NVIDIA PROFILING TOOLS OVERVIEW

François Courteille, Principal Solution Architect, fcourteille@nvidia.com

Introduction to NVIDIA profilers

Nvprof and the Visual Profiler

Nsight Systems

Profiling from CLI

NVTX (NVIDIA Tools Extension)

Getting Started Resources

NVIDIA PROFILING TOOLS

Nsight Systems

Nsight Compute

Nsight Visual Studio Edition

Nsight Eclipse Edition

NVIDIA Profiler (nvprof)

NVIDIA Visual Profiler (nvvp)

CUPTI (CUDA Profiling Tools Interface)

NVIDIA

TOOLS COMPARISON

	NVIDIA© Nsight™ Systems	NVIDIA© Nsight™ Compute	NVIDIA© Visual Profiler	Intel© VTune™ Amplifier	Linux perf OProfile
Target OS	**Linux, Windows**	Linux, Windows	Linux, Mac, Windows	Linux, Windows	Linux
GPUs	**Pascal+**	Pascal+	Kepler+	None	None
CPUs	**x86_64**	x86_64	x86, x86_64, Power	x86, x86_64	x86, x86_64, Power
Trace	**NVTX, OS runtime, CUDA, CuDNN, CuBLAS, OpenACC, OpenGL, DX12**	NVTX, CUDA	MPI, CUDA, OpenACC, NVTX	MPI, ITT	Kernel
CPU PC Sampling	**High Speed**	No	Yes	High Speed	High Speed
NVLINK, GPU Power, Thermal	**Future**		Yes	No	No
Src Code View	**No**	Yes	Yes	Yes	No
Compare Sessions	**No**	Yes	No	Yes	No

NVIDIA.

NVPROF AND THE VISUAL PROFILER

NVIDIA PROFILER

Some examples of how to use it

- GUI : nvvp

- Command line:
  nvprof <application> <application options>

- Command line with GUI output:
  nvprof –o prof.nvvp <application> <application options>

- Command line, MPI applications, GUI output:
  mpirun –np 4 nvprof –o prof_%h_%p.nvvp <application> <application options>
  (%h and %p will be replaced by host and pid)

- Command line, capturing all low level metrics for later GUI analysis (slow!)
  nvprof --analysis-metrics –o prof.nvvp <application> <application options>

- Command line, all low level metrics for later GUI analysis only for the 2nd occurrence of each kernel:
  nvprof --kernels :::2 --analysis-metrics -o prof.nvvp <application> <application options>

- Command line with GUI output, limiting the profiling to the first 60seconds (60s after GPU context creation)
  nvprof –t 60 –o prof.nvvp <application> <application options>

See the profiler documentation for more information

NVPROF - Basic Usage

```
$ nvprof ./cg
==88109== NVPROF is profiling process 88109, command: ./cg
<program output>
==88109== Profiling application: ./cg
==88109== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 95.62%  4.74211s       101  46.952ms  43.029ms  435.09ms  matvec(matrix const &, vector const &,
vector const &)_12_gpu
  3.28%  162.50ms       302  538.09us  236.55us  31.155ms  waxpby(double, vector const &, double,
vector const &, vector const &)_26_gpu
  0.73%  36.165ms       200  180.83us  138.18us  223.26us  dot(vector const &, vector const
&)_10_gpu
  0.37%  18.310ms       200  91.549us  89.664us  93.441us  dot(vector const &, vector const
&)_10_gpu_red
  0.00%  100.13us       200     500ns     480ns  1.0240us  [CUDA memcpy HtoD]
  0.00%  81.408us       200     407ns     384ns     416ns  [CUDA memcpy DtoH]

==88109== Unified Memory profiling result:
Device "Tesla P100-SXM2-16GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   13783  197.40KB  64.000KB  960.00KB  2.594727GB  99.26990ms  Host To Device
    7681         -         -         -           -  298.1818ms  GPU Page fault groups
Total CPU Page faults: 7973
```

CUDA VISUAL PROFILER

Overview of key features

Kernel profile – memory hierarchy view

Unified Memory

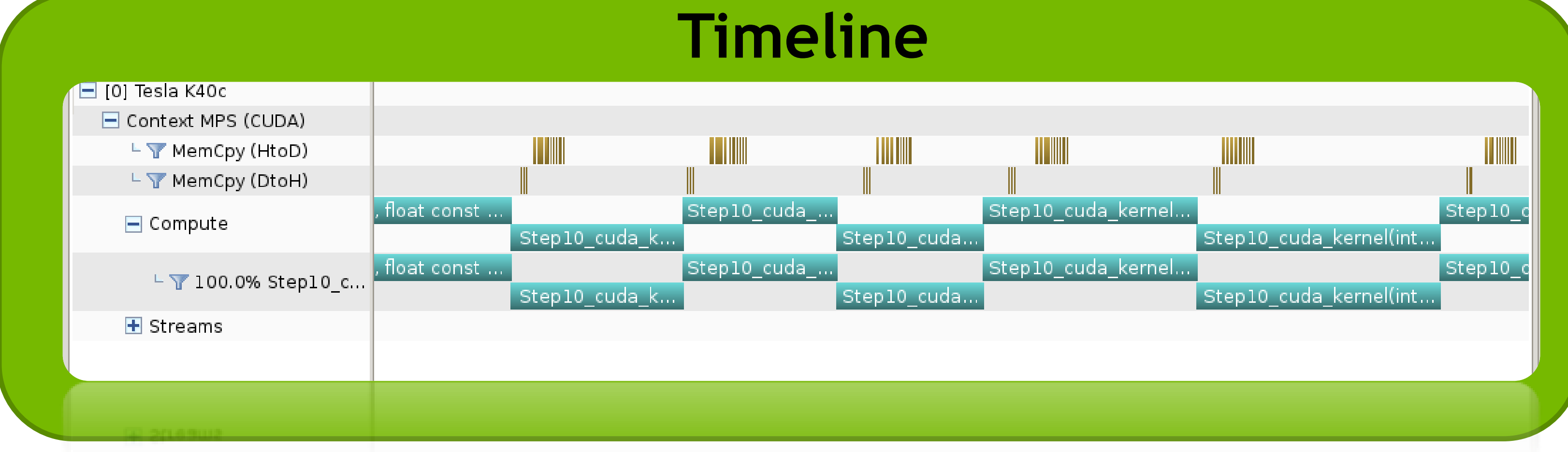
NVLink

PC sampling

OpenACC/OpenMP Profiling

NVTX

NVIDIA'S VISUAL PROFILER (NVVP)



Guided System

1. CUDA Application Analysis
2. Performance-Critical Kernels
3. Compute, Bandwidth, or Latency Bound

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "Step10_cuda_kernel" is most likely limited by compute.

Perform Compute Analysis

The most likely bottleneck to performance for this kernel is compute so you should first perform compute analysis to determine how it is limiting performance.

Perform Latency Analysis

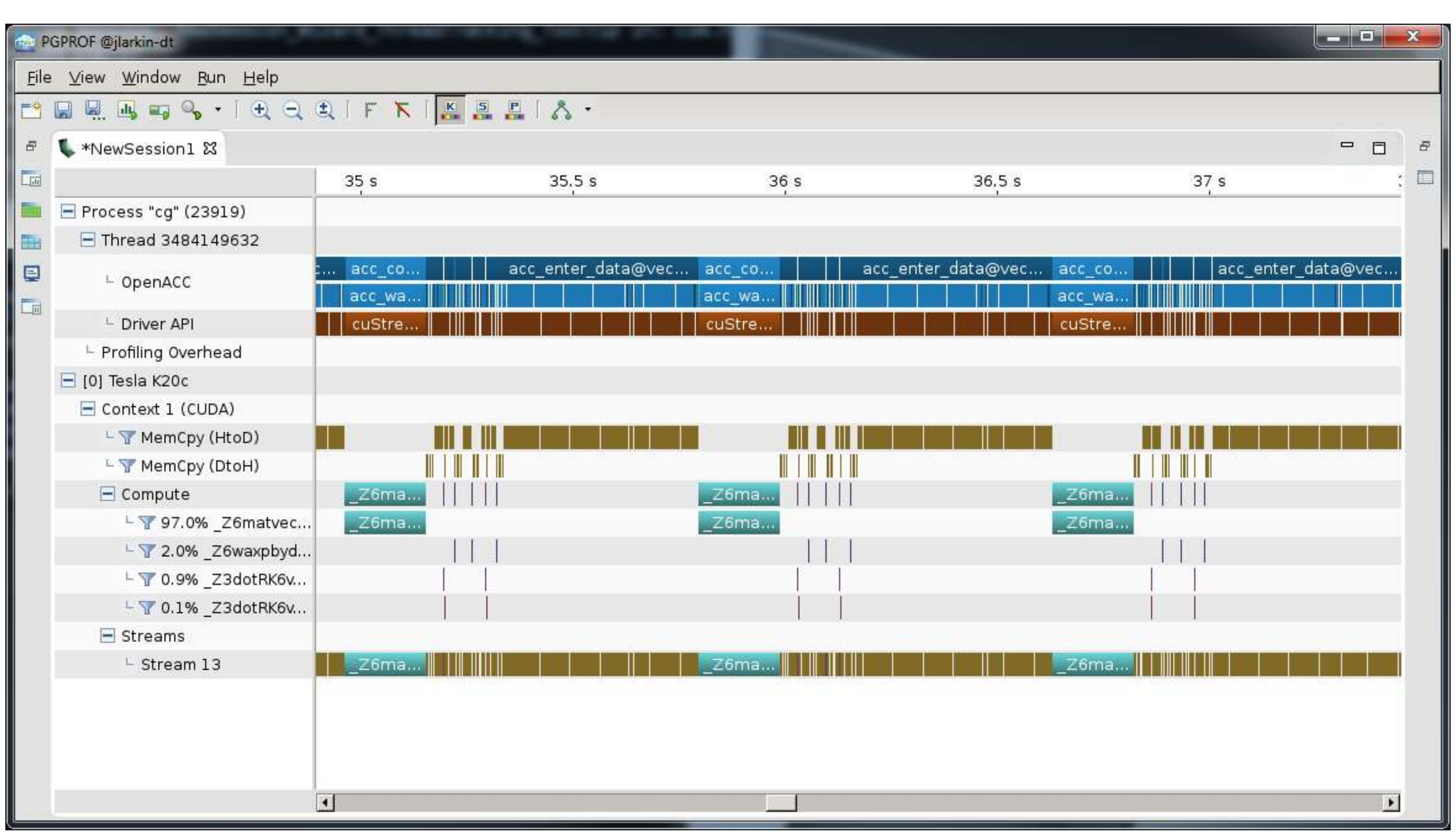
Perform Memory Bandwidth Analysis

Instruction and memory latency and memory bandwidth are likely not the primary performance bottlenecks for this kernel, but you may still want to perform those analyses.

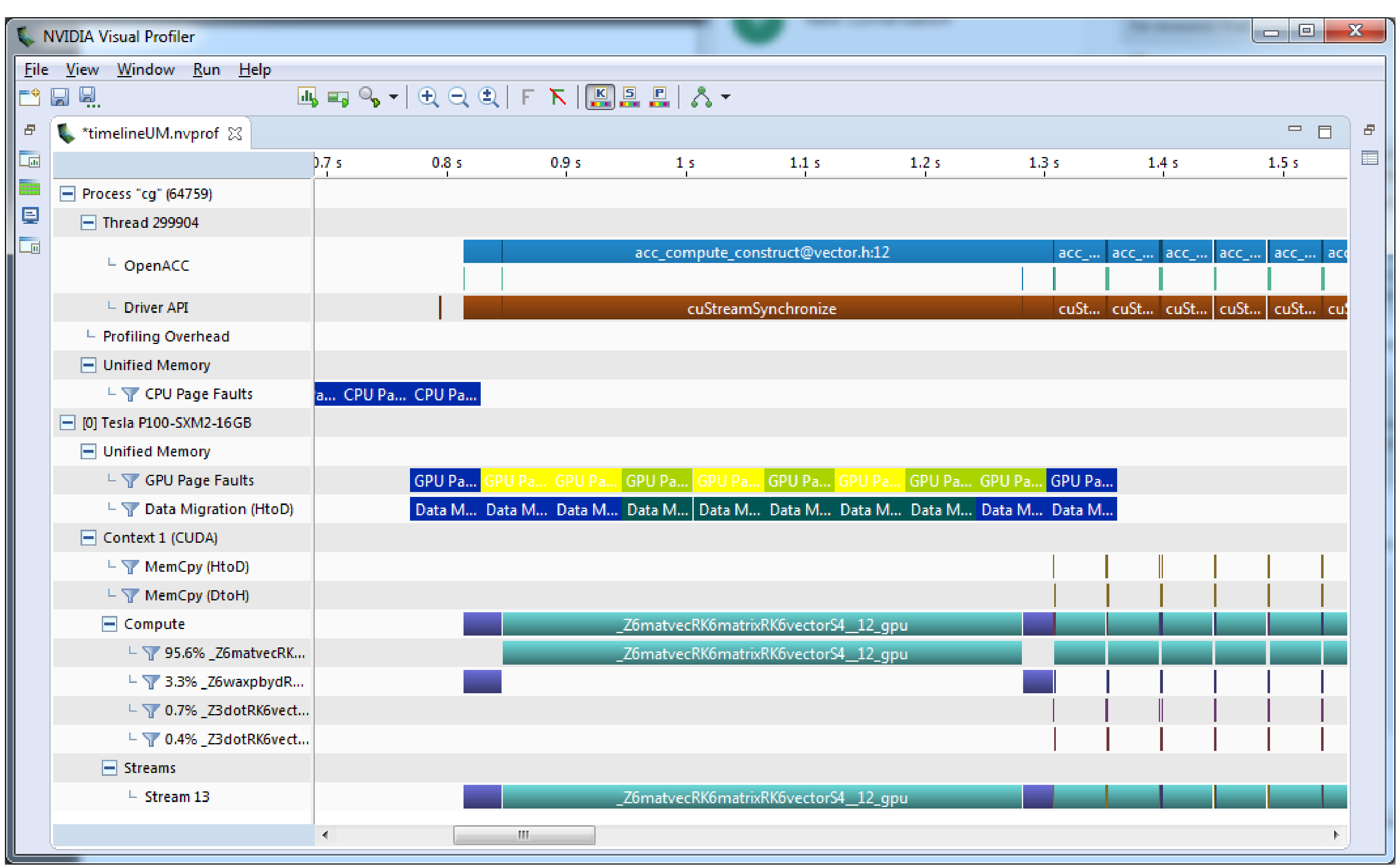
Rerun Analysis

If you modify the kernel you need to rerun your application to update this analysis.

