TEL AVIV UNIVERSITY

NVIDIA

ORB SLAM2 CUDA Project

M.Sc Student: Eran Moshe

Supervision By Yehonatan Mendel

Autonomous Drones Lab
Faculty of Electrical Engineering, Tel Aviv University

Project Description

Build CUDA Infrastructure for Parallel Programming on Nvidia Jetson TX2

In many applications in embedded devices there are more and more interactions between GPU and CPU to accelerate processes, **CUDA** allows to create efficiently and quickly **Parallel Programs on GPU's** (device side) which controlled by CPU (Host side). As the field of AI grows, there is a growing demand for autonomous surveillance by drones. **Low latency** is very important requirement for drone detection and tracking which can be reached by parallelism of operations.
The main task in the project is to build lab Cuda infrastructure to execute parallel code on **Nvidia Jetson TX2**. Jetson TX2 is the fastest, most power-efficient embedded AI computing device. It's built around an **NVIDIA Pascal™-family GPU** and loaded with 8GB of memory and 59.7GB/s of memory bandwidth.

Project Stages

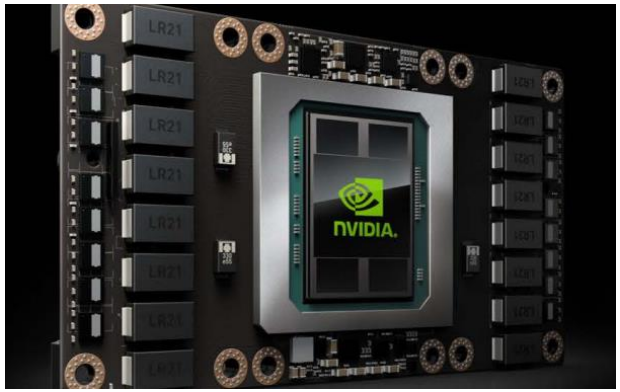
- Internet Research - Overview.
- Learning Architecture and System.
- Learning Cuda Programming.
- Ramp up Jetson TX2 Environment for Running ORB SLAM Application in Linux Environment.
- Research about Nvidia Profiling Tools to Debug Cuda Applications.

Jetson TX2 - Hardware

Jetson TX2 Hardware Specifications

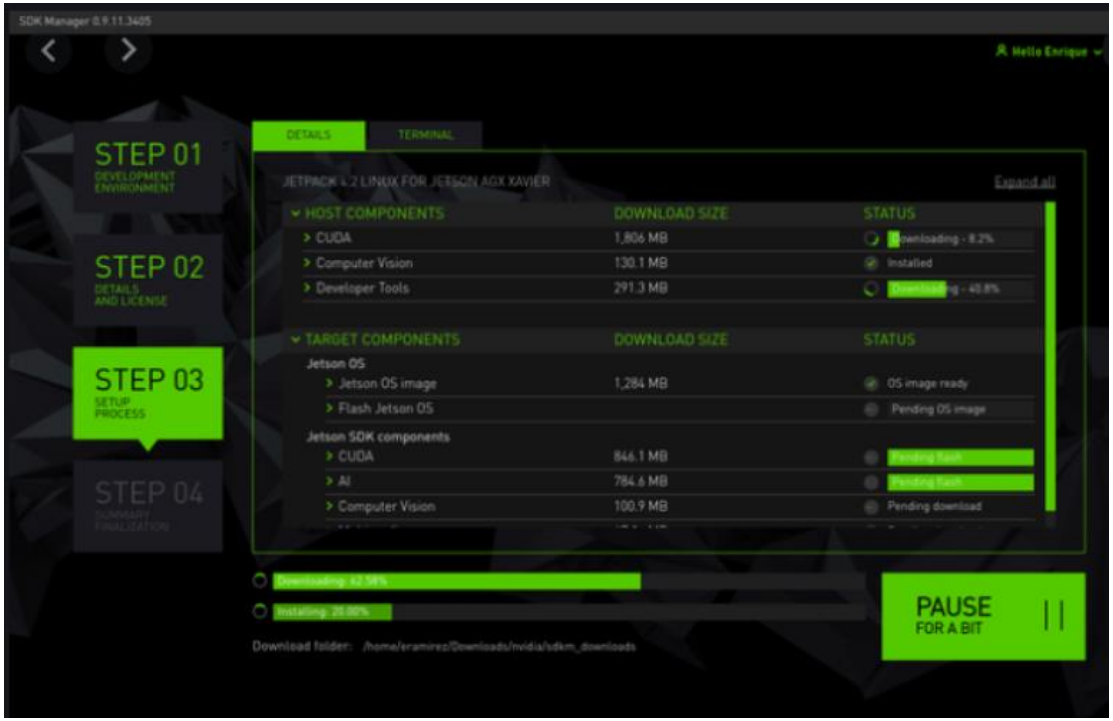
Jetson TX2 is a 7.5-watt supercomputer on a module that brings true AI computing at the edge. It's built around an NVIDIA Pascal™-family GPU and loaded with 8 GB of memory and 59.7 GB/s of memory bandwidth.

- GPU - **NVIDIA Pascal**™ architecture with 256 NVIDIA CUDA cores 1.3 TFLOPS (FP16).
- CPU - Dual-core Denver 2 64-bit CPU and quad-core ARM A57 complex
- Memory - 8 GB 128-bit LPDDR4 1866MHz - 59.7 GB/s
- Storage - 32 GB eMMC 5.1
- Connectivity - 10/100/1000 BASE-T Ethernet



Jetson TX2 - SoftWare

NVIDIA JetPack SDK is a comprehensive resource for building AI applications. It includes Linux Driver Package (L4T) which provides the Linux kernel, bootloader, NVIDIA drivers, flashing utilities, sample filesystem, and more for the Jetson platform tools together with accelerated software libraries, APIs, sample applications, developer tools, and documentation.



ORB SLAM Application

ORB-SLAM2 is a real-time SLAM (Simultaneous Localization And Mapping) library for **Monocular**, **Stereo** and **RGB-D** cameras that computes the camera path and a sparse 3D reconstruction.

Reference: https://github.com/yunchih/ORB-SLAM2-GPU2016-final

ORB AND FAST Algorithms

Fast Algorithm is a corner detection method, which could be **used to extract feature points.**

ORB performs the task of **feature detection**. ORB builds on the well-known FAST keypoint detector and then by BRIEF to create the descriptors.

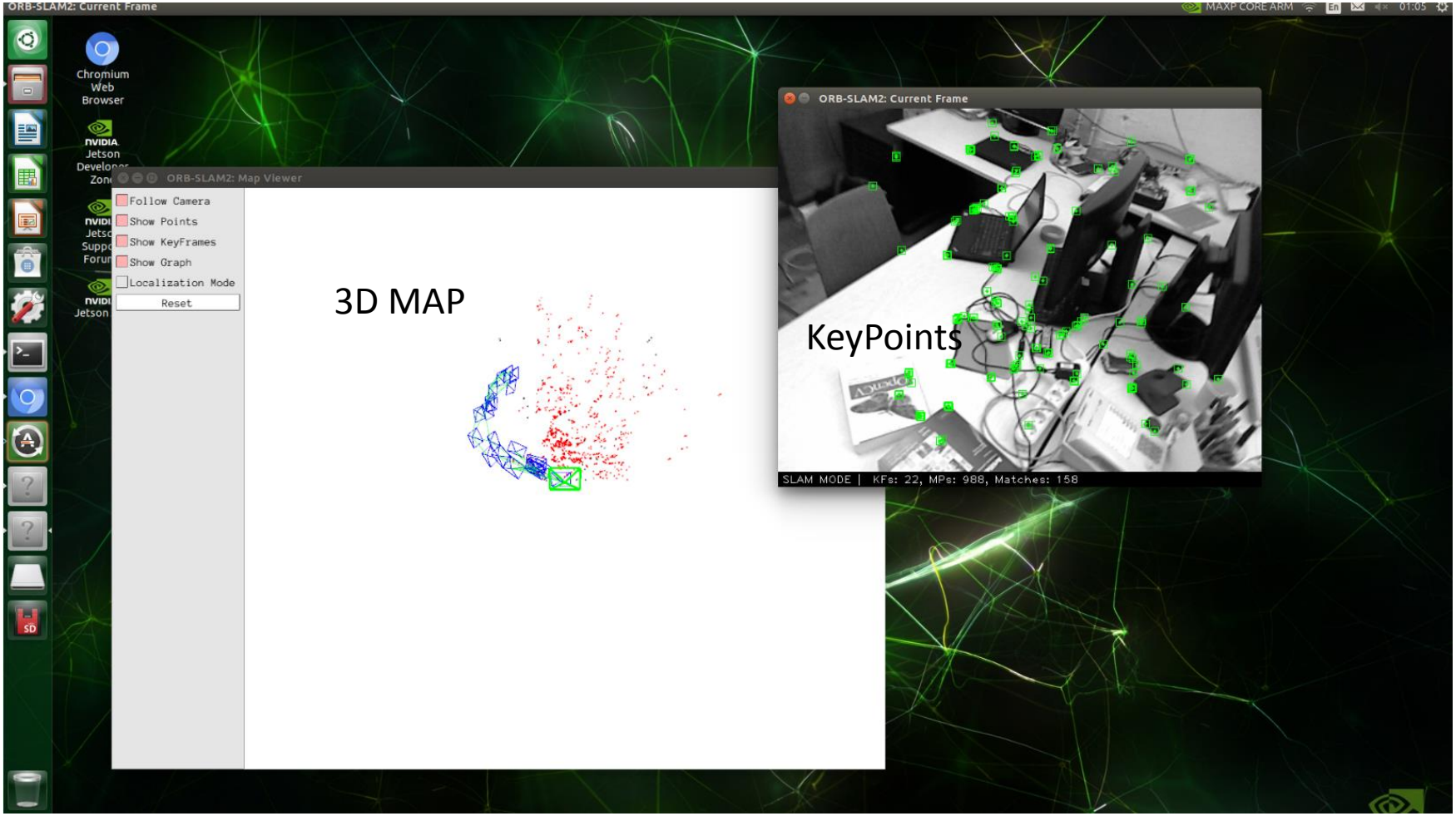
These features points used to track and map objects in many computer vision tasks.

These techniques are attractive because of their good performance and low cost.

