ORB SLAM2 CUDA Guide

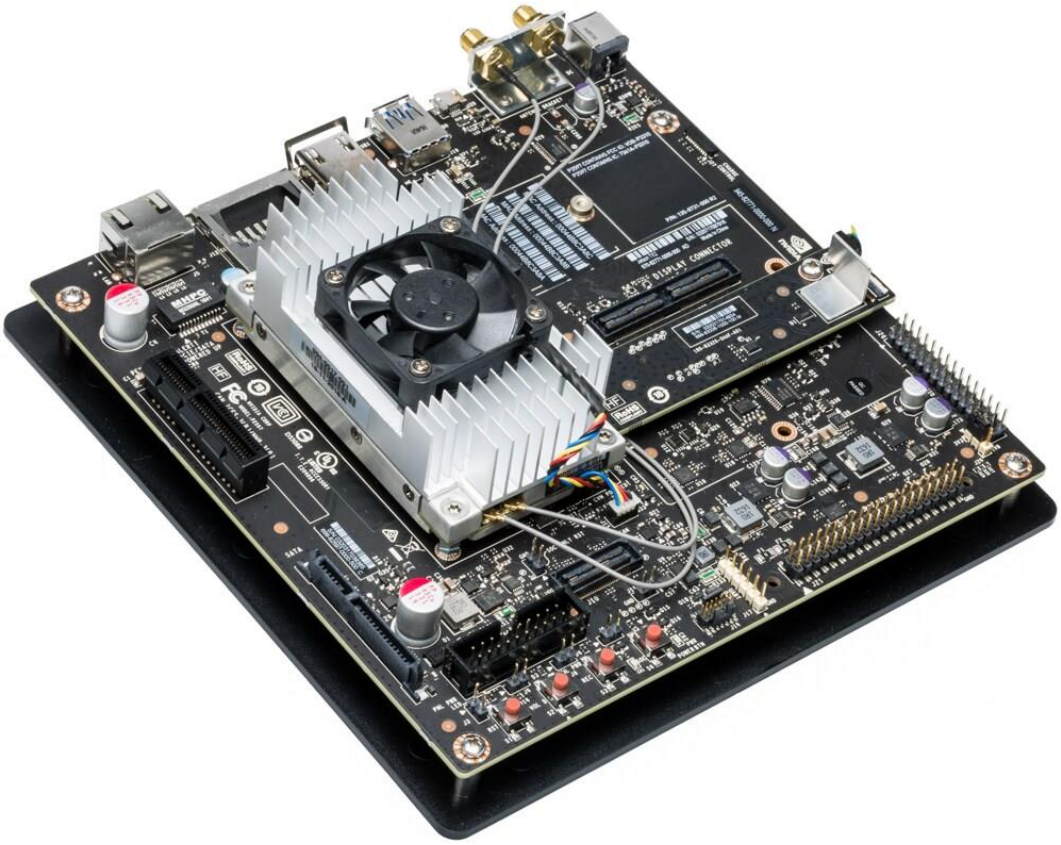
Written by Eran Moshe

Supervision Yehonatan Mendel

Autonomous Drones Lab
Faculty of Electrical Engineering, Tel Aviv University

Version 3.0

25.05.2020



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 are:

- 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.

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.

Table of Contents

Acronyms

SLAM	Simultaneous Localization And Mapping
GPU	Graphics Processing Unit
ROS	Robots Operation System
BLAS	Basic Linear Algebra Subprograms
LAPACK	Linear Algebra Package
AI	Artificial Intelligence
FLOPS	Floating Point Operations
SDK	Software development Kit
L4T	NVidia Linux Driver Package
API	Application Programming Interface
PCL	Point Cloud Library

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.

Main spefications:

GPU - NVIDIA Pascal™ architecture with 256 NVIDIA CUDA cores 1.3 TFLOPS (FP16).

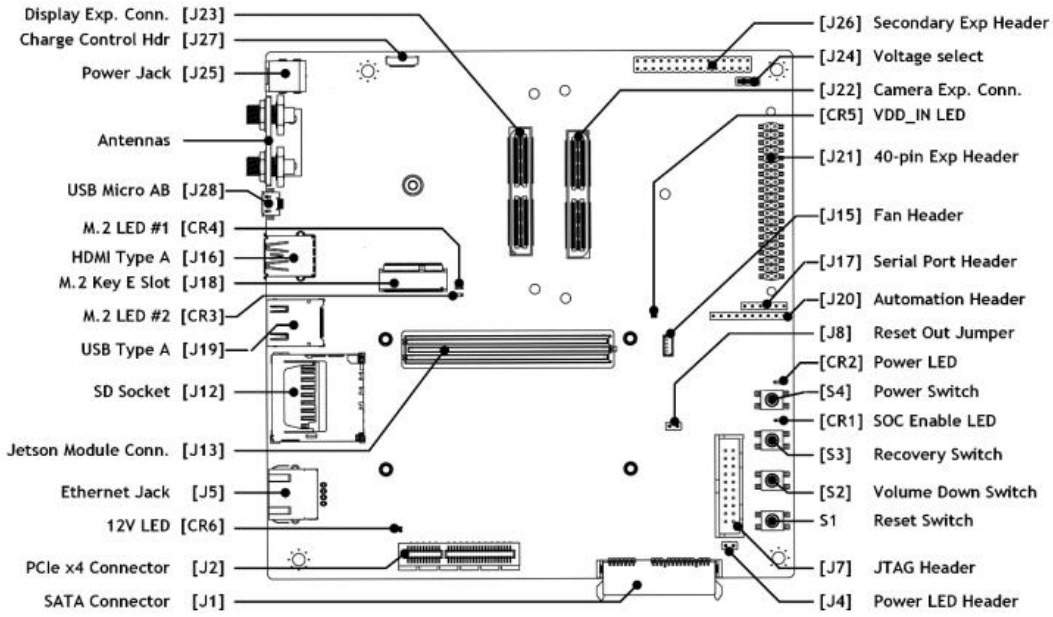
CPU - Dual-core Denver 2 64-bit CPU and quad-core ARM A57 complex

Memory - 8 GB 128-bit LPDDR4 1866MHz - 59.7 GB/s

Storage - 32 GB eMMC 5.1

Connectivity - 10/100/1000 BASE-T Ethernet

Top view of developer kit carrier board (revision C02)



HW Important link: https://elinux.org/Jetson_TX2

Jetson Software

Jetson default OS configuration is Linux Ubunto.

JetPack bundles all of the Jetson platform software, starting with the NVIDIA® Jetson™ Linux Driver Package (L4T). L4T provides the Linux kernel, bootloader, NVIDIA drivers, flashing utilities, sample filesystem, and more for the Jetson platform.

NVIDIA **JetPack SDK** is a comprehensive resource for building AI applications. It includes L4T together with accelerated software libraries, APIs, sample applications, developer tools, and documentation.

NVIDIA L4T Supported features:

- Kernel version 4.9
- Support for 64-bit user space and runtime libraries
- Vulkan Support
- V4L2 media-controller driver support for camera sensors (bypassing ISP)
- libargus provides low-level frame-synchronous API for camera applications
    - RAW output CSI cameras needing ISP can be used with either libargus or GStreamer plugin
- Media APIs:
    - OpenGL 4.6 Beta
    - OpenGL ES 3.2
    - OpenGL ES path extensions
    - EGL 1.5 with EGLImage
- X Resize, Rotate and Reflect Extension (RandR) 1.4
- X11 Support

SW Important link: https://developer.nvidia.com/embedded/develop/software

L4T link: https://docs.nvidia.com/jetson/l4t/index.html

JetPack Roadmap link: https://developer.nvidia.com/embedded/develop/roadmap

ORB SLAM Repositories

Index	Web Page Link	Description
1	https://github.com/hoangthien94/ORB_SLAM2_CUDA	Optimized for ROS 18-22 fps
2	https://github.com/connorsoohoo/ORB-SLAM2-GPU-RGBD	Optimized for RGBD 18-20 fps
3	https://github.com/yunchih/ORB-SLAM2-GPU2016-final	GPU enhancements with Cuda ~14 fps
4	https://github.com/raulmur/ORB_SLAM2	Original ORB SLAM

In this document I used the first one which including Cuda Optimizations.

Sdcard Configuration

Jetson TX2 contain eMMC chip with storage capacity of 32 GB.
It is recommended to use external SD card for expand the memory.

For new Sdcard install the exfat support:
sudo apt-get install exfat-fuse exfat-utils

Then reinsert the card.

If it does not mount automatically, try:

manual mount:

sudo mount.exfat /dev/mmcblk0p1 /your/mount/point

ORB SLAM Installation steps

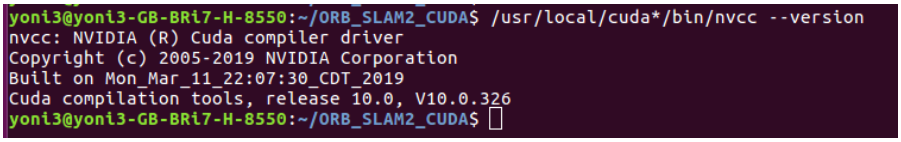
This configuration is on the target (Jetson tx2)

Verify Required Tools

1) Check cuda version

/usr/local/cuda/bin/nvcc --version*

Example:



2) Check OpenCV installation/version/configuration (run with python):

python
import cv2
print cv2.getBuildInformation()

Example:

```
>>> import cv2
>>>
>>>
>>> print cv2.getBuildInformation()

General configuration for OpenCV 3.3.1 =====================================
  Version control:               3.3.1-2-g31ccdfe11

  Platform:
    Timestamp:                   2019-02-06T09:45:58Z
    Host:                        Linux 4.9.140-tegra aarch64
    CMake:                       2.8.12.2
    CMake generator:             Unix Makefiles
    CMake build tool:            /usr/bin/make
    Configuration:               Release

  CPU/HW features:
    Baseline:                    NEON FP16
      required:                  NEON
      disabled:                  VFPV3

  C/C++:
    Built as dynamic libs?:      YES
    C++11:                       YES
    C++ Compiler:                /usr/bin/c++  (ver 7.3.0)
    C++ flags (Release):         -fsigned-char -W -Wall -Werror=return-type -Werror=non-virtual-dtor -Werror=address -Werror=sequence-point -Wformat -Werror=format-security -Wmissing-declarations -Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-promo -Wuninitialized -Winit-self -Wno-narrowing -Wno-delete-non-virtual-dtor -Wno-comment -Wno-implicit-fallthrough -fdiagnostics-show-option -pthread -fomit-frame-pointer -ffunction-sections    -fvisibility=hidden -fvisibility-inlines-hidden -O3 -DNDEBUG  -DNDEBUG
    C++ flags (Debug):           -fsigned-char -W -Wall -Werror=return-type -Werror=non-virtual-dtor -Werror=address -Werror=sequence-point -Wformat -Werror=format-security -Wmissing-declarations -Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-promo -Wuninitialized -Winit-self -Wno-narrowing -Wno-delete-non-virtual-dtor -Wno-comment -Wno-implicit-fallthrough -fdiagnostics-show-option -pthread -fomit-frame-pointer -ffunction-sections    -fvisibility=hidden -fvisibility-inlines-hidden -g  -O0 -DDEBUG -D_DEBUG
    C Compiler:                  /usr/bin/cc
    C flags (Release):           -fsigned-char -W -Wall -Werror=return-type -Werror=non-virtual-dtor -Werror=address -Werror=sequence-point -Wformat -Werror=format-security -Wmissing-declarations -Wmissing-prototypes -Wstrict-prototypes -Wundef -Winit-self -Wpointer-arith -Wshadow -Wuninitialized -Winit-self -Wno-narrowing -Wno-comment -Wno-implicit-fallthrough -fdiagnostics-show-option -pthread -fomit-frame-pointer -ffunction-sections    -fvisibility=hidden -O3 -DNDEBUG  -DNDEBUG
    C flags (Debug):             -fsigned-char -W -Wall -Werror=return-type -Werror=non-virtual-dtor -Werror=address -Werror=sequence-point -Wformat -Werror=format-security -Wmissing-declarations -Wmissing-prototypes -Wstrict-prototypes -Wundef -Winit-self -Wpointer-arith -Wshadow -Wuninitialized -Winit-self -Wno-narrowing -Wno-comment -Wno-implicit-fallthrough -fdiagnostics-show-option -pthread -fomit-frame-pointer -ffunction-sections    -fvisibility=hidden -g  -O0 -DDEBUG -D_DEBUG
    Linker flags (Release):
    Linker flags (Debug):
    ccache:                      NO
    Precompiled headers:         NO
    Extra dependencies:          dl m pthread rt /usr/lib/aarch64-linux-gnu/libtbb.so
    3rdparty dependencies:

  OpenCV modules:
    To be built:                 core flann imgproc ml objdetect photo video dnn imgcodecs shape videoio highgui superres ts features2d calib3d stitching videostab python2 python3
    Disabled:                    js world
    Disabled by dependency:      -
    Unavailable:                 cudaarithm cudabgsegm cudacodec cudafeatures2d cudafilters cudaimgproc cudalegacy cudaobjdetect cudaoptflow cudastereo cudawarping cudev java viz

  GUI:
    QT:                          NO
    GTK+ 2.x:                    YES (ver 2.24.32)
    GThread :                    YES (ver 2.56.3)
    GtkGlExt:                    NO
    OpenGL support:              NO
    VTK support:                 NO

  Media I/O:
    ZLib:                        /usr/lib/aarch64-linux-gnu/libz.so (ver 1.2.11)
    JPEG:                        /usr/lib/aarch64-linux-gnu/libjpeg.so (ver )
    WEBP:                        build (ver encoder: 0x020e)
    PNG:                         /usr/lib/aarch64-linux-gnu/libpng.so (ver 1.6.34)
    TIFF:                        /usr/lib/aarch64-linux-gnu/libtiff.so (ver 42 - 4.0.9)
    JPEG 2000:                   build (ver 1.900.1)
    OpenEXR:                     NO
    GDAL:                        NO
    GDCM:                        NO

  Video I/O:
    DC1394 1.x:                  NO
    DC1394 2.x:                  NO
    FFMPEG:                      YES
      avcodec:                   YES (ver 57.107.100)
      avformat:                  YES (ver 57.83.100)
      avutil:                    YES (ver 55.78.100)
      swscale:                   YES (ver 4.8.100)
      avresample:                NO
    GStreamer:
      base:                      YES (ver 1.14.1)
      video:                     YES (ver 1.14.1)
      app:                       YES (ver 1.14.1)
```

```
    riff:                        YES (ver 1.14.1)
    pbutils:                     YES (ver 1.14.1)
  OpenNI:                      NO
  OpenNI PrimeSensor Modules:  NO
  OpenNI2:                     NO
  PvAPI:                       NO
  GigEVisionSDK:               NO
  Aravis SDK:                  NO
  UniCap:                      NO
  UniCap ucil:                 NO
  V4L/V4L2:                    NO/YES
  XIMEA:                       NO
  Xine:                        NO
  Intel Media SDK:             NO
  gPhoto2:                     NO

Parallel framework:            TBB (ver 2017.0 interface 9107)

Trace:                         YES ()

Other third-party libraries:
  Use Intel IPP:               NO
  Use Intel IPP IW:            NO
  Use VA:                      NO
  Use Intel VA-API/OpenCL:     NO
  Use Lapack:                  NO
  Use Eigen:                   YES (ver 3.3.4)
  Use Cuda:                    NO
  Use OpenCL:                  NO
  Use OpenVX:                  NO
  Use custom HAL:              YES (carotene (ver 0.0.1))

Python 2:
  Interpreter:                 /usr/bin/python2.7 (ver 2.7.15)
  Libraries:                   /usr/lib/aarch64-linux-gnu/libpython2.7.so (ver 2.7.15rc1)
  numpy:                       /usr/lib/python2.7/dist-packages/numpy/core/include (ver 1.13.3)
  packages path:               lib/python2.7/dist-packages

Python 3:
  Interpreter:                 /usr/bin/python3 (ver 3.6.7)
  Libraries:                   /usr/lib/aarch64-linux-gnu/libpython3.6m.so (ver 3.6.7)
  numpy:                       /usr/lib/python3/dist-packages/numpy/core/include (ver 1.13.3)
  packages path:               lib/python3.6/dist-packages

Python (for build):            /usr/bin/python2.7

Java:
  ant:                         NO
  JNI:                         NO
  Java wrappers:               NO
  Java tests:                  NO

Matlab:                        Matlab not found or implicitly disabled

Documentation:
  Doxygen:                     NO

Tests and samples:
  Tests:                       YES
  Performance tests:           YES
  C/C++ Examples:              YES

Install path:                  /usr

cvconfig.h is in:              /home/nvidia/build_opencv/build
-----------------------------------------------------------------
```

OpenCV Installation

This tutorial explain how to install OpenCV version 4.1.1

Check the current version with command:

Example:

slamgpu@slamgpu-desktop:~$ opencv_version
3.3.1

To upgrade OpenCV to 4.1.1 run script below:

```
#!/bin/bash
#
# Copyright (c) 2018, NVIDIA CORPORATION.  All rights reserved.
#
# NVIDIA Corporation and its licensors retain all intellectual property
# and proprietary rights in and to this software, related documentation
# and any modifications thereto.  Any use, reproduction, disclosure or
# distribution of this software and related documentation without an express
# license agreement from NVIDIA Corporation is strictly prohibited.
#

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <Install Folder>"
    exit
fi
folder="$1"
user="nvidia"
passwd="nvidia"

echo "** Remove other OpenCV first"
sudo sudo apt-get purge *libopencv*

echo "** Install requirement"
sudo apt-get update
sudo apt-get install -y build-essential cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
sudo apt-get install -y python2.7-dev python3.6-dev python-dev python-numpy python3-numpy
sudo apt-get install -y libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libdc1394-22-dev
sudo apt-get install -y libv4l-dev v4l-utils qv4l2 v4l2ucp
sudo apt-get install -y curl
sudo apt-get update

echo "** Download opencv-4.1.1"
cd $folder
curl -L https://github.com/opencv/opencv/archive/4.1.1.zip -o opencv-4.1.1.zip
curl -L https://github.com/opencv/opencv_contrib/archive/4.1.1.zip -o opencv_contrib-4.1.1.zip
unzip opencv-4.1.1.zip
unzip opencv_contrib-4.1.1.zip
cd opencv-4.1.1/

echo "** Apply patch"
sed -i 's/include <Eigen\/Core>/include <eigen3\/Eigen\/Core>/g' modules/core/include/opencv2/core/private.hpp

echo "** Building..."
mkdir release
cd release/
cmake -D WITH_CUDA=ON -D CUDA_ARCH_BIN="5.3,6.2,7.2" -D CUDA_ARCH_PTX="" -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib-4.1.1/modules -D WITH_GSTREAMER=ON -D
WITH_LIBV4L=ON -D BUILD_opencv_python2=ON -D BUILD_opencv_python3=ON -D BUILD_TESTS=OFF -D BUILD_PERF_TESTS=OFF -D BUILD_EXAMPLES=OFF -D CMAKE_BUILD_TYPE=RELEASE -
D CMAKE_INSTALL_PREFIX=/usr/local ..
make -j3
sudo make install
echo 'export PYTHONPATH=$PYTHONPATH:'$PWD'/python_loader/' >> ~/.bashrc
source ~/.bashrc

echo "** Install opencv-4.1.1 successfully"
echo "** Bye :)"
with folder to download and compile the OpenCV package.
```

You can go to the repository updated script for installing openCV on jetsons:

https://github.com/AastaNV/JEP/tree/master/script

Note that OpenCV installation can take several hours.
Tip: You can edit the script to build in parallel with **make -j 4**
In case of compilation errors, Try run installation again to continue, or switch back to make without parallism.

ROS Installation

ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

Installation guide: http://wiki.ros.org/melodic/Installation/Ubuntu

Command for installation

```
sudo apt-get install devscripts build-essential

sudo apt install ros-melodic-desktop

sudo apt-get source ros-kinetic-opencv3

sudo echo "deb-src http://packages.ros.org/ros/ubuntu xenial main" >>
/etc/apt/sources.list.d/ros-latest.list
sudo apt-get update
sudo apt-get source ros-kinetic-opencv3
```

if you see the error :
Err:15 http://packages.ros.org/ros/ubuntu xenial InRelease
  The following signatures couldn't be verified because the public key is not available:
NO_PUBKEY F42ED6FBAB17C654
Reading package lists... Done
W: GPG error: http://packages.ros.org/ros/ubuntu xenial InRelease: The following signatures couldn't be verified because the public key is not available: NO_PUBKEY F42ED6FBAB17C654
E: The repository 'http://packages.ros.org/ros/ubuntu xenial InRelease' is not signed.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.

You can fix by run the command:

```
curl http://repo.ros2.org/repos.key | sudo apt-key add -
```

5) **Dependancies installtion for ORB-SLAM2**

Pangolin

Pangolin is a lightweight portable rapid development library for managing OpenGL display / interaction and abstracting video input.

You can download and install by instructions in the link:

https://github.com/stevenlovegrove/Pangolin.

```
Commands for install with command line:

git clone https://github.com/stevenlovegrove/Pangolin.git

cd Pangolin

mkdir build
cd build
cmake ..
cmake --build .
```

BLAS

Basic Linear Algebra Subprograms are routines that provide standard building blocks for performing basic vector and matrix operations.

LAPACK

Linear Algebra Package is a standard software library for numerical linear algebra. It provides routines for solving systems of linear equations and linear least squares, eigenvalue problems, and singular value decomposition.

```
Commands for BLAS and LAPACK installations in Ubuntu:

sudo apt-get install libblas-dev
sudo apt-get install liblapack-dev
```

Eigen3

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.

Installation instructions can be found at: http://eigen.tuxfamily.org .

For ORB SLAM required at least version 3.1.0.

You can install by the following commands:

```
git clone https://gitlab.com/libeigen/eigen.git

cd eigen

mkdir build

cd build
cmake ..
make
sudo make install
```

PCL for ROS

Point Cloud Library ROS interface stack. PCL-ROS is the preferred bridge for 3D applications involving n-D Point Clouds and 3D geometry processing in ROS.

You can install by run the following commands:

```
sudo apt-get install libopenni2-dev
sudo apt-get install ros-melodic-pcl-ros
sudo apt-get install python-vtk – not available
```

Build SLAM

Clone the repo and execute the build script for normal ORB-SLAM2:

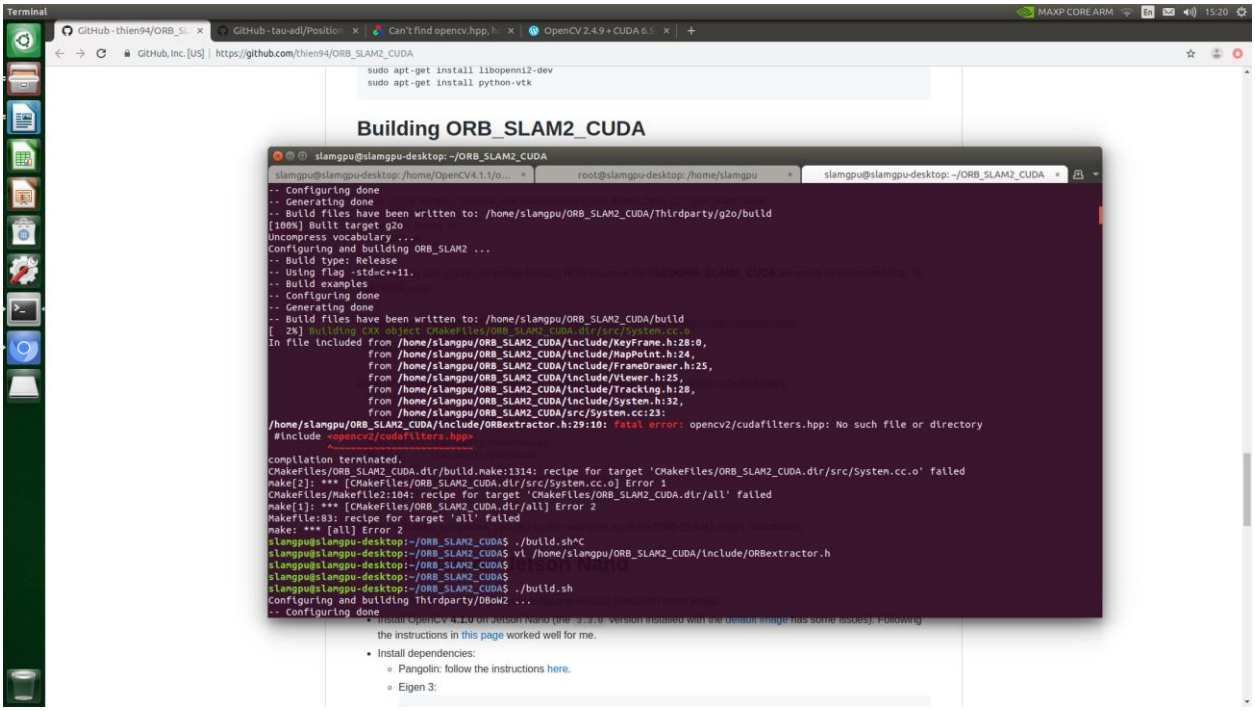
```
git clone https://github.com/hoangthien94/ORB_SLAM2_CUDA.git ORB_SLAM2_CUDA
cd ORB_SLAM2_CUDA
chmod +x build.sh
./build.sh
```

Solutions to porting issues with OpenCV 4.1

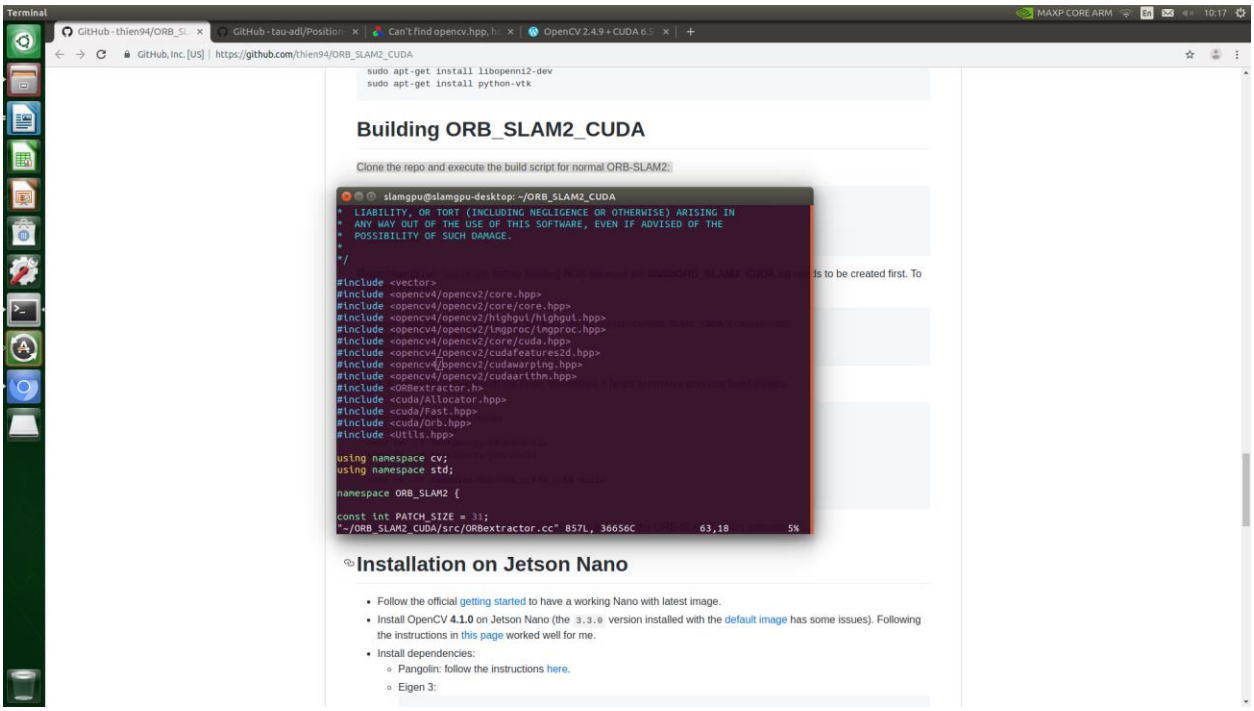
1) Issue - Include from OpenCv2 is not recognized.

