OPTIMIZING APPLICATION PERFORMANCE WITH CUDA PROFILING TOOLS

Mayank Jain, 11 May 2017

AGENDA

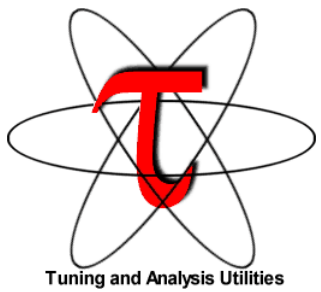
- CUDA Profiling Tools
- Unified Memory Profiling
- NVLink Profiling
- PC Sampling
- MPI Profiling
- Multi-hop Remote Profiling
- Volta Support
- Other Improvements

CUDA PROFILING TOOLS

- NVIDIA® Visual Profiler
- nvprof *
- NVIDIA® Nsight™ Visual Studio Edition

* Android CUDA APK profiling not supported (yet)

3RD PARTY PROFILING TOOLS



TAU Performance System ®

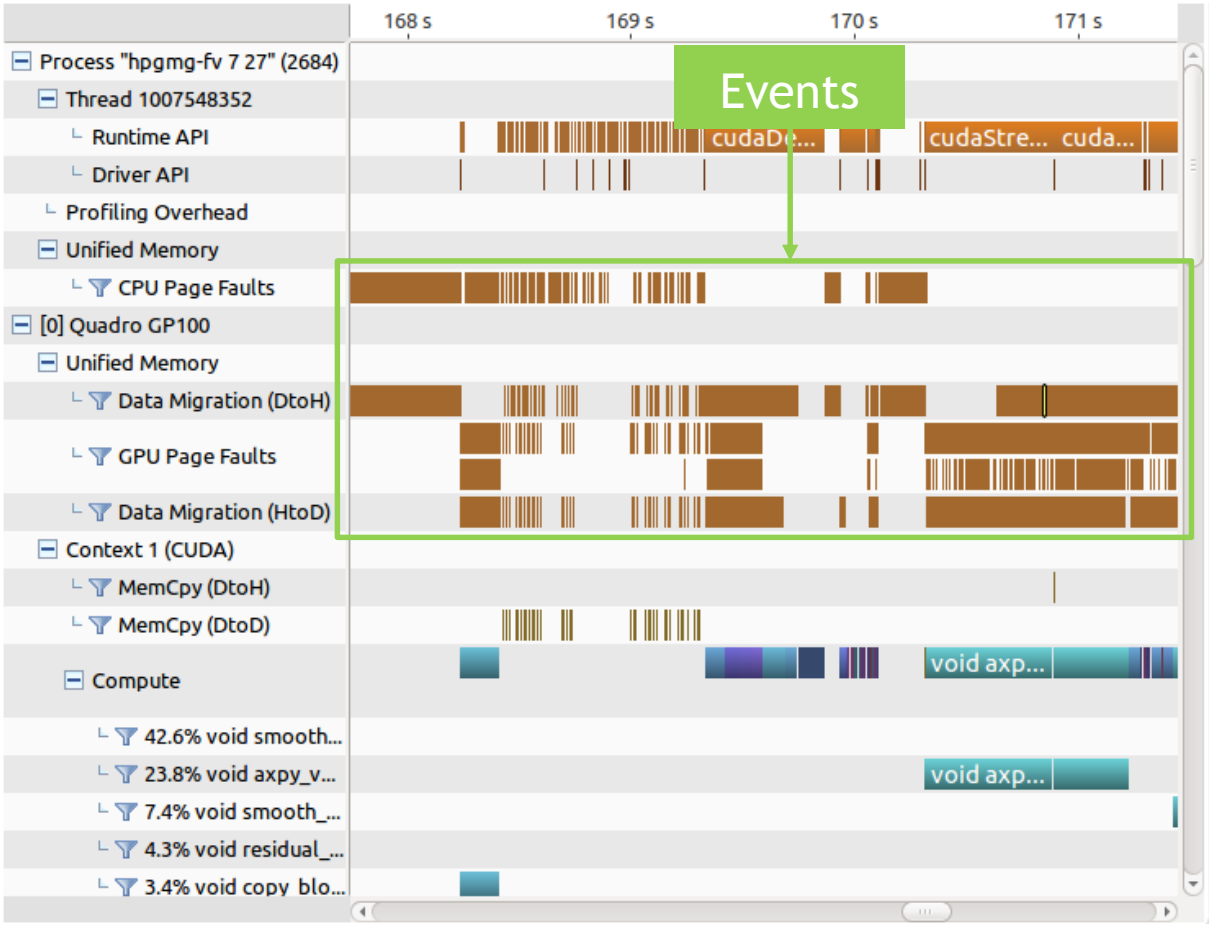
VampirTrace

PAPI CUDA Component

HPC Toolkit

UNIFIED MEMORY PROFILING

UNIFIED MEMORY EVENTS



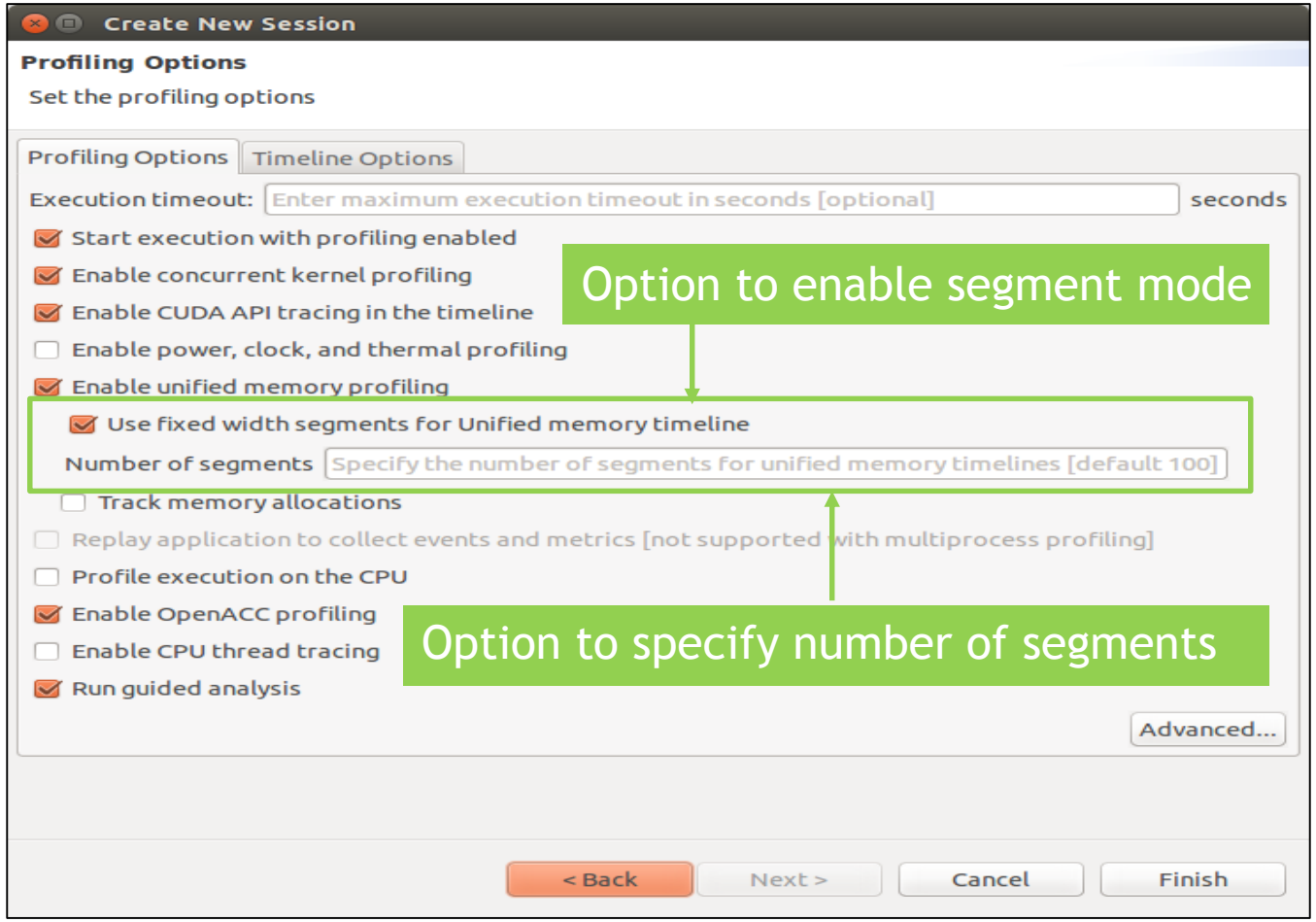
Data Migration (DtoH)

Start	170.845 s (170,844,511,496 ns)
End	170.845 s (170,844,671,976 ns)
Duration	160.48 µs
Size	2.097 MB
Migration Cause	Eviction
Throughput	13.068 GB/s
Virtual Address	0x4b2a00000
Memory operations	Browse for executable...
Process	2684

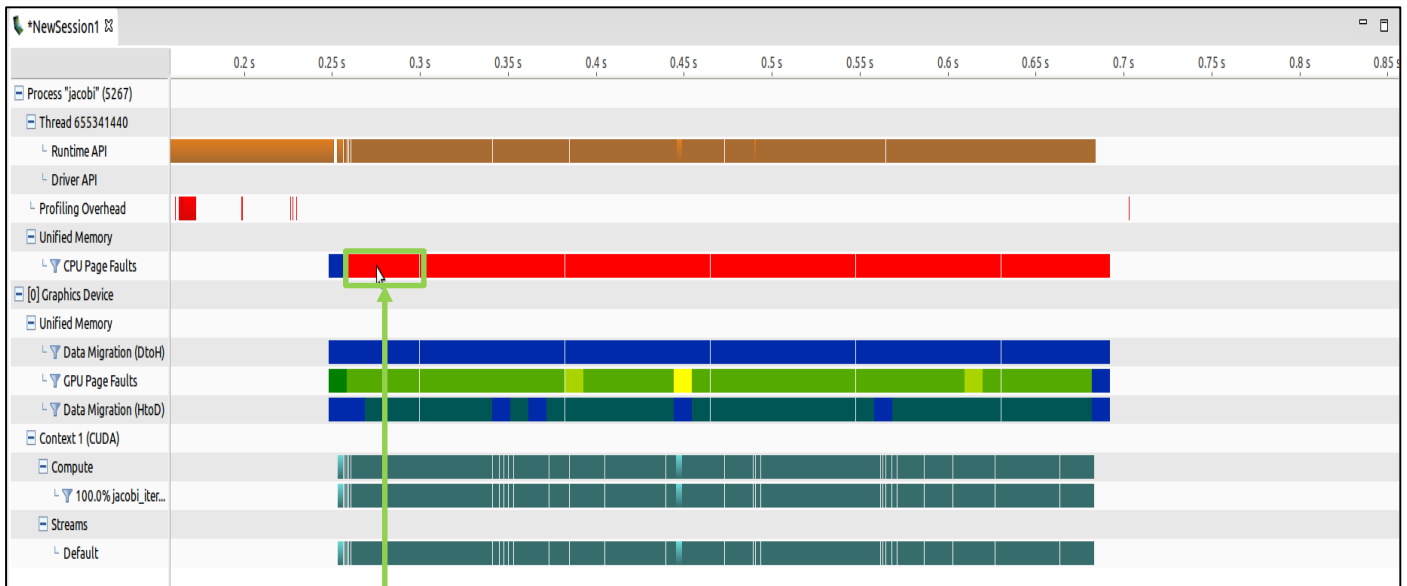
GPU Page Fault groups

Start	170.925 s (170,924,798,600 ns)
End	170.925 s (170,924,919,048 ns)
Duration	120.448 µs
Memory Acccess Type	Write
Virtual Address	0x34b680000
GPU Page Faults	2
Memory operations	Browse for executable...
Process	2684