References:
https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf
https://medium.com/data-breach/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6
https://www.youtube.com/watch?v=LeI67zjjfpQ
https://medium.com/data-breach/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65

ORB SLAM Application Environment

Configuration and Build

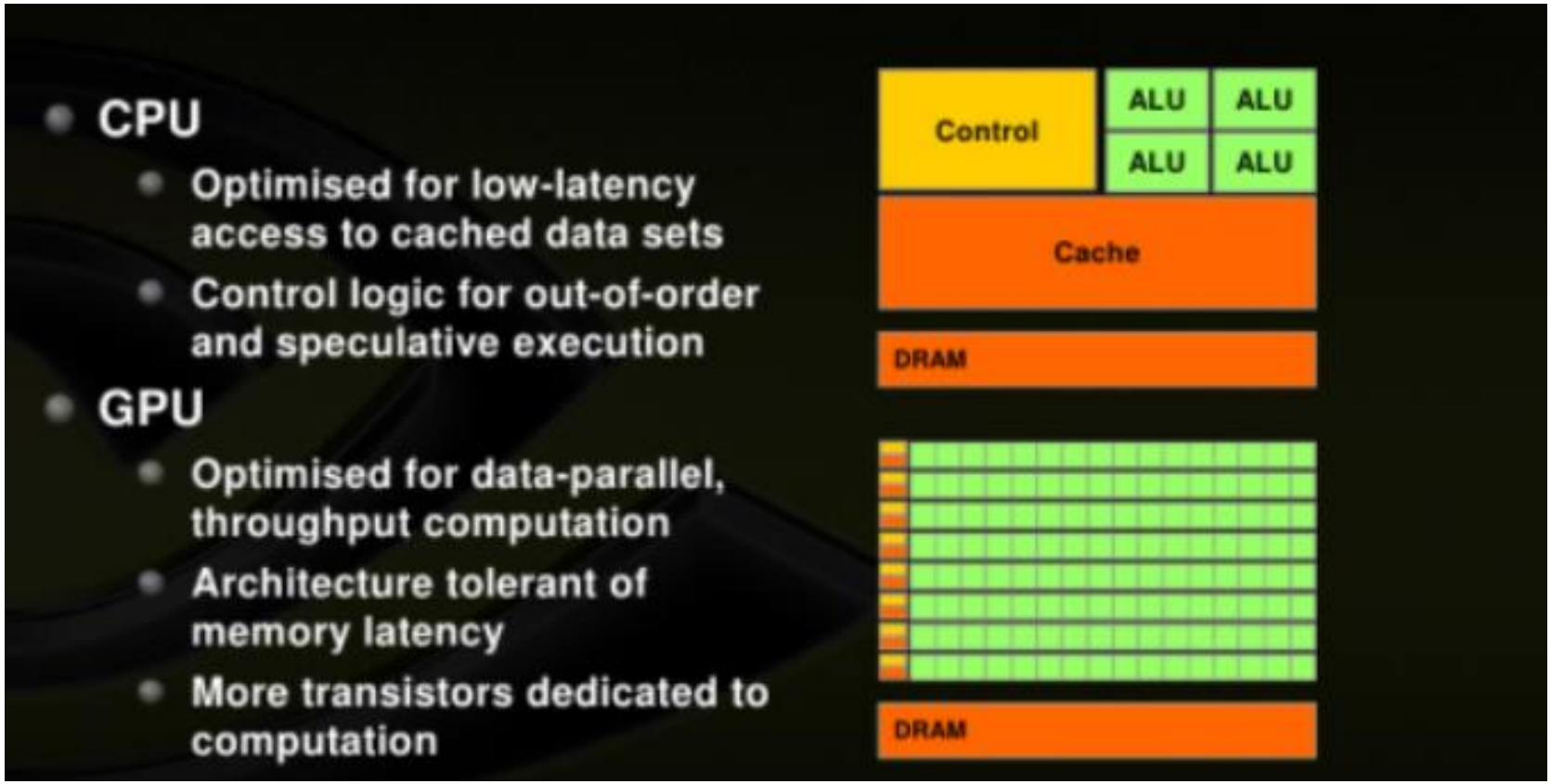


Cuda

CUDA® is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

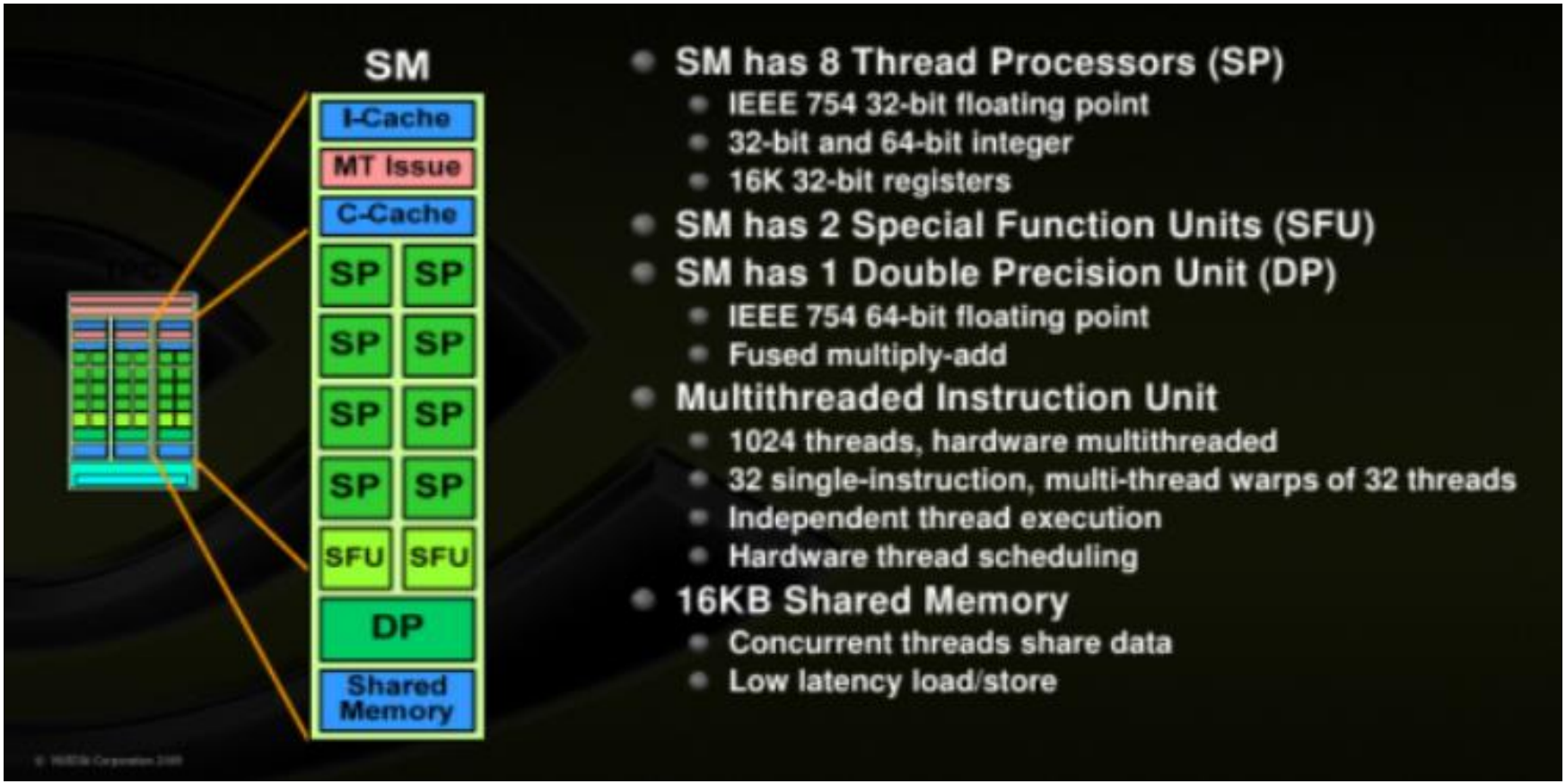
CUDA-capable GPUs have hundreds of cores that can collectively run thousands of computing threads. These cores have shared resources including a register file and a shared memory. The on-chip shared memory allows parallel tasks running on these cores to share data without sending it over the system memory bus.

Cuda CPU VS GPU



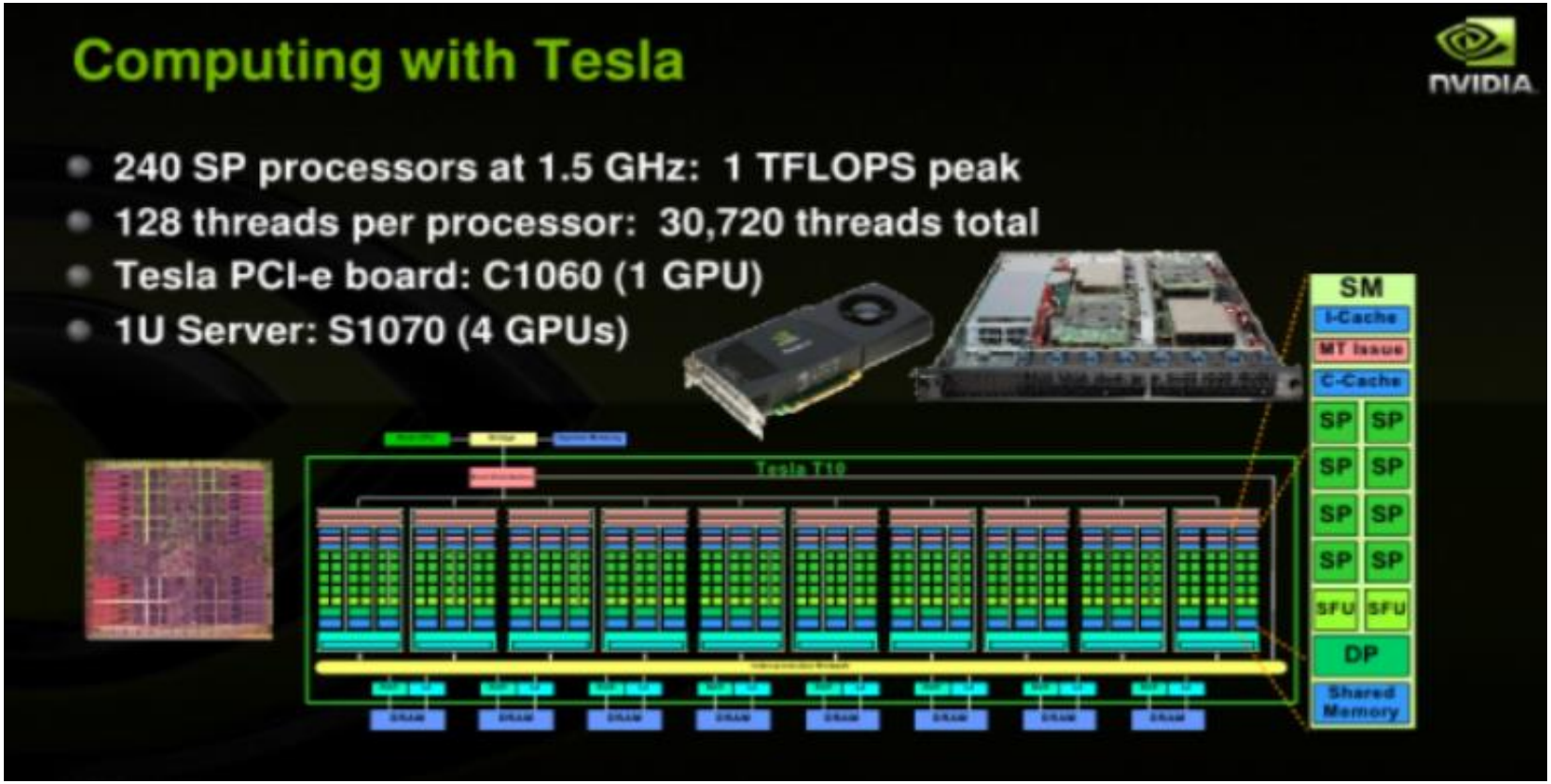
Reference: https://www.slideshare.net/maheshkha/cuda-tutorial

