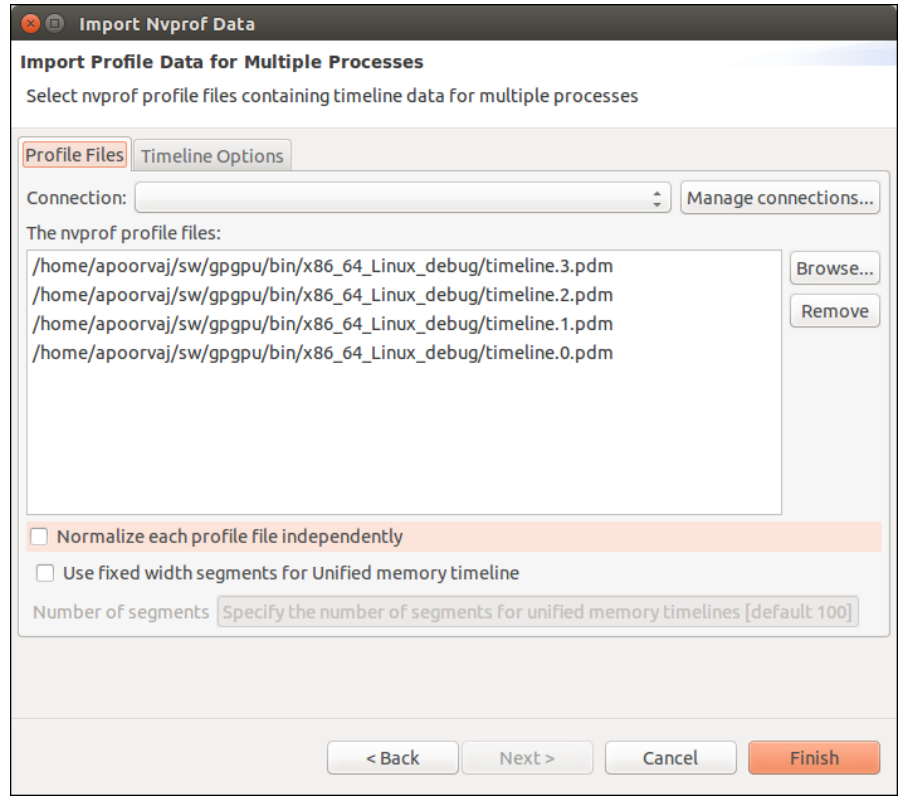
2



3



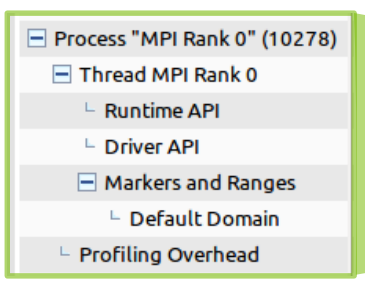
4



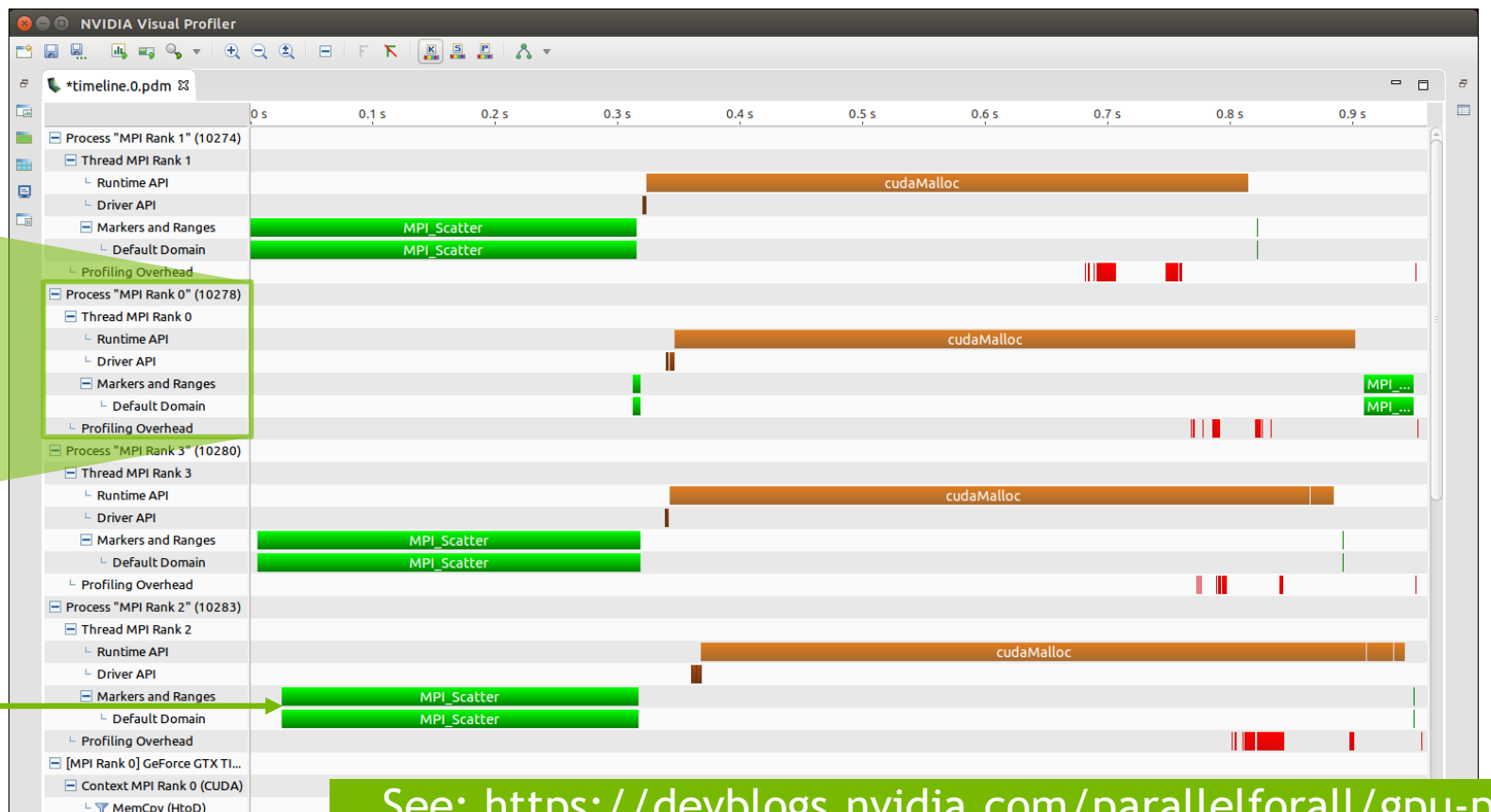
MPI PROFILING

Visual Profiler

MPI Rank-based naming



NVTX Markers & Ranges



See: https://devblogs.nvidia.com/parallelforall/gpu-pro-tip-track-mpi-calls-nvidia-visual-profiler

