NVIDIA NSIGHT SYSTEMS

Sneha Latha Kottapalli
Senior Systems Software Engineer, NVIDIA

November 7, 2018

NVIDIA

ABOUT THIS WEBINAR

Goal

Introduce viewers to performance optimization using NVIDIA Nsight Systems

Target Audience

Beginner and advanced users of GPU

Software

NVIDIA Nsight Systems 2018.3.1.

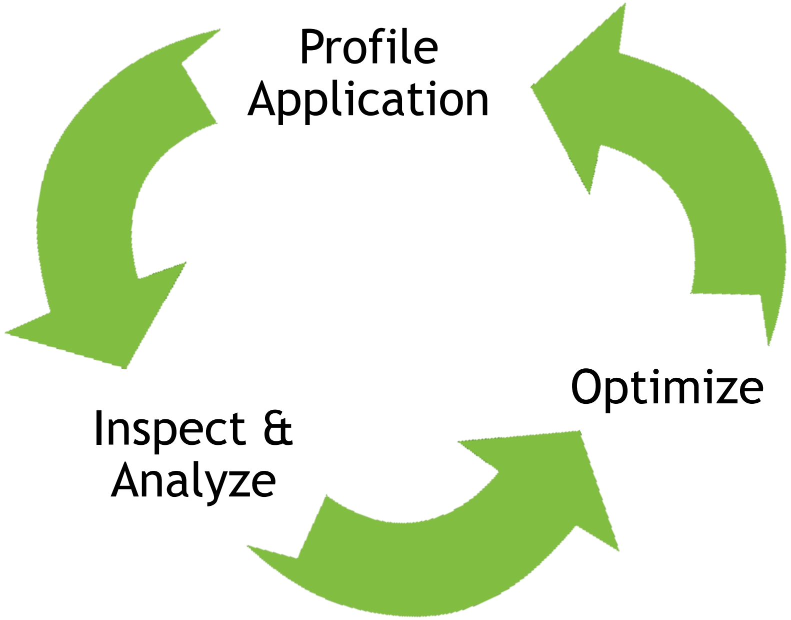
Download from http://developer.nvidia.com/nsight-systems

AGENDA

- Introduction to Nsight Systems
- Features
- Report navigation demo
- Case studies
- Common optimization opportunities
- Tools comparison
- Beyond Nsight Systems

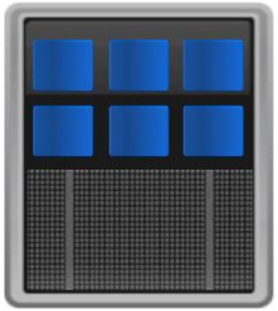
INTRODUCTION

TYPICAL OPTIMIZATION WORKFLOW



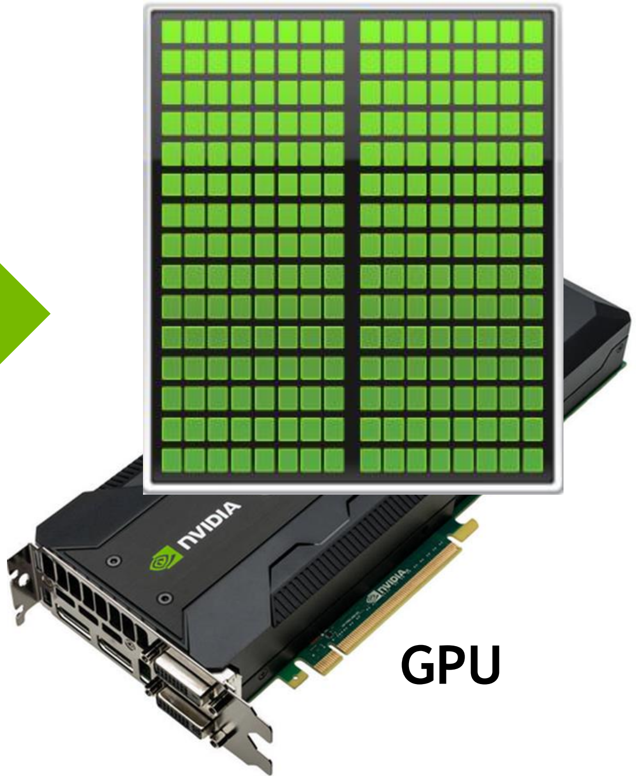
Iterative process continues until desired performance is achieved

ACCELERATED PERFORMANCE WITH GPU



CPU

Compute-intensive code!



GPU

INTRODUCING NSIGHT SYSTEMS

System-wide Performance Analysis Tool

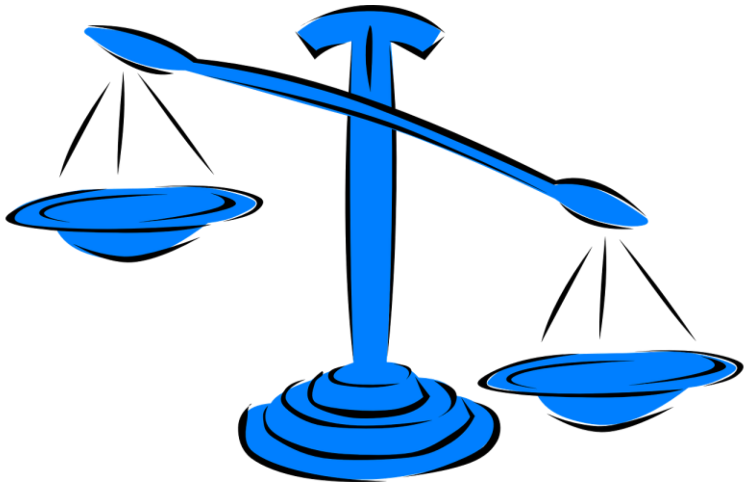
Focus on the application's algorithm – a unique perspective

Scale your application efficiently across any number of CPUs & GPUs

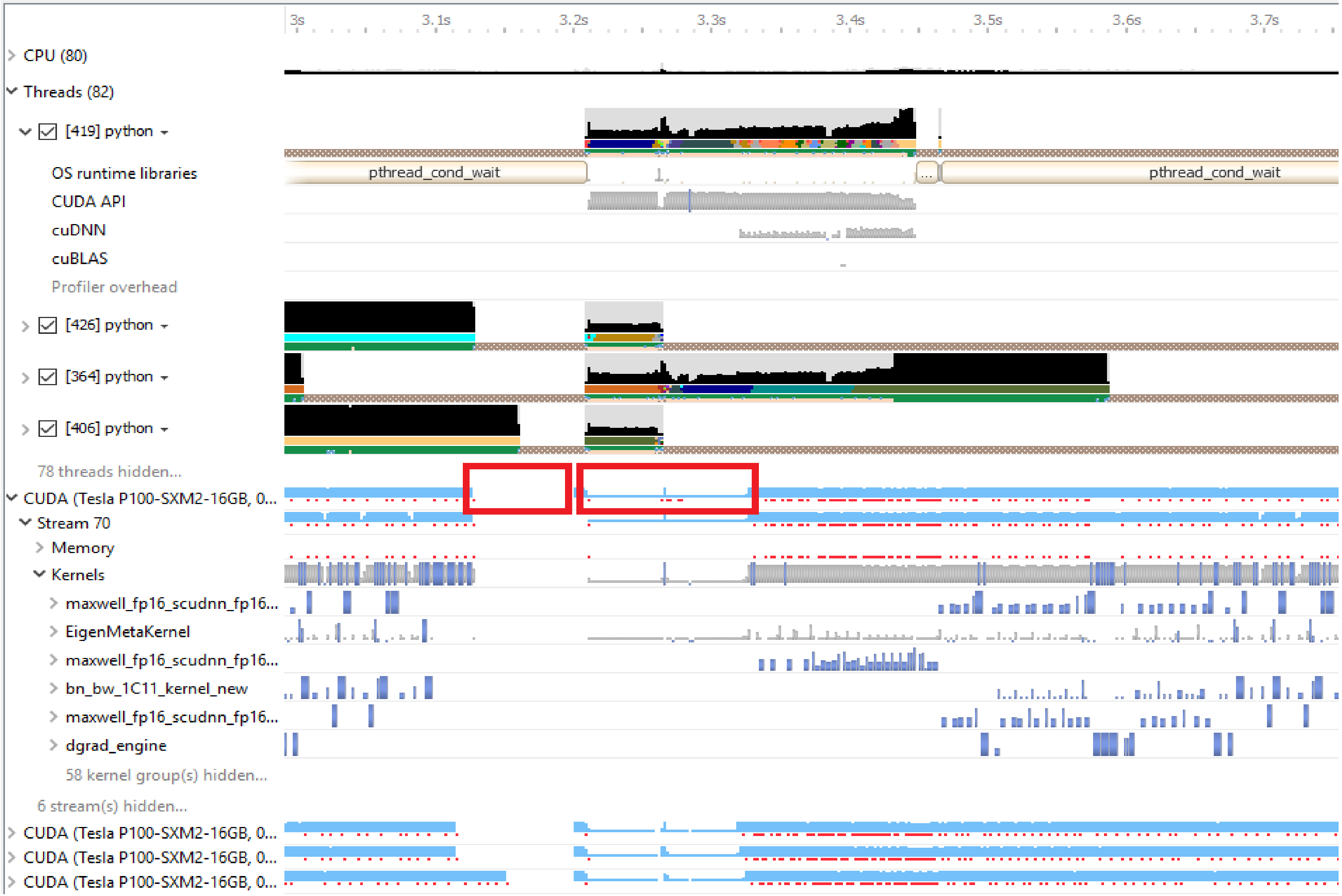
NSIGHT SYSTEMS

Maximize your GPU investment

- Balance your workload across multiple CPUs and GPUs
- Find the right optimization opportunities
- Improve application’s performance
- Support for Linux & Windows



