NVIDIA DEVELOPER TOOLS: NEW CAPABILITIES IN CUDA 9.X

Jeff Larkin, CAAR Workshop March 2018

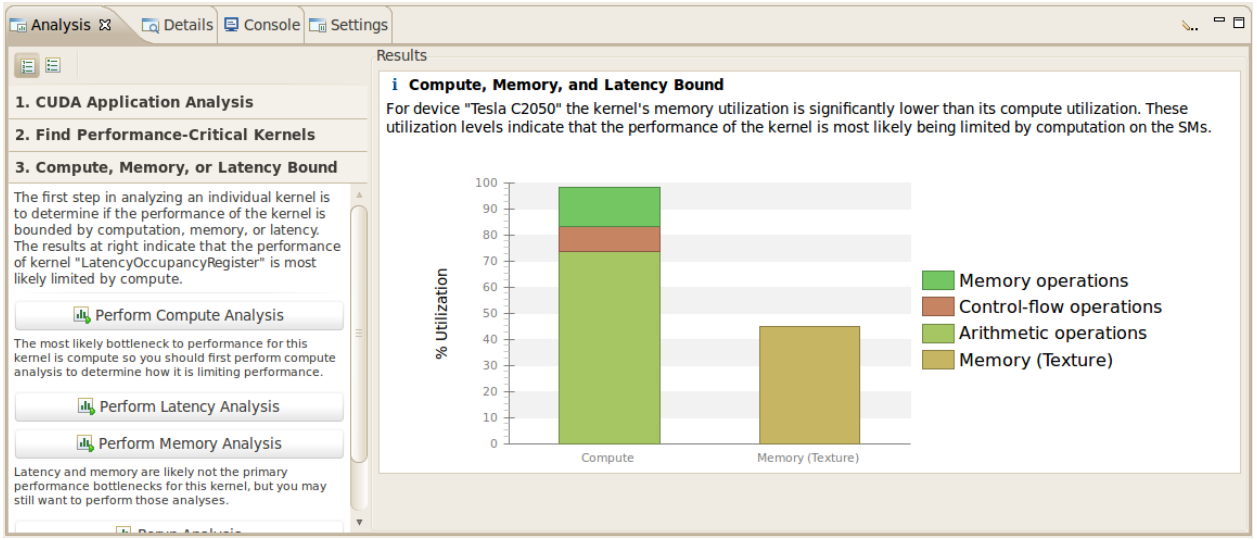
NVIDIA

CUDA TOOLS

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 9

SUPPORT FOR VOLTA ARCHITECTURE

Support for GPUs with Compute Capability 7.0

CUDA-GDB

New features post CUDA 9

- GPU core dump generation is supported on Volta-MPS
- Reading lightweight GPU core dump files is supported

```
$ time CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1
CUDA_ENABLE_LIGHTWEIGHT_COREDUMP=1 ./simple_cuda_program

Before: 9.85s user 12.33s system 173% cpu 12.794 total

After:    0.41s user   1.36s system   75% cpu   2.350 total
```

CUDA-MEMCHECK

- Support for host API functions with pitch parameter.
- ***Initial support for the Cooperative Groups programming model.***
- Support for shared memory atomic instructions.
- ***Support for detecting invalid accesses to global memory on Pascal and later architectures that extend beyond the end of an allocation.***
- Support for limiting the numbers of errors printed by cuda-memcheck.
- Racecheck analysis reports are assigned a severity level.
- Default print level changed from INFO to WARN.
- ***A new command line option to report deprecated instructions even when they are used in safe execution paths. (post CUDA 9)***