DATA MOVEMENT IN VISUAL PROFILER

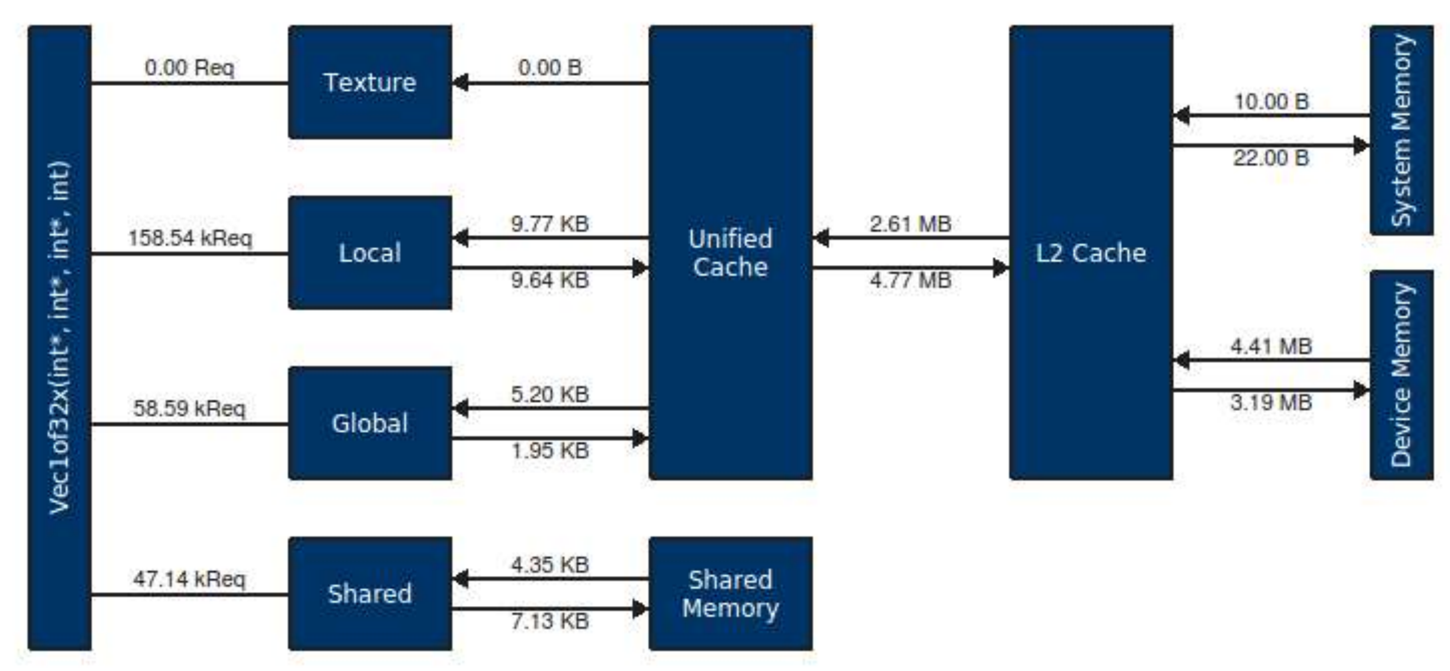


UVM IN VISUAL PROFILER



KERNEL PROFILE

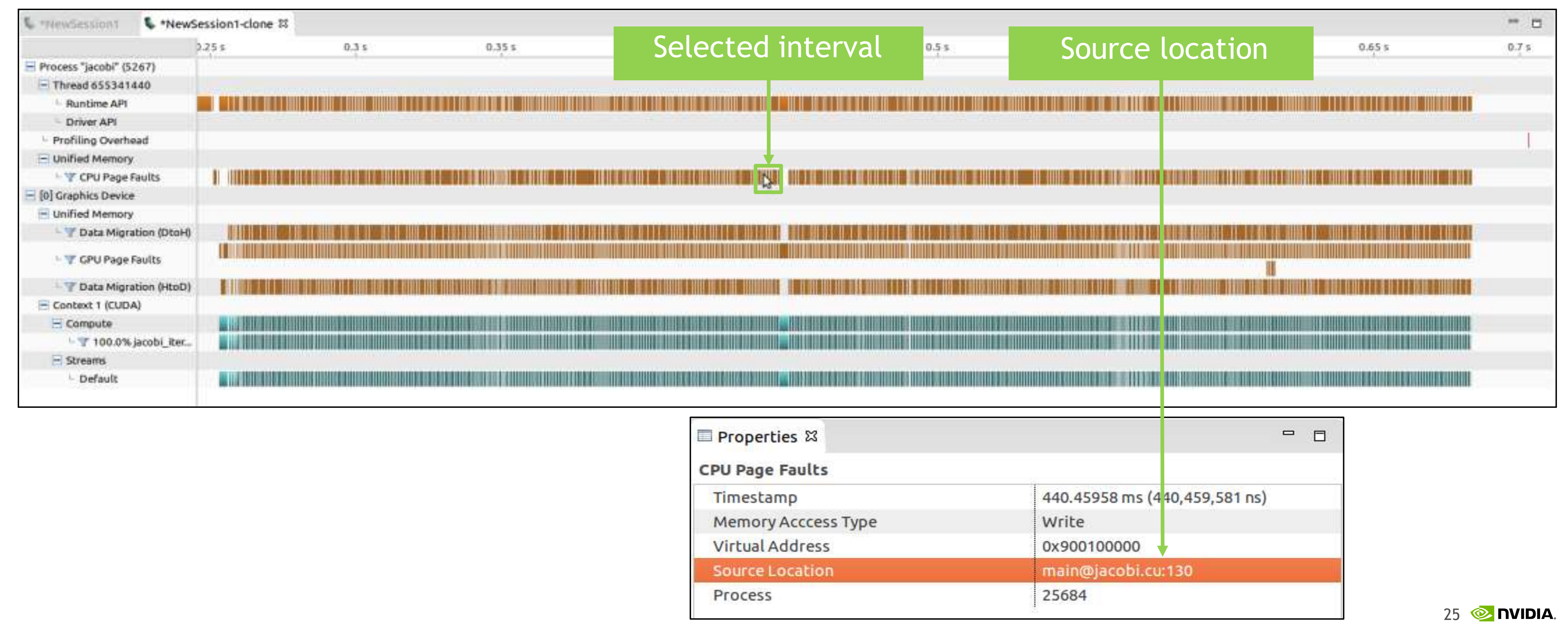
Memory hierarchy view



NVIDIA.

VISUAL PROFILER

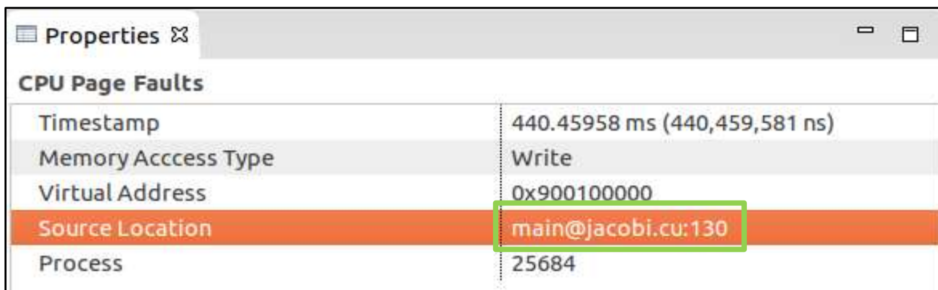
CPU Page Fault Source Correlation



NVIDIA.

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

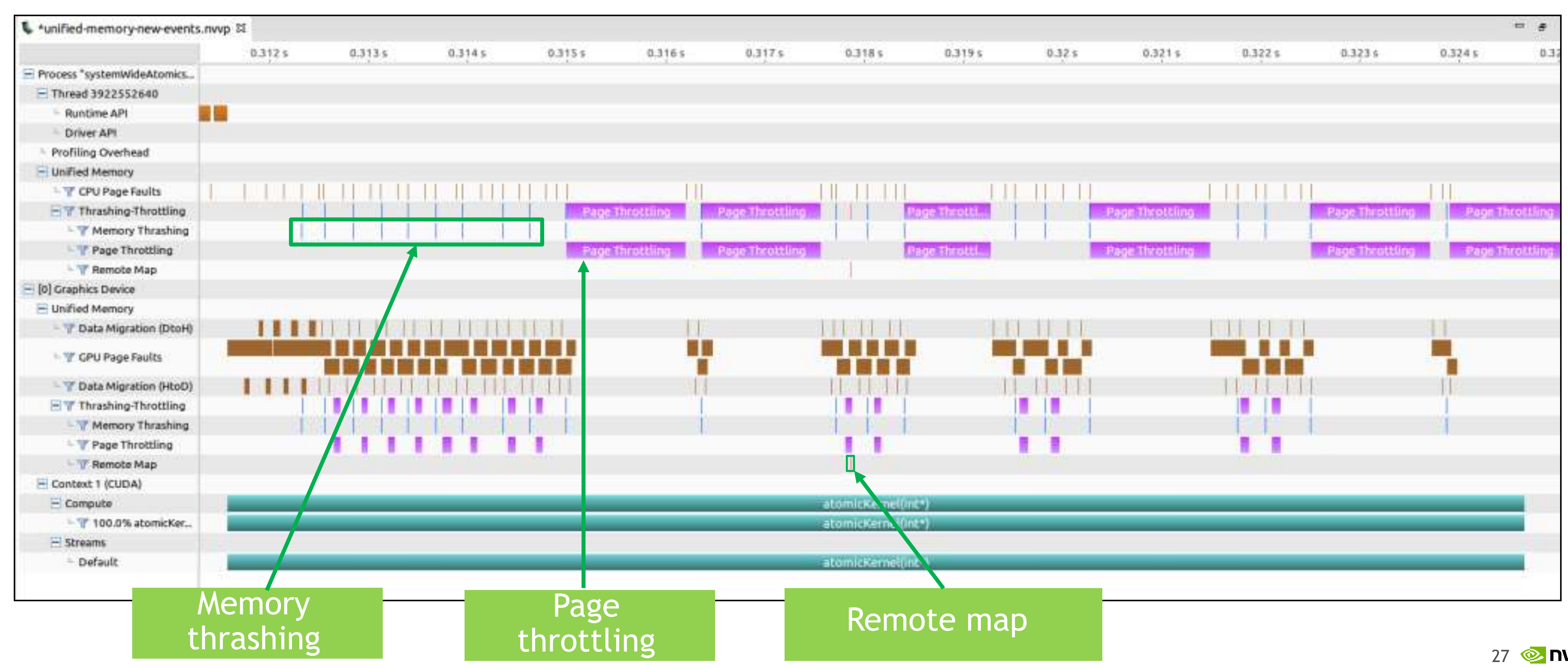
Source line causing CPU page fault

*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

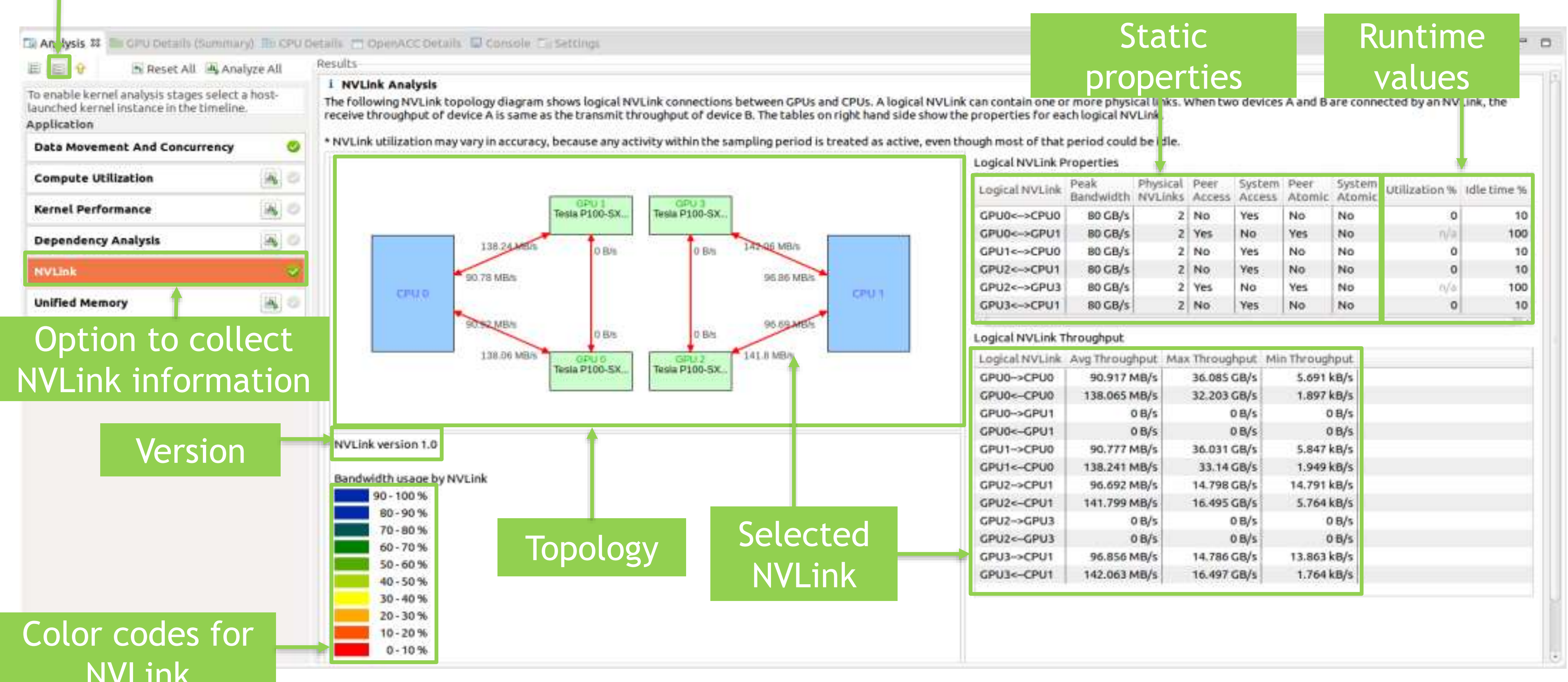


NVIDIA

VISUAL PROFILER

NVLINK visualization

Unguided Analysis

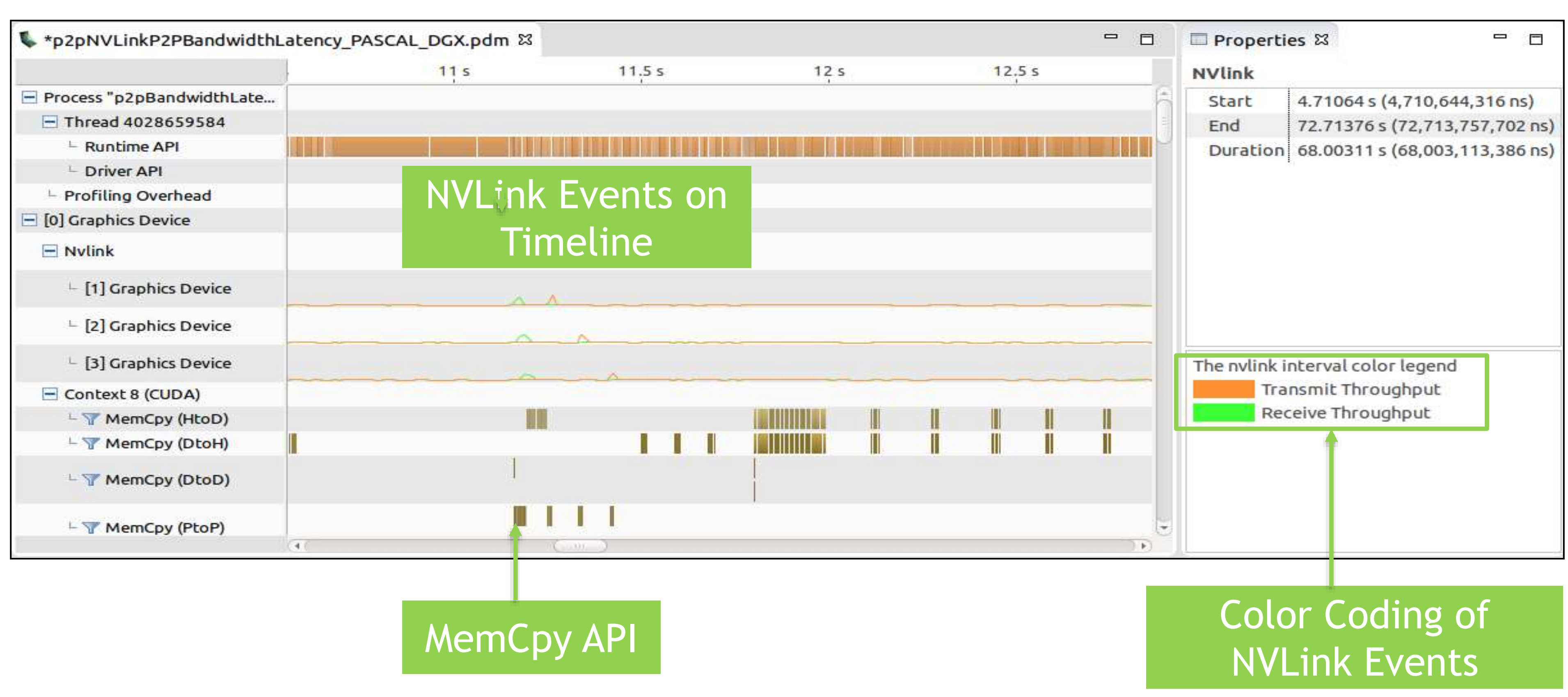


Logical NVLink	Peak Bandwidth	Physical NVLinks	Peer Access	System Access	Peer Atomic	System Atomic	Utilization %	Idle time %
GPU0<->CPU0	80 GB/s	2	No	Yes	No	No	0	10
GPU0<->GPU1	80 GB/s	2	Yes	No	Yes	No	n/a	100
GPU1<->CPU0	80 GB/s	2	No	Yes	No	No	0	10
GPU2<->CPU1	80 GB/s	2	No	Yes	No	No	0	10
GPU2<->GPU3	80 GB/s	2	Yes	No	Yes	No	n/a	100
GPU3<->CPU1	80 GB/s	2	No	Yes	No	No	0	10

Logical NVLink	Avg Throughput	Max Throughput	Min Throughput
GPU0->CPU0	90.917 MB/s	36.085 GB/s	5.691 kB/s
GPU0<-CPU0	138.065 MB/s	32.203 GB/s	1.897 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	90.777 MB/s	36.031 GB/s	5.847 kB/s
GPU1<-CPU0	138.241 MB/s	33.14 GB/s	1.949 kB/s
GPU2->CPU1	96.692 MB/s	14.798 GB/s	14.791 kB/s
GPU2<-CPU1	141.799 MB/s	16.495 GB/s	5.764 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	96.856 MB/s	14.786 GB/s	13.863 kB/s
GPU3<-CPU1	142.063 MB/s	16.497 GB/s	1.764 kB/s

