OPTIMIZING HPC SIMULATION AND VISUALIZATION CODE USING NVIDIA NSIGHT SYSTEMS

NVIDIA

Daniel Horowitz – Director of Platform Developer Tools, NVIDIA,

Robert (Bob) Knight – Principal System Software Engineer, NVIDIA

Mike Hallock - Research Programmer, U.Illinois at Urbana Champaign

John Stone – Senior Research Programmer, U.Illinos at Urbana Champaign

March 26, 2018

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

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

Stay tuned for the details

NSIGHT PRODUCT FAMILY

Standalone Performance Tools

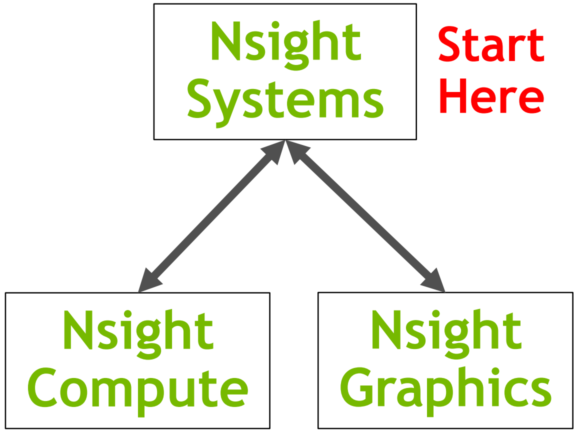
Nsight Systems - System-wide application algorithm tuning

Nsight Compute - Debug/optimize specific CUDA kernel

available @next major CUDA release - Use NVIDIA Visual Profiler today

Nsight Graphics - Debug/optimize specific graphics shader

Workflow



IDE Plugins

Nsight Visual Studio/Eclipse Edition – editor, debugger, some perf analysis

NSIGHT SYSTEMS USER

MAXIMIZE YOUR GPU INVESTMENT

Find the right optimization opportunities

Balance your workload across CPUs and GPUs

Achieve real-time performance requirements

Optimize for HPC environments – minimum time to solution

FEATURES

User Instrumentation

NVidia Tools eXtension - aka NVTX

API Tracing

CUDA, OpenGL,

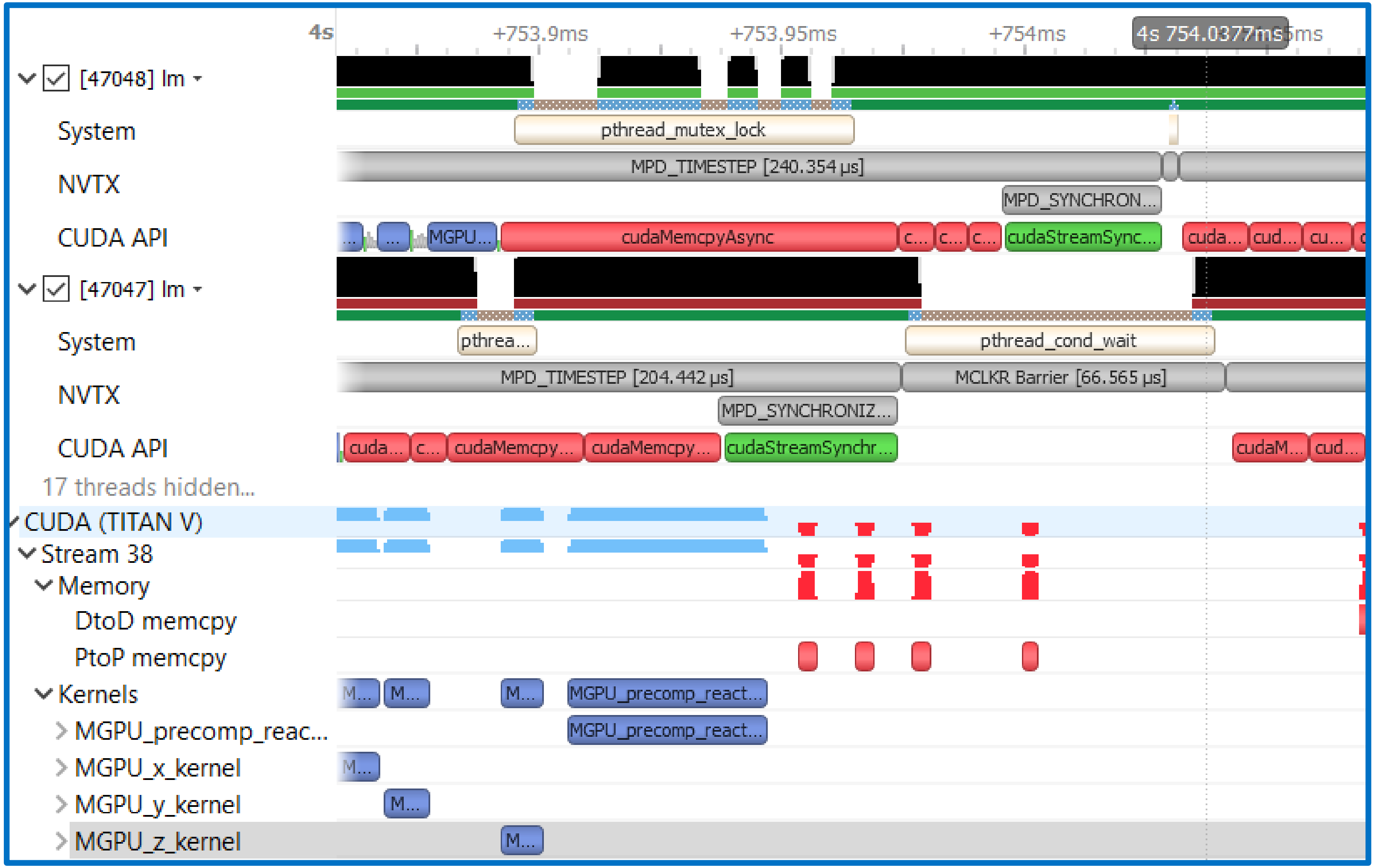
cuDNN, cuBLAS

System *strace-lite*

Backtrace Collection

Sampled IPs

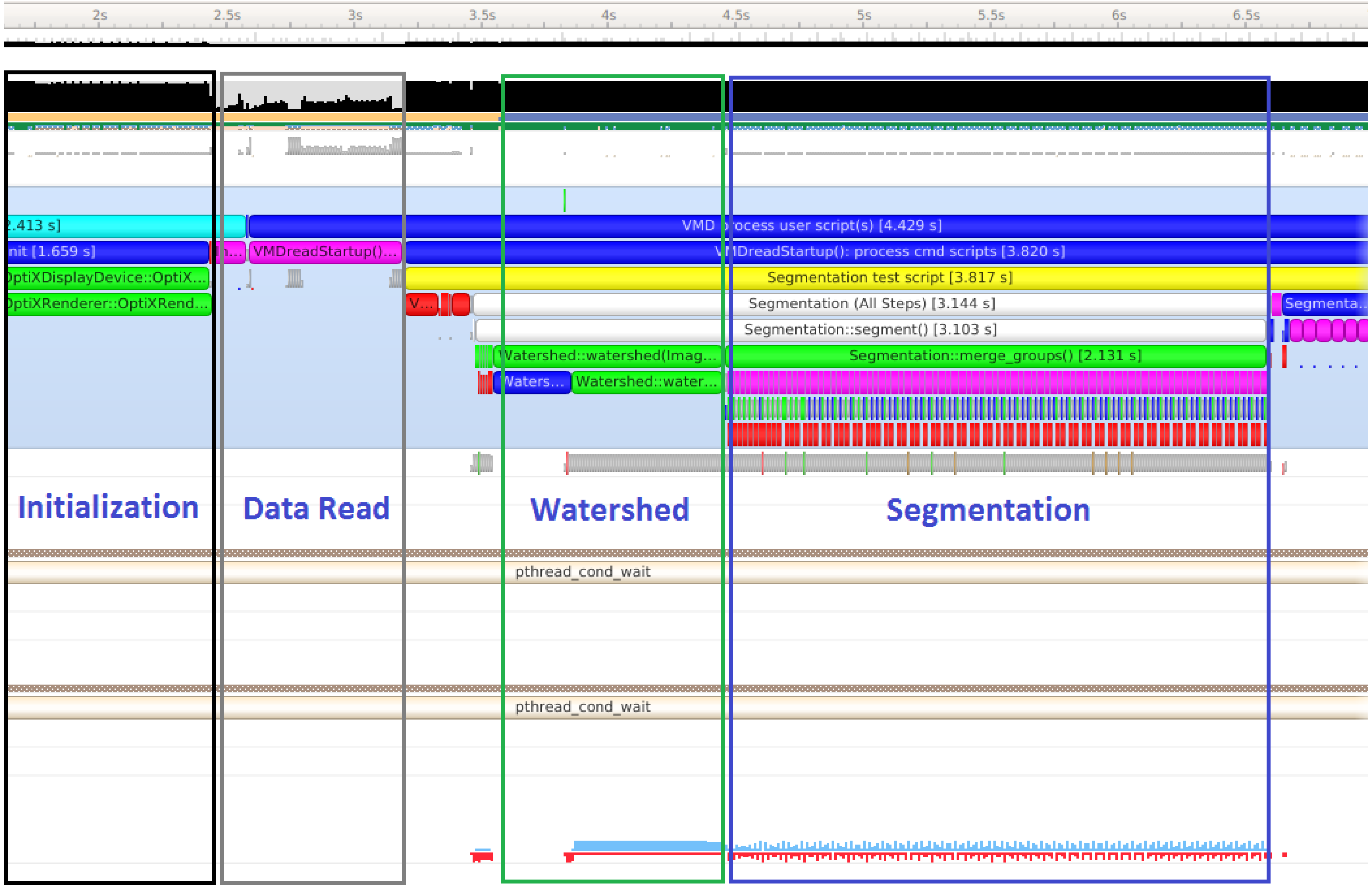
Blocked state



APPLICATION ALGORITHM

Zoom Out

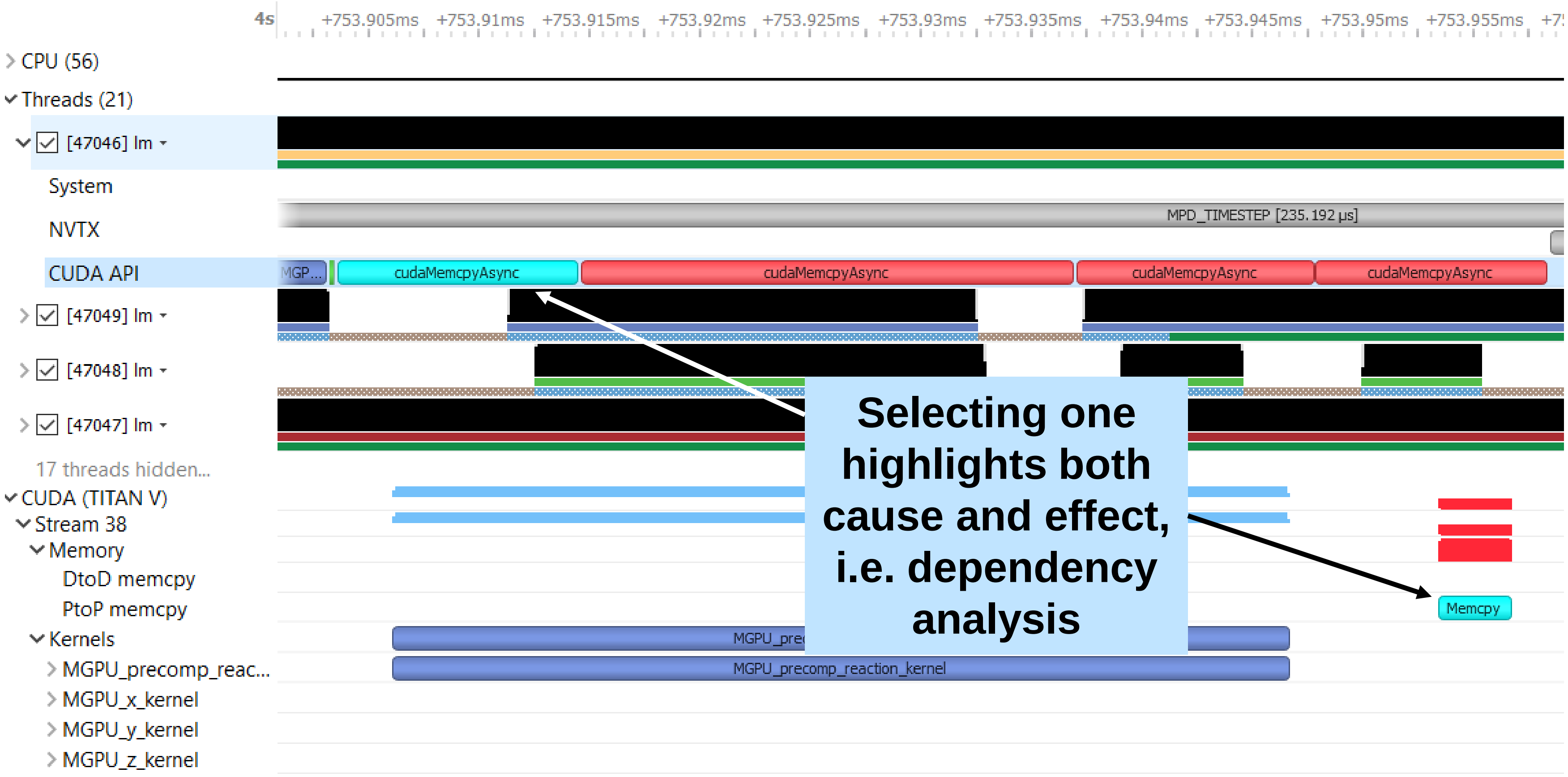
Four Distinct Phases of VMD Algorithm Become Visible



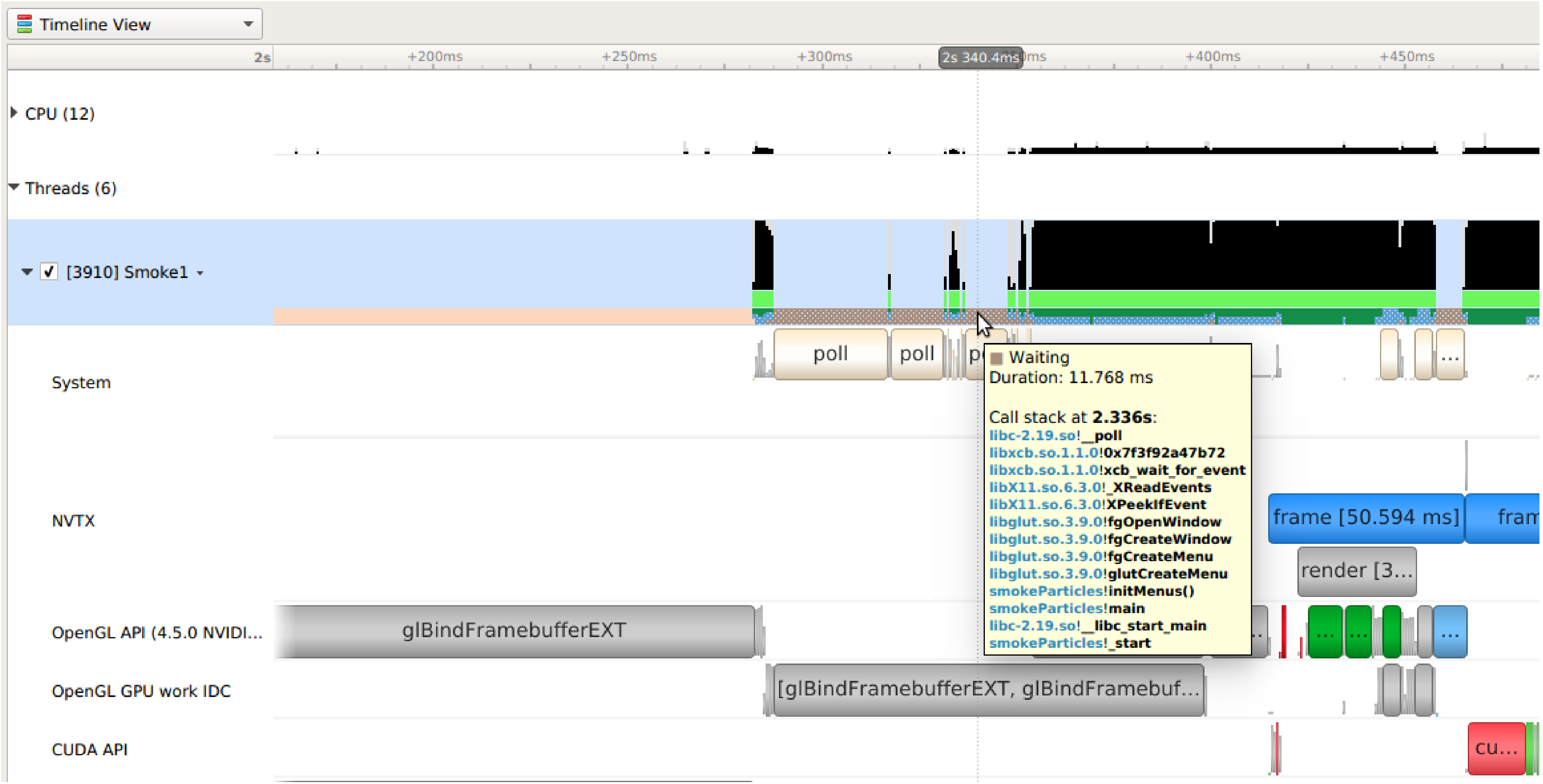
CORRELATION TIES API TO GPU BEHAVIOR

Zoom In

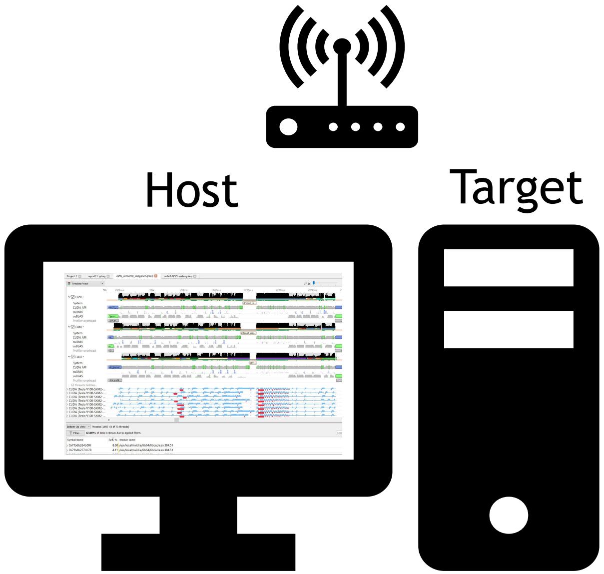
Track Algorithm from CPU to GPU or from GPU to CPU!