Cuda GPU Example



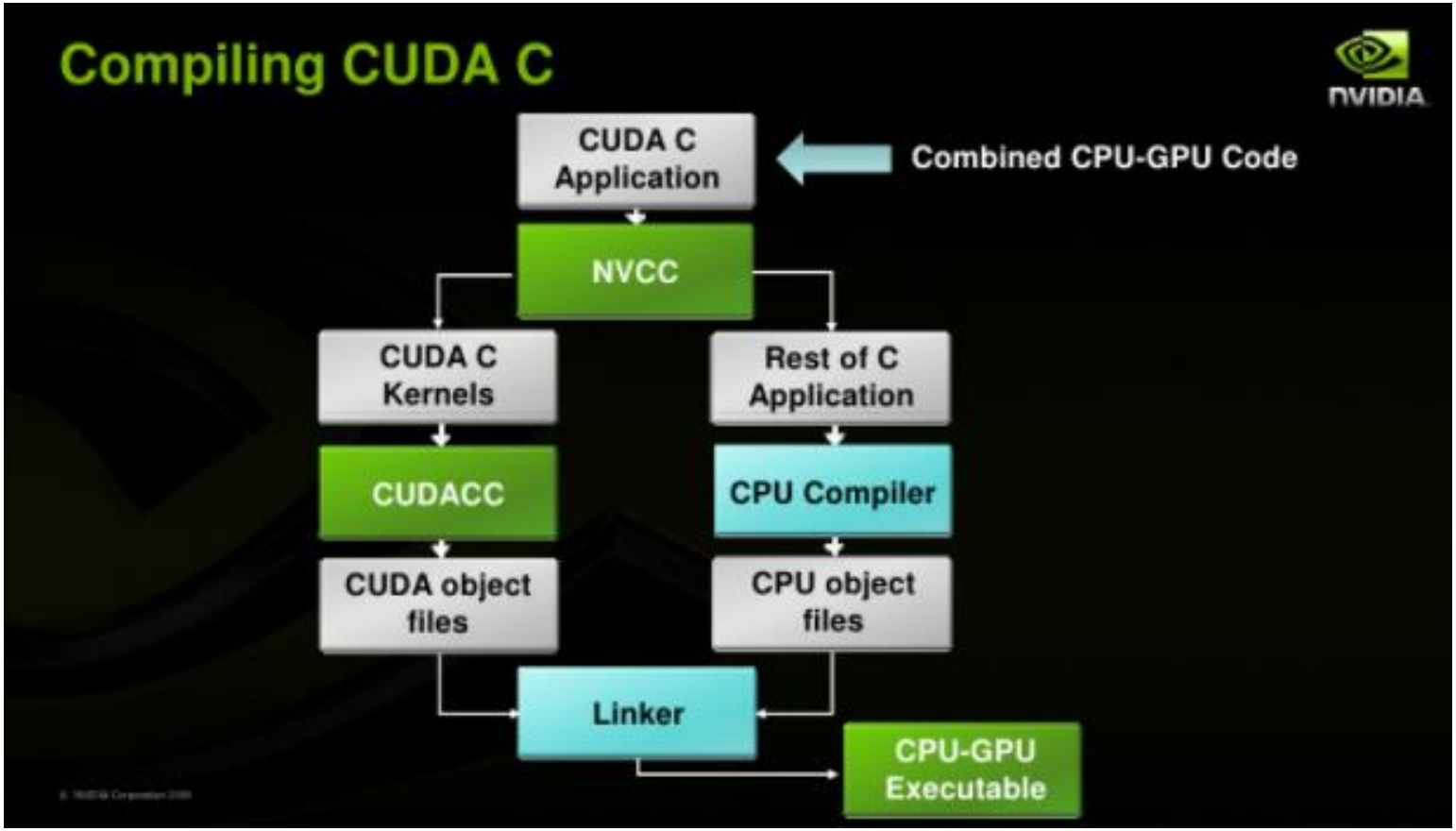
Reference: https://www.slideshare.net/maheshkha/cuda-tutorial

Cuda GPU Example – Cont.