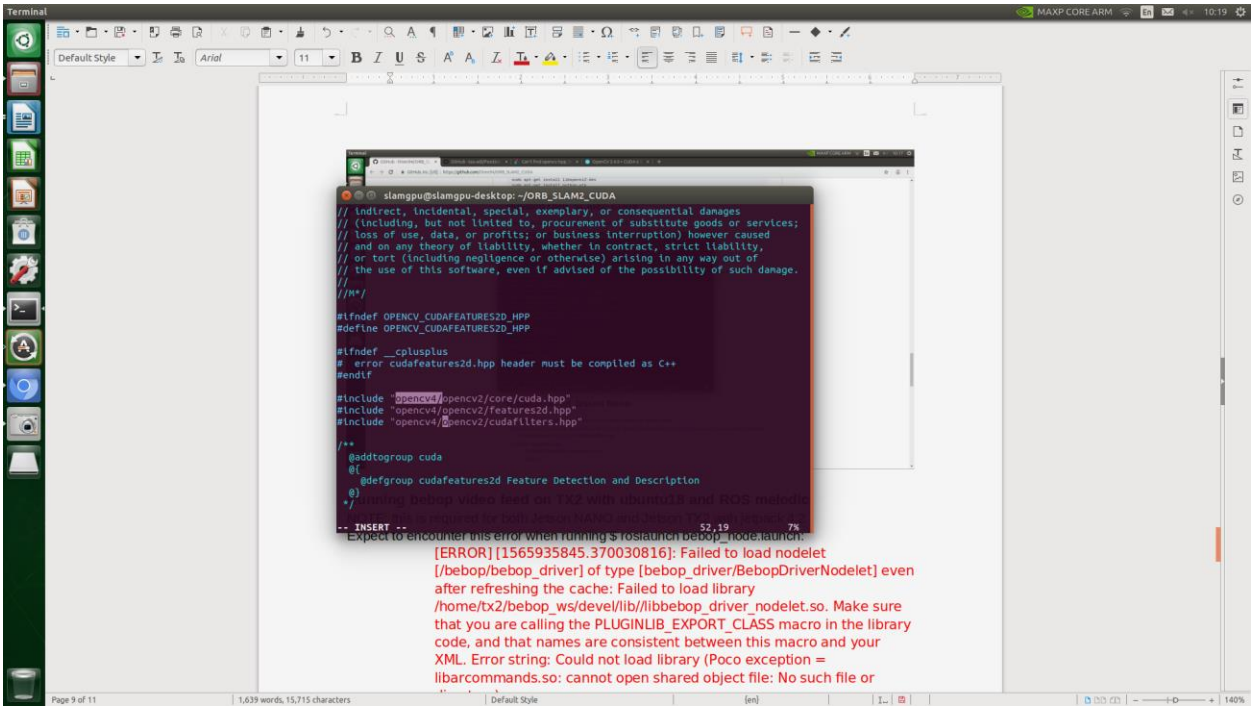
In the files:

/usr/local/include/opencv4/opencv2/cudafeatures2d.hpp
/home/slamgpu/ORB_SLAM2_CUDA/src/ORBextractor.cc
/home/slamgpu/ORB_SLAM2_CUDA/include/ORBextractor.h



Solution: **A**dd to include <opencv4/opencv2/cudaFilters.hpp to all the includes to solve the issue.


Terminal
GitHub - thien94/ORB_SL
GitHub - tau-adl/Position
Can't find opencv.hpp, h
OpenCV 2.4.9 + CUDA 6.5
GitHub, Inc. [US] | https://github.com/thien94/ORB_SLAM2_CUDA
sudo apt-get install libopenni2-dev
sudo apt-get install python-vtk
Building ORB_SLAM2_CUDA
Clone the repo and execute the build script for normal ORB-SLAM2:
slamgpu@slamgpu-desktop: ~/ORB_SLAM2_CUDA
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
* ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*/
#include <vector>
#include <opencv4/opencv2/core.hpp>
#include <opencv4/opencv2/core/core.hpp>
#include <opencv4/opencv2/highgui/highgui.hpp>
#include <opencv4/opencv2/imgproc/imgproc.hpp>
#include <opencv4/opencv2/core/cuda.hpp>
#include <opencv4/opencv2/cudafeatures2d.hpp>
#include <opencv4/opencv2/cudawarping.hpp>
#include <opencv4/opencv2/cudaarithm.hpp>
#include <ORBextractor.h>
#include <cuda/Allocator.hpp>
#include <cuda/Fast.hpp>
#include <cuda/Orb.hpp>
#include <Utils.hpp>
using namespace cv;
using namespace std;
namespace ORB_SLAM2 {
const int PATCH_SIZE = 31;
"~/ORB_SLAM2_CUDA/src/ORBextractor.cc" 857L, 36656C 63,18 5%
Installation on Jetson Nano
Follow the official getting started to have a working Nano with latest image.
Install OpenCV 4.1.0 on Jetson Nano (the 3.3.0 version installed with the default image has some issues). Following the instructions in this page worked well for me.
Install dependencies:
Pangolin: follow the instructions here.
Eigen 3:

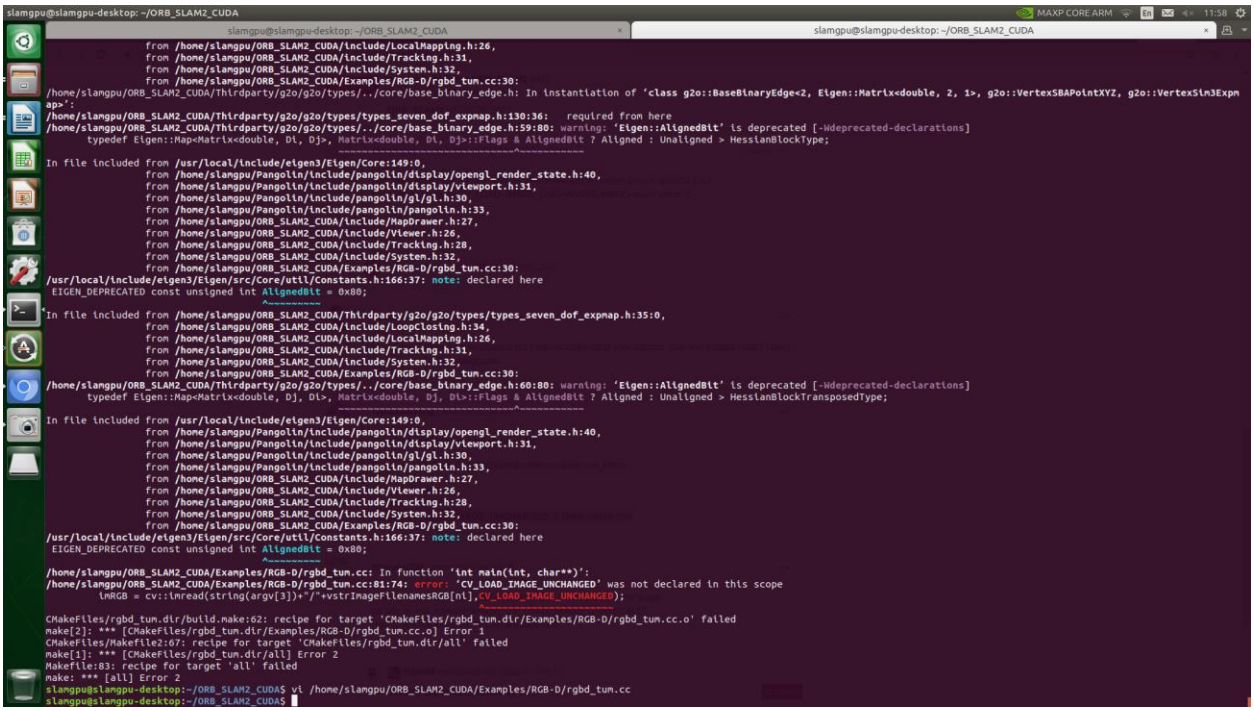

Terminal
MAXP CORE ARM
10:19
Default Style
Arial
11
slamgpu@slamgpu-desktop: ~/ORB_SLAM2_CUDA
// indirect, incidental, special, exemplary, or consequential damages
// (including, but not limited to, procurement of substitute goods or services;
// loss of use, data, or profits; or business interruption) however caused
// and on any theory of liability, whether in contract, strict liability,
// or tort (including negligence or otherwise) arising in any way out of
// the use of this software, even if advised of the possibility of such damage.
//
//M*/
#ifndef OPENCV_CUDAFEATURES2D_HPP
#define OPENCV_CUDAFEATURES2D_HPP
#ifndef __cplusplus
error cudafeatures2d.hpp header must be compiled as C++
#endif
#include "opencv4/opencv2/core/cuda.hpp"
#include "opencv4/opencv2/features2d.hpp"
#include "opencv4/opencv2/cudafilters.hpp"
/**
@addtogroup cuda
@{
@defgroup cudafeatures2d Feature Detection and Description
@}
*/
-- INSERT -- 52,19 7%
Expect to encounter this error when running $ roslaunch bebop_node.launch:
[ERROR] [1565935845.370030816]: Failed to load nodelet [/bebop/bebop_driver] of type [bebop_driver/BebopDriverNodelet] even after refreshing the cache: Failed to load library /home/tx2/bebop_ws/devel/lib//libbebop_driver_nodelet.so. Make sure that you are calling the PLUGINLIB_EXPORT_CLASS macro in the library code, and that names are consistent between this macro and your XML. Error string: Could not load library (Poco exception = libarcommands.so: cannot open shared object file: No such file or
Page 9 of 11
1,639 words, 15,715 characters
Default Style
(en)
140%

2) Issue - library is missing - /usr/bin/ld: cannot find -lnvToolsExt

Solution: run the command
sudo ln -s /usr/local/cuda-10.0/lib64/libnvToolsExt.so /usr/lib/libnvToolsExt.so

https://github.com/yunchih/ORB-SLAM2-GPU2016-final/issues/6

3) Issue /home/slamgpu/ORB_SLAM2_CUDA/Examples/RGB-D/rgbd_tum.cc: In function ‘int main(int, char**)’:
/home/slamgpu/ORB_SLAM2_CUDA/Examples/RGB-D/rgbd_tum.cc:81:74: error: ‘CV_LOAD_IMAGE_UNCHANGED’ was not declared in this scope
imRGB = cv::imread(string(argv[3])+"/"+vstrImageFilenamesRGB[ni],CV_LOAD_IMAGE_UNCHANGED);



Solution:
Add to example cc file the following define
#define CV_LOAD_IMAGE_UNCHANGED -1 //new added one

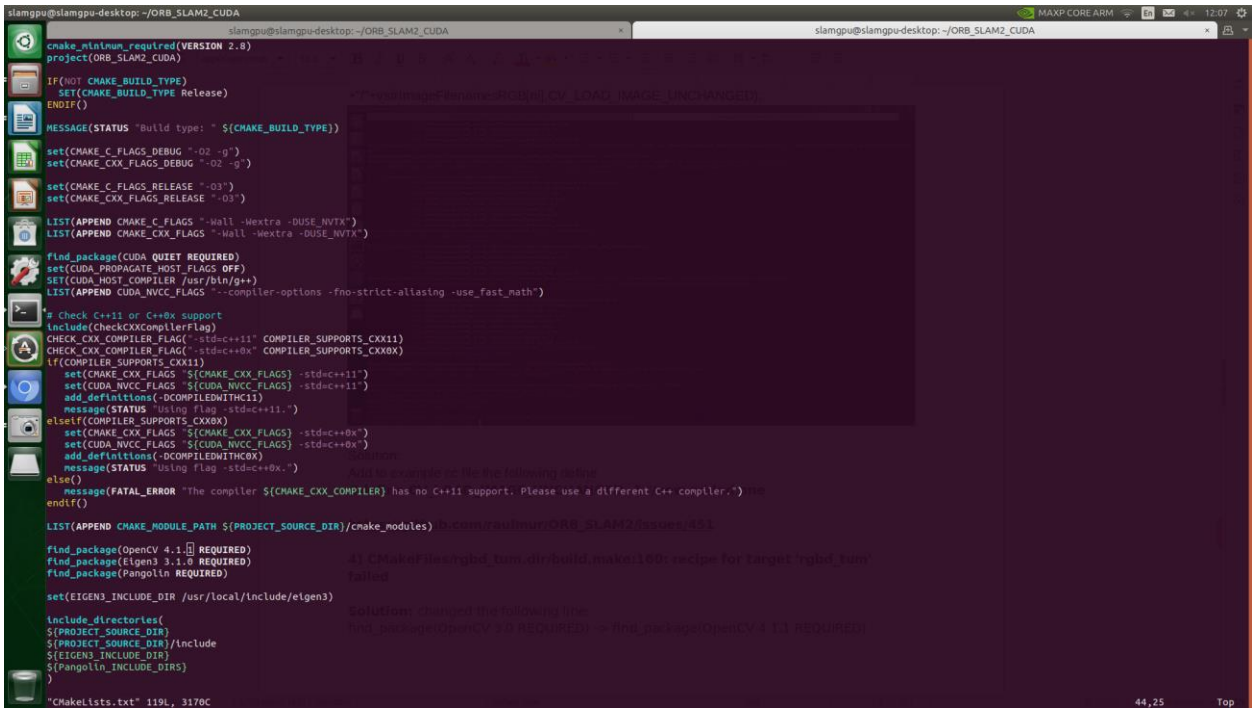
https://github.com/raulmur/ORB_SLAM2/issues/451

4) Issue - CMakeFiles/rgbd_tum.dir/build.make:160: recipe for target 'rgbd_tum' failed

Solution: changed the following line:
find_package(OpenCV 3.0 REQUIRED) -> find_package(OpenCV 4.1.1 REQUIRED)

in the file CMakeLists.txt

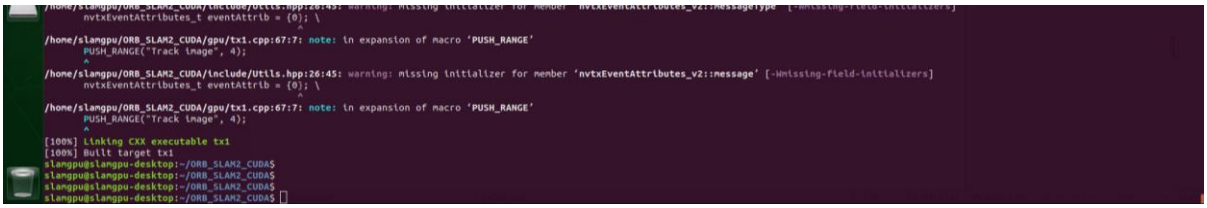
https://github.com/raulmur/ORB_SLAM2/issues/209



5) /home/slamgpu/ORB_SLAM2_CUDA/Examples/Stereo/stereo_kitti.cc: In function ‘int main(int, char**)’:
/home/slamgpu/ORB_SLAM2_CUDA/Examples/Stereo/stereo_kitti.cc:70:47: error:
‘CV_LOAD_IMAGE_UNCHANGED’ was not declared in this scope
imLeft = cv::imread(vstrImageLeft[ni],CV_LOAD_IMAGE_UNCHANGED);

Solution: For the next examples should also add the macro to fix the error:
#define CV_LOAD_IMAGE_UNCHANGED -1 //new added one

After fix all the issues we should see the follow:



Remember to run `build.sh` before building ROS because
the **lib/libORB_SLAM2_CUDA.so** needs to be created first. To build ROS node:

Next step to build ROS by following commands:

export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:/home/slamgpu/ORB_SLAM2_CUDA/Examples/ROS
chmod +x build_ros.sh
./build_ros.sh

Pay attention to the path for ROS
In case of issues run the following commands:
rosdep init
rosdep update

When building finished successfully we will show the follow:

Done.

If you have problem with the build, sometimes it helps to remove previous build folders:

```
# in ORB_SLAM2_CUDA folder
sudo rm -rf build
sudo rm -rf Thirdparty/DBoW2/build
sudo rm -rf Thirdparty/g2o/build
./build.sh
sudo rm -rf Examples/ROS/ORB_SLAM2_CUDA/build
./build_ros.sh
```

Link to web tutorial: https://github.com/thien94/ORB_SLAM2_CUDA

Performance Tip commands

These commands help to increase the Jetson performance.
You should see the fan be active.

- Set power mode to MAXN: (once, saved after reboot)
  sudo nvpmodel -m 0
- Set clocks to high: (every reboot)
  sudo jetson_clocks

Run SLAM application on the target

Do the following steps:

1) Add to end of file ~/.bashrc the following lines:
export PATH=/usr/local/cuda-10.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64:$LD_LIBRARY_PATH
source /opt/ros/melodic/setup.bash
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:/home/slamgpu/ORB_SLAM2_CUDA/Examples/ROS

2) Put the following highlight content to file ros_mono.launch in the path:

~/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/launch/ros_mono.launch

```
<?xml version="1.0"?>
<launch>

   <arg name="vocabularty_path" default="$(find ORB_SLAM2_CUDA)/../../../Vocabulary/ORBvoc.txt" />
   <arg name="camera_setting_path" default="$(find ORB_SLAM2_CUDA)/conf/euroc.yaml" />
   <arg name="bUseViewer" default="true" />
   <arg name="bEnablePublishROSTopic" default="false" />

   <node name="ORB_SLAM2_CUDA" pkg="ORB_SLAM2_CUDA" type="Mono" output="screen"
      args="$(arg vocabularty_path) $(arg camera_setting_path) $(arg bUseViewer) $(arg
bEnablePublishROSTopic)">
      <remap from="/camera/image_raw" to="/cam0/image_raw"/>
      </node>

</launch>
```

3) Add the file euroc.yaml to conf directory as ~/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/conf/euroc.yaml You can see the content below

```
%YAML:1.0

#--------------------------------------------------------------------------------------------
# Camera Parameters. Adjust them!
#--------------------------------------------------------------------------------------------

# Camera calibration and distortion parameters (OpenCV)
Camera.fx: 435.2046959714599
Camera.fy: 435.2046959714599
Camera.cx: 367.4517211914062
Camera.cy: 252.2008514404297

Camera.k1: -0.28340811
Camera.k2: 0.07395907
Camera.p1: 0.00019359
Camera.p2: 1.76187114e-05

Camera.width: 752
Camera.height: 480

# Camera frames per second
Camera.fps: 20.0

# stereo baseline times fx
Camera.bf: 47.90639384423901

# Color order of the images (0: BGR, 1: RGB. It is ignored if images are grayscale)
Camera.RGB: 1

# Close/Far threshold. Baseline times.
ThDepth: 35

#--------------------------------------------------------------------------------------------
# Stereo Rectification. Only if you need to pre-rectify the images.
# Camera.fx, .fy, etc must be the same as in LEFT.P
#--------------------------------------------------------------------------------------------
LEFT.height: 480
LEFT.width: 752
LEFT.D: !!opencv-matrix
   rows: 1
   cols: 5
   dt: d
   data:[-0.28340811, 0.07395907, 0.00019359, 1.76187114e-05, 0.0]
LEFT.K: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [458.654, 0.0, 367.215, 0.0, 457.296, 248.375, 0.0, 0.0, 1.0]
LEFT.R:  !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [0.999966347530033, -0.001422739138722922, 0.008079580483432283, 0.001365741834644127, 0.9999741760894847,
0.007055629199258132, -0.008089410156878961, -0.007044357138835809, 0.9999424675829176]
LEFT.P:  !!opencv-matrix
   rows: 3
   cols: 4
   dt: d
   data: [435.2046959714599, 0, 367.4517211914062, 0,  0, 435.2046959714599, 252.2008514404297, 0,  0, 0, 1, 0]

RIGHT.height: 480
```

```
RIGHT.width: 752
RIGHT.D: !!opencv-matrix
   rows: 1
   cols: 5
   dt: d
   data:[-0.28368365, 0.07451284, -0.00010473, -3.555907e-05, 0.0]
RIGHT.K: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [457.587, 0.0, 379.999, 0.0, 456.134, 255.238, 0.0, 0.0, 1]
RIGHT.R:  !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [0.9999633526194376, -0.003625811871560086, 0.007755443660172947, 0.003680398547259526, 0.9999684752771629, -
0.007035845251224894, -0.007729688520722713, 0.007064130529506649, 0.999945173484644]
RIGHT.P:  !!opencv-matrix
   rows: 3
   cols: 4
   dt: d
   data: [435.2046959714599, 0, 367.4517211914062, -47.90639384423901, 0, 435.2046959714599, 252.2008514404297, 0, 0, 0, 1, 0]

#--------------------------------------------------------------------------------------------
# ORB Parameters
#--------------------------------------------------------------------------------------------

# ORB Extractor: Number of features per image
ORBextractor.nFeatures: 2000

# ORB Extractor: Scale factor between levels in the scale pyramid
ORBextractor.scaleFactor: 1.2

# ORB Extractor: Number of levels in the scale pyramid
ORBextractor.nLevels: 8

# ORB Extractor: Fast threshold
# Image is divided in a grid. At each cell FAST are extracted imposing a minimum response.
# Firstly we impose iniThFAST. If no corners are detected we impose a lower value minThFAST
# You can lower these values if your images have low contrast
ORBextractor.iniThFAST: 20
ORBextractor.minThFAST: 7

#--------------------------------------------------------------------------------------------
# Viewer Parameters
#--------------------------------------------------------------------------------------------
Viewer.KeyFrameSize: 0.05
Viewer.KeyFrameLineWidth: 1
Viewer.GraphLineWidth: 0.9
Viewer.PointSize:2
Viewer.CameraSize: 0.08
Viewer.CameraLineWidth: 3
Viewer.ViewpointX: 0
Viewer.ViewpointY: -0.7
Viewer.ViewpointZ: -1.8
Viewer.ViewpointF: 500
```

Run ORB SLAM with ROS

1)Set clocks to high: (every reboot) - Optional

sudo jetson_clocks

2)Run SLAM application:

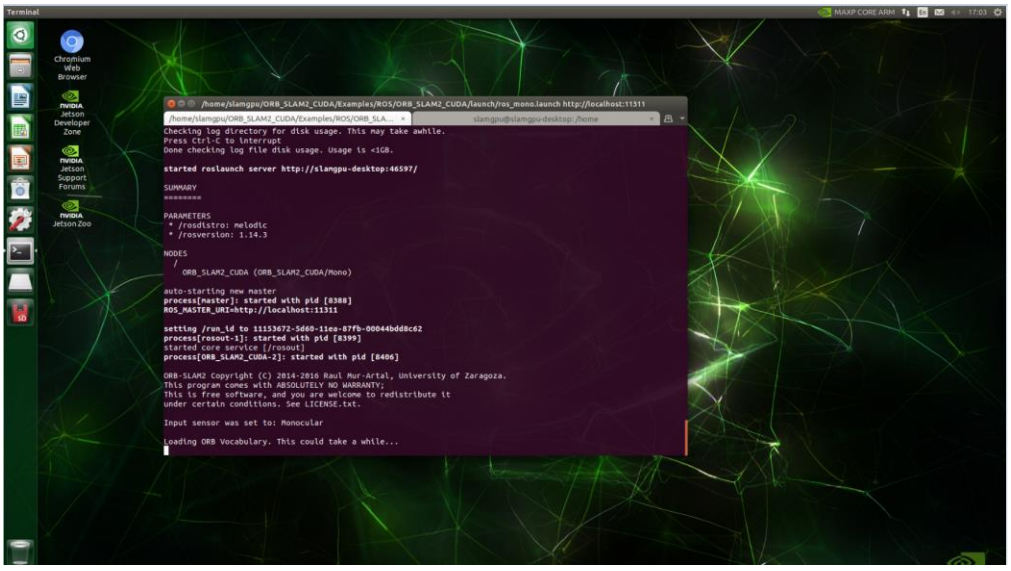
roslaunch /home/slamgpu/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/launch/ros_mono.launch

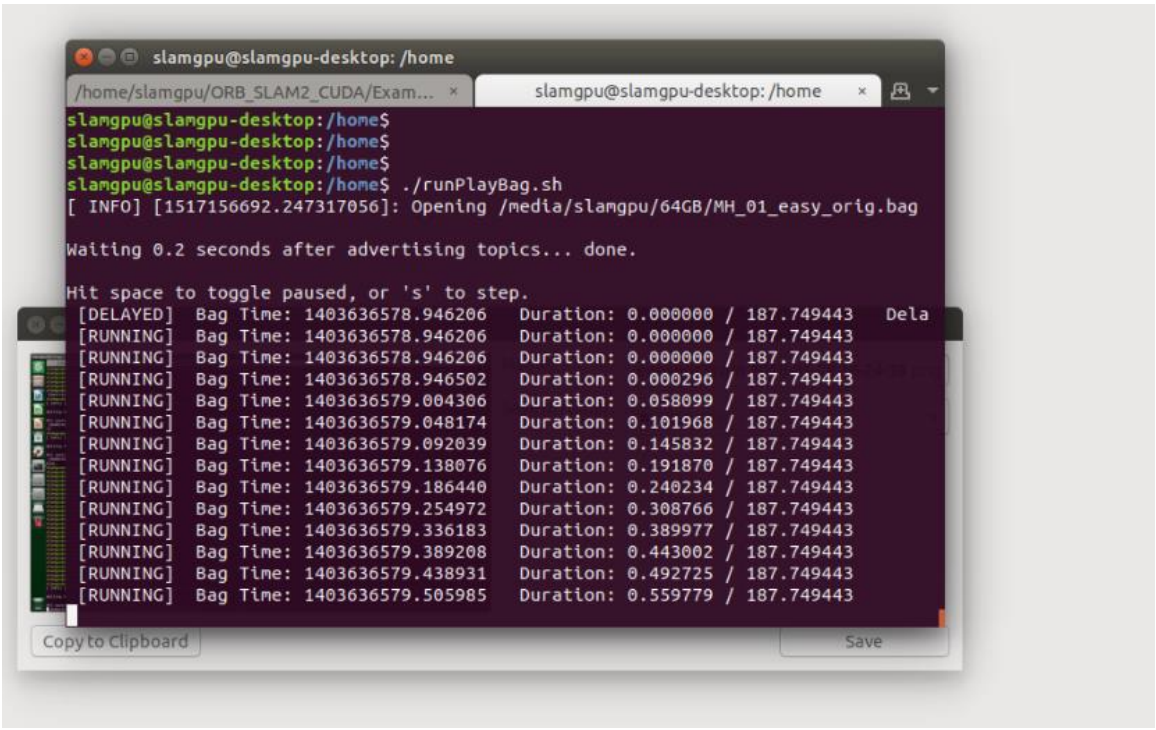
3)Play bag file to SLAM application (bag file located in SDcard)

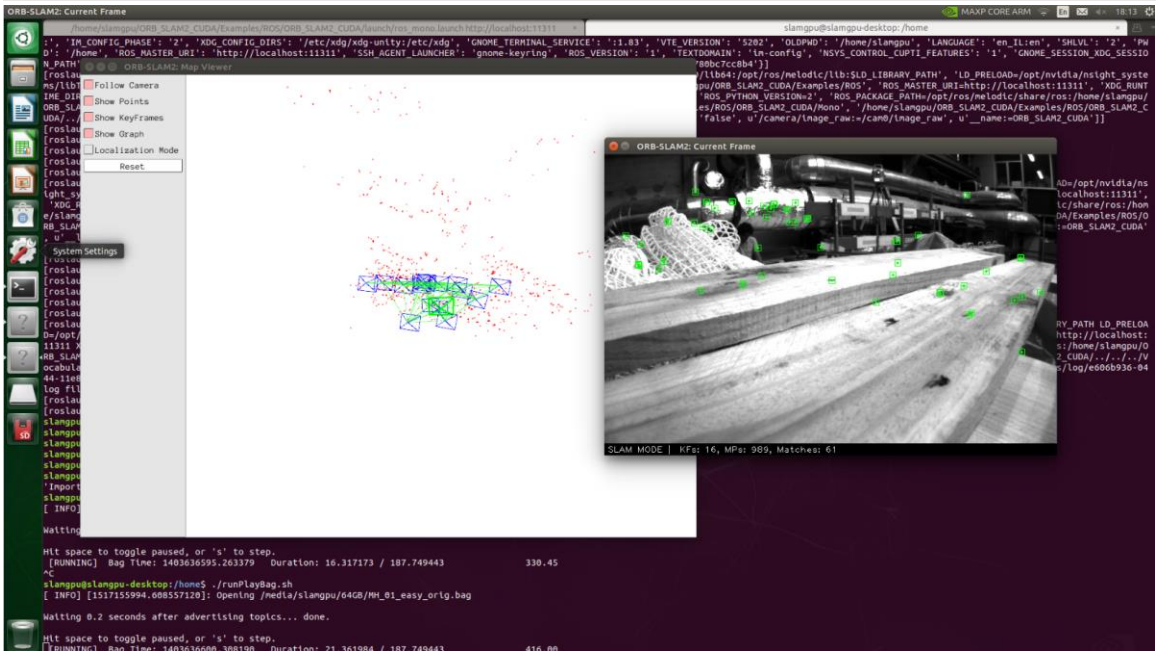
rosbag play /media/slamgpu/64GB/MH_01_easy_orig.bag

You can run it from device\host.