PROFILER API

Real applications frequently produce too much data to manage.

Profiling can be programmatically toggled:

```
#include <cuda_profiler_api.h>

cudaProfilerStart();

…

cudaProfilerStop();
```

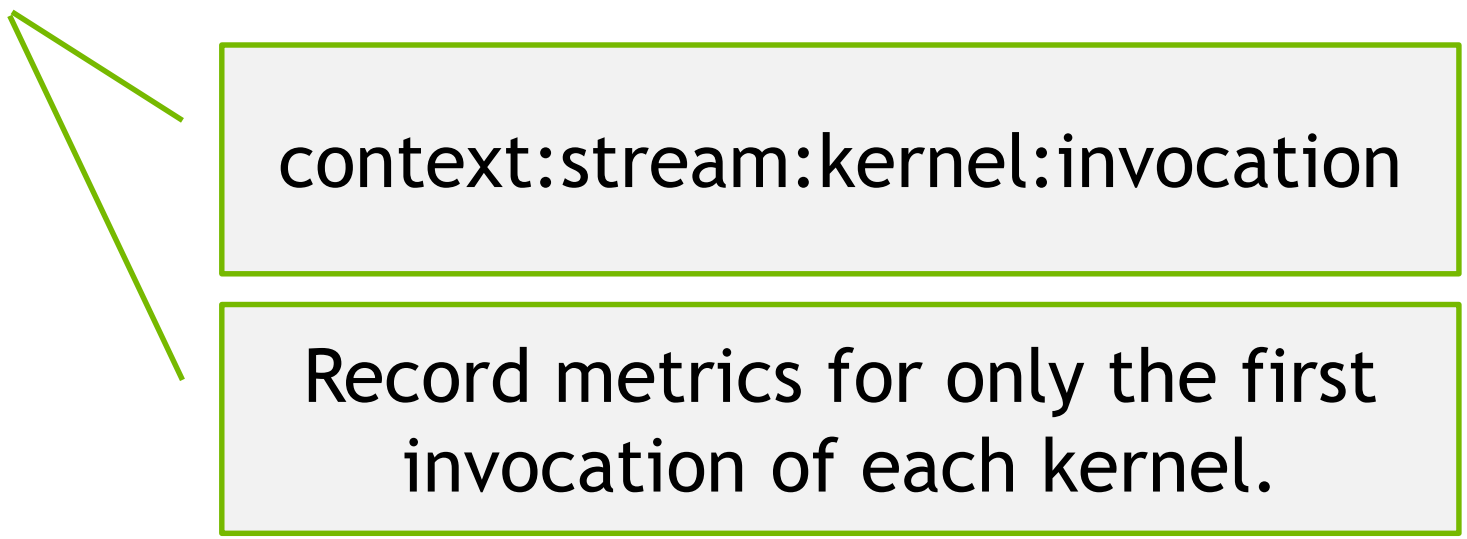
This can be paired with nvprof:

$ nvprof --profile-from-start off …

SELECTIVE PROFILING

When the profiler API still isn't enough, selectively profile kernels, particularly with performance counters.

```
$ nvprof --kernels :::1 --analysis-metrics …
```



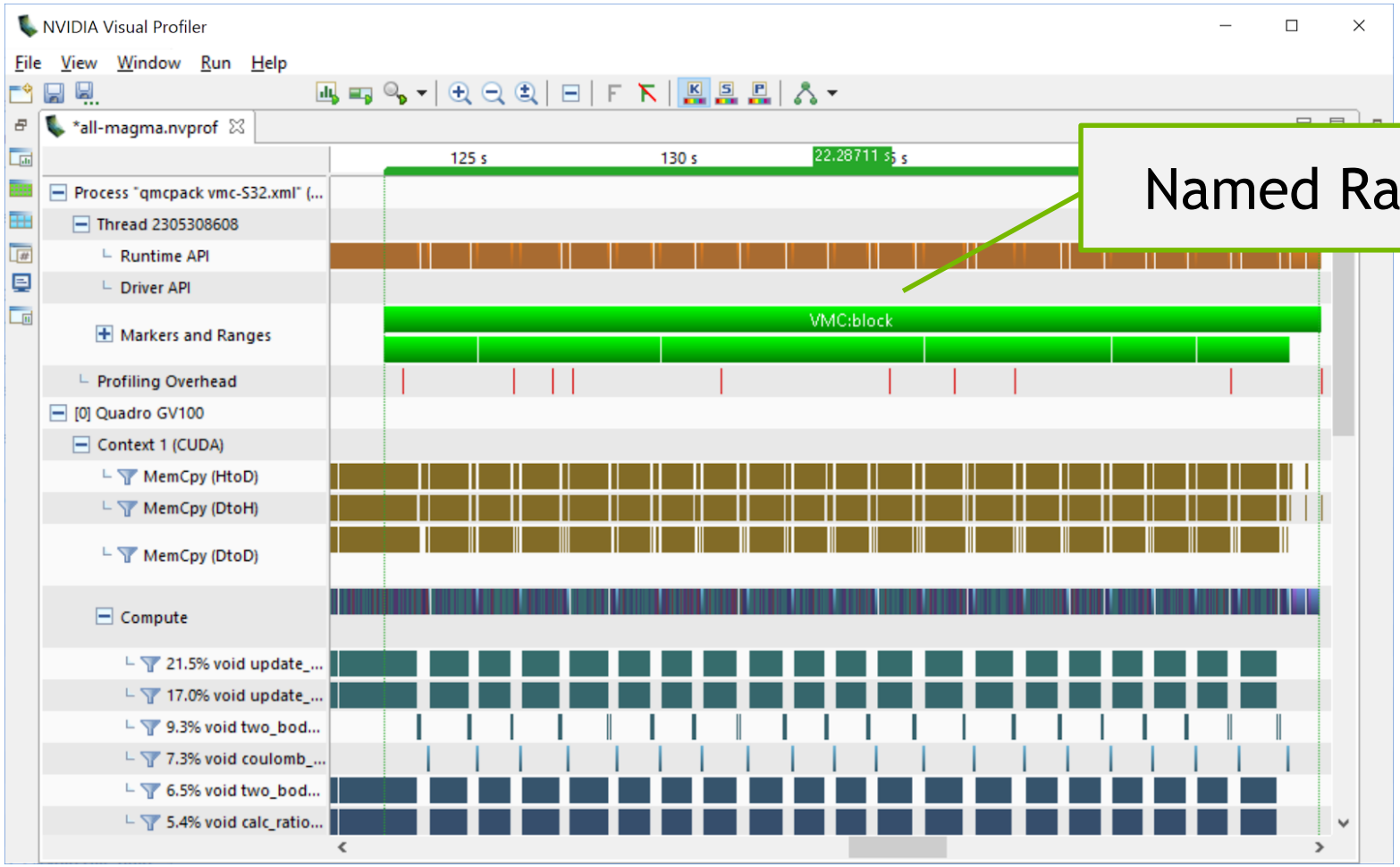
NVTX ANNOTATIONS

The NVIDIA Tools Extensions (NVTX) allow you to annotate the profile:

```
#include <nvToolsExt.h> // Link with -lnvToolsExt

nvtxRangePushA("timestep");

timestep();