CUDA VISUAL PROFILER

Enhancements in CUDA 9.0

Unified Memory

NVLink

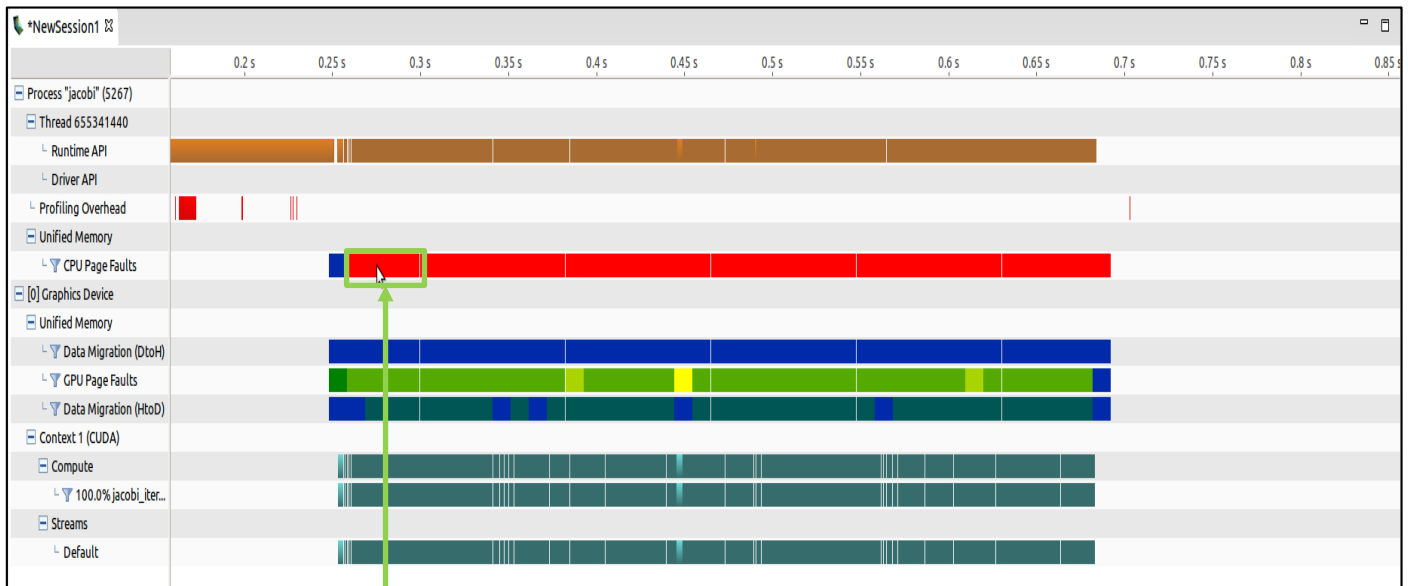
Multi-hop remote profiling

Tracing and profiling of Cooperative Kernel launches

PC sampling

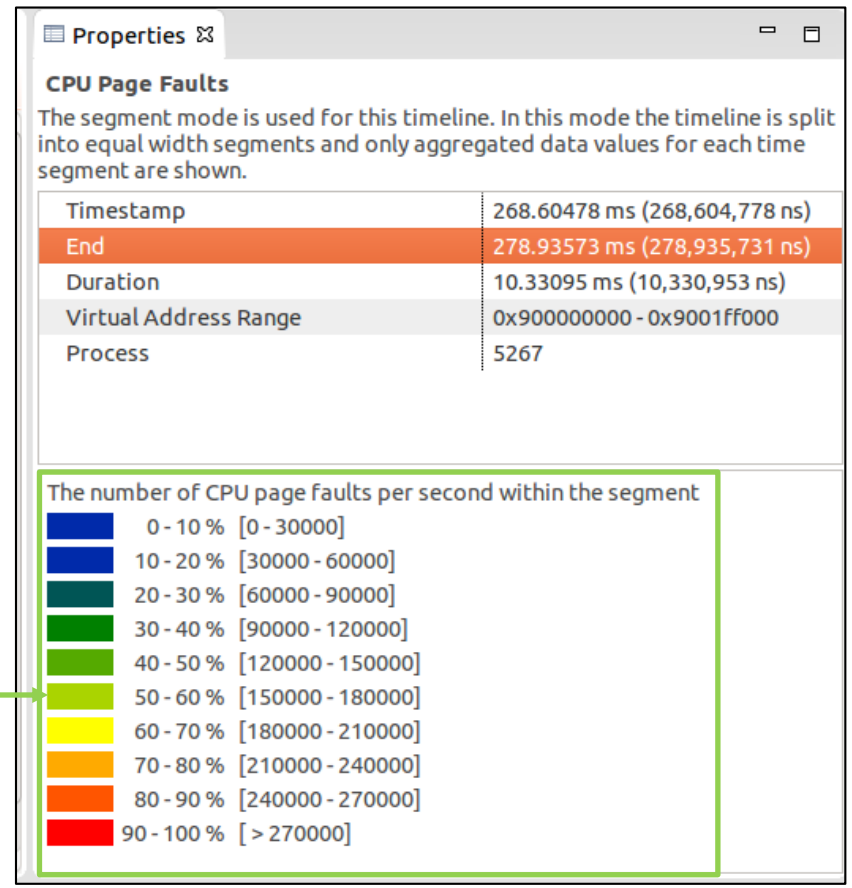
VISUAL PROFILER

Segment mode timeline



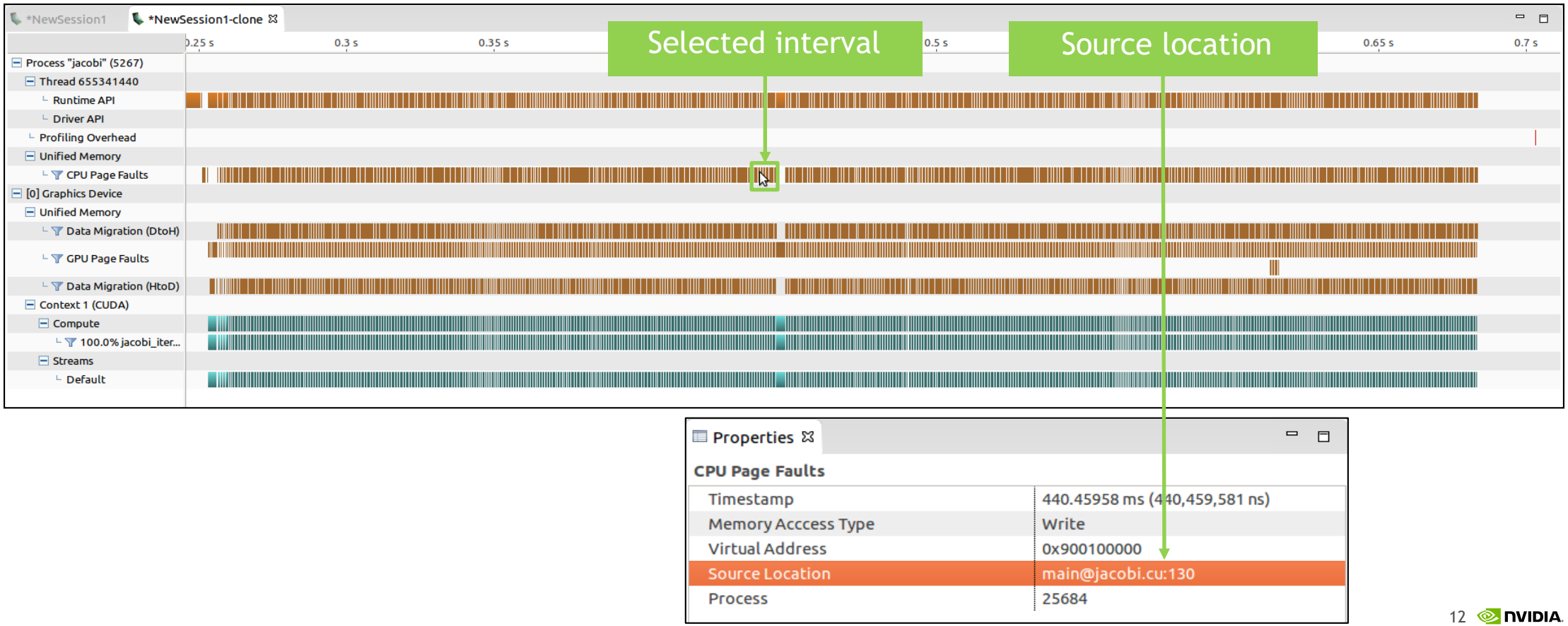
Segment mode interval

Heat map for CPU page faults



VISUAL PROFILER

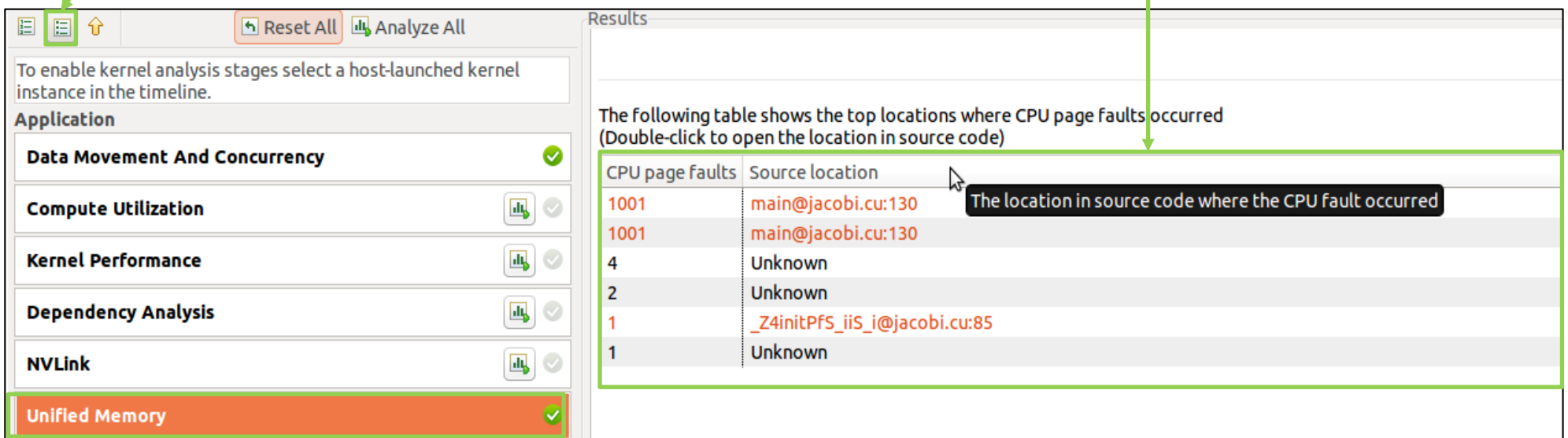
CPU Page Fault Source Correlation



CPU PAGE FAULT SOURCE CORRELATION

Unguided Analysis