VISUAL PROFILER

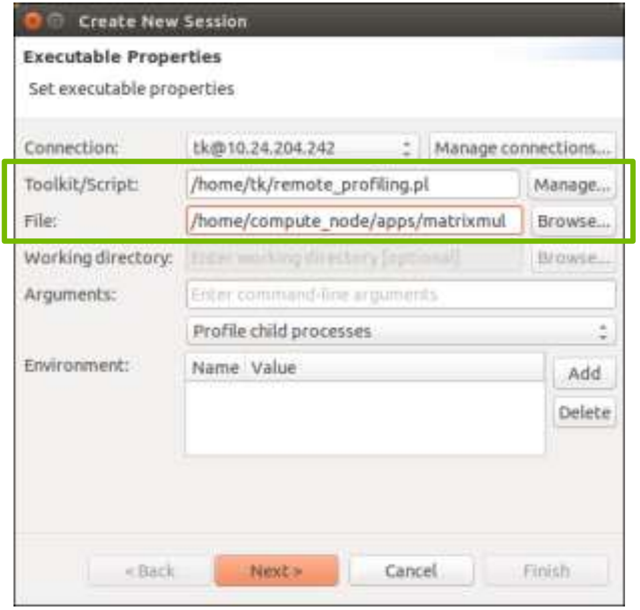
NVLink events on timeline



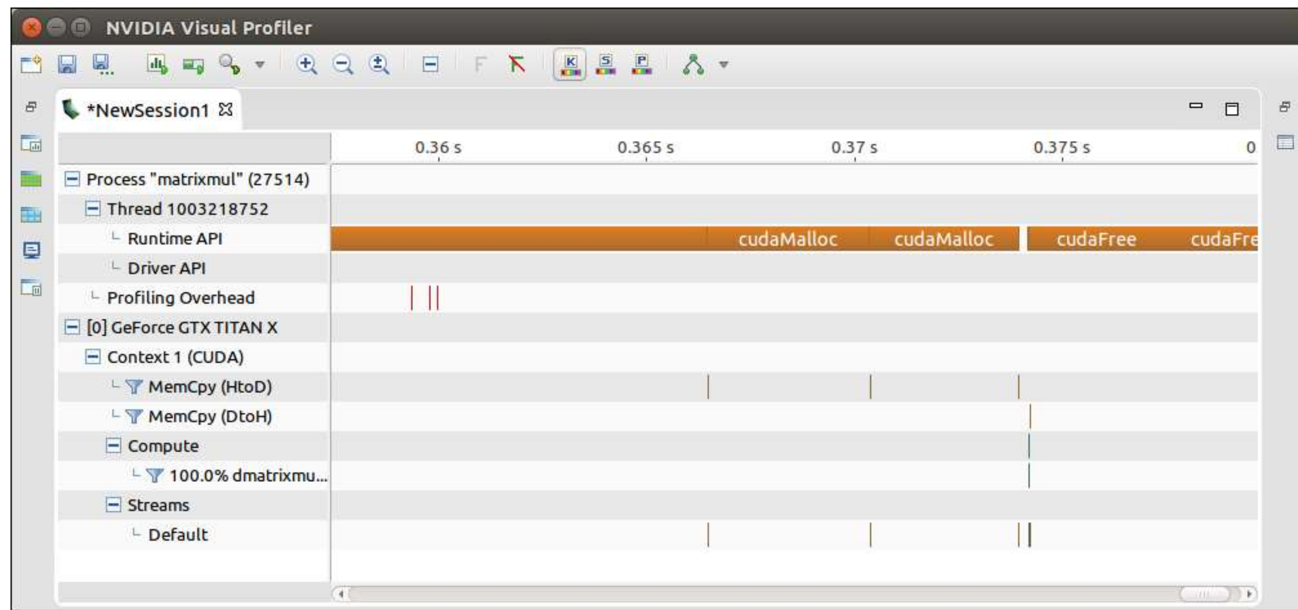
VISUAL PROFILER

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



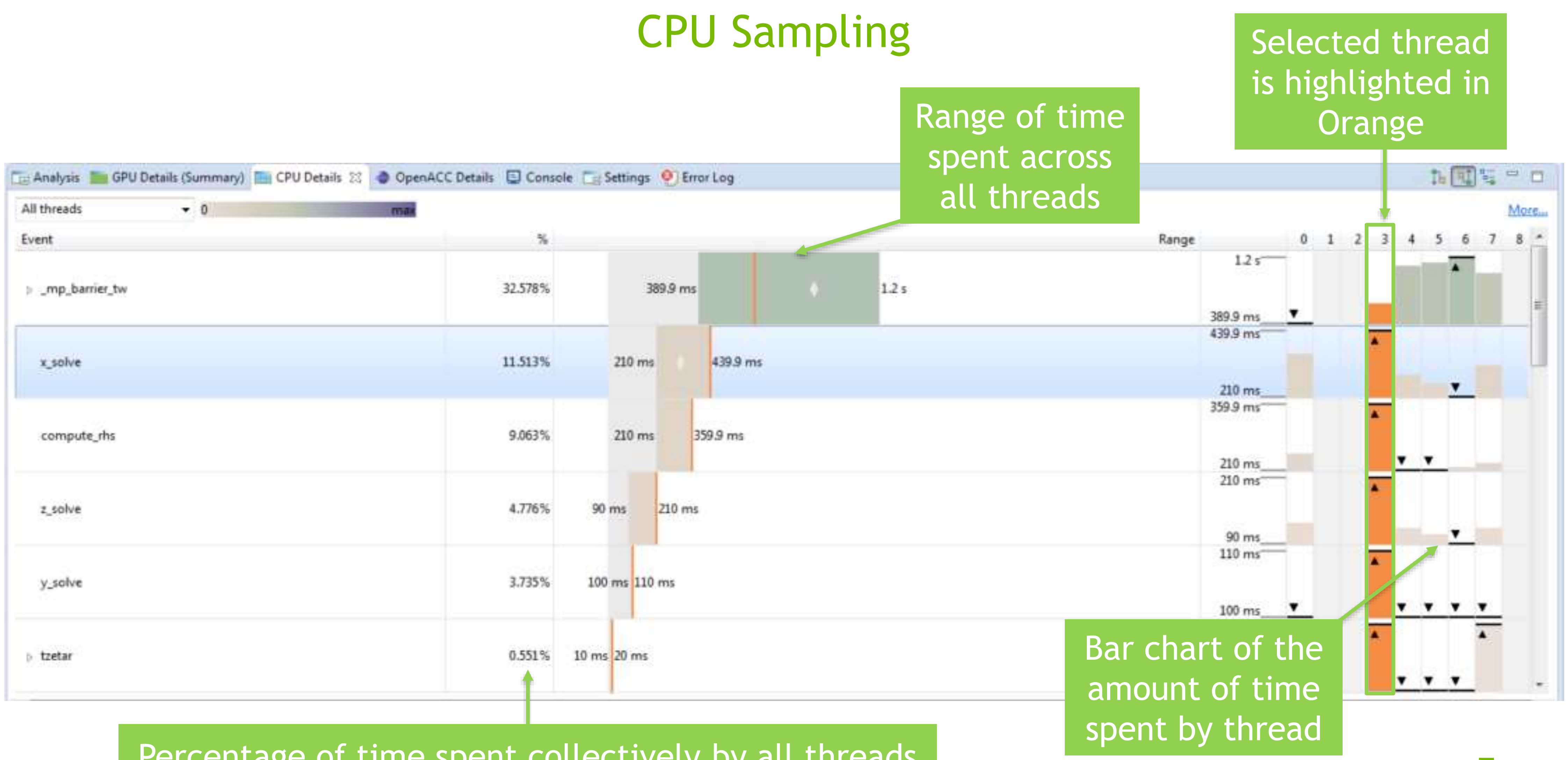
NVIDIA.

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



NVIDIA

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

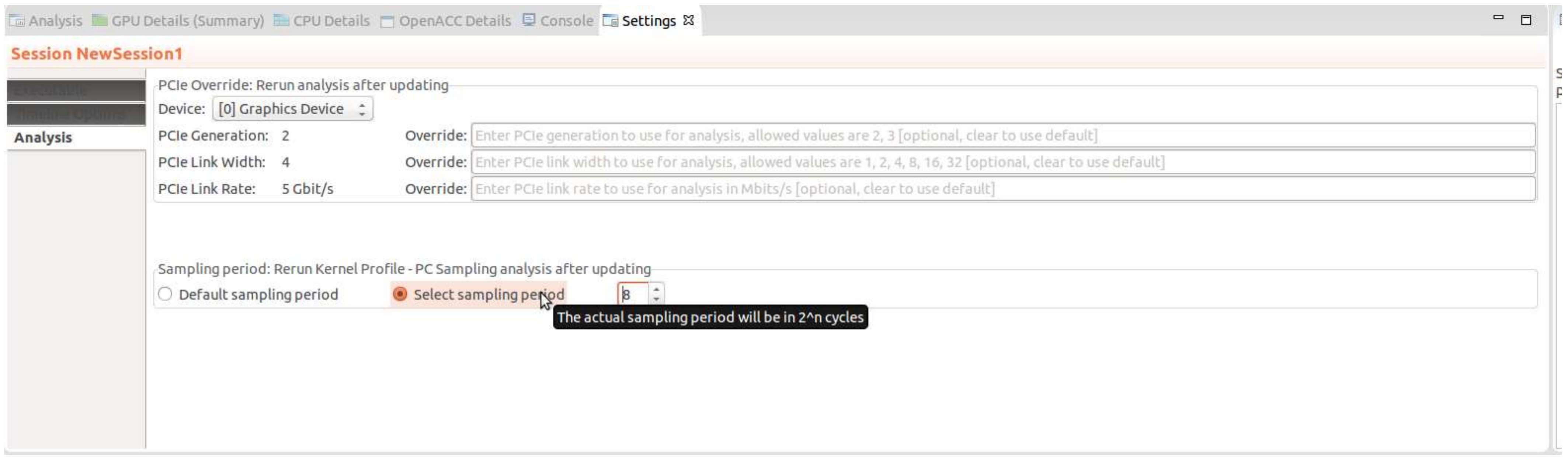
Effective in optimizing large kernels, pinpoints performance bottlenecks at specific lines in source code or assembly instructions

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

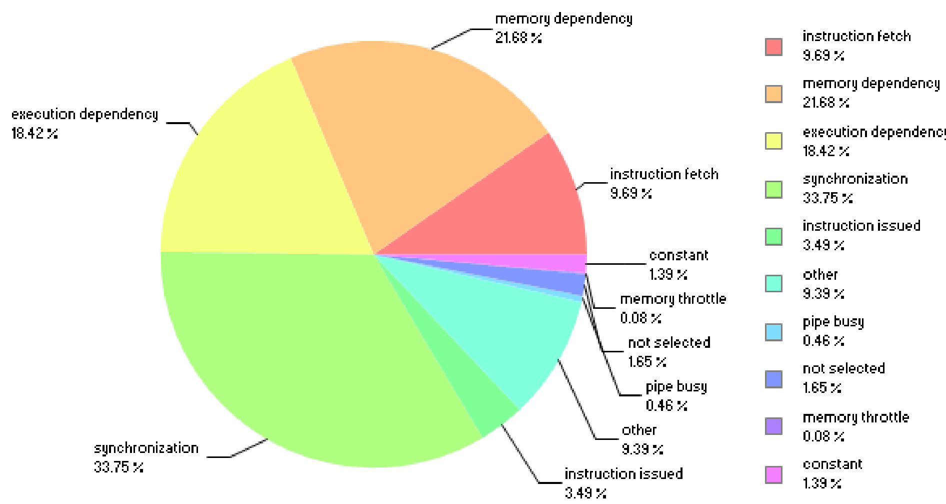


NVIDIA

VISUAL PROFILER

PC SAMPLING UI

Pie chart for sample distribution for a CUDA function



Line Warp State File - /C:/swapnaofficial/Maxwell/sampling/SFM/session1/estimate_combined1.cu

Stack bar with stall reasons

Hotspots

Hotspot source line with highest number of samples

Warp State Disassembly

Total Sample Count = 1288
synchronization = 847 (65.8%)
other = 263 (20.4%)
instruction fetch = 69 (5.4%)
execution dependency = 36 (2.8%)
memory dependency = 32 (2.5%)
instruction issued = 23 (1.8%)
pipe busy = 9 (0.7%)
not selected = 8 (0.6%)
memory throttle = 1 (0.1%)

Tooltip with distribution of stall reasons

Source-Assembly view

NVPROF - MPI Profiling

NVPROF & Visual Profiler do not natively understand MPI

It is possible to load data from multiple MPI ranks (same or different GPUS) into Visual Profiler though

Trick: Label your data to know which MPI rank it came from:

- `--process-name "MPI Rank %q{OMPI_COMM_WORLD_RANK}" --context-name "MPI Rank %q{OMPI_COMM_WORLD_RANK}”`

Trick: Give every rank its own filename

- `-o timeline.%q{OMPI_COMM_WORLD_RANK}.nvprof`

See https://devblogs.nvidia.com/parallelforall/gpu-pro-tip-track-mpi-calls-nvidia-visual-profiler/ for one more trick for showing MPI calls on your timeline.

NVPROF - MPI Profiling

```
$ mpirun -n 4 --gpu 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}.nvprof ./laplace2d.solution
==89073== NVPROF is profiling process 89073, command: ./laplace2d.solution
==89070== NVPROF is profiling process 89070, command: ./laplace2d.solution
==89075== NVPROF is profiling process 89075, command: ./laplace2d.solution
==89066== NVPROF is profiling process 89066, command: ./laplace2d.solution
<program output>
==89075== Generated result file: timeline.3.nvprof
==89073== Generated result file: timeline.2.nvprof
==89070== Generated result file: timeline.1.nvprof
==89066== Generated result file: timeline.0.nvprof
```

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 …
```

PROFILING MPI+CUDA APPLICATIONS

Using nvprof+NVVP

New since CUDA 9

Embed MPI rank in output filename, process name, and context name (OpenMPI)

```
nvprof --output-profile profile.%q{OMPI_COMM_WORLD_RANK}      \
                   --process-name "rank %q{OMPI_COMM_WORLD_RANK}"      \
                   --context-name "rank %q{OMPI_COMM_WORLD_RANK}"      \
                   --annotate-mpi openmpi
```

Alternatives:

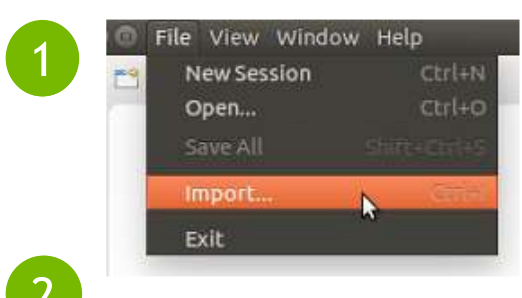
Only save the textual output (--log-file)

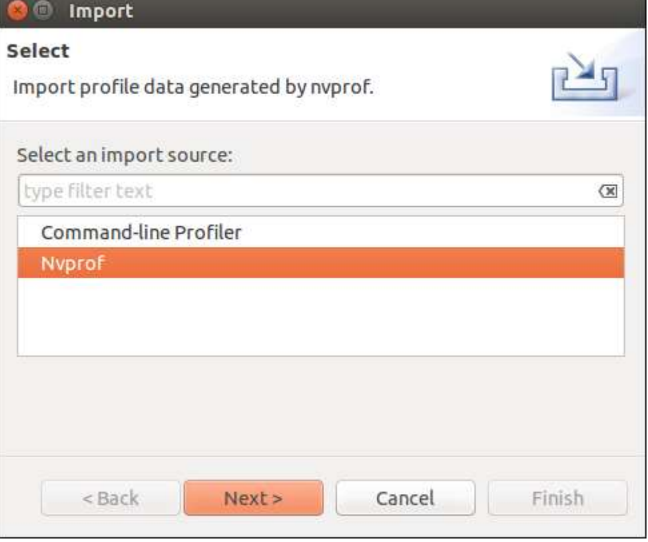
Collect data from all processes that run on a node (--profile-all-processes)

MVAPICH2: MV2_COMM_WORLD_RANK
--annotate-mpi mpich

MPI PROFILING

Importing into the Visual Profiler





3

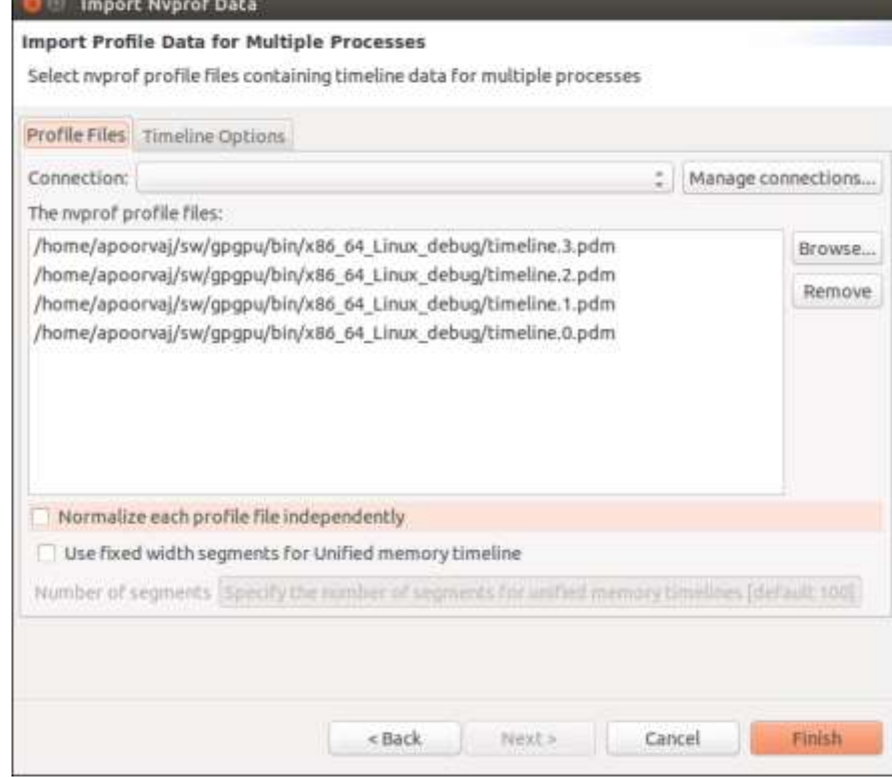
Import Nvprof Data

Nvprof profile files

Import profile data for a single process or for multiple processes

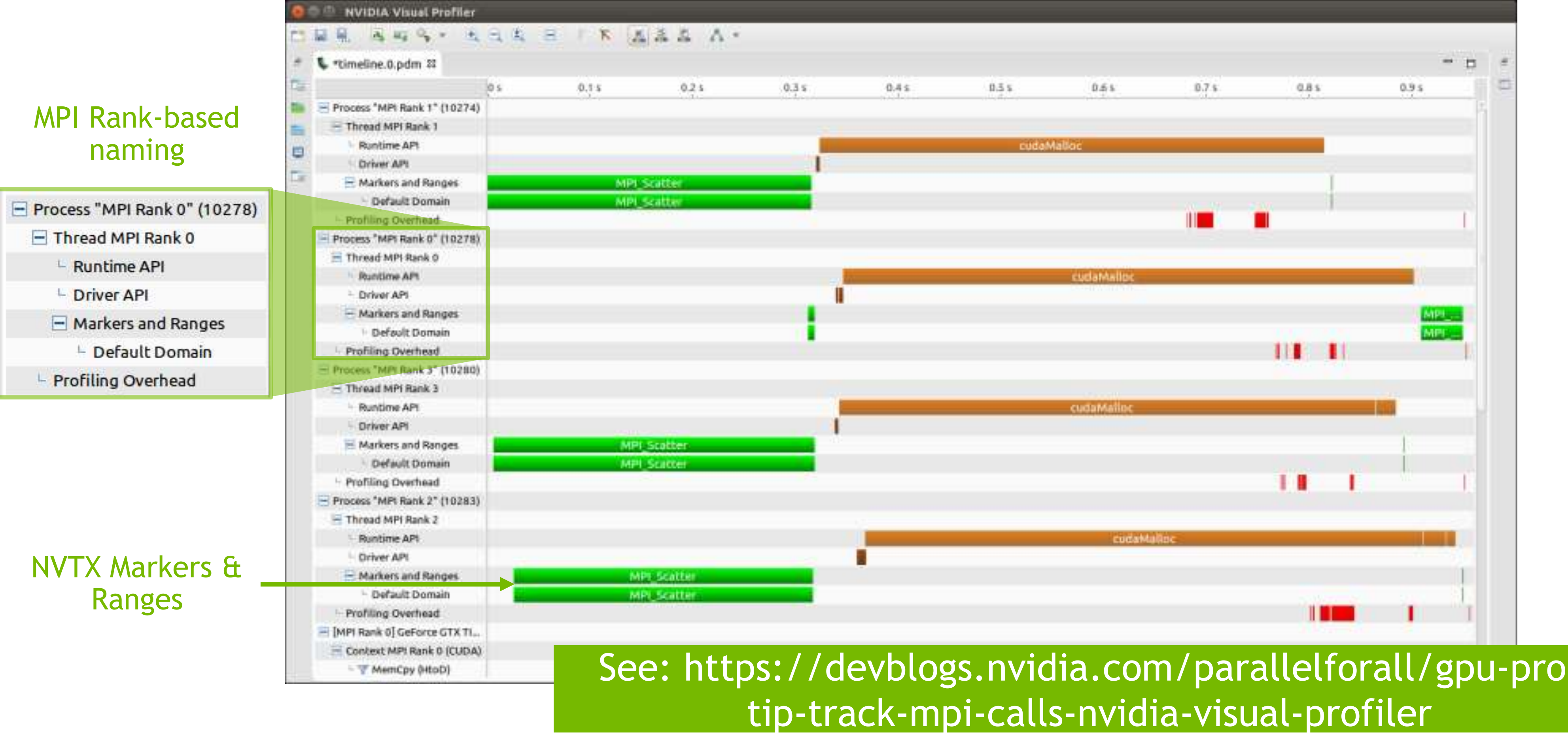
- Single process
- Multiple processes

< Back | Next > | Cancel | Finish



MPI PROFILING

Visual Profiler



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 …
```

context:stream:kernel:invocation

Record metrics for only the first invocation of each kernel.