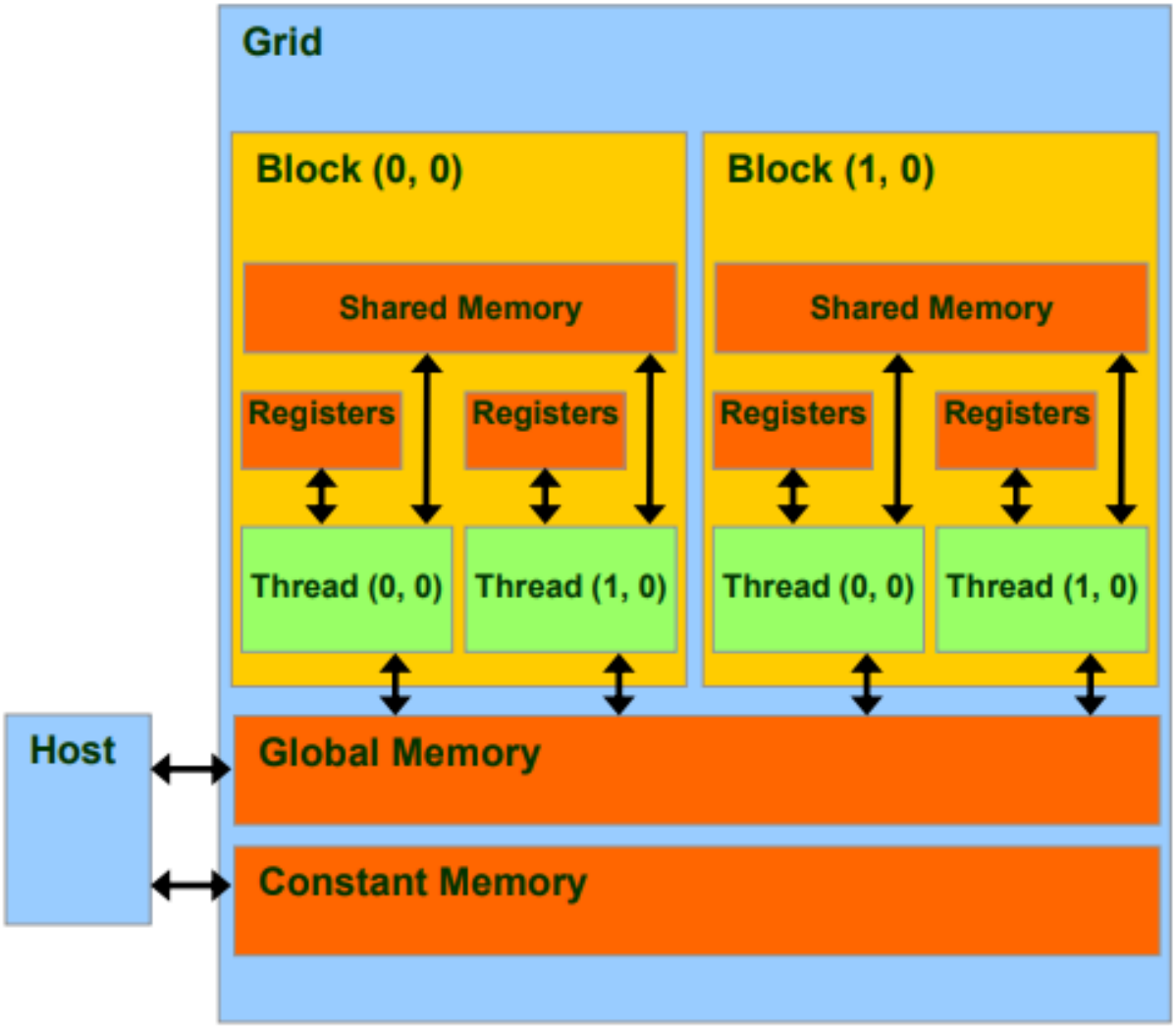


Reference: https://www.slideshare.net/maheshkha/cuda-tutorial

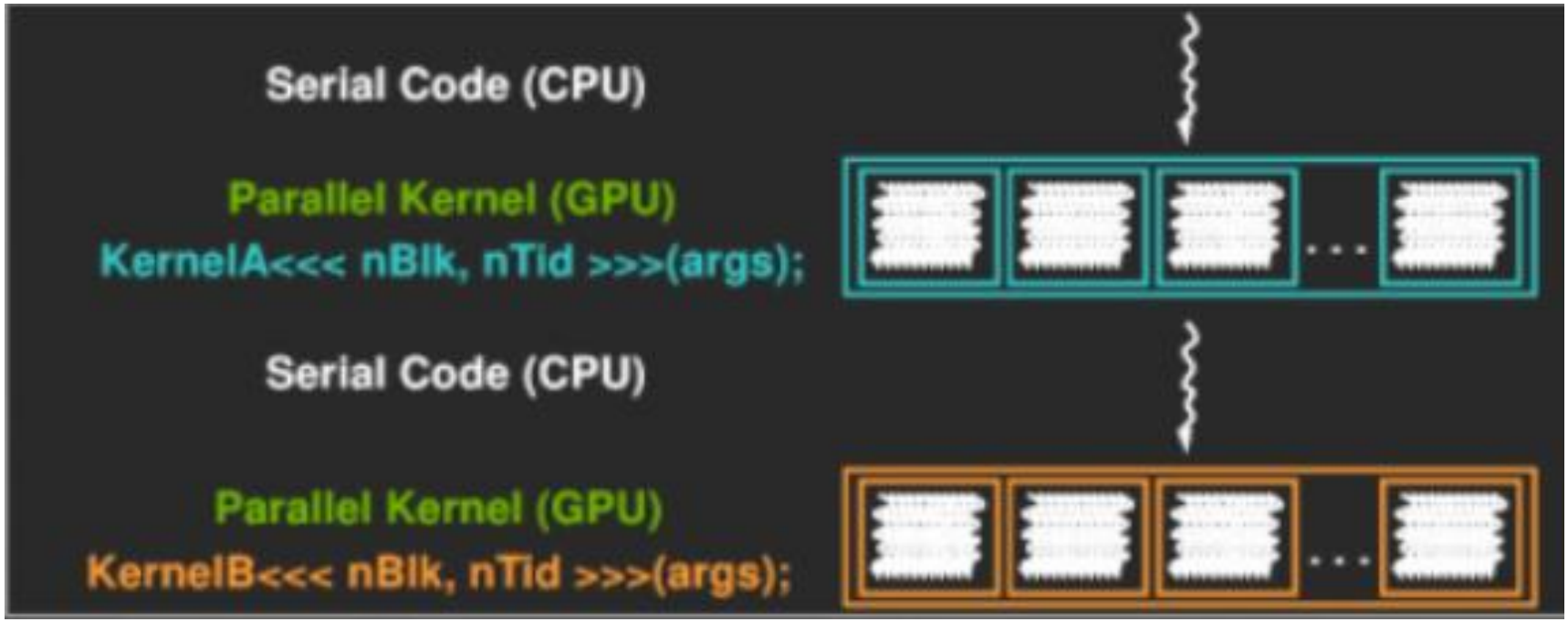
Cuda – Compilation



Cuda – Memory View

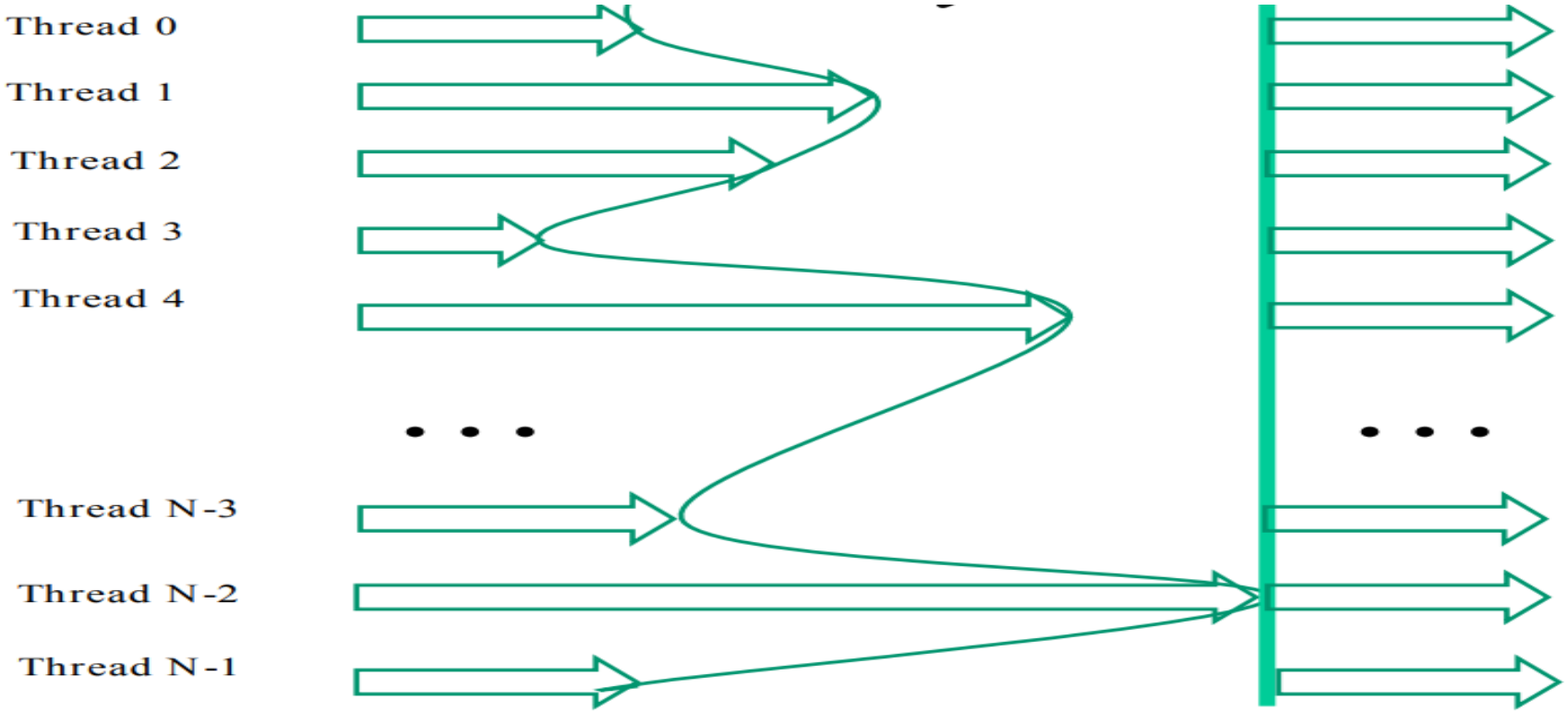


Cuda – CPU-GPU Interaction

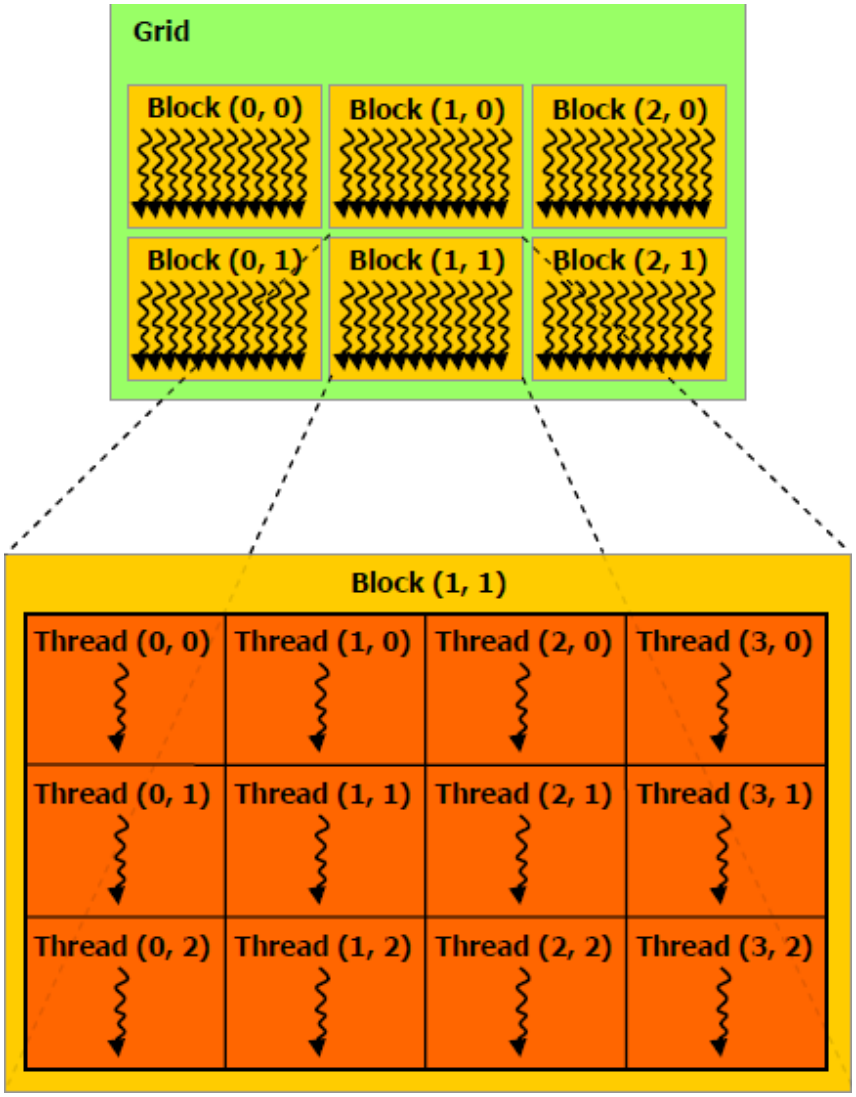


Reference: https://www.slideshare.net/maheshkha/cuda-tutorial

Cuda – Run with Barrier.