nvtxRangePop();
```

See https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvtx for more features, including V3 usage.

NVTX IN VISUAL PROFILER



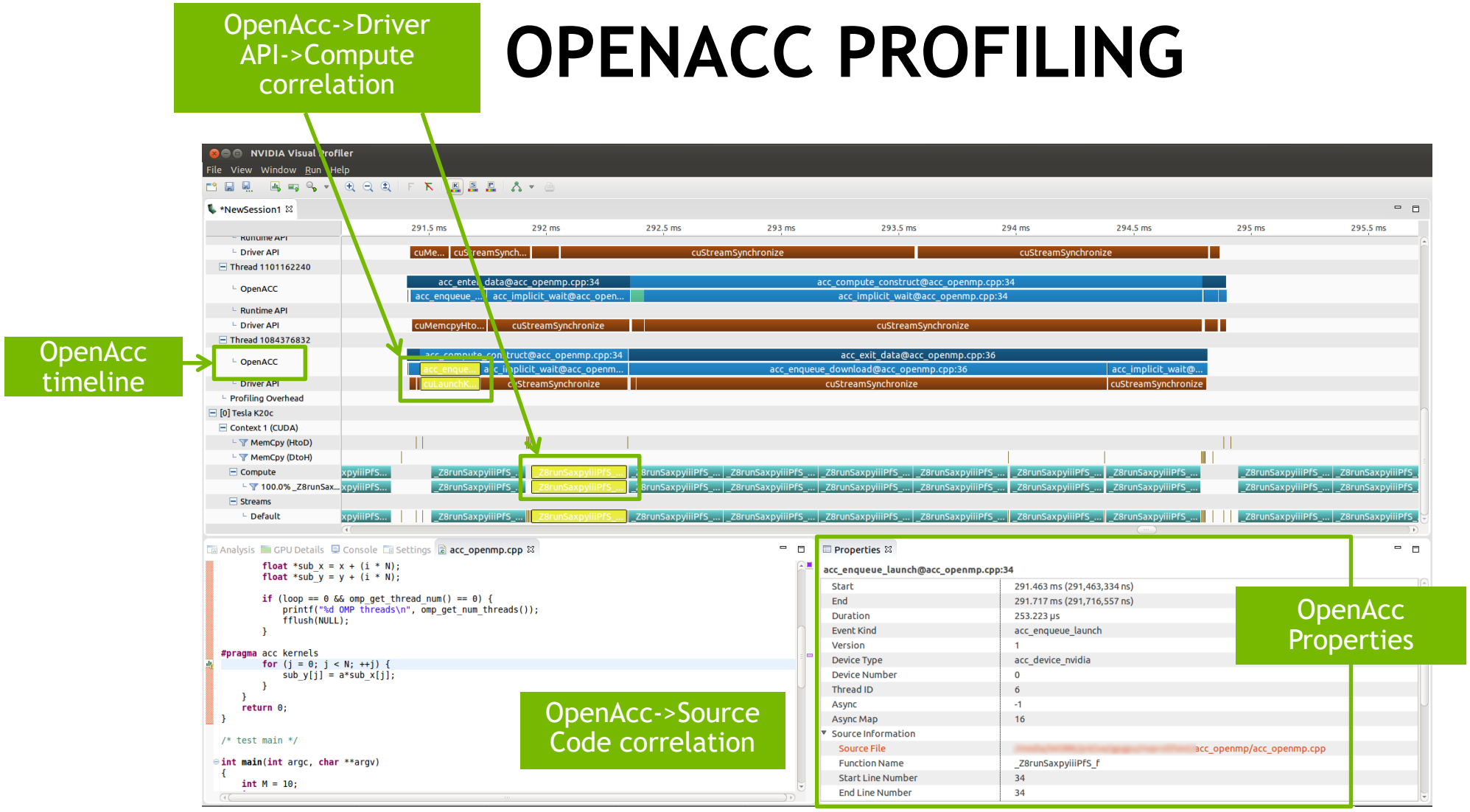
EXPORTING DATA

It’s often useful to post-process nvprof data using your favorite tool (Python, Excel, …):

```
$ nvprof --csv --log-file output.csv \
         -i profile.nvprof
```

It’s often necessary to massage this file before loading into your favorite tool.

