Lab assistant: Andreas Axelsson

Contemporary Computer Architecture
Lab 3 CUDA + Camera

Introduction

The NVIDIA Jetson Nano is a small, powerful computer that enables the development of millions of new small, low-power AI systems. It's suitable for edge AI applications and research purposes. In this second lab we will start to explore the GPU of the Jetson Nano. Using CUDA (which is an acronym for Compute Unified Device Architecture) we can exploit the parallel power of the many cores found in the Nvidia GPU.

Learning Outcomes

By the end of this lab session:

- Understand how to clone, build and install utility packages.
- Create and run a simple CUDA + Camera program on the Jetson Nano.
- Do a simple image process task using CUDA.
- See how parallel code can speed up calculations.
- Use profile tools to measure performance.

Setup

In this lab we will use the Raspberry Pi camera to acquire images that we can do calculations on before we display them on the screen. There are many ways to capture images on the jetson. But since we want to be able to do this within our own C++/CUDA program, the easiest way is to use some kind of library. We will use a library called jetson-utils which is part of another library called Jetson-inference. The jetson-inference is a toolkit to do some machine learning tasks such as classification etc. on the jetson platform. Since we want to be able to compile our own programs and link these libraries, we will clone the jetson-inference project from Github and build it from sources.

Installation of new power supply

As some of you have already noticed, the power supplies used sometimes don't provide enough current to power the board. We have ordered other power supplies with 5V, 4A current capacity. To use this instead, do not connect the power supply to the micro-USB port. Instead use the newly provided power supply that connects to the barrel jack at the other end of the board. For the board to work with the new power supply we need to add a jumper to J48 (two header pins close to the barrel jack). Ensure this jumper is in place.

Installation of software

Step one is to make sure we have the needed tools. In a terminal run:

$ sudo apt-get install git

$ sudo apt-get install cmake

$ sudo apt-get install libpython3-dev python3-numpy

Now in your home directory make a directory called 'jetson' by running:

$ mkdir jetson

$ cd jetson

It is in this directory we will clone the jetson-inference project and build it:

$ git clone --recursive --depth=1 https://github.com/dusty-nv/jetson-inference

$ cd jetson-inference

$ mkdir build

$ cd build

$ cmake ../

$ make -j$(nproc)

$ sudo make install

$ sudo make ldconfig

This process can take quite a while, and there might also be some interaction where you need to answer some questions. You can use the default answer. There will be a question whether you want to install pytorch. It is used for deep learning. You may install it if you want, but at this point we don't need it.

After you have finished this installation, you can use the libraries in your own projects. It simplifies the task of acquiring an image from the camera, do some image processing and display the result.

Installation of the camera

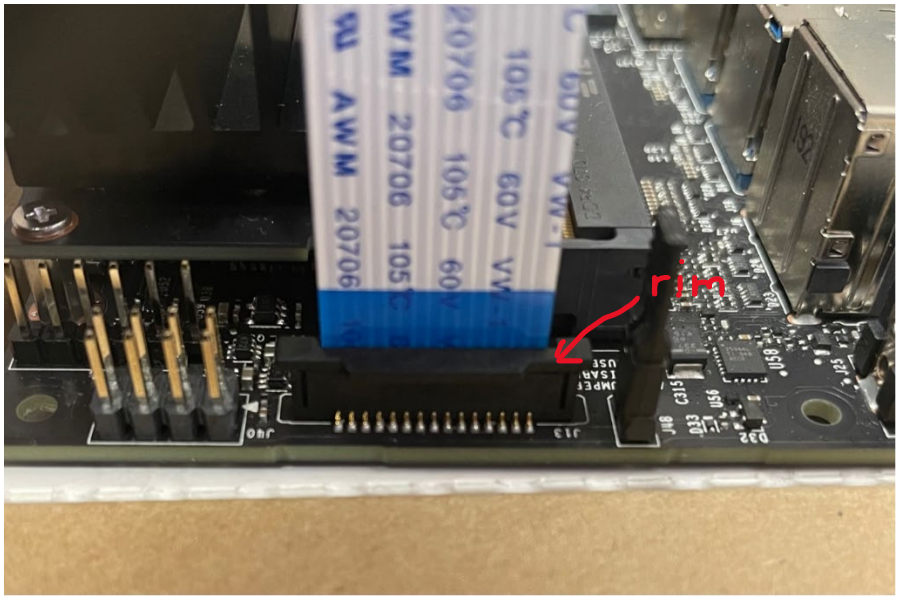
NOTE THAT YOU MUST NOT INSTALL THE CAMERA WHILE THE JETSON NANO IS POWERED ON.