Summary of all CPU page faults



Option to collect Unified Memory information

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

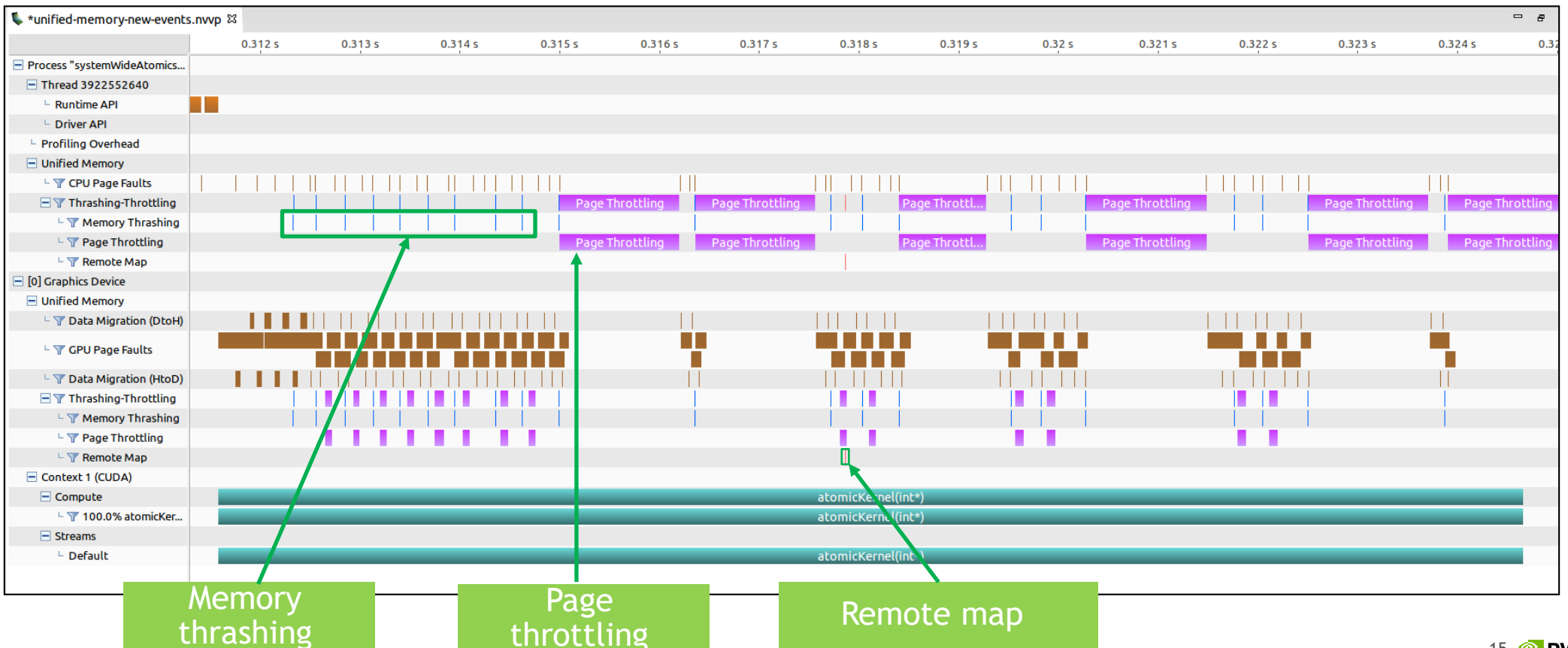
Source line causing CPU page fault

*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
```

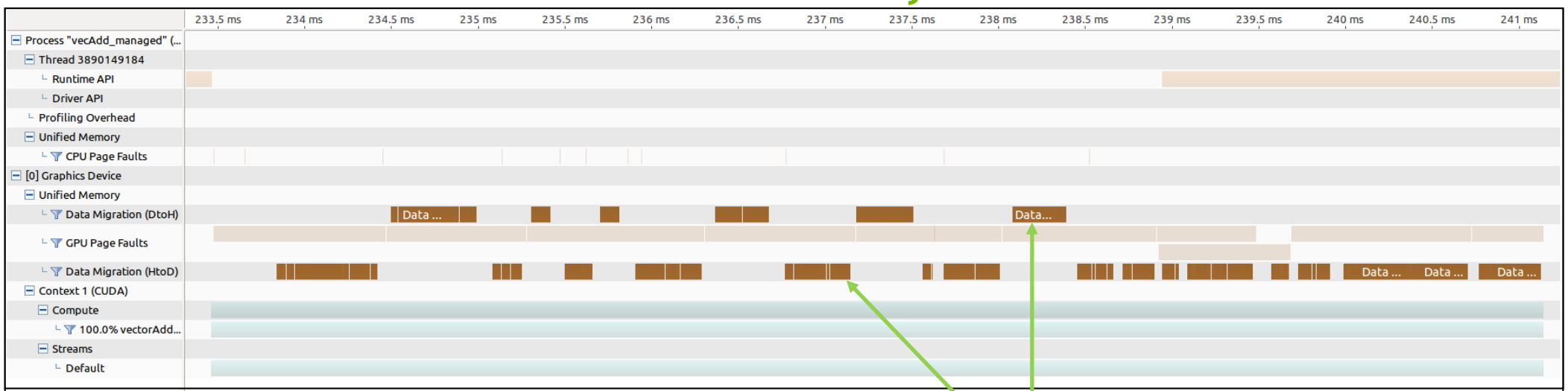
VISUAL PROFILER - NEW UNIFIED MEMORY EVENTS

Page throttling, Memory thrashing, Remote map



VISUAL PROFILER

Filter and Analyze



Filtered intervals

OPTIMIZATION

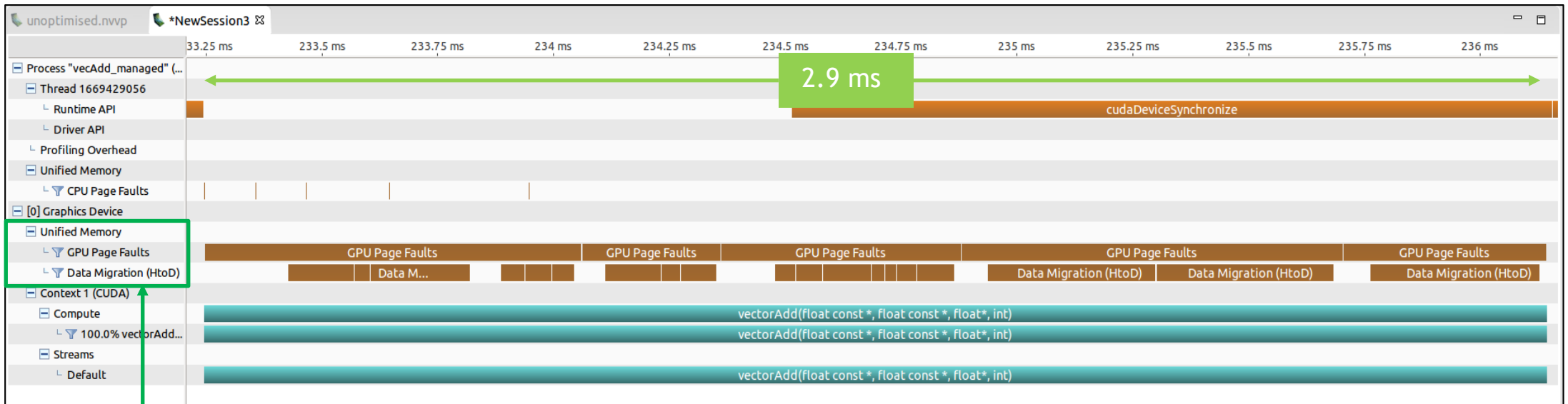
OLD