SEGMENT MODE TIMELINE OPTIONS

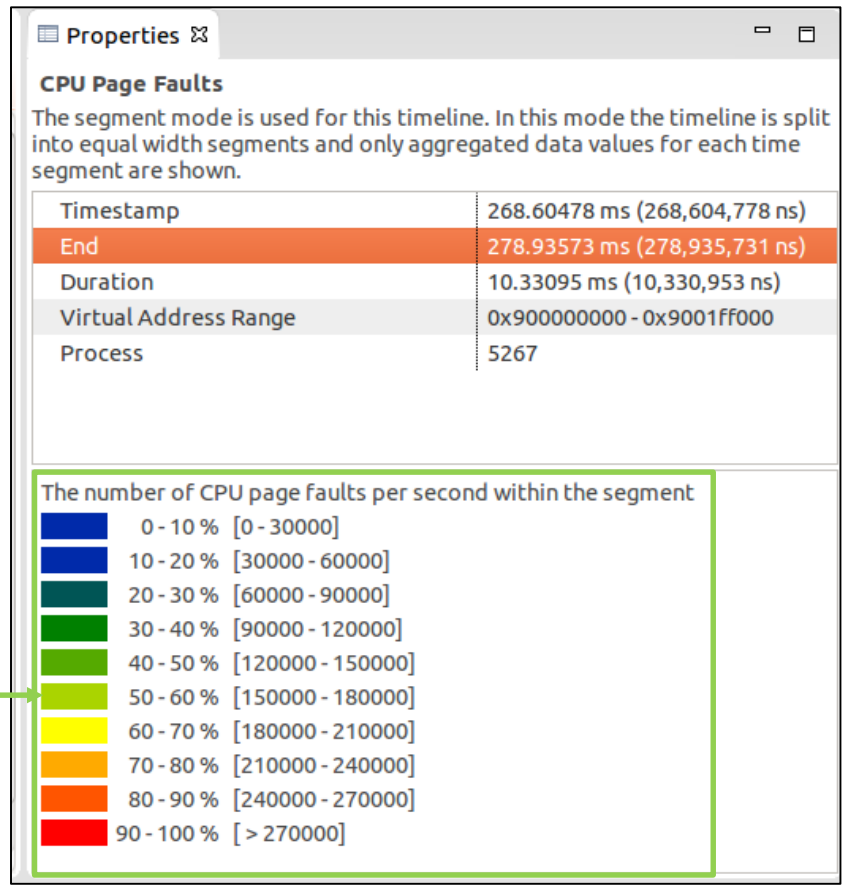


SEGMENT MODE TIMELINE



Segment mode interval

Heat map for CPU page faults



SWITCH TO NON-SEGMENT VIEW

Uncheck

Select settings view

Analysis | GPU Details (Summary) | CPU Details | OpenACC Details | Console | Settings

Session NewSession1

Executable

Timeline Options

Analysis

Use fixed width segments for Unified memory timeline

Number of segments 100

Load data for time range

Start time 225 ms to End time 700 ms

Enable timelines in session view

- All
  - Process
    - Thread
    - Profiling Overhead
    - Markers and Ranges
    - OpenACC
    - Pthread
    - CPU Page Faults
  - Device
    - Unified Memory
    - Context