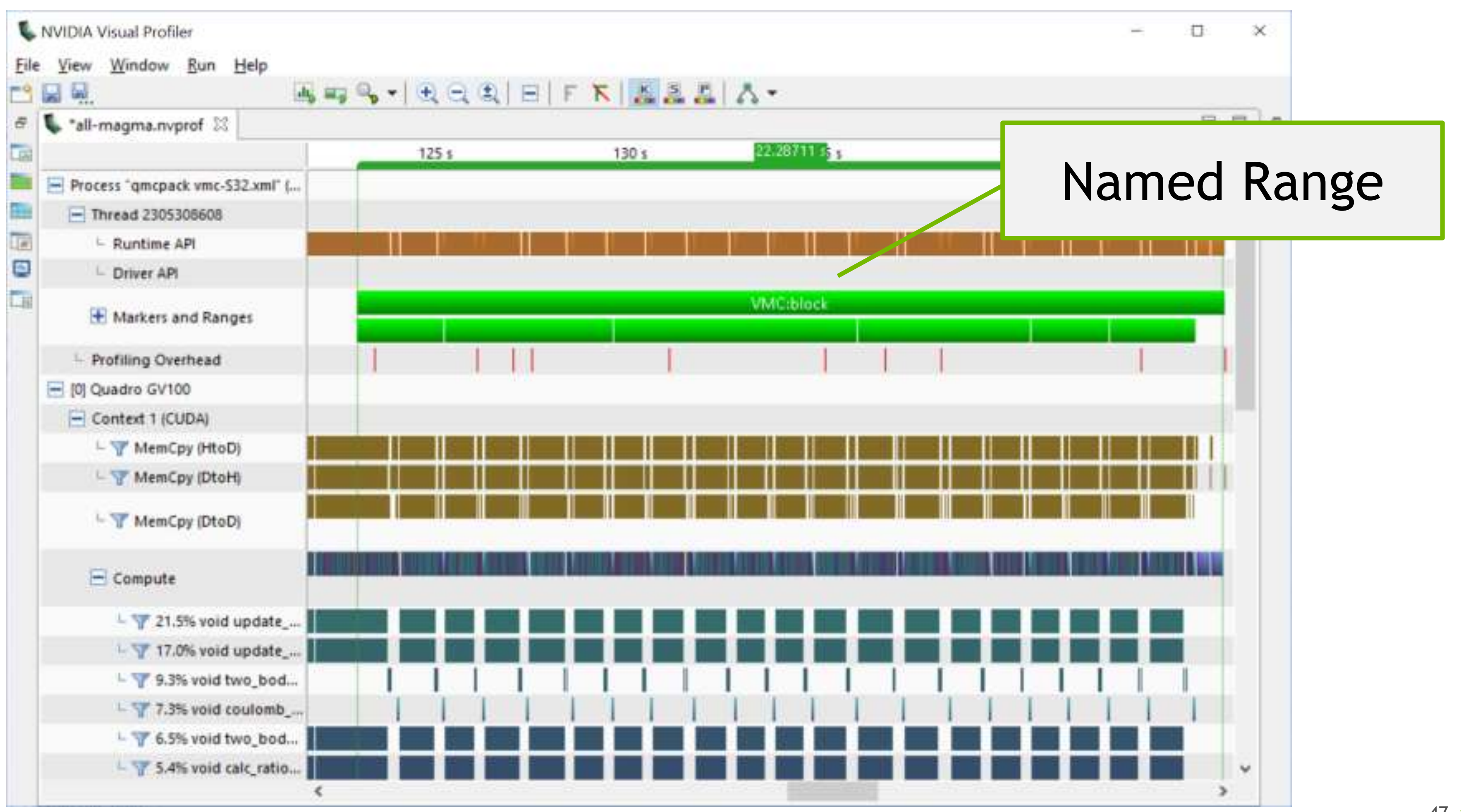
NVTX ANNOTATIONS

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



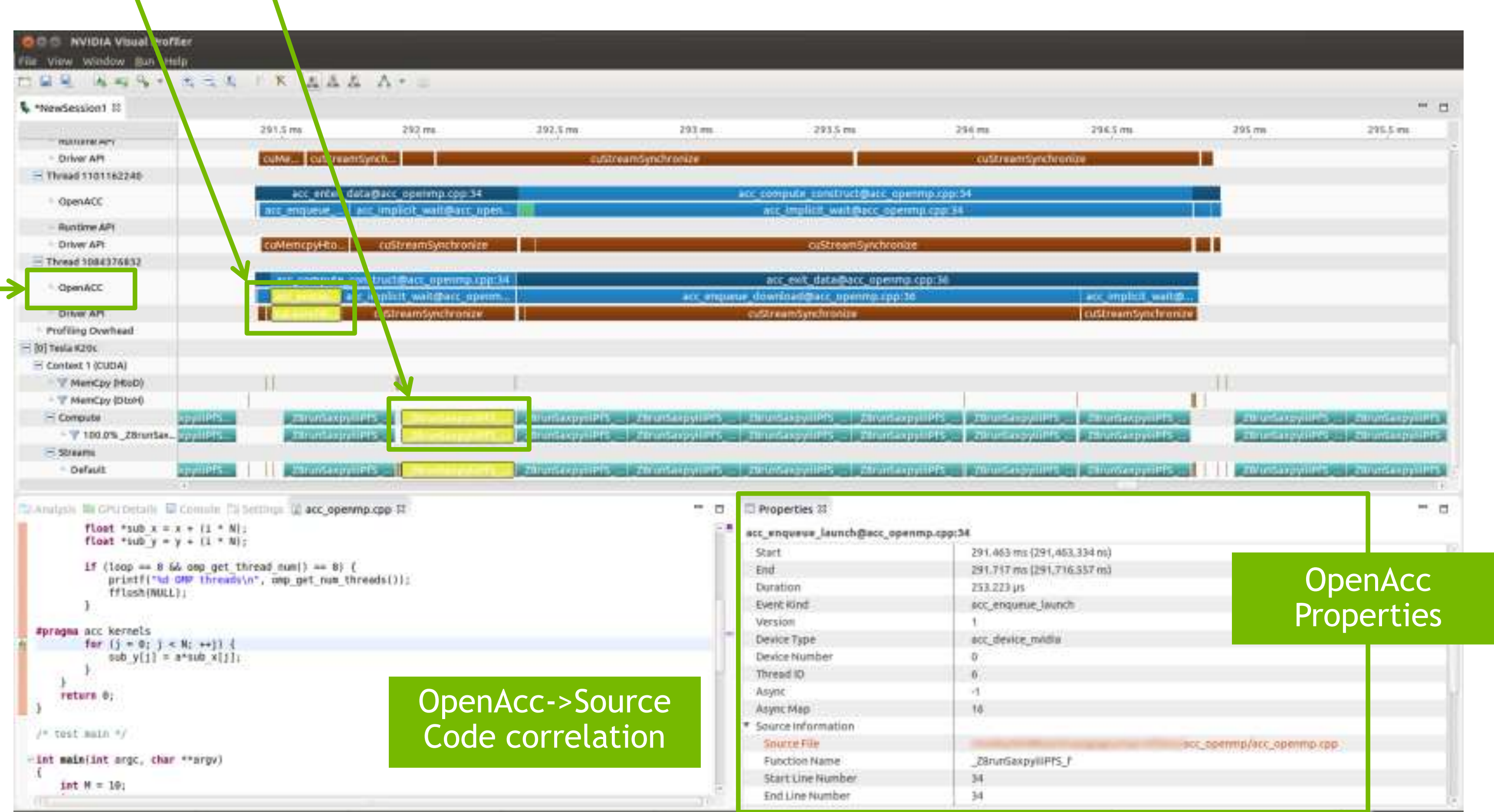
EXPORTING DATA

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

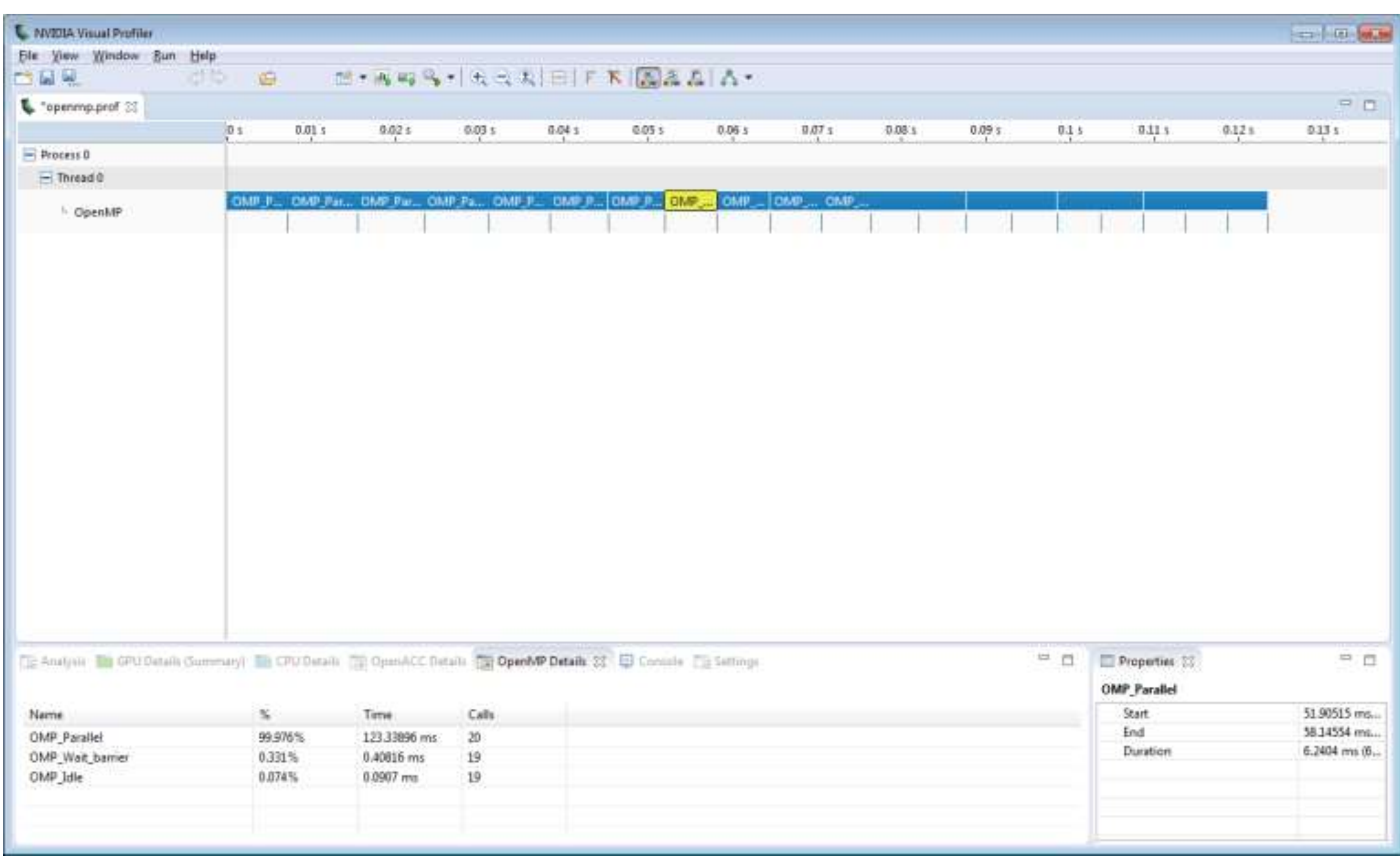
nvprof option openmp-profiling to enable/disable the OpenMP profiling, default on

```
$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

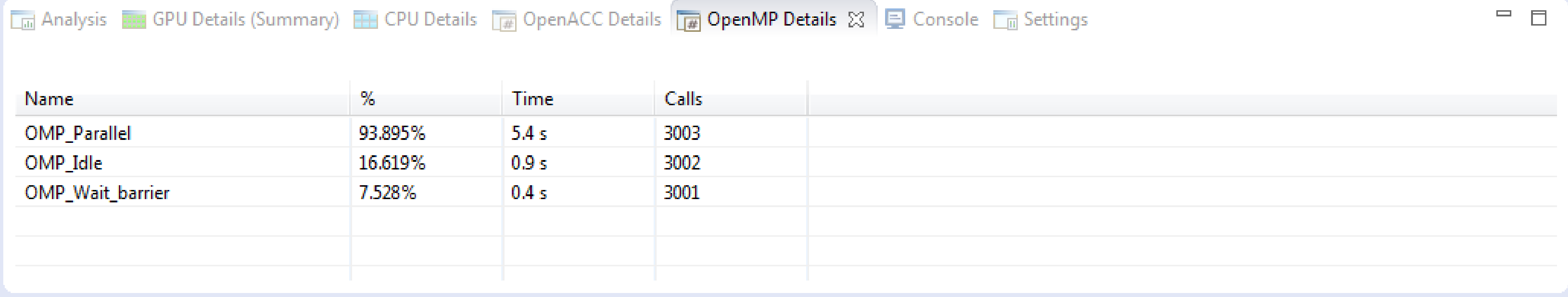
OPENMP PROFILING IN VISUAL PROFILER



NVIDIA

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

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

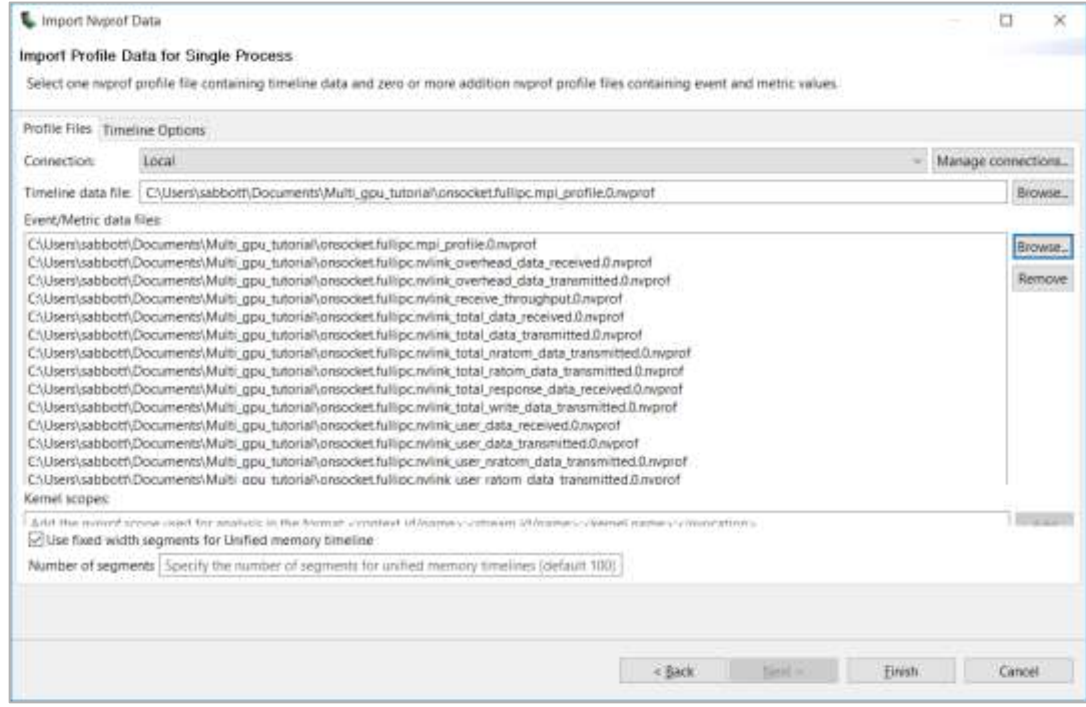
Use `--query-metrics` and `--query-events` for full list of metrics (-m) or events (-e)

Combine with an MPI annotated timeline file for full picture

NVIDIA

PROFILING NVLINK USAGE

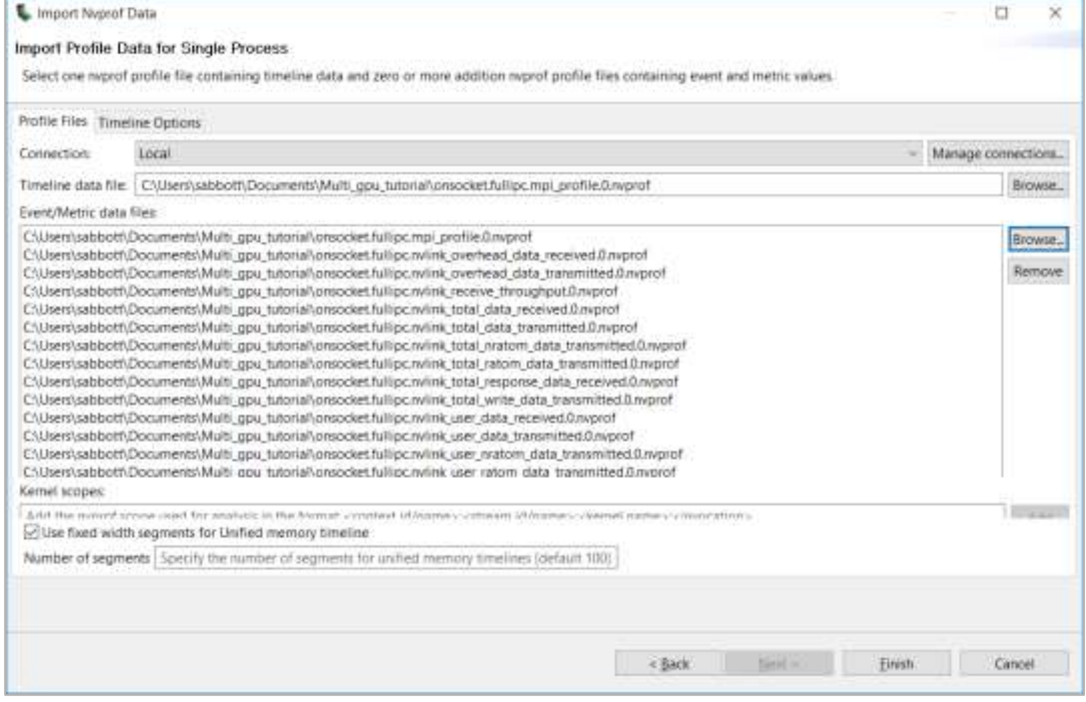
Using `nvprof`+NVVP

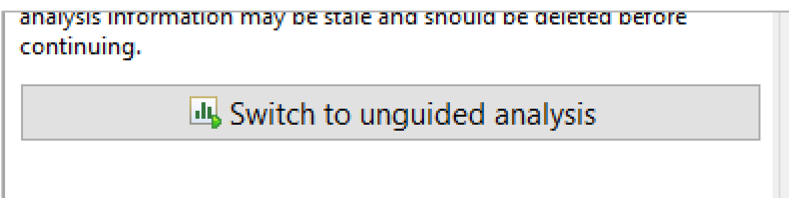


NVIDIA

PROFILING NVLINK USAGE

Using `nvprof`+NVVP

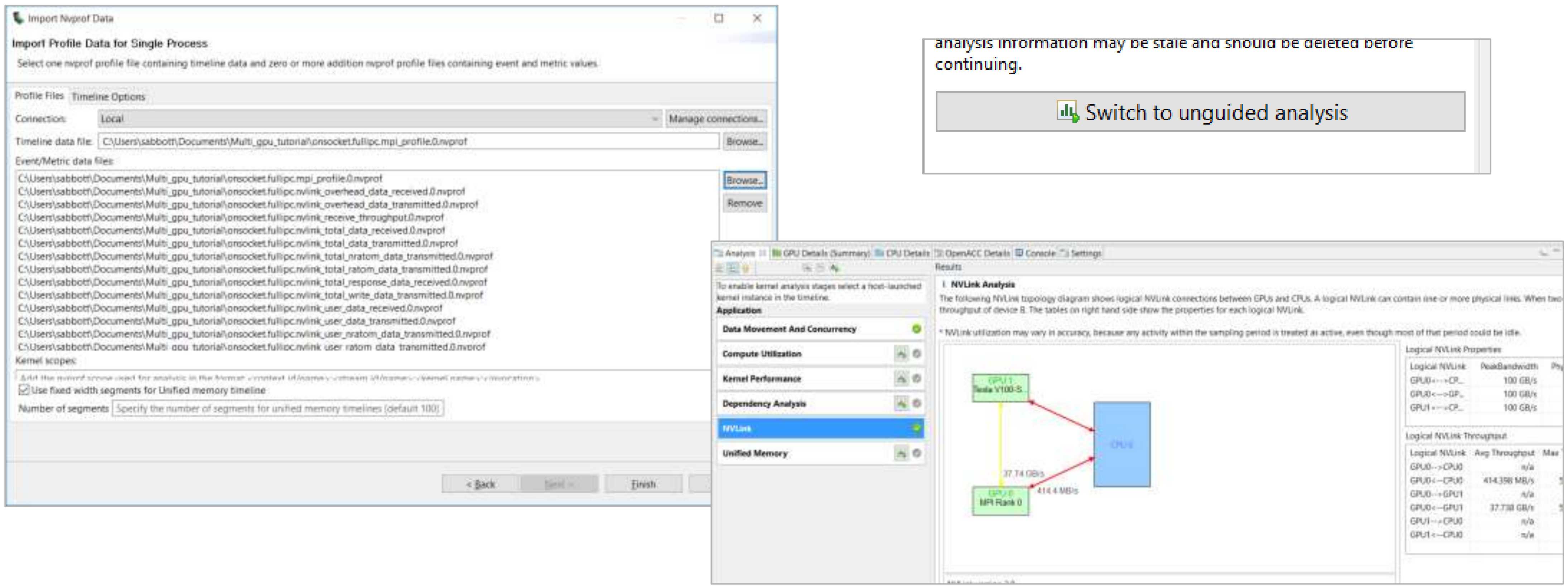




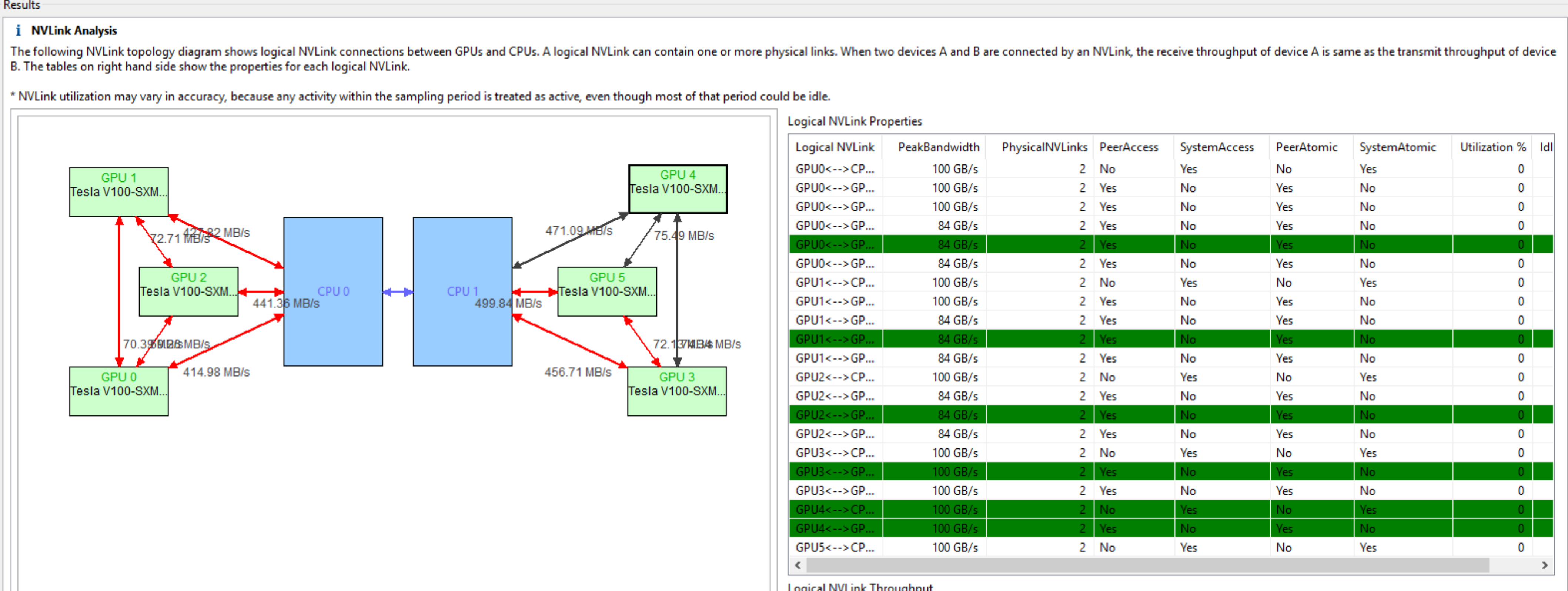
NVIDIA.

PROFILING NVLINK USAGE

Using `nvprof`+NVVP



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.

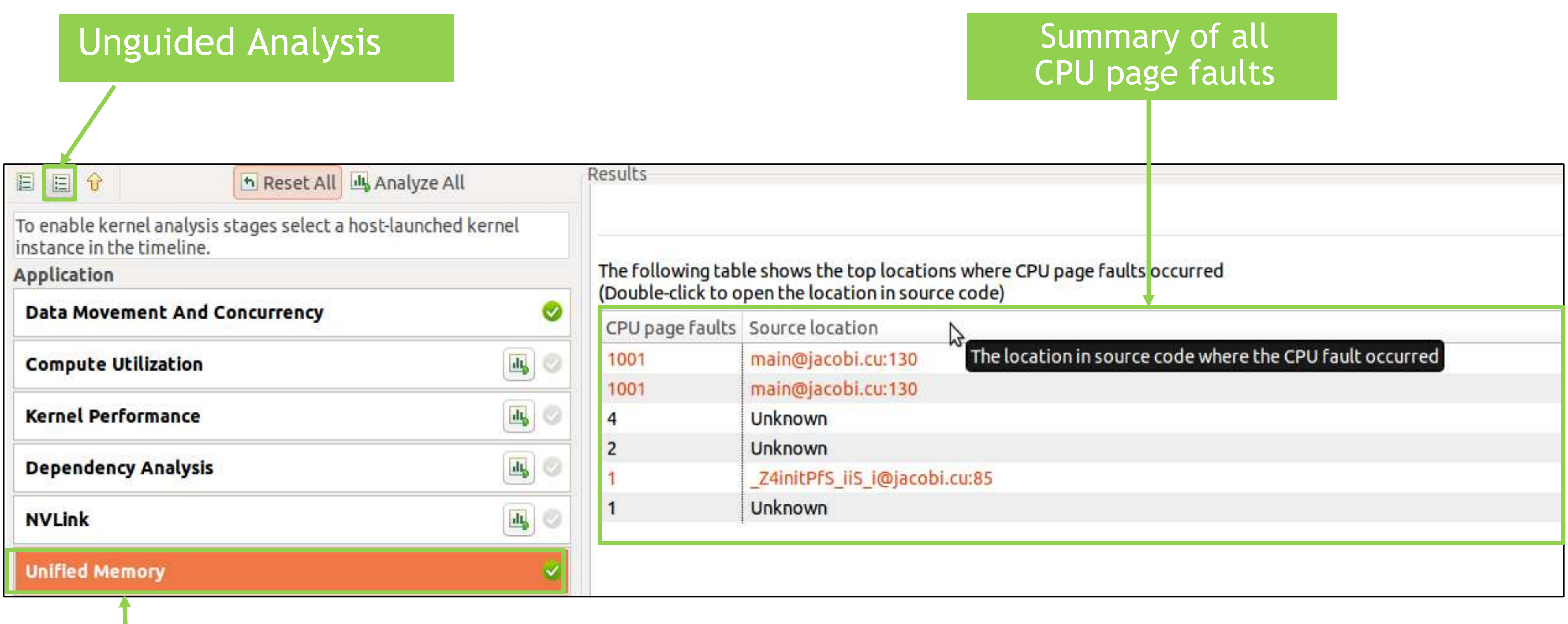
Logical NVLink Properties

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

NVIDIA

CPU PAGE FAULT SOURCE CORRELATION

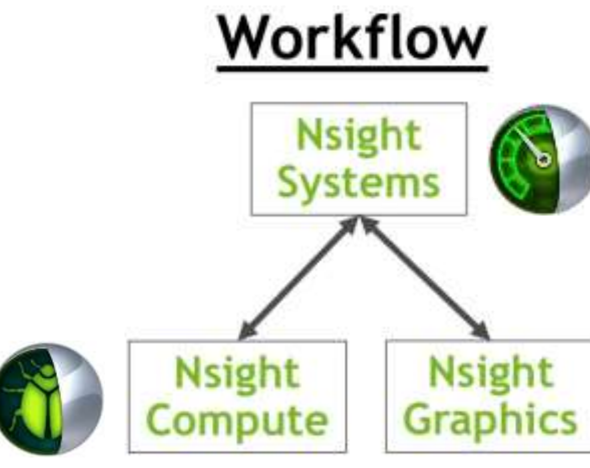


NVIDIA

INTRODUCTION TO
NSIGHT SYSTEMS

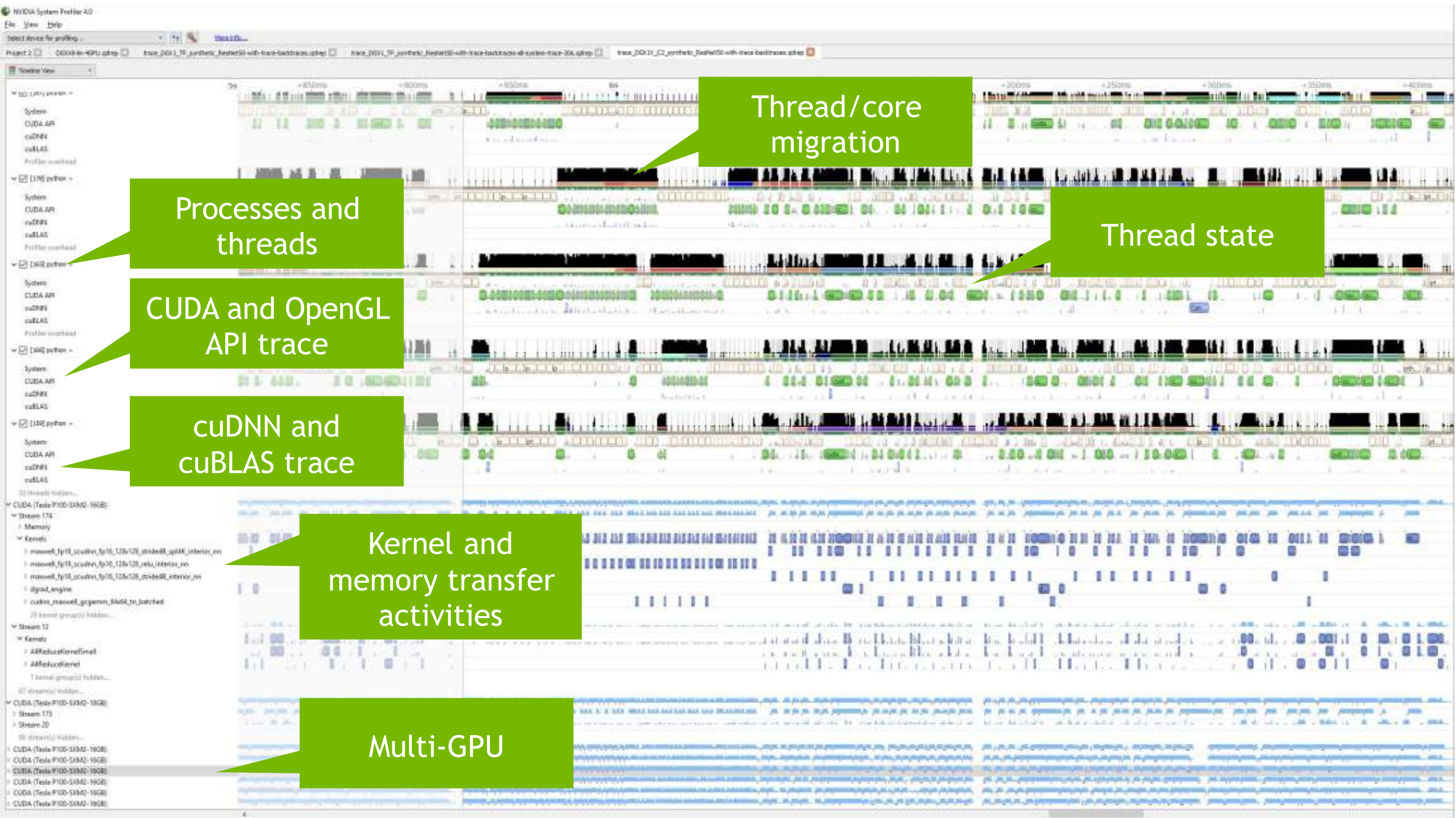
NSIGHT PRODUCT FAMILY

- Standalone Performance Tools
  - Nsight Systems system-wide application algorithm tuning
  - Nsight Compute Debug/optimize specific CUDA kernel
  - Nsight Graphics Debug/optimize specific graphics
- IDE plugins
  - Nsight Eclipse Edicion/Visual Studio editor, debugger, some perf analysis

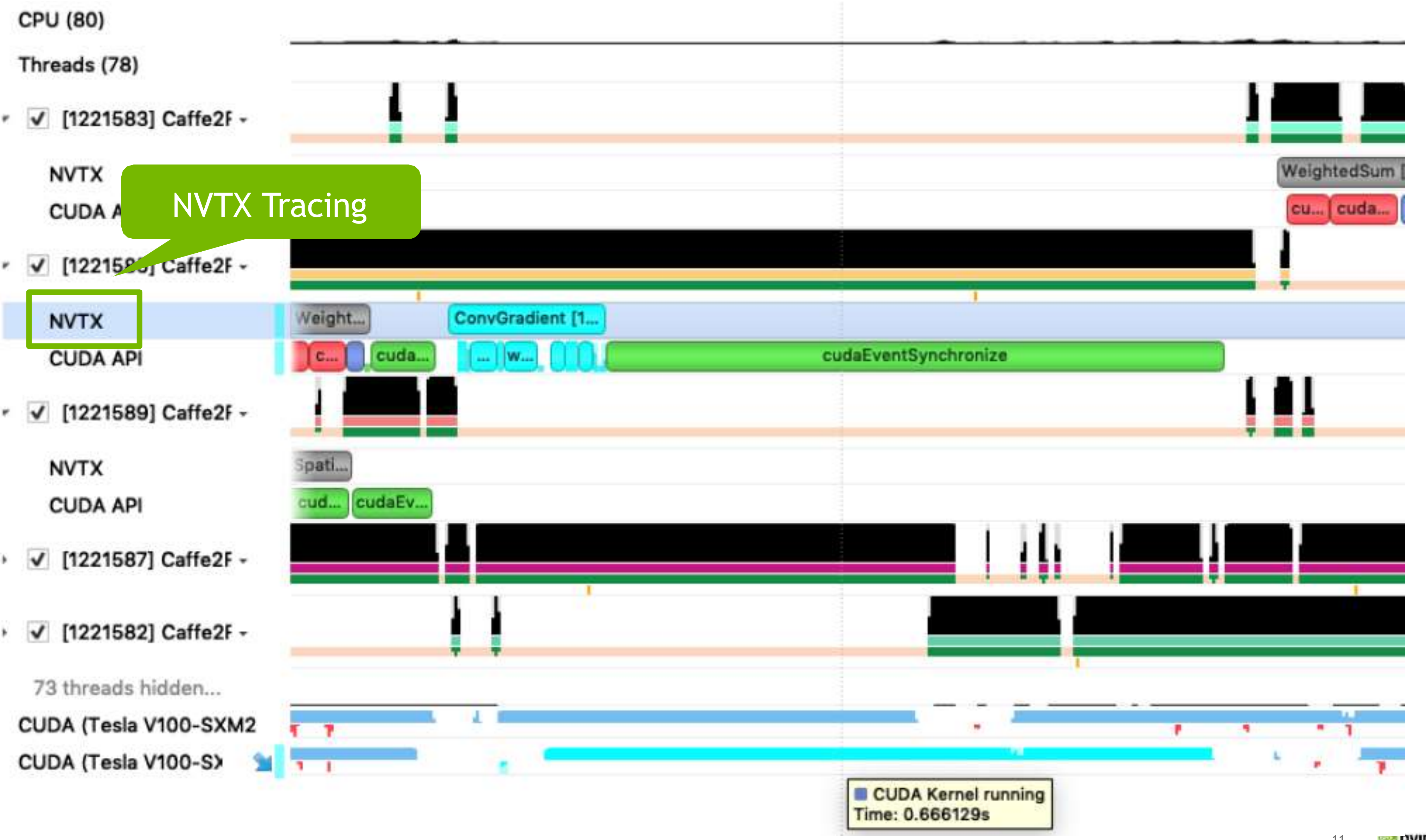


NSIGHT SYSTEMS
Overview

- Profile System-wide application
  - Multi-process tree, GPU workload trace, etc