```
int threadsPerBlock = 256;
int numBlocks = (length + threadsPerBlock – 1) / threadsPerBlock;

kernel<<< numBlocks , threadsPerBlock >>>(A, B, C, length);
```

NEW

```
int threadsPerBlock = 256;
int numBlocks = (length + threadsPerBlock – 1) / threadsPerBlock;

cudaMemAdvise(A, size, cudaMemAdviseSetReadMostly, 0);
cudaMemAdvise(B, size, cudaMemAdviseSetReadMostly, 0);

kernel<<< numBlocks , threadsPerBlock >>>(A, B, C, length);
```

OPTIMIZED APPLICATION

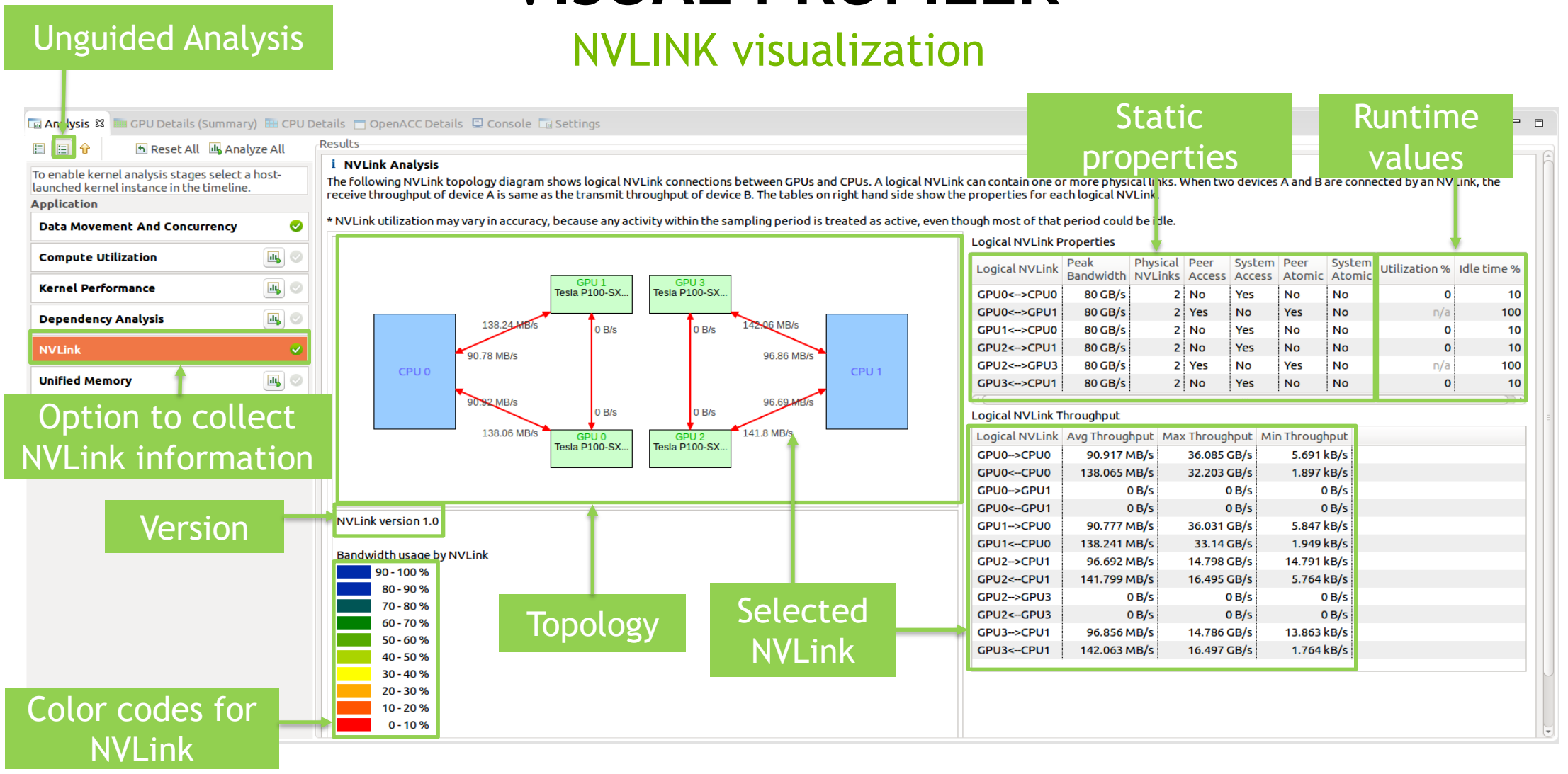


No DtoH Migrations and thrashing

Speedup 4x (2.9 vs 12.2)

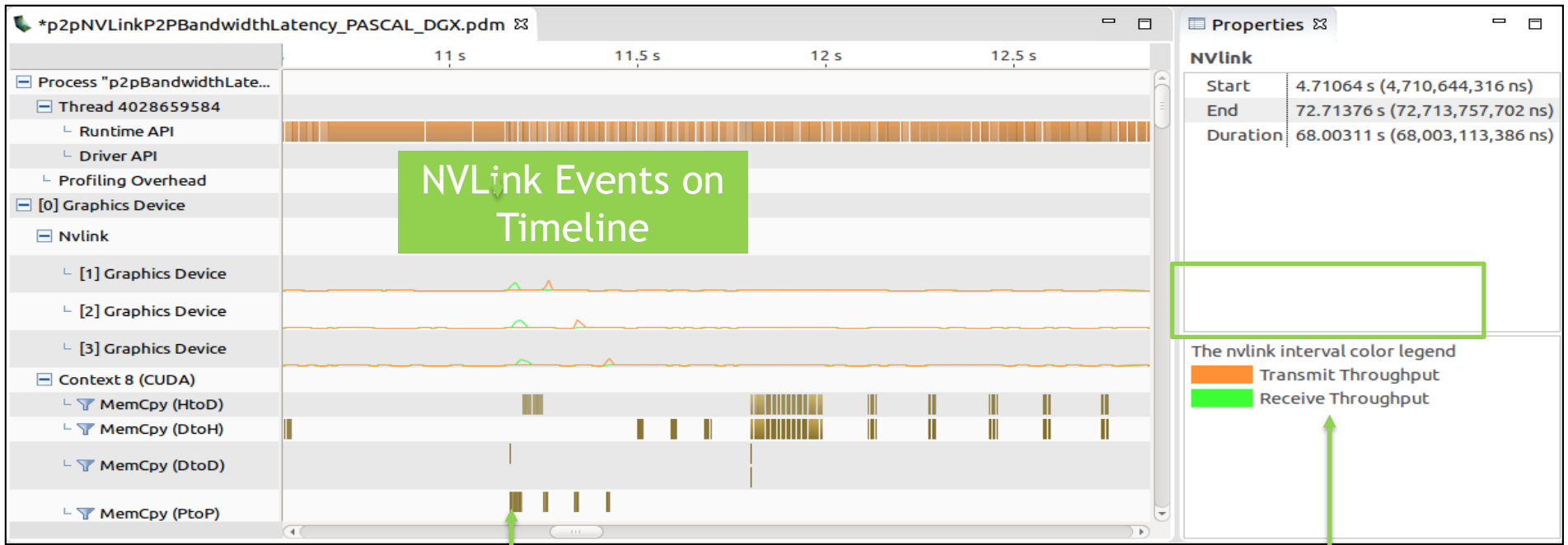
VISUAL PROFILER

NVLINK visualization



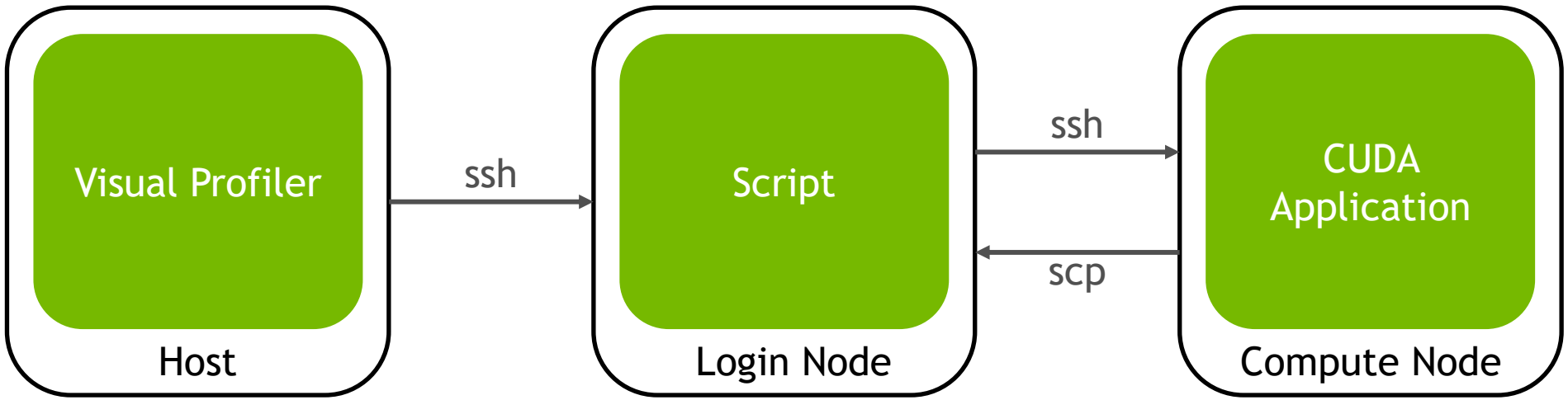
VISUAL PROFILER

NVLink events on timeline - Segment mode



VISUAL PROFILER

Multi-hop remote profiling

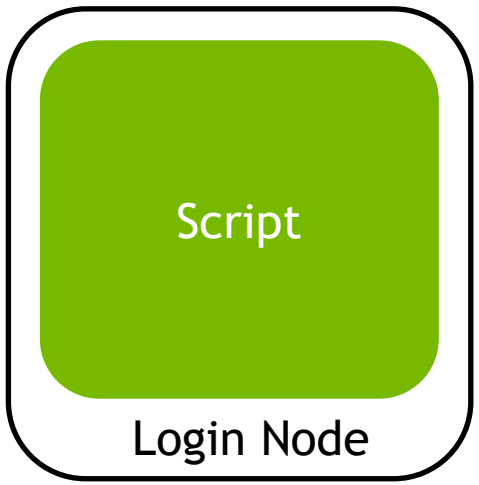


Script available on github: https://github.com/NVIDIA/cuda-profiler/tree/master/one_hop_profiling

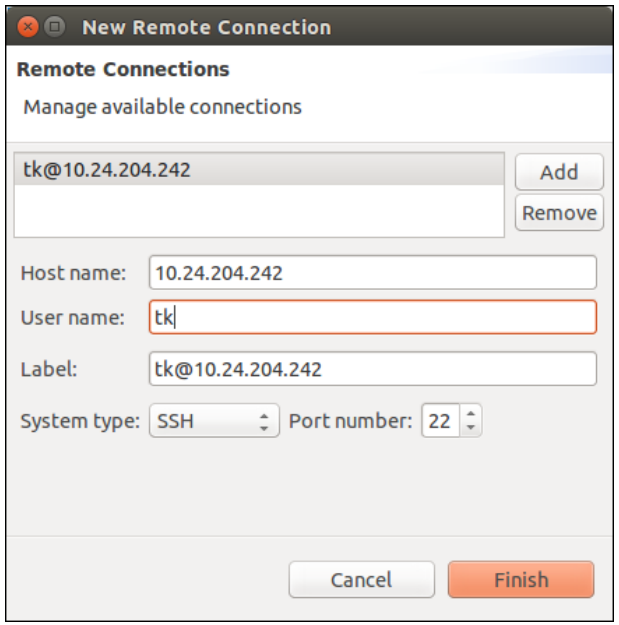
VISUAL PROFILER