For running in remote (host) you should change the display id.




slamgpu@slamgpu-desktop: /home
/home/slamgpu/ORB_SLAM2_CUDA/Exam...
slamgpu@slamgpu-desktop: /home
slamgpu@slamgpu-desktop:/home$
slamgpu@slamgpu-desktop:/home$
slamgpu@slamgpu-desktop:/home$
slamgpu@slamgpu-desktop:/home$ ./runPlayBag.sh
[INFO] [1517156692.247317056]: Opening /media/slamgpu/64GB/MH_01_easy_orig.bag
Waiting 0.2 seconds after advertising topics... done.
Hit space to toggle paused, or 's' to step.
[DELAYED] Bag Time: 1403636578.946206 Duration: 0.000000 / 187.749443 Dela
[RUNNING] Bag Time: 1403636578.946206 Duration: 0.000000 / 187.749443
[RUNNING] Bag Time: 1403636578.946206 Duration: 0.000000 / 187.749443
[RUNNING] Bag Time: 1403636578.946502 Duration: 0.000296 / 187.749443
[RUNNING] Bag Time: 1403636579.004306 Duration: 0.058099 / 187.749443
[RUNNING] Bag Time: 1403636579.048174 Duration: 0.101968 / 187.749443
[RUNNING] Bag Time: 1403636579.092039 Duration: 0.145832 / 187.749443
[RUNNING] Bag Time: 1403636579.138076 Duration: 0.191870 / 187.749443
[RUNNING] Bag Time: 1403636579.186440 Duration: 0.240234 / 187.749443
[RUNNING] Bag Time: 1403636579.254972 Duration: 0.308766 / 187.749443
[RUNNING] Bag Time: 1403636579.336183 Duration: 0.389977 / 187.749443
[RUNNING] Bag Time: 1403636579.389208 Duration: 0.443002 / 187.749443
[RUNNING] Bag Time: 1403636579.438931 Duration: 0.492725 / 187.749443
[RUNNING] Bag Time: 1403636579.505985 Duration: 0.559779 / 187.749443
Copy to Clipboard
Save


ORB-SLAM2: Current Frame
ORB-SLAM2: Map Viewer
Follow Camera
Show Points
Show KeyFrames
Show Graph
Localization Mode
Reset
System Settings
ORB-SLAM2: Current Frame
SLAM MODE | KFs: 16, MPs: 989, Matches: 61

Run ORB SLAM without ROS

1)Set clocks to high: (every reboot)

sudo jetson_clocks

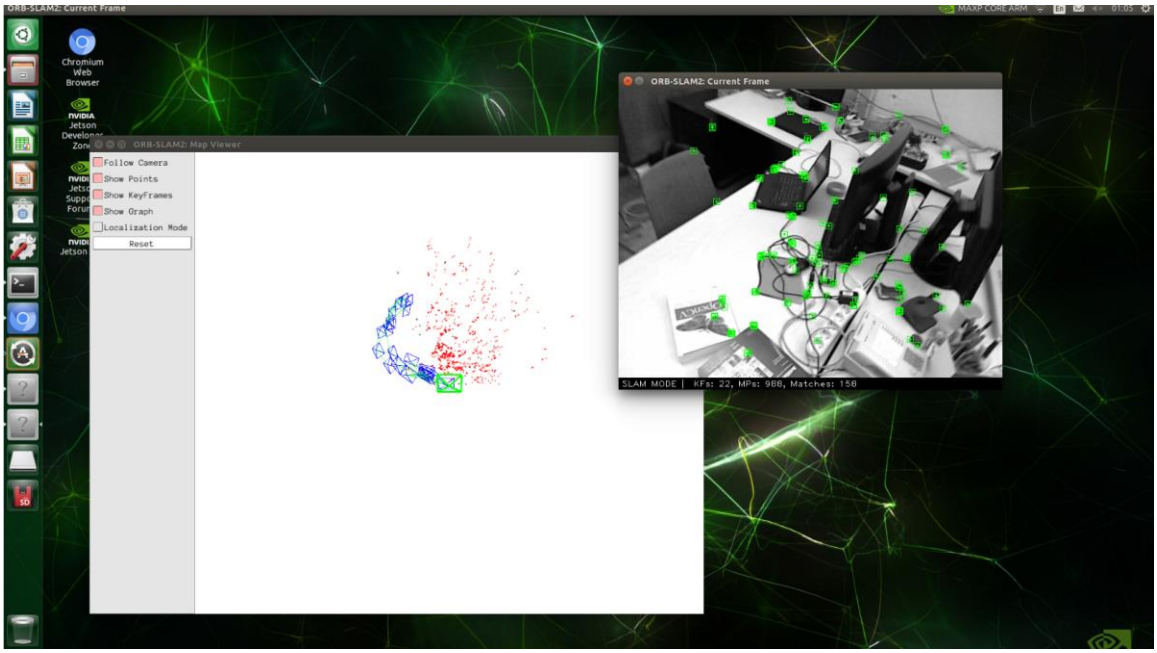
2)Run SLAM application including parameters:

cd /home/slamgpu/ORB_SLAM2_CUDA

./build/mono_tum Vocabulary/ORBvoc.txt Examples/Monocular/TUM1.yaml /media/slamgpu/64GB/rgbd_dataset_freiburg1_desk true

You can run it from device\host.

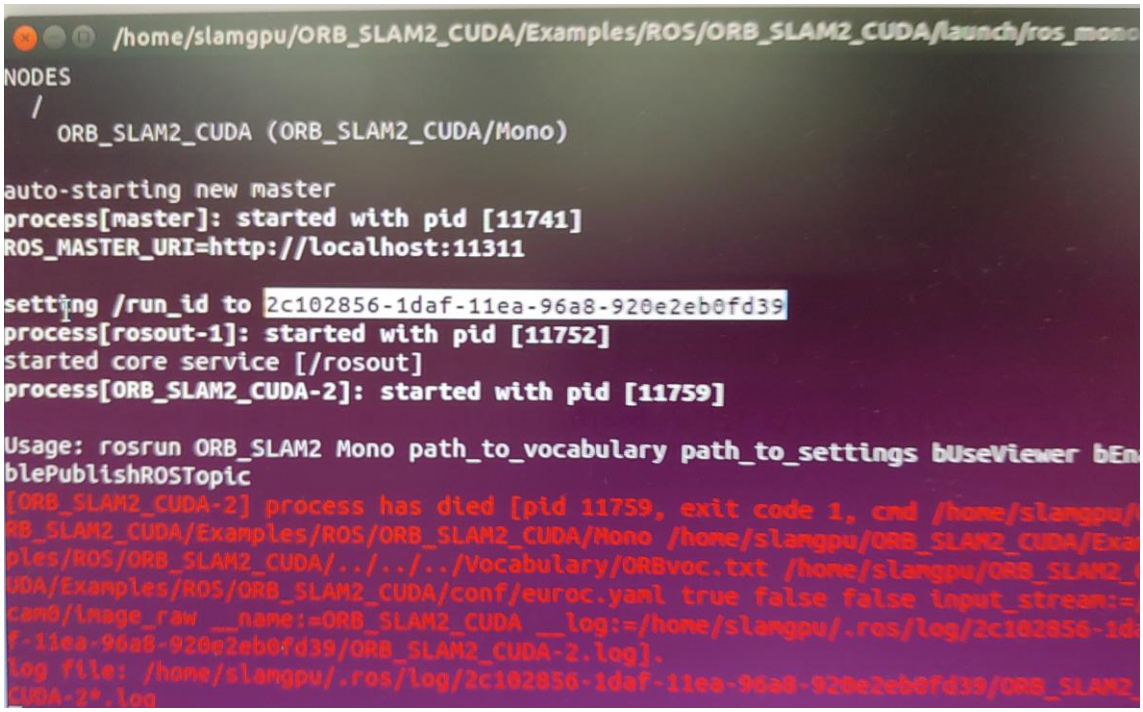
For running in remote (host) you should change the display id.




ORB-SLAM2: Map Viewer
Follow Camera
Show Points
Show KeyFrames
Show Graph
Localization Mode
Reset
SLAM MODE | KFs: 15, MPs: 700, Matches: 127

Configuration issue during SLAM running

In case of error as below you should configure the enviroment and modify the configruation files
1)**~/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/conf/euroc.yaml**
2)**~/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/launch/ros_mono.launch**



ROS Useful commands

You can read about the ROS commands in the following links:

- https://github.com/bosch-io/bcx18-openADx-examples/wiki/Useful-ROS-Commands
- http://wiki.ros.org/ROS/CommandLineTools
- https://subscription.packtpub.com/book/hardware_and_creative/9781782175193/1/ch01lvl1sec15/ros-commands-summary

`roscore command`

- roscore -p <port>: start a roscore process directly on given port

`roslaunch command`

- roslaunch <launchfile>: run a launch file (which starts a roscore if none is running)

`rosnode command`

- `rosnode list`: show all running nodes
- `rosnode info <nodename>`: show infos for node
- `rosnode ping <nodename>`: try to reach a running node

`rosbag command`

- `rosbag info <bagfile>`: show all topics contained in a bagfile
- `rosbag play <bagfile>`: play all messages recorded in a bagfile
- `rosbag record -o <bagfile> -a`: record all topics into a bagfile

`rostopic command`

- `rostopic echo <topic_name>`: listen to topic
- `rostopic list`: show all published and subscribed topics
- `rostopic pub <topic_name> <message_type> 'data'`: manually publish a message to a topic, e.g. `rostopic pub /flip_image std_msgs/Bool '{data:true}'` or `rostopic pub /mavros/rc/override mavros_msgs/OverrideRCIn '[yaw, 0, throttle, 0, 0, 0, 0, 0]'`
- `rostopic info <topic_name>`: print infos about topic
- `rostopic type <topic_name>`: the type of the topic

`rvizrviz tool`

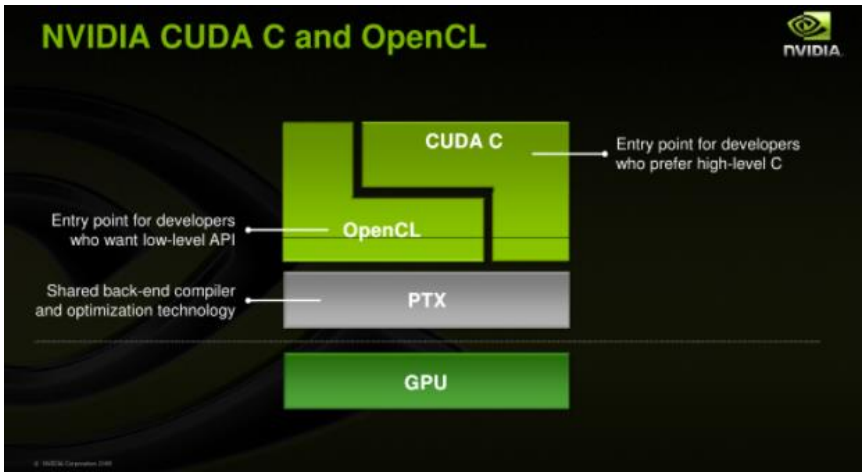
Tool for visualizing sensor data

`rqt_console`

GUI for displaying and filtering ROS messages

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).

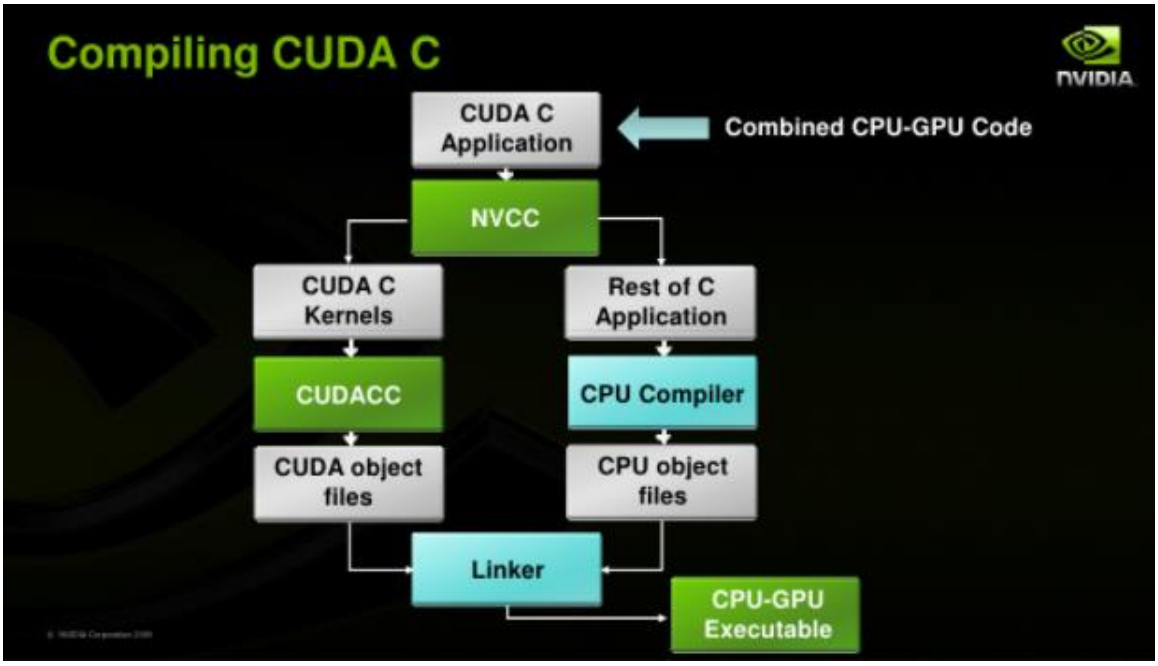


CUDA was developed with several design goals in mind:

- Provide a small set of extensions to standard programming languages, like C, that enable a straightforward implementation of parallel algorithms. With CUDA C/C++, programmers can focus on the task of parallelization of the algorithms rather than spending time on their implementation.
- Support heterogeneous computation where applications use both the CPU and GPU. Serial portions of applications are run on the CPU, and parallel portions are offloaded to the GPU. As such, CUDA can be incrementally applied to existing applications. The CPU and GPU are treated as separate devices that have their own memory spaces. This configuration also allows simultaneous computation on the CPU and GPU without contention for memory resources.

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 Compilation Flow



Cuda application is compiled with NVCC compiler which create the object files for CPU and GPU. These object files are linked to execution file which ready for running on CPU-GPU enviroment.

Cuda Entry Point and Reference
NVIDIA Site:

Cuda toolkit link: https://docs.nvidia.com/cuda/index.html

Installation Guide: https://docs.nvidia.com/cuda/index.html#installation-guides

Programming Guide: https://docs.nvidia.com/cuda/index.html#programming-guides

Cuda API: https://docs.nvidia.com/cuda/index.html#cuda-api-references

Cuda Samples: https://docs.nvidia.com/cuda/cuda-samples/index.html

Debuging API: https://docs.nvidia.com/cuda/debugger-api/index.html

From Internet:

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

Communication between Host and Device

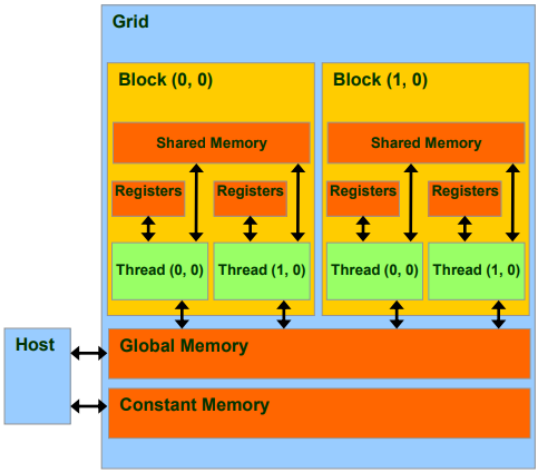
The communication between the Host (CPU) and Device (GPU) is occurred by transfer memory buffers and threads synchronization (barriers) for indication all the threads are finished their work.
These buffers are used to transfer data to GPU for perform parallel calculations and then the buffer results is returned to the CPU.
When the one thread achieved the desirable result we can update other threads to stop their calculation and jump to the barrier for waiting all the other threads will finish their work and when all the threads ready to complete the task and return the results to the host.

Memories Hierarchy

1. Registers ~1 cycle
2. Shared memory ~5 cycles
3. Global memory ~500 cycles

There is one more memory for constants ~5 cycles with caching.

Host can copy data (RW) to the global memory.



The bus between CPU and GPU is PCIe.

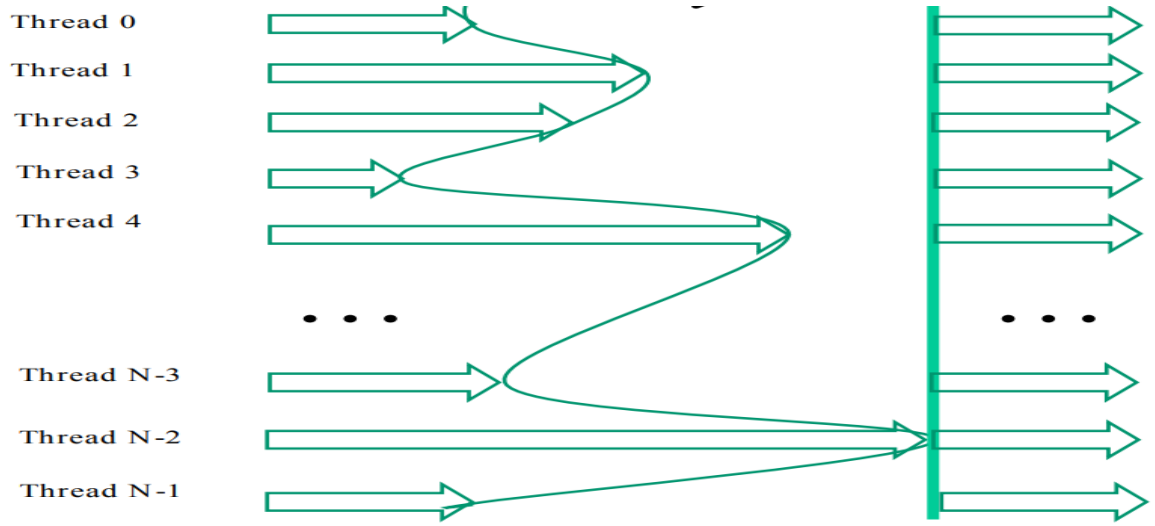
By using shared memory we can increase the memory bandwidth as we can use each memory data several times and improve the bus efficiency.

Cuda API for transfer data

Fucntion	Usage	Note
cudaMalloc()	Allocates buffer in the device memory	
cudaFree()	Free allocated device buffer	
cudaMemcpy()	memory data transfer (Copy data from\to Host to\from Device)	Two flags: cudaMemcpyHostToDevice cudaMemcpyDeviceToHost

Cuda API for synchronization

Fucntion	Usage	Note
__syncthreads()	All threads in the same block must reach the __syncthreads() before any can move on	



ORB SLAM GPU Cuda Code

Code Structure

src\cuda\Fast_gpu.cu – Fast feature detection implementation using GPU.
src\cuda\Orb_gpu.cu – ORB compute descriptors using GPU.
src\cuda\Allocator_gpu.cu – allocation/free buffer in GPU.

src\ORBextractor.cc – CPU code entry point for CPU->GPU calls.

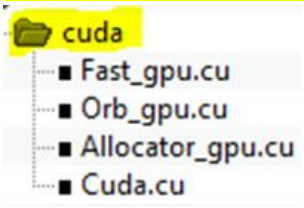
Jetson code path (Target - Tetson): /home/slamgpu/ORB_SLAM2_CUDA

Main Kernel Cuda functions

Tasks from CPU to GPU to improve performance are:

- calcOrb_kernel()
- tileCalcKeypoints_kernel()
- IC_Angle_kernel()
- addBorder_kernel()

The following files contains these function:



I will explain the first two mains kernel functions:

tileCalcKeypoints_kernel()

```
__global__ void tileCalcKeypoints_kernel(const PtrStepSzb img, short2 * kpLoc, float * kpScore, const unsigned int maxKeypoints, const int highThreshold, const int lowThreshold, PtrStepi scoreMat,
unsigned int * counter_ptr) {
    const int j = threadIdx.x + blockIdx.x * blockDim.x + 3;
    const int i = (threadIdx.y + blockIdx.y * blockDim.y) * 4 + 3;
    const int tid = threadIdx.y * blockDim.x + threadIdx.x;
    __shared__ bool hasKp;
    if (tid == 0) {
      hasKp = false;
    }

    bool isKp[4] = {0};
    for (int t = 0; t < 4; ++t) {
      if (i+t < img.rows - 3 && j < img.cols - 3) {
        isKp[t] = isKeyPoint2(img, i+t, j, highThreshold, scoreMat);
      }
    }
    // barrieer
    __syncthreads();
    for (int t = 0; t < 4; ++t) {
      if (isKp[t]) {
        isKp[t] = false;
        short2 loc = make_short2(j, i+t);
        if (isMax(loc, scoreMat)) {
          int score = scoreMat(loc.y, loc.x);
          hasKp = true;
          const unsigned int ind = atomicInc(counter_ptr, (unsigned int)(-1));
          if (ind < maxKeypoints) {
            kpLoc[ind] = loc;
            kpScore[ind] = static_cast<float>(score);
          }
        }
      }
    }
    // barrieer
    __syncthreads();
    if (hasKp) return ;

    // lower the threshold and try again
    for (int t = 0; t < 4; ++t) {
      if (i+t < img.rows - 3 && j < img.cols - 3) {
        isKp[t] = isKeyPoint2(img, i+t, j, lowThreshold, scoreMat);
      }
    }
    // barrieer
    __syncthreads();
    for (int t = 0; t < 4; ++t) {
      if (isKp[t]) {
        short2 loc = make_short2(j, i+t);
        if (isMax(loc, scoreMat)) {
          int score = scoreMat(loc.y, loc.x);
          const unsigned int ind = atomicInc(counter_ptr, (unsigned int)(-1));
          if (ind < maxKeypoints) {
            kpLoc[ind] = loc;
            kpScore[ind] = static_cast<float>(score);
          }
        }
      }
```

This function calculates key points in parallel computation in GPU.

For input image:

```
void GpuFast::detectAsync(InputArray _image) {
  const cv::cuda::GpuMat image = _image.getGpuMat();
  if (scoreMat.empty()) {
    // If it is not empty, then it's already allocated by previous iteration
    // and I ASSUME THE DIMENSIONS ARE CONSISTENT ACROSS EVERY ITERATION
    // else THIS WILL BREAK
    scoreMat = GpuMat(image.size(), CV_32SC1);
  }
  scoreMat.setTo(Scalar::all(0), cvStream);
  checkCudaErrors( cudaMemsetAsync(counter_ptr, 0, sizeof(unsigned int), stream) );

  dim3 dimBlock(32, 8);
  dim3 dimGrid(divUp(image.cols, dimBlock.x), divUp(image.rows, dimBlock.y * 4));
  tileCalcKeypoints_kernel<<<dimGrid, dimBlock, 0, stream>>>(image, kpLoc, kpScore, maxKeypoints, highThreshold,
lowThreshold, scoreMat, counter_ptr);
  checkCudaErrors( cudaGetLastError() );
}
```

We calculate the Grid size (number of blocks) and block size (Threads) per image size.

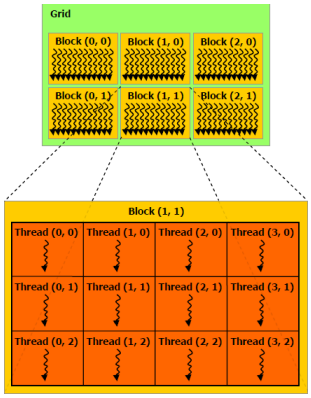


Image source : https://www.3dgep.com/cuda-thread-execution-model/

In our case threads per block are 32*8

By dividing image size cordination by coresponding thread count we get the blocks per X or Y.

For dim3 dimGrid(divUp(image.cols, dimBlock.x), divUp(image.rows, dimBlock.y * 4));

Number blocks in X dimention = divUp(image.cols, dimBlock.x)

Number blocks in Y dimention = divUp(image.rows, dimBlock.y * 4))

For Y dimention we create 4 calutlations per thread.

The key point calculation per thread is divided to several steps:

1) Indexs calculation per thread
2) thread 0 in the block set shared memory flag to 0 which indicate if key point is found by threads
3) Check 4 potentional key points per thread
4) Syncronize all the threads in block
5) Verify the potentional key points results
6) Syncronize all the threads
7) if any key point is found all the threads will finish caluclation
8) if not found key point retry steps 3-7 with lower threshold

The target of all threads is to collect several key points.
When key point is found and is verfied then atomic counter is incremented and the key point is saved with his score.
The counter is shared between all the threads in all the blocks.
The array for all key points is shared between all the threads in all the blocks.

calcOrb_kernel()

Calculation per key point

```
void GpuOrb::launch_async(InputArray _image, const KeyPoint * _keypoints, const int npoints) {
  if (npoints == 0) {
    POP_RANGE;
    return ;
  }
  const GpuMat image = _image.getGpuMat();

  checkCudaErrors( cudaMemcpyAsync(keypoints, _keypoints, sizeof(KeyPoint) * npoints, cudaMemcpyHostToDevice, stream) );
  desc = descriptors.rowRange(0, npoints);
  desc.setTo(Scalar::all(0), cvStream);

  dim3 dimBlock(32);
  dim3 dimGrid(npoints);
  calcOrb_kernel<<<dimGrid, dimBlock, 0, stream>>>(image, keypoints, npoints, desc);
  checkCudaErrors( cudaGetLastError() );
}
```

Block size 32 threads
Grid size is per number of key points

```
 __global__ void calcOrb_kernel(const PtrStepb image, KeyPoint * keypoints, const int npoints,
PtrStepb descriptors) {
    int id = blockIdx.x;
    int tid = threadIdx.x;
    if (id >= npoints) return;

    const KeyPoint &kpt = keypoints[id];
    short2 loc = make_short2(kpt.pt.x, kpt.pt.y);
    const Point * pattern = ((Point *)c_pattern) + 16 * tid;

    uchar * desc = descriptors.ptr(id);
    const float factorPI = (float)(CV_PI/180.f);
    float angle = (float)kpt.angle * factorPI;
    float a = (float)cosf(angle), b = (float)sinf(angle);

    int t0, t1, val;
    t0 = GET_VALUE(0); t1 = GET_VALUE(1);
    val = t0 < t1;
    t0 = GET_VALUE(2); t1 = GET_VALUE(3);
    val |= (t0 < t1) << 1;
    t0 = GET_VALUE(4); t1 = GET_VALUE(5);
    val |= (t0 < t1) << 2;
    t0 = GET_VALUE(6); t1 = GET_VALUE(7);
    val |= (t0 < t1) << 3;
    t0 = GET_VALUE(8); t1 = GET_VALUE(9);
    val |= (t0 < t1) << 4;
    t0 = GET_VALUE(10); t1 = GET_VALUE(11);
    val |= (t0 < t1) << 5;
    t0 = GET_VALUE(12); t1 = GET_VALUE(13);
    val |= (t0 < t1) << 6;
    t0 = GET_VALUE(14); t1 = GET_VALUE(15);
    val |= (t0 < t1) << 7;

    desc[tid] = (uchar)val;
  }
```

Per thread in block there is check for existing key point.
Each thread do calculation on the key point.

Profiling Tools

The Visual Profiler is a graphical profiling tool that displays a timeline of your application's CPU and GPU activity, and that includes an automated analysis engine to identify optimization opportunities.

Old tools:

- **`Nvprof profiling`** tool enables you to collect and view profiling data from the command-line.
- **`Visual Profiler`** allow to view the data in graphical view.

These old tools will be depressed in the future.

New tools: Nvidia tools overview

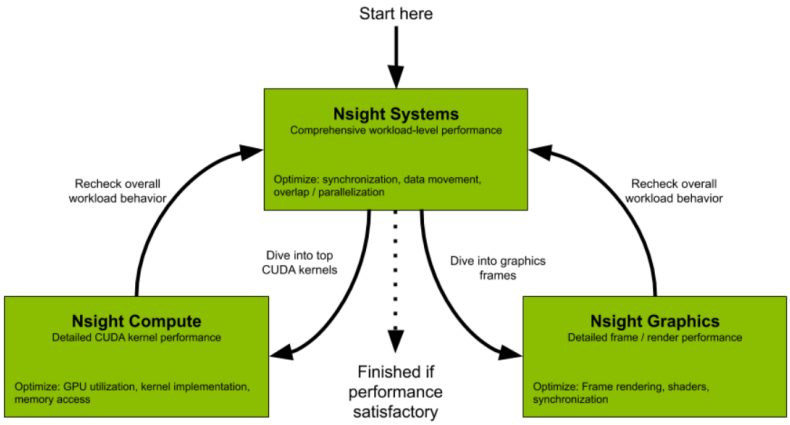


Figure 1. Flowchart describing working with new NVIDIA Nsight tools for performance optimization

NVIDIA Nsight Compute is an interactive kernel profiler for CUDA applications. It provides detailed performance metrics and API debugging via a user interface and command line tool. In addition, its baseline feature allows users to compare results within the tool. Nsight Compute provides a customizable and data-driven user interface and metric collection and can be extended with analysis scripts for post-processing results.

NVIDIA Nsight Systems is a system-wide performance analysis tool designed to visualize an application's algorithms, help you identify the largest opportunities to optimize, and tune to scale efficiently across any quantity or size of CPUs and GPUs; from large server to our smallest SoC.

Nsight Graphics is a standalone developer tool that enables you to debug, profile, and export frames built with Direct3D (11, 12, DXR), Vulkan (1.2, NV Vulkan Ray Tracing Extension), OpenGL, OpenVR, and the Oculus SDK.