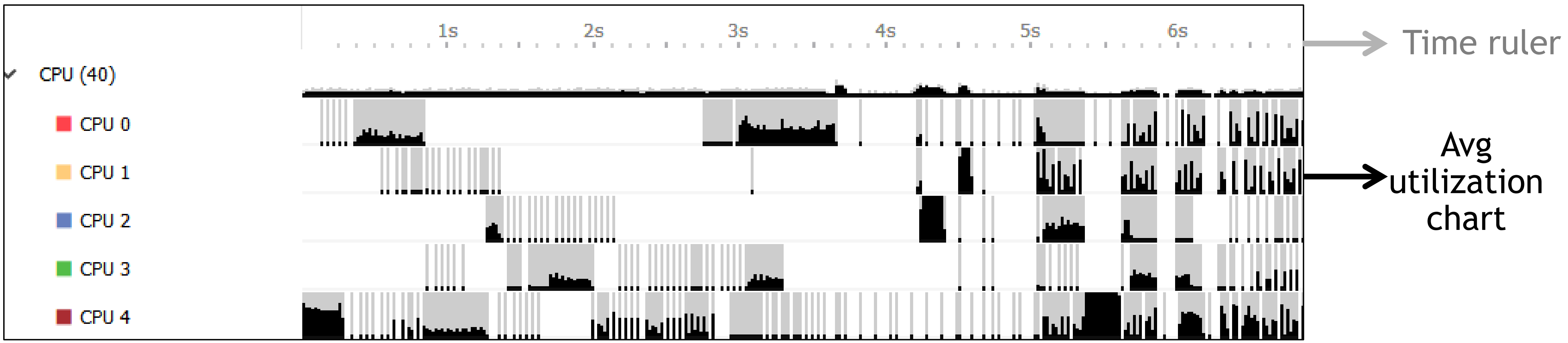
FEATURES



3s
3.1s
3.2s
3.3s
3.4s
3.5s
3.6s
3.7s
CPU (80)
Threads (82)
[419] python
OS runtime libraries
CUDA API
cuDNN
cuBLAS
Profiler overhead
pthread_cond_wait
pthread_cond_wait
[426] python
[364] python
[406] python
78 threads hidden...
CUDA (Tesla P100-SXM2-16GB, 0...
Stream 70
Memory
Kernels
maxwell_fp16_scudnn_fp16...
EigenMetaKernel
maxwell_fp16_scudnn_fp16...
bn_bw_1C11_kernel_new
maxwell_fp16_scudnn_fp16...
dgrad_engine
58 kernel group(s) hidden...
6 stream(s) hidden...
CUDA (Tesla P100-SXM2-16GB, 0...
CUDA (Tesla P100-SXM2-16GB, 0...
CUDA (Tesla P100-SXM2-16GB, 0...

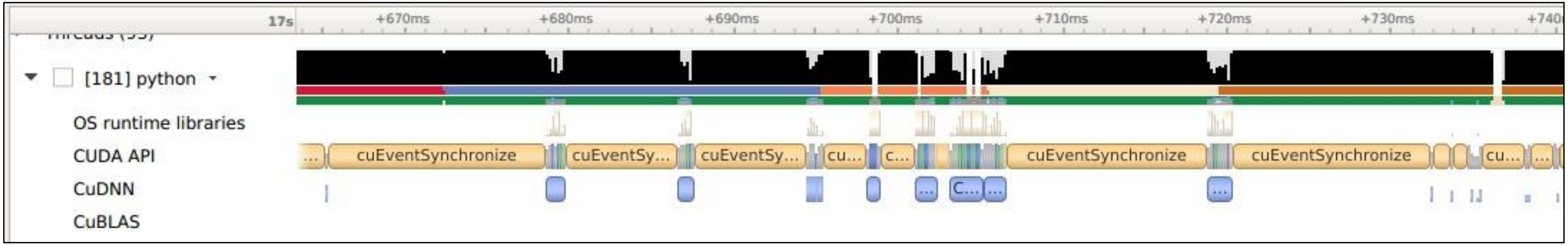
CPU CORES WORKLOAD

- See CPU core utilization by application's threads
- Locate idle time on CPU cores



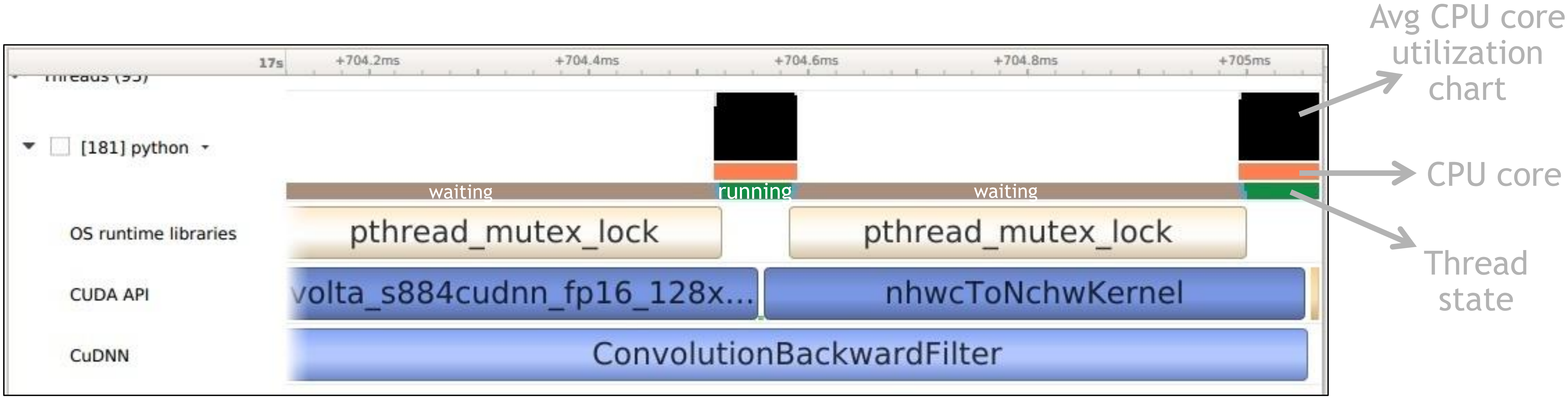
THREADS

- Get an overview of each thread's activities
  - OS runtime libraries usage
  - API usage: CUDA, CuDNN, CuBLAS, OpenACC, OpenGL, DX12 (More to come!)



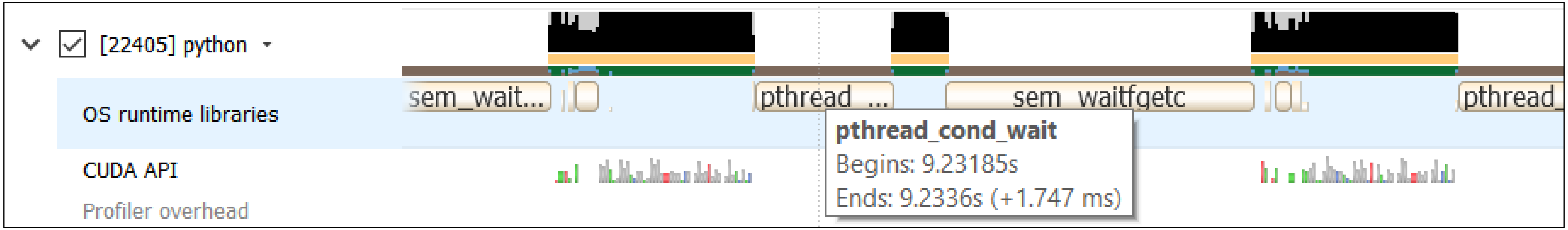
THREADS

- Get an overview of each thread's activities
  - OS runtime libraries usage
  - API usage: CUDA, CuDNN, CuBLAS, OpenACC, OpenGL, DX12 (More to come!)