OPENACC PROFILING



OPENMP PROFILING

Information about OpenMP regions using the OpenMP tools interface (OMPT) starting CUDA 10.0

Supported on x86_64 and Power Linux with PGI runtime 18.1+

Supported added in the CUPTI, nvprof and Visual Profiler

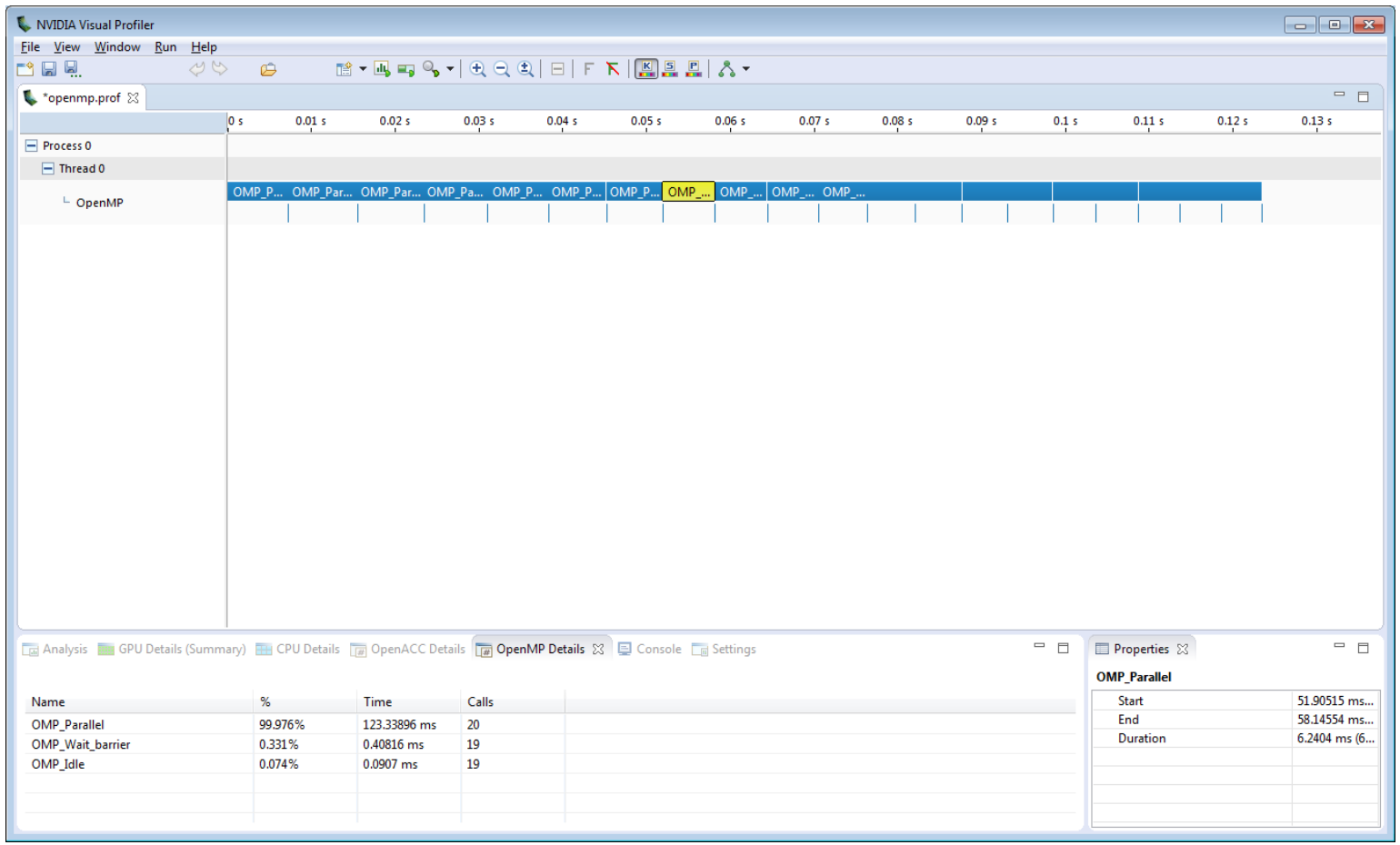
OPENMP PROFILING IN NVPROF

nvprof option openmp-profiling to enable/disable the OpenMP profiling, default on

```
$nvprof openmp-profiling on ./omp-app

            Type  Time(%)      Time     Calls       Avg       Min       Max  Name

  OpenMP (incl):   99.97%  277.10ms        20  13.855ms  13.131ms  18.151ms  omp_parallel

                    0.03%  72.728us        19  3.8270us  2.9840us  9.5610us  omp_idle

                    0.00%  7.9170us         7  1.1310us  1.0360us  1.5330us  omp_wait_barrier