BLOCKED STATE BACKTRACE



DATA COLLECTION



Host-Target Remote Collection

Command Line Interface

No connection! Import later

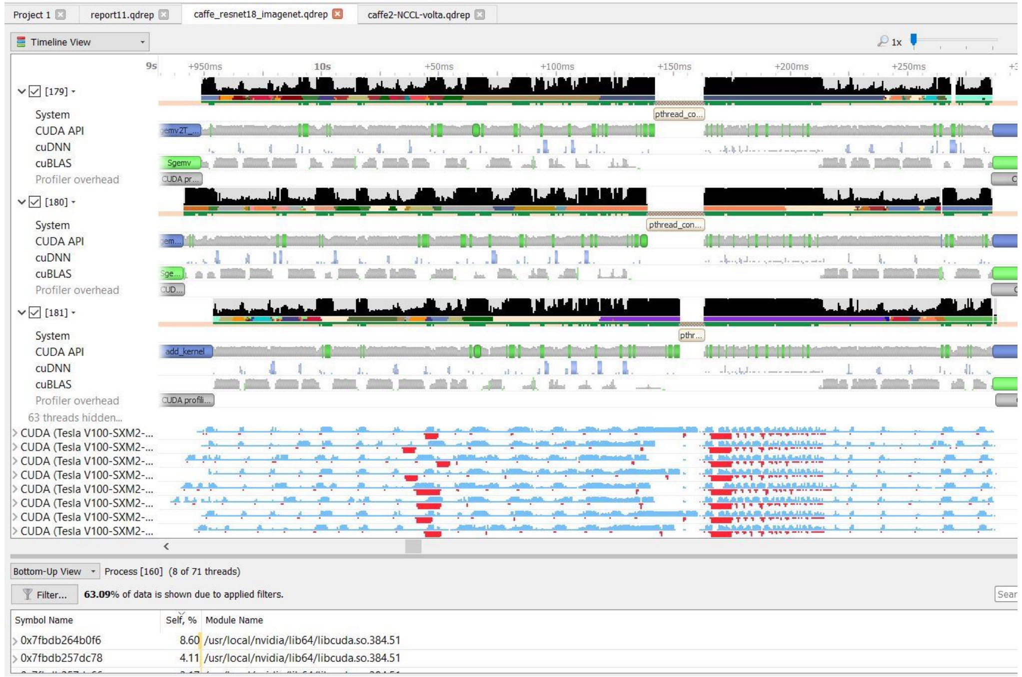
CLI enables easy collection on servers and in containers

REPORT NAVIGATION DEMO

New Tool - *Outstanding* Interactive Performance and Level of Detail Available

Core Areas

- Algorithm Overview Using NVTX Tags
- OS Thread Timeline including APIs Traced
- Correlation of OS Thread API Use with GPU Activity
- CPU Sampling Shows Hot OS Thread Code/Bottlenecks

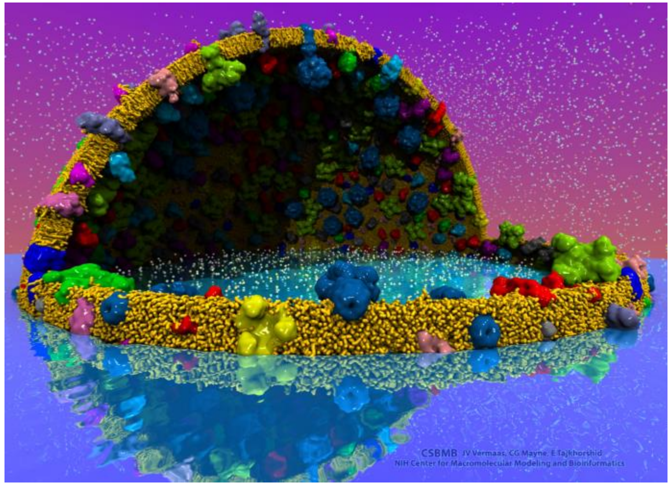


COMMON OPTIMIZATION OPPORTUNITIES

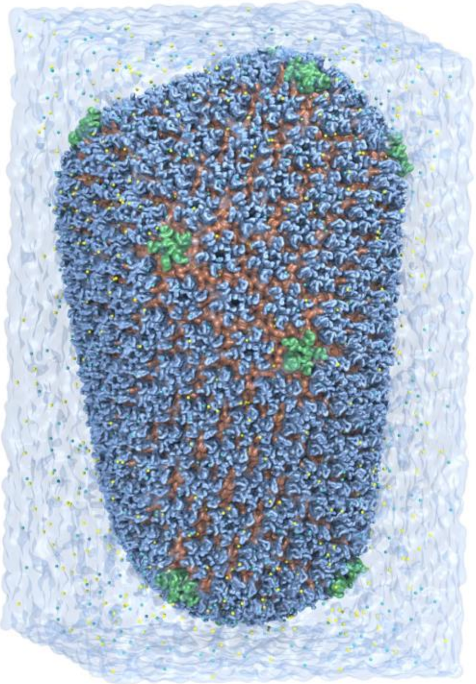
- CPU
  - Thread synchronization
  - Algorithm bottlenecks starve the GPUs
- Multi GPU
  - Communication between GPUs
  - Lack of stream overlap in memory management, kernel execution
- Single GPU
  - Memory operations – blocking, serial, unnecessary
  - Excessive synchronization - device, context, stream, default stream, implicit
  - CPU/GPU overlap – avoid excessive communication

VMD – "Visual Molecular Dynamics"

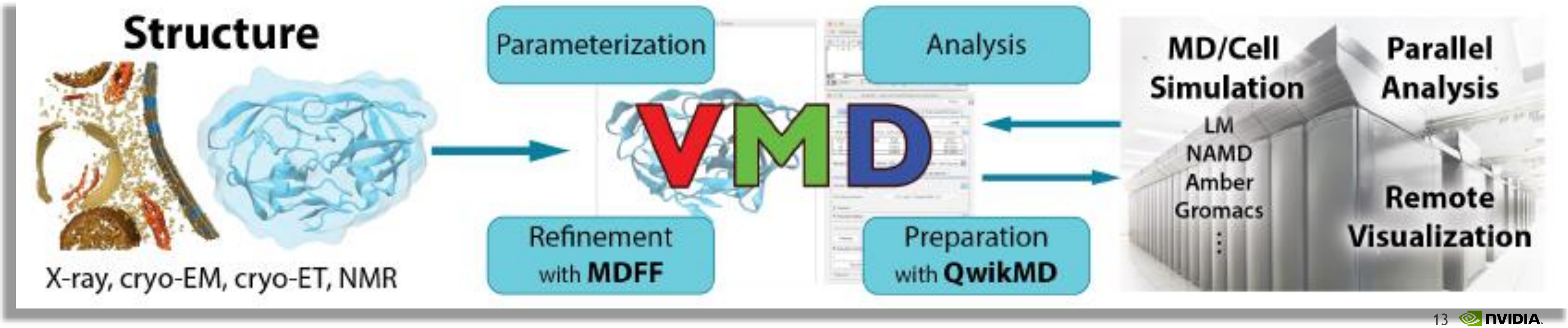
- Visualization and analysis of:
  - Molecular dynamics simulations
  - Lattice cell simulations
  - Quantum chemistry calculations
  - Sequence information
- User extensible scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/



Cell-Scale Modeling



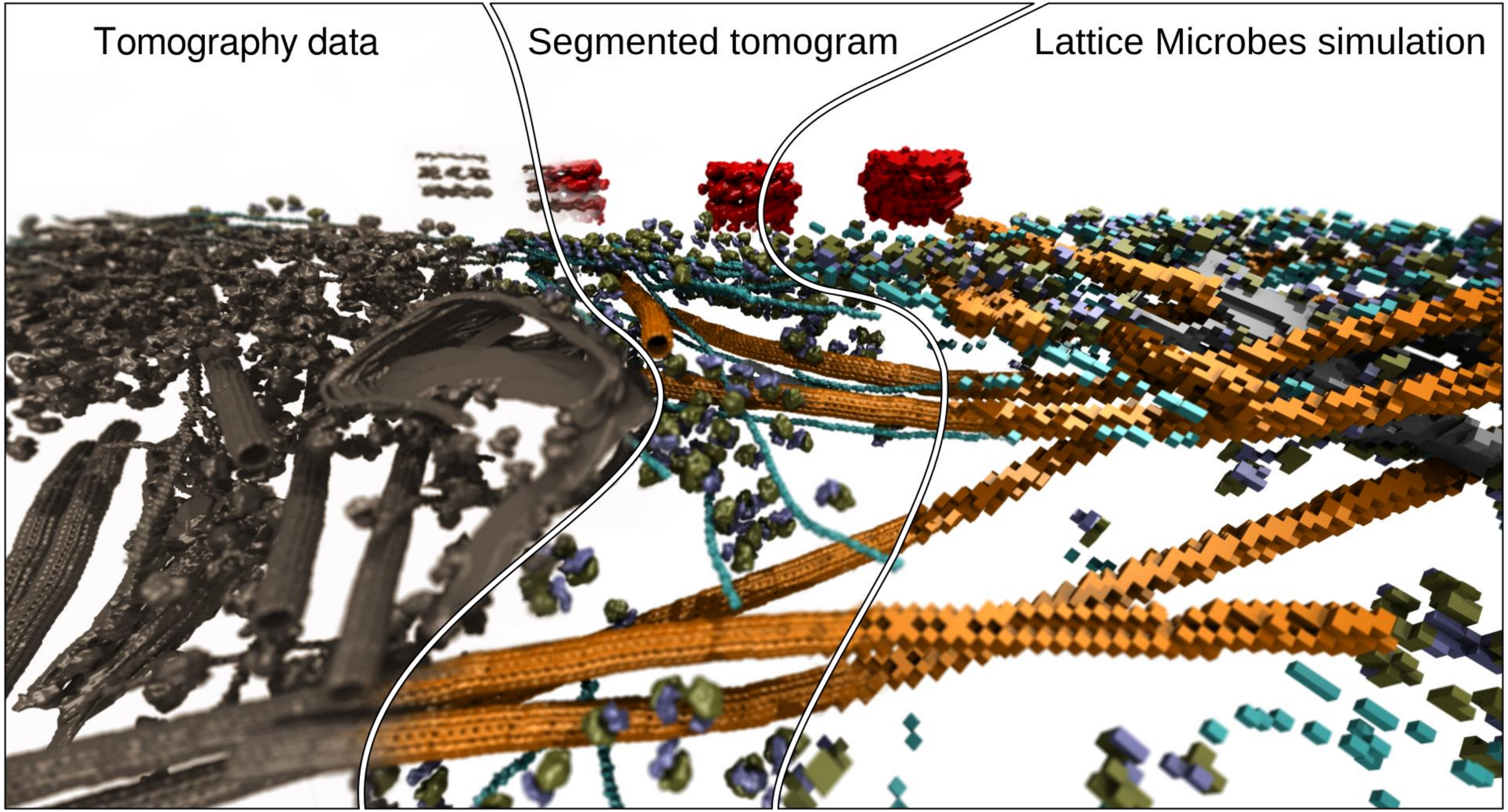
MD Simulation



CRYO-EM / CRYO-ET IMAGE SEGMENTATION

Evaluate 3-D volumetric electron density maps and segment them, to identify key structural components

Index/label components so they can be referred to, colored, analyzed, and simulated…



CRYO-EM DENSITY MAP SEGMENTATION APPROACH, GOALS

Watershed segmentation:

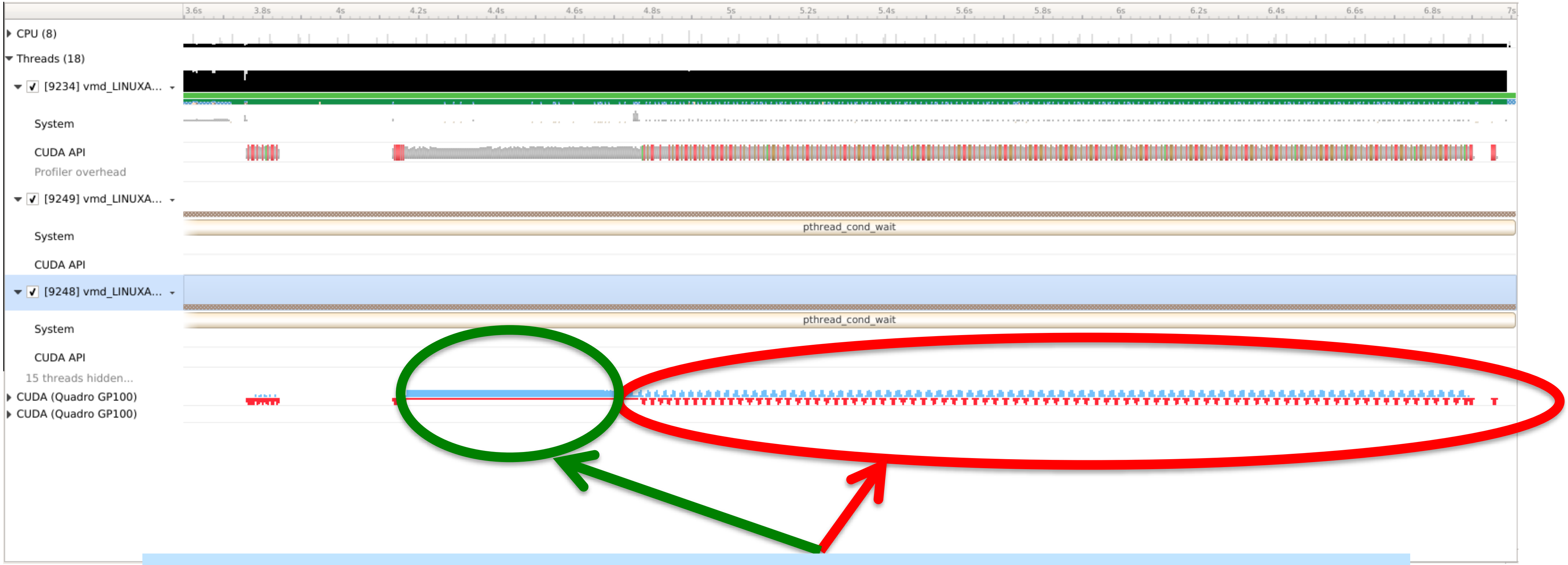
- Smooth/denoise image (e.g. blur)
- Find local minima of image/gradients
- Connect minimum voxels with neighbors of similar intensity, marking them with the same “group” number
- “Grow” each group (merging groups where rules allow) until no more updates occur

Scale-space segmentation variant does further blurring and group merging

Goals:

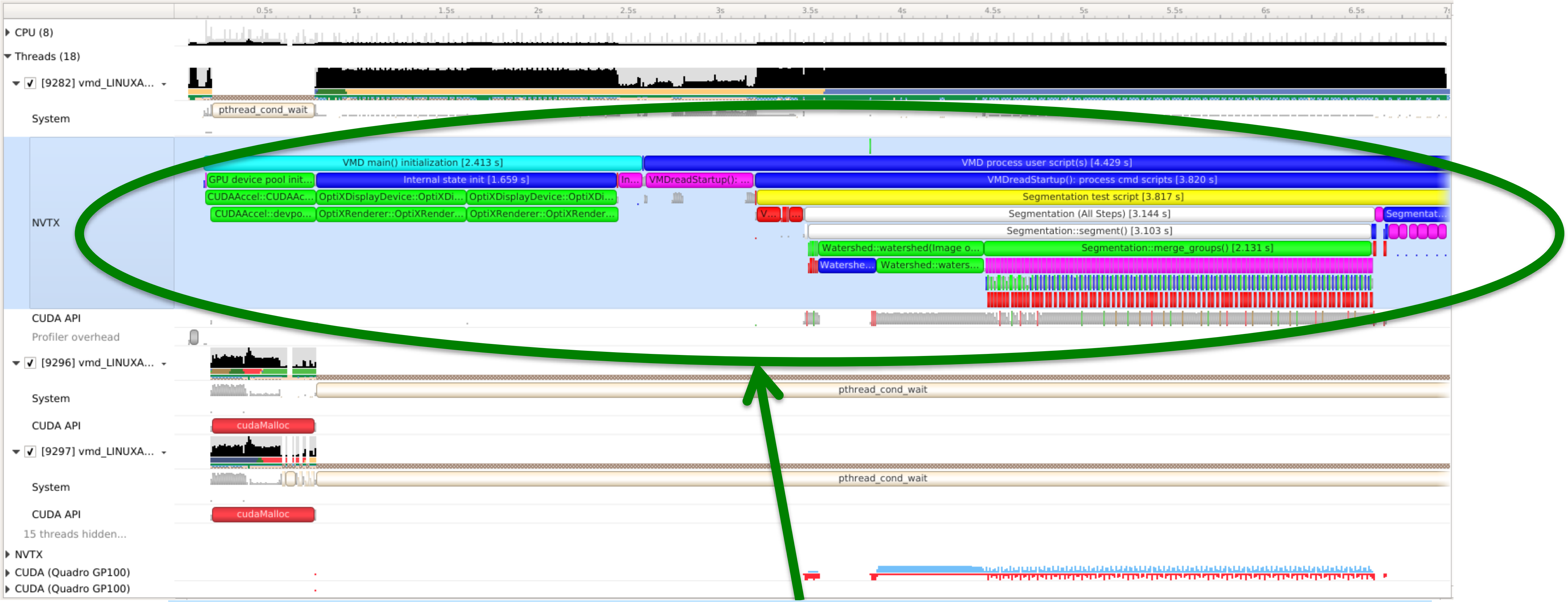
- **Reach interactive performance rates (under 1 second)** for common density map sizes between 128^3 to 256^3 voxels
- Handle next-generation problem sizes (768^3 to 2048^3) smoothly with only a brief wait