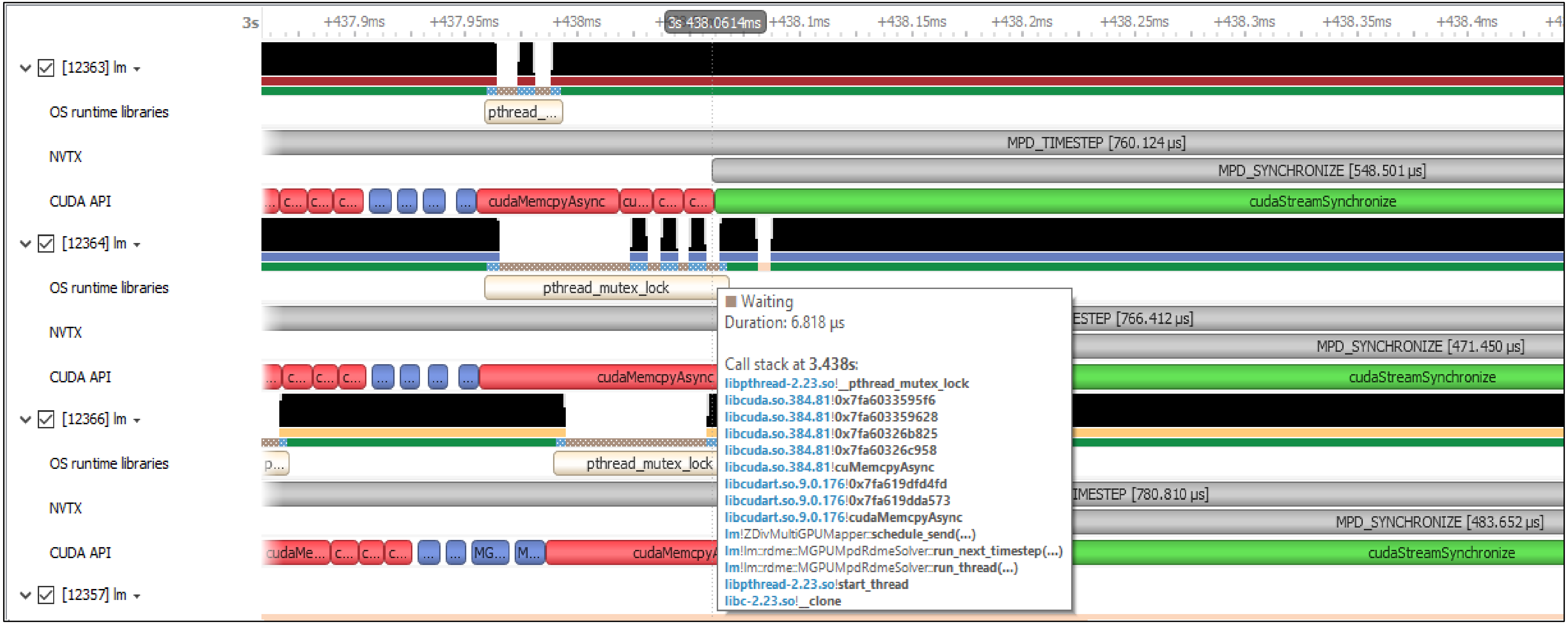
OS RUNTIME LIBRARIES

- Identify time periods where threads are blocked
- See the block reason
- Locate redundant synchronizations



OS RUNTIME LIBRARIES

Backtrace for time-consuming calls to OS runtime libs

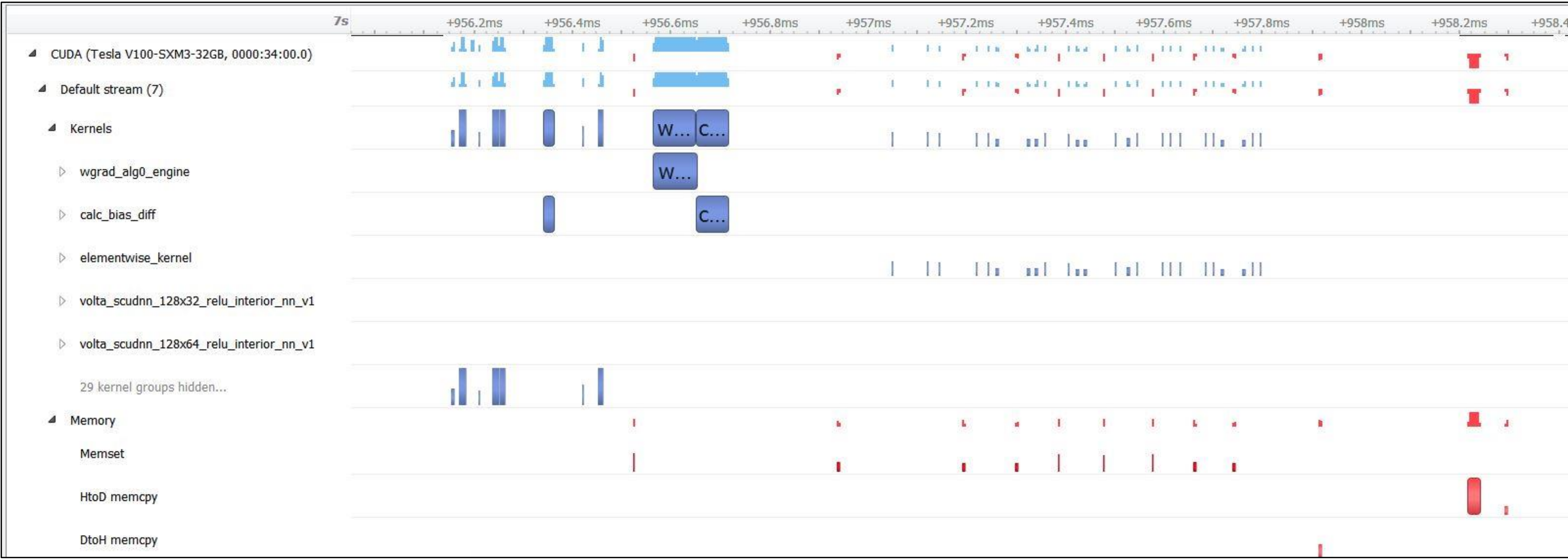


CUDA API

- Trace CUDA API Calls on OS thread
- See when kernels are dispatched
- See when memory operations are initiated
- Locate the corresponding CUDA workload on GPU

GPU WORKLOAD

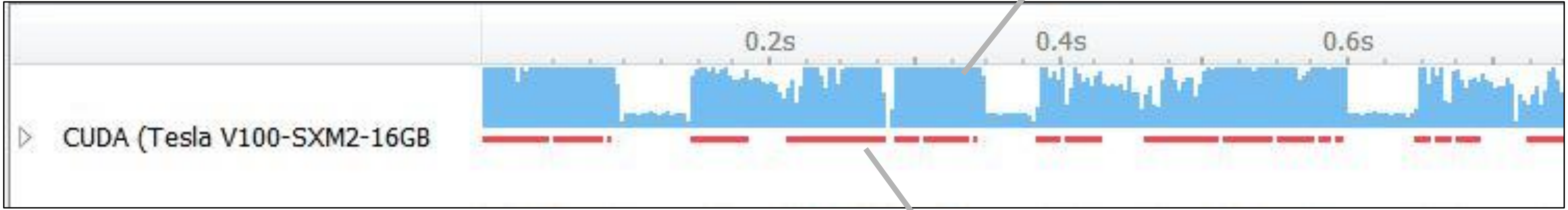
- See CUDA workloads execution time
- Locate idle GPU times



GPU WORKLOAD

- See trace of GPU activity
- Locate idle GPU times

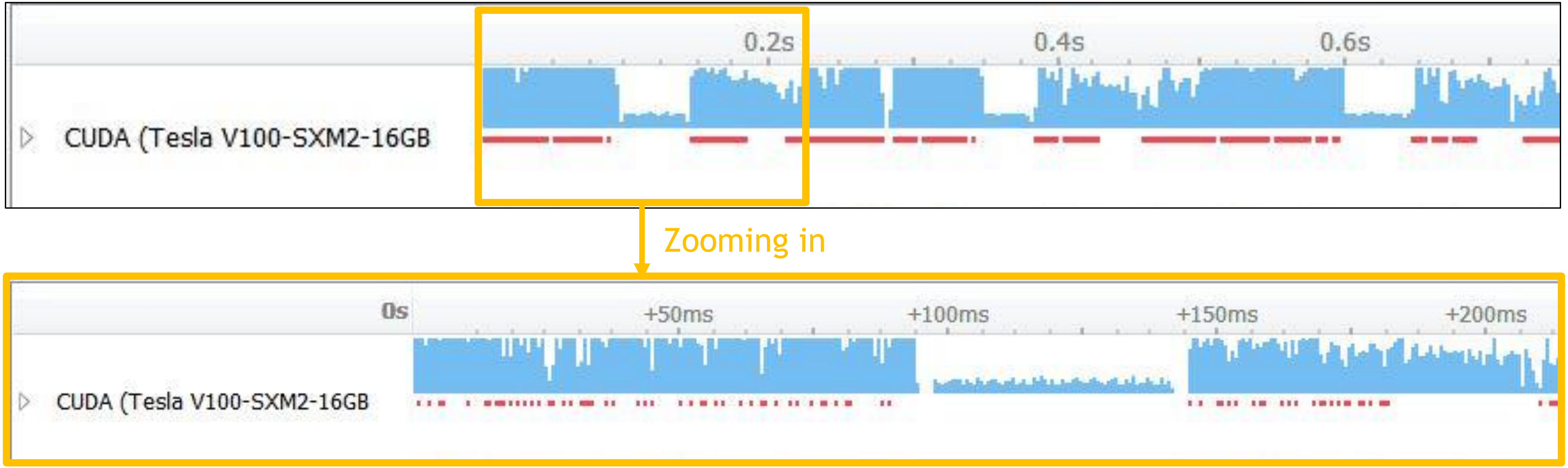
% Chart for
Avg. CUDA kernel coverage
(Not SM occupancy)