- Investigate your workload across multiple CPUs and GPUs
  - CPU algorithms, utilization, and thread states
  - GPU streams kernels, memory transfers, etc
  - NVTX, CUDA & Library API, etc
- Ready for Big Data
  - docker, user privilege (linux), cli, etc



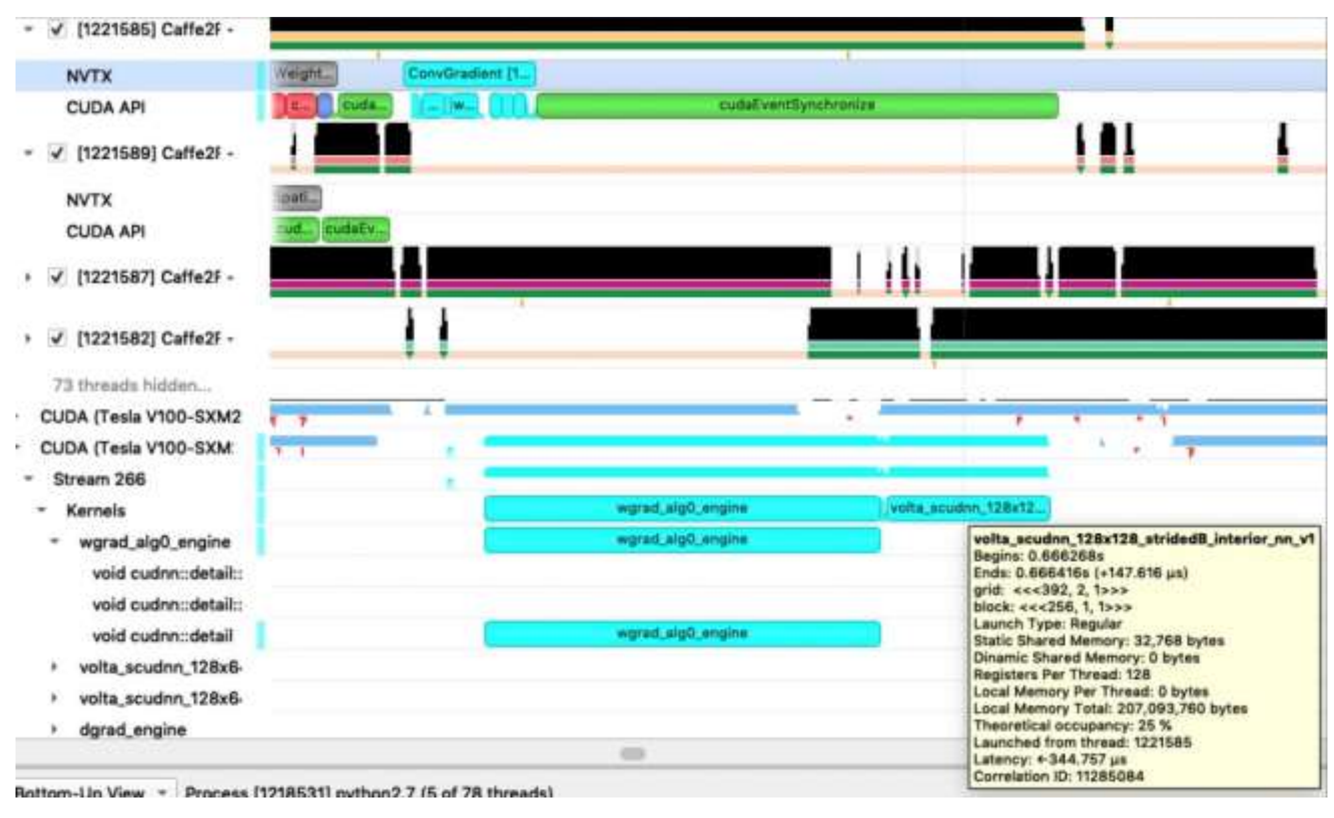
Thread/core migration
Processes and threads
Thread state
CUDA and OpenGL API trace
cuDNN and cuBLAS trace
Kernel and memory transfer activities
Multi-GPU



CPU (80)
Threads (78)
[1221583] Caffe2F
NVTX
WeightedSum
CUDA A
cu...
cuda...
NVTX Tracing
Caffe2F
NVTX
Weight...
ConvGradient [1...
CUDA API
c...
cuda...
w...
cudaEventSynchronize
[1221589] Caffe2F
NVTX
Spati...
CUDA API
cud...
cudaEv...
[1221587] Caffe2F
[1221582] Caffe2F
73 threads hidden...
CUDA (Tesla V100-SXM2
CUDA (Tesla V100-SX
CUDA Kernel running
Time: 0.666129s

TRANSITIONING TO PROFILE A KERNEL

Dive into kernel analysis



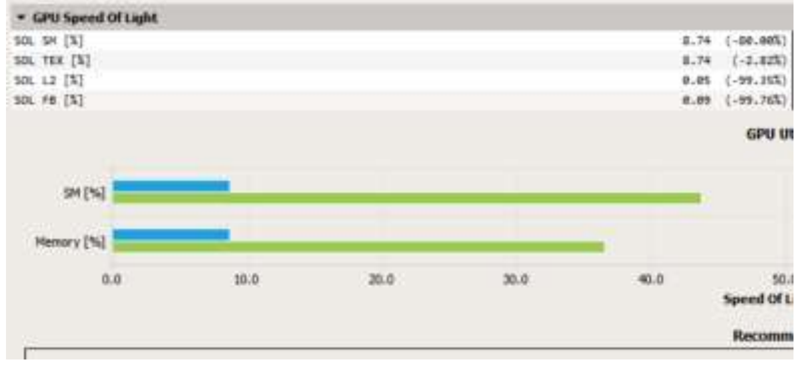
NVIDIA

NVIDIA NSIGHT COMPUTE

Next Generation Kernel Profiler

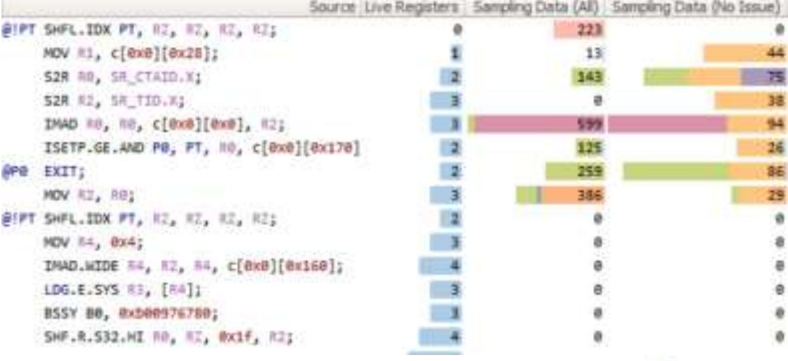
- Interactive CUDA API debugging and kernel profiling
- Fast Data Collection
- Graphical and multiple kernel comparison reports
- Improved Workflow and Fully Customizable (Baselining, Programmable UI/Rules)
- Command Line, Standalone, IDE Integration
- Platform Support
  - OS: Linux(x86,ARM), Windows, OSX (host only)
  - GPUs: Pascal, Volta, Turing

Kernel Profile Comparisons with Baseline



Metric Data

Source Correlation

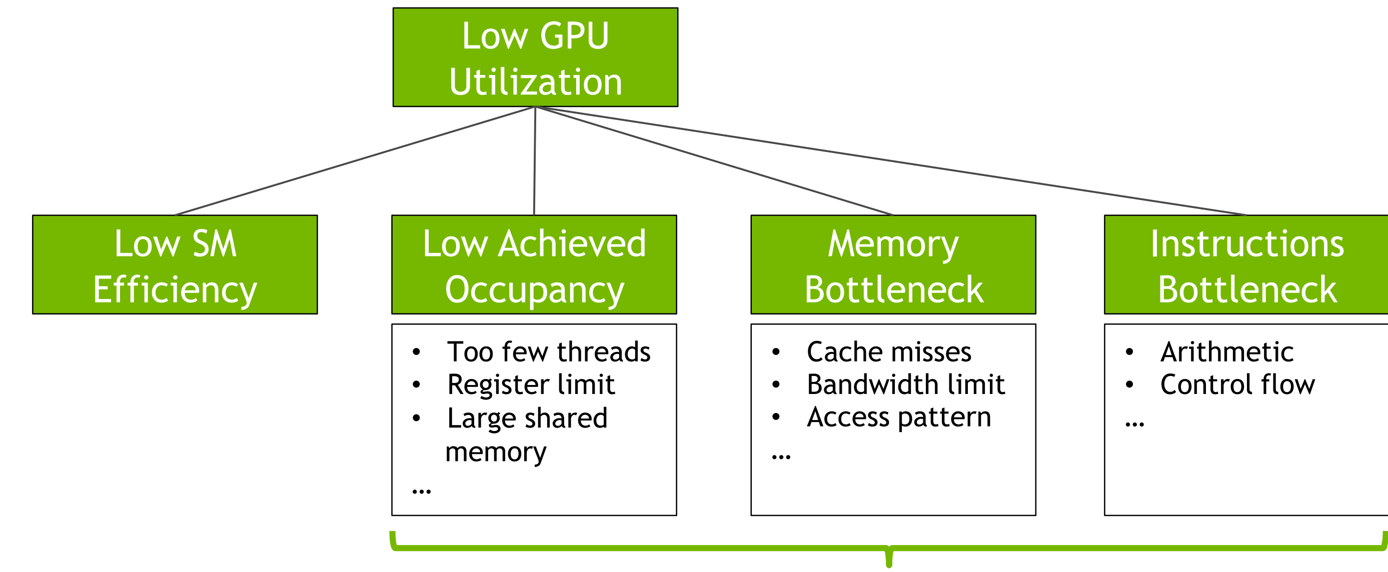


PROFILING GPU APPLICATION

Focusing GPU Computing

How to measure

- GPU Profiling
- CPU/GPU Tracing
- Application Tracing



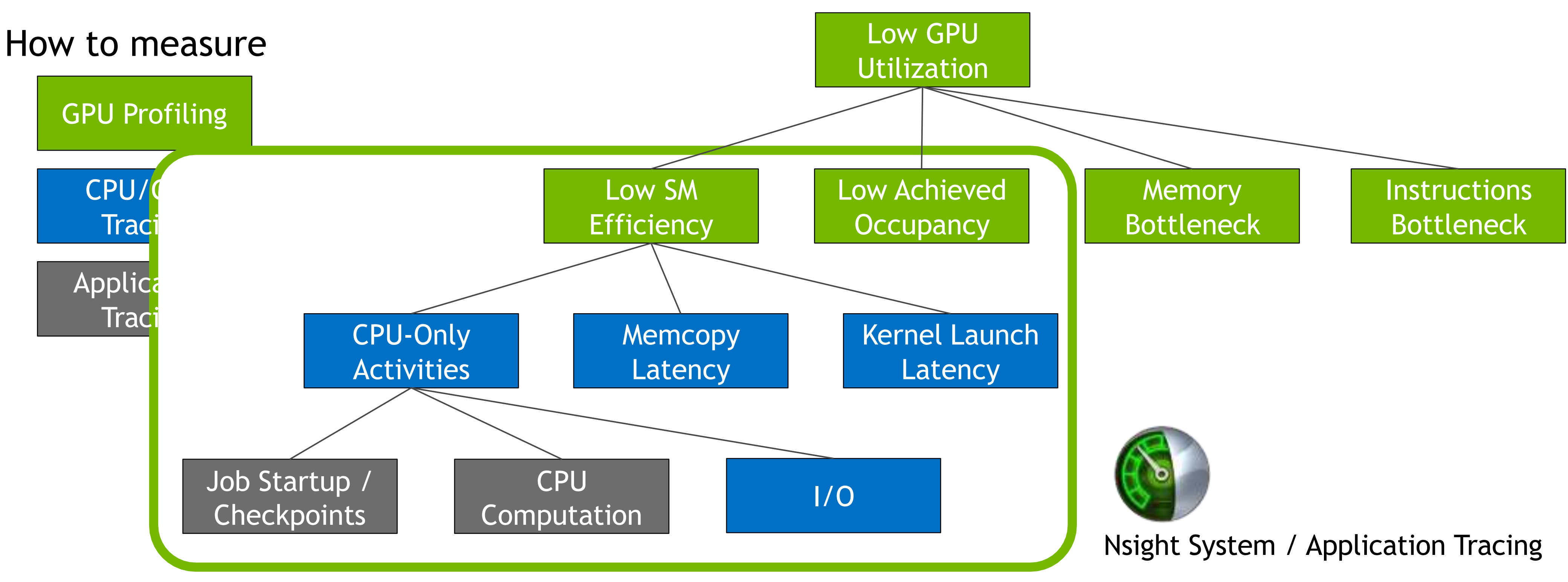
NVIDIA (Visual) Profiler / Nsight Compute

NVIDIA Supports them with cuDNN, cuBLAS, and so on

NVIDIA

PROFILING GPU APPLICATION

Focusing System Operation



NVIDIA

HOW TO USE

NSIGHT SYSTEMS PROFILE

Profile with CLI

APIs to be traced