Comparisons Table

Tool	System View (OS)	Cuda View	Cuda Debuger	Debug on Remote Host	Debug on Device (Jetson)	For which tasks the tool is optimized
nvprof	X	Y	X	Y	Y	Capture Cuda trace for analysis.
Visual Profiler	X	Y	X	Y	Y	Graphical view
Nsight Compute	X	Y	Y	Y	N	Optimized for Cuda We can use this tool as debugger to follow the code.
Nsight System	Y	High level Cuda information	X	Y	N	Optimized for System view You can see events and timelines for threads in CPU and GPU. In each time which libraries were used. Understand the Cuda calls. Percentage for Computation VS Memory. Context switches in GPU.
Nsight Graphics	?	?	?	Y	N	Allow real-time analysis

How to Install JetPack

Depending on your Jetson device, there are one or two ways to install JetPack.

NVIDIA SDK Manager supports JetPack installation on these Jetson products:

- NVIDIA Jetson Nano modules on a Jetson Nano Developer Kit carrier board
- NVIDIA Jetson TX1 on a Jetson TX2 Developer Kit carrier board
- NVIDIA Jetson TX2 series modules on a Jetson TX2 Developer Kit carrier board
- NVIDIA Jetson AGX Xavier on a Jetson AGX Xavier Developer Kit carrier board

A Linux host computer running Ubuntu Linux x64 version 18.04 or 16.04 is required to run SDK Manager.

Detailed instructions can be found here: https://docs.nvidia.com/sdk-manager/index.html

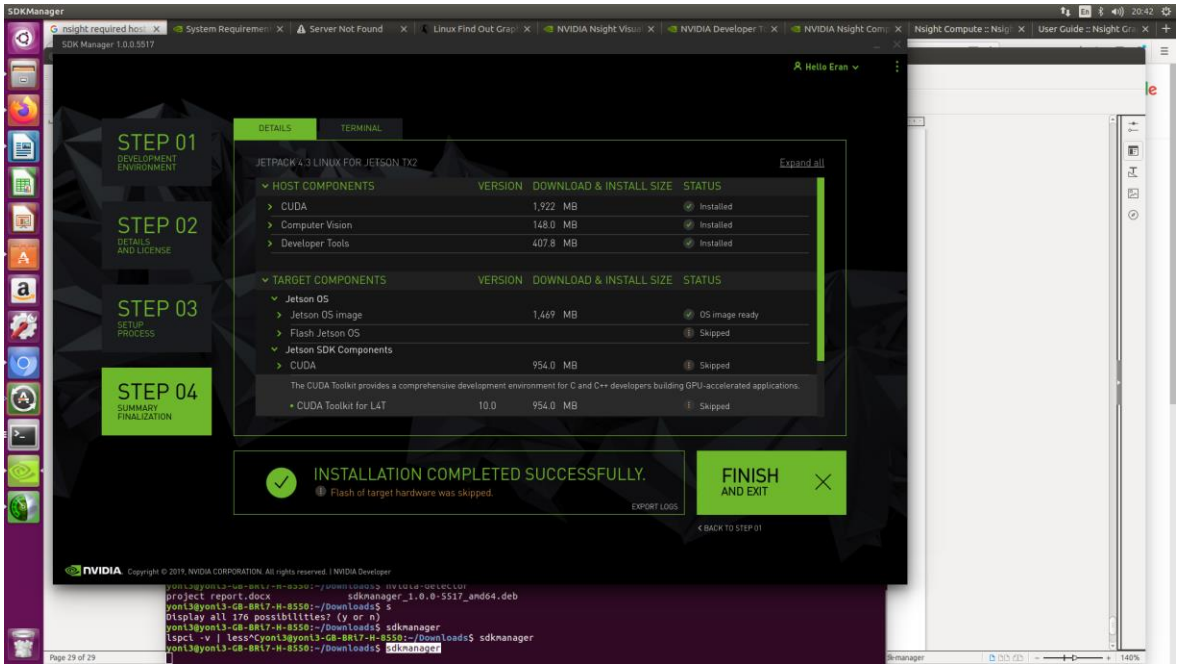
You can install in other options see following link:

https://docs.nvidia.com/jetson/jetpack/install-jetpack/index.html

After sdkmanager installed, you can run the GUI by the command:

sdkmanager

Then do the step1 – step4 to install Jetpack with all packages as you can see below



When done you will see INSTALLATIOM COMPLATED SUCCESSFULLY.

SLAM APPLICATION RUNNING MODES

There are two modes I used during the reasearch:

1) SLAM Application with ROS.

    a) First command launch ROS core:
    roslaunch /home/slamgpu/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/launch/ros_mono.launch
    b) Second command to provide the bag file:
    rosbag play /media/slamgpu/64GB/MH_01_easy_orig.bag

2) SLAM Application in standalone mode.

This mode uses mono_tum application with some paramters to load the dataset

Application:
./build/mono_tum

Working Direcotry:
/home/slamgpu/ORB_SLAM2_CUDA

Application parameters:
Vocabulary/ORBvoc.txt Examples/Monocular/TUM1.yaml
/media/slamgpu/64GB/rgbd_dataset_freiburg1_desk true

enviroment
DISPLAY=:1

In this mode I used with the tools as we must connect in root mode.

I tried also in ROS mode however ROS isn't supported in default to run in root mode so it can be new task for the next project.

Nvprof (Nvidia profiling tool)

Configuration nvprof to capture and profile Cuda applications

1) Move to root

sudo su

2) Configure envirment varibles permanent for terminal windows

Add to end of file ~/.bashrc (root bashrc) the following lines:

export PATH=/usr/local/cuda-10.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64:$LD_LIBRARY_PATH
source /opt/ros/melodic/setup.bash
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:/home/slamgpu/ORB_SLAM2_CUDA/Examples/ROS

Or you can run manually on each terminal window the command
source /home/tx2/.bashrc to prepare the environment

3) Verify you have nvprof installed by run command nvprof

In case the tool is missing use sudo apt get to install the tool
sudo apt-get install nvprof

4) Run the nvprof profiler command with ORB SLAM application

With Sudo (Root mode - Prefered)

sudo nvprof --profile-child-processes --export-profile /media/slamgpu/64GB/output_%p.prof roslaunch /home/slamgpu/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/launch/ros_mono.launch

Without Sudo

nvprof --profile-child-processes --export-profile /media/slamgpu/64GB/output_%p.prof roslaunch /home/slamgpu/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/launch/ros_mono.launch

5) Run the bag file (dataset) with the captured data on different termainl window
rosbag play /media/slamgpu/64GB/MH_01_easy_orig.bag

6) The output file will be saved in the sdcard (optional) as /media/slamgpu/64GB/output_%p.prof

For example /media/slamgpu/64GB/output_12212.prof
%p is for the process id

Analysys the output file can be done with nvprof or visual profiler:

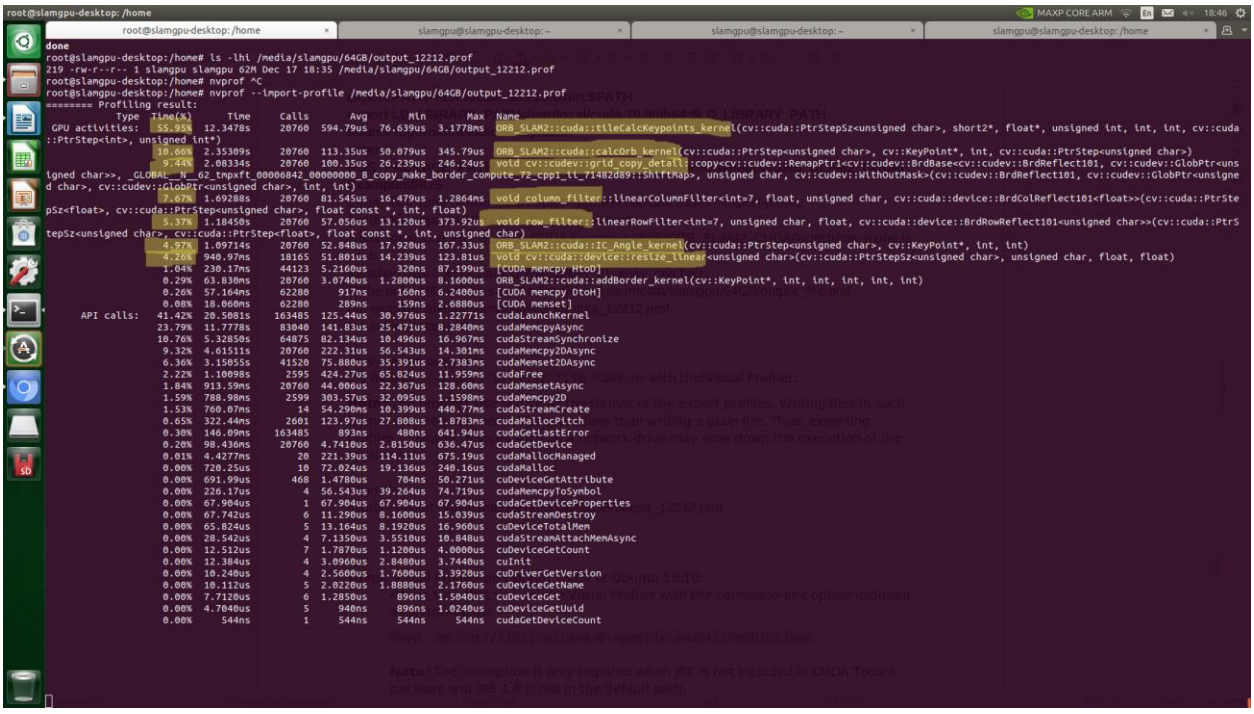
nvprof command for analyze:

nvprof --import-profile file_name

Note: The profilers use SQLite as the format of the export profiles. Writing files in such format may require more disk operations than writing a plain file. Thus, exporting profiles to slower devices such as a network drive may slow down the execution of the application.

Example:
nvprof --import-profile /media/slamgpu/64GB/output_12212.prof

For analysis:
Transfer the output .prof files to the host PC.
run /usr/local/cuda-10.0/bin/nvvp (nvidia visual profiler).
Use file->import to open the file (nvprof type).



To run Visual Profiler on Ubuntu 18.04 or Ubuntu 18.10:
Make sure that you invoke Visual Profiler with the command-line option included as shown below:

```
nvvp -vm /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
```

Note: The `-vm` option is only required when JRE is not included in CUDA Toolkit package and JRE 1.8 is not in the default path.On Ubuntu 18.10, if you get error "`no swt-pi-gtk in java.library.path`" when running Visual Profiler, then you need to install GTK2. Type the below command to install the required GTK2.

```
apt-get install libgtk2.0-0
```

Remote profiling is the process of collecting profile data from a remote system that is different than the host system at which that profile data will be viewed and analyzed. There are two ways to perform remote profiling. You can profile your remote application directly from `nsight` or the Visual Profiler. Or you can use `nvprof` to collect the profile data on the remote system (target) and then use `nvvp` on the host system to view and analyze the data.

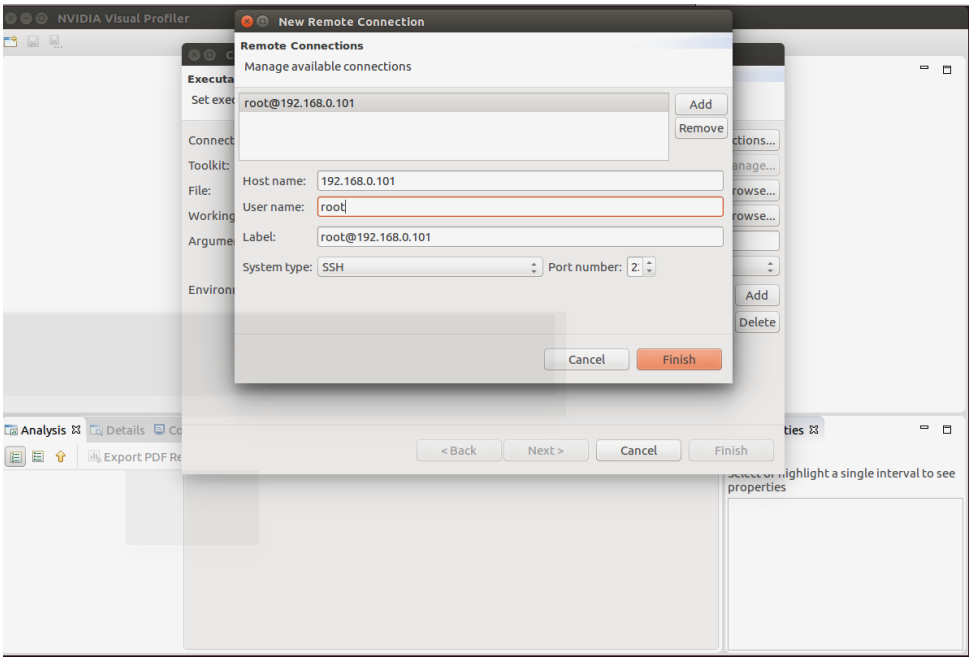
https://docs.nvidia.com/cuda/profiler-users-guide/index.html

Visual profiler

Make sure you have visual profiler install on the target by running the following command

```
nvvp -vm /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java

Configure the connection
```



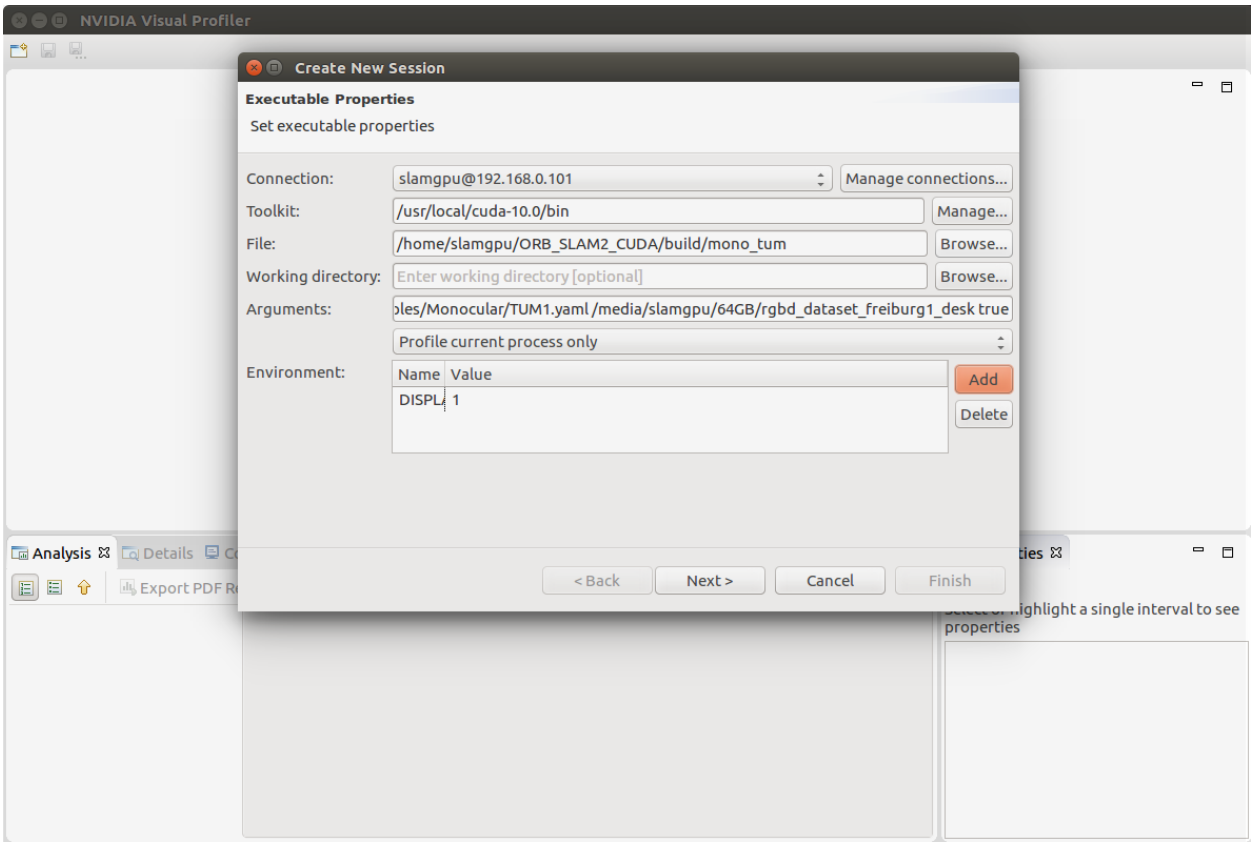
Configure application details:

Application:
./build/mono_tum

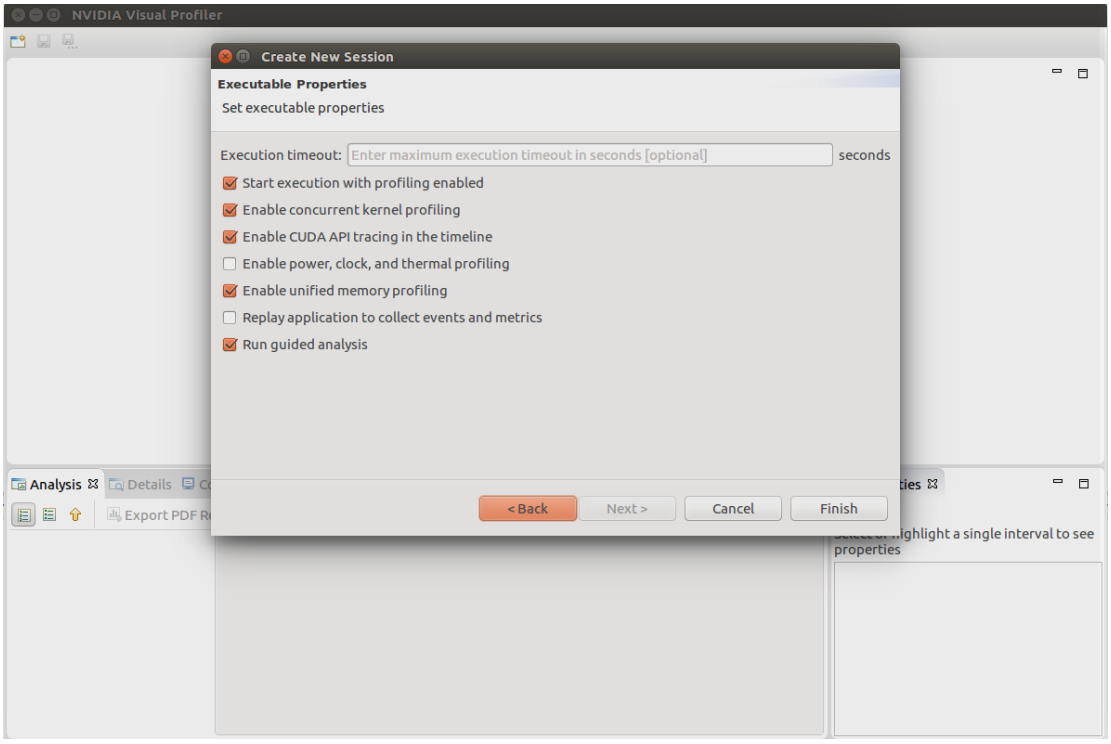
Working Direcotry:
/home/slamgpu/ORB_SLAM2_CUDA

Application parameters:
Vocabulary/ORBvoc.txt Examples/Monocular/TUM1.yaml /media/slamgpu/64GB/rgbd_dataset_freiburg1_desk true

enviroment
DISPLAY=:1



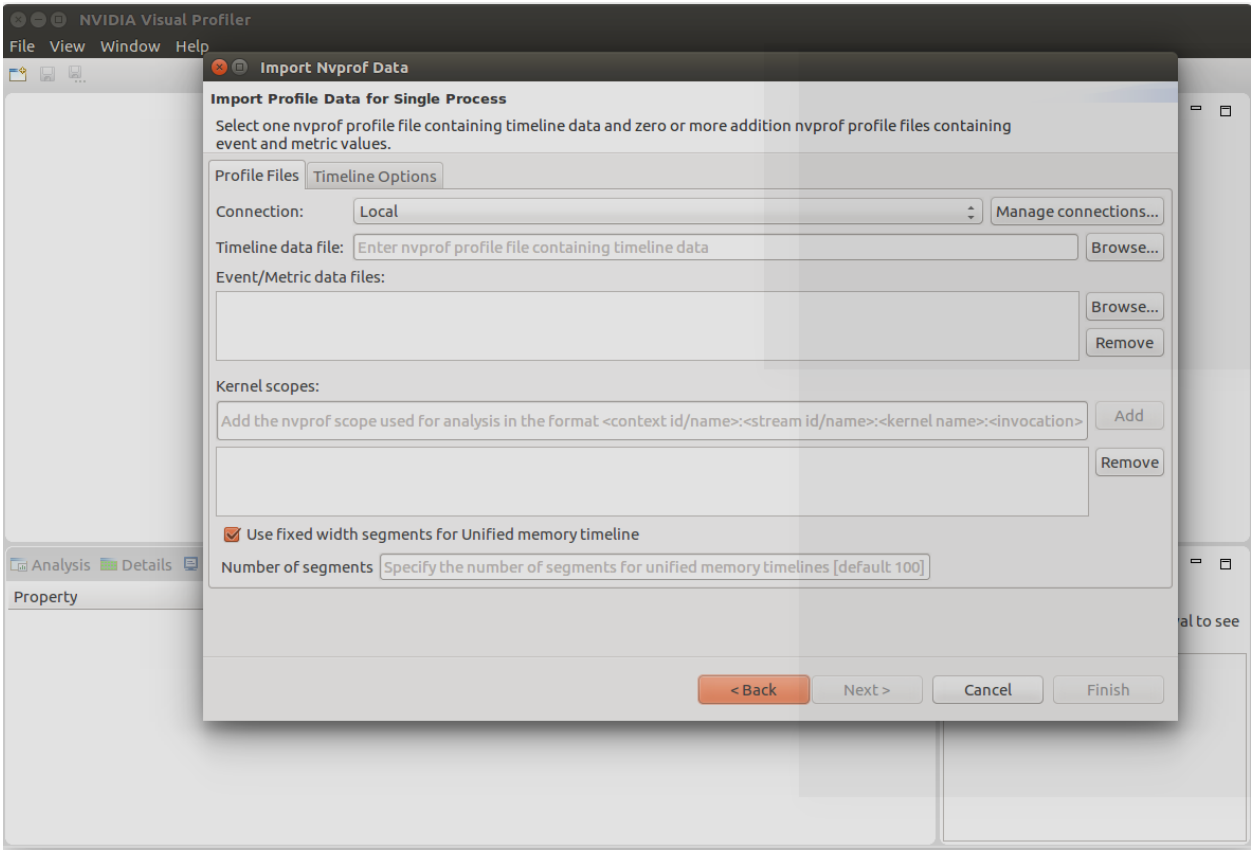
Click Next for the next window



Click finish.

You can start the application on the host side by also with the command /usr/local/cuda-10.0/bin/nvvp

This command will allow you to load saved nvprof file.

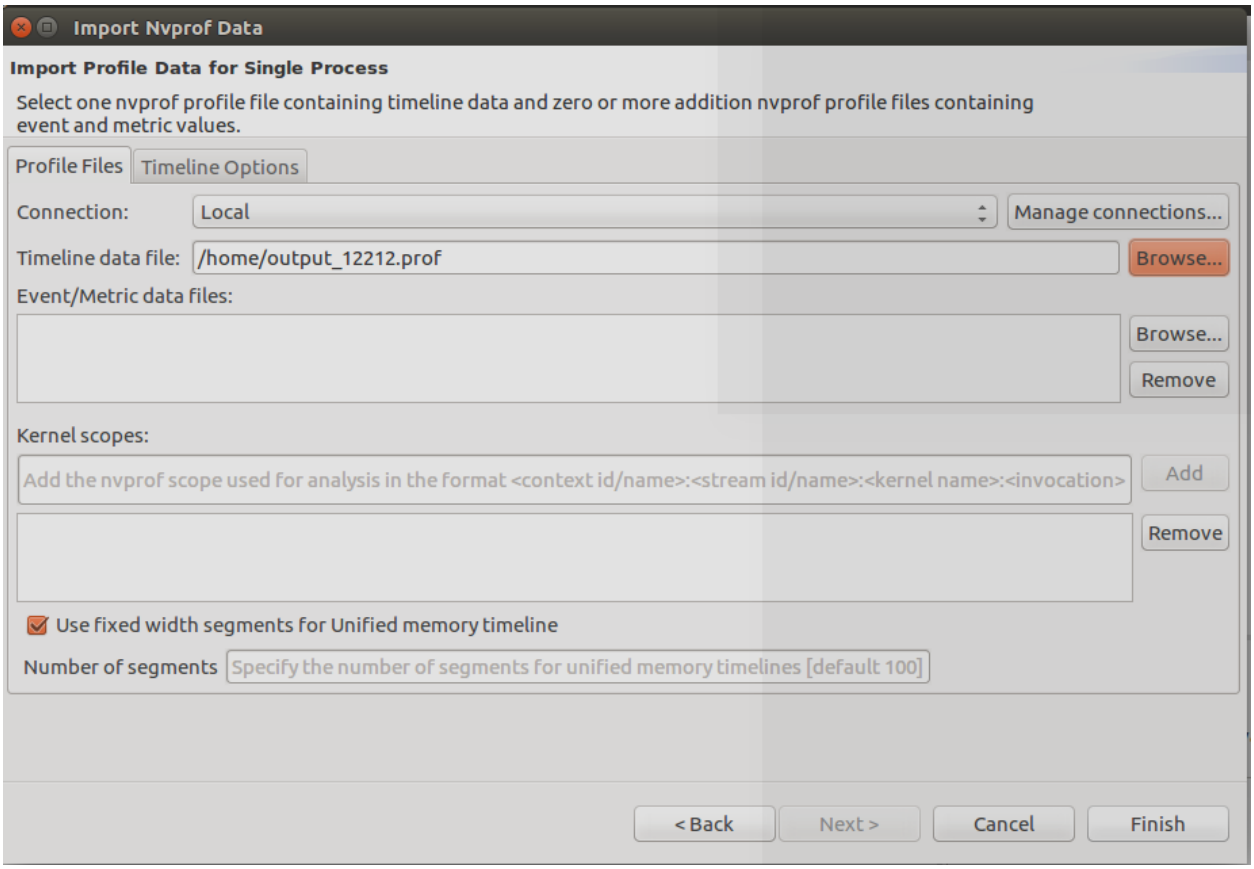


You can copy files from device by SCP

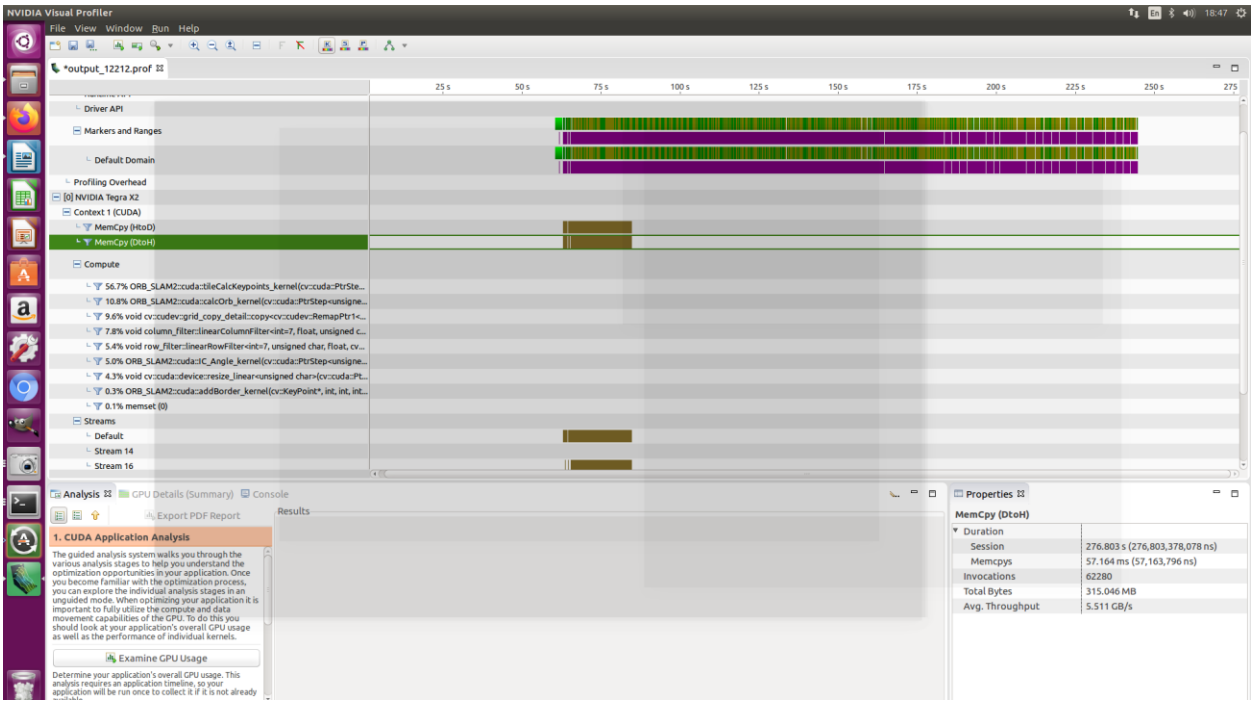
Example:

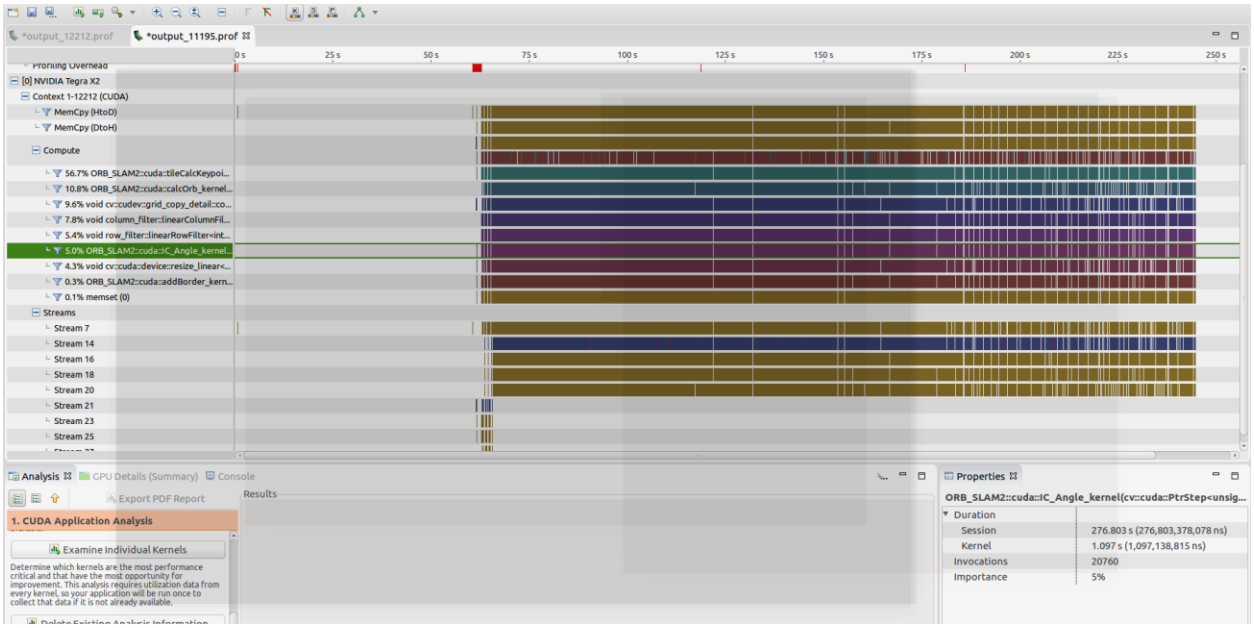
sudo scp root@192.168.0.101:/media/slamgpu/64GB/output_12212.prof /home/output_12212.pro

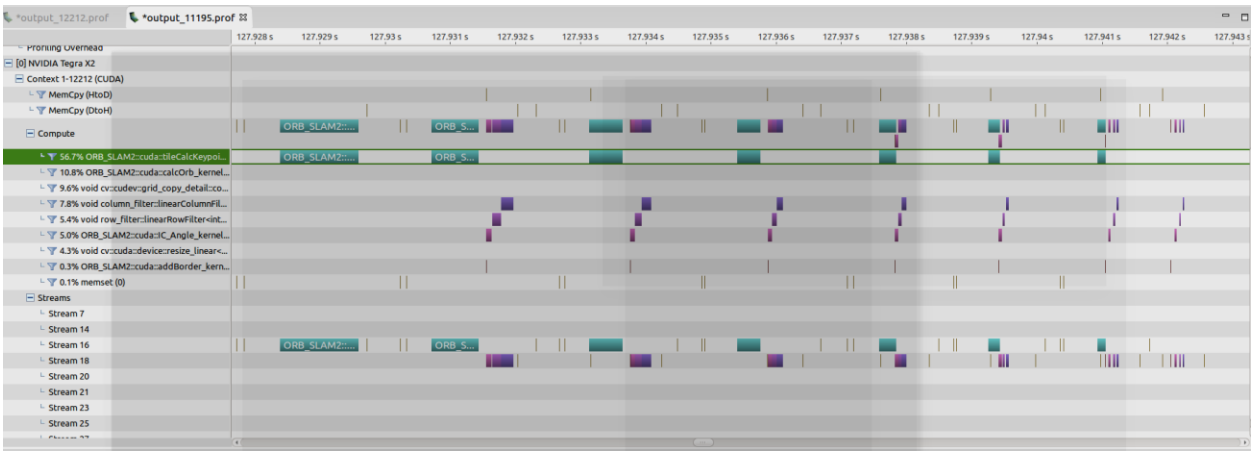
Fill the path to the file and then click Finish.



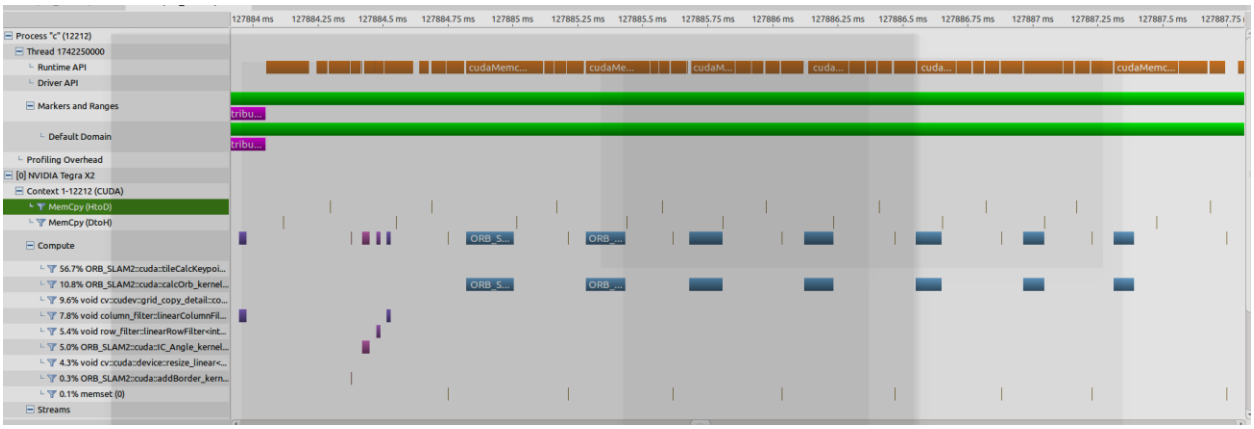
The tool will import and will show the timelines.


NVIDIA Visual Profiler
File View Window Run Help
*output_12212.prof
Driver API
Markers and Ranges
Default Domain
Profiling Overhead
[0] NVIDIA Tegra X2
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
Streams
Default
Stream 14
Stream 16
Analysis
GPU Details (Summary)
Console
Export PDF Report
Results
1. CUDA Application Analysis
Examine GPU Usage
Properties
MemCpy (DtoH)
Duration
Session 276.803 s (276,803,378,078 ns)
Memcpys 57.164 ms (57,163,796 ns)
Invocations 62280
Total Bytes 315.046 MB
Avg. Throughput 5.511 GB/s


*output_12212.prof
*output_11195.prof
[0] NVIDIA Tegra X2
Context 1-12212 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
Streams
Stream 7
Stream 14
Stream 16
Stream 18
Stream 20
Stream 21
Stream 23
Stream 25
Analysis
GPU Details (Summary)
Console
Export PDF Report
Results
1. CUDA Application Analysis
Examine Individual Kernels
Properties
Duration
Session 276.803 s (276,803,378,078 ns)
Kernel 1.097 s (1,097,138,815 ns)
Invocations 20760
Importance 5%



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



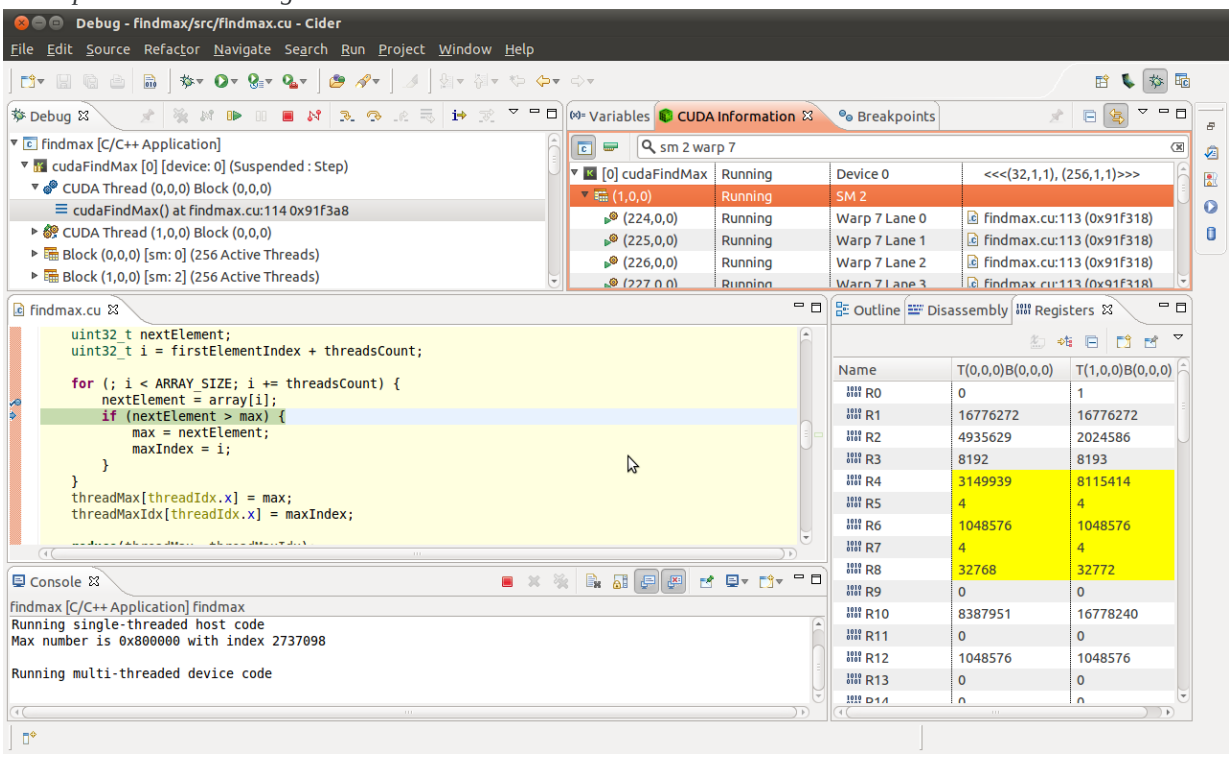
Nsight eclipse edition

NVIDIA® Nsight™ Eclipse Edition is a unified CPU plus GPU integrated development environment (IDE) for developing CUDA® applications on Linux and Mac OS X for the x86, POWER and ARM platforms.

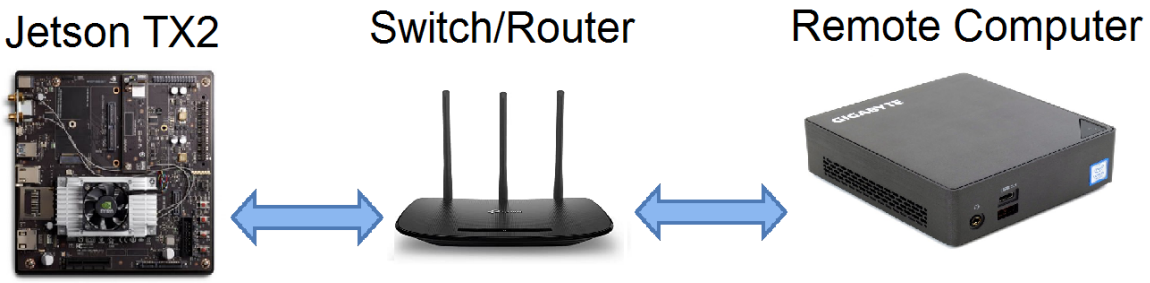
You can see information about the tool in the following path:

https://docs.nvidia.com/cuda/nsight-eclipse-edition-getting-started-guide/index.html

For manully tool installtion run the command:
sudo apt install nvidia-nsight



Network Configuration for Remote Profiling



I used TPLink Switch\Router.

1) Check device IP (Jetson) by command: ifconfig

```
root@slamgpu-desktop:~#
root@slamgpu-desktop:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.101  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::ae0a:61fa:1c0b:23da  prefixlen 64  scopeid 0x20<link>
        ether 00:04:4b:dd:8c:62  txqueuelen 1000  (Ethernet)
        RX packets 6089  bytes 475646 (475.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11669  bytes 975562 (975.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 41

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 1729  bytes 128065 (128.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
```

Internal interface for local connection eth0 with IP address 192.168.0.101

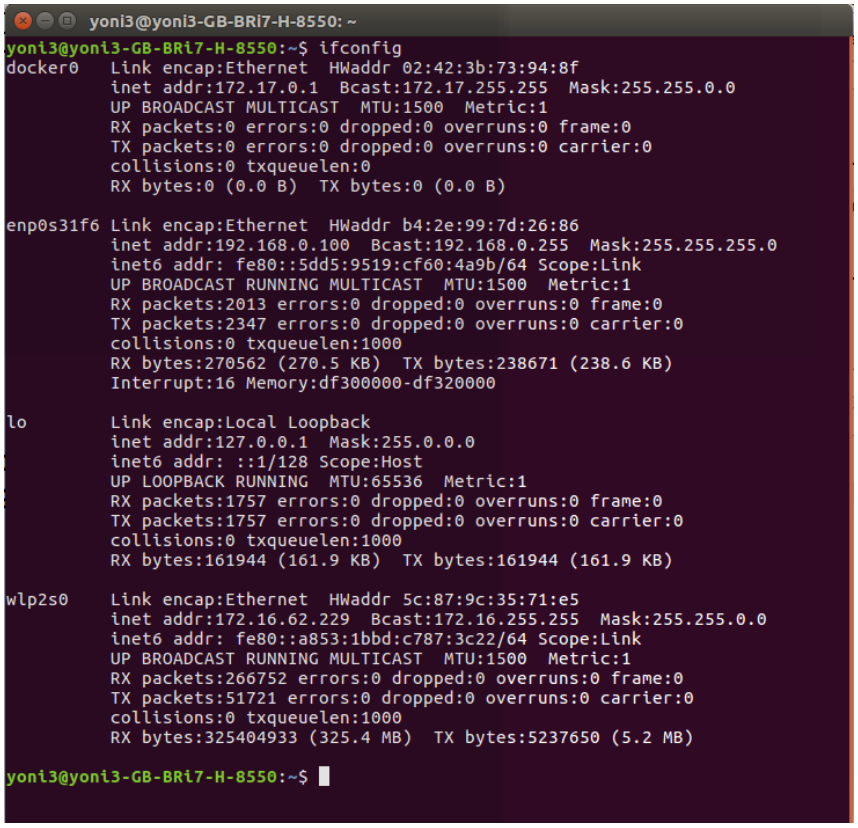
External interface for Wifi:

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 172.16.205.6 netmask 255.255.0.0 broadcast 172.16.255.255
inet6 fe80::d7d1:43ce:6780:b374 prefixlen 64 scopeid 0x20<link>
ether 00:04:4b:dd:8c:60 txqueuelen 1000 (Ethernet)
RX packets 51176 bytes 32003734 (32.0 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 14495 bytes 1672565 (1.6 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

2) Host side Configuration (Remote computer)

Make sure the internal interface netmask is the same as in device side.

In the current configuration internal interface IP address is 192.168.0.100



The default GW is 192.168.0.1

In our setup all the three on the same subnet mask 255.255.255.0

3) Device side configuration

The device is configured with the same network configuration as the host and configured with unique internal IP 192.168.0.101

CLONE JETSON TX2 INTO NEW SD CARD

- connect jetson via USB to host PC with jetpack (same version that used for installaion)
- download the script described in this post:
  https://devtalk.nvidia.com/default/topic/1000105/jetson-tx2/tx2-cloning/post/5111893/#5111893
  to flash.sh file to the host PC.
- run with the following command on the host PC:

```
sudo ./flash.sh -r -k APP -G my_backup.img jetson-tx2 mmcblk0p1
```

- (not sure about this. Try to skip first) squash the new image on host with:

```
sudo mksparse -v --fillpattern=0 system.img.raw system.img
```

- rename the resulting file to be system.img and place it under the jetpack root filesystem that is used for the jetson installation. It should be under **bootloader/** directory. Note that the old system.img file should be erased/renamed/moved.
- connect a new jetson to the host pc. (you might need to force it into emergency mode if not identified as emmc device in the host PC). run the following command to flash the **new** device:

```
sudo ./flash.sh -r -k APP jetson-tx2 mmcblk0p1
```

GPU PCIe Configuration Check

To can check which GPU you have on host by the Linux util commands:

http://manpages.ubuntu.com/manpages/xenial/man8/lspci.8.html

Detailed Device Information

lspci -v

Dump PCI Info in Different Format

lspci -m

Link for tools

https://developer.nvidia.com/tools-overview

Nsight compute
https://docs.nvidia.com/nsight-compute/NsightCompute/index.html

https://developer.nvidia.com/nsight-compute-2019_5

https://docs.nvidia.com/nsight-graphics/UserGuide/index.html#getting_started_how_to_launch_and_connect

- /usr/local/cuda-10.0/NsightCompute-1.0/nv-nsight-cu
- nv-nsight-gfx-for-l4t
- nsight-sys

How to run device SLAM application via SSH connection

Note : This application mode is ROS mode.

Open SSH connection to Device from the host

Check the DISPLAY id

```
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$
slamgpu@slamgpu-desktop:~$ who
slamgpu  :1           2019-12-24 09:24 (:1)
slamgpu  pts/2        2019-12-24 09:33 (192.168.0.100)
slamgpu@slamgpu-desktop:~$
```

In our case :1 is the Display id to run on the device

Change the the global varible to open the display in the device by

export DISPLAY=:1

In case of error as below :

```
ORB Extractor Parameters:
- Number of Features: 1000
- Scale Levels: 8
- Scale Factor: 1.2
- Initial Fast Threshold: 20
- Minimum Fast Threshold: 7

-------
Start processing sequence ...
Images in the sequence: 613

error: XDG_RUNTIME_DIR not set in the environment.
No protocol specified
terminate called after throwing an instance of 'std::runtime_error'
  what():  Pangolin X11: Failed to open X display
Aborted
root@slamgpu-desktop:/home/slamgpu/ORB_SLAM2_CUDA#
```

export XDG_RUNTIME_DIR=/run/user/1000

run the ROS application in one SSH terminal

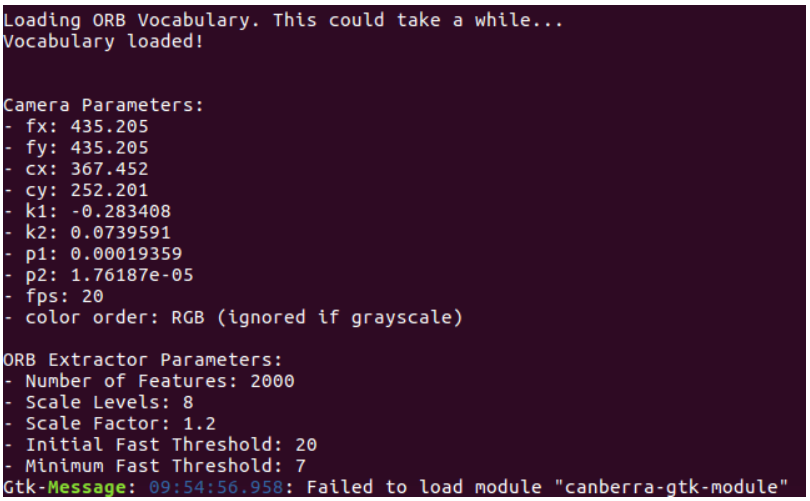
roslaunch
/home/slamgpu/ORB_SLAM2_CUDA/Examples/ROS/ORB_SLAM2_CUDA/launch/ros_mono.launch

Then run in second SSH terminal (new terminal) the bag script

rosbag play /media/slamgpu/64GB/MH_01_easy_orig.bag

Solutions for issues during SLAM ROS application

1) Error during running ROS:
Failed to load module "canberra-gtk-module"

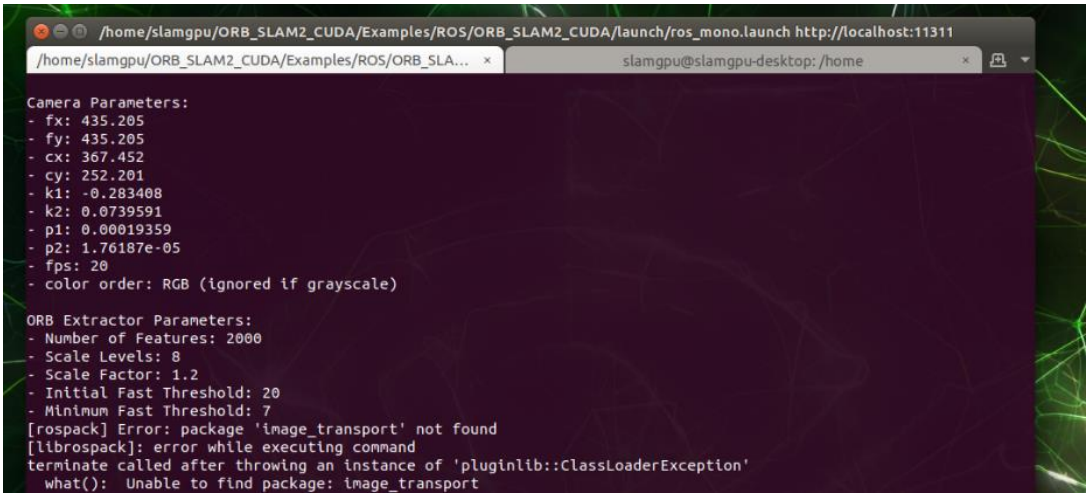


Fix the issue by:

sudo apt install libcanberra-gtk-module libcanberra-gtk3-module

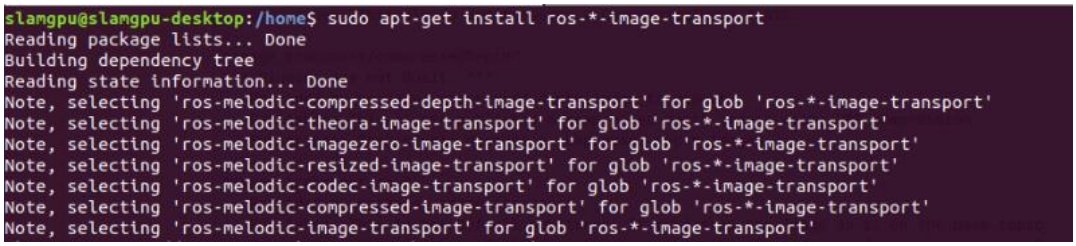
2) Issue during running ROS application:

"unable to find package: image transport"



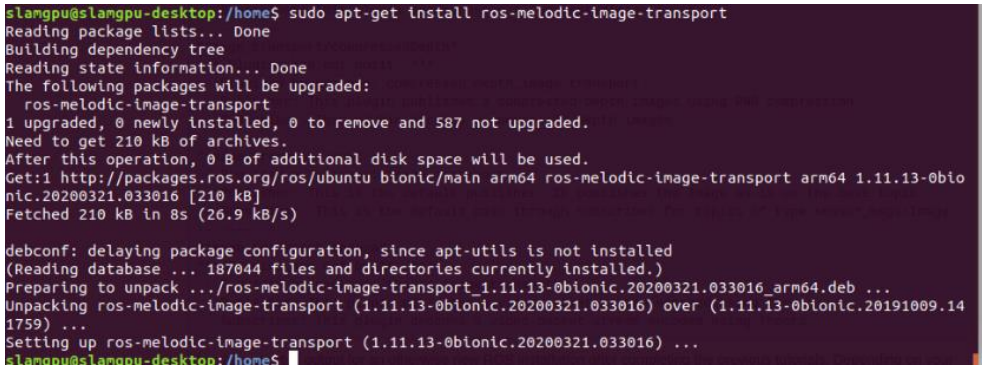
Solution:

Search for the relevant package :



By command:

sudo apt-get install ros--image-transport*



In our case ROS version is melodic so I used the command:

sudo apt-get install ros-melodic-image-transport

Compilation Cuda Application with Debug Symbols

`Compile Cuda example on Jetson TX2 with debug symbols:`

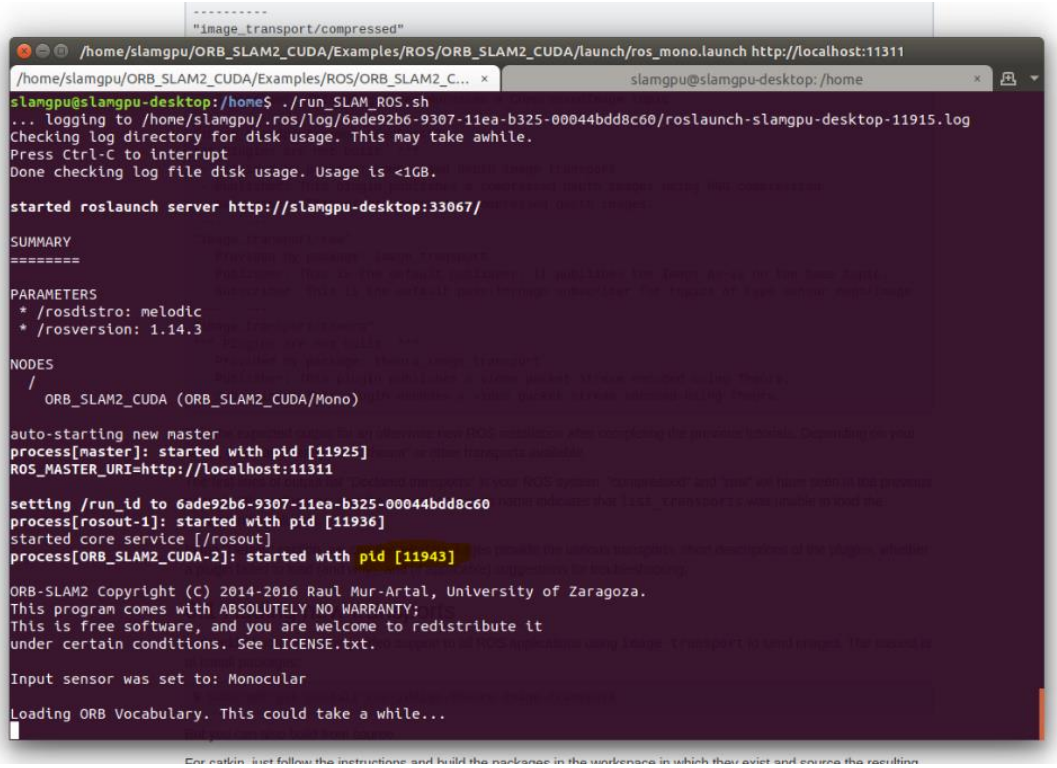
sudo sh -c "dbg=1 make"

Example
slamgpu@slamgpu-desktop:/usr/local/cuda-10.0/samples/0_Simple/vectorAdd$ sudo sh -c "dbg=1 make"/usr/local/cuda-10.0/bin/nvcc -ccbin g++ -I../../common/inc -m64 -g -G   -gencode arch=compute_30,code=sm_30 -gencode arch=compute_32,code=sm_32 -gencode arch=compute_53,code=sm_53 -gencode arch=compute_61,code=sm_61 -gencode arch=compute_62,code=sm_62 -gencode arch=compute_70,code=sm_70 -gencode arch=compute_72,code=sm_72 -gencode arch=compute_75,code=sm_75 -gencode arch=compute_75,code=compute_75 -o vectorAdd.o -c vectorAdd.cu
/usr/local/cuda-10.0/bin/nvcc -ccbin g++   -m64 -g -G   -gencode arch=compute_30,code=sm_30 -gencode arch=compute_32,code=sm_32 -gencode arch=compute_53,code=sm_53 -gencode arch=compute_61,code=sm_61 -gencode arch=compute_62,code=sm_62 -gencode arch=compute_70,code=sm_70 -gencode arch=compute_72,code=sm_72 -gencode arch=compute_75,code=sm_75 -gencode arch=compute_75,code=compute_75 -o vectorAdd vectorAdd.o
mkdir -p ../../bin/aarch64/linux/debug
cp vectorAdd ../../bin/aarch64/linux/debug

Nsight System

SLAM application in ROS mode

In the following tutorial I run the SLAM application in ROS mode and I connect the profiling tool (Nsight System) to process id (PID).