% Chart for
Avg. no. of memory operations

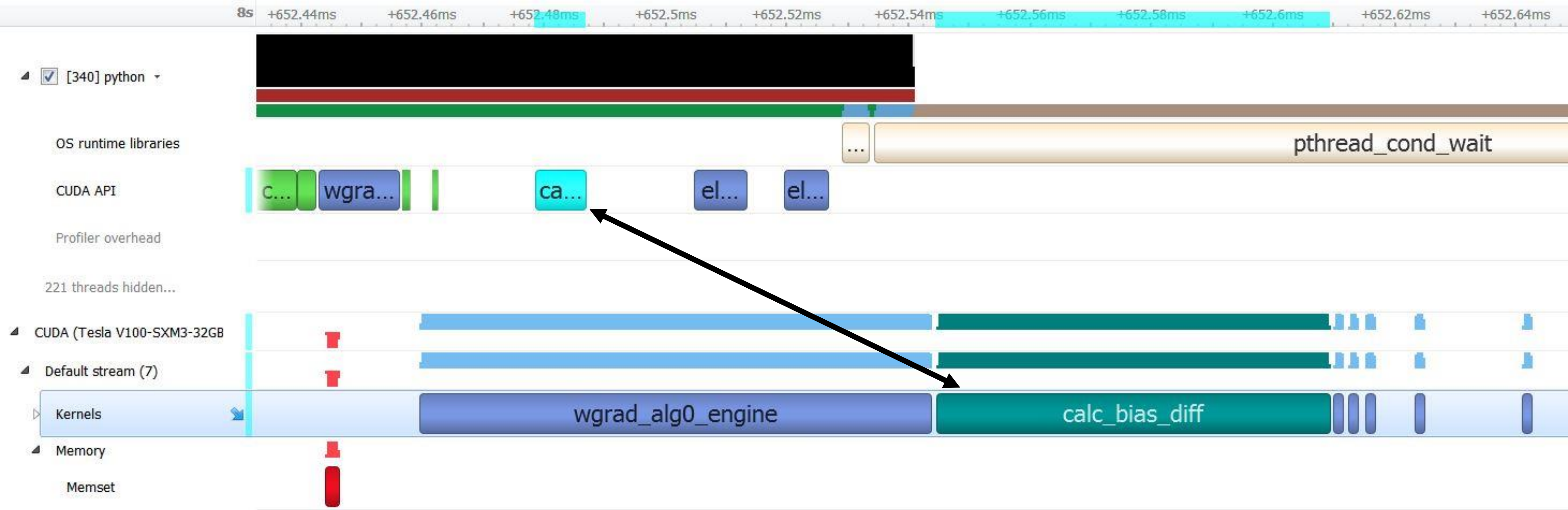
GPU WORKLOAD

- See trace of GPU activity
- Locate idle GPU times



GPU utilization level of detail shows valleys for sparse kernel time coverage

CORRELATION TIES API TO GPU WORKLOAD



Selecting one highlights both cause and effect, i.e. dependency analysis

STATISTICAL SAMPLING

Periodic sampling of thread's callstack

Bottom-Up View | Process [9695] vmd_LINUXAMD64.11 (3 of 19 threads)

Filter... | **99.82%** (23,260 samples) of data is shown due to applied filters.

IP sample

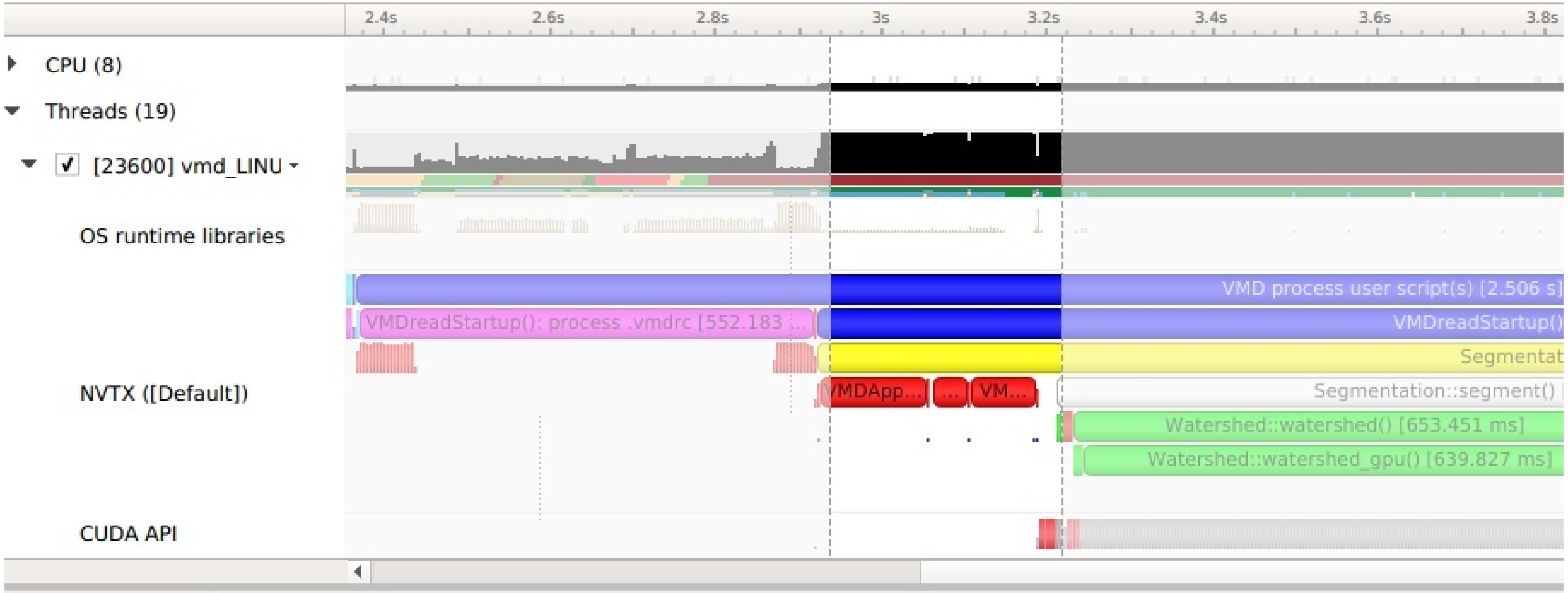
Code path 1

Code path 2 leading to IP sample

Symbol Name	Self, %	Module Name
VolumetricData::compute_volume_gradient()	20.14	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VolumetricData::compute_volume_gradient()	20.14	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
BaseMolecule::add_volume_data(char const*, double const*, double const*, double const*, double const*, int, int, int, float*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::molecule_add_volumetric(int, char const*, double const*, double const*, double const*, double const*, int, int, int, float*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
obj_segmentation(void*, Tcl_Interp*, int, Tcl_Obj* const*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
[Max depth]	18.30	[Max depth]
BaseMolecule::add_volume_data(char const*, float const*, float const*, float const*, float const*, int, int, int, float*, float*, float*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
MolFilePlugin::read_volumetric(Molecule*, int, int const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::molecule_load(int, char const*, char const*, FileSpec const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
text_cmd_mol(void*, Tcl_Interp*, int, char const**)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclInvokeStringCommand	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclEvalObjvInternal	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclExecuteByteCode	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclCompEvalObj	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclEvalObjEx	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
Tcl_RecordAndEvalObj	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclTextInterp::evalFile(char const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::logfile_read(char const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDreadStartup(VMDApp*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
[Max depth]	1.84	[Max depth]
0x7f10ca7022d6	5.13	/usr/lib64/libcuda.so.390.25
obj_segmentation(void*, Tcl_Interp*, int, Tcl_Obj* const*)	3.44	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11

STATISTICAL SAMPLING

Filter samples based on time



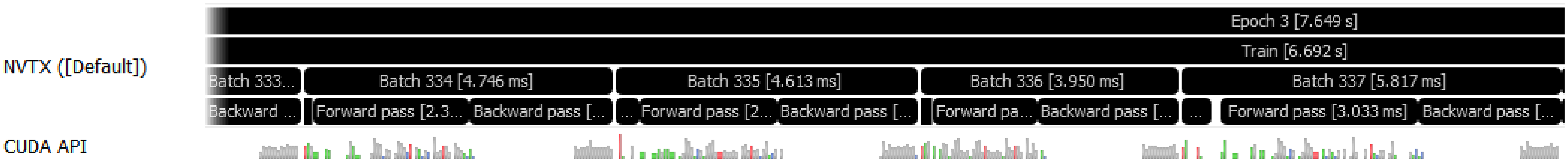
Bottom-Up View ▾ Process [23600] vmd_LINUXAMD64 (3 of 19 threads)

Filter... **9.68**% (2,349 samples) of data is shown due to applied filters. Time filter: 2.94 to 3.22 (0.28 seconds or **2.8**%).