```
$ nsys profile –t cuda,osrt,nvtx,cudnn,cublas \
     –o baseline.qdstrm –w true python main.py
```

Name of output file

Show output on console

Application command

Automatic conversion of .qdstrm temp results file to .qdrep format if converter utility is available.

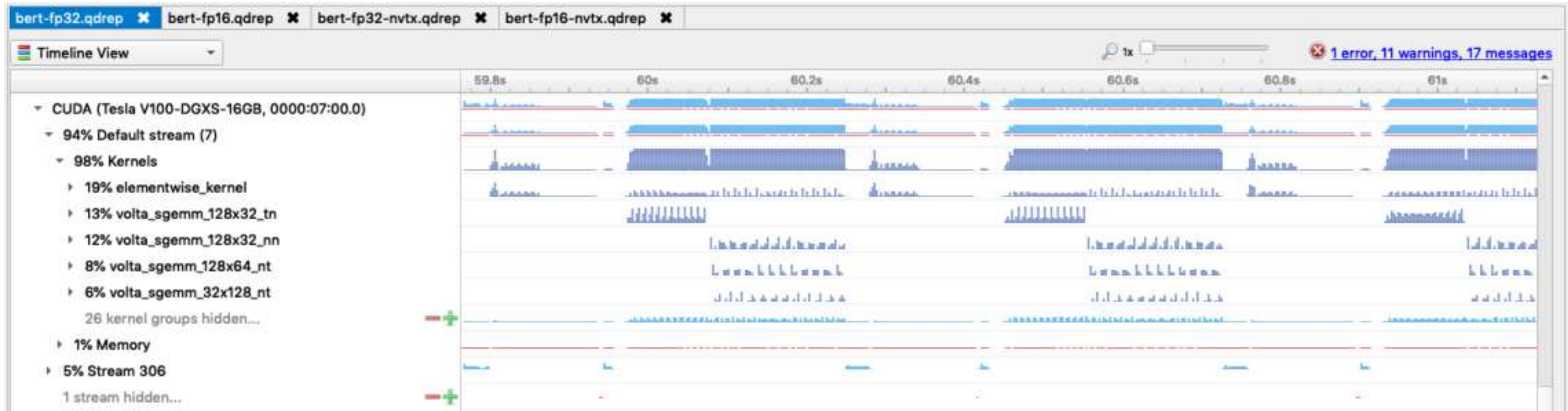
cuda - GPU kernel
osrt - OS runtime
nvtx - NVIDIA Tools Extension
cudnn - CUDA Deep NN library
cublas - CUDA BLAS library

https://docs.nvidia.com/nsight-systems/#nsight_systems/2019.3.6-x86/06-cli-profiling.htm

NVIDIA

NSIGHT SYSTEMS PROFILE

No NVTX



- Difficult to understand → no useful

NVIDIA

NVTX (NVIDIA TOOLS EXTENSION)

NVTX ANNOTATIONS

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):


        data, target = data.to(device), target.to(device)


        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
```

https://pytorch.org/docs/stable/_modules/torch/cuda/nvtx.html

NVIDIA

NVTX ANNOTATIONS

NVTX in PyTorch

```
import torch.cuda.nvtx as nvtx
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        nvtx.range_push("Batch " + str(batch_idx))
        nvtx.range_push("Copy to device")
        data, target = data.to(device), target.to(device)
        nvtx.range_pop()
        nvtx.range_push("Forward pass")
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        nvtx.range_pop()
        nvtx.range_pop()
```