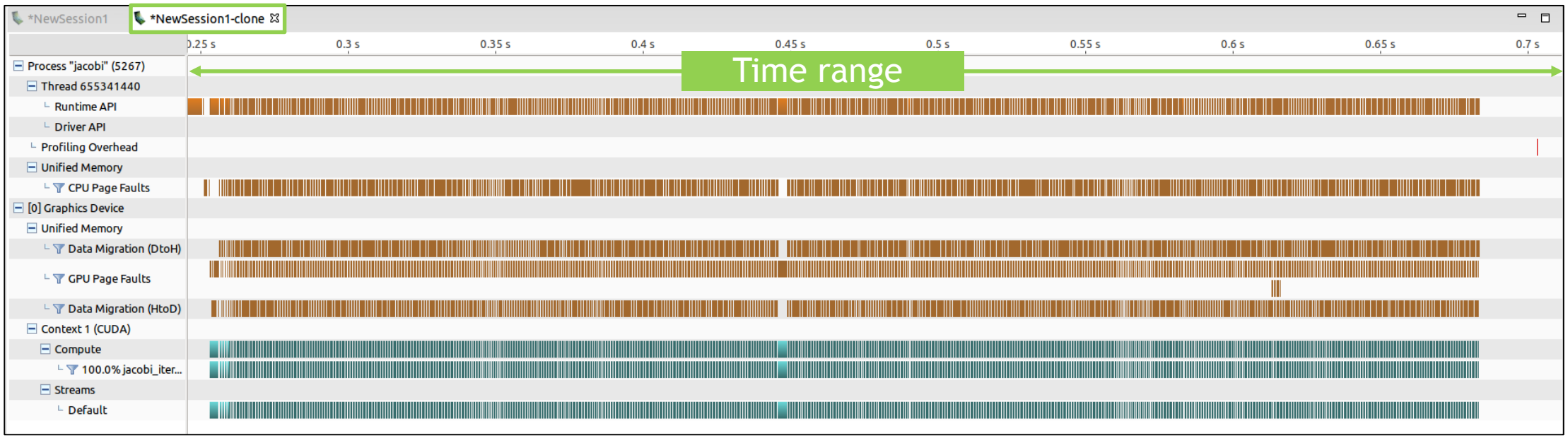
Open in new session

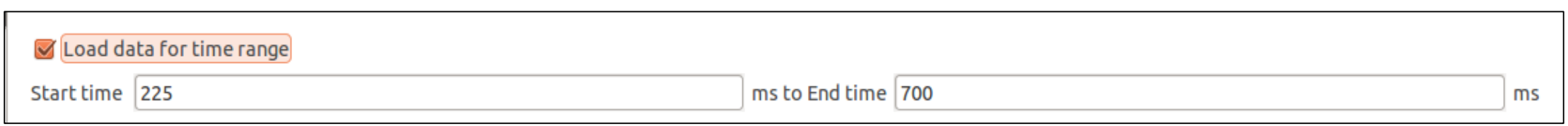
Apply

Select this tab

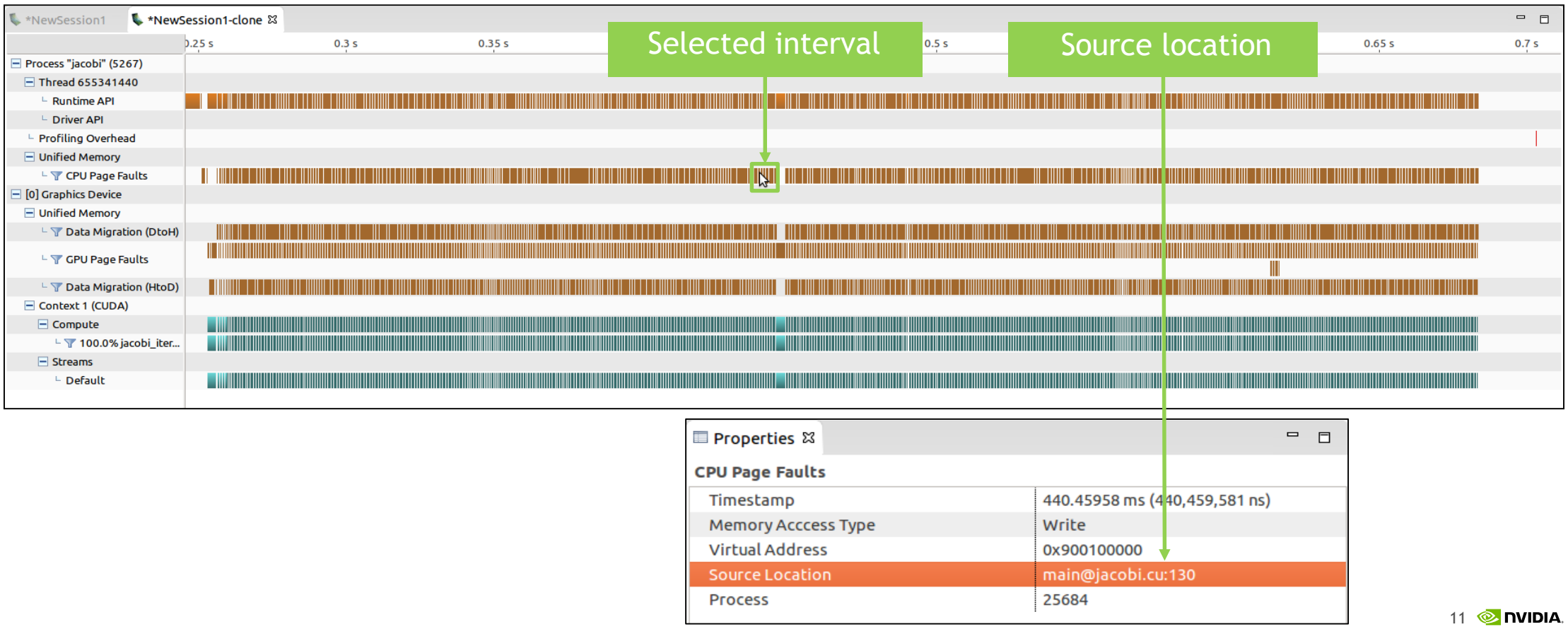
Load data within a specific time range

NON-SEGMENTED MODE TIMELINE





CPU PAGE FAULT SOURCE CORRELATION



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
```

CPU PAGE FAULT SOURCE CORRELATION

Unguided Analysis

Summary of all CPU page faults

Reset All | Analyze All

To enable kernel analysis stages select a host-launched kernel instance in the timeline.

Application

- **Data Movement And Concurrency**
- **Compute Utilization**
- **Kernel Performance**
- **Dependency Analysis**
- **NVLink**
- **Unified Memory**

Results

The following table shows the top locations where CPU page faults occurred
(Double-click to open the location in source code)

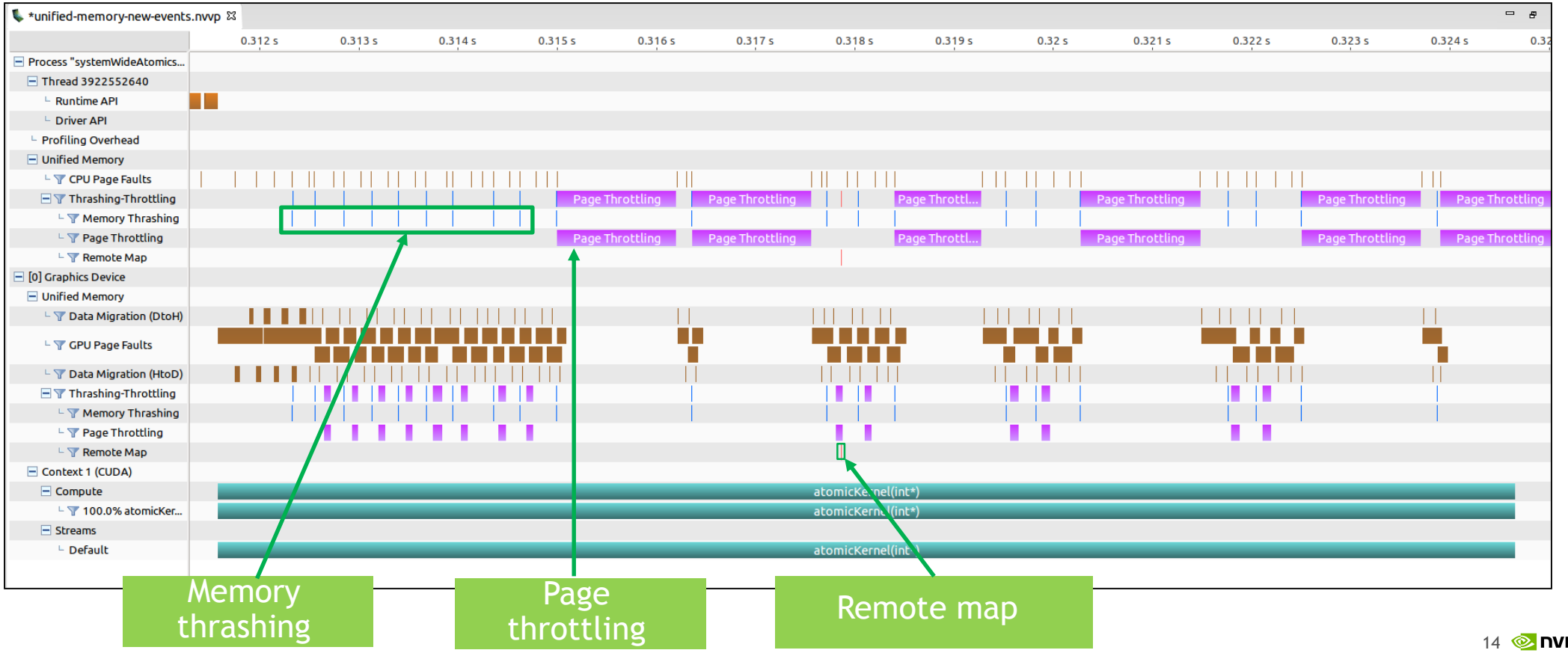
CPU page faults	Source location
1001	main@jacobi.cu:130
1001	main@jacobi.cu:130
4	Unknown
2	Unknown
1	_Z4initPfS_iiS_i@jacobi.cu:85
1	Unknown

The location in source code where the CPU fault occurred

Option to collect Unified Memory information

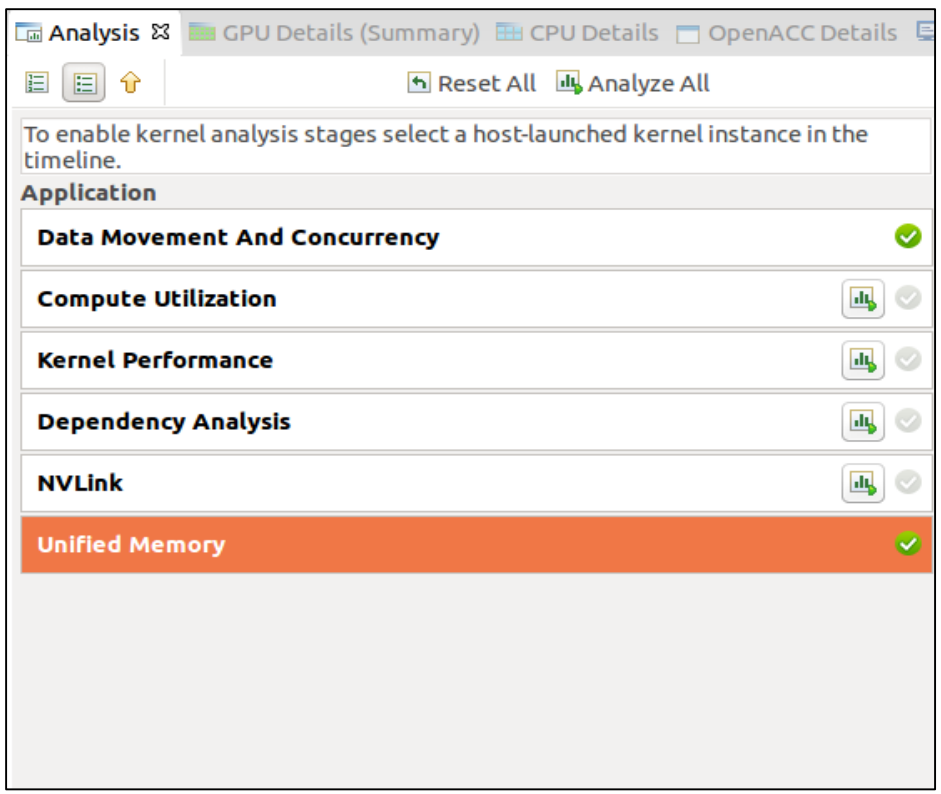
NEW UNIFIED MEMORY EVENTS

Page throttling, Memory thrashing, Remote map

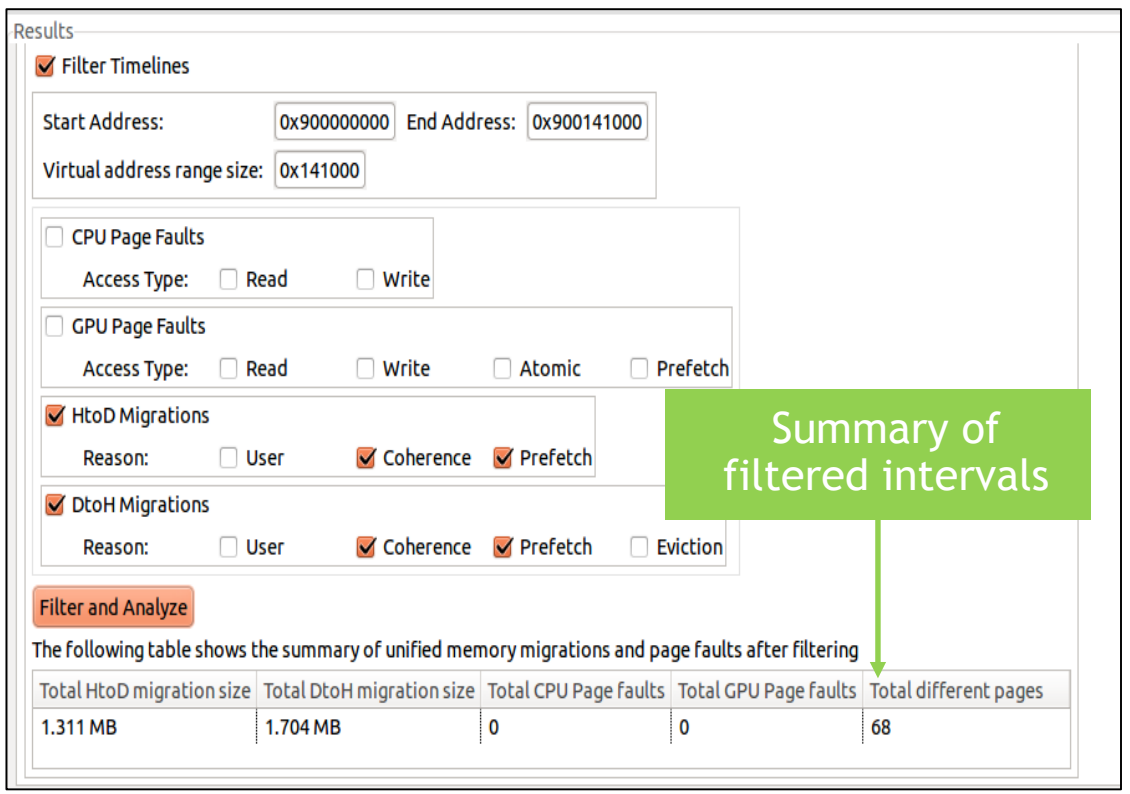


FILTER AND ANALYZE

1. Select unified memory in the unguided analysis section

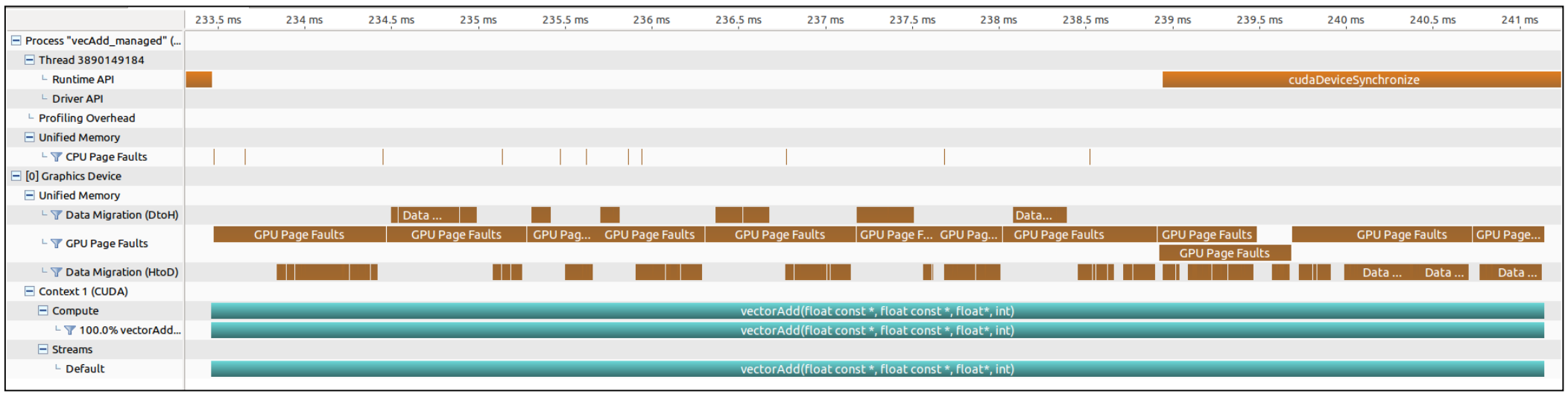


2. Select required events and click on 'Filter and Analyze'



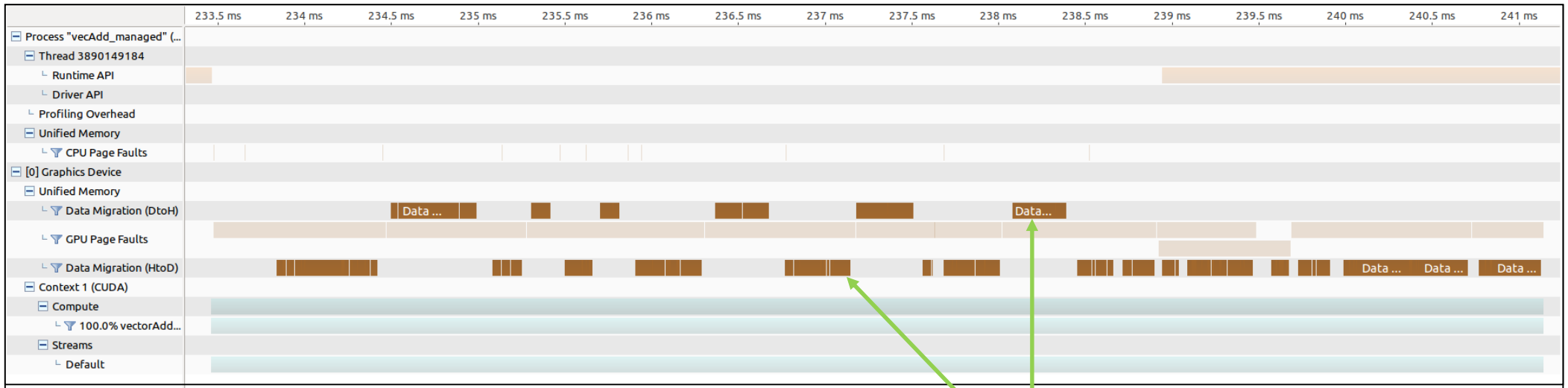
FILTER AND ANALYZE

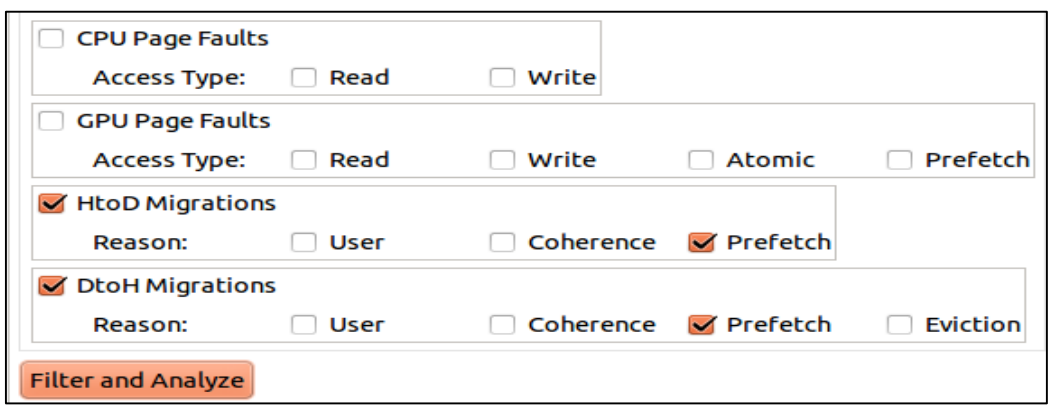
Unfiltered



FILTER AND ANALYZE

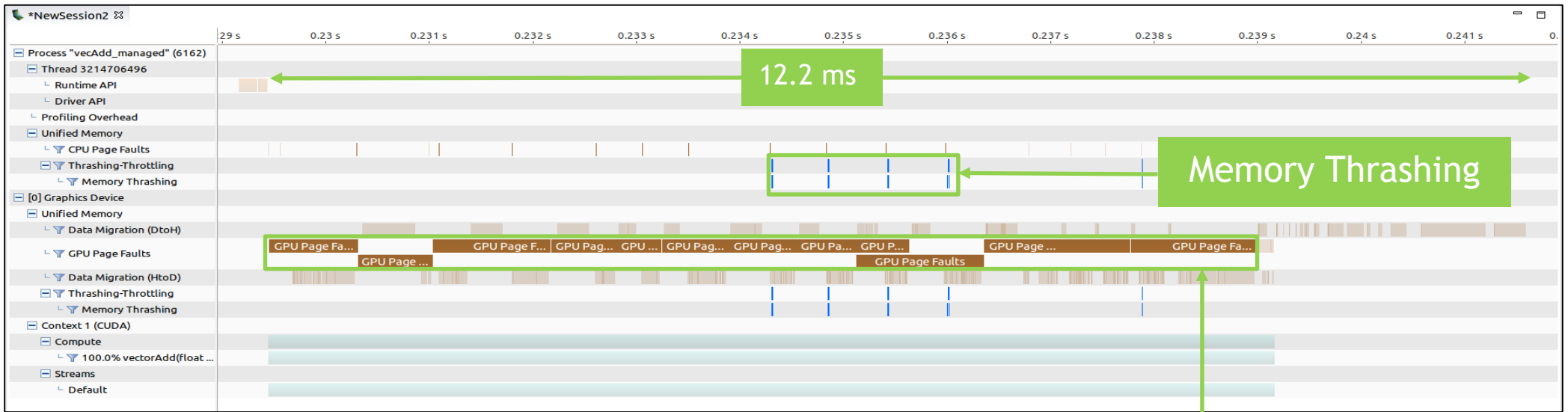
Filtered



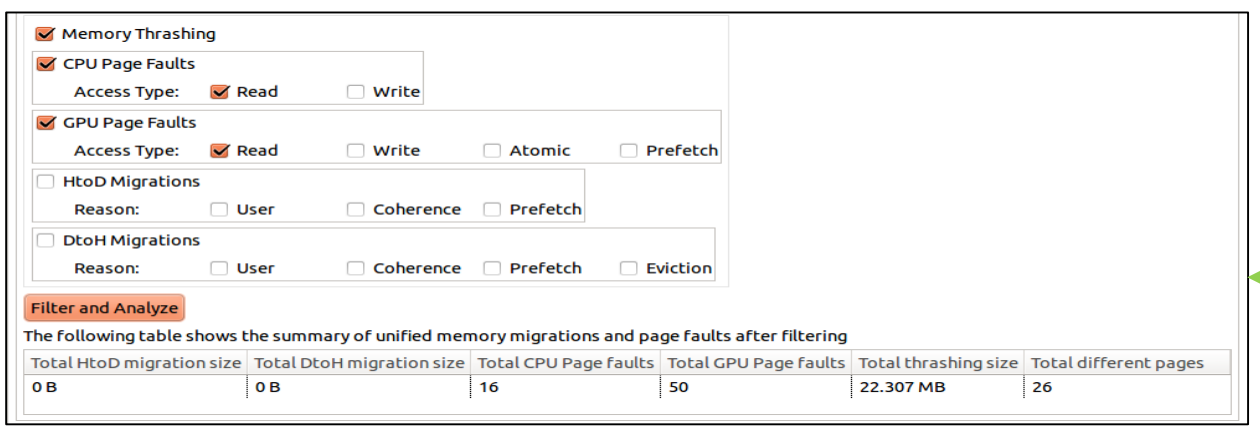


Filtered intervals

UNOPTIMIZED APPLICATION



Read access page faults



Total HtoD migration size	Total DtoH migration size	Total CPU Page faults	Total GPU Page faults	Total thrashing size	Total different pages
0 B	0 B	16	50	22.307 MB	26

Analyze read access page faults and thrashing

OPTIMIZATION

OLD

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

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

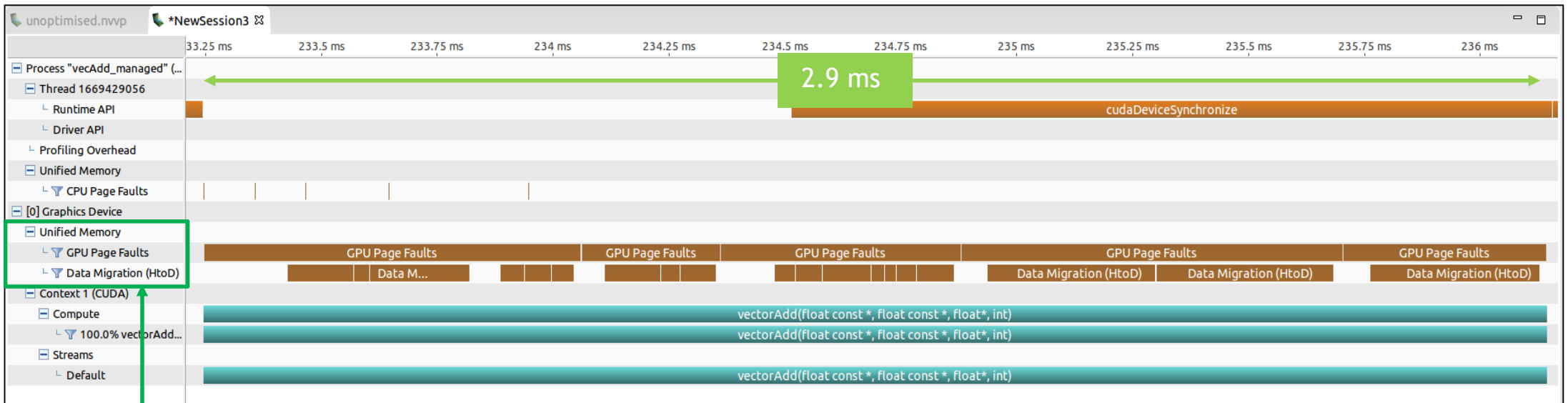
NEW