1: INITIAL VMD IMAGE SEGMENTATION TRACE



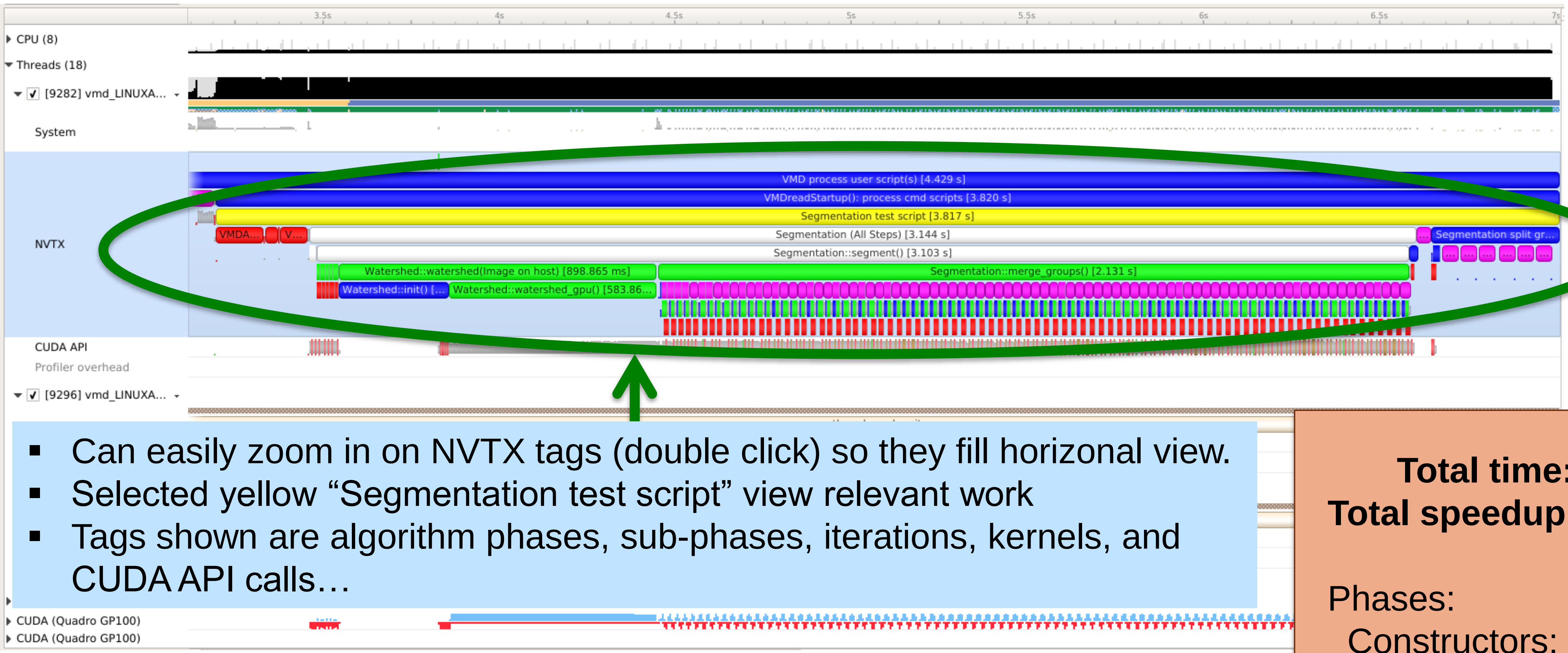
- GPU compute activity shown in BLUE.
- Memory transfer activity shown in RED.
- Trace shows memory transfers taking a lot of the time in the second phase…
- What is the algorithm doing here? Why?

2: VMD PROFILE W/ NVTX TAGS



- Added NVTX tags clearly show algorithm phases in the Nsight System timeline.

2: VMD IMAGE SEGMENTATION W/ NVTX

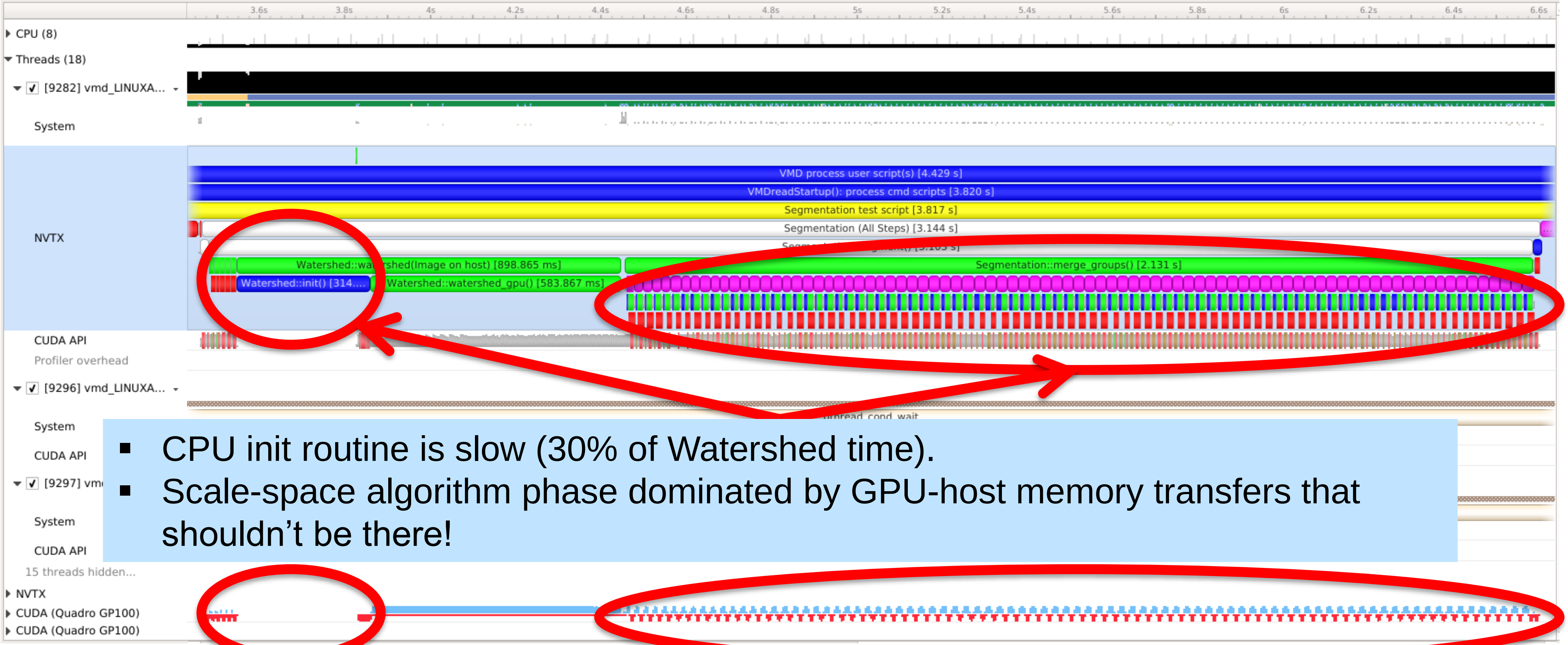


- Can easily zoom in on NVTX tags (double click) so they fill horizonal view.
- Selected yellow “Segmentation test script” view relevant work
- Tags shown are algorithm phases, sub-phases, iterations, kernels, and CUDA API calls…

Total time: 3.14s
Total speedup: 1.0x

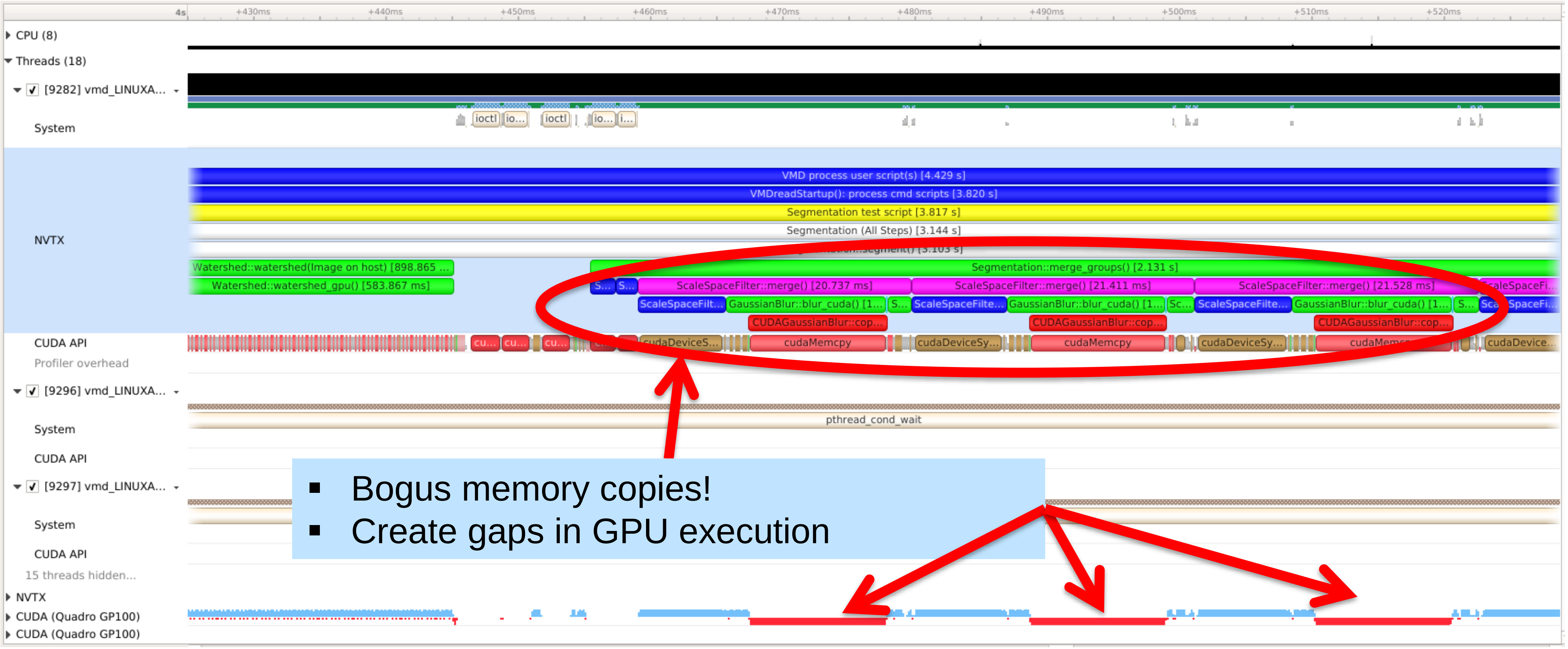
Phases:
Constructors: 0.1s
Watershed: 0.9s
Scale-Space: 2.13s
Other: 0.014s

2: IDENTIFIED BOGUS COPIES, SLOW CPU INIT

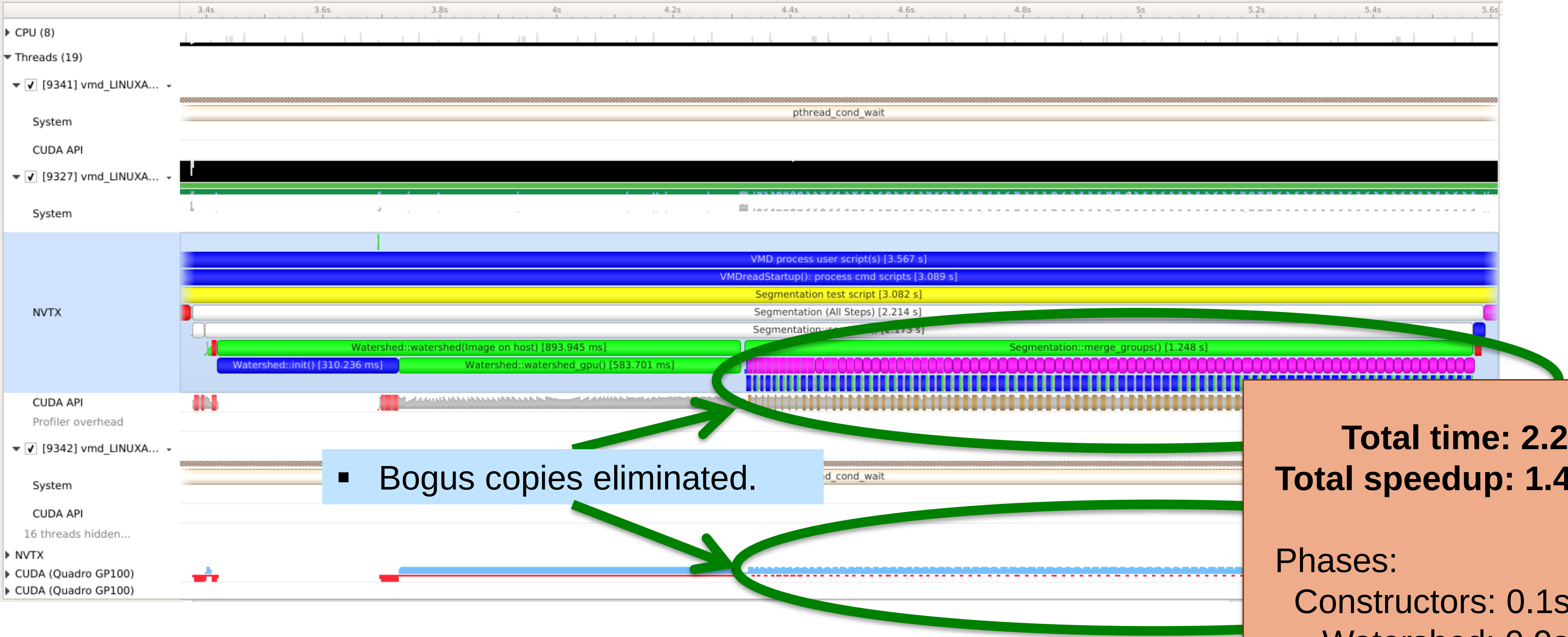


- CPU init routine is slow (30% of Watershed time).
- Scale-space algorithm phase dominated by GPU-host memory transfers that shouldn’t be there!

2: DETAIL: IDENTIFIED BOGUS GPU-HOST COPIES



3: BOGUS COPIES ELIMINATED

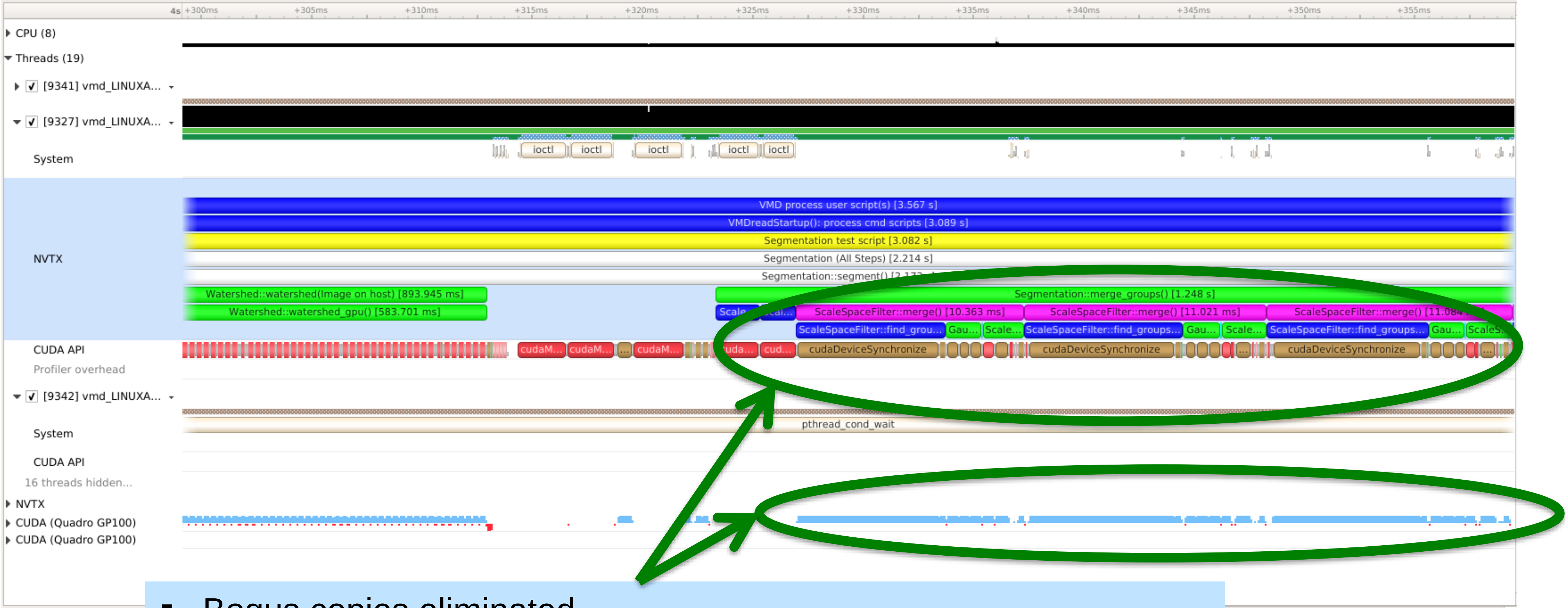


- Bogus copies eliminated.