```

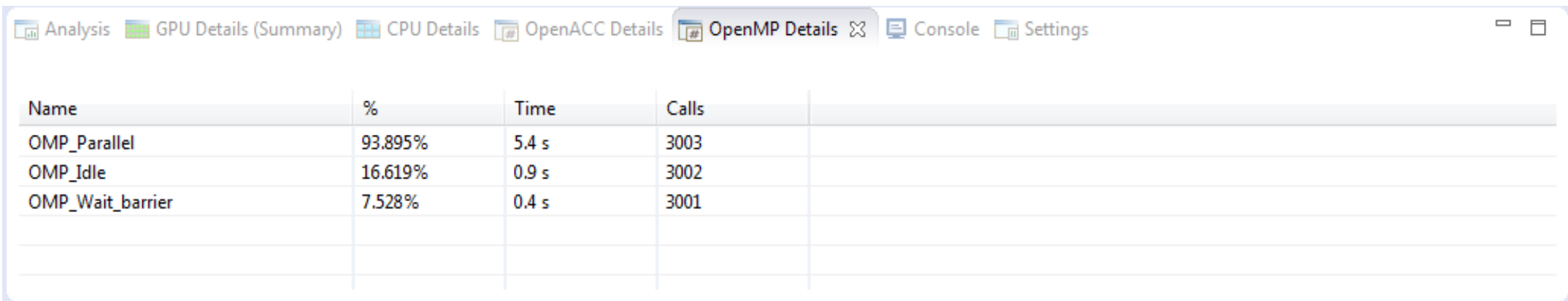
Option --print-openmp-summary to print a summary of all recorded OpenMP activities

OPENMP PROFILING IN VISUAL PROFILER



OPENMP PROFILING IN VISUAL PROFILER

Table View



Name	%	Time	Calls
OMP_Parallel	93.895%	5.4 s	3003
OMP_Idle	16.619%	0.9 s	3002
OMP_Wait_barrier	7.528%	0.4 s	3001

PROFILING NVLINK USAGE

Using nvprof+NVVP

Run nvprof multiple times to collect metrics

```
jsrun <args> nvprof --output-profile profile.<metric>.%q{OMPI_COMM_WORLD_RANK}\
                    --aggregate-mode off --event-collection-mode continuous \
                    --metrics <metric> -f
```

Use `--query-metrics` and `--query-events` for full list of metrics (-m) or events (-e)