```
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)

NVLINK PROFILING

NVLINK VISUALIZATION

Visual Profiler

Unguided Analysis

Static properties

Runtime values

Analysis | GPU Details (Summary) | CPU Details | OpenACC Details | Console | Settings

Reset All | Analyze All

To enable kernel analysis stages select a host-launched kernel instance in the timeline.

Application

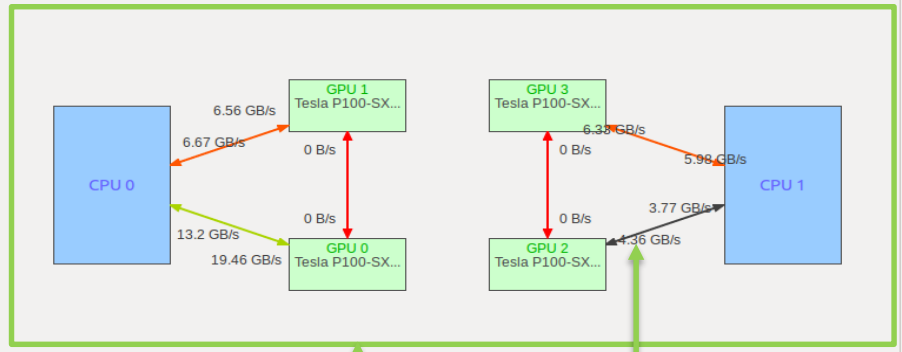
- Data Movement And Concurrency
- Compute Utilization
- Kernel Performance
- Dependency Analysis
- NVLink
- Unified Memory

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.



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<-->CPU0	80 GB/s	2	No	Yes	No	No	40	99
GPU0<-->GPU1	80 GB/s	2	Yes	No	Yes	No	n/a	100
GPU1<-->CPU0	80 GB/s	2	No	Yes	No	No	16	99
GPU2<-->CPU1	80 GB/s	2	No	Yes	No	No	10	99
GPU2<-->GPU3	80 GB/s	2	Yes	No	Yes	No	n/a	100
GPU3<-->CPU1	80 GB/s	2	No	Yes	No	No	15	99

Logical NVLink Throughput

Logical NVLink	Avg Throughput	Max Throughput	Min Throughput
GPU0-->CPU0	13.203 GB/s	13.203 GB/s	28.343 kB/s
GPU0<--CPU0	19.457 GB/s	20.509 GB/s	10.268 kB/s
GPU0-->GPU1	0 B/s	0 B/s	0 B/s
GPU0<--GPU1	0 B/s	0 B/s	0 B/s
GPU1-->CPU0	6.667 GB/s	13.223 GB/s	24.576 kB/s
GPU1<--CPU0	6.561 GB/s	20.507 GB/s	8.777 kB/s
GPU2-->CPU1	3.77 GB/s	14.806 GB/s	28.337 kB/s
GPU2<--CPU1	4.363 GB/s	16.49 GB/s	6.31 kB/s
GPU2-->GPU3	0 B/s	0 B/s	0 B/s
GPU2<--GPU3	0 B/s	0 B/s	0 B/s
GPU3-->CPU1	5.984 GB/s	14.823 GB/s	24.331 kB/s
GPU3<--CPU1	6.326 GB/s	16.495 GB/s	12.25 kB/s