Symbol Name	Self, %	Module Name
_IO_vfscanf	21.97	/usr/lib64/libc-2.17.so
_IO_vfscanf	21.97	/usr/lib64/libc-2.17.so
__isoc99_vsscanf	21.97	/usr/lib64/libc-2.17.so
__isoc99_sscanf	21.97	/usr/lib64/libc-2.17.so
atoifw	20.65	/home/johns/vmd/LINUXAMD64/vmd_LINUXAMD64
read_pdb_structure	0.72	/home/johns/vmd/LINUXAMD64/vmd_LINUXAMD64
get_psf_atom	0.60	/home/johns/vmd/LINUXAMD64/vmd_LINUXAMD64
VolumetricData::compute_volume_gradient()	18.26	/home/johns/vmd/LINUXAMD64/vmd_LINUXAMD64
read_ccp4_data(void*, int, float*, float*)	8.77	/home/johns/vmd/LINUXAMD64/vmd_LINUXAMD64

NVTX INSTRUMENTATION

- NVIDIA Tools Extension (NVTX) to annotate the timeline with application's logic
- Helps understand the profiler's output in app's algorithmic context



FEATURES SUMMARY

User Instrumentation

NVidia Tools eXtension - aka NVTX

OS threads timeline with API Tracing

OS runtime, CUDA, CuDNN, CuBLAS, OpenACC, OpenGL

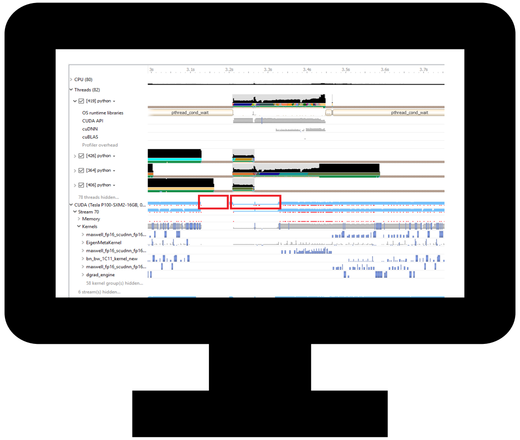
Correlate with GPU workloads

Backtrace Collection

Sampled IPs

Blocked state

GUI

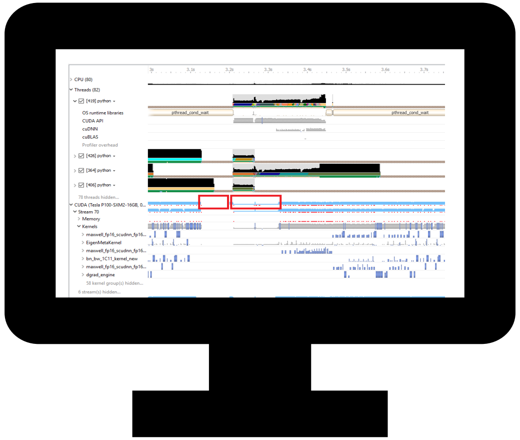


- Fast
- Visualize millions of events
- Incredible level of detail

DATA COLLECTION (GUI)

Host-Target
Remote Collection

DATA COLLECTION (GUI)



Supports Local collection	✓	✓	⦸

DATA COLLECTION (CLI)

Command Line Interface

No connection! Import later

CLI enables easy collection on servers and in containers

REPORT NAVIGATION DEMO

Outstanding **Interactive Performance and Level of Detail Available**

CASE STUDY 1: SIMPLE DNN TRAINING

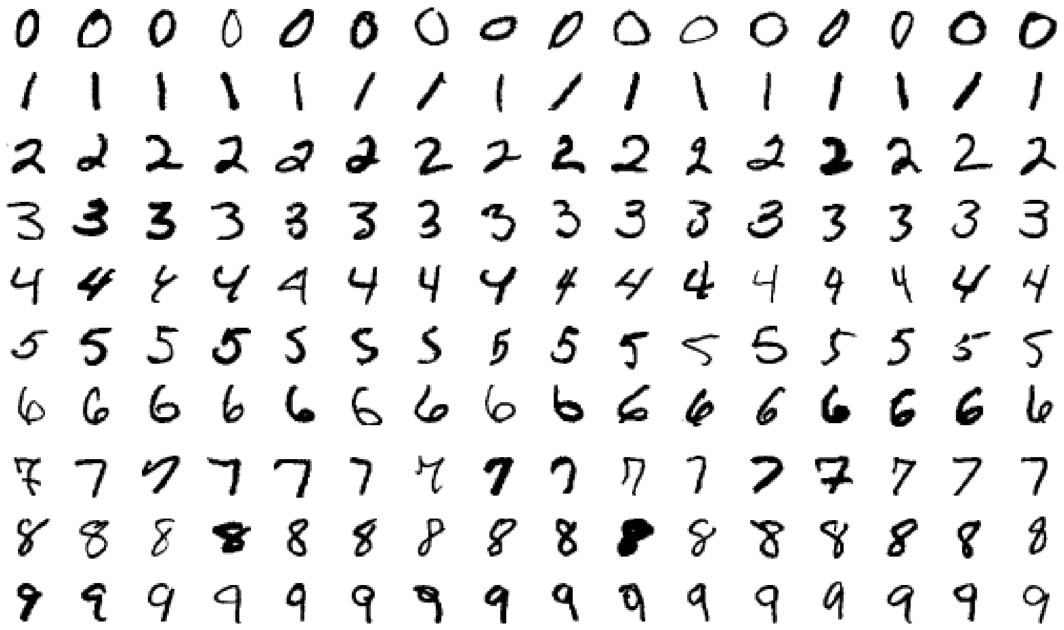
DATA SET

The MNIST database

A database of handwritten digits

Will be used for training a DNN

that recognizes handwritten digits



SIMPLE TRAINING PROGRAM

- A simple DNN training program from https://github.com/pytorch/examples/tree/master/mnist
- Uses PyTorch, accelerated using a Volta GPU
- Training is done in batches and epochs
  1. Data is copied to the device
  2. Forward pass
  3. Backward pass

SIMPLE TRAINING PROGRAM

```
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)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

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

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

Backward pass (loss.backward(), optimizer.step())

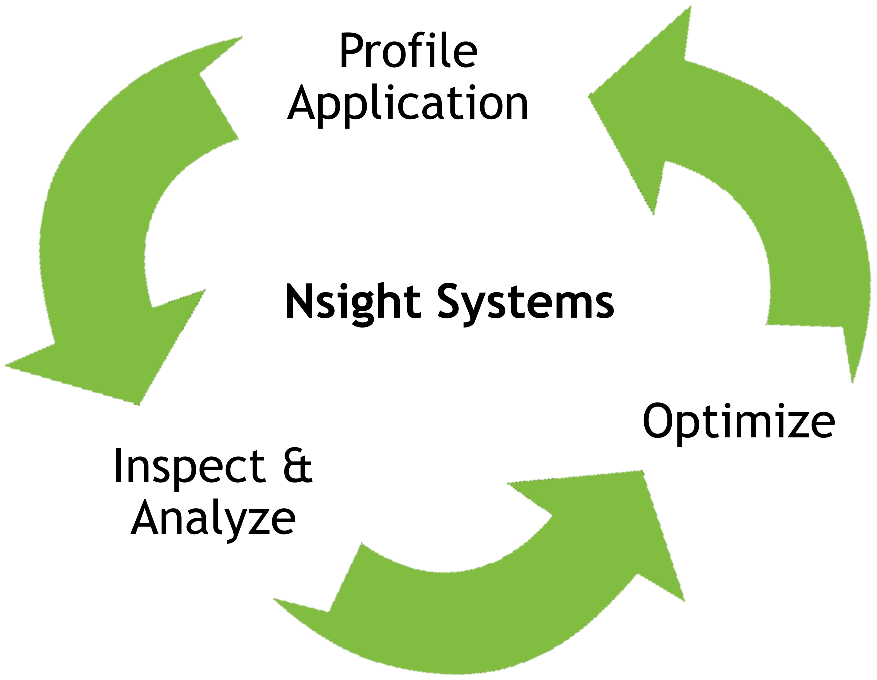
TRAINING PERFORMANCE

Execution time

> python main.py

Takes 89 seconds on a Quadro Volta GPU

OPTIMIZATION WORKFLOW



STEP 1: PROFILE

APIs to be traced

Show output on console

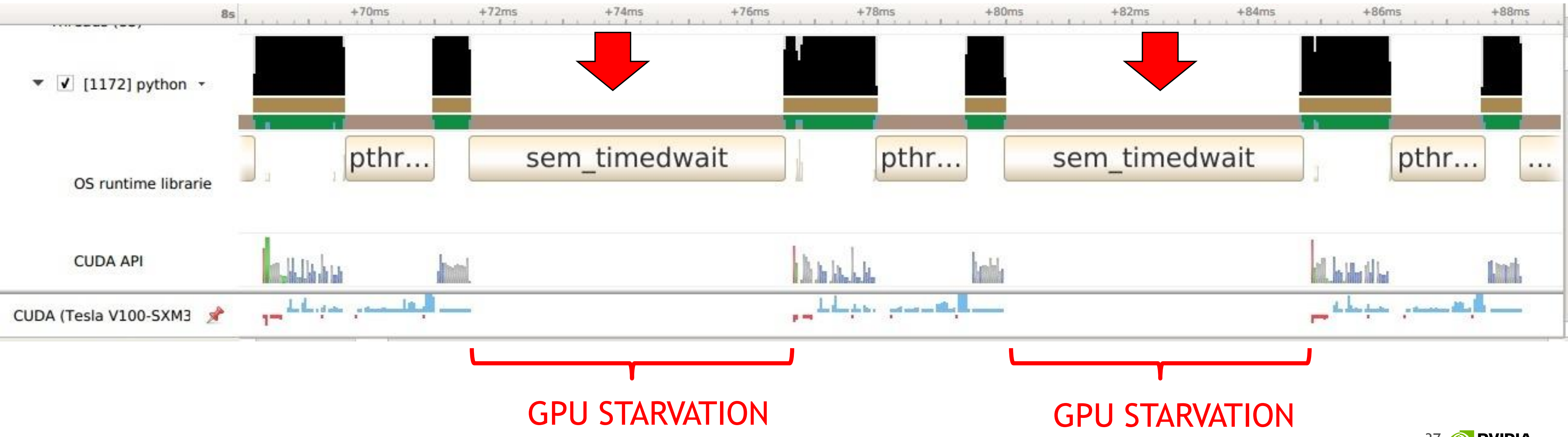
> nsys profile –t cuda,osrt,nvtx –o baseline –w true python main.py

Name for output file

Application command

BASELINE PROFILE

- Training time = 89 seconds
- CPU waits on a semaphore and starves the GPU!



NVTX ANNOTATIONS

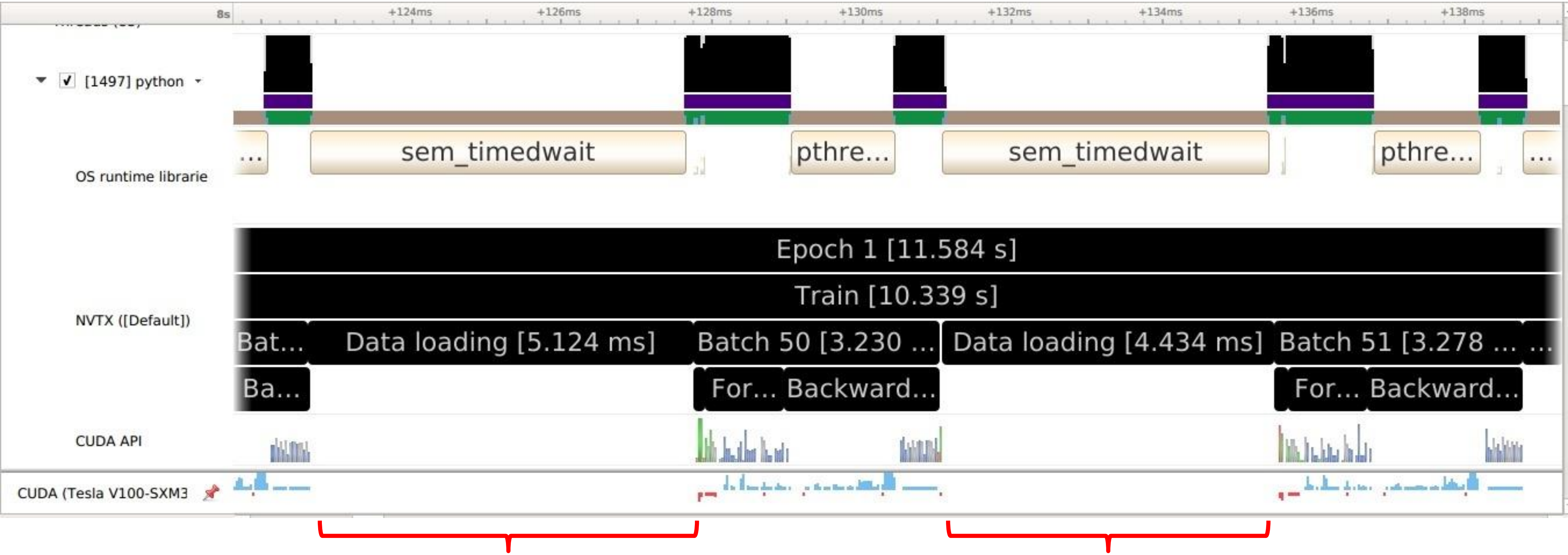
Add NVTX to annotate the timeline with application logic