Total time: 2.21s
Total speedup: 1.4x

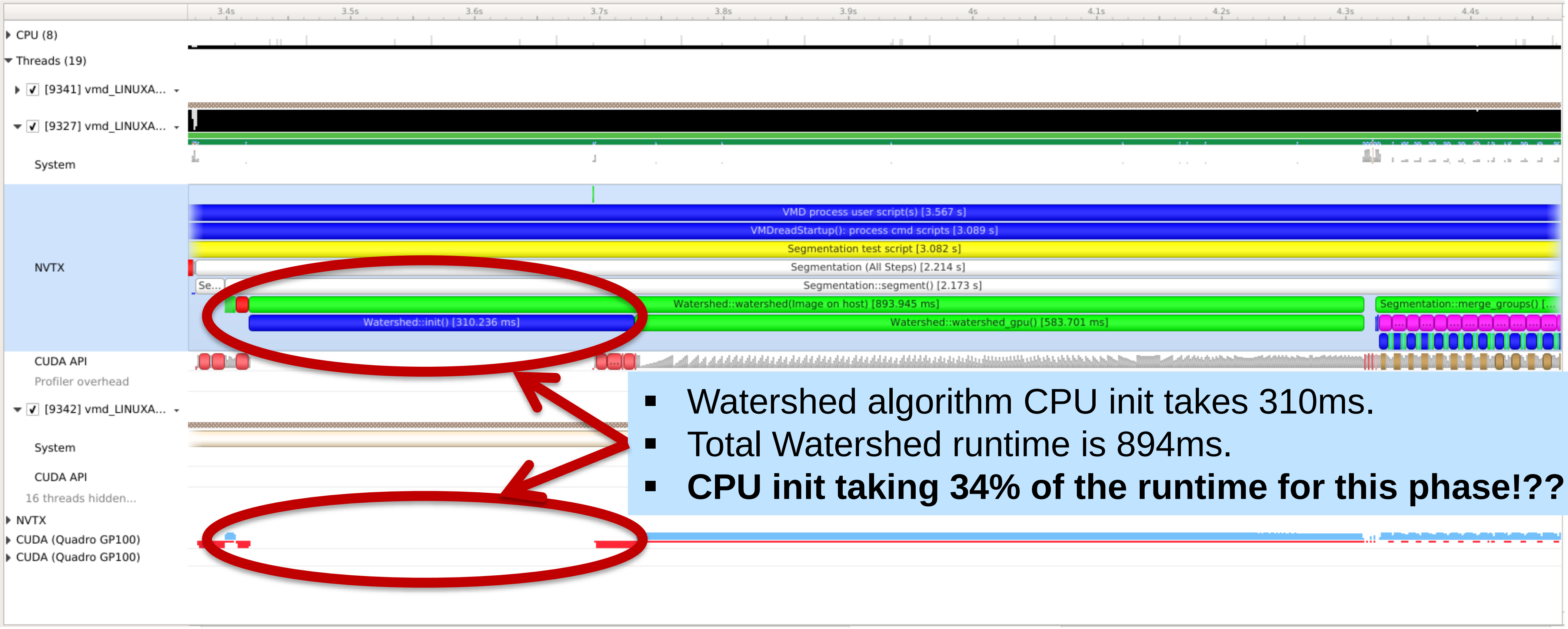
Phases:
Constructors: 0.1s
Watershed: 0.9s
Scale-Space: 1.25s
Other: -

3: DETAIL: BOGUS COPIES ELIMINATED



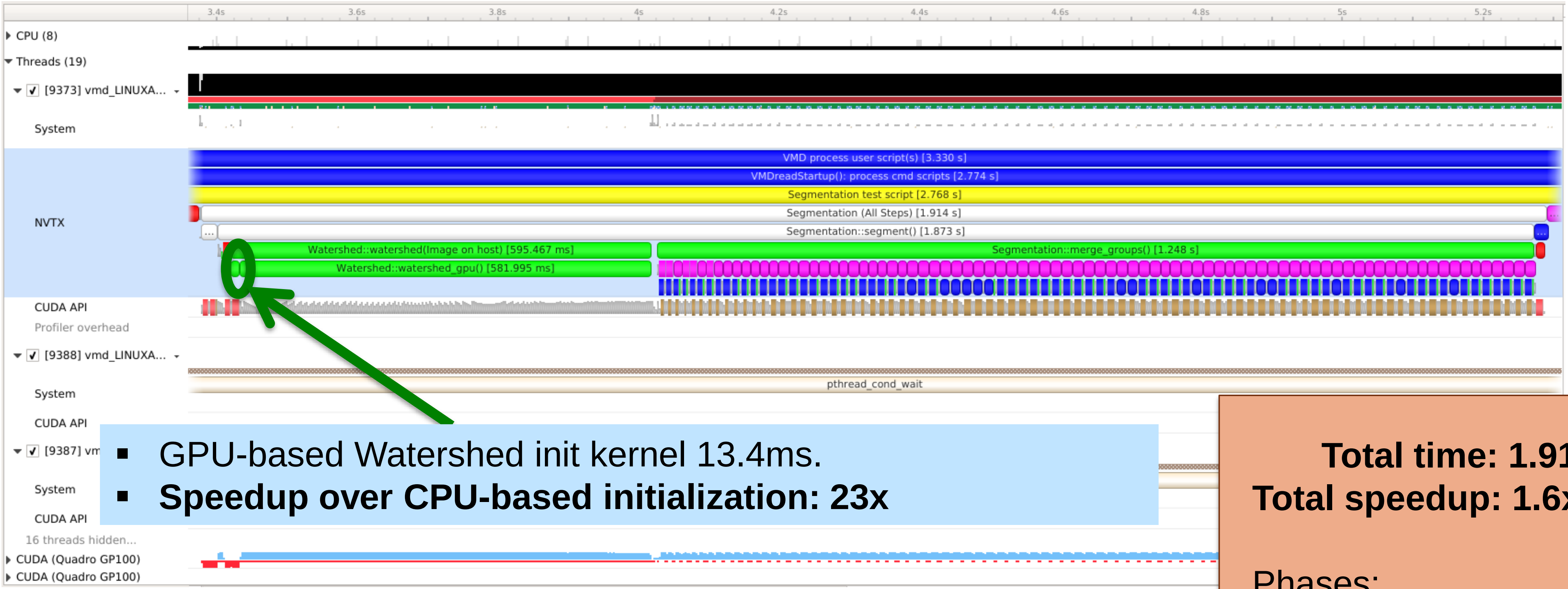
- Bogus copies eliminated.
- Gaps between GPU kernels are now very short.
- **Speedup for just the scale-space algorithm phase is 1.7x**

3: DETAIL: SLOW CPU INIT ROUTINE



- Watershed algorithm CPU init takes 310ms.
- Total Watershed runtime is 894ms.
- **CPU init taking 34% of the runtime for this phase!??!**

4: FAST GPU INIT ROUTINE

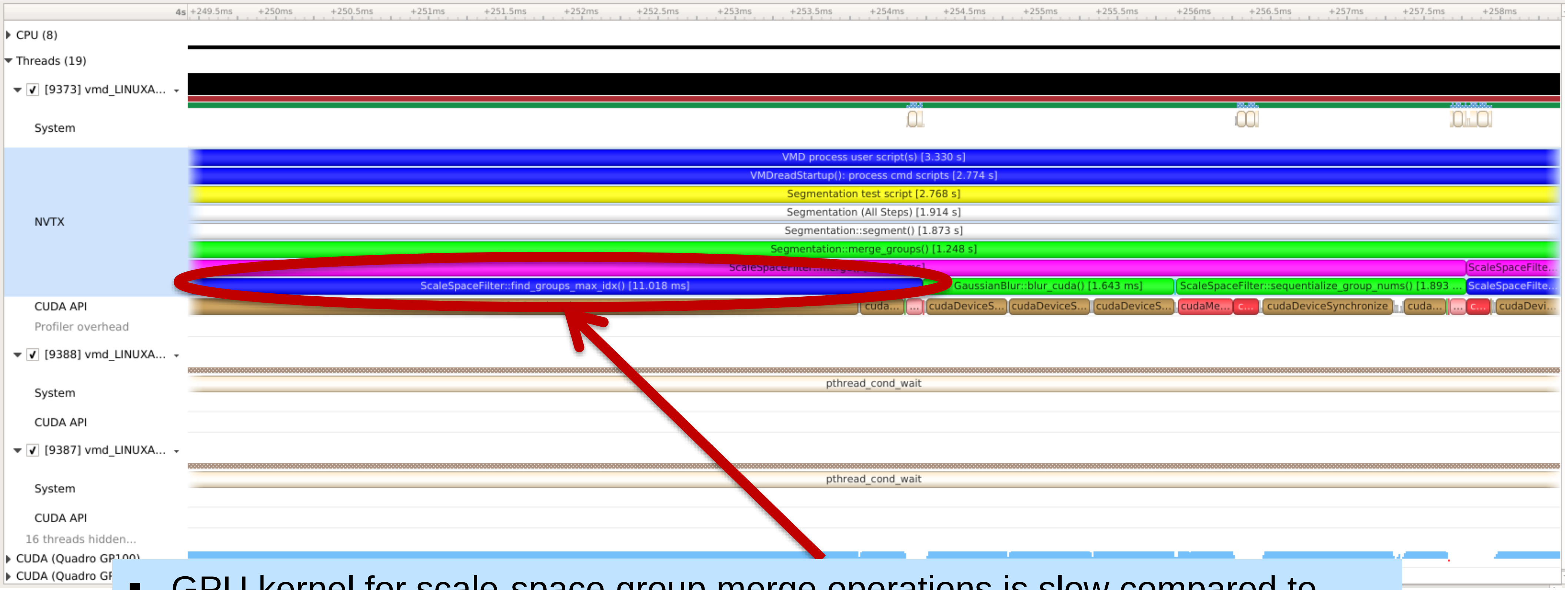


- GPU-based Watershed init kernel 13.4ms.
- **Speedup over CPU-based initialization: 23x**

Total time: 1.91s
Total speedup: 1.6x

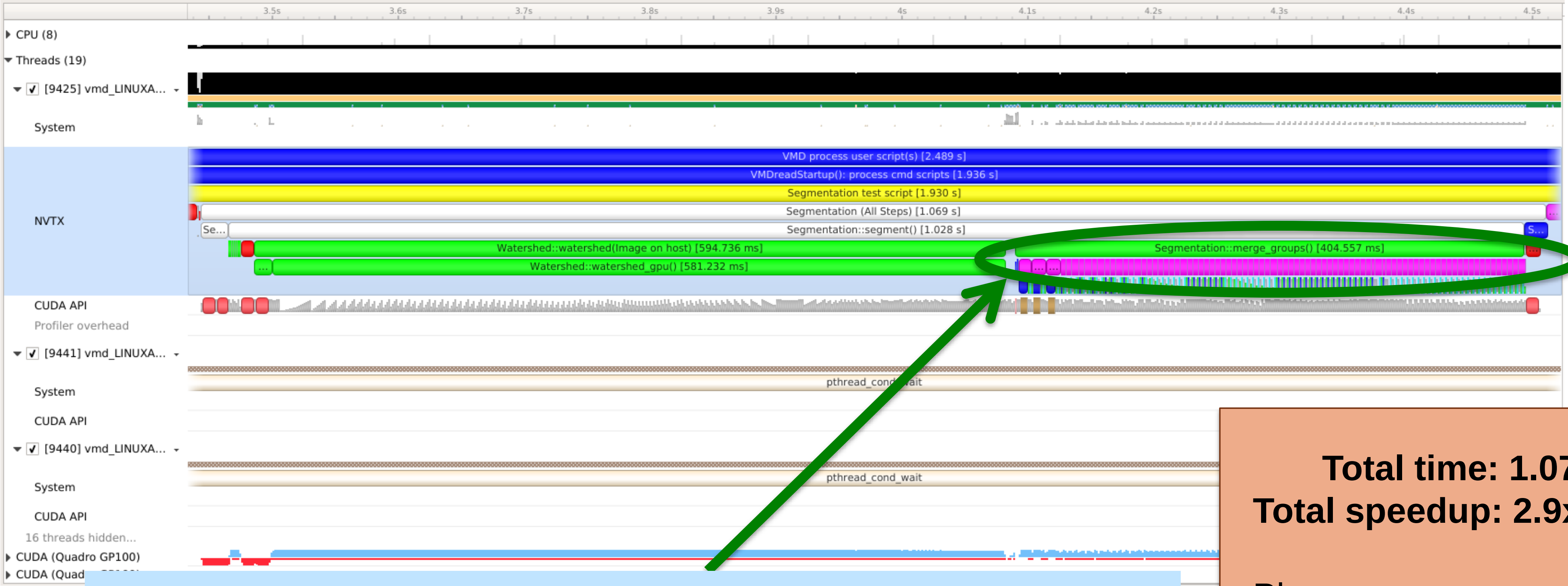
Phases:
Constructors: 0.1s
Watershed: 0.59s
Scale-Space: 1.25s
Other: -

4: DETAIL: SLOW SCALE-SPACE MERGE GROUPS KERNEL



- GPU kernel for scale-space group merge operations is slow compared to other kernels, opportunity!
- Write new special-case scale-space merge kernels for problem sizes small enough to allow atomic ops in shared memory rather than global memory.

5: FASTER MERGE GROUPS KERNELS

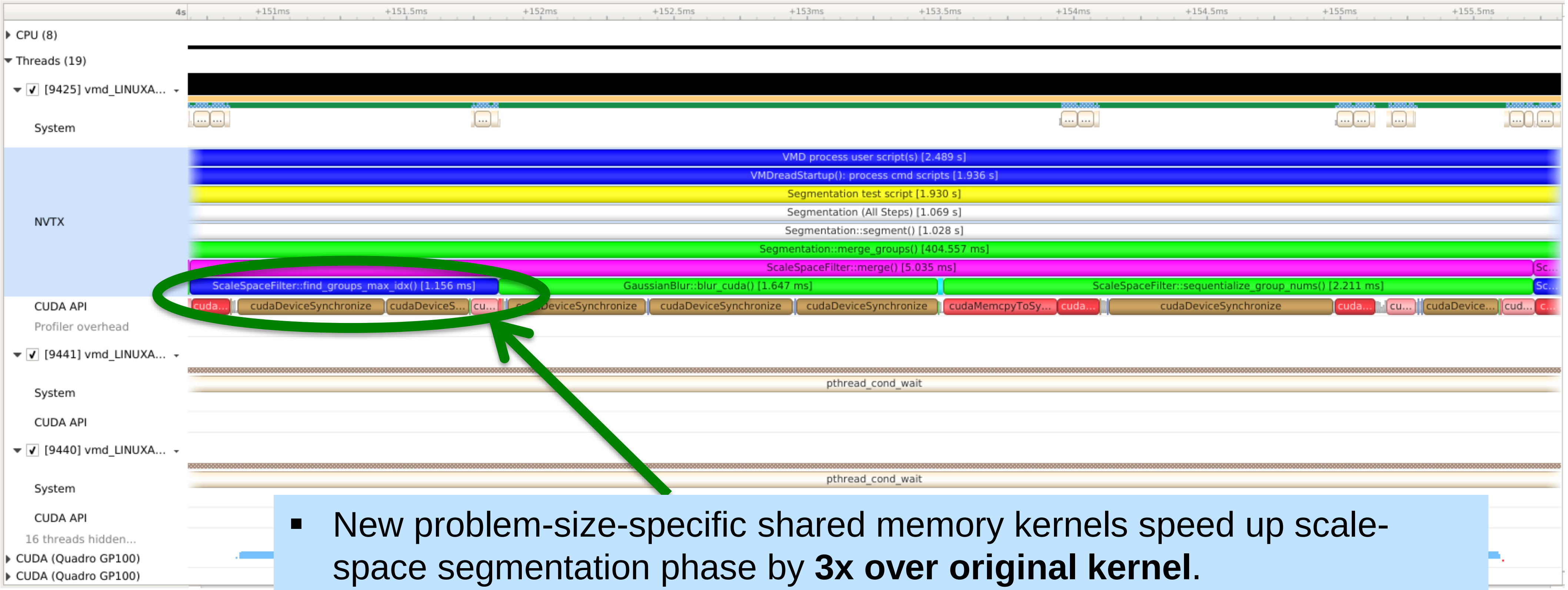


- Use Nsight Compute to examine kernel in detail.
- New problem-size-specific shared memory kernels speed up scale-space segmentation phase by **3x over original kernel.**