Multi-hop remote profiling - One-Time Setup

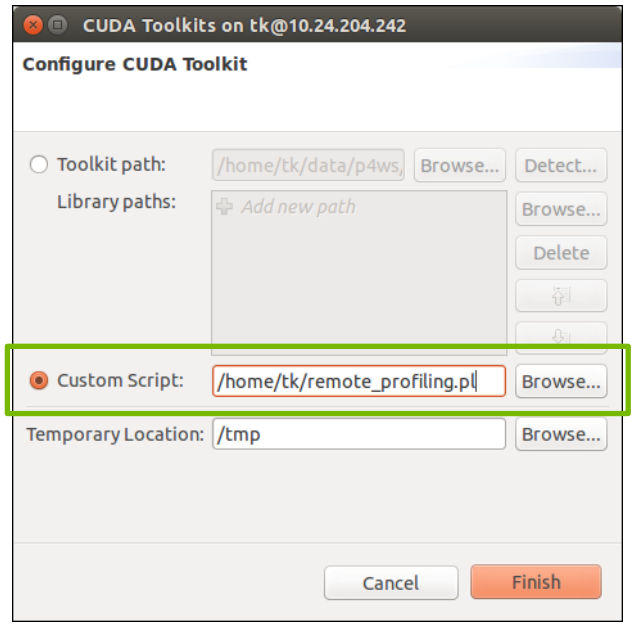
1 Configure script on the login node



2 Connect Visual Profiler to the login node



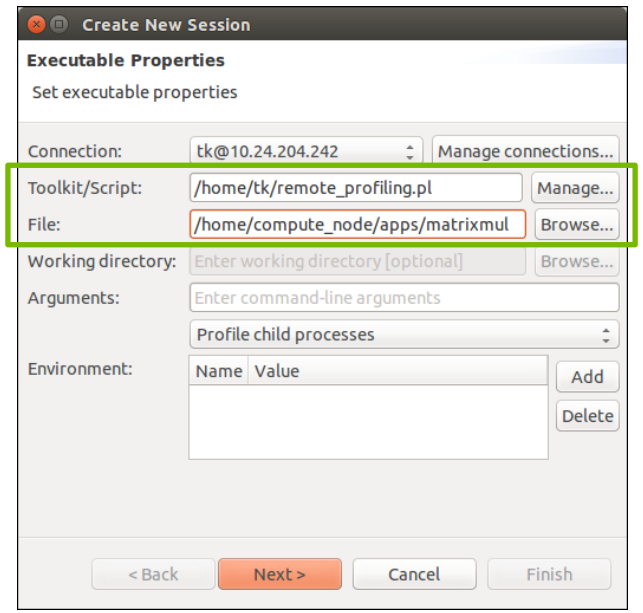
3 Use the custom script option



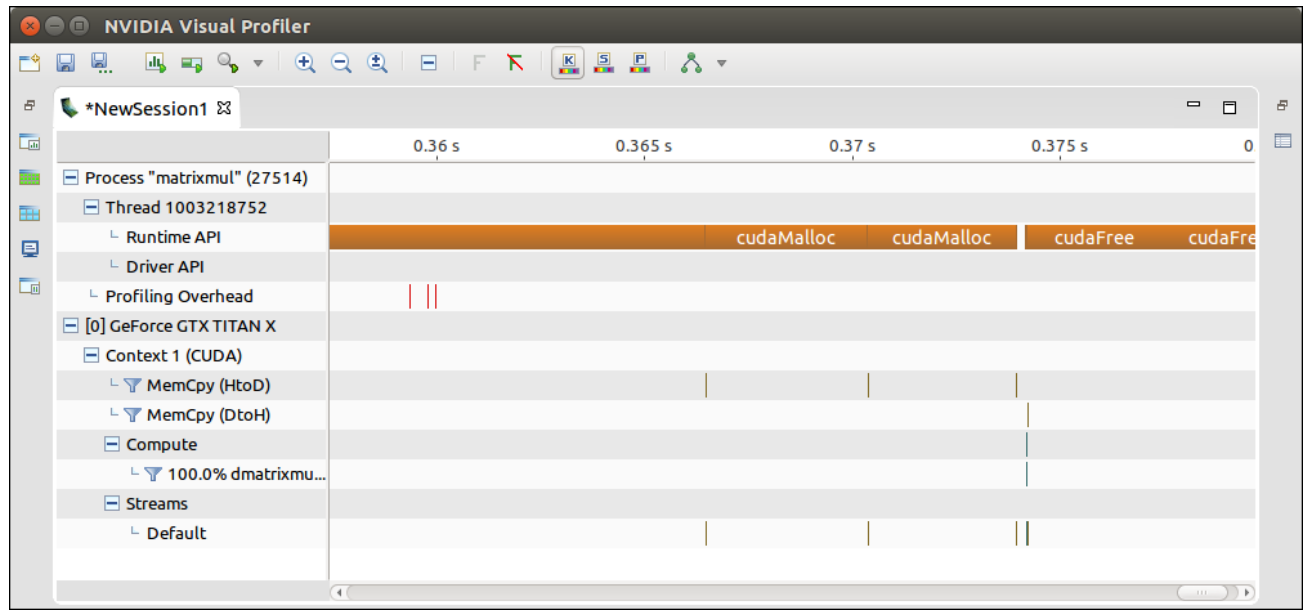
VISUAL PROFILER

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

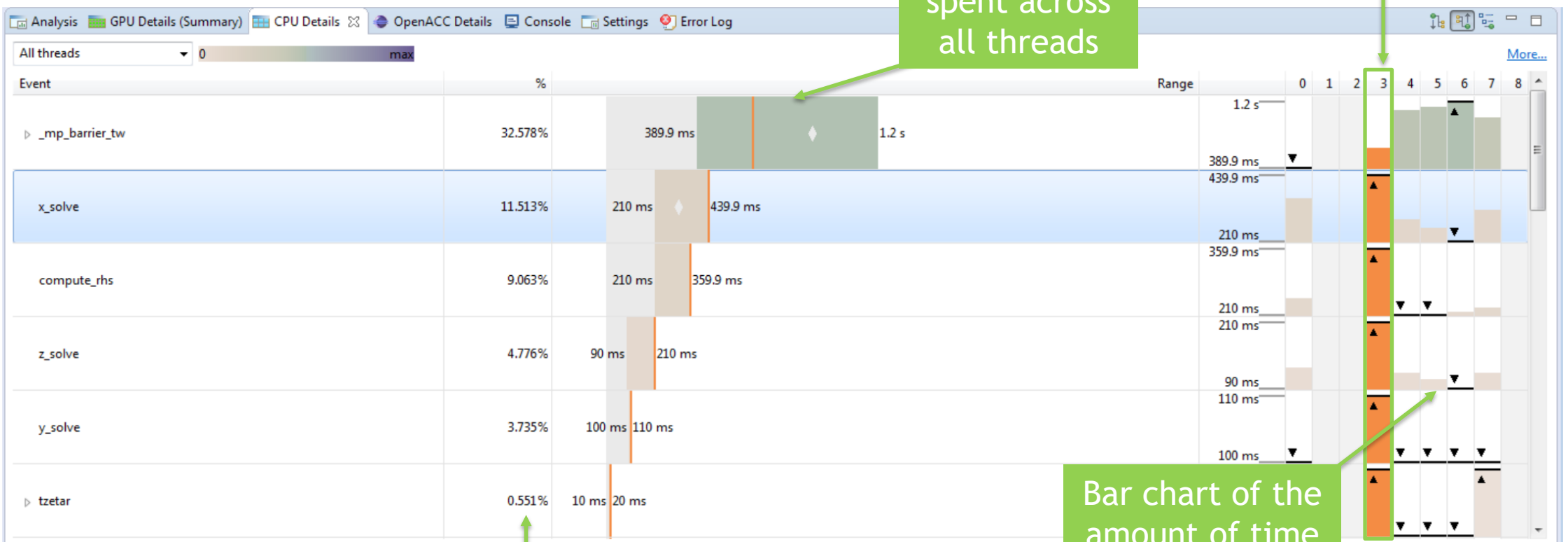


VISUAL PROFILER

CPU Sampling (post CUDA 9)

Selected thread is highlighted in Orange

Range of time spent across all threads

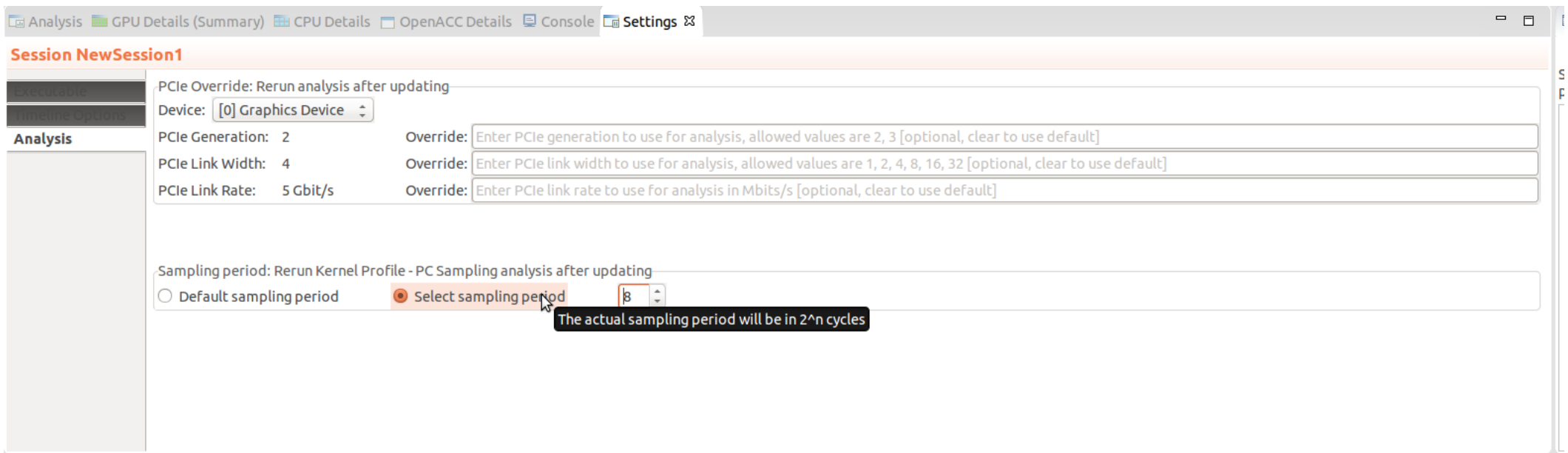


Bar chart of the amount of time spent by thread

Percentage of time spent collectively by all threads

VISUAL PROFILER - PC SAMPLING

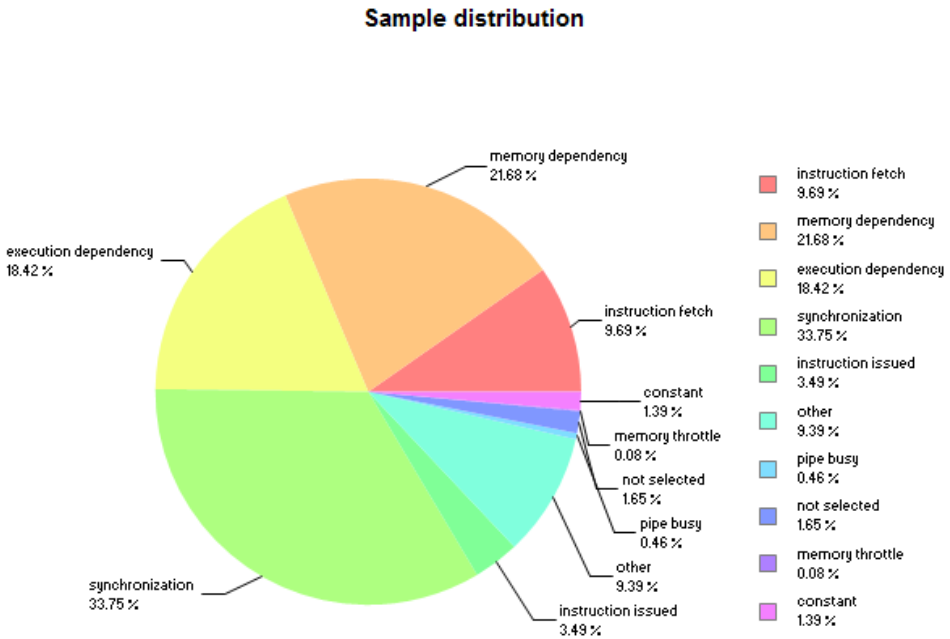
Option to select sampling period (post CUDA 9)

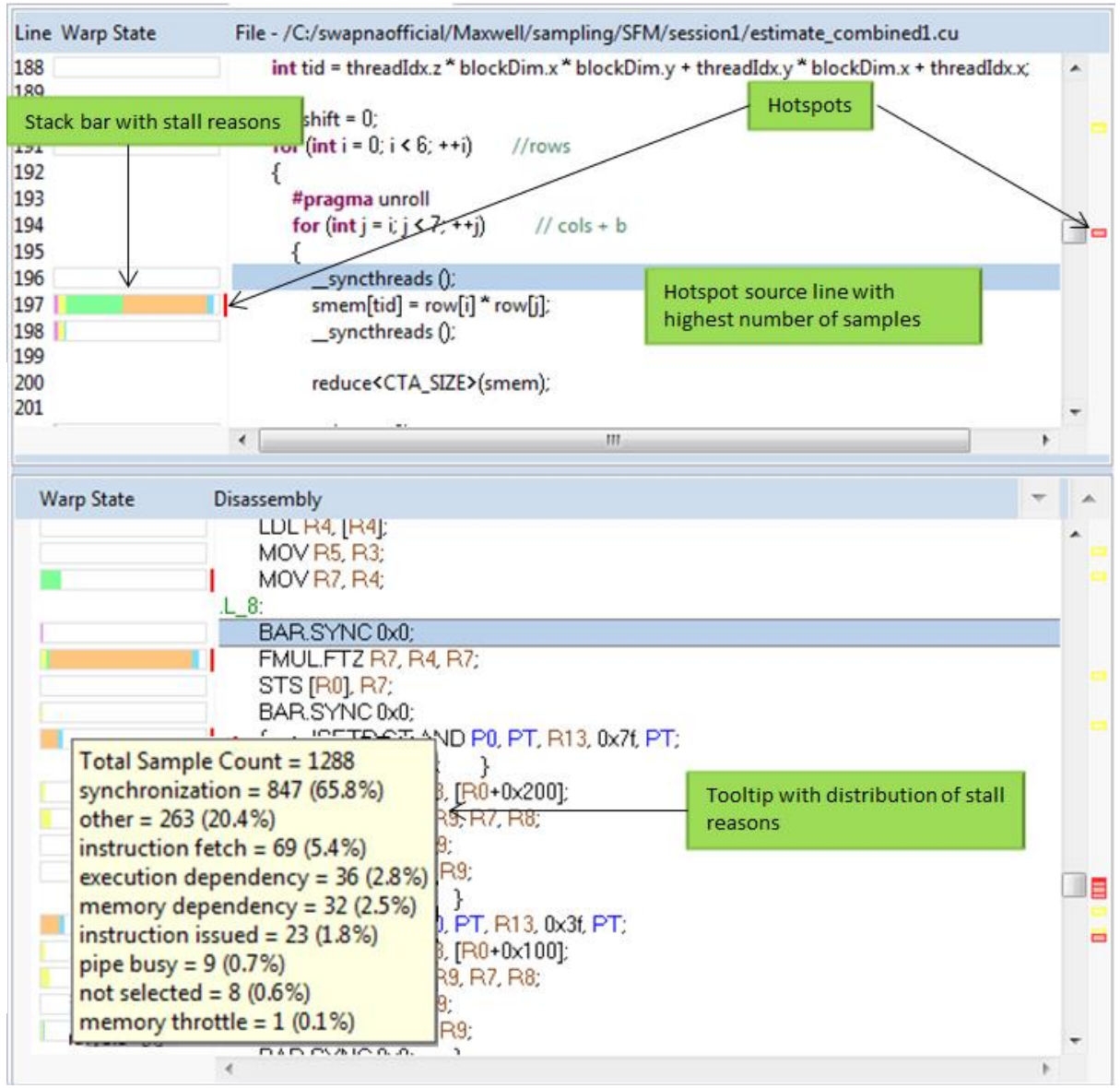


VISUAL PROFILER

PC SAMPLING UI

Pie chart for sample distribution for a CUDA function





Source-Assembly view

MPI PROFILING

MPI PROFILING

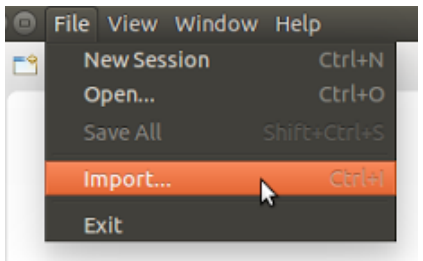
nvprof