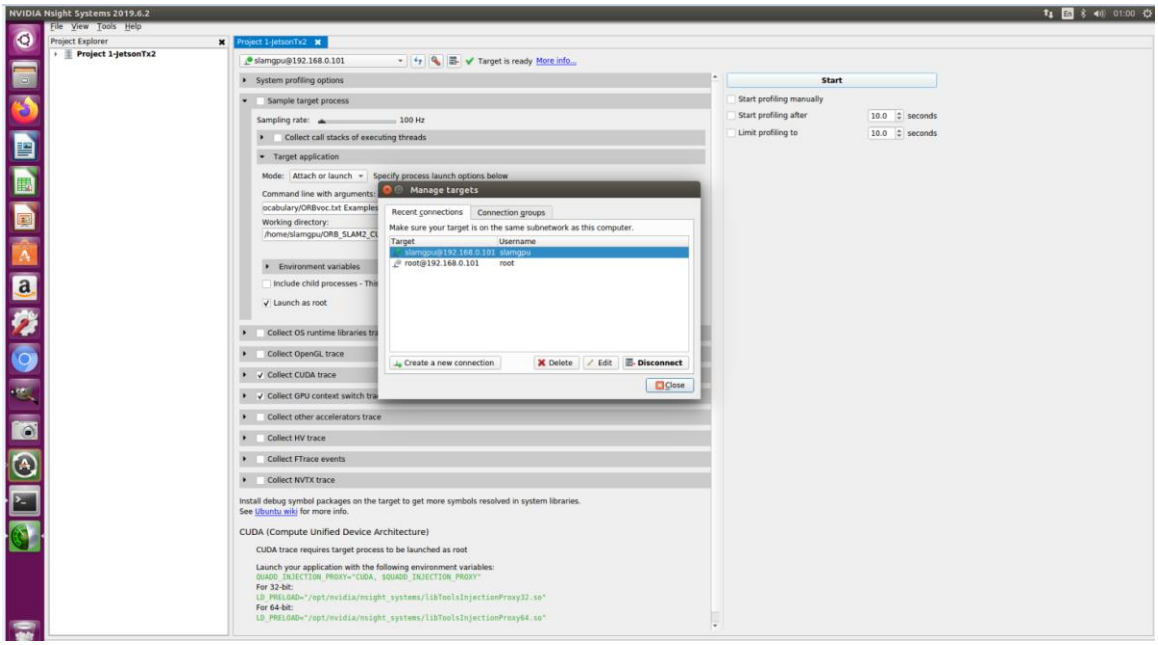
1) Open the tool on the host by the command :
nsight-sys

2) Open new project

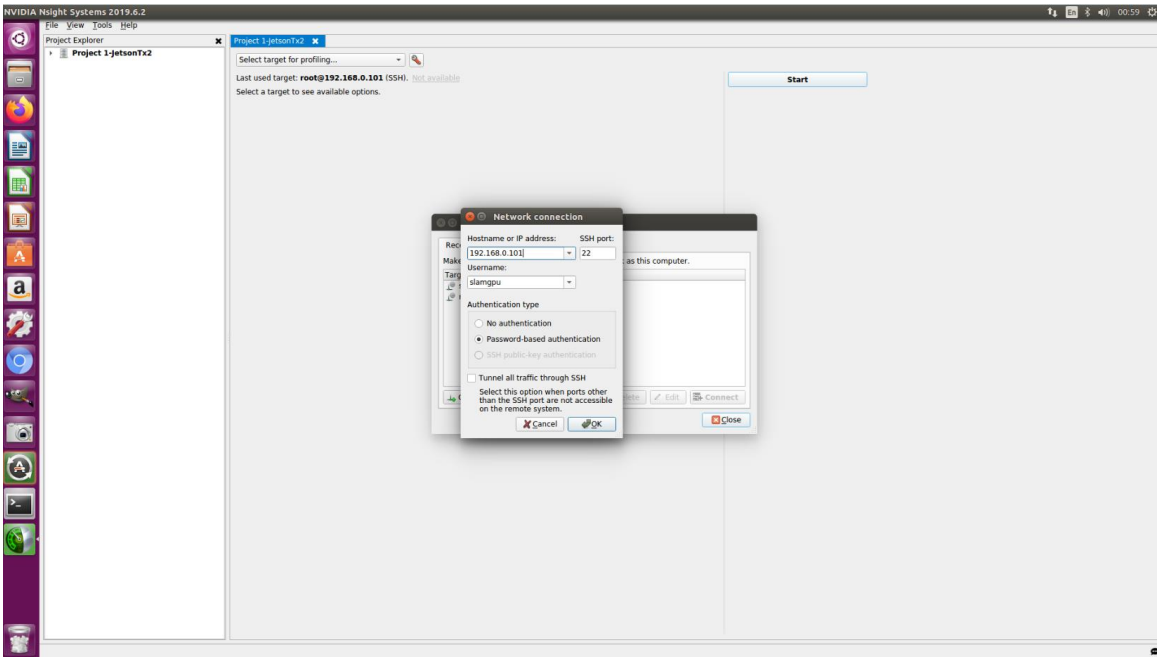
3) Configure SSH connection to device

   Create new connection regular user or root.

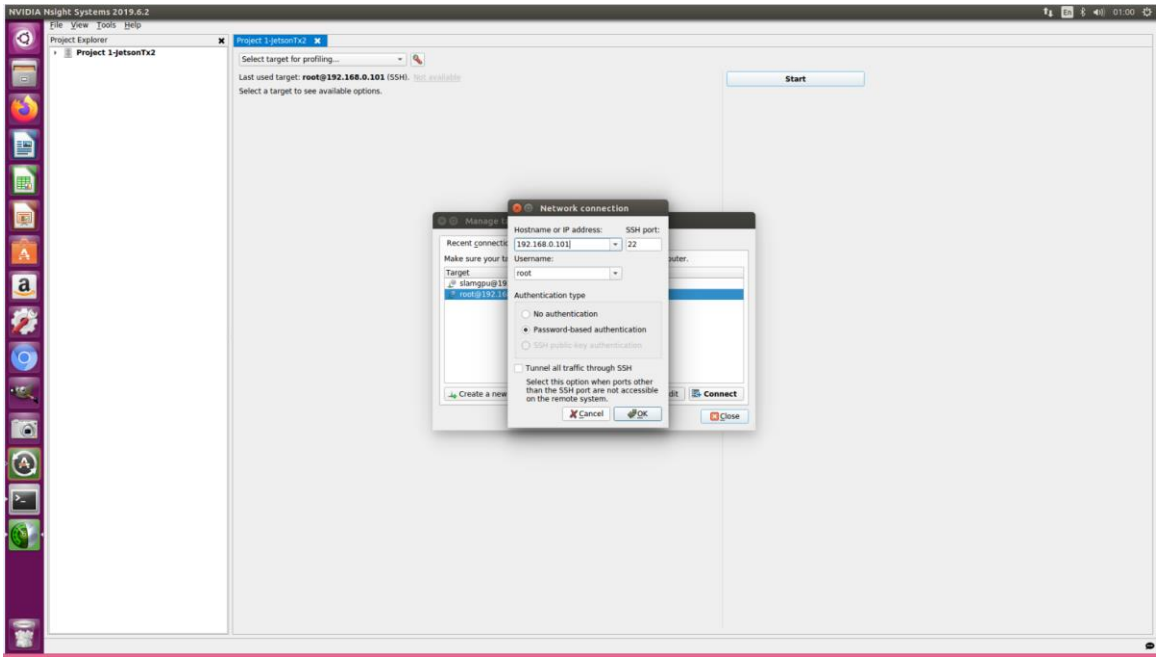
   Fill IP address and user name for login to the device.



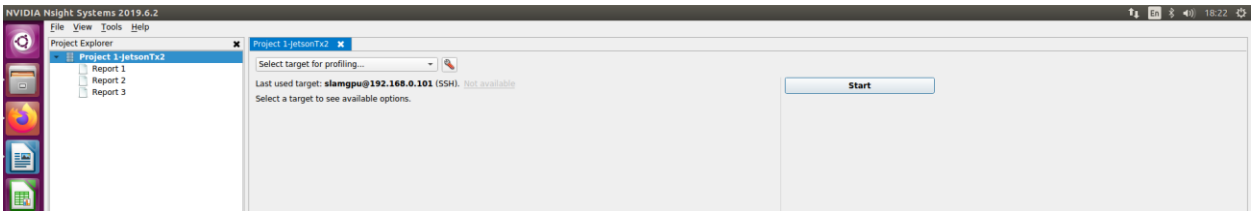
User: slamgpu



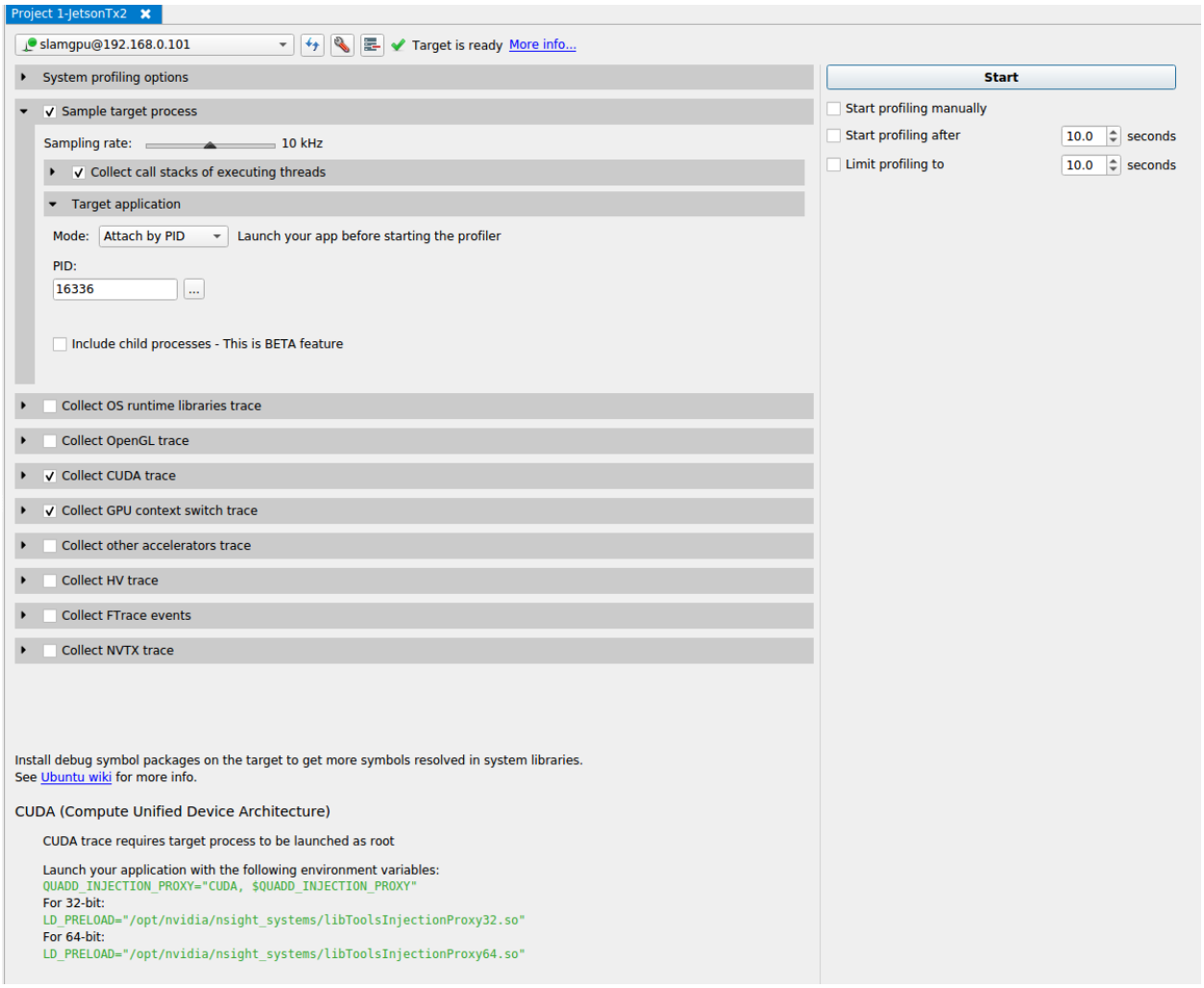
User: Root



4) Select the remote target (User or Root) and click on Start

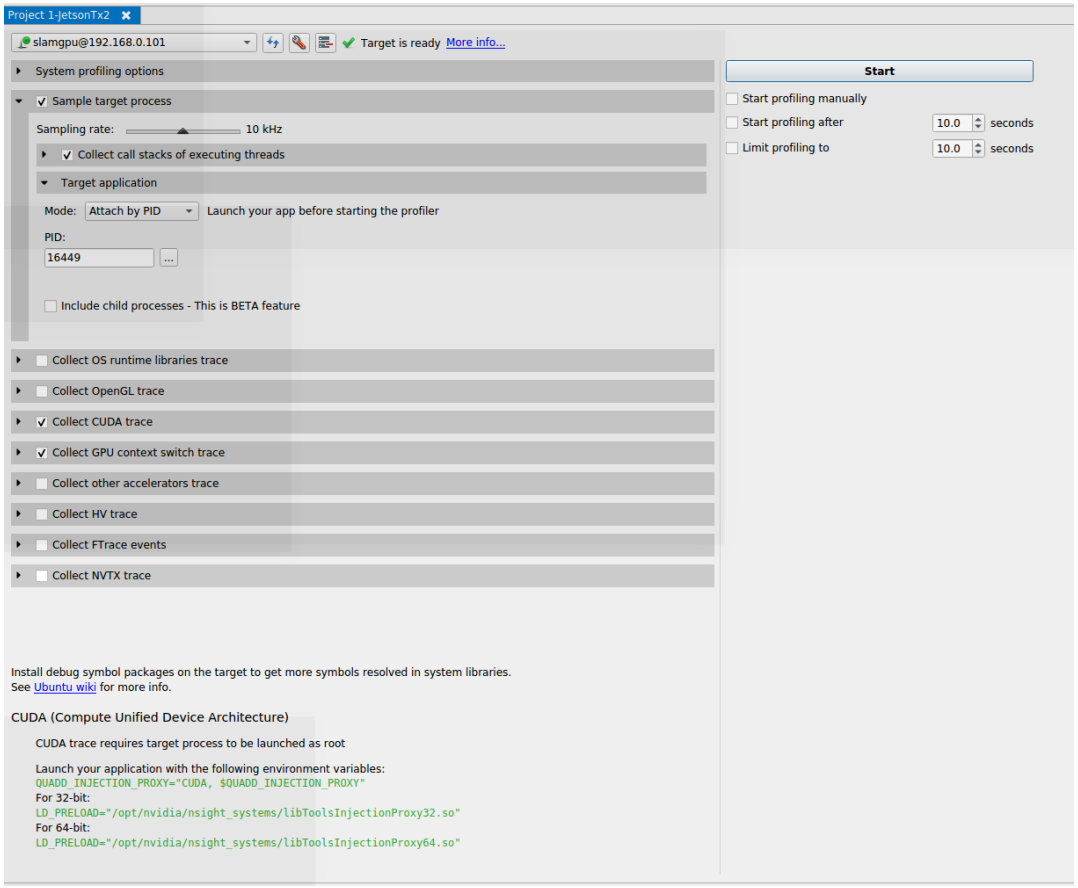


5) Configure the profiling options

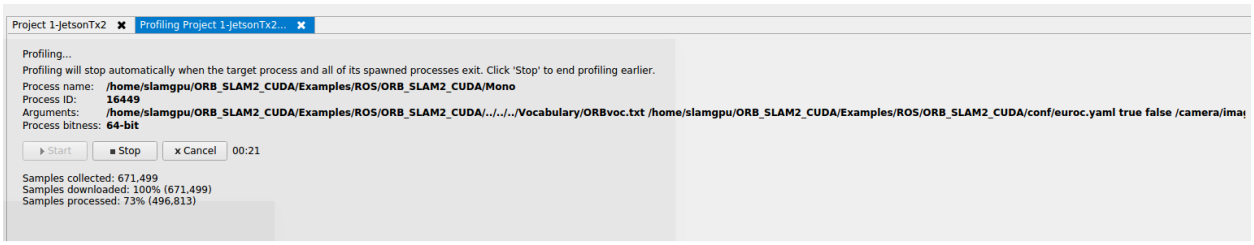


6) Run the application in the device side and configure the process id to the nsight system

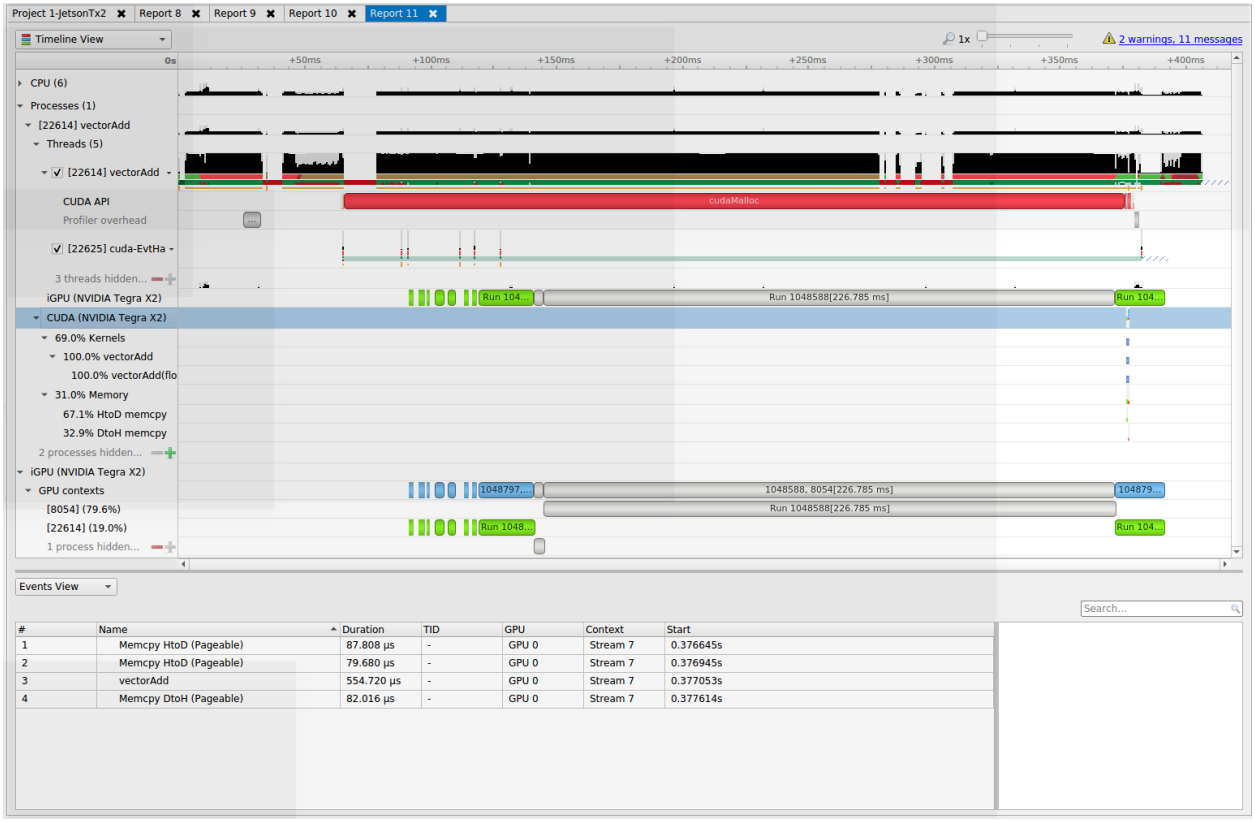
In the below image I used PID 16449



Then click on start



When finish you could and analyze the results.



For capture Cuda Information better use :

CUDA trace requires target process to be launched as root

> Launch your application with the following environment variables:
> `QUADD_INJECTION_PROXY="CUDA, $QUADD_INJECTION_PROXY"`
> For 32-bit:
> `LD_PRELOAD="/opt/nvidia/nsight_systems/libToolsInjectionProxy32.so"`
> For 64-bit:
> `LD_PRELOAD="/opt/nvidia/nsight_systems/libToolsInjectionProxy64.so"`

By adding the defines to bashrc you can reload by the command:
source ~/.bashrc

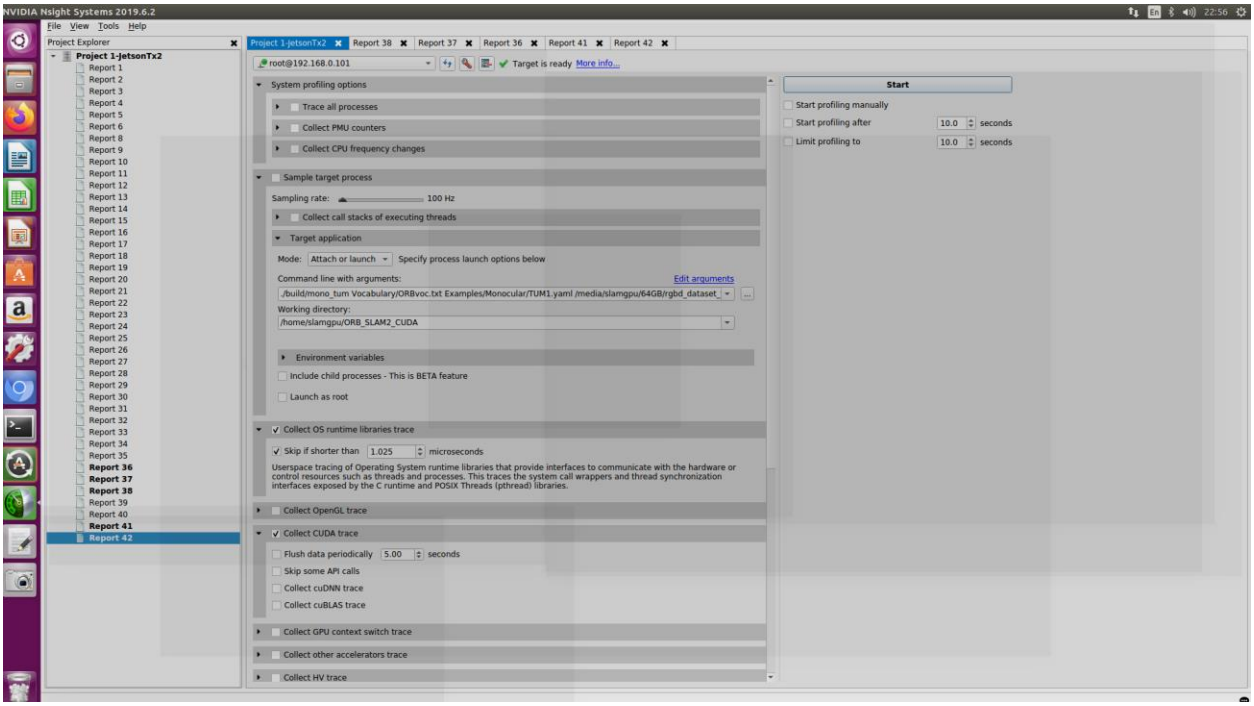
SLAM application in Standalone Mode

This mode is preferred in our research as the SLAM application is run in root mode.

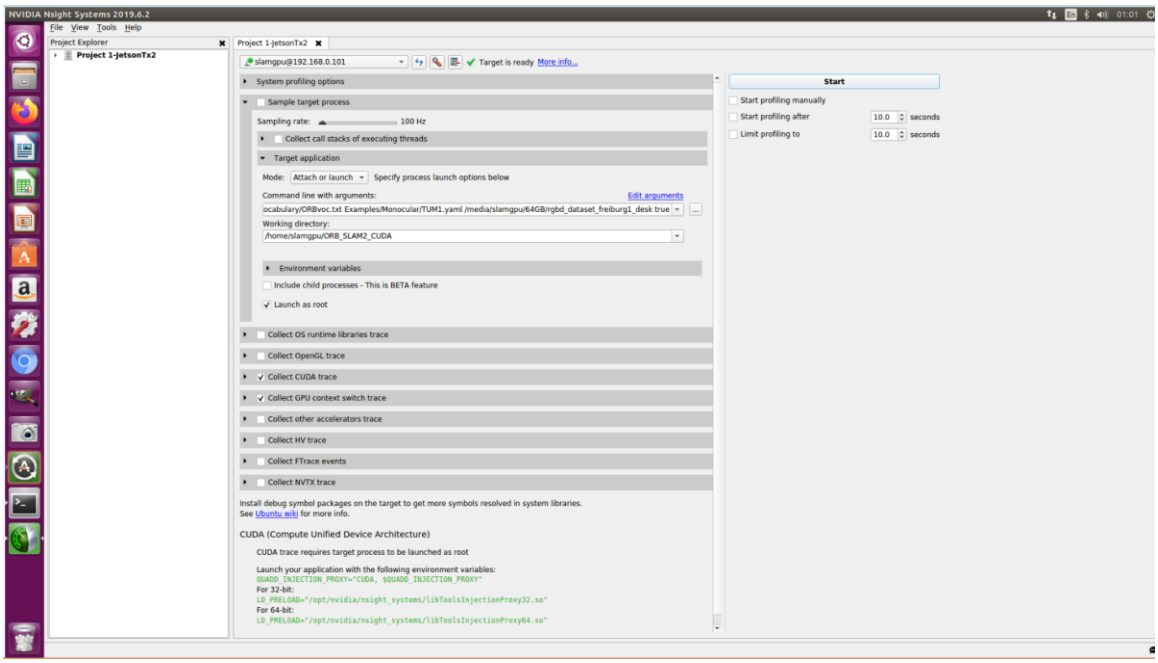
You can launch the tool by command:

nsight-sys

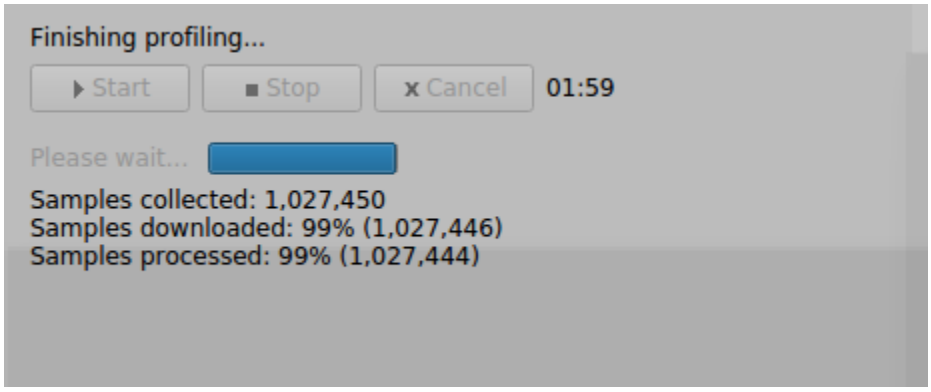
Configure connection and application parameters with root permissions:



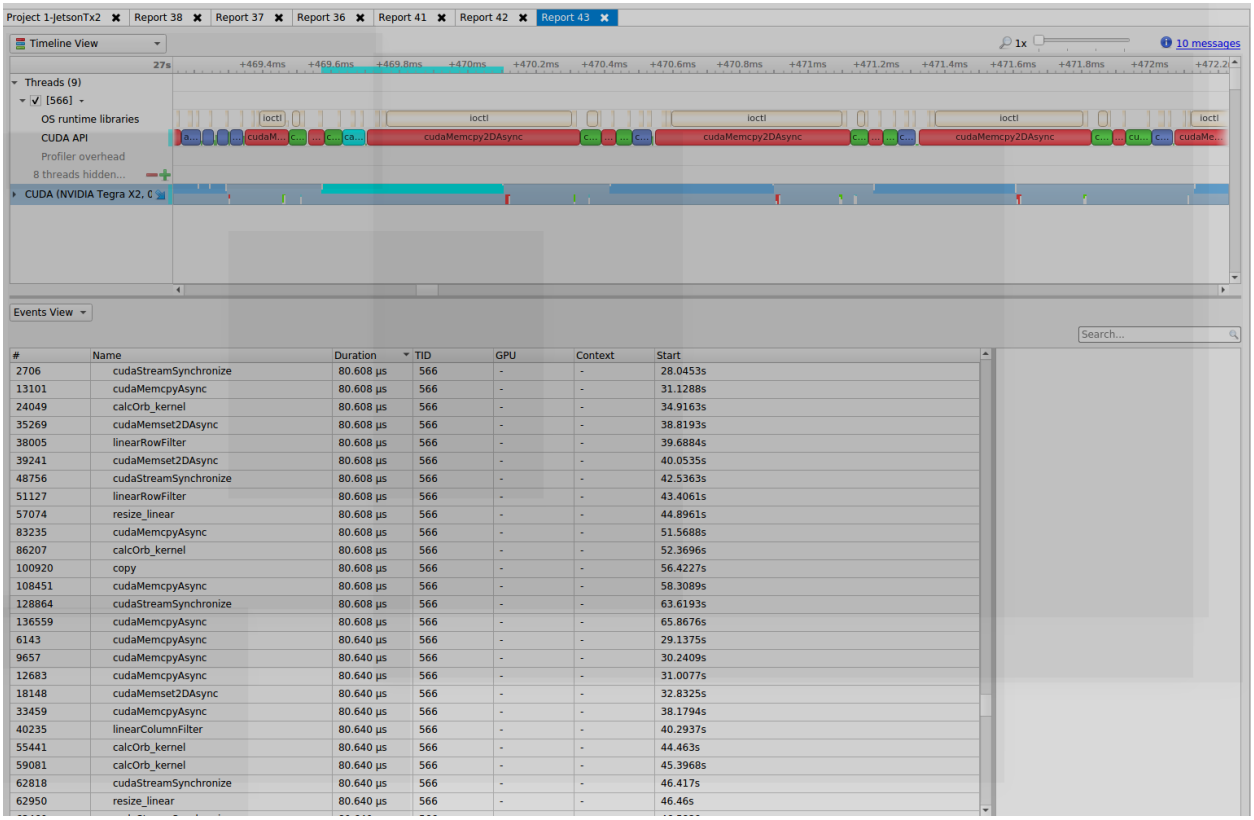
Preferred to click on Launch as root to collect more information:



Click start for launch the session then waiting for parsing

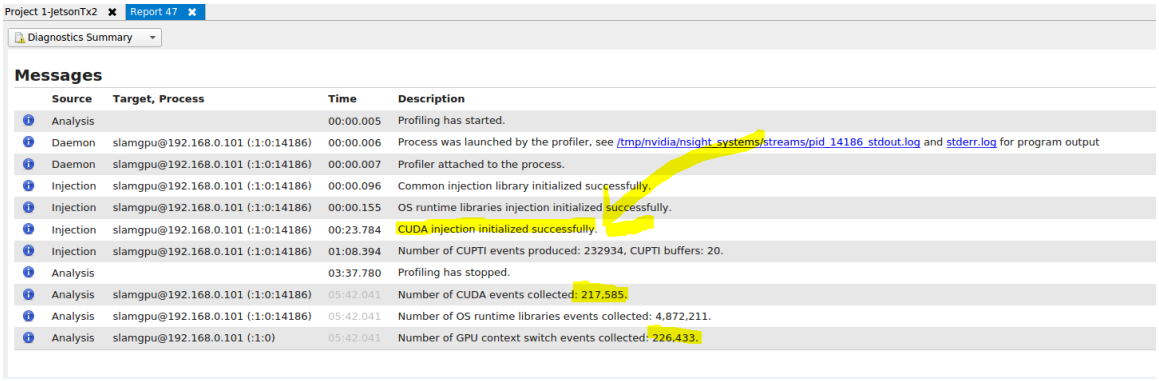


When processing is finished you can start analyze the results:

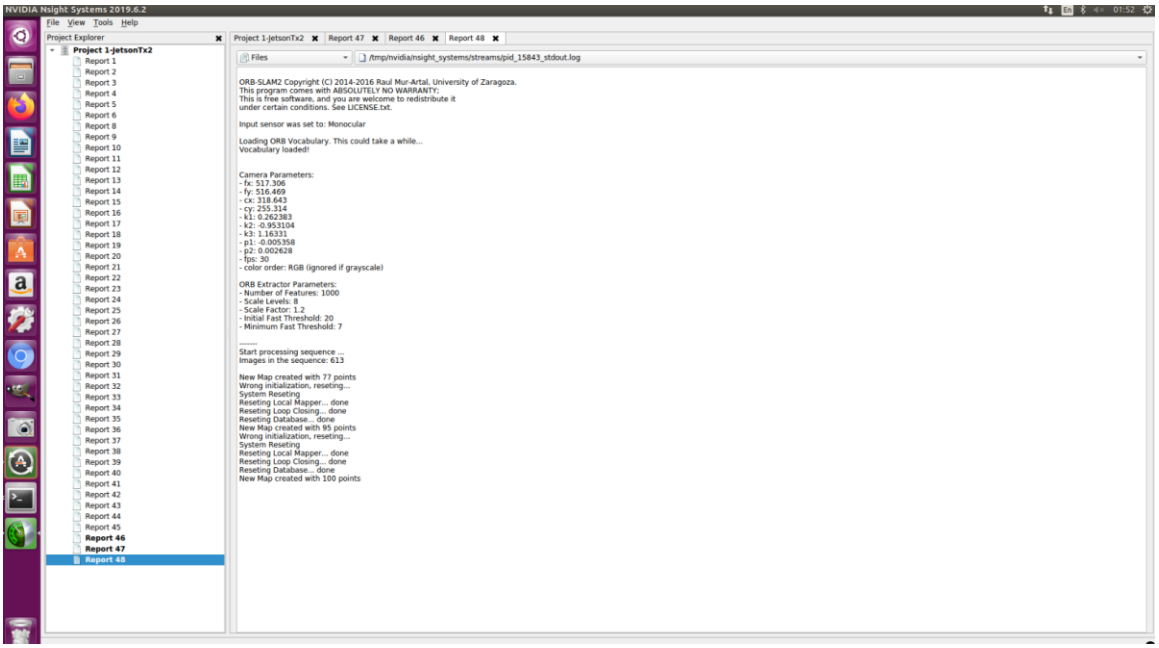


#	Name	Duration	TID	GPU	Context	Start
2706	cudaStreamSynchronize	80.608 µs	566	-	-	28.0453s
13101	cudaMemcpyAsync	80.608 µs	566	-	-	31.1288s
24049	calcOrb_kernel	80.608 µs	566	-	-	34.9163s
35269	cudaMemset2DAsync	80.608 µs	566	-	-	38.8193s
38005	linearRowFilter	80.608 µs	566	-	-	39.6884s
39241	cudaMemset2DAsync	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

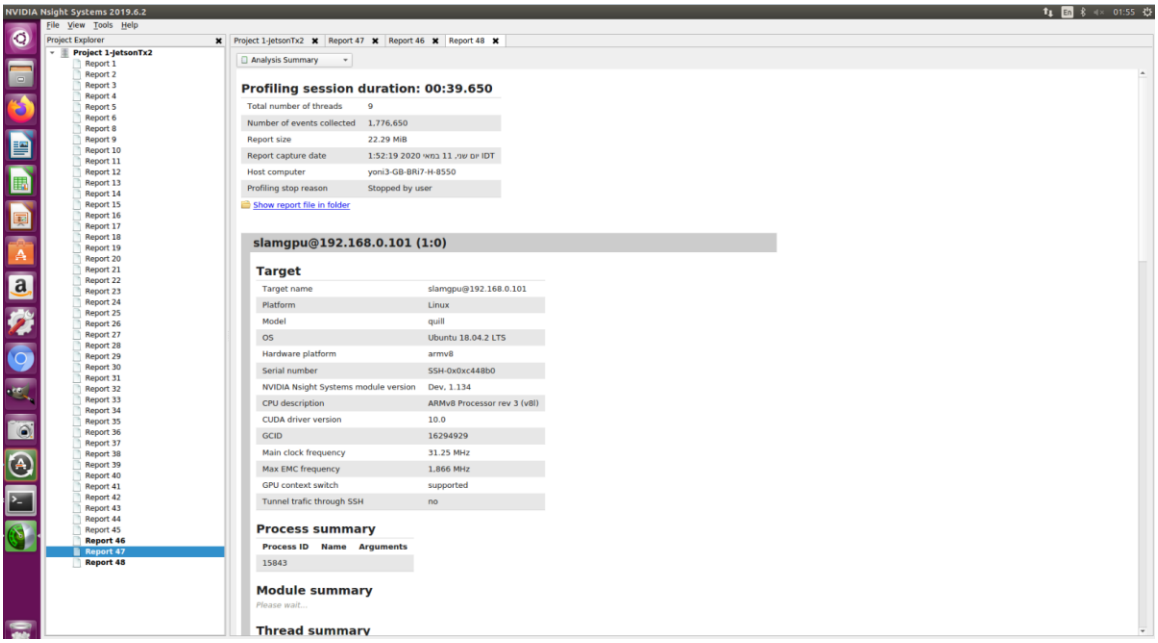
You can see the Cuda profiling done correctly by



You can see application output parameters:



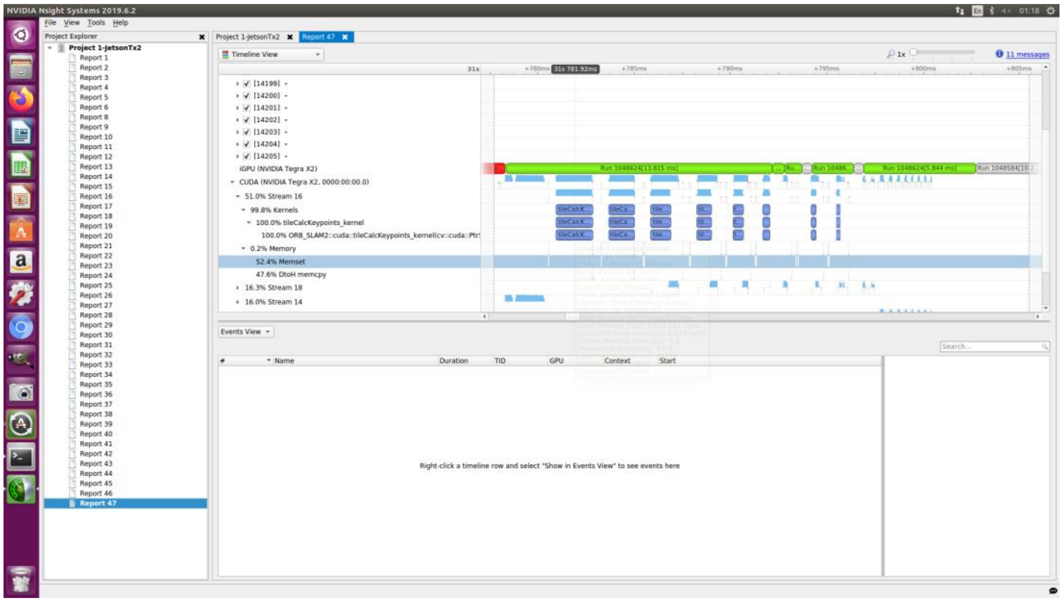
You can see high level information about the profiling session:

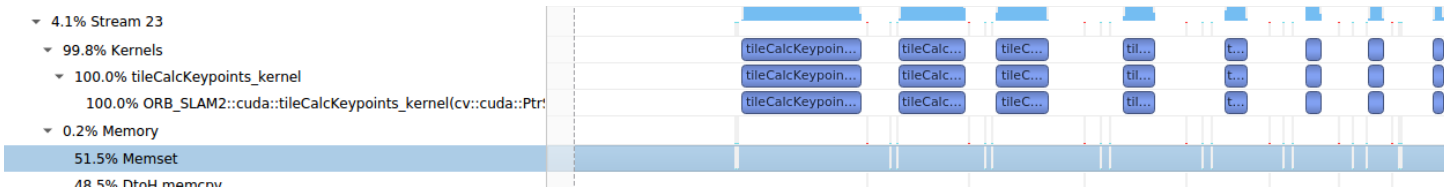


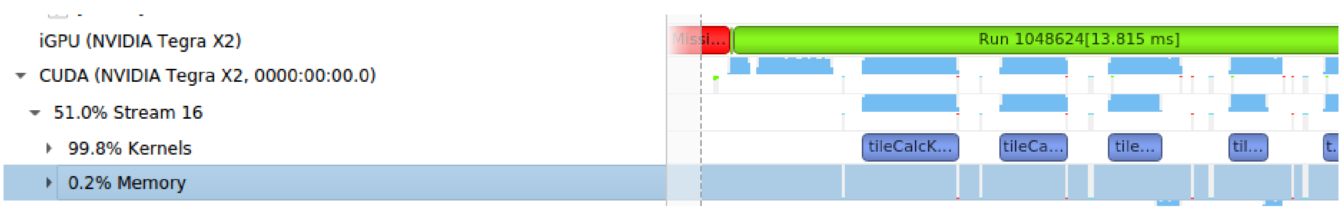
Analyze the SLAM CUDA application with Nsight system

Cuda Streams :

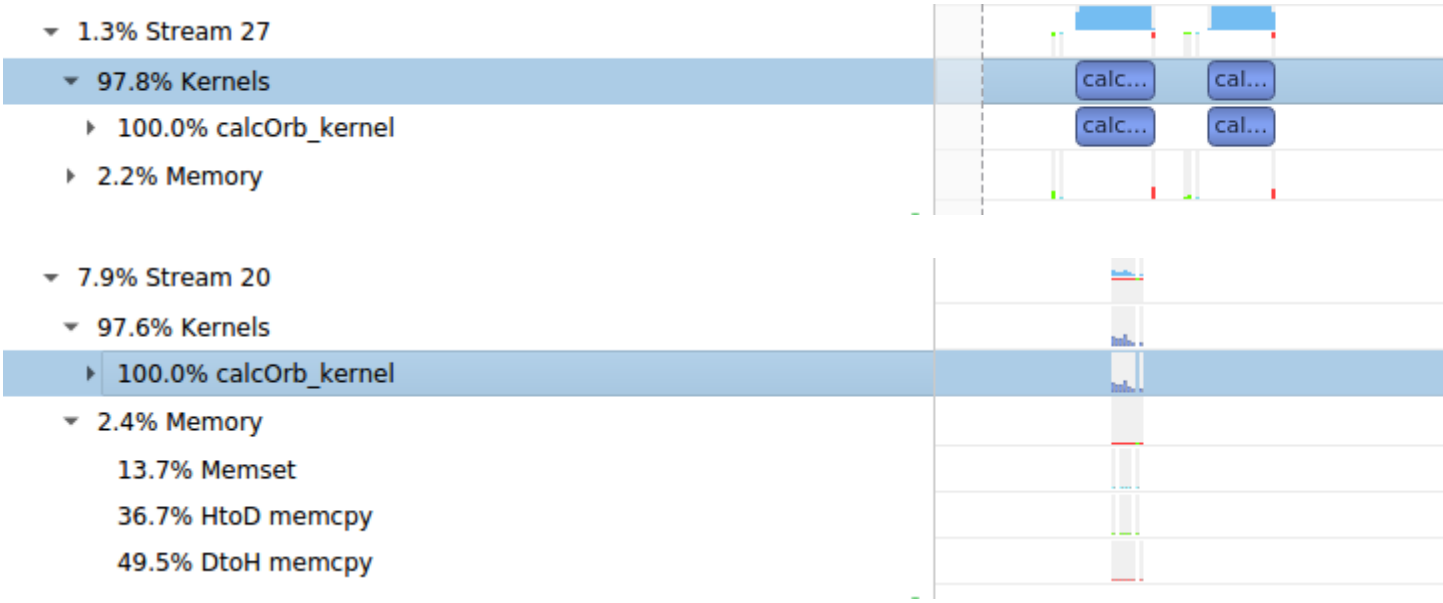
Most of the time run kernels:
Example:
tileCalckeypoints_kernel()

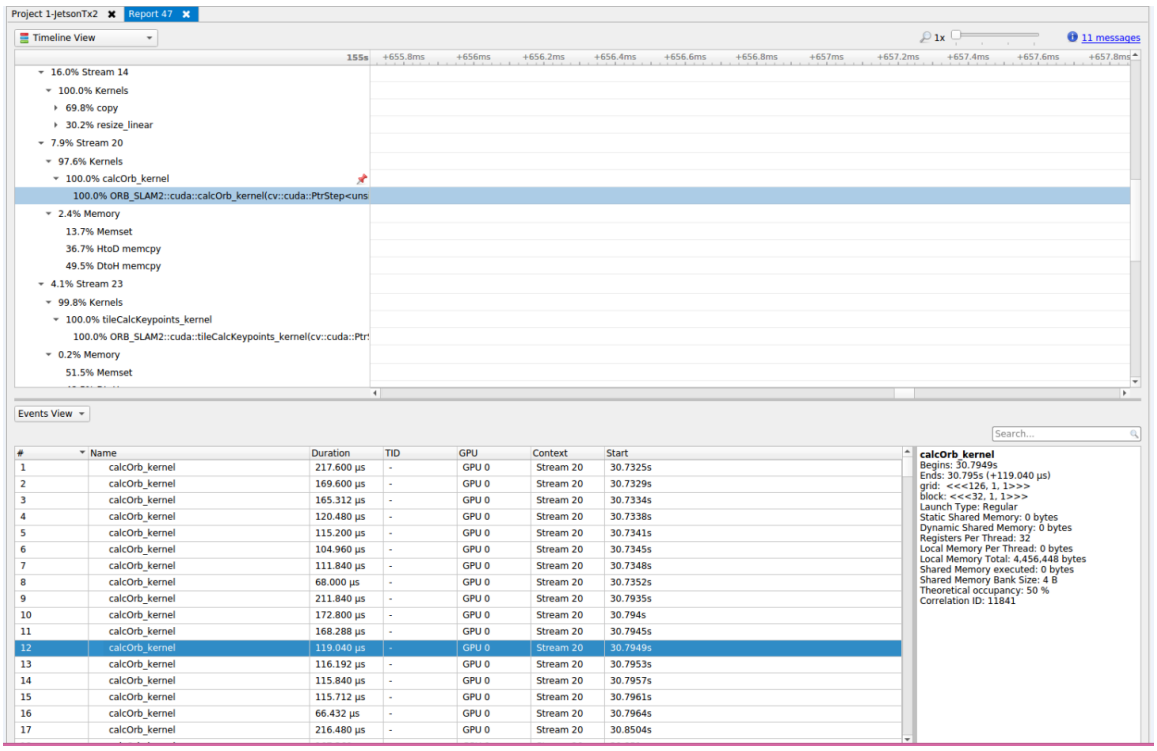






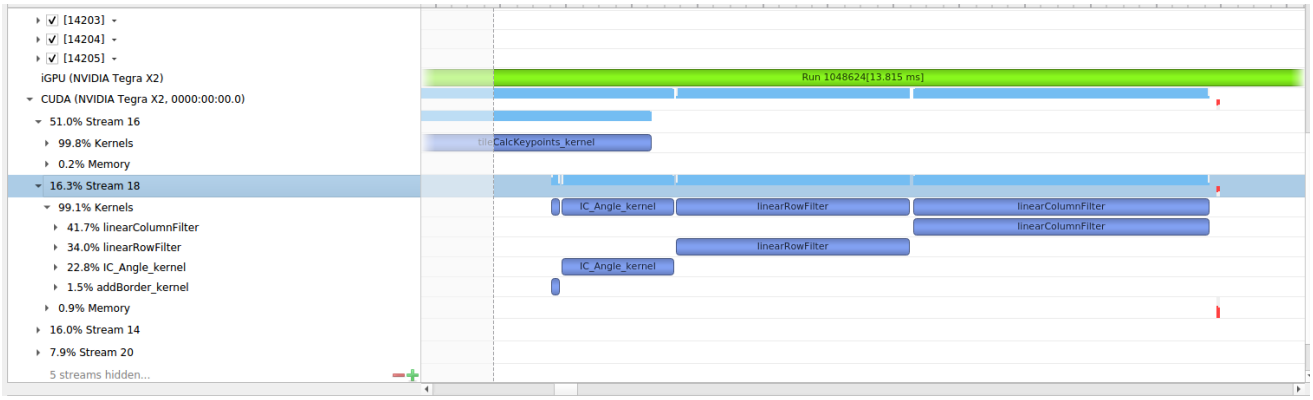
CalcOrb_kernel()



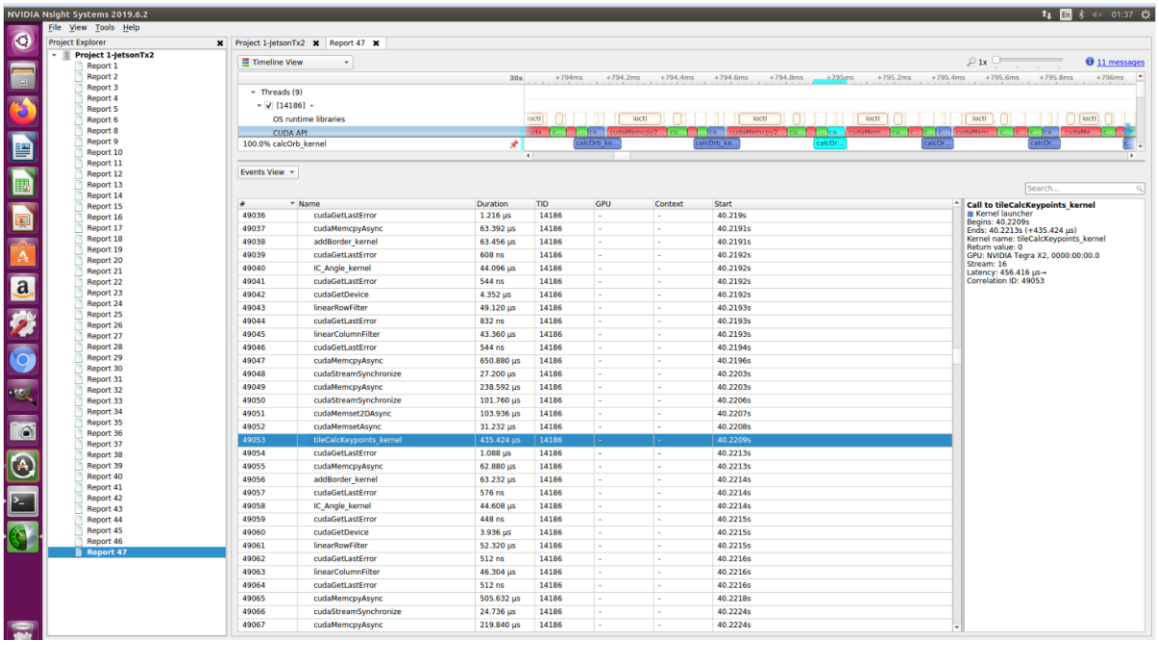


Another example:

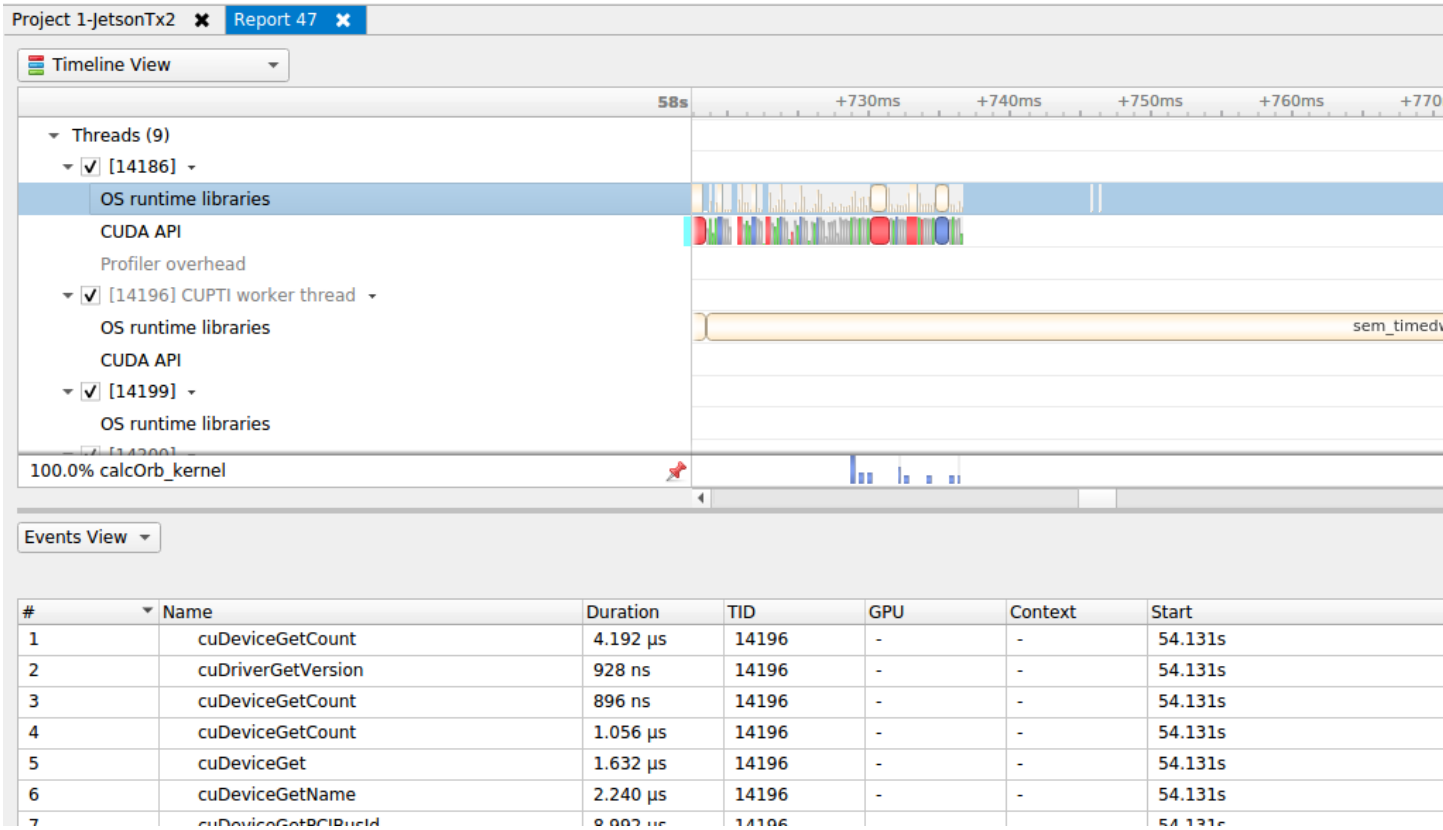
IC_Angle_kernel()



You can see timeline for the CUDA function and Cuda API:



You can see OS runtime libraries usage:



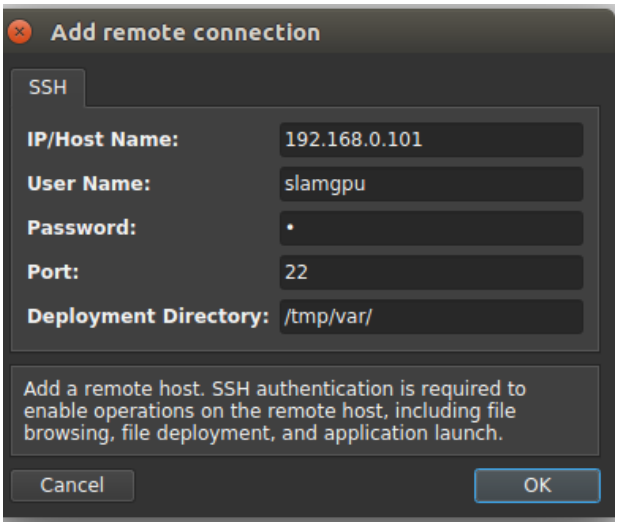
Nsight Compute

SLAM application in Standalone mode

Start the application with the following command

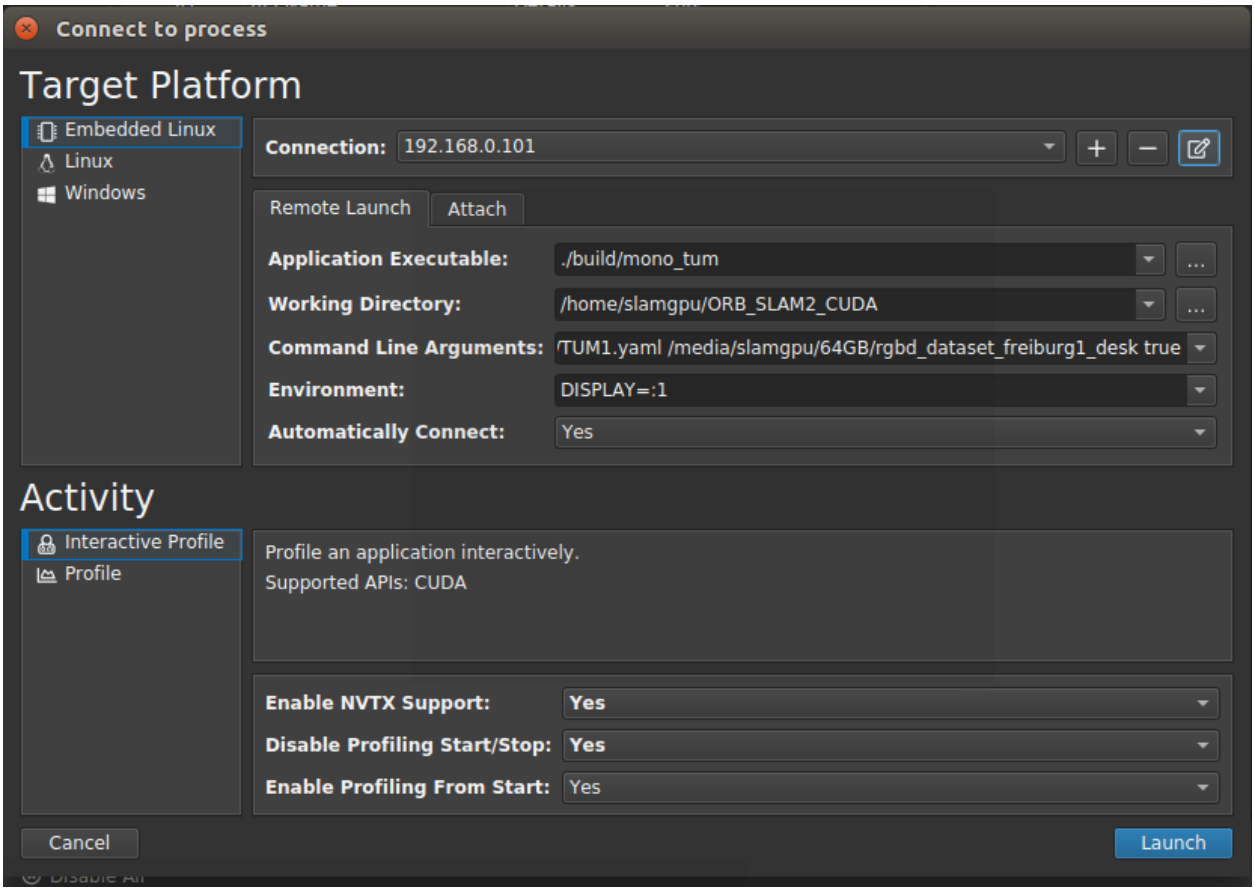
/usr/local/cuda-10.0/NsightCompute-1.0/nv-nsight-cu