Total time: 1.07s
Total speedup: 2.9x

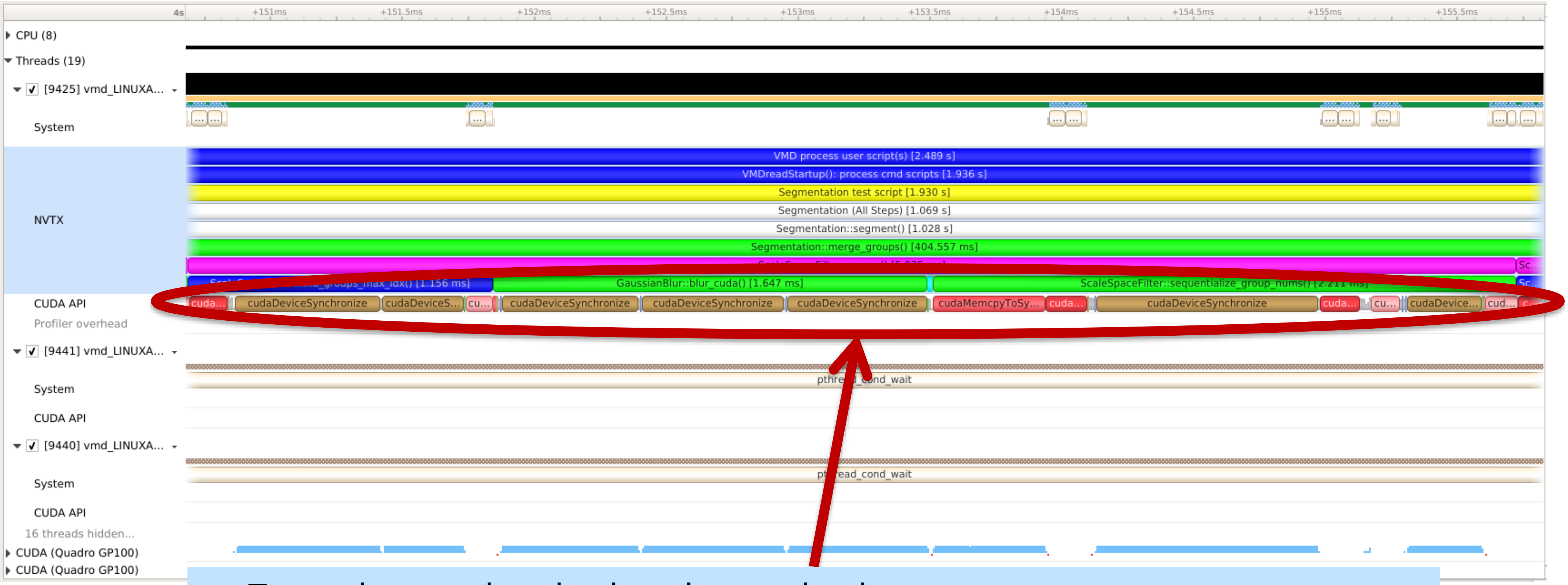
Phases:
Constructors: 0.1s
Watershed: 0.59s
Scale-Space: 0.4s
Other: -

5: DETAIL: FASTER MERGE GROUPS KERNELS



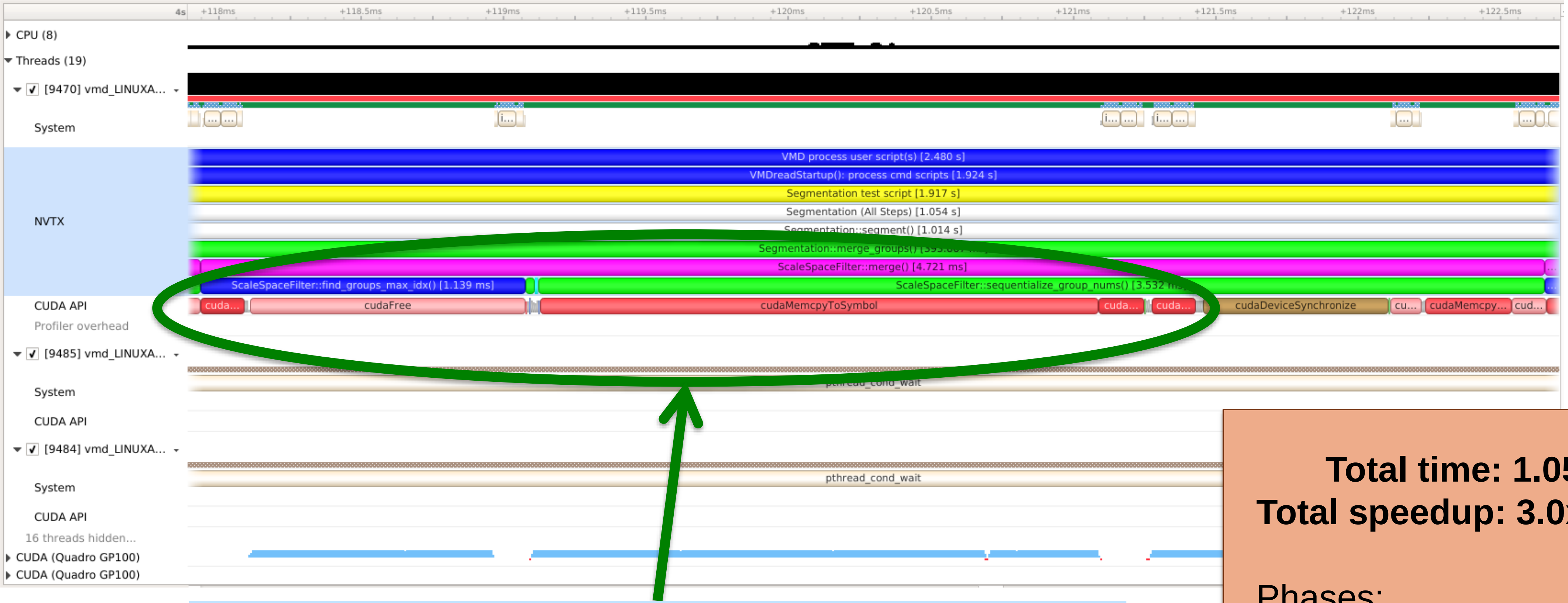
- New problem-size-specific shared memory kernels speed up scale-space segmentation phase by **3x over original kernel**.
- **New kernels have comparable runtime to neighboring scale space kernels, no longer an outstanding optimization opportunity.**

5: DETAIL: EXCESSIVE ERROR CHECKING



- Excessive synchronizations happening here
- Many calls to cudaDeviceSynchronize(), checking error status, etc.

6: DETAIL: STREAMLINED ERROR CHECKING

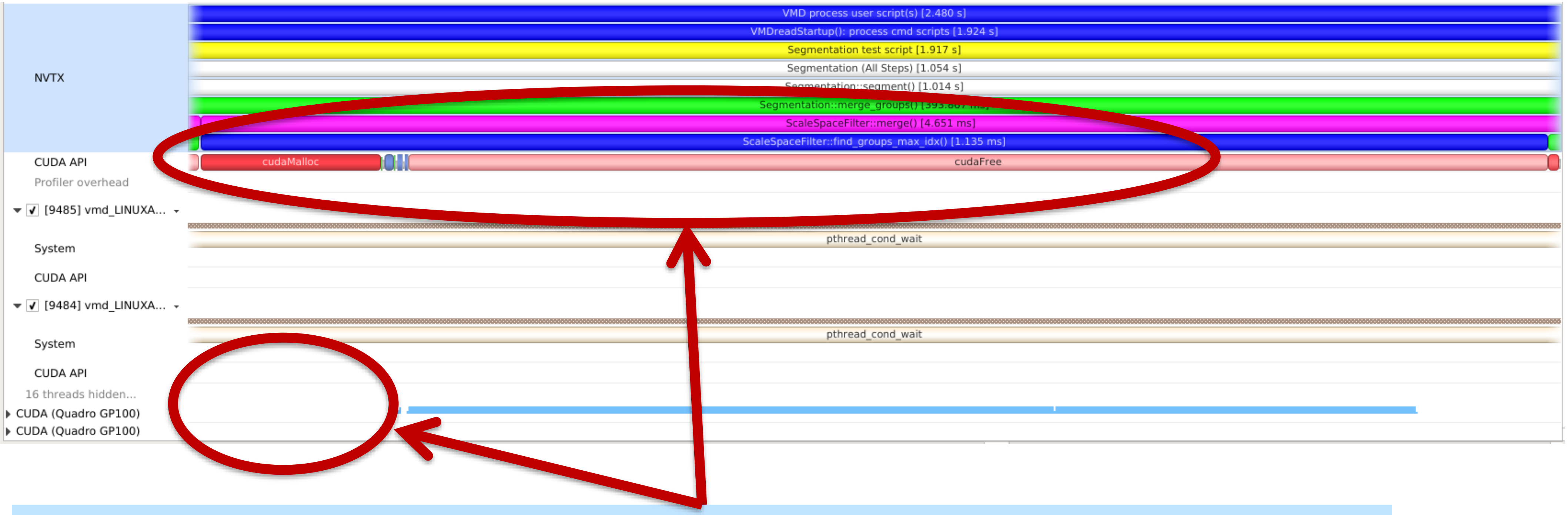


- Excessive cudaDeviceSynchronize() calls eliminated

Total time: 1.05s
Total speedup: 3.0x

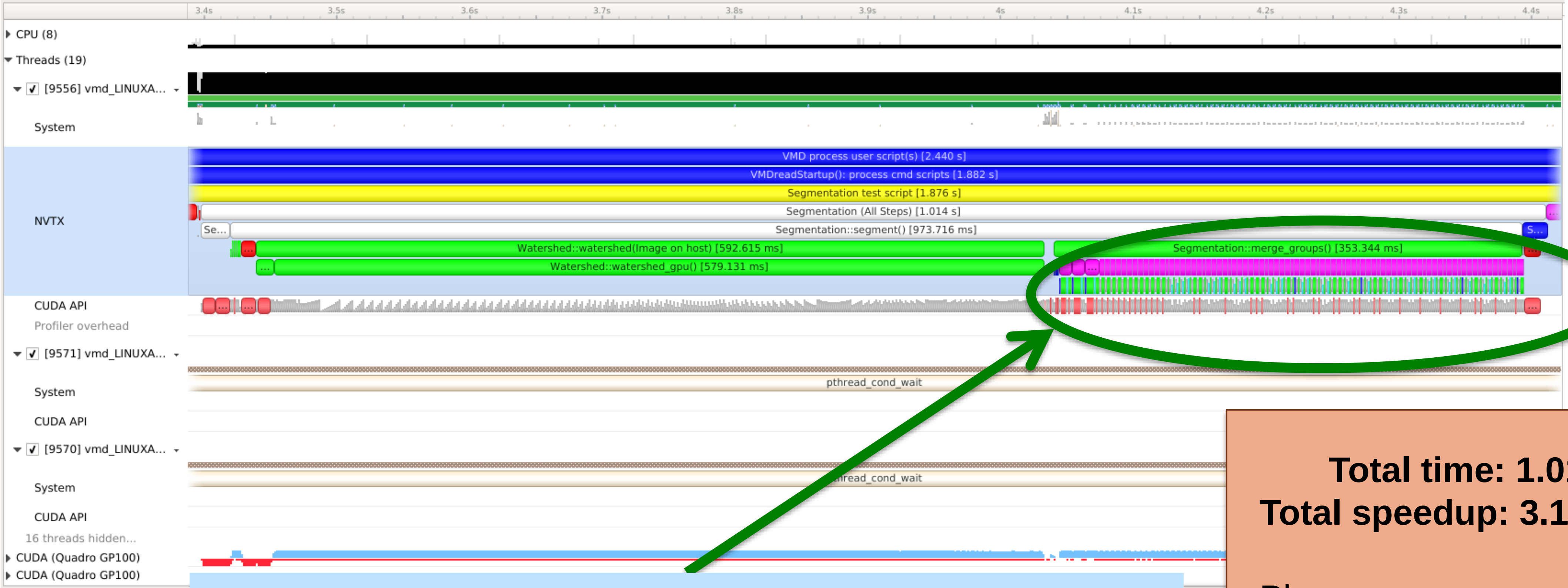
Phases:
Constructors: 0.1s
Watershed: 0.59s
Scale-Space: 0.4s
Other: -

6: DETAIL: ITERATED MALLOC/FREE



- Several areas of trace show CUDA malloc/free calls in iterative algorithm phase
- (Re)allocation APIs create gaps in GPU execution stream

7: MADE WORK BUFFERS PERSISTENT

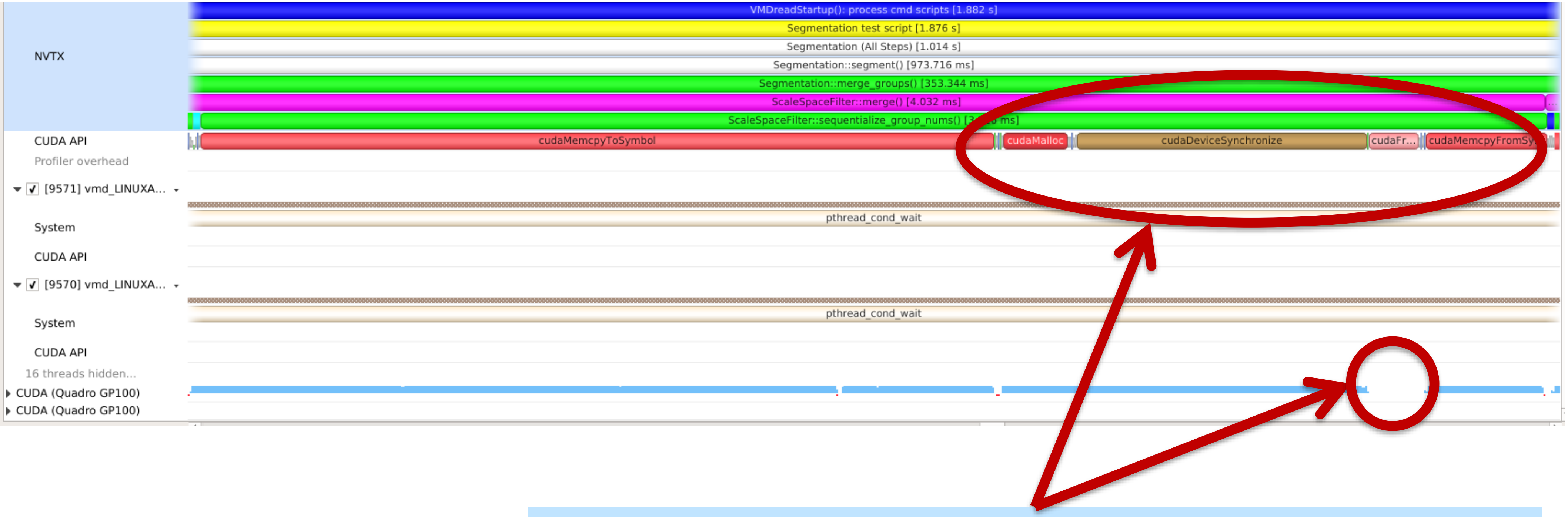


- All VMD kernel work buffers persistent across iterations

Total time: 1.01s
Total speedup: 3.1x

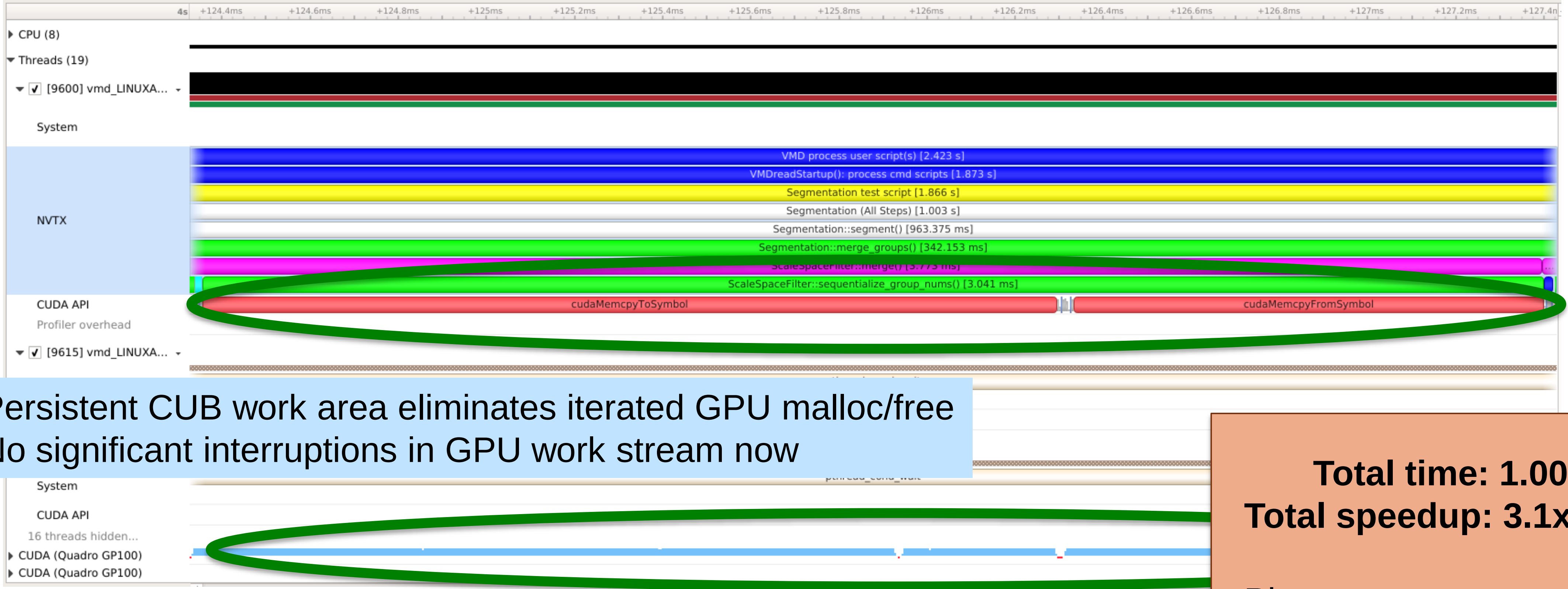
Phases:
Constructors: 0.1s
Watershed: 0.57s
Scale-Space: 0.35s
Other: -

7: DETAIL: … EXCEPT THRUST SCAN()



- Thrust scan performs GPU malloc/free
- Allocations disrupt GPU work stream slightly
- Can use special allocation scheme or use CUB instead