Cuda – Grid\Block\Threads



Profiling Tools

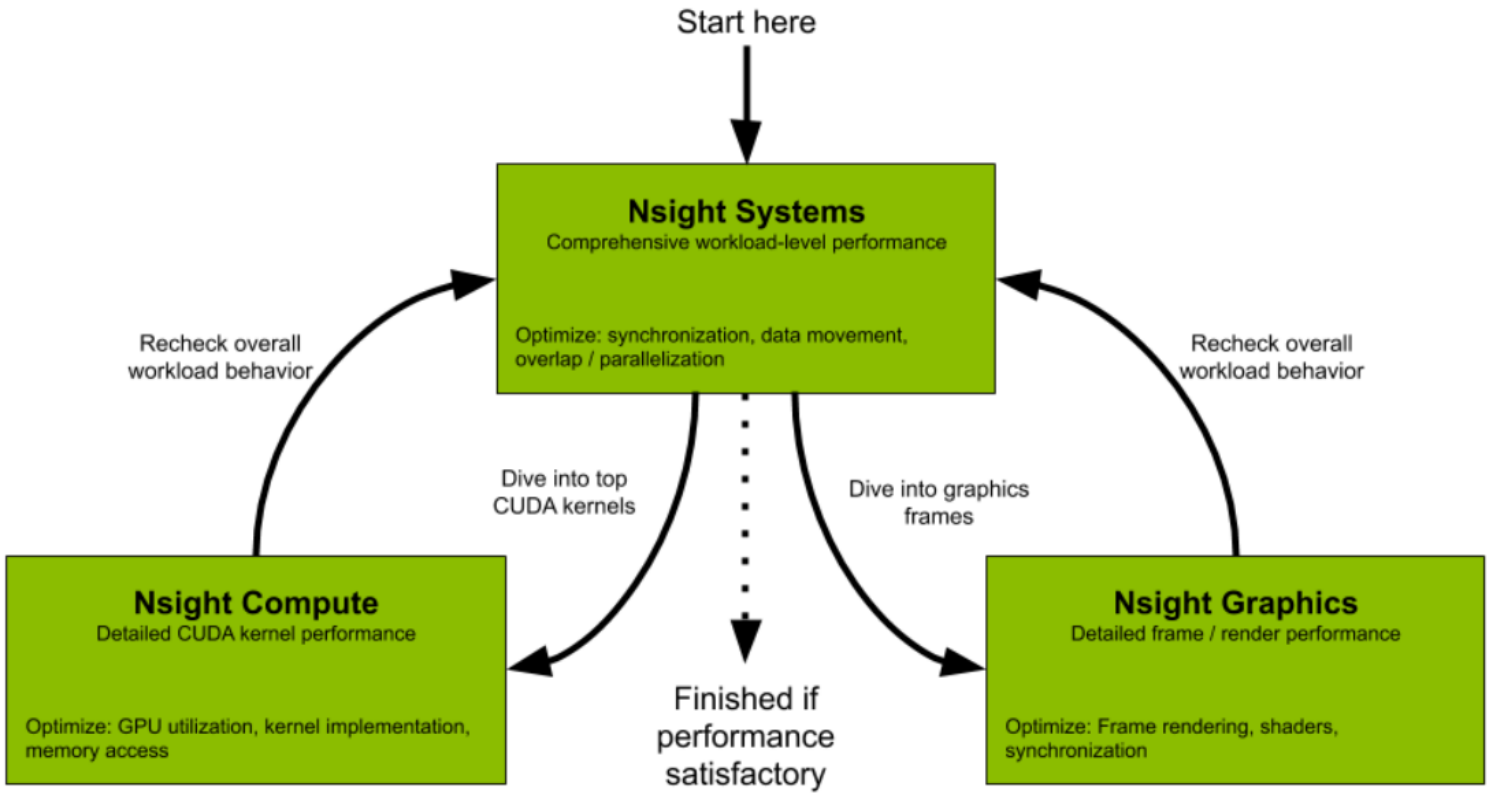
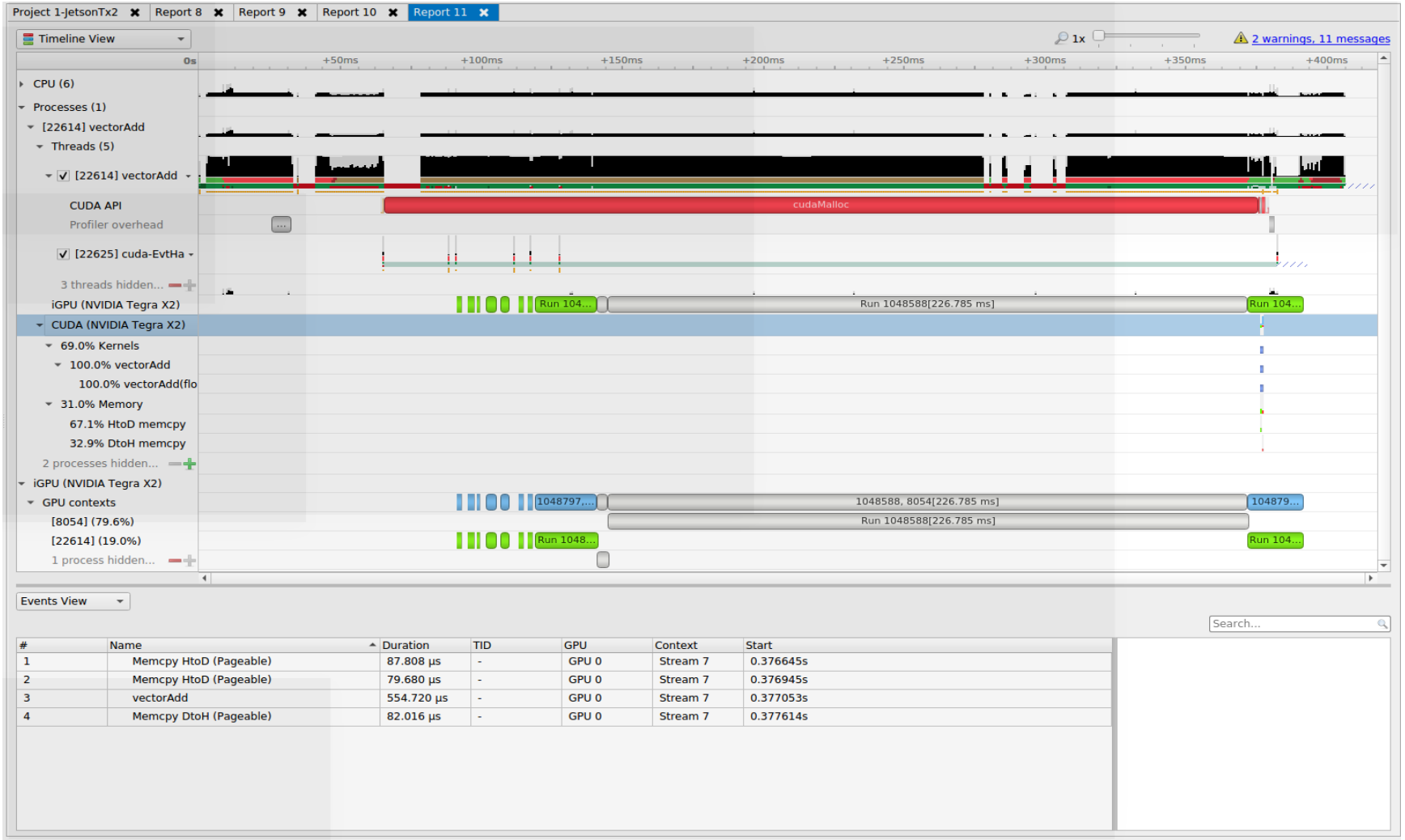
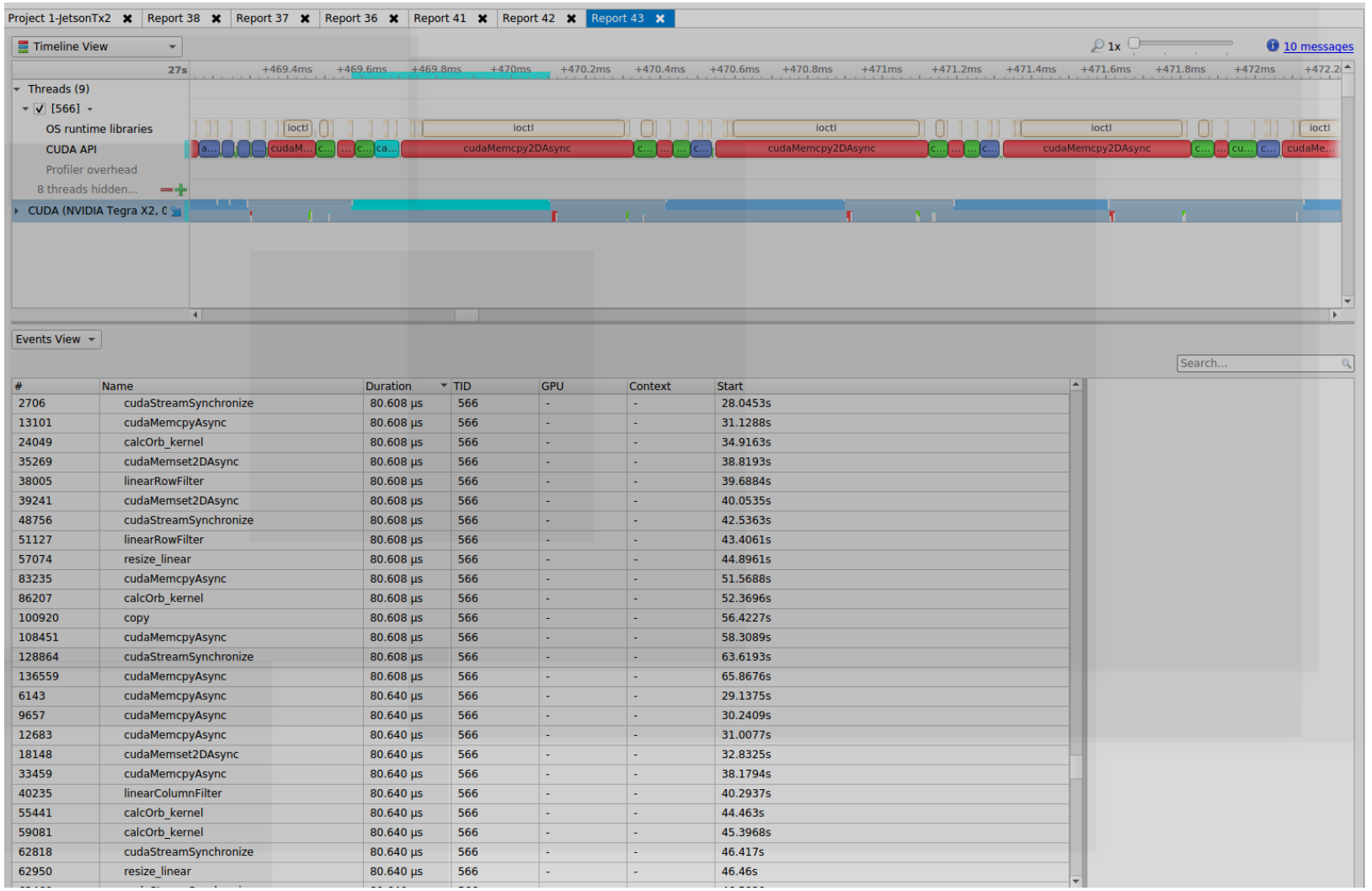