```
$ LD_PRELOAD="libnvtx_pmpi.so mpirun -n 4 nvprof --process-name "MPI Rank
%q{PMIX_RANK}" --context-name "MPI Rank %q{PMIX_RANK}" -o
timeline.%q{PMIX_RANK}.nvprof ./simpleMPI
Running on 4 nodes
==21977== NVPROF is profiling process 21977, command: ./simpleMPI
==21983== NVPROF is profiling process 21983, command: ./simpleMPI
==21979== NVPROF is profiling process 21979, command: ./simpleMPI
==21982== NVPROF is profiling process 21982, command: ./simpleMPI
<program output>
==21982== Generated result file: timeline.0.nvprof
==21977== Generated result file: timeline.3.nvprof
==21983== Generated result file: timeline.1.nvprof
==21979== Generated result file: timeline.2.nvprof
```

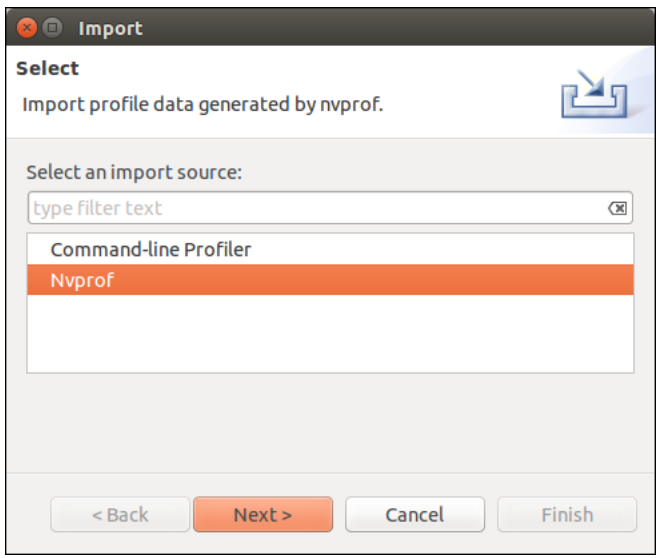
MPI PROFILING

Importing into the Visual Profiler

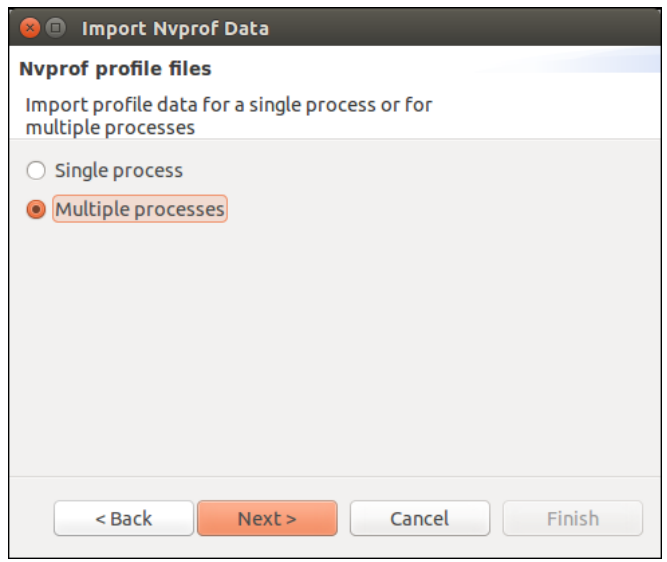
1



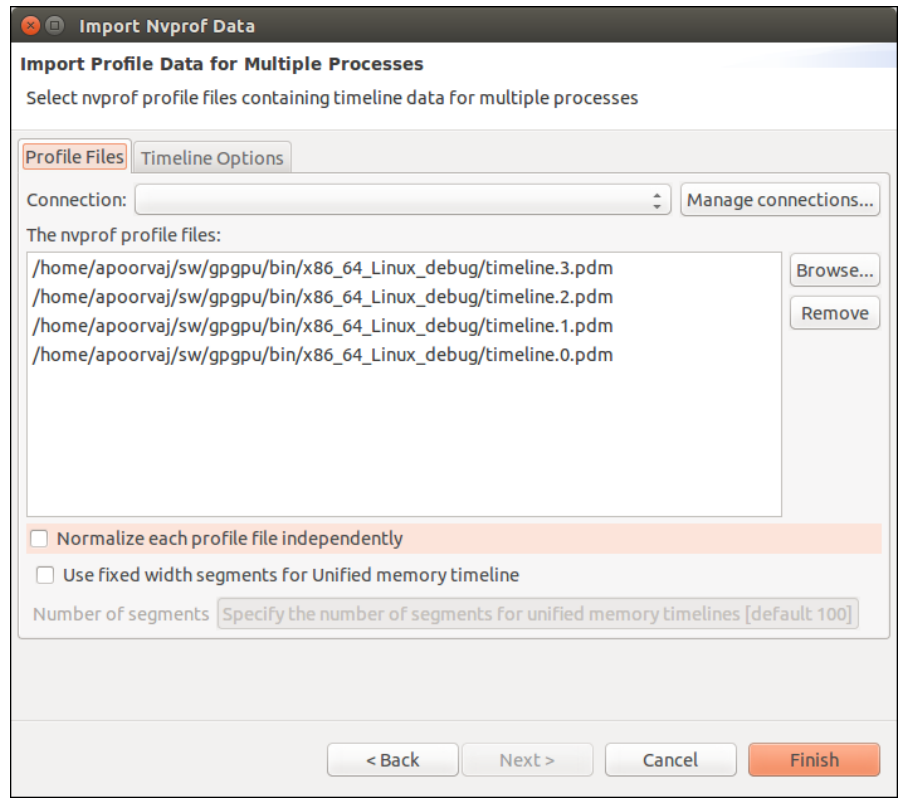
2



3



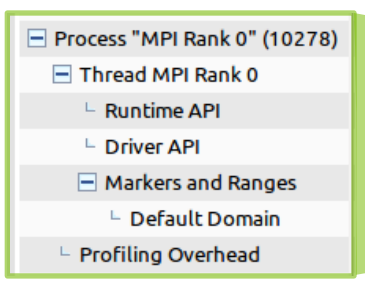
4



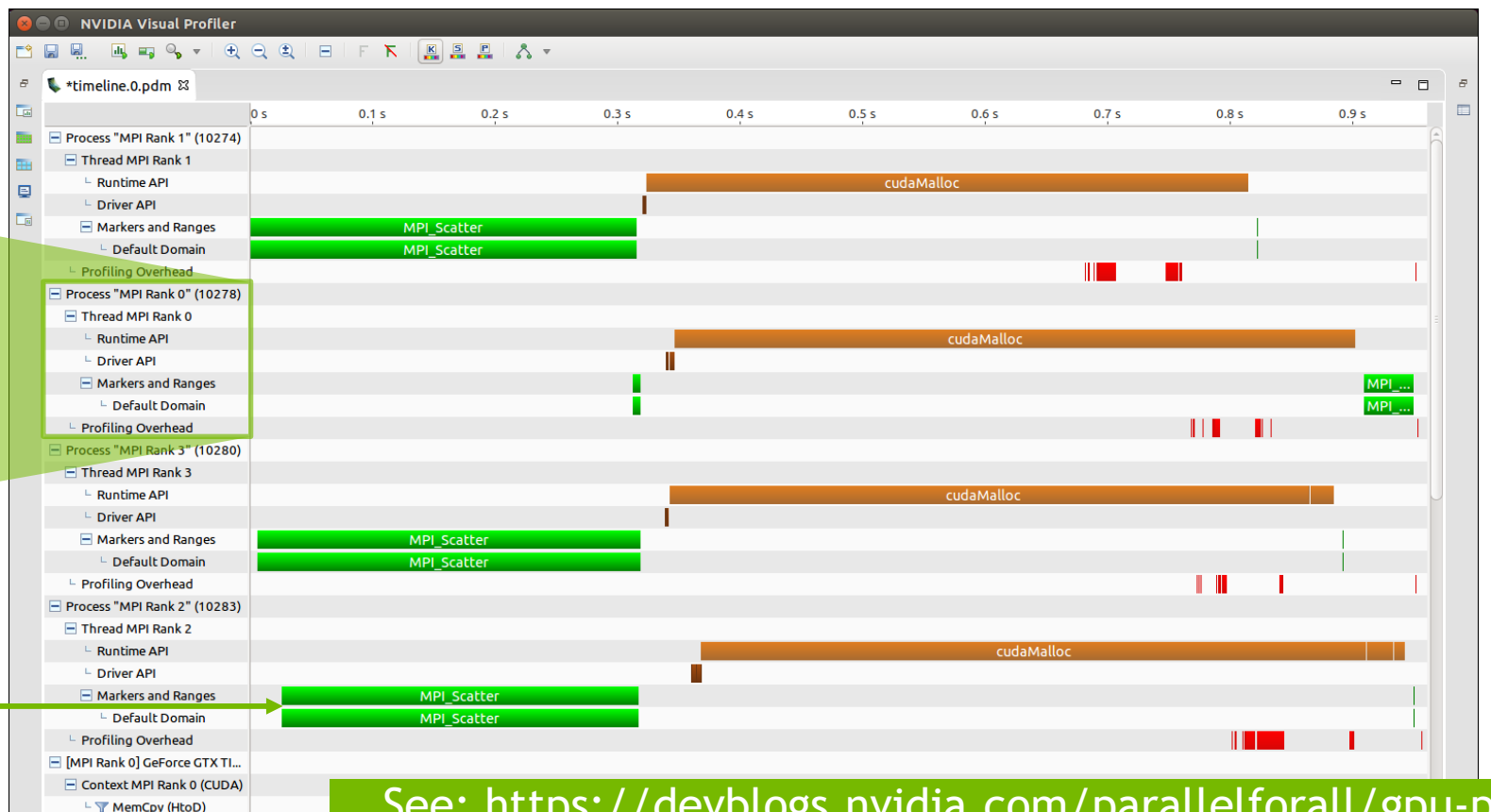
MPI PROFILING

Visual Profiler

MPI Rank-based naming



NVTX Markers & Ranges



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

MPI PROFILING

MPI + NVTX

Manual mode

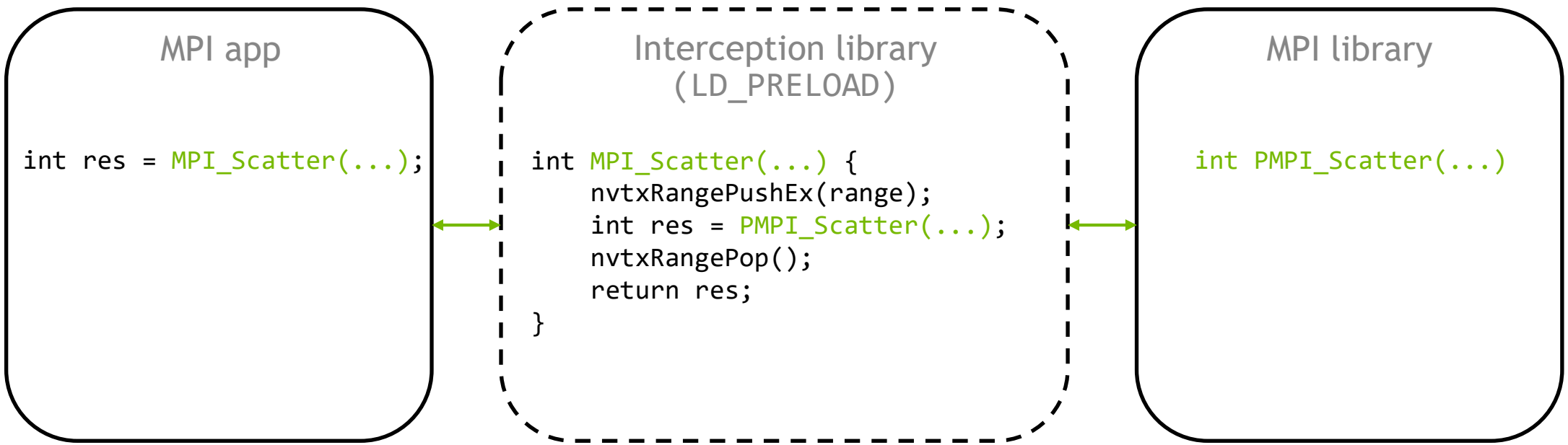
```
nvtxEventAttributes_t range = {0};
range.message.ascii = "MPI_Scatter";
nvtxRangePushEx(range);
int result = MPI_Scatter(...);
nvtxRangePop();
```

Interception mode

1. Auto-generate `mpi_interception.so`
   https://devblogs.nvidia.com/parallelforall/gpu-pro-tip-track-mpi-calls-nvidia-visual-profiler/
2. `LD_PRELOAD=mpi_interception.so`