8: DETAIL: CUB SCAN PERSISTENT WORK BUFFERS

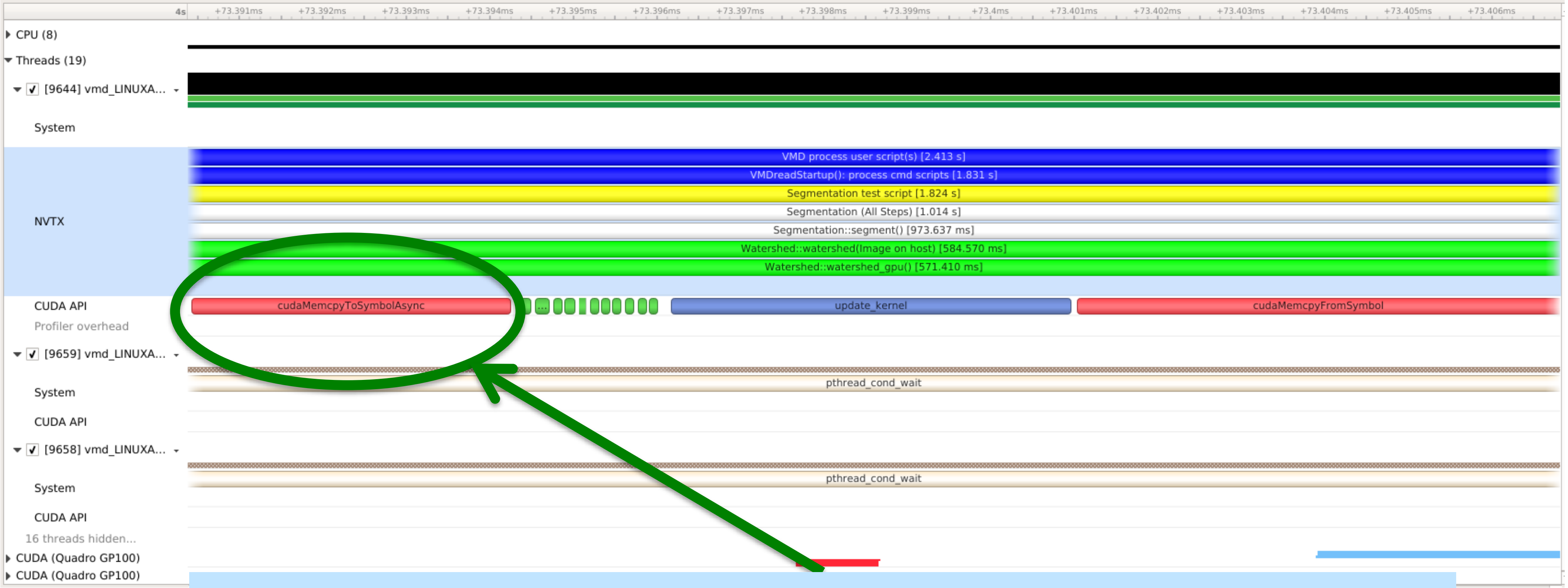


- Persistent CUB work area eliminates iterated GPU malloc/free
- No significant interruptions in GPU work stream now

Total time: 1.00s
Total speedup: 3.1x

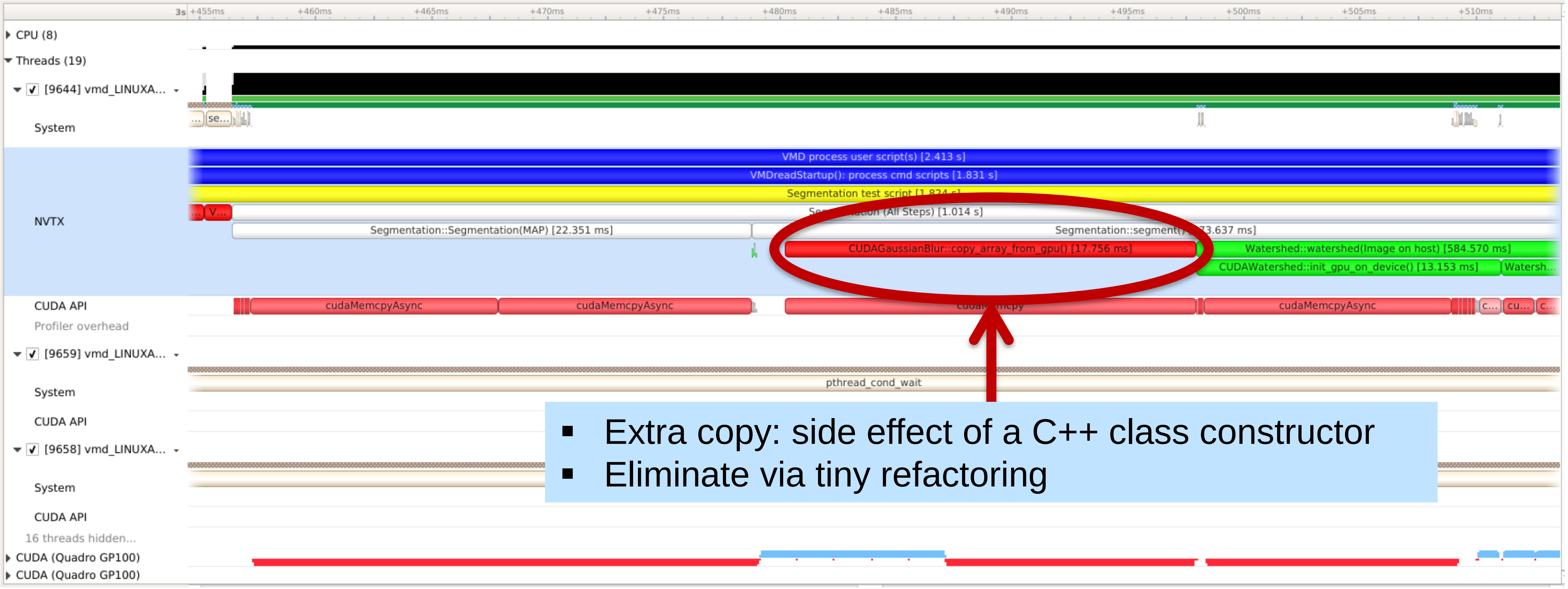
Phases:
Constructors: 0.1s
Watershed: 0.57s
Scale-Space: 0.35s
Other: -

9: DETAIL: USE OF CUDA ASYNC APIS



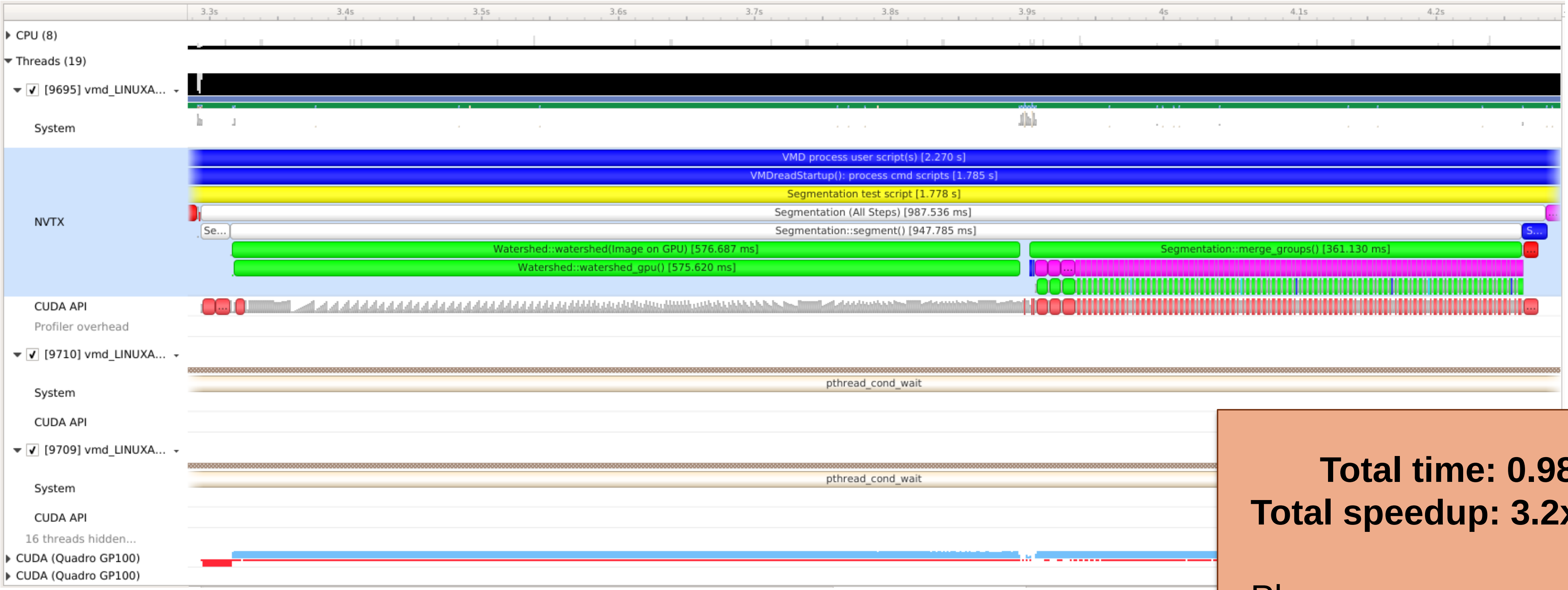
- Use of cudaMemcpyToSymbolAsync() allows CPU to enqueue subsequent kernel and result copy-back efficiently while first copy is still running

9: DETAIL: CONSTRUCTOR HOST-GPU COPY



- Extra copy: side effect of a C++ class constructor
- Eliminate via tiny refactoring

10: FINAL RESULT



Total time: 0.98s
Total speedup: 3.2x

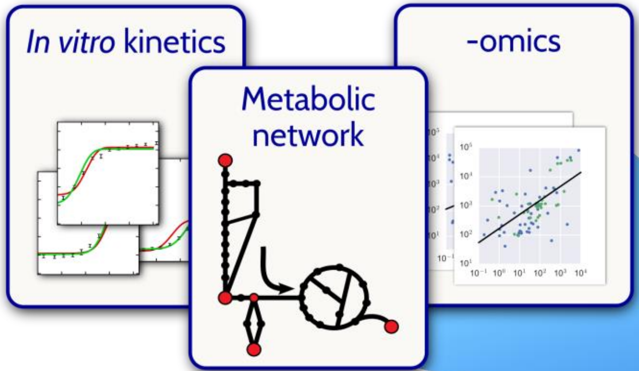
Phases:
Constructors: 0.03s
Watershed: 0.57s
Scale-Space: 0.36s
Other: -

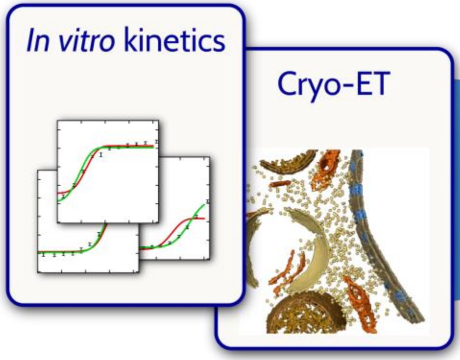
VMD CRYO-EM SEGMENTATION: LESSONS LEARNED

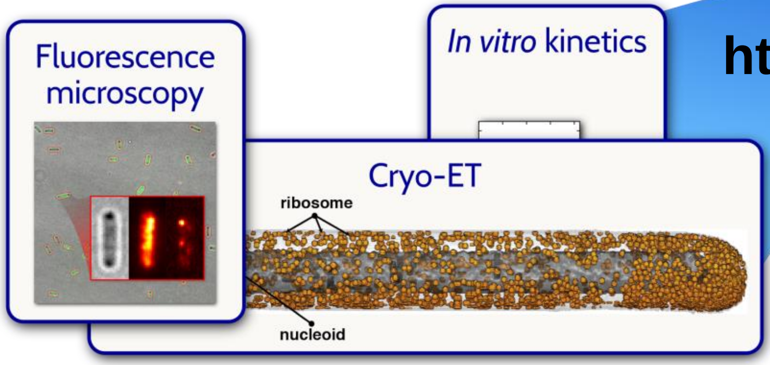
- Nsight Sytems helped identify unintentional copies caused by indirect side-effects of C++ class designs
- Demonstrates the value of applying profiling tool during ongoing algorithm development
- Final performance result on Quadro GP100 is 3.2x faster
- **Speedup on Tesla V100 (Volta) is even more dramatic:**
  - **Initial runtime 2.66 seconds**
  - **Final optimized runtime: 0.64 seconds, 4.1x faster**
- **VMD GPU image segmentation is now 12x faster than competing tools**

Lattice Microbes

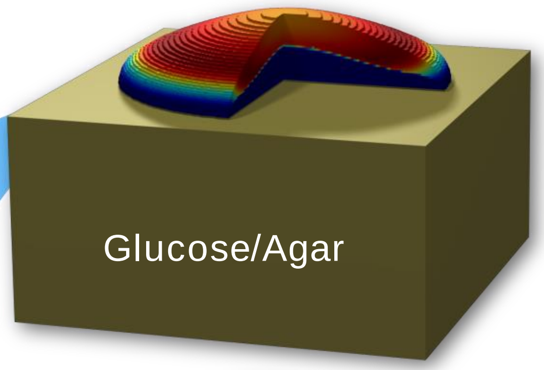
- Whole-cell modeling and simulation, including heterogeneous environments and kinetic network of thousands of reactions
- Incorporate multiple forms of experimental imaging for model construction
- Scriptable in Python