Combine with an MPI annotated timeline file for full picture

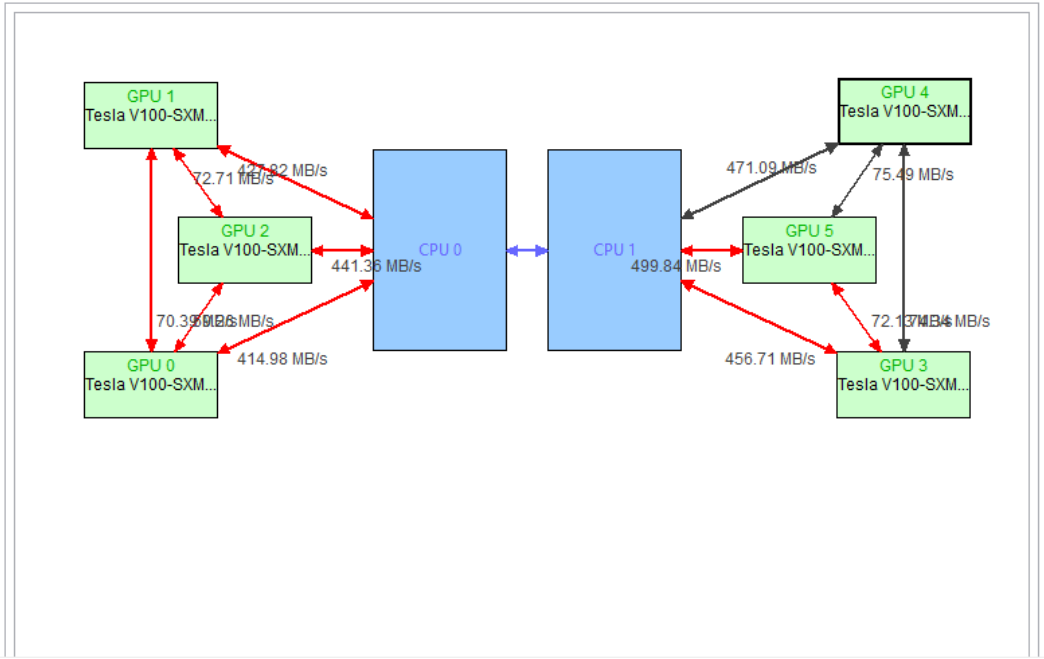
SUMMIT NVLINK TOPOLOGY

Results

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

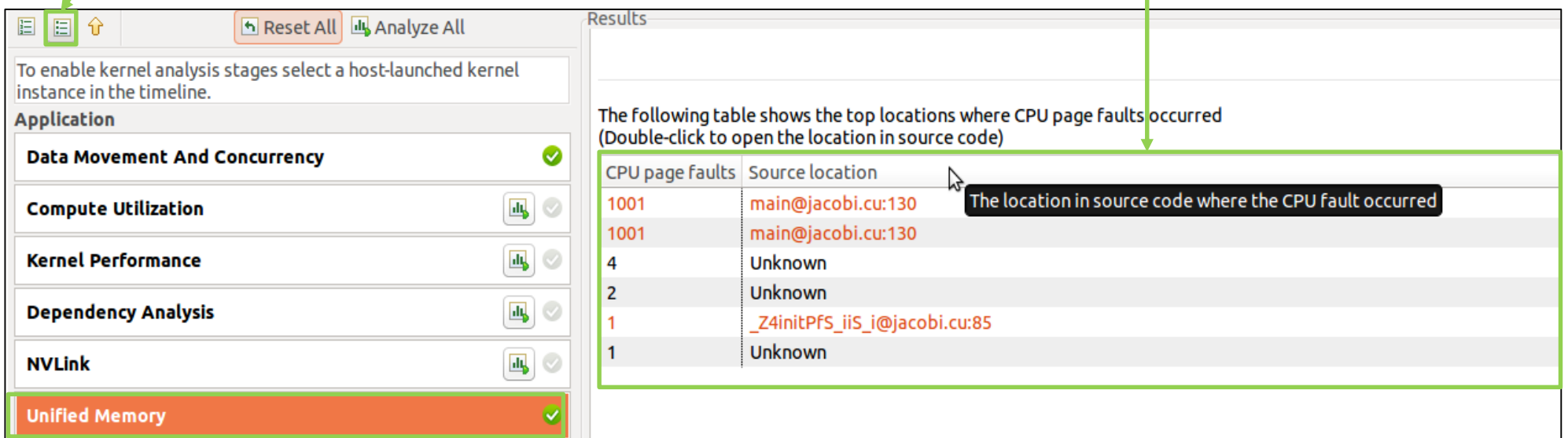
Logical NVLink	PeakBandwidth	PhysicalNVLinks	PeerAccess	SystemAccess	PeerAtomic	SystemAtomic	Utilization %	Idl
GPU0<-->CP...	100 GB/s	2	No	Yes	No	Yes	0	
GPU0<-->GP...	100 GB/s	2	Yes	No	Yes	No	0	
GPU0<-->GP...	100 GB/s	2	Yes	No	Yes	No	0	
GPU0<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU0<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU0<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU1<-->CP...	100 GB/s	2	No	Yes	No	Yes	0	
GPU1<-->GP...	100 GB/s	2	Yes	No	Yes	No	0	
GPU1<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU1<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU1<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU2<-->CP...	100 GB/s	2	No	Yes	No	Yes	0	
GPU2<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU2<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU2<-->GP...	84 GB/s	2	Yes	No	Yes	No	0	
GPU3<-->CP...	100 GB/s	2	No	Yes	No	Yes	0	
GPU3<-->GP...	100 GB/s	2	Yes	No	Yes	No	0	
GPU3<-->GP...	100 GB/s	2	Yes	No	Yes	No	0	
GPU4<-->CP...	100 GB/s	2	No	Yes	No	Yes	0	
GPU4<-->GP...	100 GB/s	2	Yes	No	Yes	No	0	
GPU5<-->CP...	100 GB/s	2	No	Yes	No	Yes	0	