Figure 1. Flowchart describing working with new NVIDIA Nsight tools for performance optimization

Nsight Systems



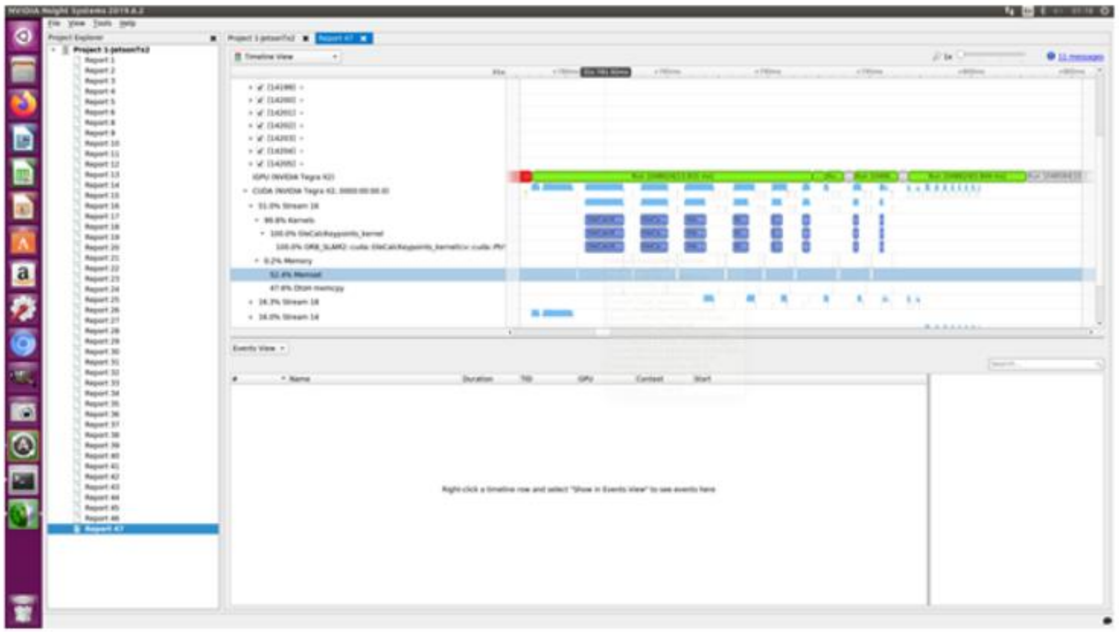
Nsight Systems – Cont.

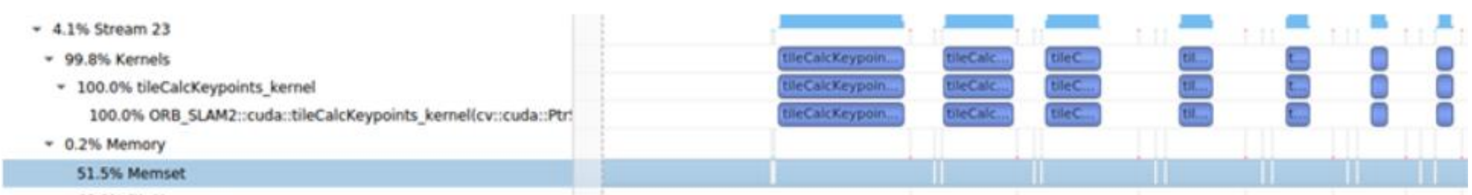


#	Name	Duration	TID	GPU	Context	Start
2706	cudaStreamSynchronize	80.608 µs	566	-	-	28.0453s
13101	cudaMemcpyAsync	80.608 µs	566	-	-	31.1288s
24049	calcOrb_kernel	80.608 µs	566	-	-	34.9163s
35269	cudaMemset2DAsync	80.608 µs	566	-	-	38.8193s
38005	linearRowFilter	80.608 µs	566	-	-	39.6884s
39241	cudaMemset2DAsync	80.608 µs	566	-	-	40.0535s
48756	cudaStreamSynchronize	80.608 µs	566	-	-	42.5363s
51127	linearRowFilter	80.608 µs	566	-	-	43.4061s
57074	resize_linear	80.608 µs	566	-	-	44.8961s
83235	cudaMemcpyAsync	80.608 µs	566	-	-	51.5688s
86207	calcOrb_kernel	80.608 µs	566	-	-	52.3696s
100920	copy	80.608 µs	566	-	-	56.4227s
108451	cudaMemcpyAsync	80.608 µs	566	-	-	58.3089s
128864	cudaStreamSynchronize	80.608 µs	566	-	-	63.6193s
136559	cudaMemcpyAsync	80.608 µs	566	-	-	65.8676s
6143	cudaMemcpyAsync	80.640 µs	566	-	-	29.1375s
9657	cudaMemcpyAsync	80.640 µs	566	-	-	30.2409s
12683	cudaMemcpyAsync	80.640 µs	566	-	-	31.0077s
18148	cudaMemset2DAsync	80.640 µs	566	-	-	32.8325s
33459	cudaMemcpyAsync	80.640 µs	566	-	-	38.1794s
40235	linearColumnFilter	80.640 µs	566	-	-	40.2937s
55441	calcOrb_kernel	80.640 µs	566	-	-	44.463s
59081	calcOrb_kernel	80.640 µs	566	-	-	45.3968s
62818	cudaStreamSynchronize	80.640 µs	566	-	-	46.417s
62950	resize_linear	80.640 µs	566	-	-	46.46s

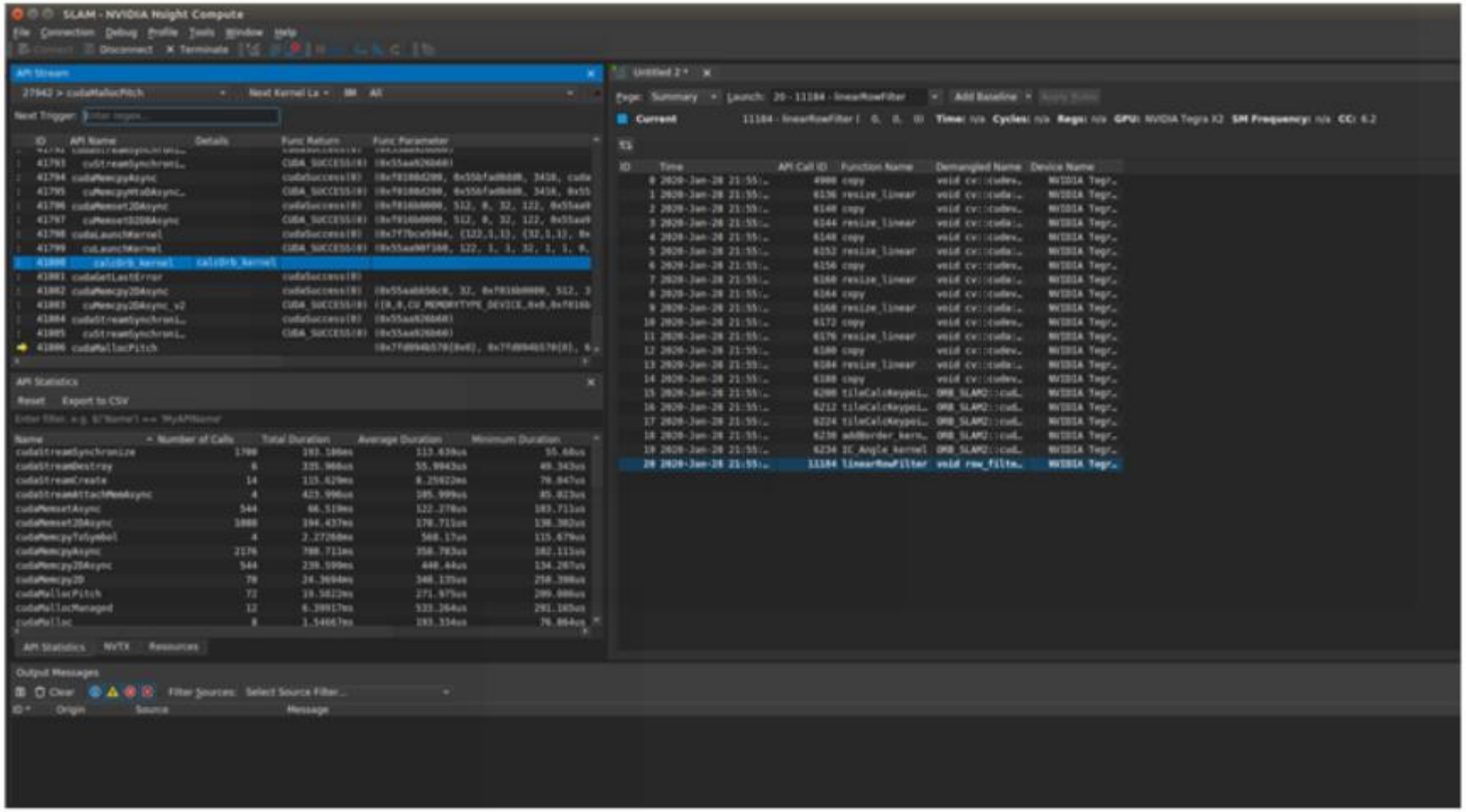
Nsight Systems – Cont.

Nsight Systems – Cont.