Batch %d

copy to device

Forward pass

https://pytorch.org/docs/stable/_modules/torch/cuda/nvtx.html

NVIDIA

NVTX ANNOTATIONS

NVTX using cupy pakage

```
from cupy.cuda import nvtx
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        nvtx.RangePush("Batch " + str(batch_idx))
        nvtx.RangePush("Copy to device")
        data, target = data.to(device), target.to(device)
        nvtx.RangePop()
        nvtx.RangePush("Forward pass")
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        nvtx.RangePop()
        nvtx.RangePop()
```

Batch %d

copy to device

Forward pass

https://docs-cupy.chainer.org/en/stable/

NVIDIA.

NSIGHT SYSTEM PROFILE

NVTX range marker tip

- NVTX for data loading (data augmetnation)

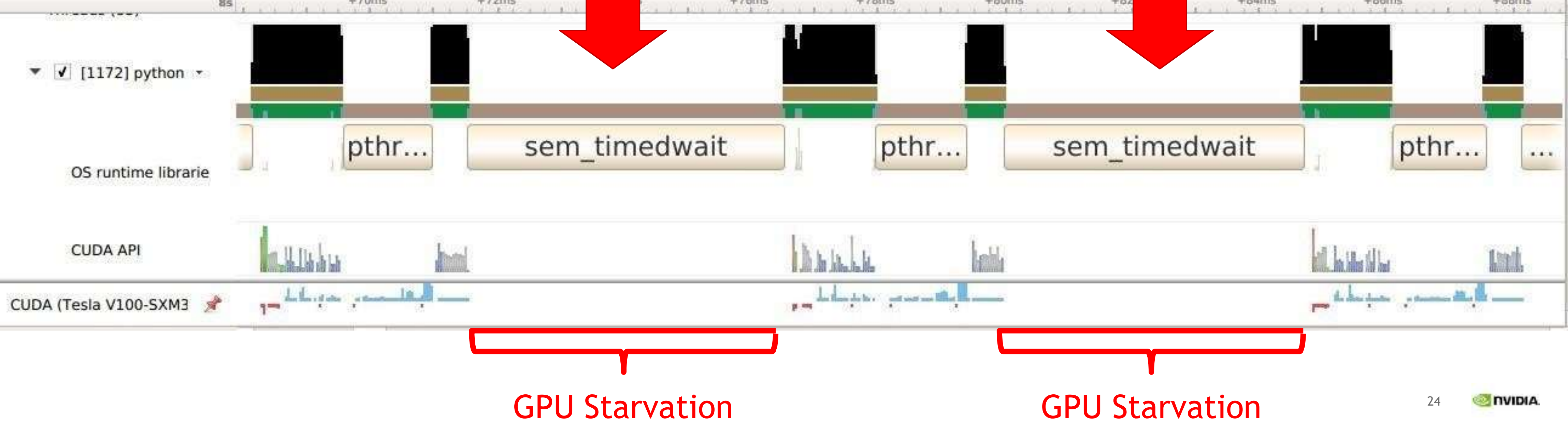
```
for batch_idx, (data,target) in enumerate(train_loader):
```

```
nvtx.range_push('Data loading')
(data,target) = train_loader.next()
nvtx.range_pop()
```

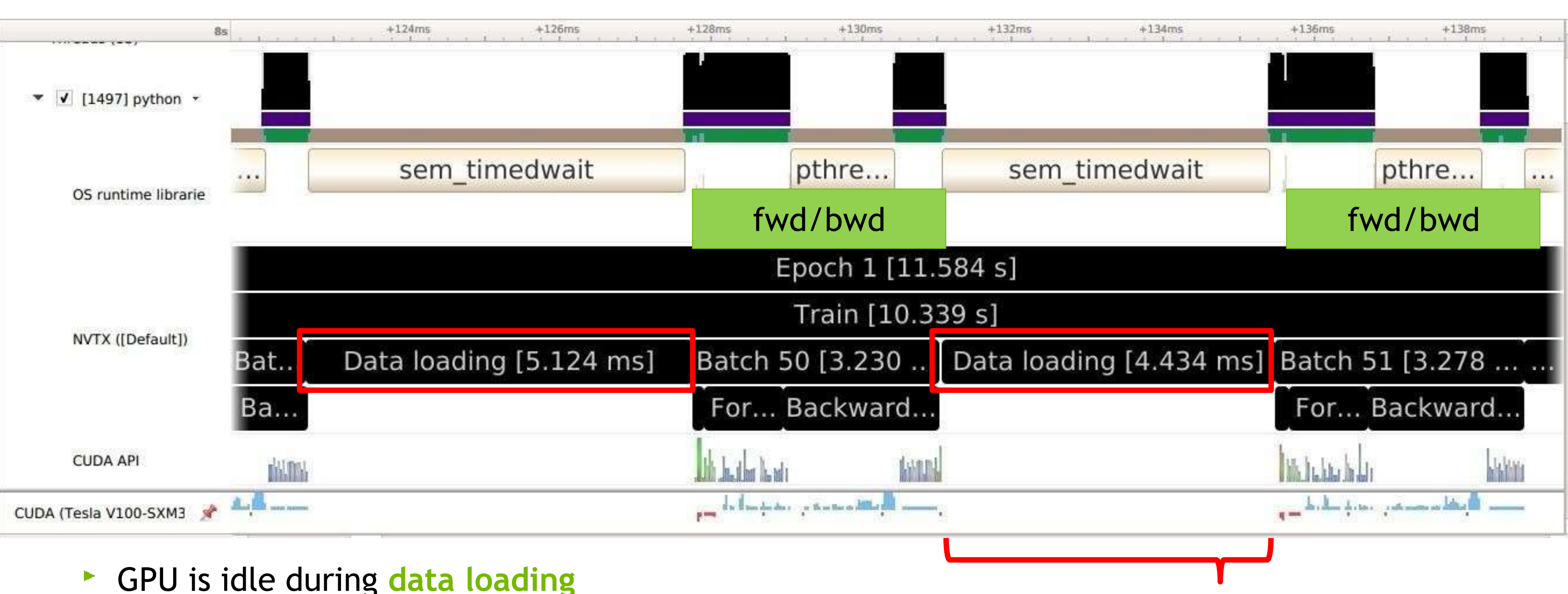

NVIDIA

BASELINE PROFILE

- MNIST Training: 89 sec, <5% utilization
- CPU waits on a semaphore and starves the GPU!



BASELINE PROFILE (WITH NVTX)



- GPU is idle during data loading
- Data is loaded using a single thread. This starves the GPU!

OPTIMIZE SOURCE CODE

- Data loader was configured to use 1 worker thread

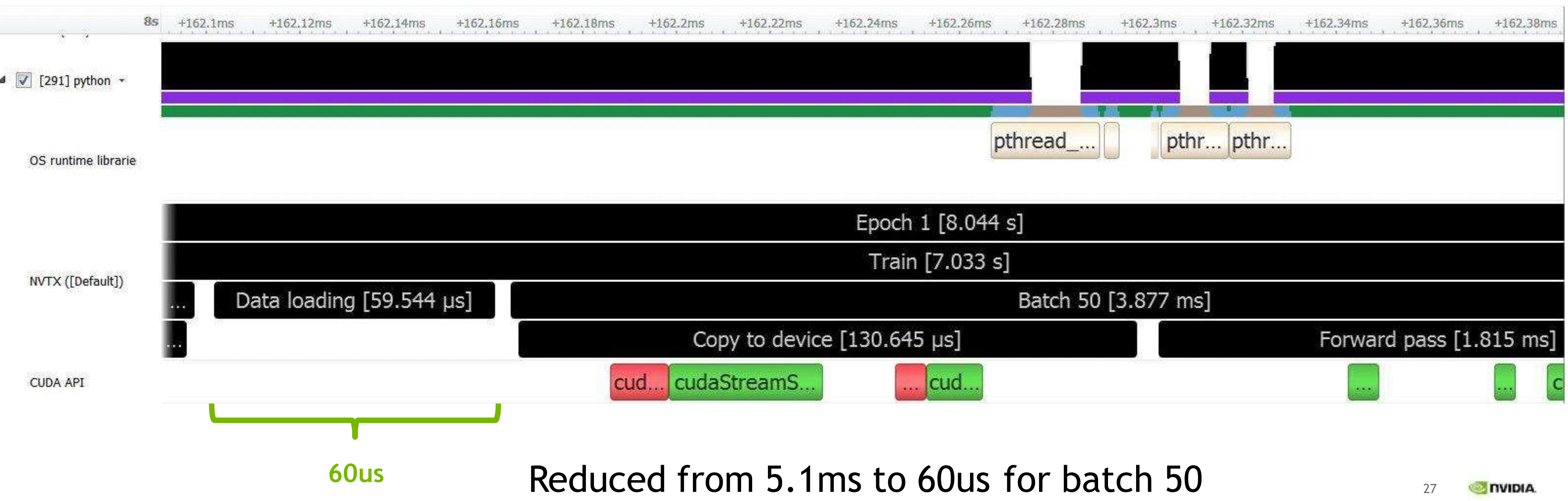
```
kwargs = {'num_workers': 1, 'pin_memory'  True  if use_cuda else {}
```

- Let’s switch to using 8 worker threads:

```
kwargs = {'num_workers’: 8, 'pin_memory'  True  if use_cuda else {}
```

AFTER OPTIMIZATION

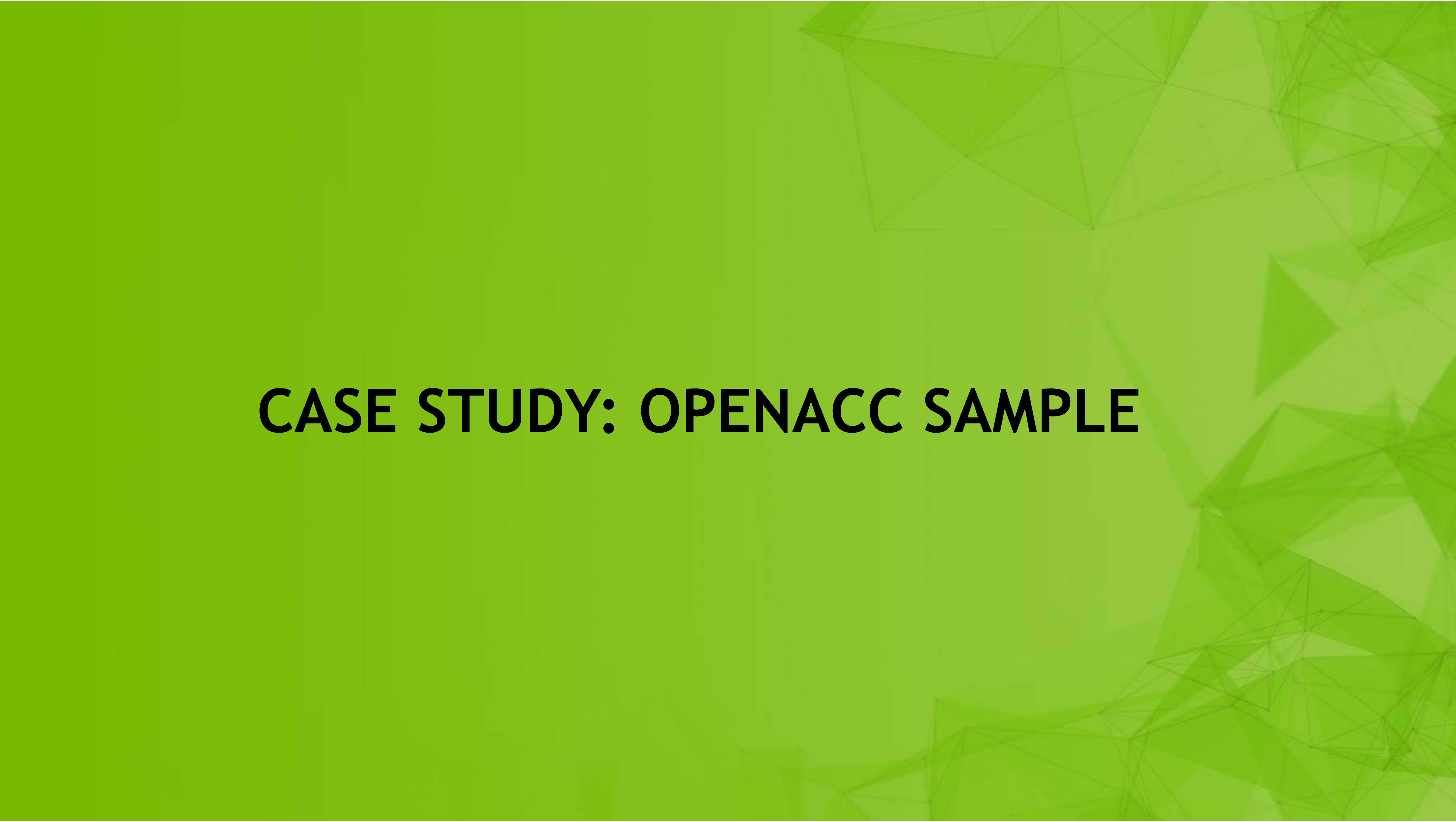
- Time for data loading reduced for each bath



Reduced from 5.1ms to 60us for batch 50

NVIDIA

NVIDIA


CASE STUDY: OPENACC SAMPLE

OPENACC SAMPLE

- Sample from https://devblogs.nvidia.com/getting-started-openacc
- Solves 2-D Laplace equation with iterative Jacobi solver
- Each iteration
  1. A stencil calculation
  2. Update the matrix
  3. Check if error tolerance is met. If not, go to step 1.

SAMPLE (CPU VERSION)

```
while ( error > tol && iter < iter_max ) {
  error = 0.0;
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] +
                            A[j-1][i] + A[j+1][i]);
      error = fmax( error, fabs(Anew[j][i] - A[j][i]));
    }
  }

  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
  iter++;
}
```

Convergence loop

Stencil calculation

Update matrix

NVIDIA.