3. Run your MPI app with nvprof.
   MPI calls will be auto-annotated using NVTX.

MPI PROFILING

Interception

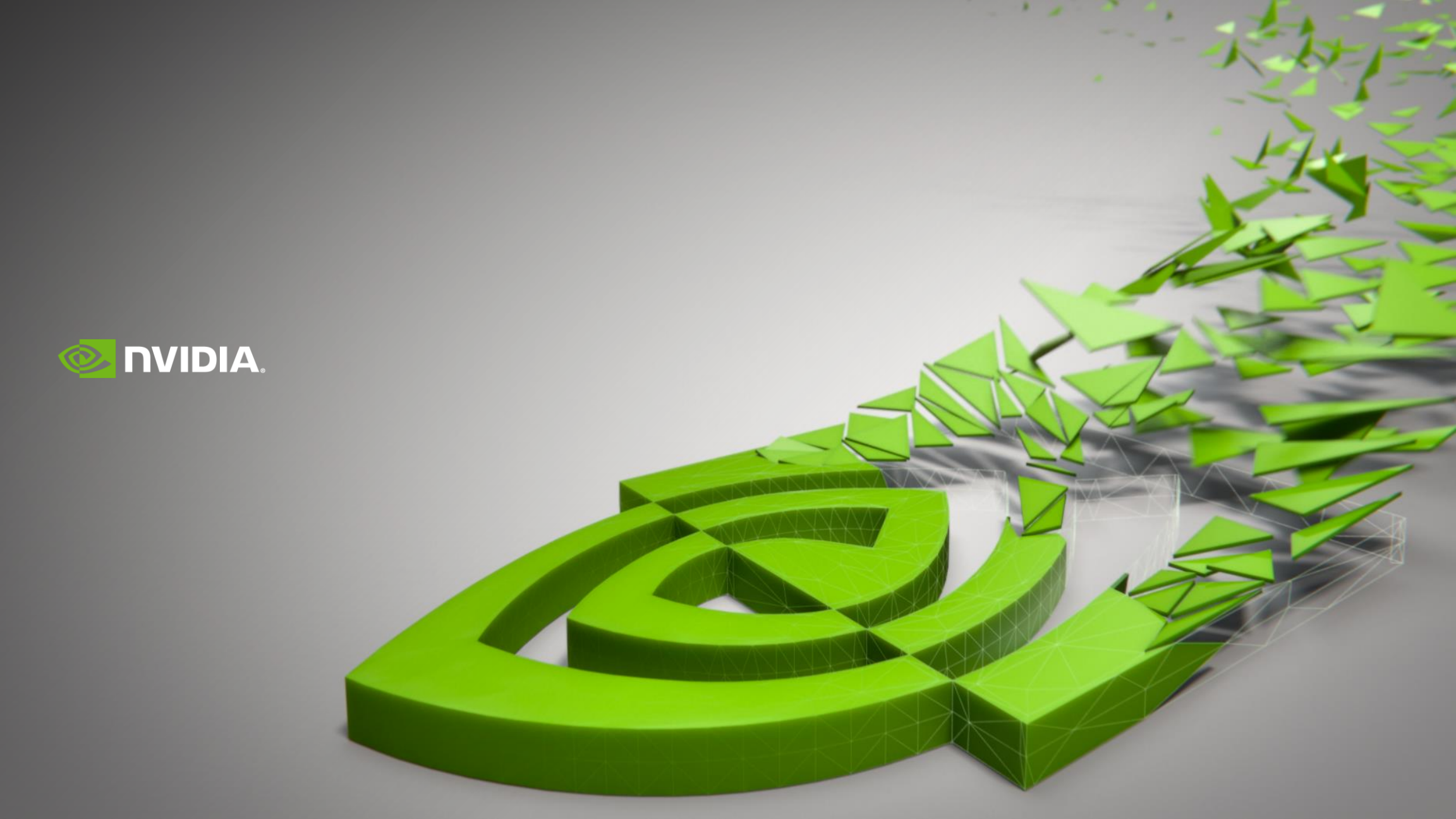


FOR MORE INFORMATION …

CUDA 9 Features Revealed Parallel Forall Blog Post :
https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/

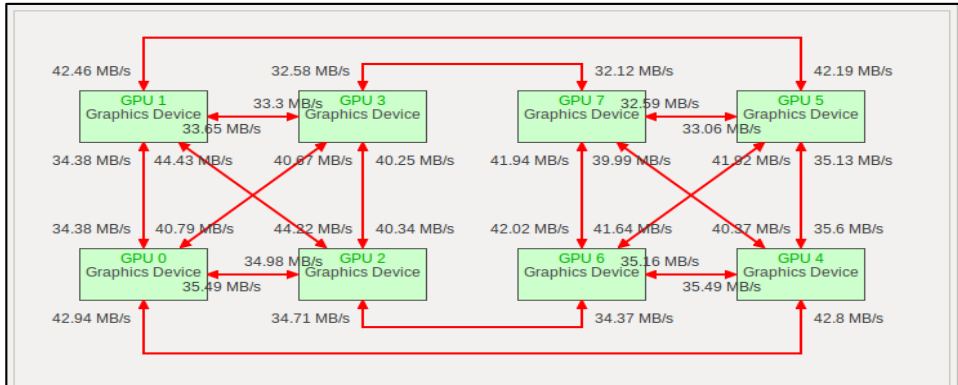
CUDA Documentation: http://docs.nvidia.com/cuda/

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


NVIDIA

BACKUP SLIDES

DGX-1V NVLINK TOPOLOGY



Bandwidth usage by NVLink

- 90 - 100 %
- 80 - 90 %
- 70 - 80 %
- 60 - 70 %
- 50 - 60 %
- 40 - 50 %
- 30 - 40 %
- 20 - 30 %
- 10 - 20 %
- 0 - 10 %

Logical NVLink Properties

Logical NVLink	Peak Bandwidth	Physical NVLinks	Peer Access	System Access	Peer Atomic	System Atomic	Utilization %	Idle time %
GPU0<-->GPU1	50 GB/s	1	Yes	No	Yes	No	0	51
GPU0<-->GPU2	50 GB/s	1	Yes	No	Yes	No	0	50
GPU0<-->GPU3	100 GB/s	2	Yes	No	Yes	No	0	47
GPU0<-->GPU4	100 GB/s	2	Yes	No	Yes	No	0	50
GPU1<-->GPU2	100 GB/s	2	Yes	No	Yes	No	0	51
GPU1<-->GPU3	50 GB/s	1	Yes	No	Yes	No	0	47
GPU1<-->GPU5	100 GB/s	2	Yes	No	Yes	No	0	48
GPU2<-->GPU3	100 GB/s	2	Yes	No	Yes	No	0	48
GPU2<-->GPU6	50 GB/s	1	Yes	No	Yes	No	0	48
GPU3<-->GPU7	50 GB/s	1	Yes	No	Yes	No	0	46
GPU4<-->GPU5	50 GB/s	1	Yes	No	Yes	No	0	50
GPU4<-->GPU6	50 GB/s	1	Yes	No	Yes	No	0	50
GPU4<-->GPU7	100 GB/s	2	Yes	No	Yes	No	0	48
GPU5<-->GPU6	100 GB/s	2	Yes	No	Yes	No	0	49
GPU5<-->GPU7	50 GB/s	1	Yes	No	Yes	No	0	46
GPU6<-->GPU7	100 GB/s	2	Yes	No	Yes	No	0	49

Logical NVLink Throughput

Logical NVLink	Avg Throughput	Max Throughput	Min Throughput