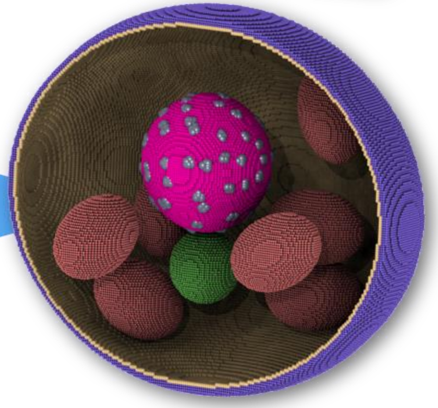






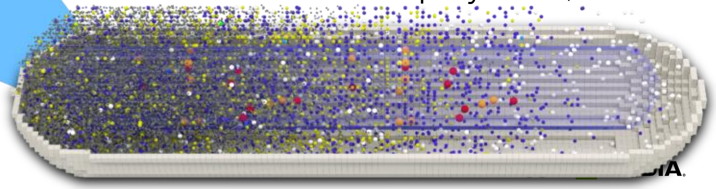
BMC Sys. Biol. 2015





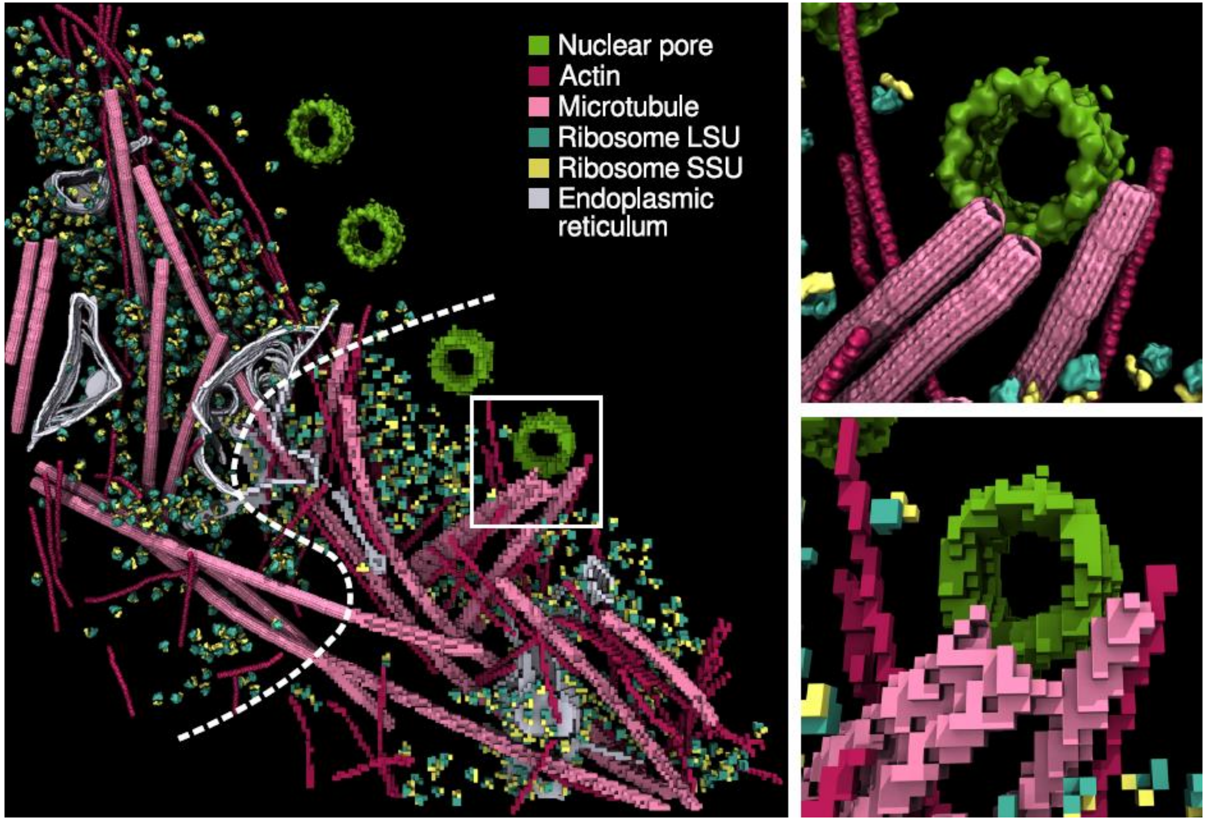
http://www.scs.illinois.edu/schulten/lm

Biophys. J. 2015
Biopolymers, 2016



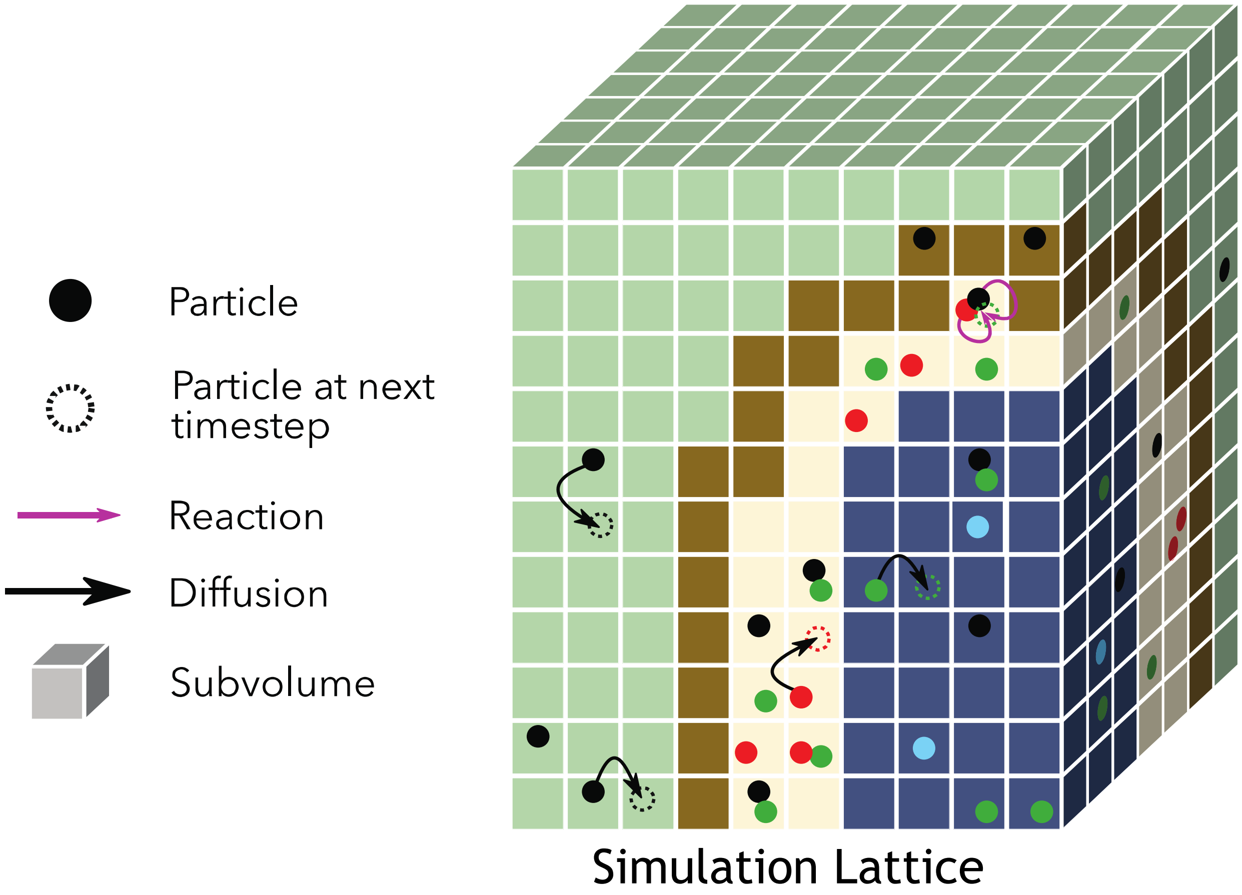
LATTICE MICROBES DESCRIPTION

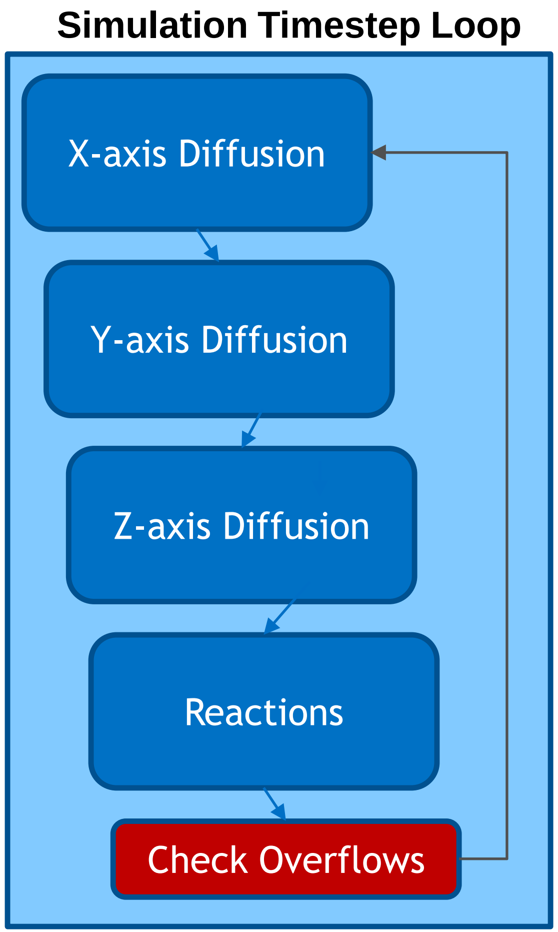
Simulate cell dynamics on biologically relevant timescales using a lattice-based model



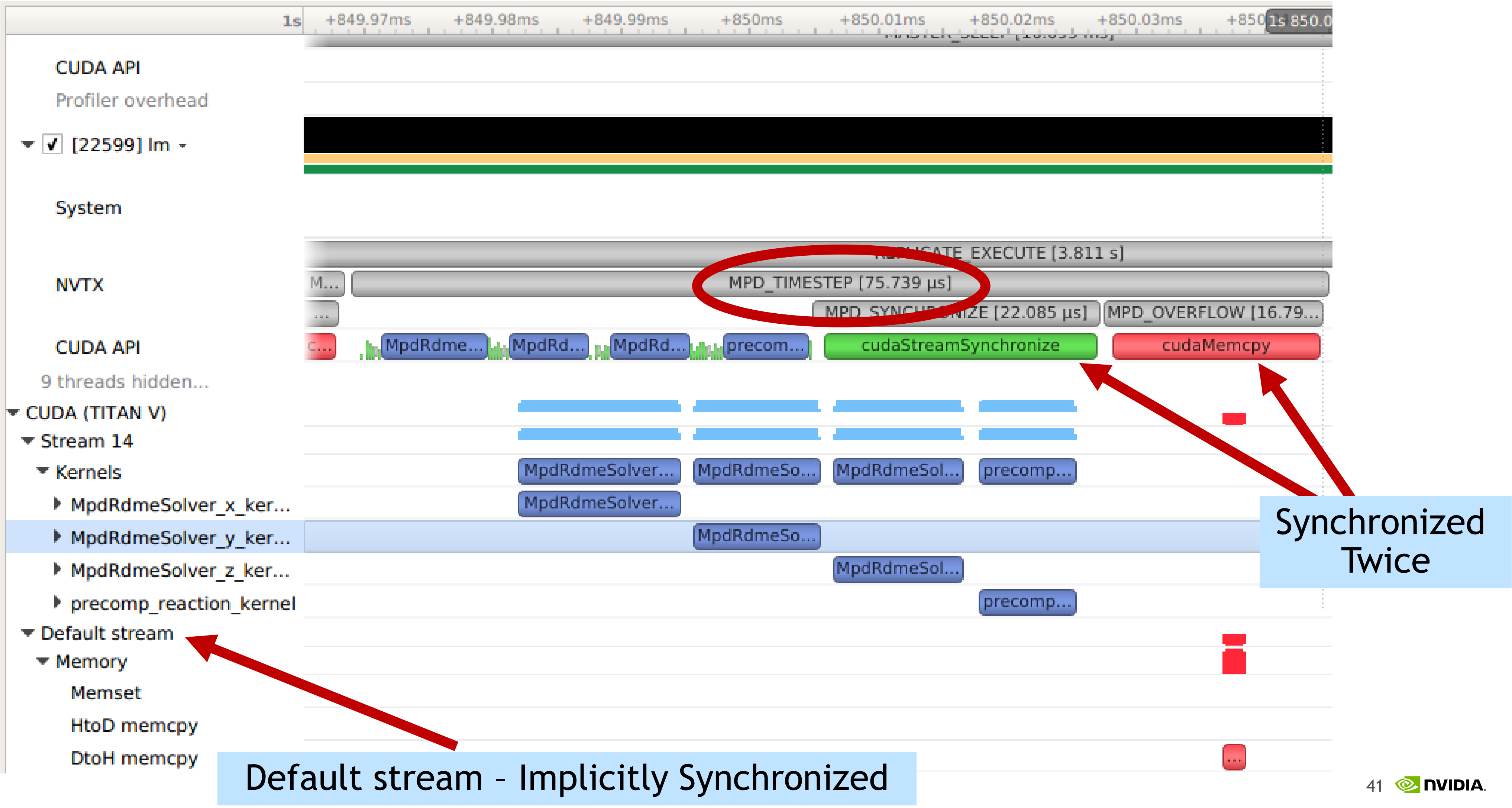
Earnest, et al. J. Physical Chemistry B, 121(15): 3871-3881, 2017.