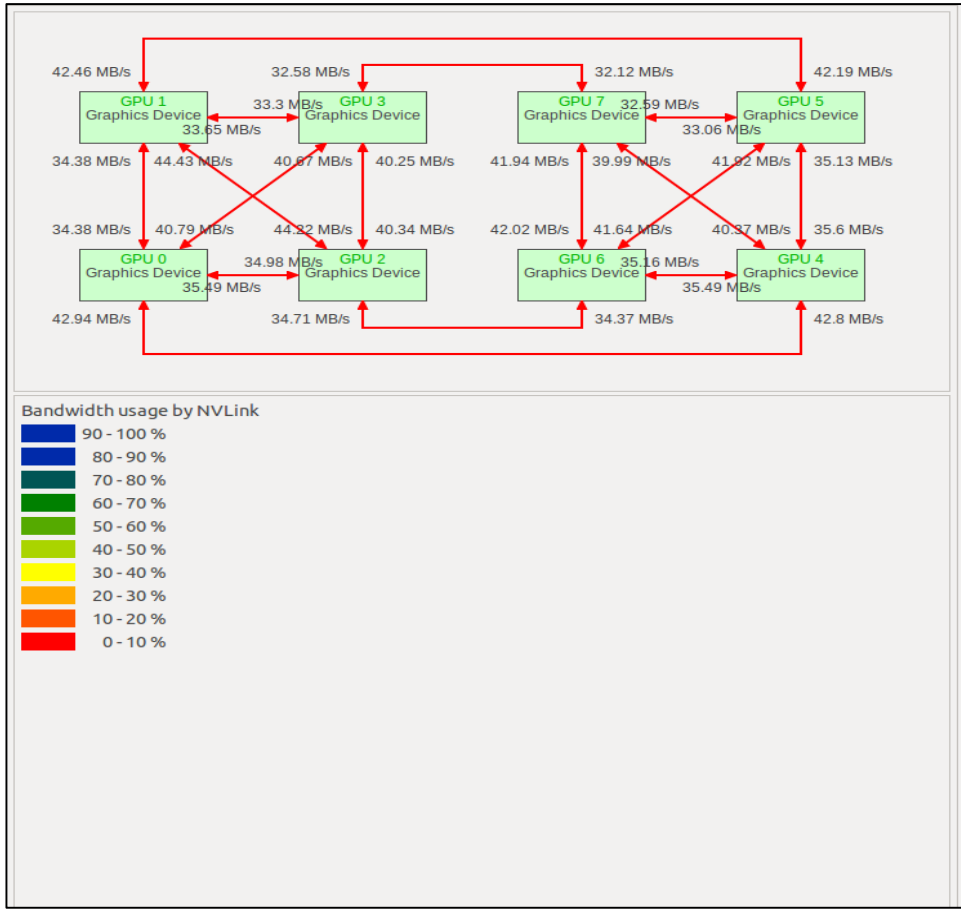
Option to collect NVLink information

Topology

Selected NVLink

Color codes for NVLink

DGX-1V NVLINK TOPOLOGY



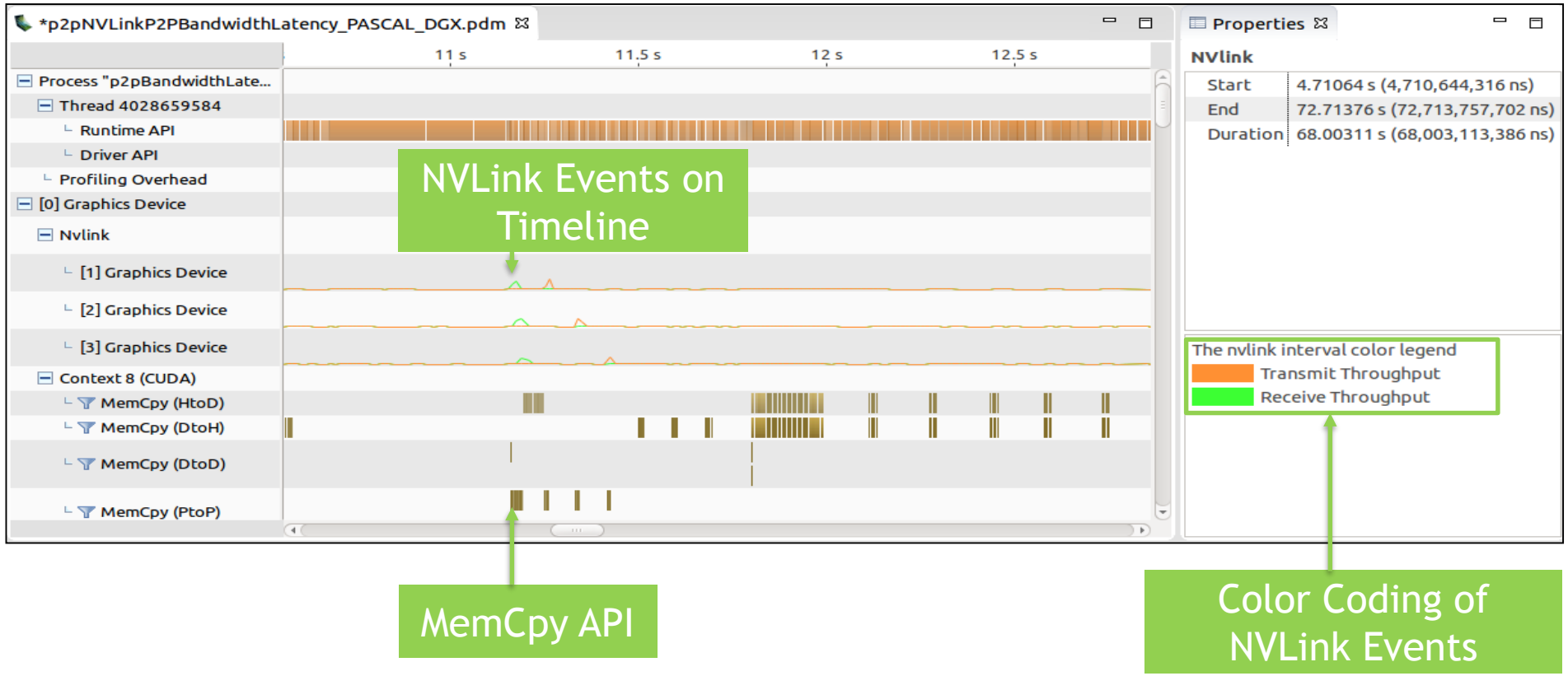
Logical NVLink Properties

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

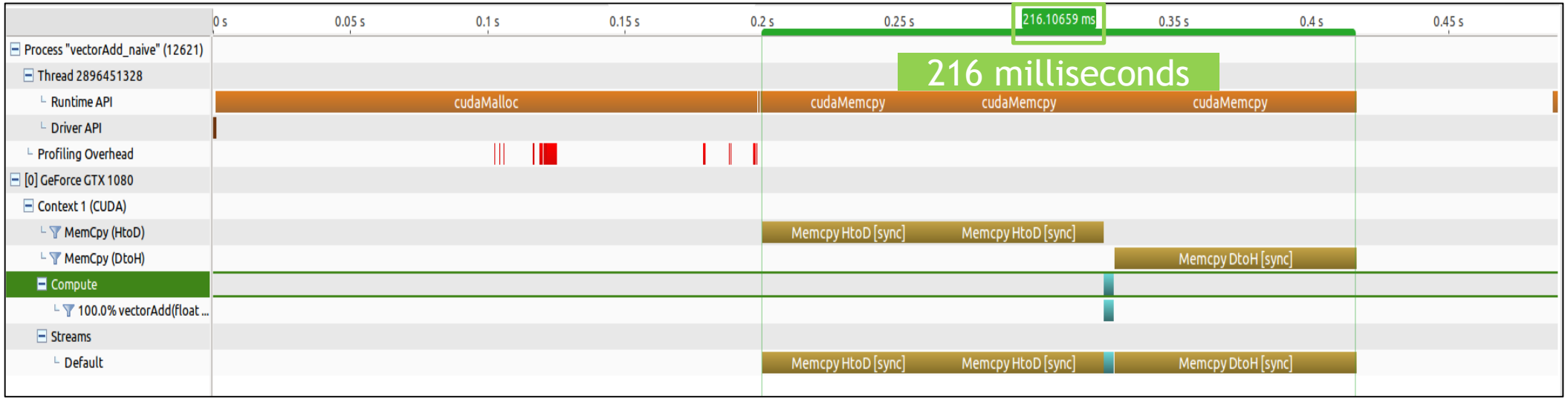
Logical NVLink	Avg Throughput	Max Throughput	Min Throughput
GPU0-->GPU1	34.381 MB/s	7.719 GB/s	2.028 kB/s
GPU0<--GPU1	34.376 MB/s	7.73 GB/s	2.088 kB/s
GPU0-->GPU2	34.98 MB/s	10.472 GB/s	1.433 kB/s
GPU0<--GPU2	35.494 MB/s	10.781 GB/s	1.911 kB/s
GPU0-->GPU3	40.665 MB/s	11.511 GB/s	4.146 kB/s
GPU0<--GPU3	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

NVLINK EVENTS ON TIMELINE



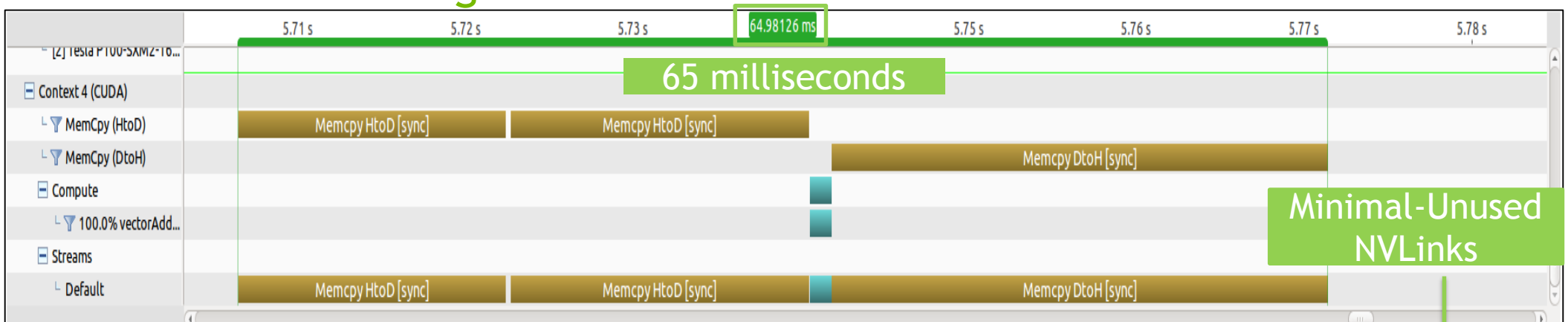
NVLINK ANALYSIS

Stage I: Data Movement Over PCIe

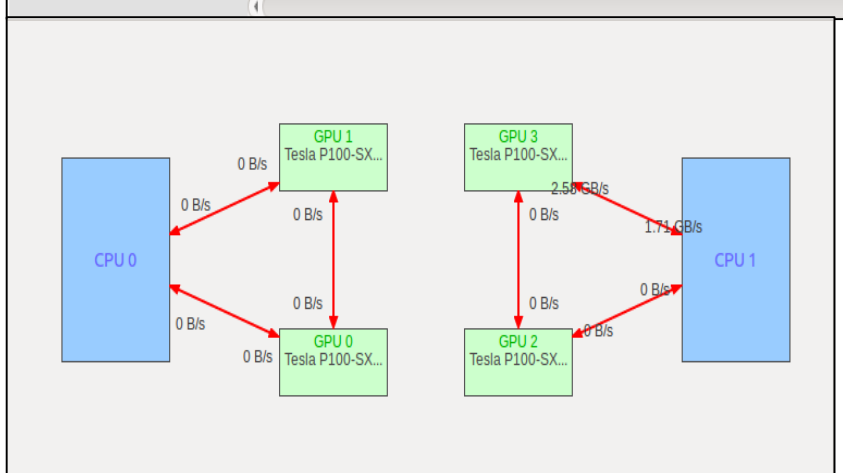


NVLINK ANALYSIS

Stage II: Data Movement Over NVLink



Minimal-Unused NVLinks



Speedup 4x (216 vs 65)

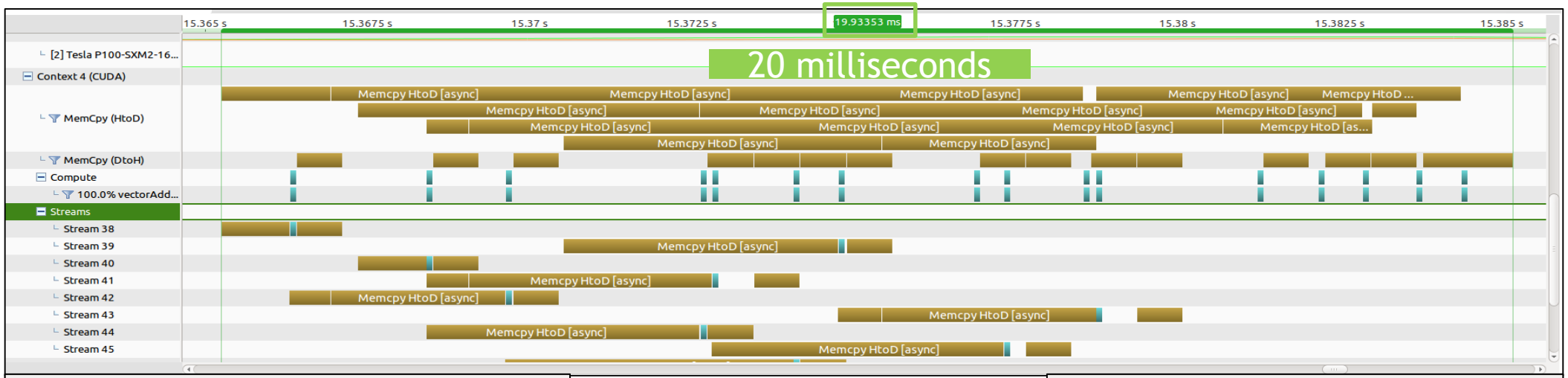
Under-utilized NVLink

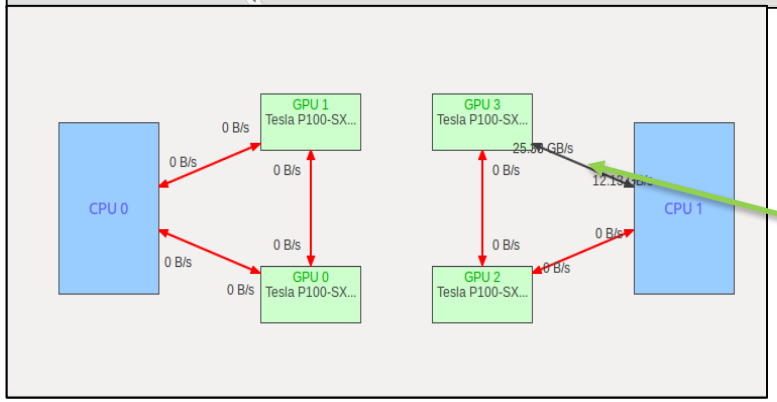
Logical NVLink Throughput

Logical NVLink	Avg Throughput	Max Throughput	Min Throughput
GPU0-->CPU0	0 B/s	816.309 kB/s	64.756 kB/s
GPU0<--CPU0	0 B/s	372.575 kB/s	29.14 kB/s
GPU0-->GPU1	0 B/s	0 B/s	0 B/s
GPU0<--GPU1	0 B/s	0 B/s	0 B/s
GPU1-->CPU0	0 B/s	777.593 kB/s	79.474 kB/s
GPU1<--CPU0	0 B/s	351.521 kB/s	31.789 kB/s
GPU2-->CPU1	0 B/s	829.071 kB/s	67.267 kB/s
GPU2<--CPU1	0 B/s	362.956 kB/s	28.588 kB/s
GPU2-->GPU3	0 B/s	0 B/s	0 B/s
GPU2<--GPU3	0 B/s	0 B/s	0 B/s
GPU3-->CPU1	1.71 GB/s	1.71 GB/s	26.606 kB/s
GPU3<--CPU1	2.577 GB/s	2.577 GB/s	11.402 kB/s