Logical NVLink Throughput

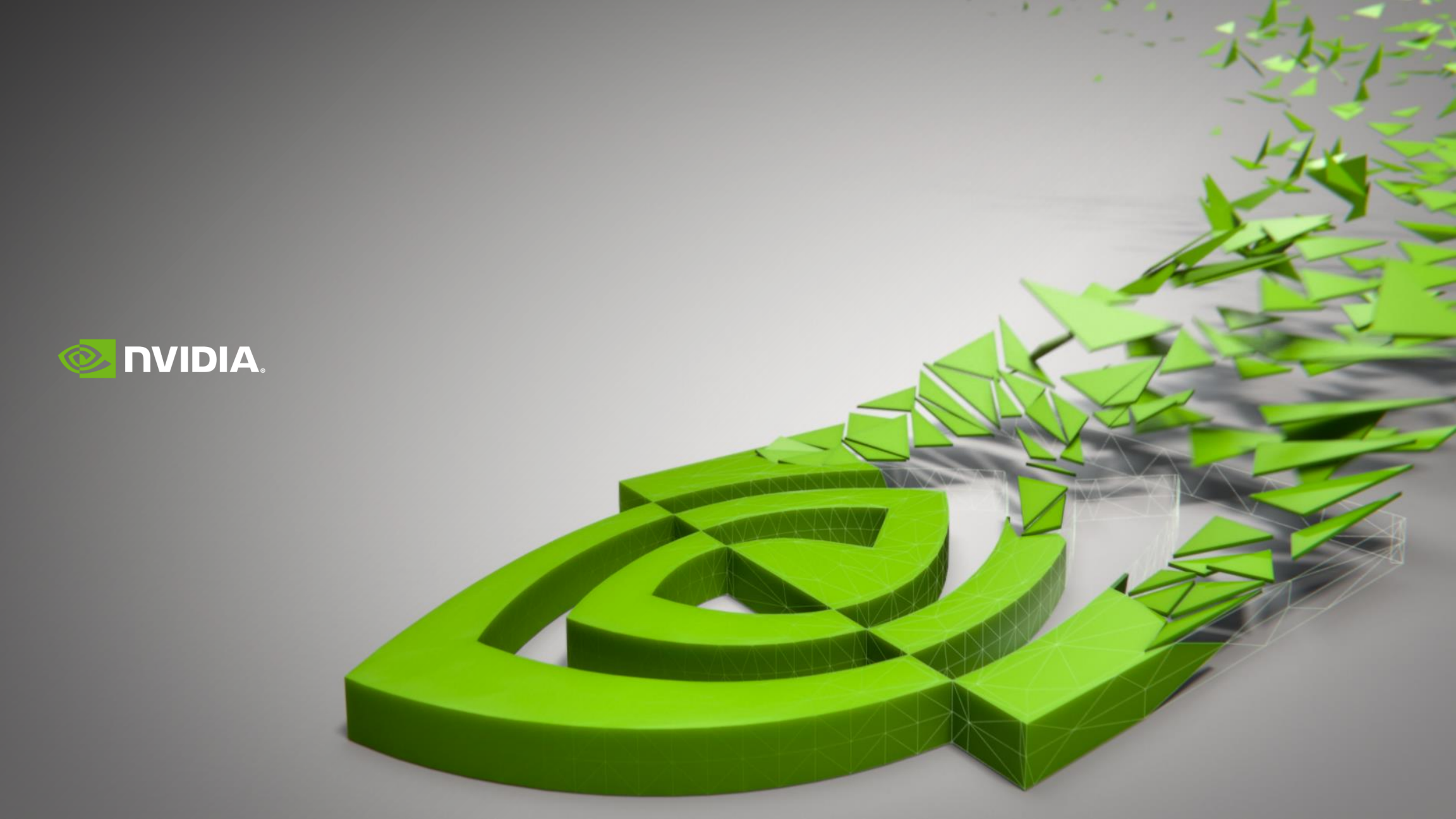
CPU PAGE FAULT SOURCE CORRELATION

Unguided Analysis

Summary of all CPU page faults



Option to collect Unified Memory information


NVIDIA