OPENACC SAMPLE

```
while ( error > tol && iter < iter_max ) {
  error = 0.0;
  #pragma acc kernels
  {
    for( int j = 1; j < n-1; j++) {
      for( int i = 1; i < m-1; i++ ) {
        Anew[j][i] = …
        error = fmax( error, fabs(Anew[j][i] - A[j][i]));
      }
    }
    for( int j = 1; j < n-1; j++) {
      for( int i = 1; i < m-1; i++ ) {
        A[j][i] = Anew[j][i];
      }
    }
  }
  iter++;
}
```

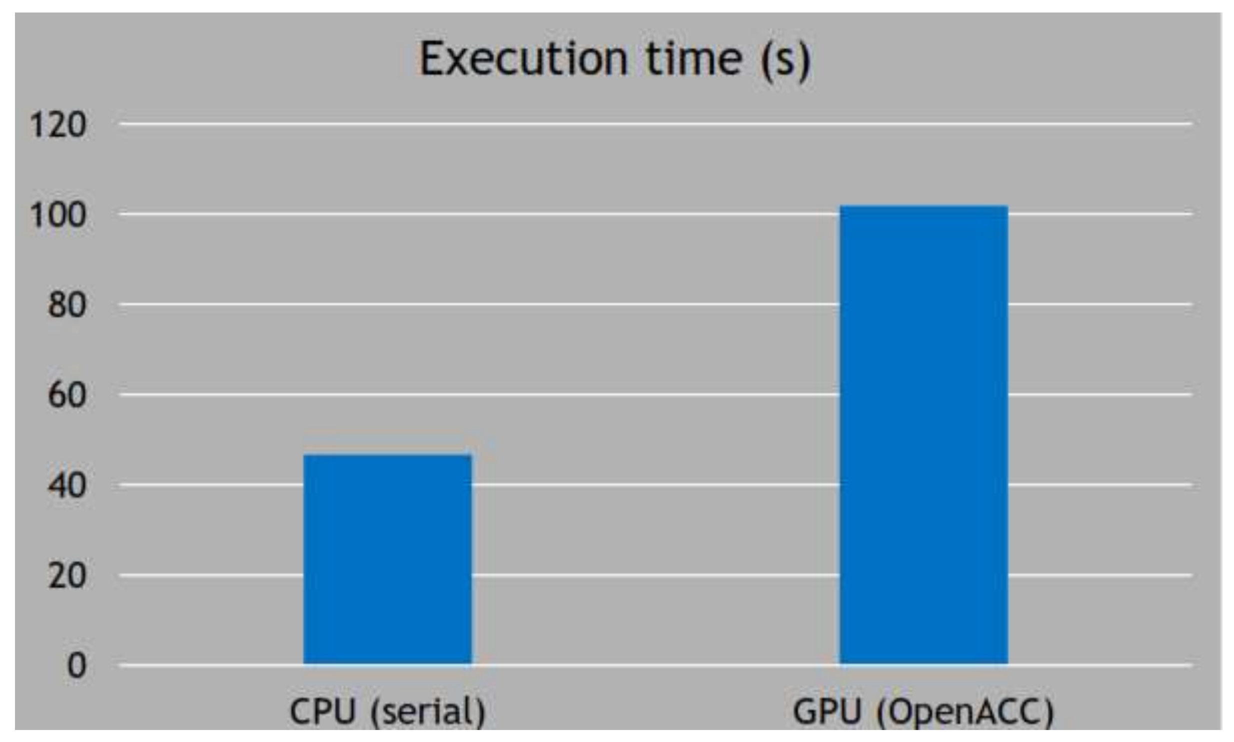
Convergence loop

Stencil calculation

Update matrix

NVIDIA.

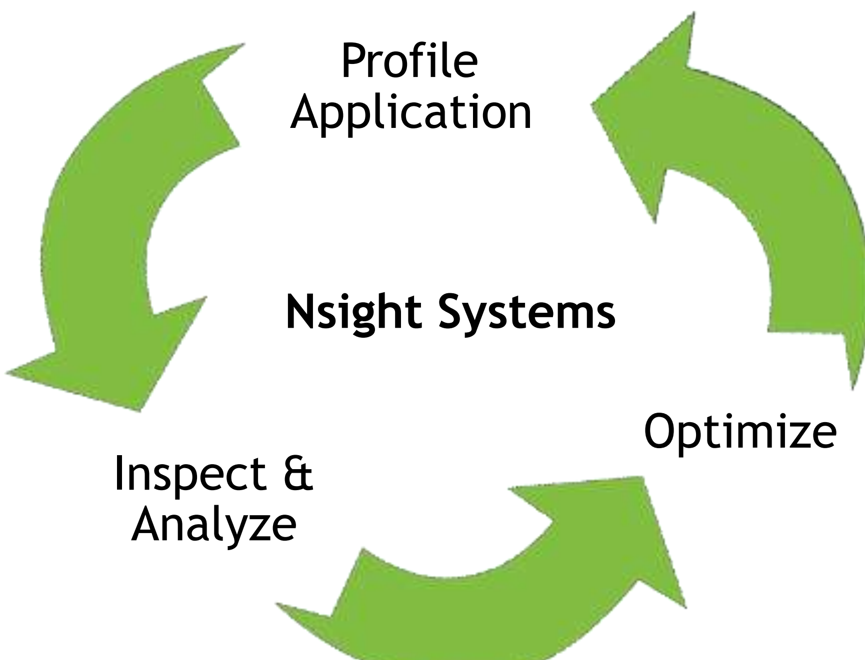
PERFORMANCE



Execution time for 1000 iterations on a system with:
Intel® Core™ i7-6850K CPU
NVIDIA TITAN X (Pascal) GPU

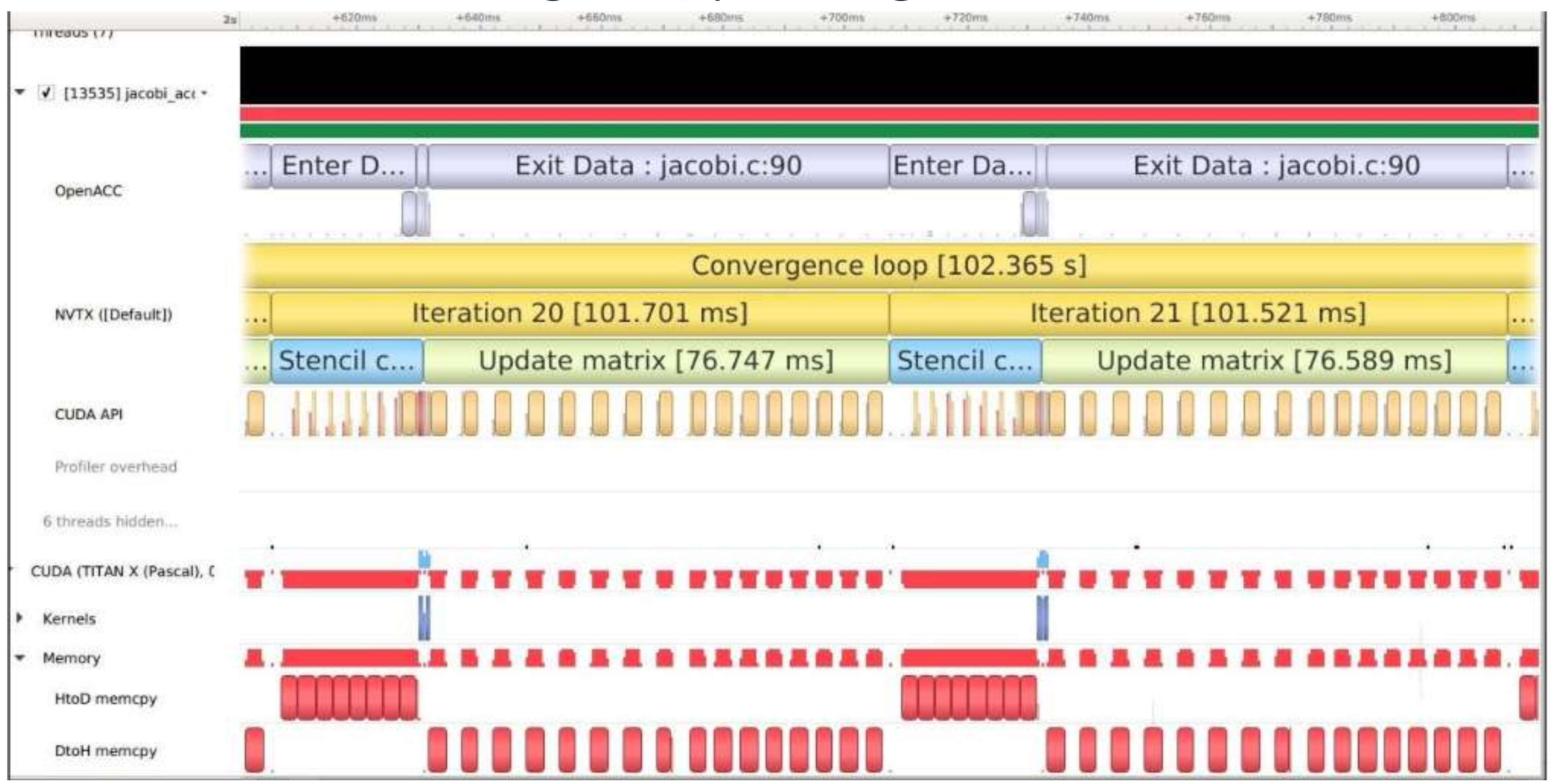
That is unexpected!

OPTIMIZATION WORKFLOW



NVIDIA.

BASELINE PROFILE



Excessive data copies slowing down GPU

OPENACC SAMPLE

```
while ( error > tol && iter < iter_max ) {
  error = 0.0;
  #pragma acc kernels
  {
    for( int j = 1; j < n-1; j++) {
      for( int i = 1; i < m-1; i++ ) {
        Anew[j][i] = …
        error = fmax( error, fabs(Anew[j][i] - A[j][i]));
      }
    }
    for( int j = 1; j < n-1; j++) {
      for( int i = 1; i < m-1; i++ ) {
        A[j][i] = Anew[j][i];
      }
    }
  }
  iter++;
}
```

Convergence loop

Stencil calculation

Update matrix

NVIDIA.

OPENACC SAMPLE

```
#pragma acc data copy(A) create(Anew)
while ( error > tol && iter < iter_max ) {
  error = 0.0;
  #pragma acc kernels
  {
    for( int j = 1; j < n-1; j++) {
      for( int i = 1; i < m-1; i++ ) {
        Anew[j][i] = …
        error = fmax( error, fabs(Anew[j][i] - A[j][i]));
      }
    }
    for( int j = 1; j < n-1; j++) {
      for( int i = 1; i < m-1; i++ ) {
        A[j][i] = Anew[j][i];
      }
    }
  }
  iter++;
}
```

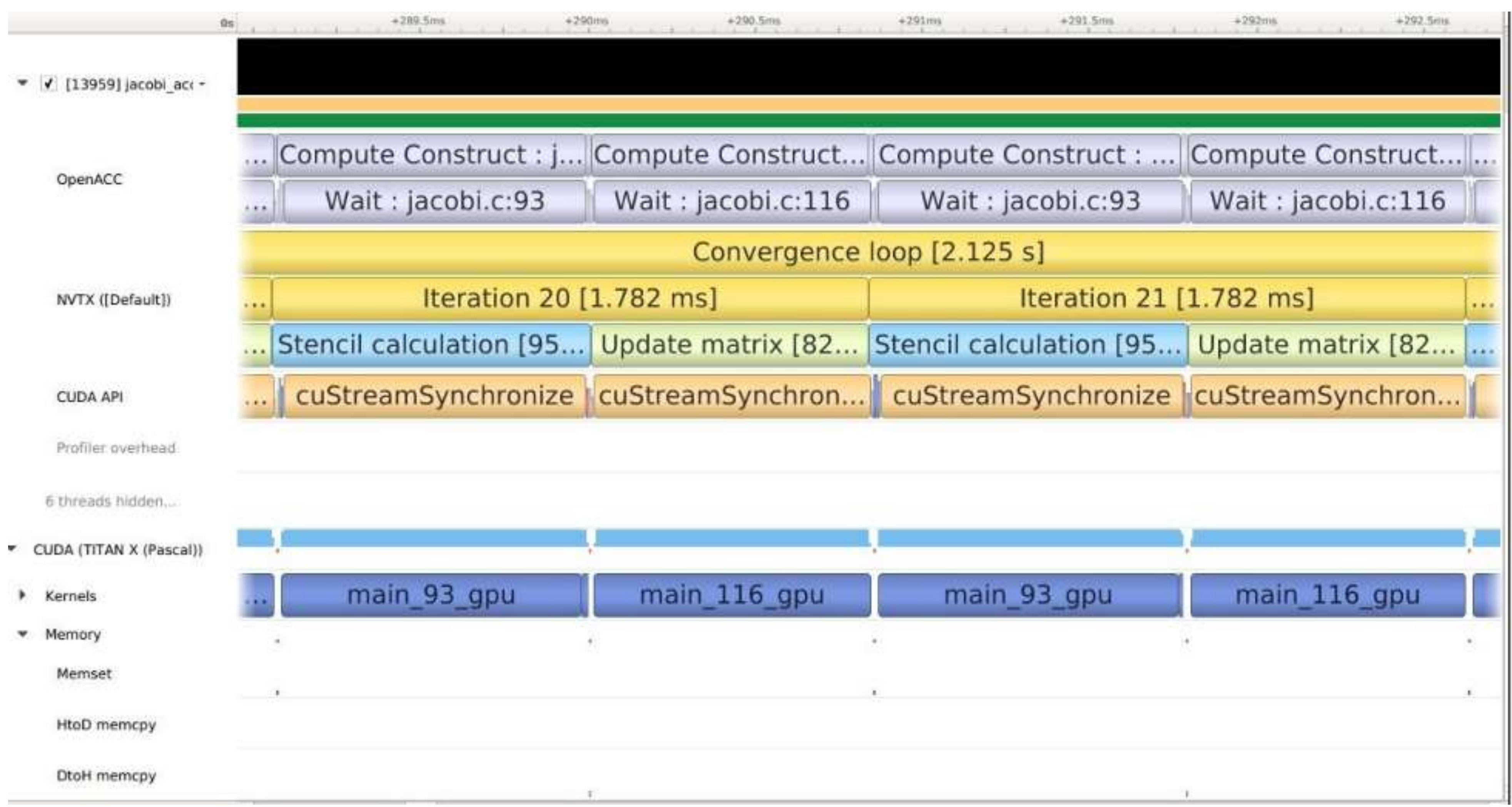
Convergence loop

Stencil calculation

Update matrix

NVIDIA.

AFTER OPTIMIZATION

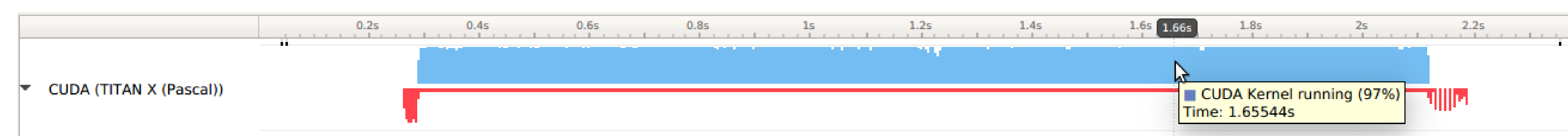


Excessive data copies eliminated

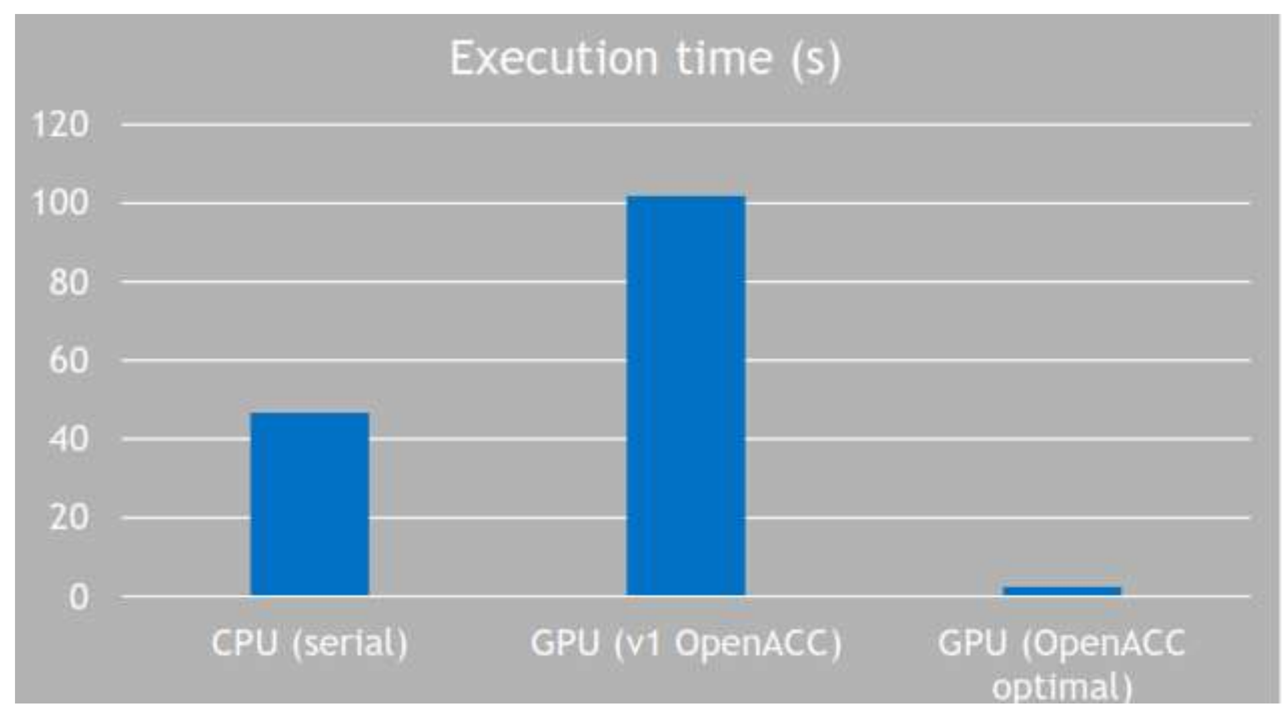
NVIDIA

AFTER OPTIMIZATION

CUDA kernel coverage on GPU is ~97%



AFTER OPTIMIZATION



Execution time for 1000 iterations on a system with:
Intel® Core™ i7-6850K CPU
NVIDIA TITAN X (Pascal) GPU

44x speedup!

https://www.openacc.org/resources for more best practices

COMMON OPTIMIZATION OPPORTUNITIES

- **CPU**
  - Thread synchronization
  - Algorithm bottlenecks starve the GPUs (case study 1)
- **Multi GPU**
  - Communication between GPUs
  - Lack of Stream Overlap in memory management, kernel execution
- **Single GPU**
  - Memory operations – blocking, serial, unnecessary (case study 2)
  - Too much synchronization - device, context, stream, default stream, implicit
  - CPU GPU Overlap – avoid excessive communication