NVLINK ANALYSIS

Stage III: Data Movement Over NVLink with Streams





Speedup 3x (65 vs 20)

Changed color based on selection

Logical NVLink Throughput

Logical NVLink	Avg Throughput	Max Throughput	Min Throughput
GPU0-->CPU0	0 B/s	5.601 MB/s	77.82 kB/s
GPU0<--CPU0	0 B/s	11.604 MB/s	31.207 kB/s
GPU0-->GPU1	0 B/s	0 B/s	0 B/s
GPU0<--GPU1	0 B/s	0 B/s	0 B/s
GPU1-->CPU0	0 B/s	5.146 MB/s	54.503 kB/s
GPU1<--CPU0	0 B/s	11.059 MB/s	20.438 kB/s
GPU2-->CPU1	0 B/s	5.194 MB/s	77.932 kB/s
GPU2<--CPU1	0 B/s	11.201 MB/s	33.849 kB/s
GPU2-->GPU3	0 B/s	0 B/s	0 B/s
GPU2<--GPU3	0 B/s	0 B/s	0 B/s
GPU3-->CPU1	12.131 GB/s	20.671 GB/s	30.775 kB/s
GPU3<--CPU1	25.333 GB/s	28.72 GB/s	10.991 kB/s

INSTRUCTION LEVEL PROFILING (PC SAMPLING)

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

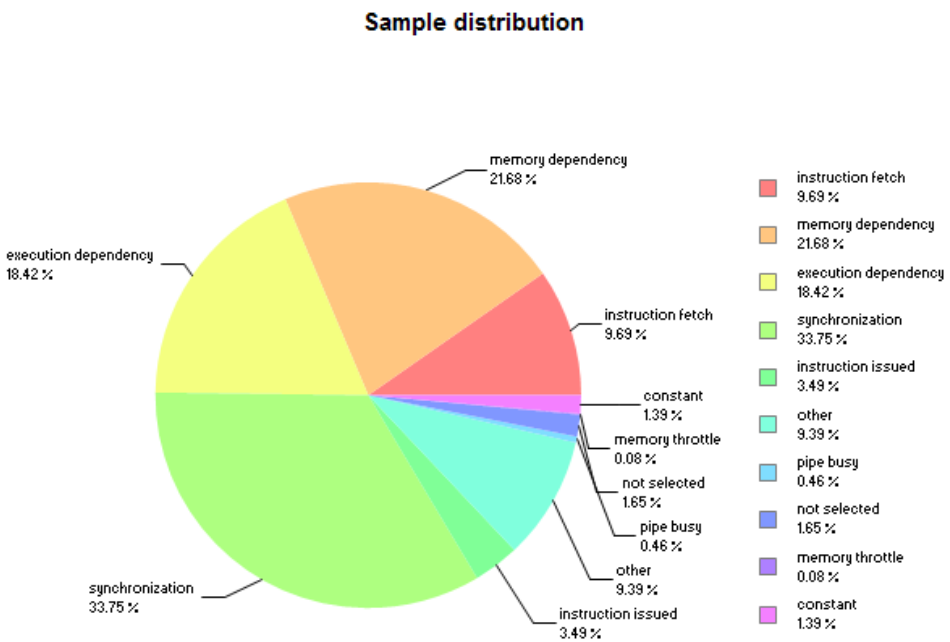
Effective in optimizing large kernels, pinpoints performance bottlenecks at specific lines in source code or assembly instructions

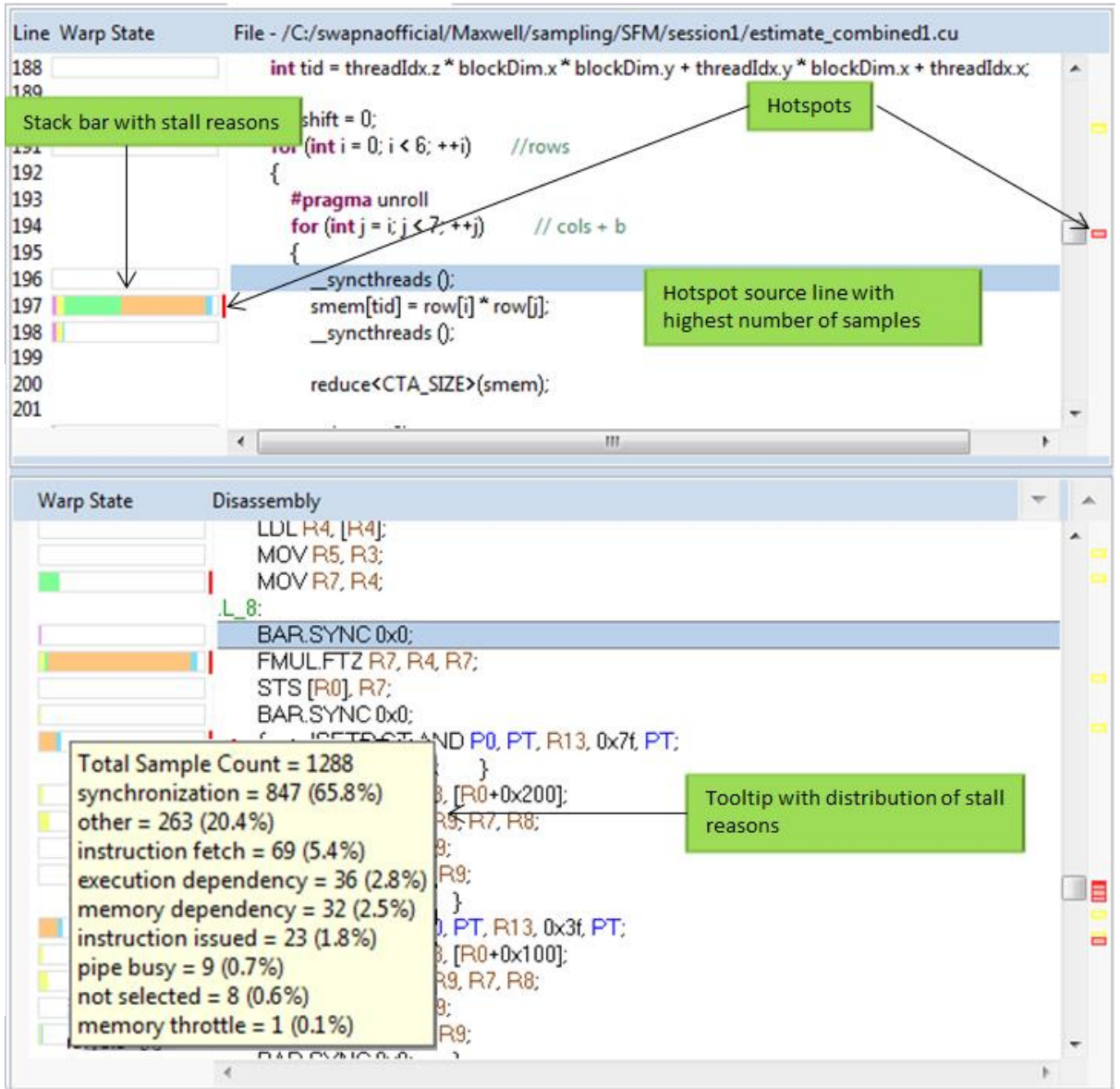
Samples warp states periodically in round robin order over all active warps

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

PC SAMPLING UI

Pie chart for sample distribution for a CUDA function





Source-Assembly view

MPI PROFILING

MPI PROFILING

nvprof

```
$ mpirun -n 4 nvprof --process-name "MPI Rank %q{OMPI_COMM_WORLD_RANK}" --
context-name "MPI Rank %q{OMPI_COMM_WORLD_RANK}" -o
timeline.%q{OMPI_COMM_WORLD_RANK}.pdm ./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.pdm
==21977== Generated result file: timeline.3.pdm
==21983== Generated result file: timeline.1.pdm
==21979== Generated result file: timeline.2.pdm
```

MPI PROFILING

nvprof daemon mode

Shell 1

1
```
$ nvprof --profile-all-processes
-o out.%h.%p.%q{OMPI_COMM_WORLD_RANK}
<nvprof listens in daemon mode>
```