```
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()
...
```

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:

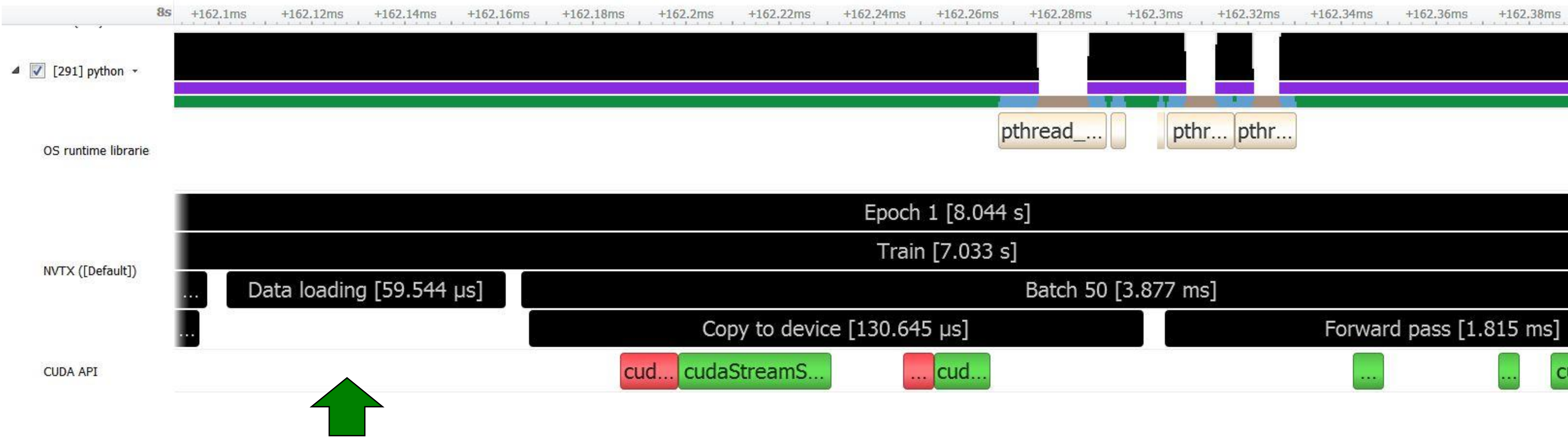
```
kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else {}
```

Let's switch to using 8 worker threads:

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

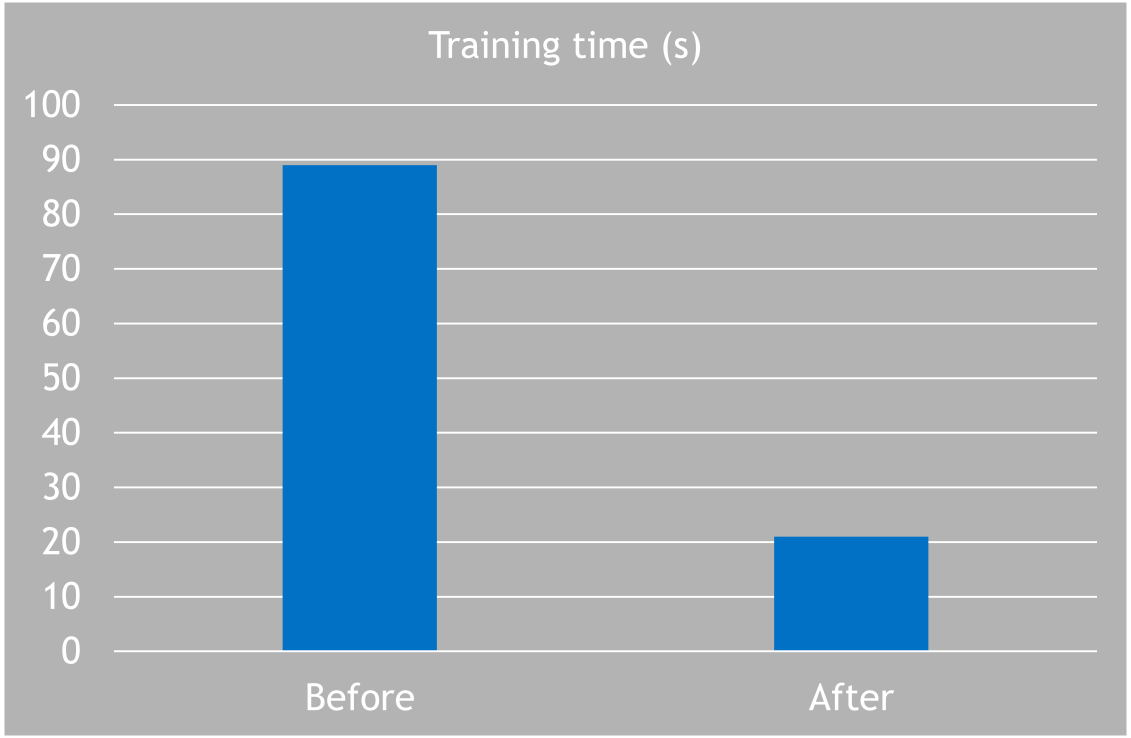
AFTER OPTIMIZATION

- Time for data loading reduced for each batch



Reduced from 5.1ms to 60us
for batch 50

AFTER OPTIMIZATION



4.2x speedup on Tesla V100 GPU!

CASE STUDY 2: 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

OPENACC SAMPLE

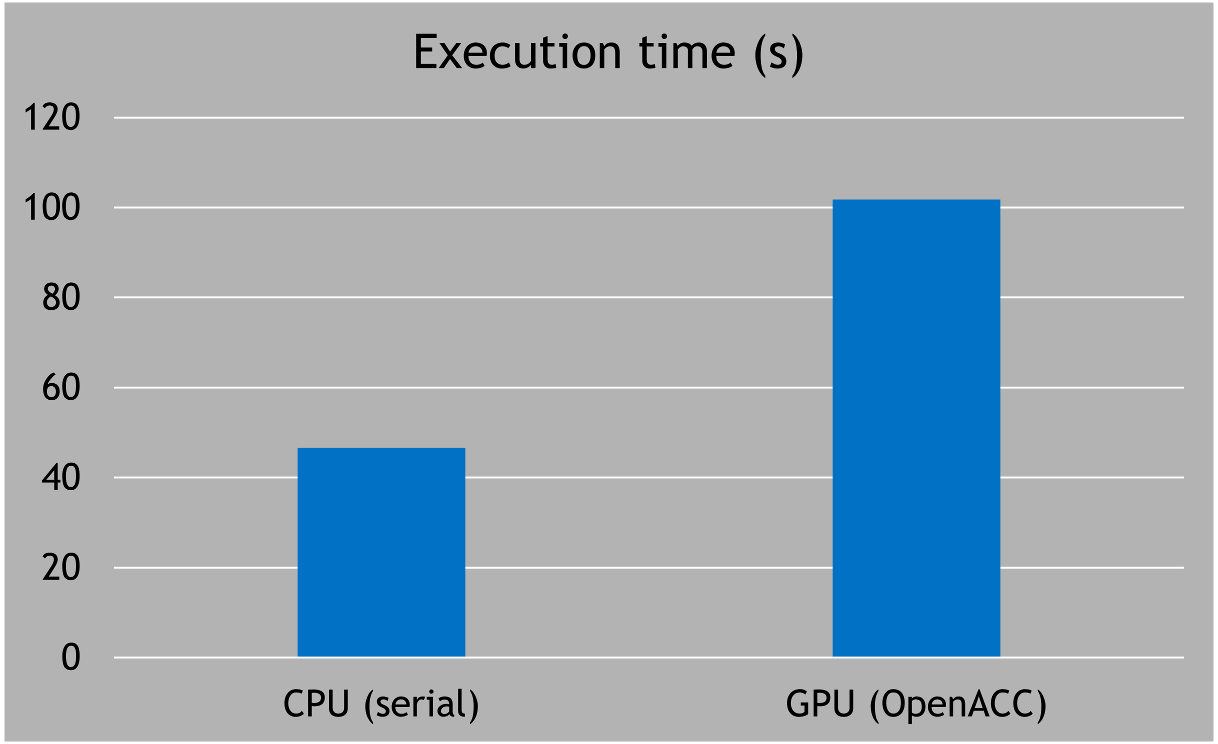
```
while ( error > tol && iter < iter_max ) {
  error = 0.0;
  #pragma acc kernels
  {
    for( int j = 1; j < n-1; j++) {
      for( int i = 1; i < m-1; i++ ) {
        Anew[j][i] = …
        error = fmax( error, fabs(Anew[j][i] - A[j][i]));
      }
    }
    for( int j = 1; j < n-1; j++) {
      for(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