Configure connection to Jetson TX2 :



Recommendation use root user instead slamgpu user

Configure the application parameters:



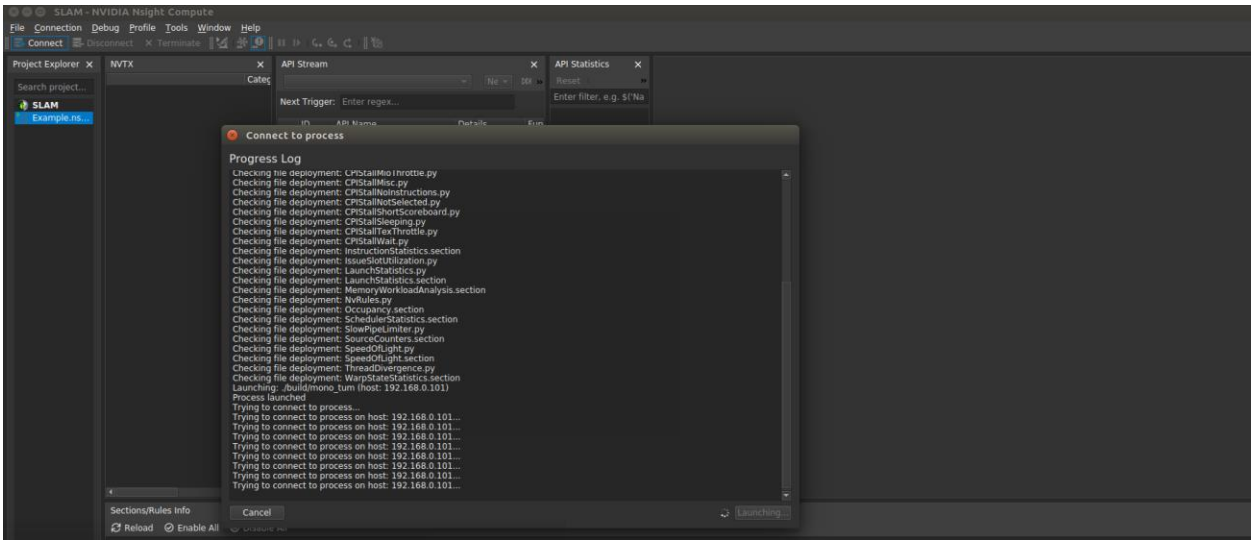
Application:
./build/mono_tum

Working Direcotry:
/home/slamgpu/ORB_SLAM2_CUDA

Application parameters:
Vocabulary/ORBvoc.txt Examples/Monocular/TUM1.yaml /media/slamgpu/64GB/rgbd_dataset_freiburg1_desk true

enviroment
DISPLAY=:1

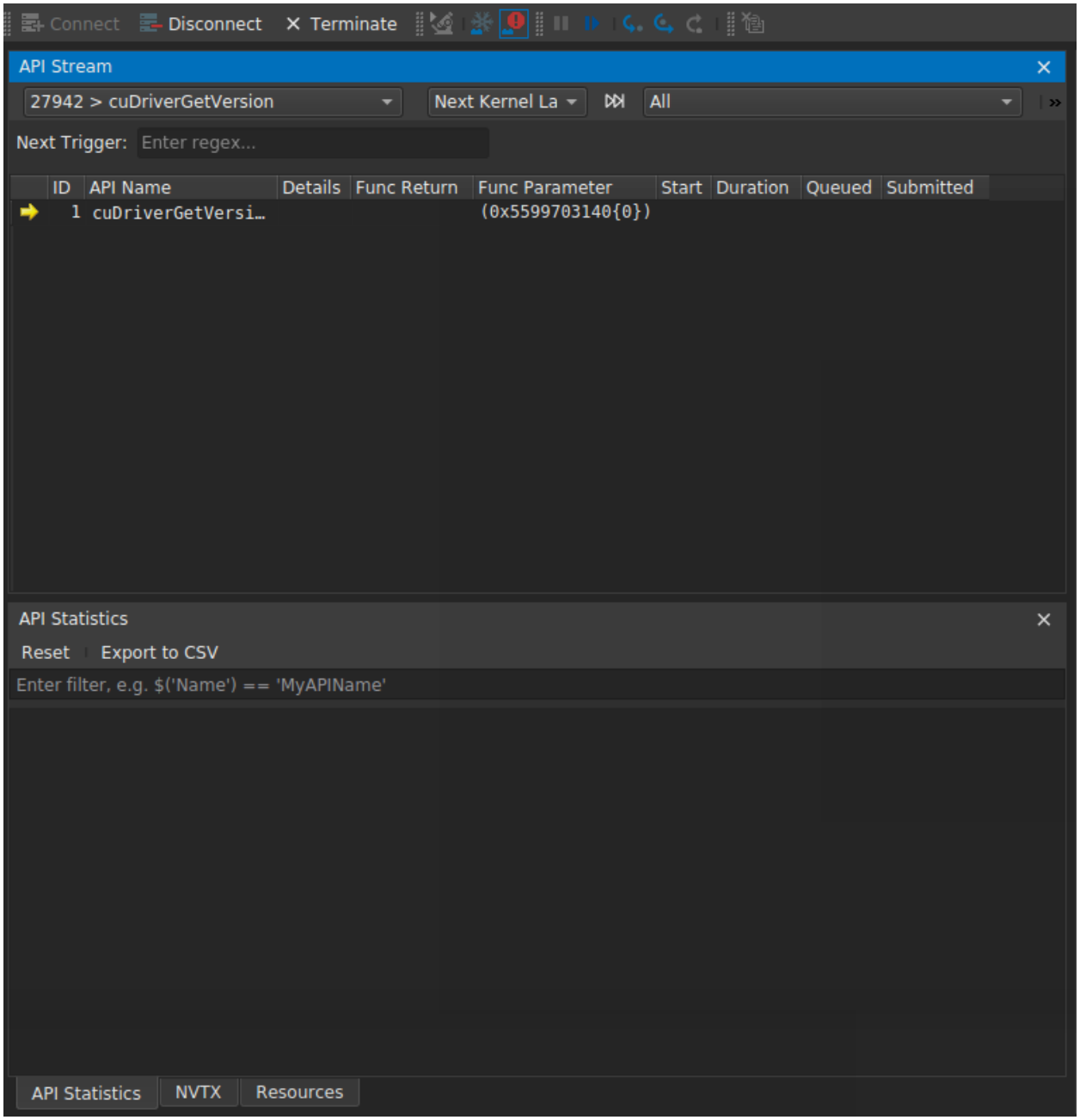
Use User root for interactive mode and click Launch to start


SLAM - NVIDIA Nsight Compute
File Connection Debug Profile Tools Window Help
Connect
Disconnect
Terminate
Project Explorer
Search project...
SLAM
Example.ns...
NVTX
API Stream
Next Trigger: Enter regex...
API Statistics
Reset
Enter filter, e.g. $('Na
Connect to process
Progress Log
Checking file deployment: CPIStallMioThrottle.py
Checking file deployment: CPIStallMisc.py
Checking file deployment: CPIStallNoInstructions.py
Checking file deployment: CPIStallNotSelected.py
Checking file deployment: CPIStallShortScoreboard.py
Checking file deployment: CPIStallSleeping.py
Checking file deployment: CPIStallTexThrottle.py
Checking file deployment: CPIStallWait.py
Checking file deployment: InstructionStatistics.section
Checking file deployment: IssueSlotUtilization.py
Checking file deployment: LaunchStatistics.py
Checking file deployment: LaunchStatistics.section
Checking file deployment: MemoryWorkloadAnalysis.section
Checking file deployment: NvRules.py
Checking file deployment: Occupancy.section
Checking file deployment: SchedulerStatistics.section
Checking file deployment: SlowPipeLimiter.py
Checking file deployment: SourceCounters.section
Checking file deployment: SpeedOfLight.py
Checking file deployment: SpeedOfLight.section
Checking file deployment: ThreadDivergence.py
Checking file deployment: WarpStateStatistics.section
Launching: ./build/mono_tum (host: 192.168.0.101)
Process launched
Trying to connect to process...
Trying to connect to process on host: 192.168.0.101...
Trying to connect to process on host: 192.168.0.101...
Trying to connect to process on host: 192.168.0.101...
Trying to connect to process on host: 192.168.0.101...
Trying to connect to process on host: 192.168.0.101...
Trying to connect to process on host: 192.168.0.101...
Trying to connect to process on host: 192.168.0.101...
Trying to connect to process on host: 192.168.0.101...
Sections/Rules Info
Reload
Enable All
Cancel
Launching...

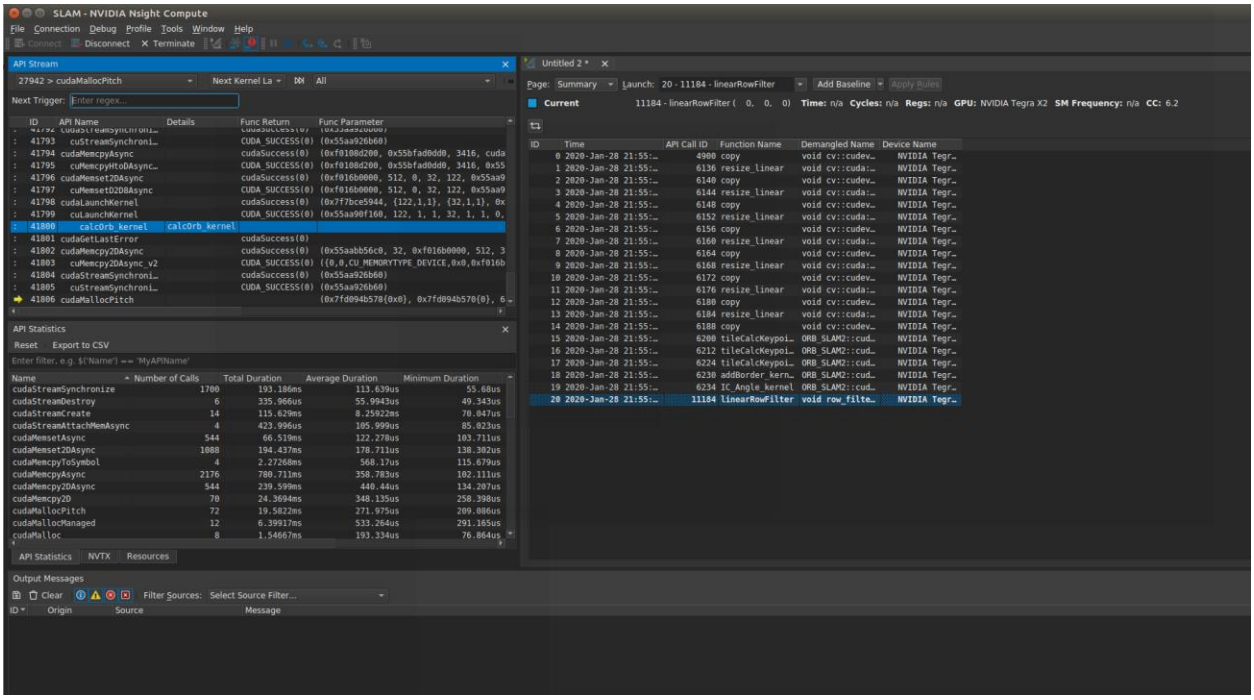
Waiting to application loading

When finished you start debug the application and jump between kernel and Cuda API.

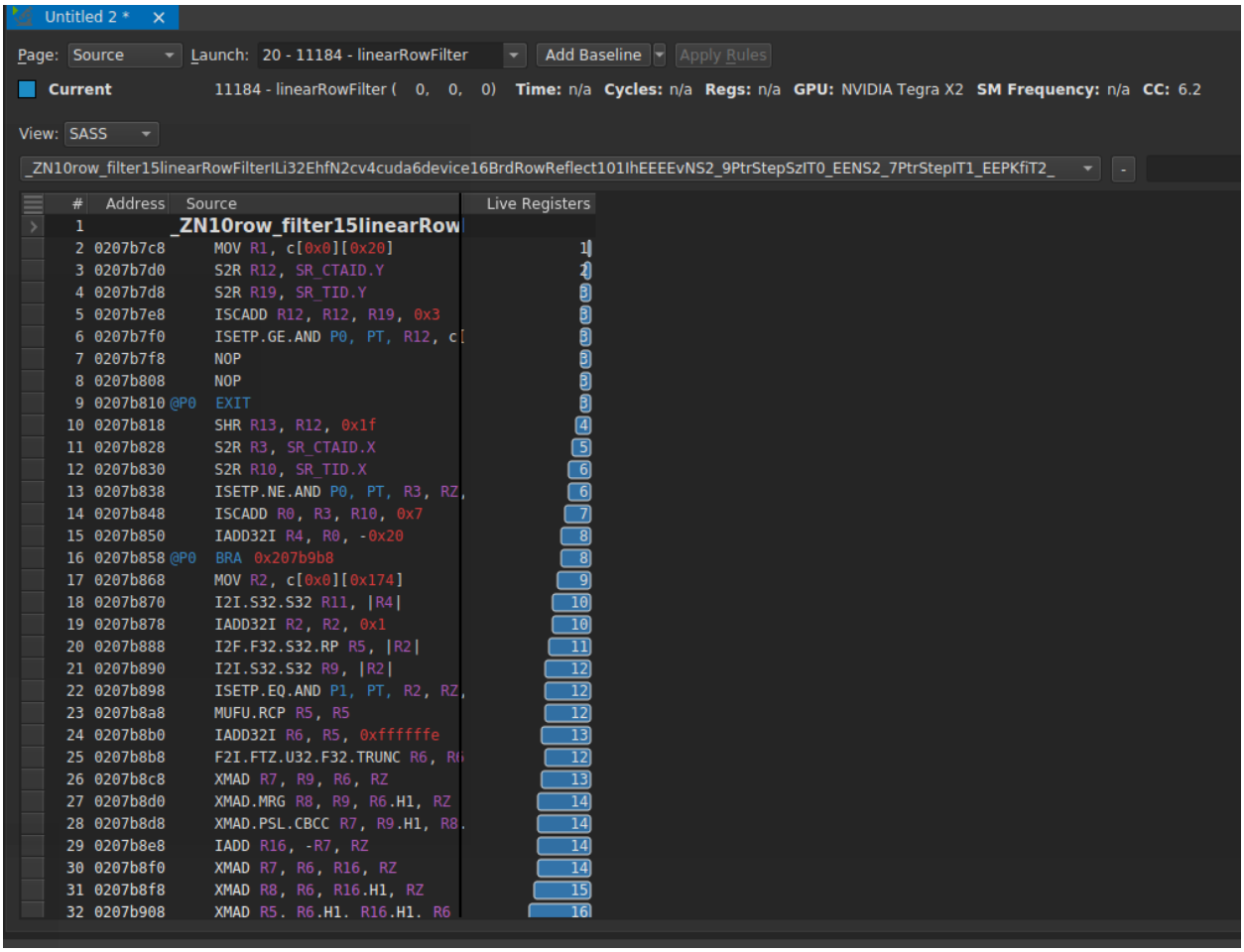
In the following image you can see the first call (Entry point) to Cuda API call.



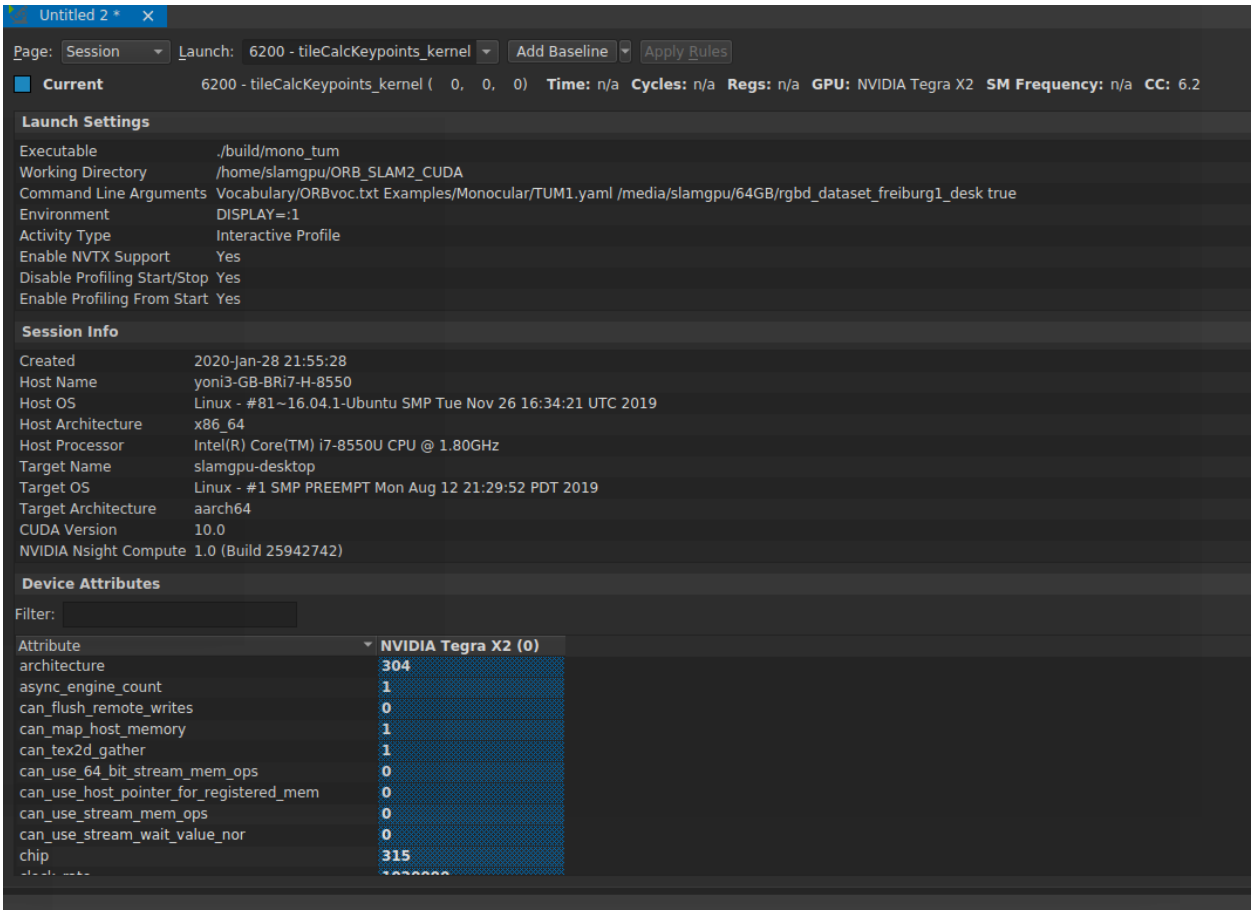
You can see the functions calls



You can debug source code



You can see infomation about the session


Untitled 2 *
Page: Session
Launch: 6200 - tileCalcKeypoints_kernel
Add Baseline
Apply Rules
Current 6200 - tileCalcKeypoints_kernel (0, 0, 0) Time: n/a Cycles: n/a Regs: n/a GPU: NVIDIA Tegra X2 SM Frequency: n/a CC: 6.2
Launch Settings
Executable ./build/mono_tum
Working Directory /home/slamgpu/ORB_SLAM2_CUDA
Command Line Arguments Vocabulary/ORBvoc.txt Examples/Monocular/TUM1.yaml /media/slamgpu/64GB/rgbd_dataset_freiburg1_desk true
Environment DISPLAY=:1
Activity Type Interactive Profile
Enable NVTX Support Yes
Disable Profiling Start/Stop Yes
Enable Profiling From Start Yes
Session Info
Created 2020-Jan-28 21:55:28
Host Name yoni3-GB-BRi7-H-8550
Host OS Linux - #81~16.04.1-Ubuntu SMP Tue Nov 26 16:34:21 UTC 2019
Host Architecture x86_64
Host Processor Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Target Name slamgpu-desktop
Target OS Linux - #1 SMP PREEMPT Mon Aug 12 21:29:52 PDT 2019
Target Architecture aarch64
CUDA Version 10.0
NVIDIA Nsight Compute 1.0 (Build 25942742)
Device Attributes
Filter:
Attribute
NVIDIA Tegra X2 (0)
architecture 304
async_engine_count 1
can_flush_remote_writes 0
can_map_host_memory 1
can_tex2d_gather 1
can_use_64_bit_stream_mem_ops 0
can_use_host_pointer_for_registered_mem 0
can_use_stream_mem_ops 0
can_use_stream_wait_value_nor 0
chip 315

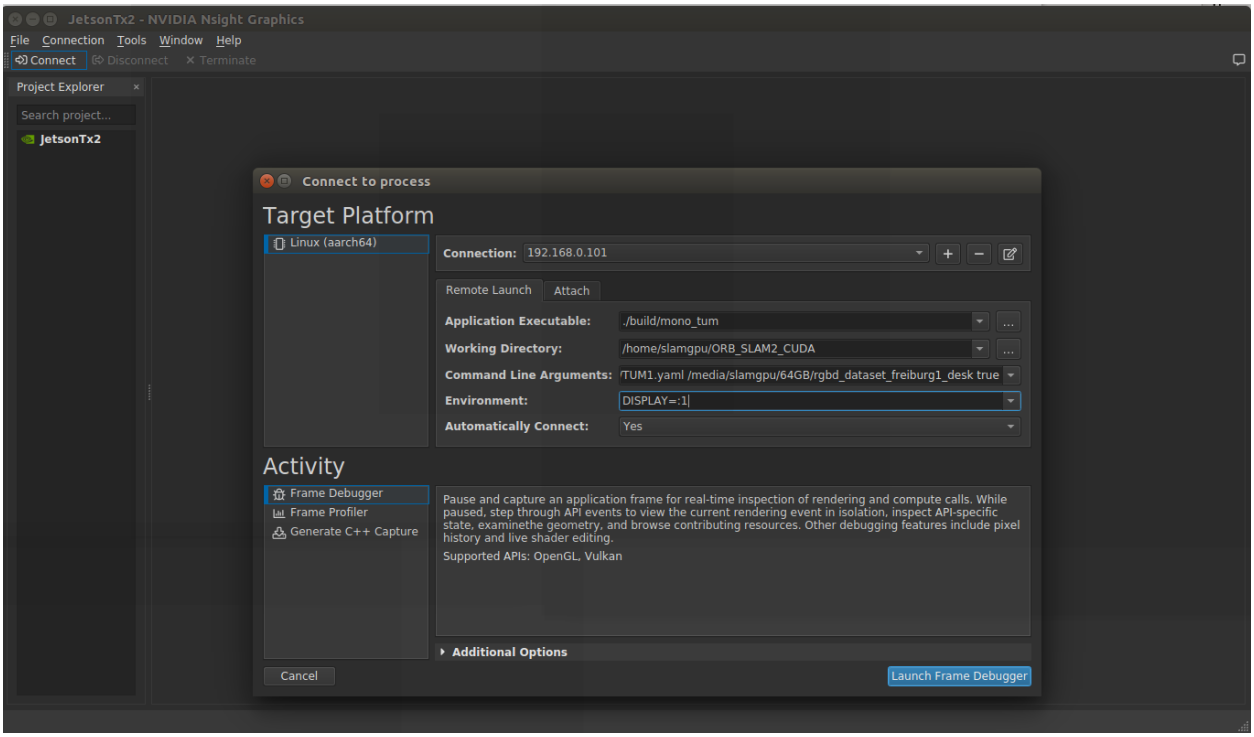
Nsight Graphics

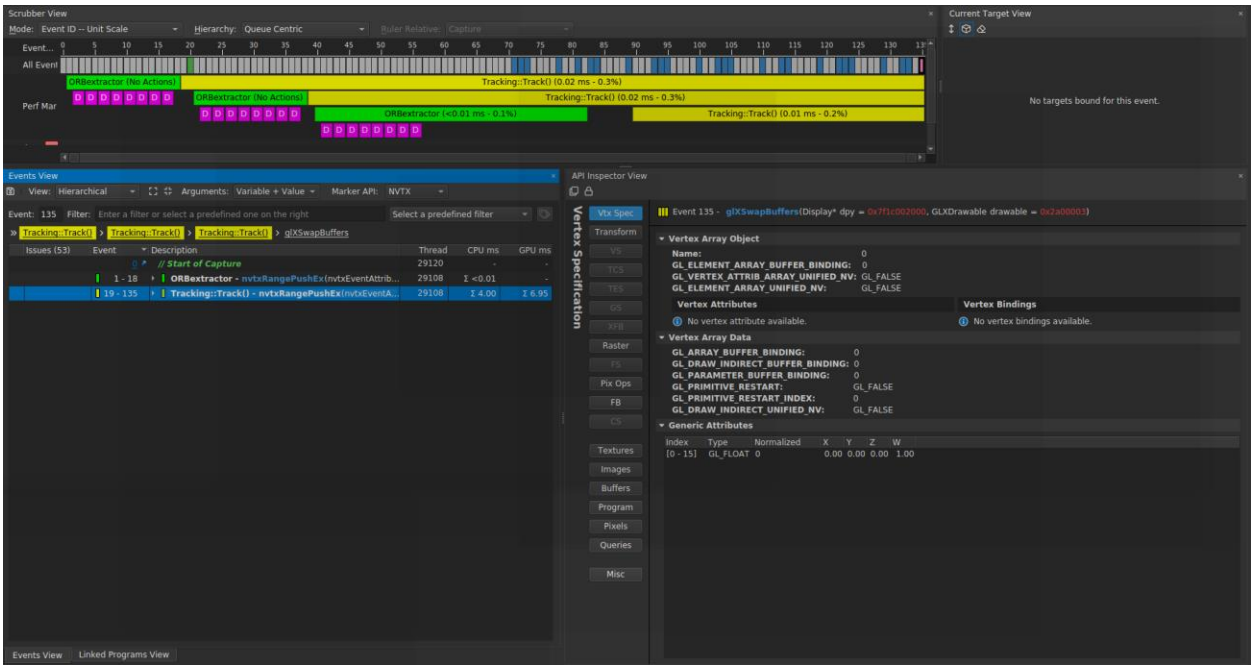
SLAM application in Standalone mode

You can launch the tool by command:

nv-nsight-gfx-for-l4t

Configure the connection to application




Scrubber View
Mode: Event ID -- Unit Scale
Hierarchy: Queue Centric
All Event
Perf Mar
ORBextractor (No Actions)
Tracking::Track() (0.02 ms - 0.3%)
ORBextractor (<0.01 ms - 0.1%)
Tracking::Track() (0.01 ms - 0.2%)
Current Target View
No targets bound for this event.
Events View
Hierarchical
Tracking::Track()
glXSwapBuffers
// Start of Capture
ORBextractor - nvtxRangePushEx
Tracking::Track() - nvtxRangePushEx
Events View
Linked Programs View
API Inspector View
Vertex Specification
Event 135 - glXSwapBuffers
Vertex Array Object
Vertex Attributes
No vertex attribute available.
Vertex Bindings
No vertex bindings available.
Vertex Array Data
Generic Attributes

Extensions

Enable root login over SSH

1. Login to your server as root.
2. As the root user, edit the sshd_config file found in */etc/ssh/sshd_config*:`vim /etc/ssh/sshd_config`(*For details on working with Vim check out our !)*
3. Add the following line to the file, you can add it anywhere but it's good practice to find the block about authentication and add it there.

   ```
   PermitRootLogin yes
   ```

4. Save and exit the file.
5. Restart the SSH server:
   `systemctl restart sshd`