3
```
<profiling data is generated>
```

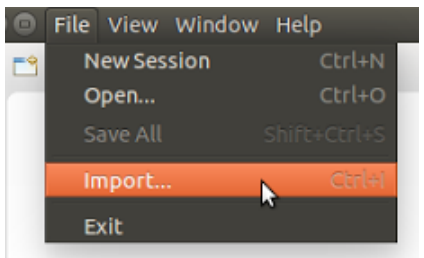
Shell 2

2
```
$ mpirun -n 4 ./simpleMPI
```

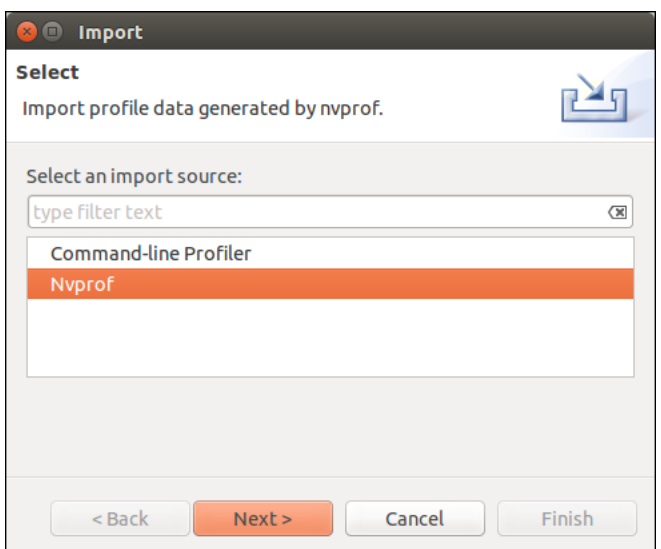
MPI PROFILING

Importing into the Visual Profiler

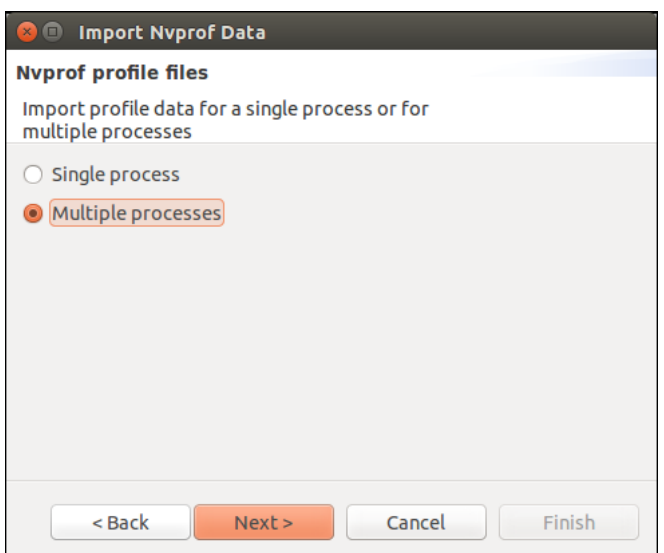
1



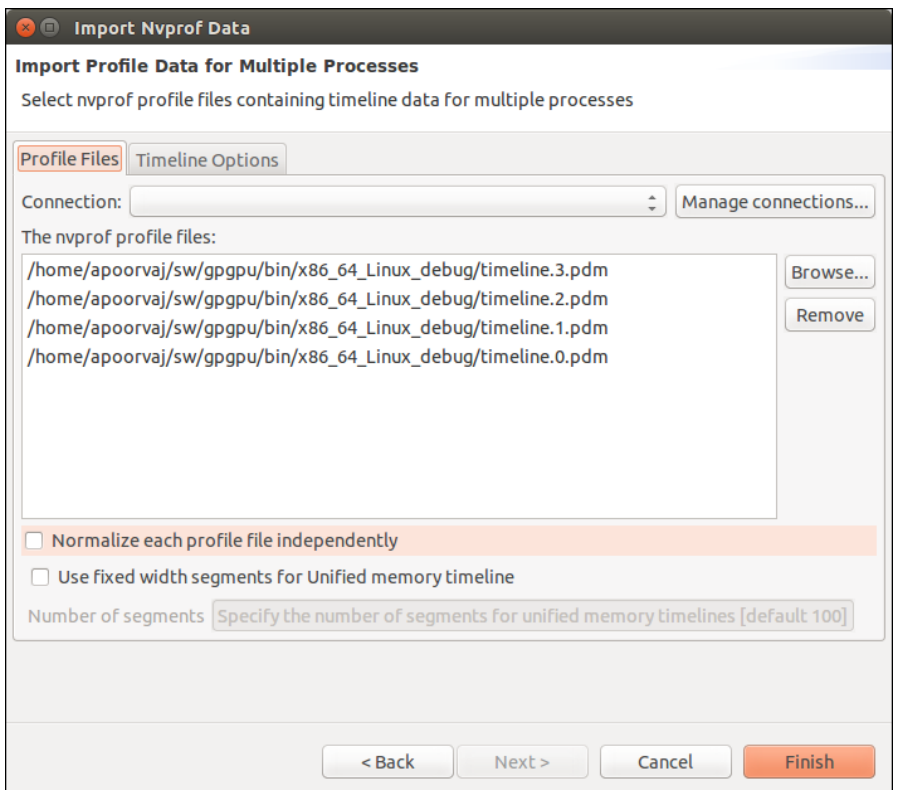
2



3



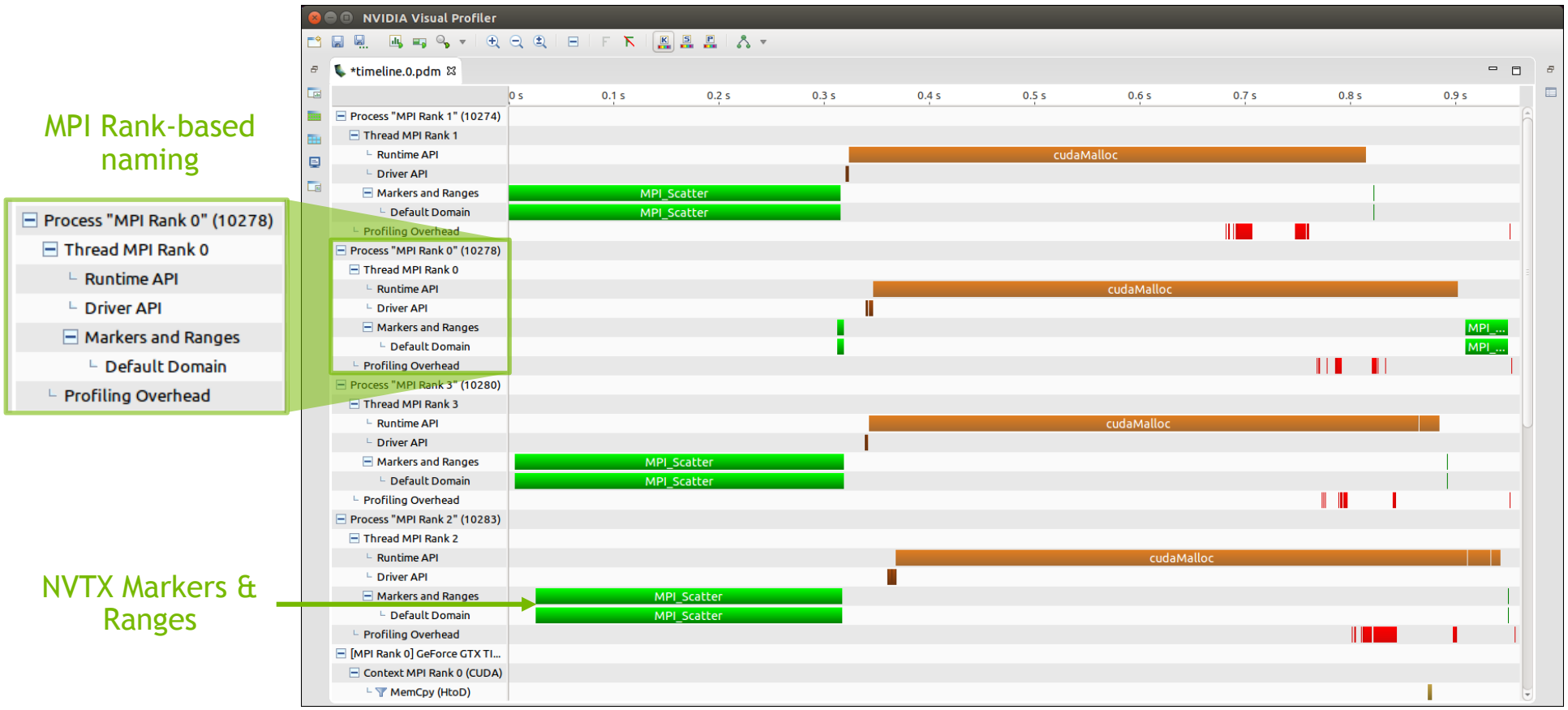
4



MPI PROFILING

Visual Profiler

MPI Rank-based naming



NVTX Markers & Ranges

MPI PROFILING

MPI + NVTX

Manual mode

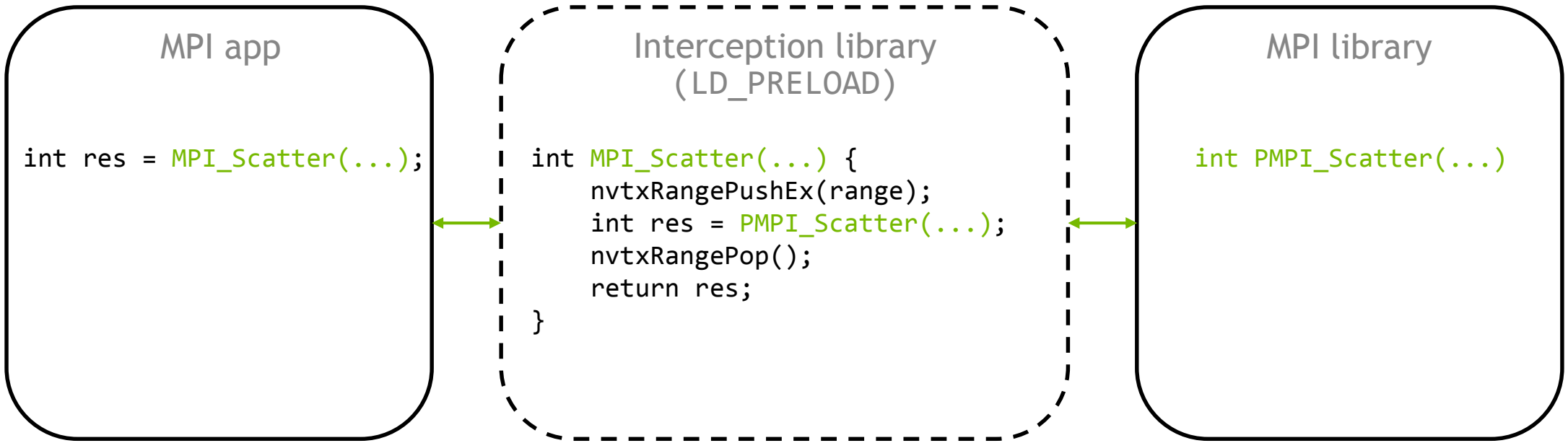
```
nvtxEventAttributes_t range = {0};
range.message.ascii = "MPI_Scatter";
nvtxRangePushEx(range);
int result = MPI_Scatter(...);
nvtxRangePop();
```

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.