Convergence loop

Stencil calculation

Update matrix

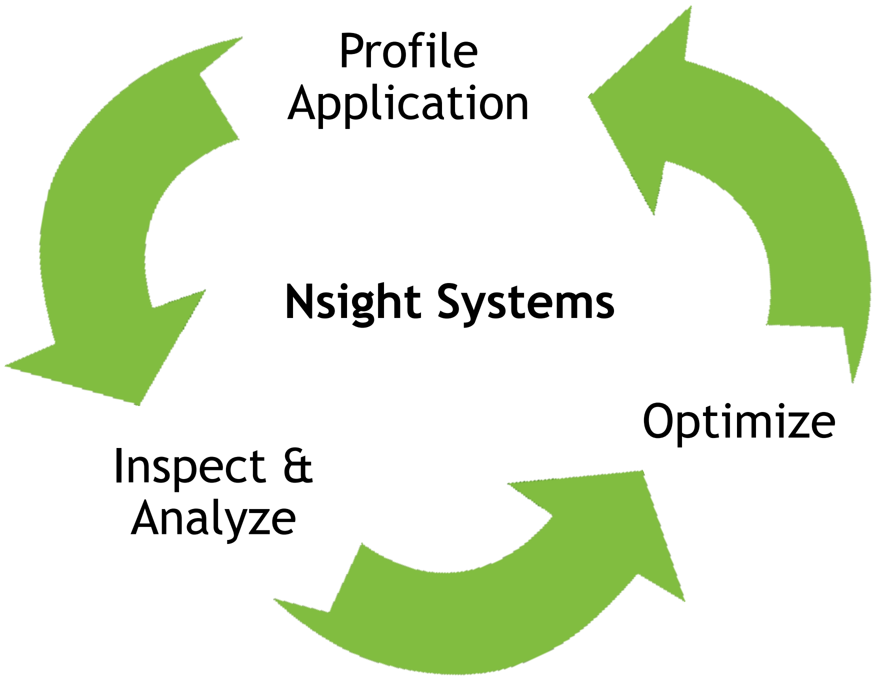
PERFORMANCE



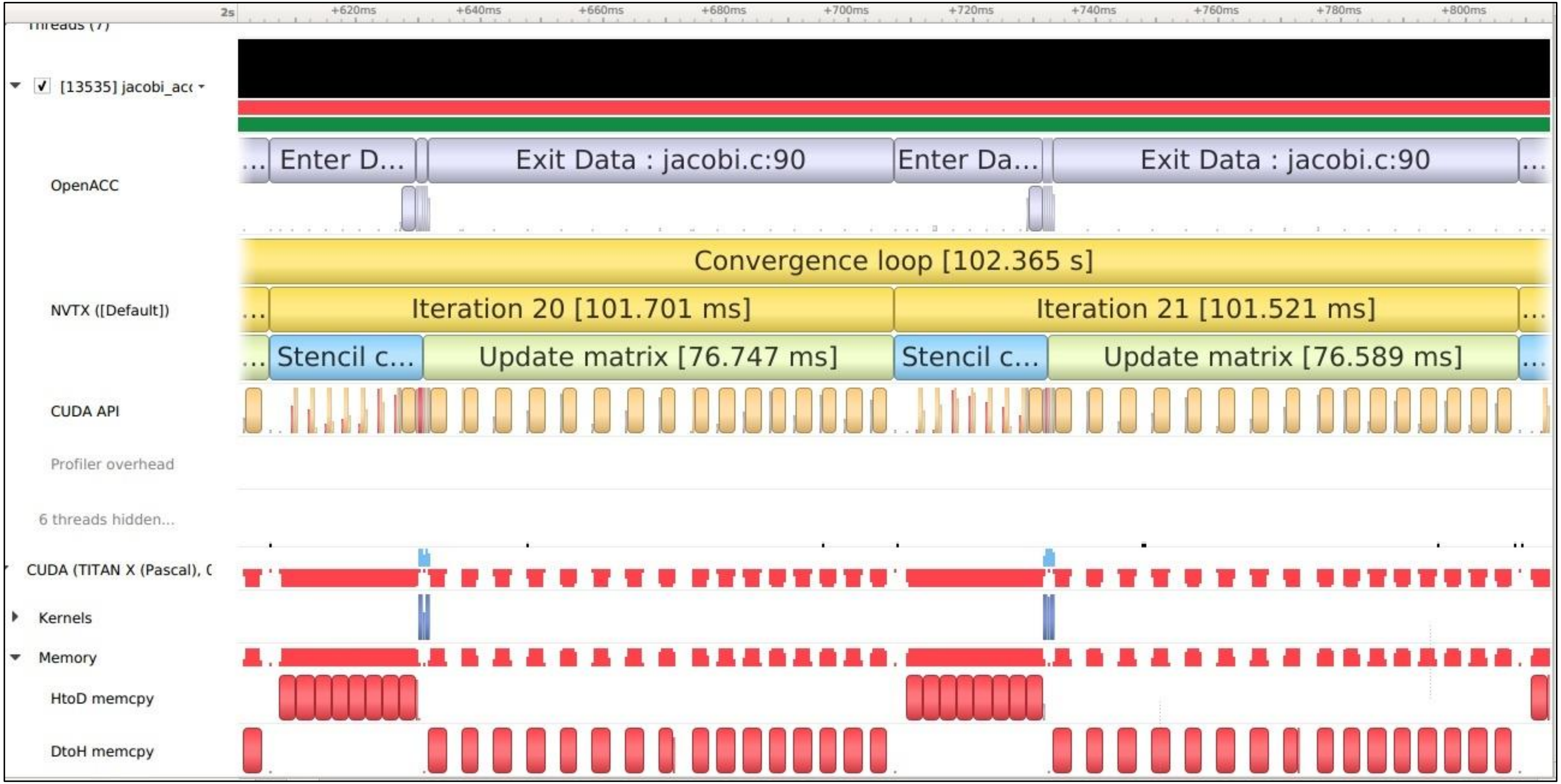
Execution time for 1000 iterations on a system with:
Intel® Core™ i7-6850K CPU
NVIDIA TITAN X (Pascal) GPU

That is unexpected!