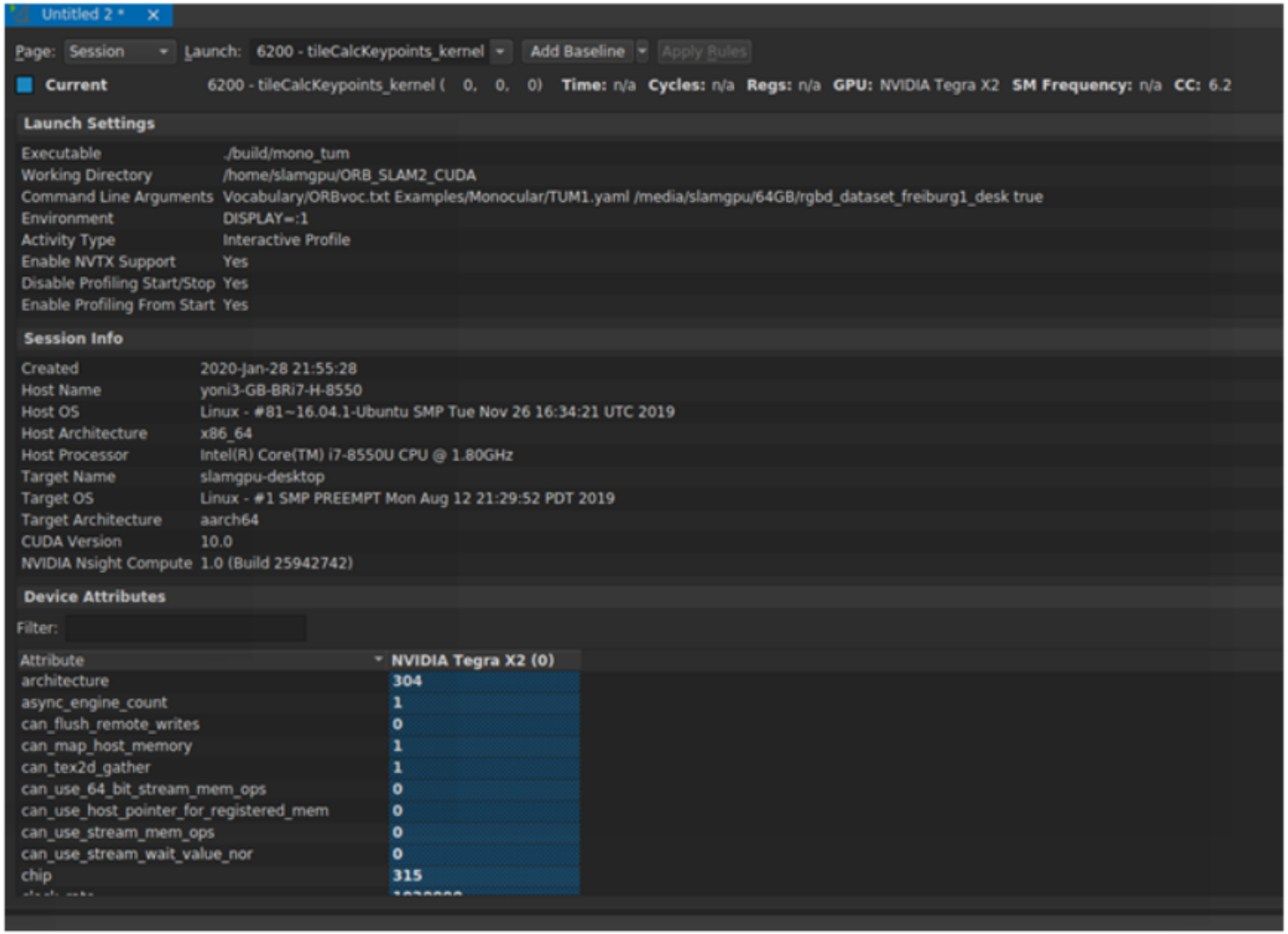




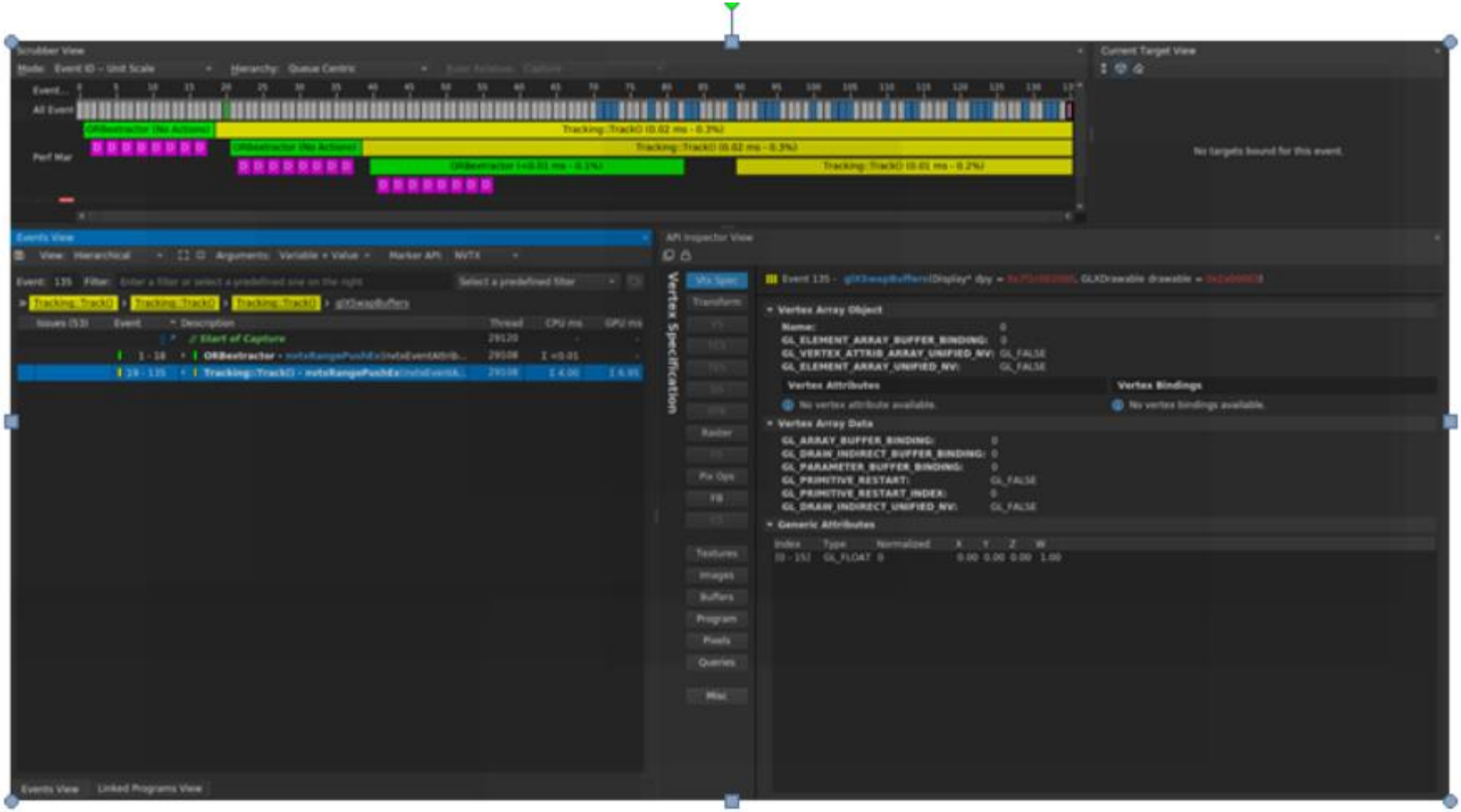
Nsight Compute



Nsight Compute – Cont.