GPU0-->GPU1	34.381 MB/s	7.719 GB/s	2.028 kB/s
GPU0<--GPU1	34.376 MB/s	7.73 GB/s	2.088 kB/s
GPU0-->GPU2	34.98 MB/s	10.472 GB/s	1.433 kB/s
GPU0<--GPU2	35.494 MB/s	10.781 GB/s	1.911 kB/s
GPU0-->GPU3	40.665 MB/s	11.511 GB/s	4.146 kB/s
GPU0<--GPU3	40.79 MB/s	11.402 GB/s	4.176 kB/s
GPU0-->GPU4	42.801 MB/s	6.556 GB/s	1.982 kB/s
GPU0<--GPU4	42.938 MB/s	6.413 GB/s	2.644 kB/s
GPU1-->GPU2	44.221 MB/s	11.762 GB/s	4.216 kB/s
GPU1<--GPU2	44.429 MB/s	11.676 GB/s	4.088 kB/s
GPU1-->GPU3	33.301 MB/s	10.662 GB/s	1.949 kB/s
GPU1<--GPU3	33.651 MB/s	10.871 GB/s	1.462 kB/s
GPU1-->GPU5	42.189 MB/s	12.298 GB/s	2.004 kB/s
GPU1<--GPU5	42.455 MB/s	12.346 GB/s	2.672 kB/s
GPU2-->GPU3	40.25 MB/s	9.675 GB/s	3.422 kB/s

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

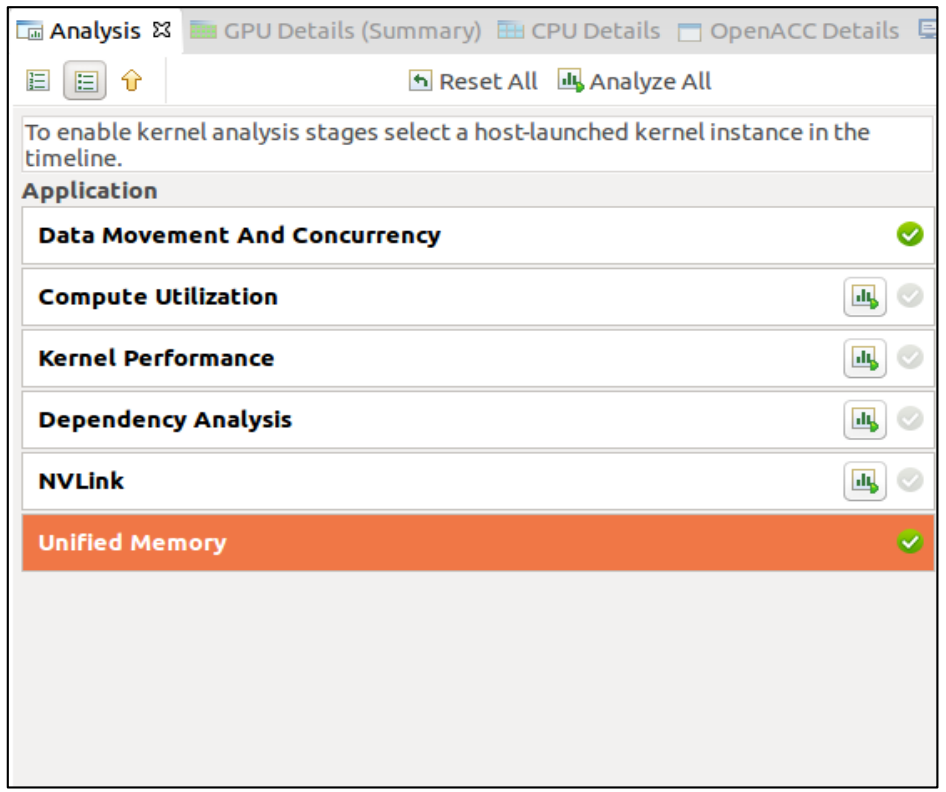
Effective in optimizing large kernels, pinpoints performance bottlenecks at specific lines in source code or assembly instructions

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

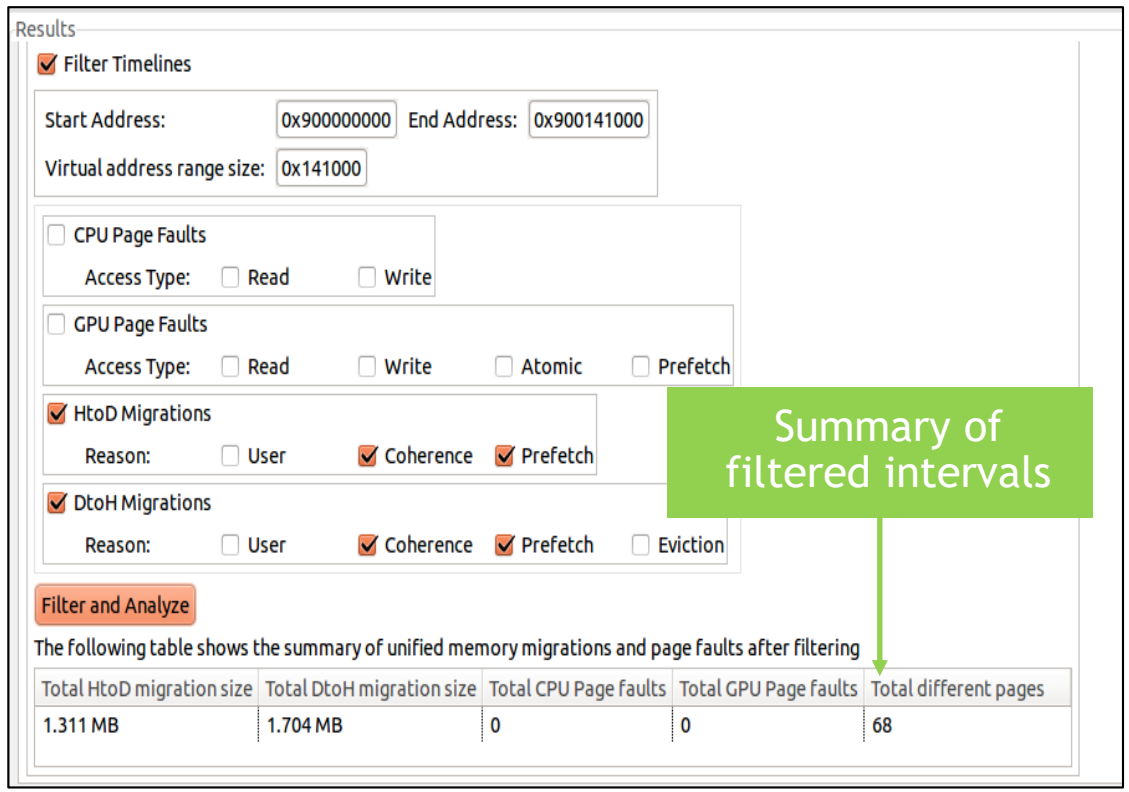
No overheads in kernel runtime, CPU overheads to parse the records

FILTER AND ANALYZE

1 Select unified memory in the unguided analysis section

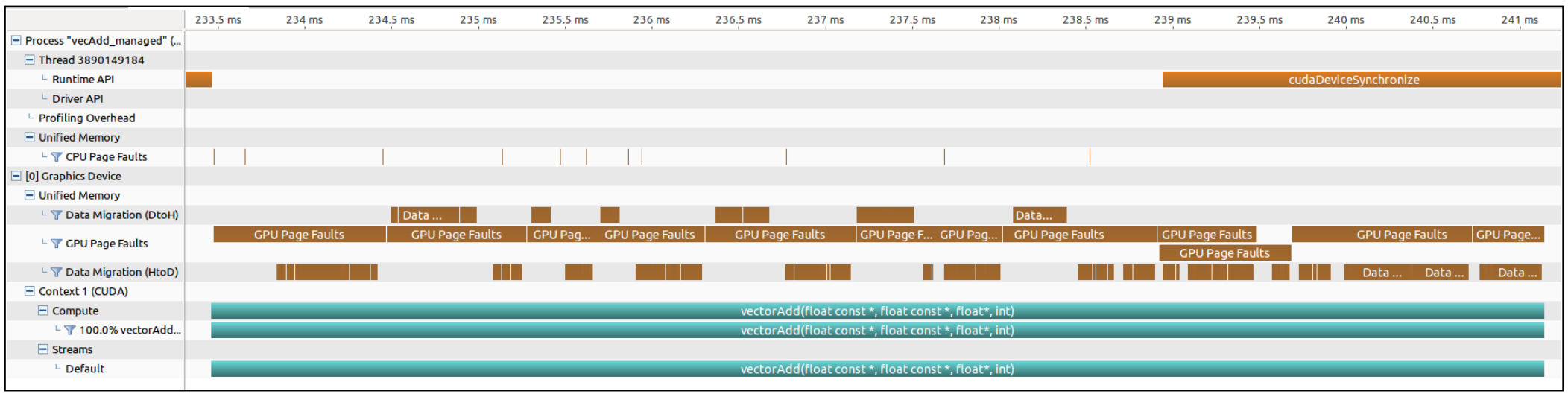


2 Select required events and click on ‘Filter and Analyze’



FILTER AND ANALYZE

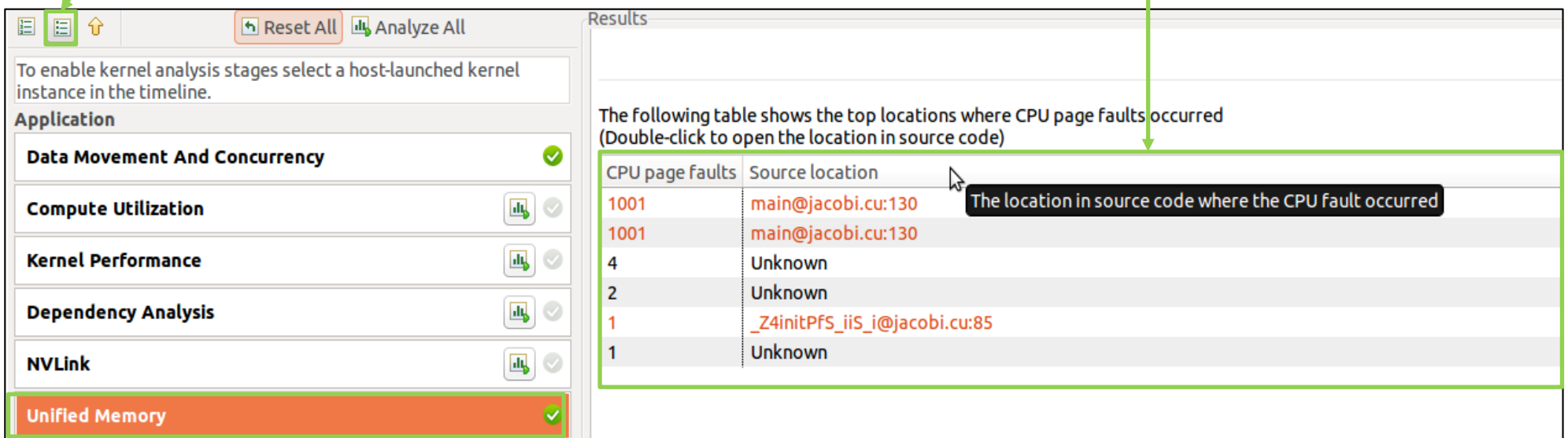
Unfiltered



CPU PAGE FAULT SOURCE CORRELATION

Unguided Analysis

Summary of all CPU page faults



Option to collect Unified Memory information

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.