LATTICE MICROBES SIMULATION

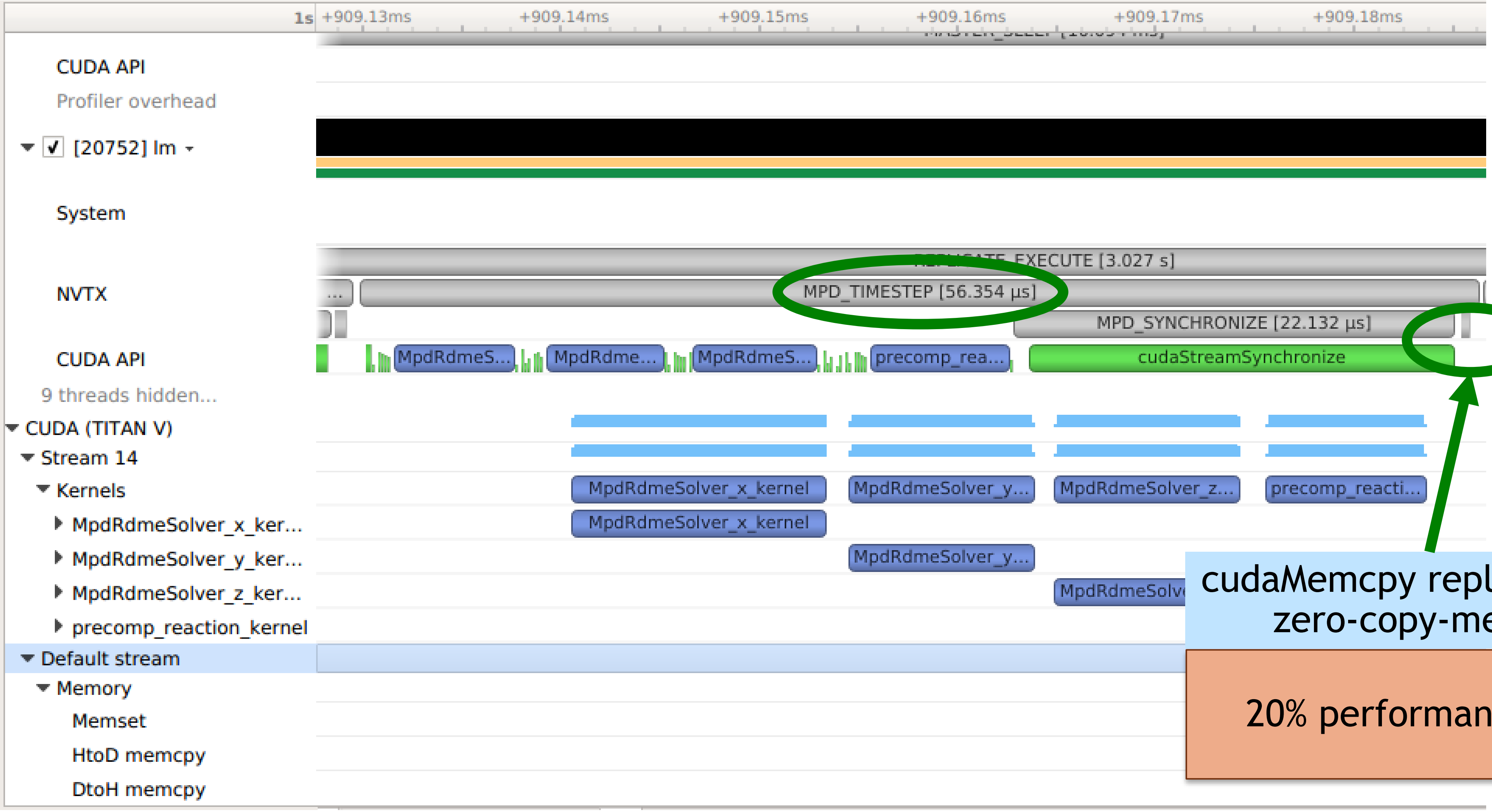




DEFAULT STREAM SYNCHRONIZATION



DEFAULT STREAM SYNCHRONIZATION 2

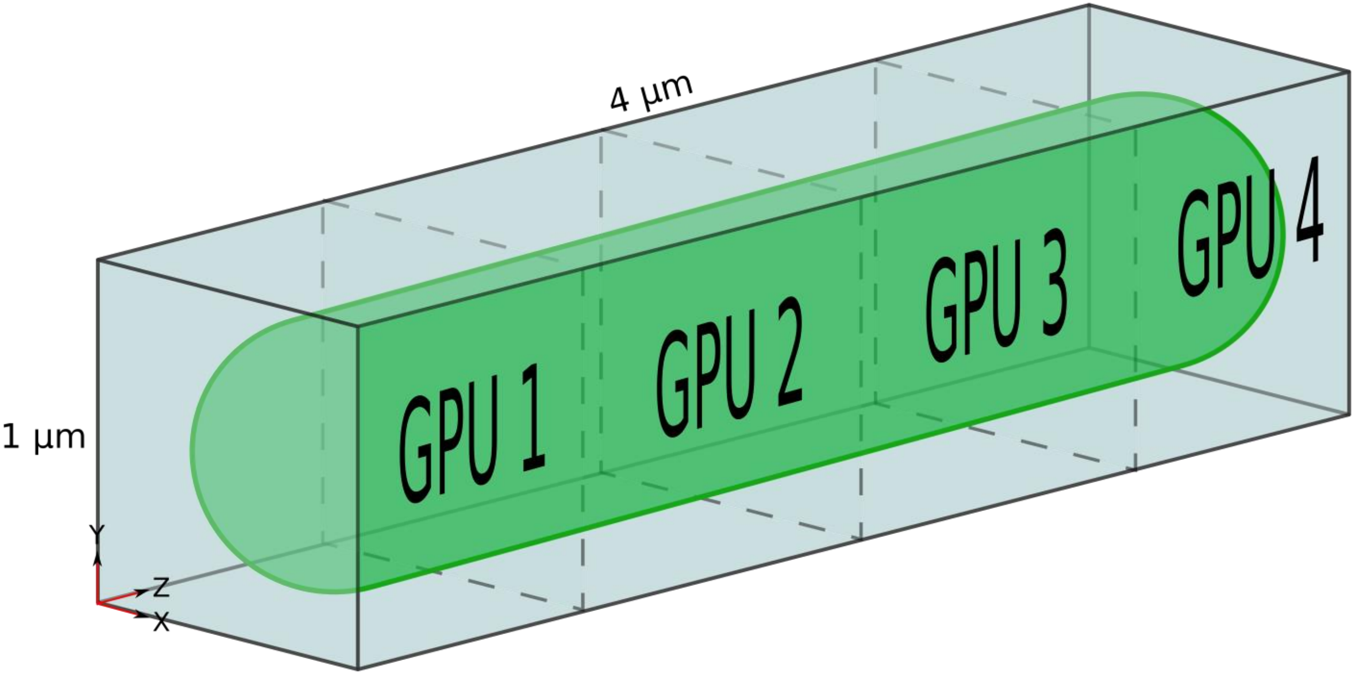


cudaMemcpy replaced with zero-copy-memory

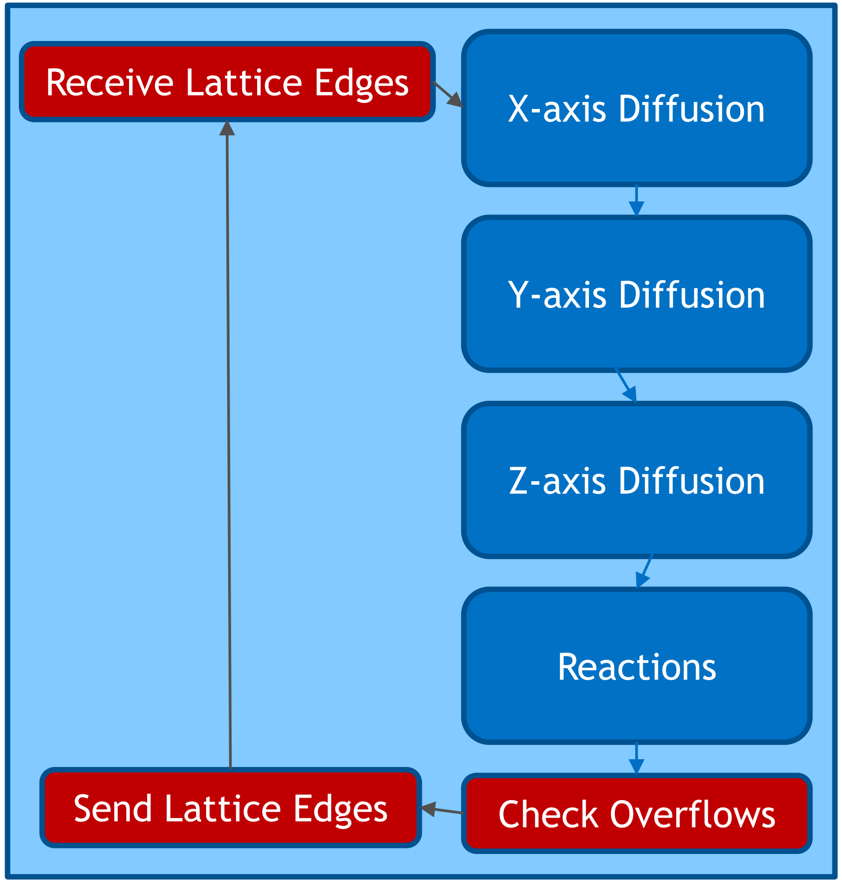
20% performance gain

LATTICE MICROBES MULTI-GPU SIMULATION

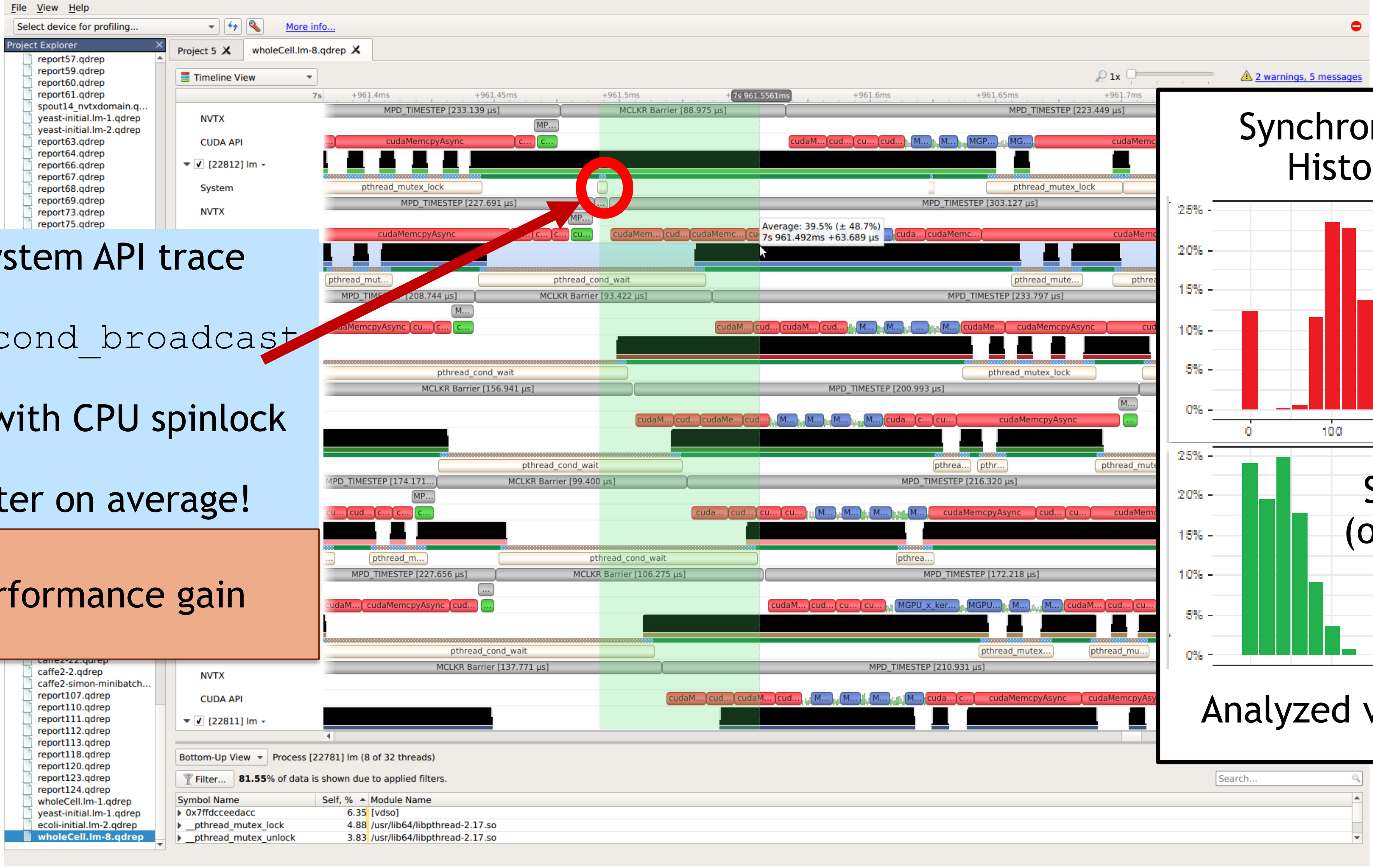
- Divide the cell into chunks for each GPU to process
- Communicate particles on the edge of each volume to neighboring GPU



Simulation Timestep Loop



HOST THREAD PARALLEL OPTIMIZATION



Using System API trace

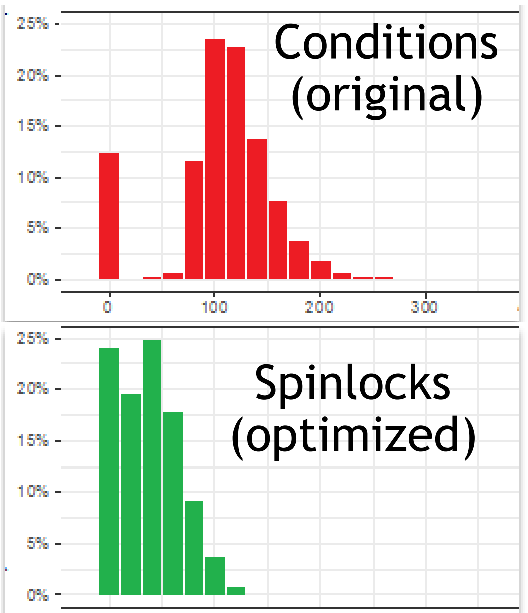
`pthread_cond_broadcast`

Replaced with CPU spinlock

60us faster on average!

~25% performance gain

Synchronization Histogram



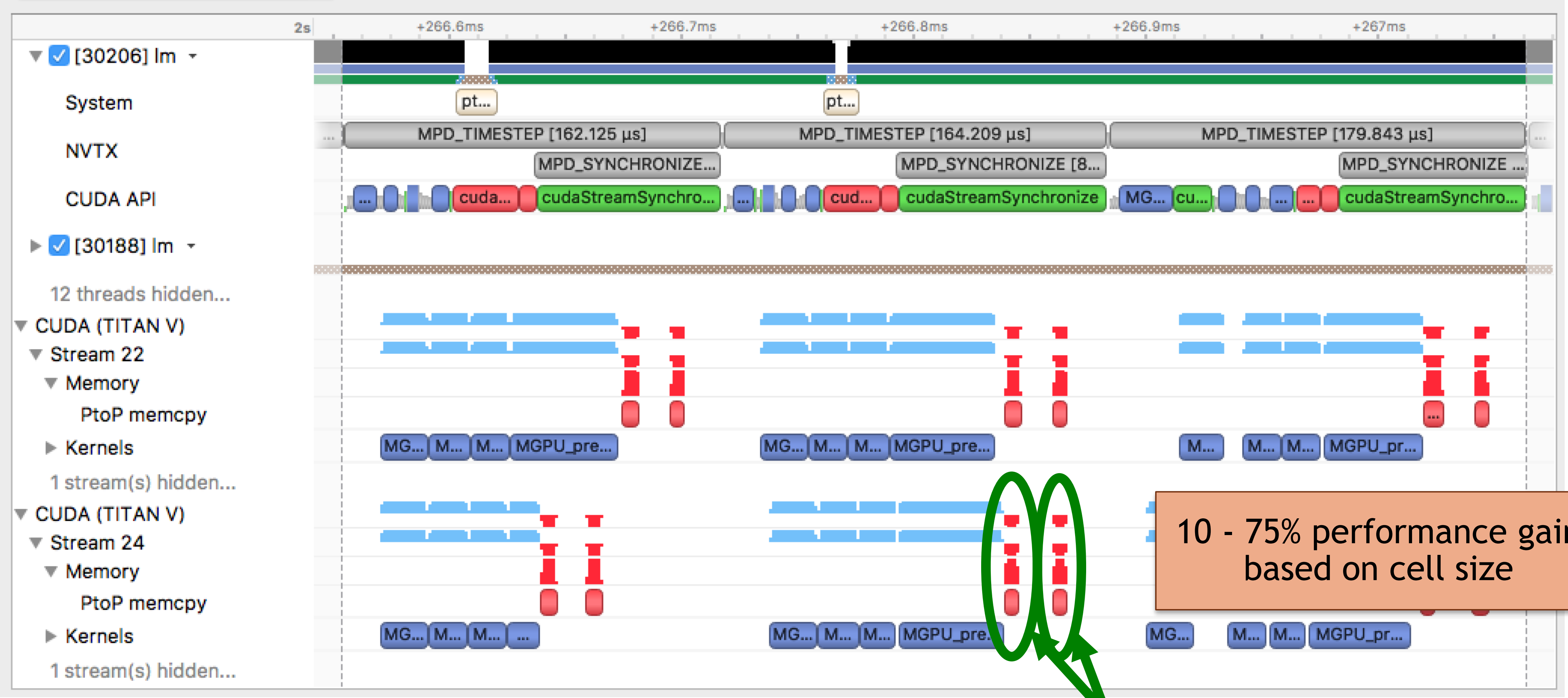
Analyzed via Python

INTER-GPU TRANSFER IMPROVEMENTS

Eliminate small D2D Copies

Reduce transfer overhead by packing multiple transfers in to one

INTER-GPU TRANSFER IMPROVEMENTS

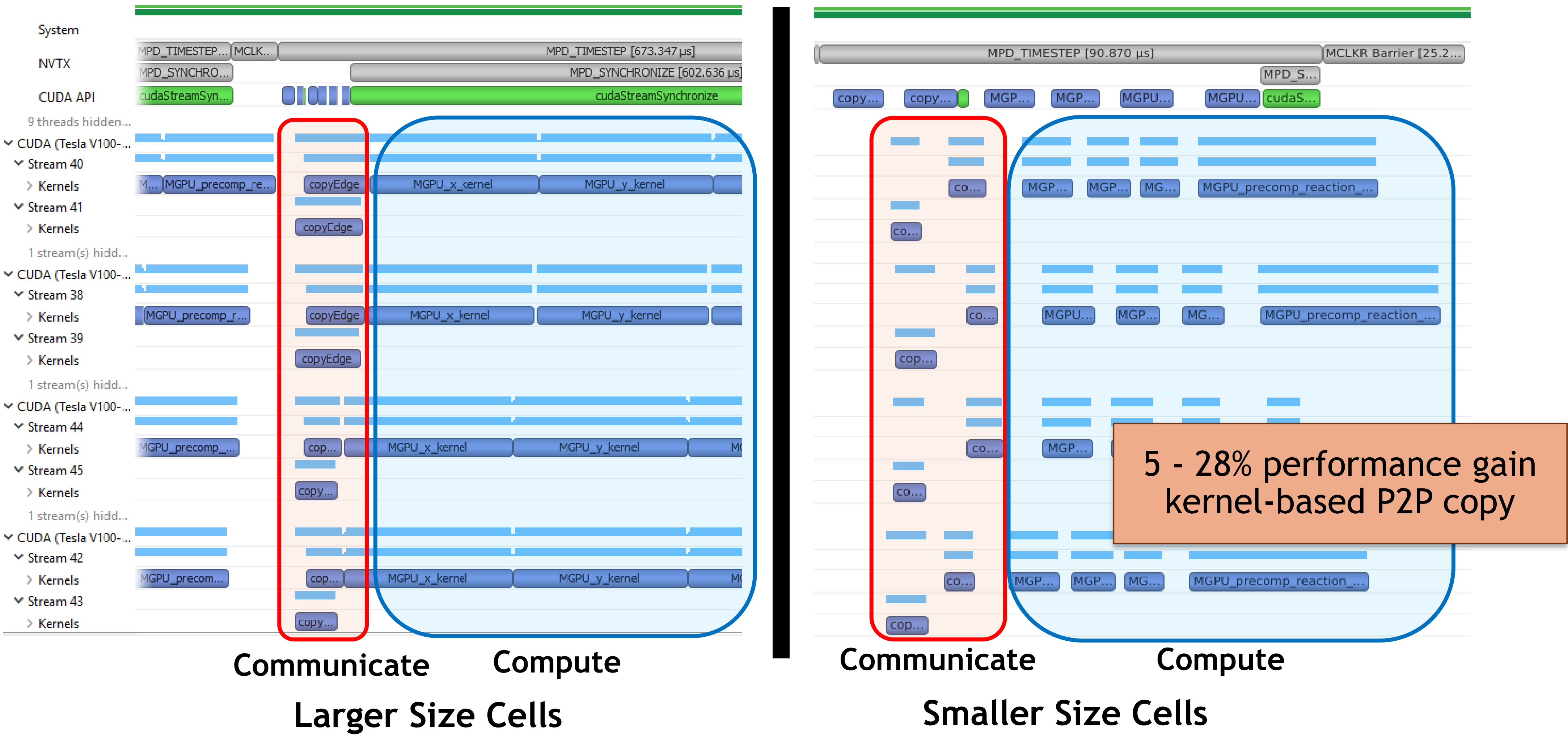


10 - 75% performance gain based on cell size

Only one copy to each neighbor

NVIDIA.

4-WAY DGX INTER-GPU TRANSFERS WITH NVLINK



A kernel directly accessing remote lattice via P2P copies can achieve concurrent bidirectional transfers

COMMON OPTIMIZATION OPPORTUNITIES

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

TOOL COMPARISON

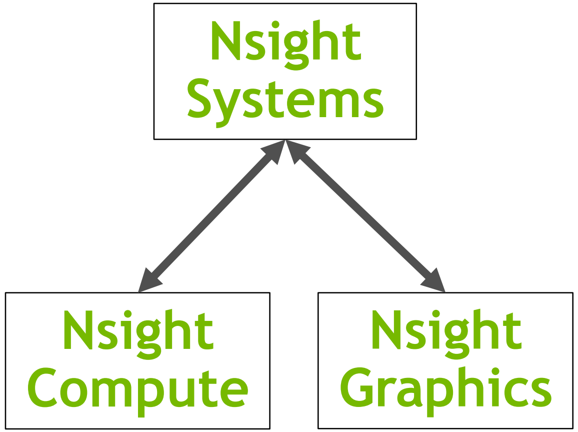
	NVIDIA © Nsight ™ Systems	NVIDIA© Nsight™ Compute	NVIDIA© Visual Profiler	Intel © VTune ™ Amplifier	Linux perf OProfile
Target OS	**Linux**	Linux, Windows	Linux, Mac, Windows	Linux, Windows	Linux
GPUs	**Pascal, Volta, Future**	Pascal, Volta, Future	Kepler, Maxwell, Pascal, Volta, Future	None	None
CPUs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

NSIGHT SYSTEMS

Visit us at the NVIDIA booth in the exhibit hall for a live demo!

- When can you get it?
  - Soon. Fixing the last issues now.
- Where can you get it?
  - http://developer.nvidia.com/nsight-systems
- Questions/Requests/Comments?
  - nsight-systems@nvidia.com

Workflow



For Tegra-based systems
Codeworks
JetPack
DriveInstall

Note: Currently still
NVIDIA System Profiler
in some packages

NSIGHT SYSTEMS

Upcoming features:

- NVIDIA GPU Cloud (near future)
- Future GPUs
- Future CUDA Releases
- Windows targets
- Many more HPC and cluster features

DON'T MISS THESE PRESENTATIONS

S8481: CUDA Kernel Profiling: Deep-Dive Into NVIDIA's Next-Gen Tools (Thursday 11:00AM)

S8337: NVIDIA SDK Manager - Simplify Your Development Environment Setup (Wednesday 3:30 PM)

S8275: Introducing NVIDIA's New Graphics Debugger (Wednesday 4:00 PM)

S8665: VMD: Biomolecular Visualization from Atoms to Cells Using Ray Tracing, Rasterization, and VR (Thursday 11:00AM)

S8709: Accelerating Molecular Modeling Tasks on Desktop and Pre-Exascale Supercomputers (Monday 4:00PM)

*Show floor **demos** available:*

Tuesday 11-1 and 5:30-7:30; Wednesday 12-2 and 5-7; Thursday 12-2

Q & A

BACKUP

COMMAND LINE INTERFACE

```
usage: sp profile [<args>] [application] [<application args>]
args:
        -y, --delay=
           Collection start delay in seconds. Default is 0.

        -d, --duration=
           Collection duration in seconds. Default is 10 seconds.

        -e, --env-var=
           Set environment variable(s) for application process to be launched.
           Environment variable(s) should be defined as 'A=B'. Multiple environment variables can be specified as 'A=B,C=D'

        -h, --help
           This help message.

        -n, --inherit-environment=
           Inherit environment variables. Possible values are 'true' or 'false'. Default is 'true'.

        -o, --output=
           Output QDSTRM filename. Default is report#.qdstrm.

        -s, --sample=
           Select the entity to sample. Possible values are 'cpu' or 'none'. Select 'none' to disable sampling. Default is 'cpu'.

        -b, --backtrace=
           Select the backtrace method to use while sampling. Possible values are 'lbr', 'dwarf', 'fp', or 'none'.
           Select 'none' to disable backtrace collection. Default is 'lbr'.

        -w, --show-output=
           If true, send target process’ stdout and stderr streams to both the console and stdout/stderr files which are added to the QDSTRM file.
           If false, only send target process stdout and stderr streams to the stdout/stderr files which are added to the QDSTRM file.
           Possible values are 'true' or 'false'. Default is 'false'.

        -t, --trace=
           Select the API(s) to trace. Possible values are 'cublas', 'cuda', 'cudnn', 'nvtx', 'opengl', 'system', or 'none'.
           Multiple APIs can be selected, separated by commas only (no spaces). If 'none' is selected, no APIs are traced.
           Default is 'cuda,opengl,nvtx,system'.
```

NVIDIA