OPTIMIZATION WORKFLOW



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

OPENACC SAMPLE

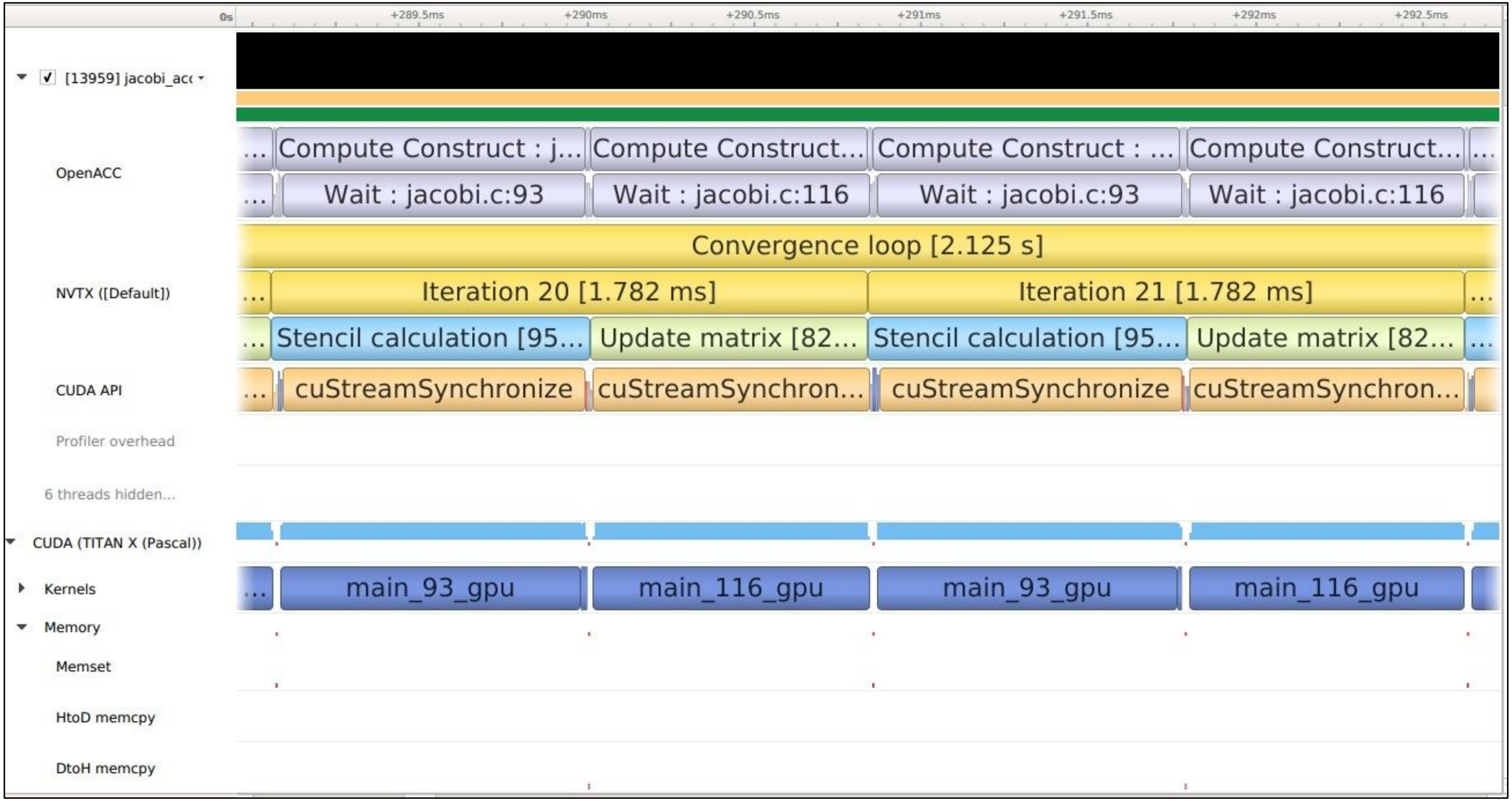
```
#pragma acc data copy(A) create(Anew)
while ( error > tol && iter < iter_max ) {
  error = 0.0;
  #pragma acc kernels
  {
    for( int j = 1; j < n-1; j++) {
      for( int i = 1; i < m-1; i++ ) {
        Anew[j][i] = …
        error = fmax( error, fabs(Anew[j][i] - A[j][i]));
      }
    }
    for(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

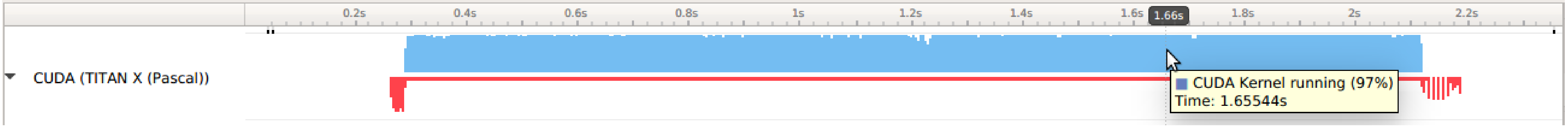
AFTER OPTIMIZATION



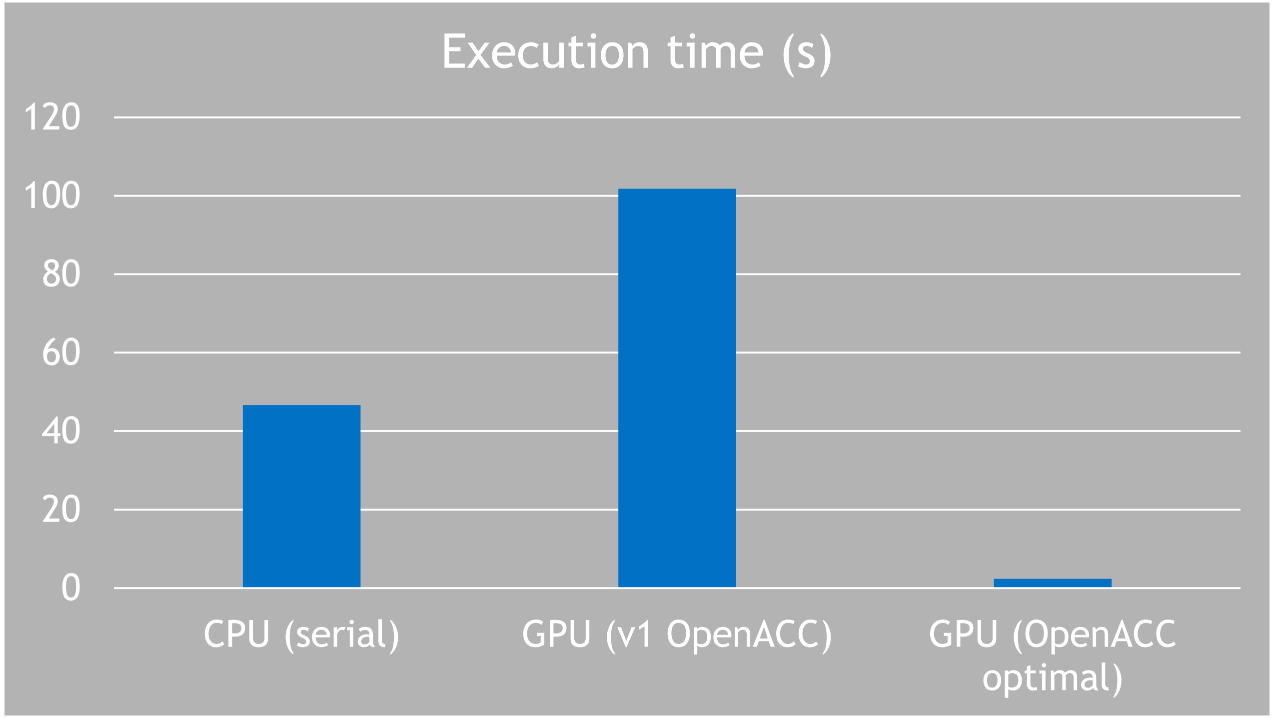
Excessive data copies eliminated

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

COMMON OPTIMIZATION OPPORTUNITIES

- Blog post

https://devblogs.nvidia.com/nsight-systems-exposes-gpu-optimization

- Watch GTC, San Jose 2018 talk
  - By John Stone of UIUC & Robert Knight of NVIDIA

3.2x-4.1x Speedup Achieved on Visual Molecular Dynamics!

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

PROFILING ON BLUEWATERS

Nsight Systems requirements:

- GLIBC >= v2.14
    - BlueWaters nodes with x86_64 CPUs have v2.11, so use a Shifter container with newer OS.
- CPU sampling requires Linux kernel version >= 4.3
    - BlueWaters nodes with x86_64 CPUs have older kernel. No CPU sampling available.

Other requirements in docs

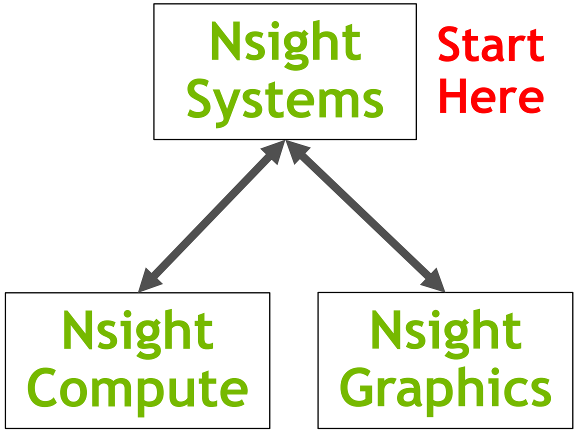
NSIGHT PRODUCT FAMILY

Nsight Systems - System-wide application algorithm tuning

Nsight Compute - Debug/optimize specific CUDA kernel

Nsight Graphics - Debug/optimize specific graphics frame/shader

Workflow



NSIGHT SYSTEMS

- Download from http://developer.nvidia.com/nsight-systems
- Training
  - Docs at https://docs.nvidia.com/nsight-systems/index.html
  - Blog post
  - GTC, San Jose 2018 talk
  - GTC, Israel 2018 talk
- Questions/Requests/Comments?
  - nsight-systems@nvidia.com
  - Developer Forums

NVIDIA