Nsight Grapics

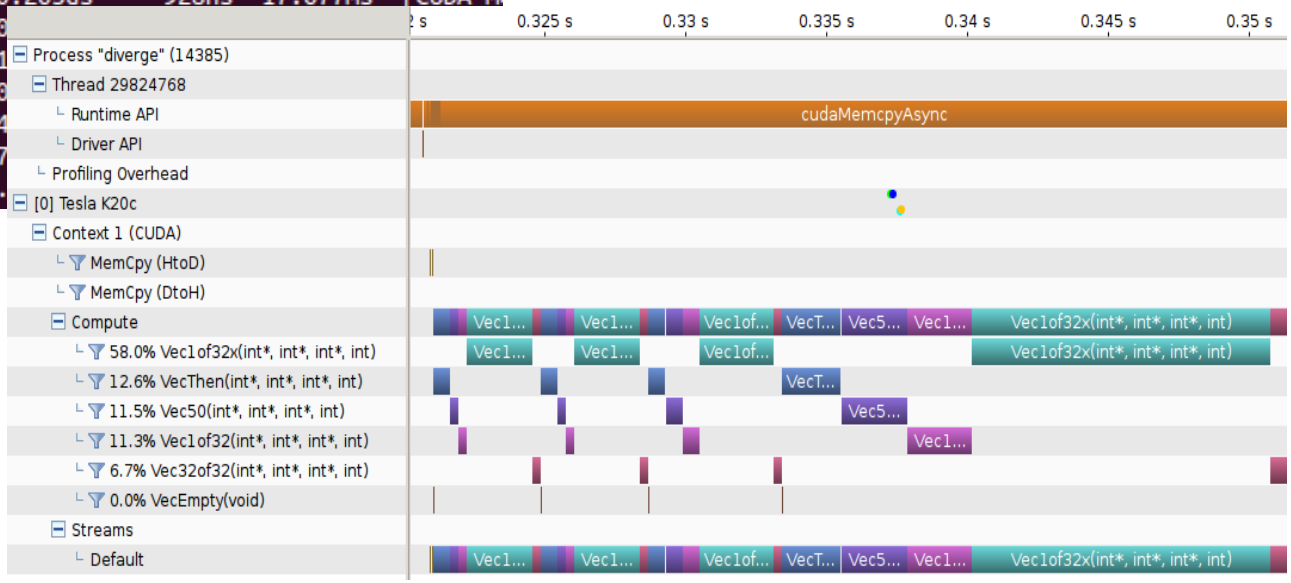


Profiling Tools - Legacy

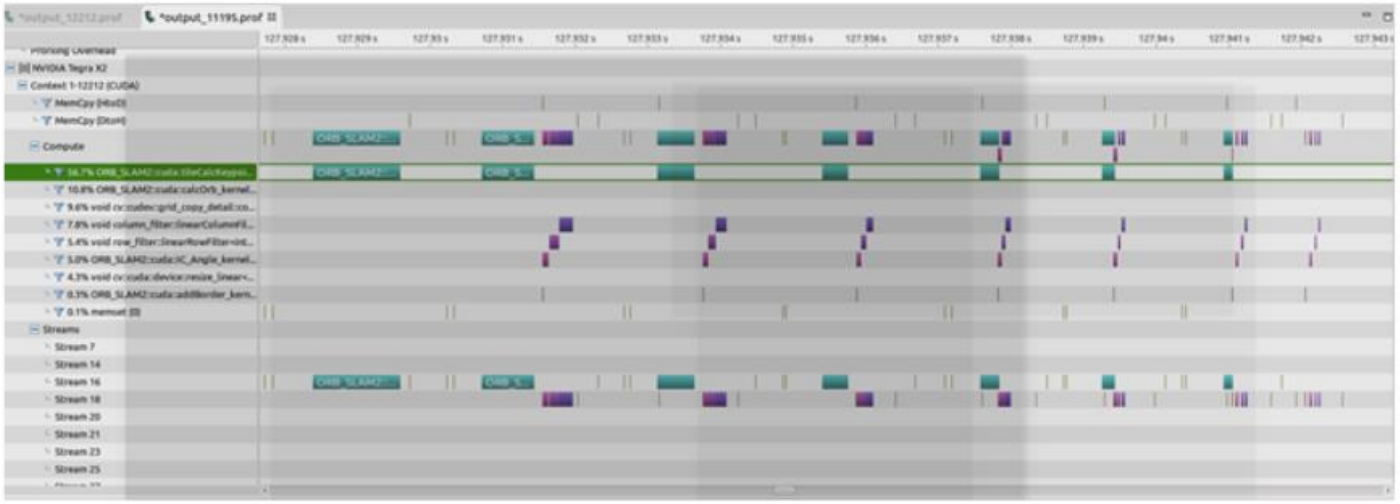
NVPROF

```
==20561== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 49.88%  866.69ms    504758  1.7170us  1.5040us  2.0160us  void th
int, thrust::detail::device_generate_functor<thrust::detail::fill_
 25.33%  440.05ms    252662  1.7410us  1.5360us  2.3680us  void th
t, thrust::detail::device_generate_functor<thrust::detail::fill_fu
 17.07%  296.60ms       200  1.4830ms  1.2040ms  1.7253ms  kerComp
  2.98%  51.819ms       200  259.09us  246.97us  264.83us  kerMake
  1.16%  20.173ms       501  40.265us     928ns  17.677ms  [CUDA m
  0.93%  16.198ms       200  80
  0.73%  12.636ms       400  31
  0.69%  12.075ms       200  60
  0.63%  10.993ms       200  54
  0.32%  5.5524ms       200  27
  0.12%  2.1342ms         1  2.
```

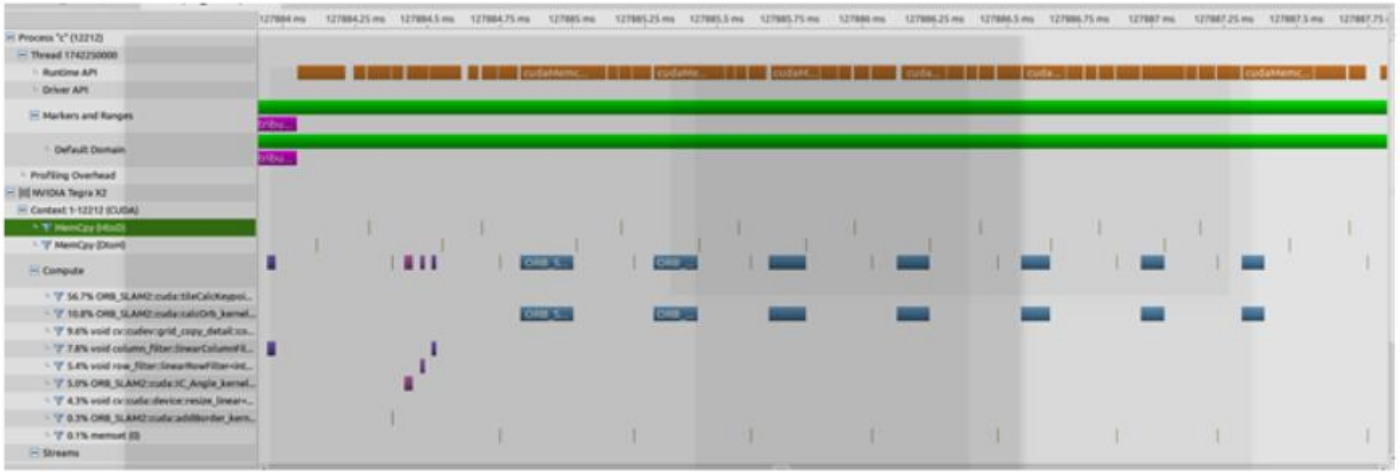
Visual Profiler



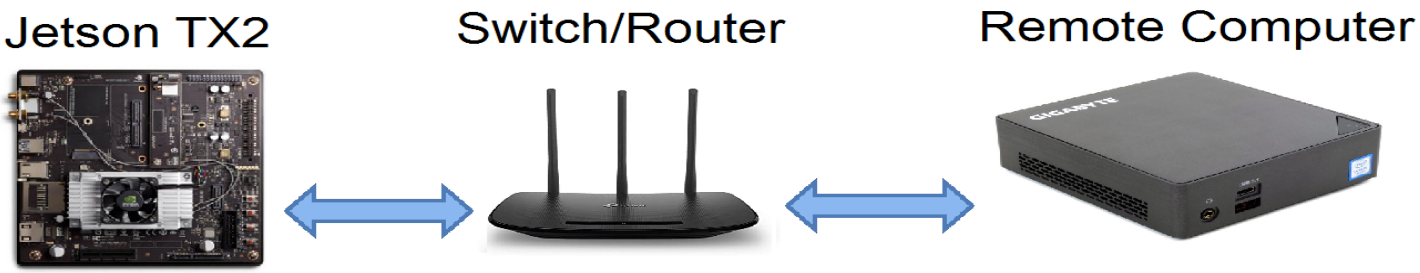
Profiling Tools – Legacy – Cont.

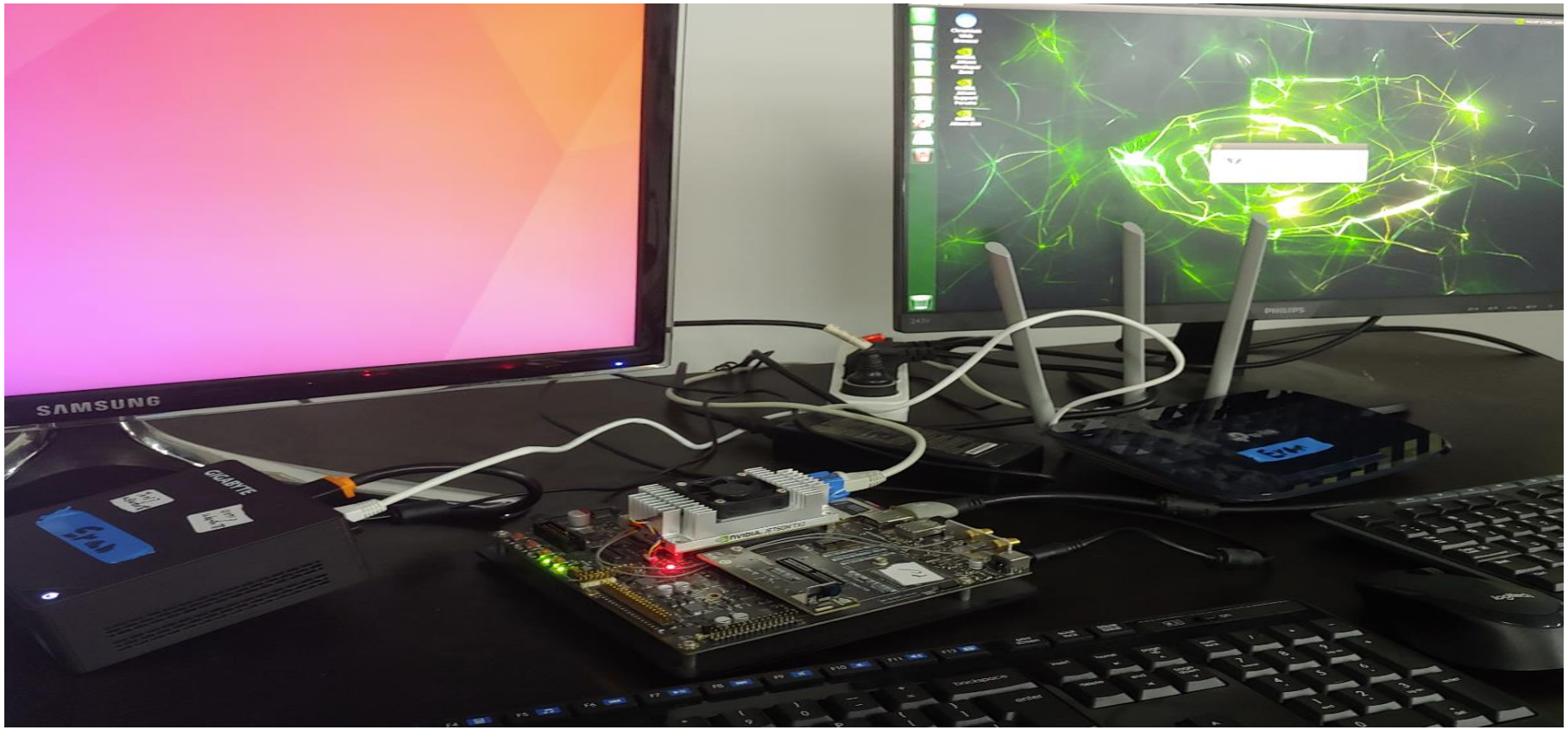


You can see the times for transfer data between host to device and vice versa.



Remote Profiling Configuration





Future Research

This project will help in the future to students and people which interesting in **Parallel Programming** study **Cuda programming** in easy and fast way then could implement, improve and optimize parallel applications. By using Nvidia profiling tools students and people could understand how to **debug** in better way application which run on CPU and especially on GPU, understand more **deeply** about the **Parallel concepts** and how to improve the **Performance**.

Suggestion For Next Research Project

In the next project, the student can learn about the Cuda programming then deep into the computer vision algorithms (Specially FAST, ORB algorithms) then run the ORB SLAM application and change the kernels (GPU main function calls) parameters like grid size (blocks count) and threads per block and try to optimize the application and use the profiling tools to archive performance and latency benefits. Optimize the code and add more functionalities can be useful for the future.

Questions ???

Thanks