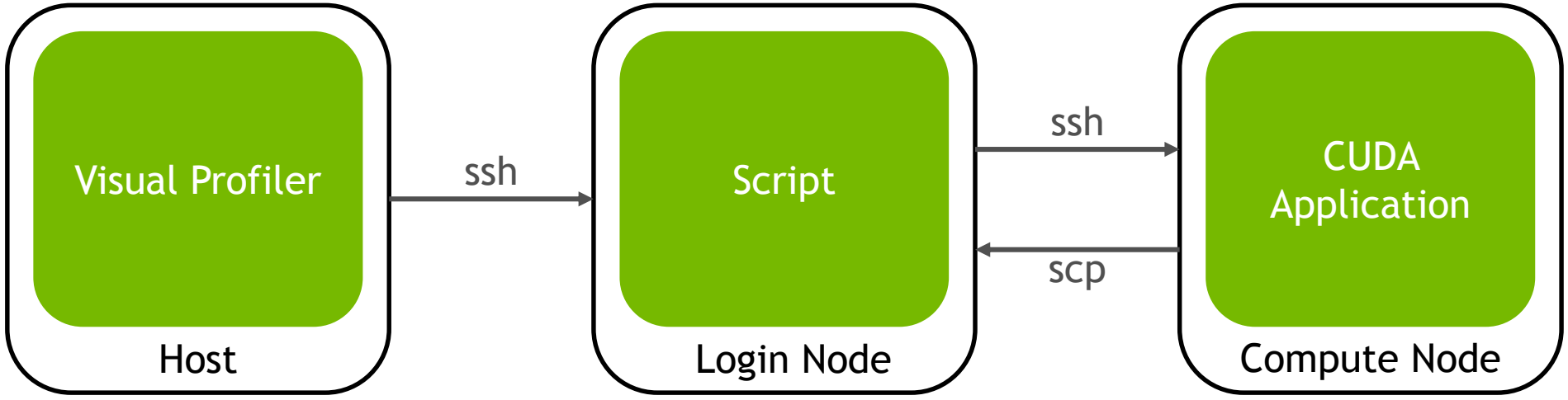
MPI PROFILING

Interception



REMOTE PROFILING

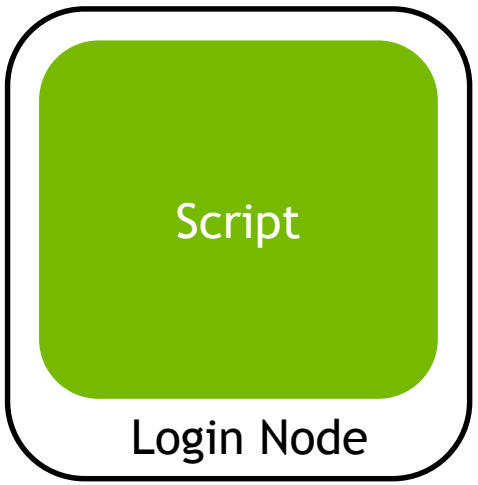
NVVP: MULTI-HOP REMOTE PROFILING



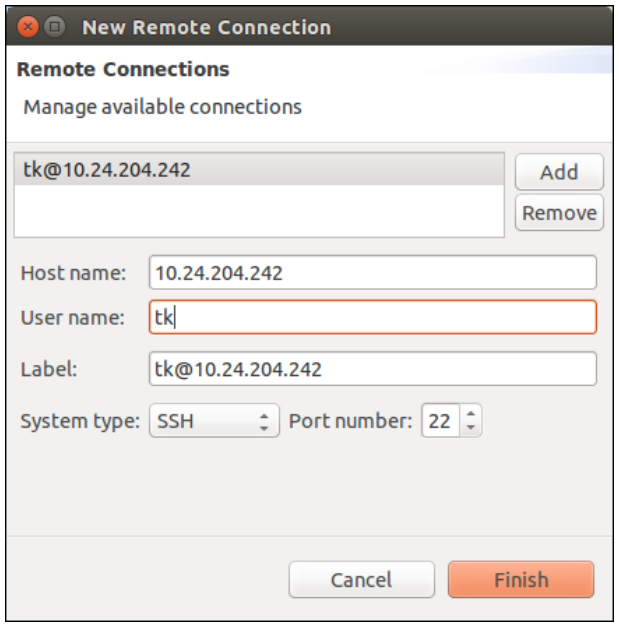
NVVP: MULTI-HOP REMOTE PROFILING

One-Time Setup

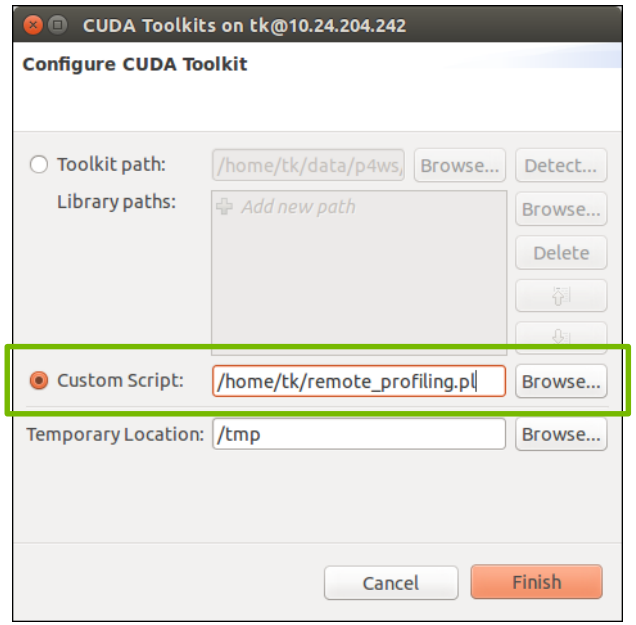
1. Configure script on the login node



2. Connect Visual Profiler to the login node



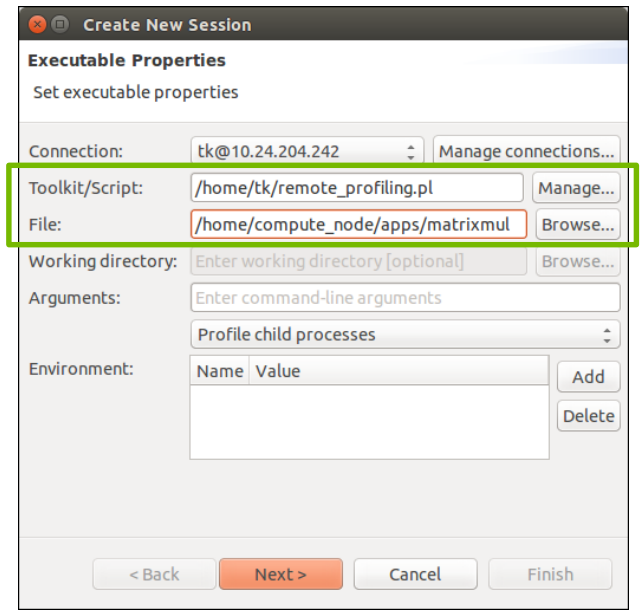
3. Use the custom script option



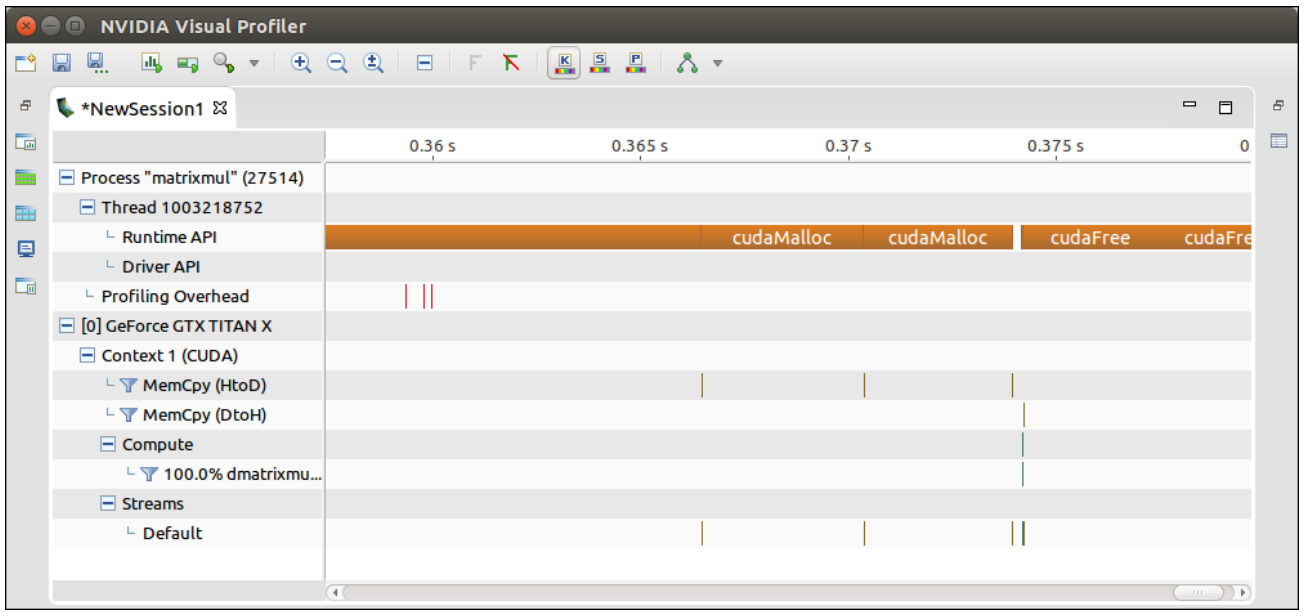
NVVP: MULTI-HOP REMOTE PROFILING

Application Profiling

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

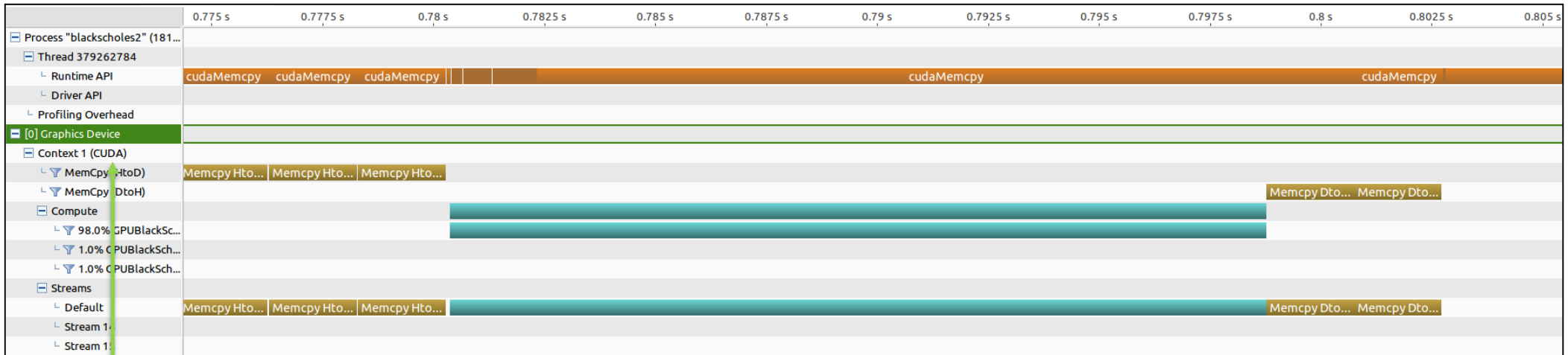


2. Application transparently runs on compute node and profiling data is displayed in the Visual Profiler



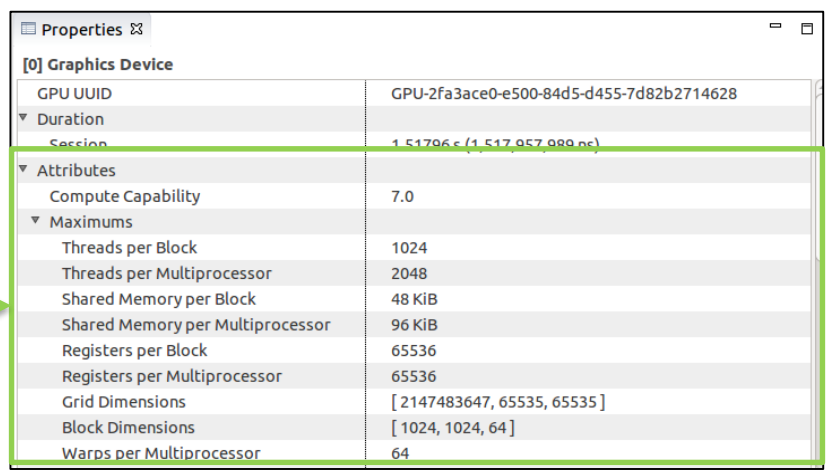
VOLTA SUPPORT

VOLTA SUPPORT



GPU Trace

GV100 Device Attributes



OTHER IMPROVEMENTS

OTHER IMPROVEMENTS

Tracing and profiling of Cooperative Kernel launches is supported

The Visual Profiler supports remote profiling to systems supporting ssh algorithms with a key length of 2048 bits

OpenACC profiling is now supported on systems without CUDA setup

nvprof flushes all profiling data when a SIGINT or SIGKILL signal is encountered

REFERENCES

NVIDIA toolkit documentation: http://docs.nvidia.com

CUDA Profiler User Guide: http://docs.nvidia.com/cuda/profiler-users-guide/index.html

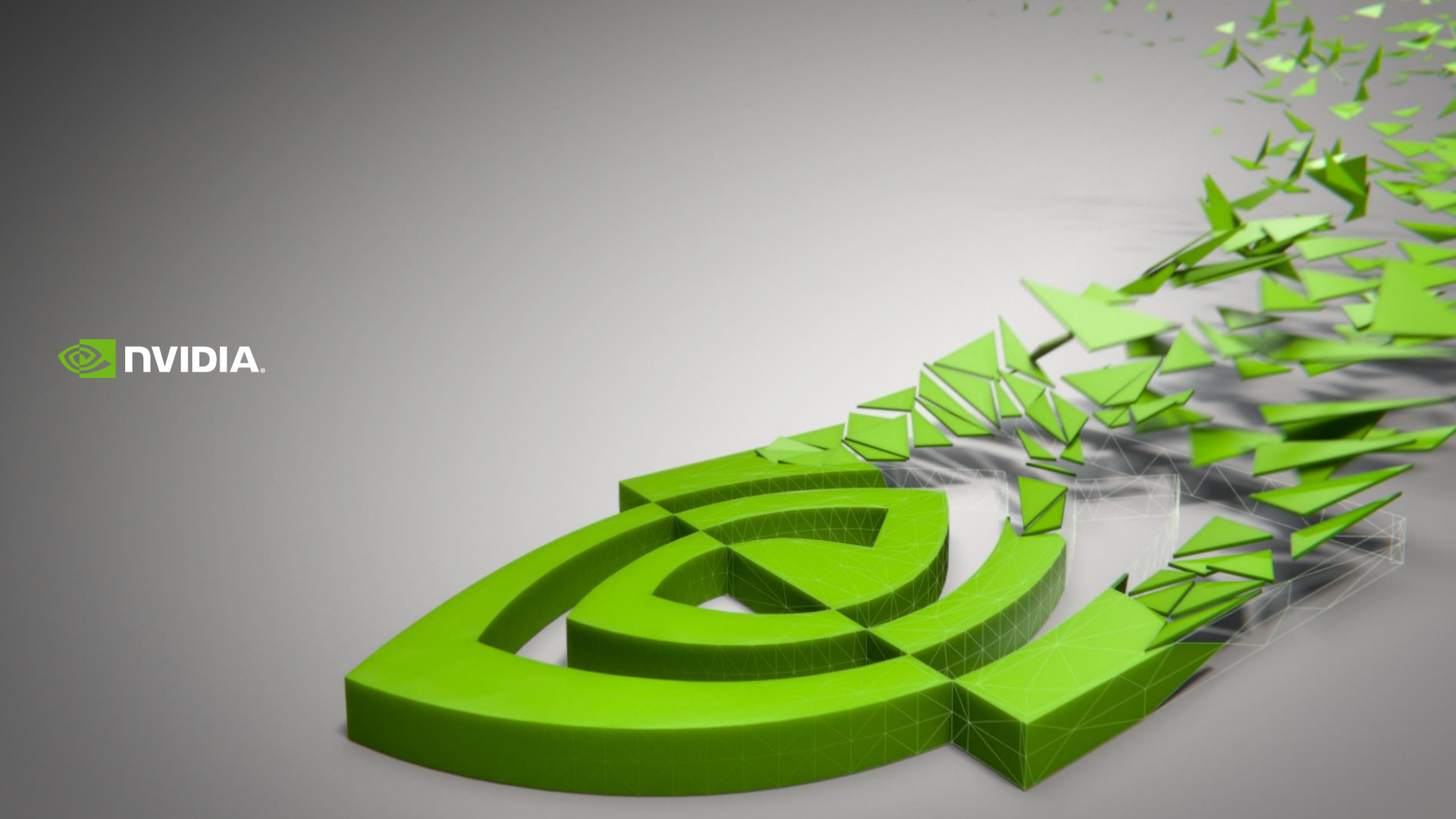
Other GTC 2017 sessions:

S7824 - DEVELOPER TOOLS UPDATE IN CUDA 9

S7519 - DEVELOPER TOOLS FOR AUTOMOTIVE, DRONES AND INTELLIGENT CAMERAS APPLICATIONS

S7445 - WHAT THE PROFILER IS TELLING YOU: OPTIMIZING WHOLE APPLICATION PERFORMANCE

S7444 - WHAT THE PROFILER IS TELLING YOU: OPTIMIZING GPU KERNELS


NVIDIA