As you have read above you may not install the camera hardware while we have power to the board, first do shutdown, and when the board has come to a full stop, remove the power cord to it.

NOW we can install the camera. It is attached via a small flat cable to a connector on the

edge of the board. Some of the jetson nano have two camera connectors and some have only one. If you have a device with two camera connectors, use the one designated CAM0, which is located closest to the power jack.

To attach the camera cable, you first must lift a locking latch on the connector on the board. If you look closely at the camera connector you can see (and feel) a rim around the connector. If you lift that rim gently you loosen the latch. When the latch is loose, you can insert the flat cable correctly oriented (see image below) and then press down the rim again to latch the cable in place.



Another important note: The camera can be a bit flimsy due to the long and springy flat cable. Ensure the camera doesn't fall on the jetson board as it might cause short circuits or other nasty things.

Now we have finalized the setup.

Exercise 1

Now in your come directory make a folder for lab 3, and add a new file ex1.cu and add the code on the next page into it. This program opens up the camera device and captures an image into a buffer and then views this buffer in a window. To compile and run our program:

$ nvcc ex1.cu -o ex1 -ljetson-utils

$ ./ex1

Note the -l directive. You have used it before with pthread. Now we link the jetson-utils library to our program.

```
#include <jetson-utils/videoSource.h>
#include <jetson-utils/videoOutput.h>

int main( int argc, char** argv )
{
    // create input/output streams
    videoSource* input = videoSource::Create(argc, argv, ARG_POSITION(0));
    videoOutput* output = videoOutput::Create(argc, argv, ARG_POSITION(1));

    if ( !input )
        return 0;

    // capture/display loop
    while (true)
    {
        uchar3* image = NULL;   // can be uchar3, uchar4, float3, float4
        int status = 0;         // see videoSource::Status (OK, TIMEOUT, EOS,
ERROR)

        if ( !input->Capture(&image, 1000, &status) )   // 1000ms timeout (default)
        {
                if (status == videoSource::TIMEOUT)
                    continue;

                break; // EOS
        }

        if ( output != NULL )
        {
            output->Render(image, input->GetWidth(), input->GetHeight());

            // Update status bar
            char str[256];
            sprintf(str, "Camera Viewer (%ux%u) | %0.1 FPS", input->GetWidth(),
input->GetHeight(), output->GetFrameRate());
            output->SetStatus(str);

            if (!output->IsStreaming()) // check if the user quit
                break;
        }
    }
}
```

Questions:

What is the purpose of the videoSource object: Create input stream

What is the purpose of the videoOutput object: Create output stream

What frame rate and image size are reported when running the application:

Where is the captured image stored: input

In what format is each pixel stored: uchar3

How much memory is needed per pixel: 3 bytes

How much memory is needed for the entire image: 2764800 bytes

Now if you try to profile your application using nvprof:

$ sudo nvprof --print-gpu-summary ./ex1

Run it for a while and then stop the application using ctrl+c in the terminal. If everything works correctly you will see some profiling results. It actually has run a CUDA kernel if you look under GPU activities.

Questions:

What is the name of the CUDA kernel, and what do you think is the purpose of it:

In average, how much time does it consume?

As you have seen in the code our image buffer is using the type uchar3.

Try to change the type to uchar4 and rerun the profiling.

In average, how much time does it consume now? _____

Try to explain any differences from before: Should take longer time _____

Does the image consume more or less memory after the type-change? Should be more _____

Hmmm……. Hmmm again…….

Exercise 2

The image buffer acquired from the videoSource object is allocated as a managed buffer as default. That means that we can use it both in CPU and GPU directly without explicit memory copying operations between host and device and vice versa.

That is good, because then it is easy to create a GPU kernel to launch and operating on the image and then display the results.

Your task is to create a kernel function that is called rgb2grayKernel that takes three parameters:

1) A pointer to the image data (uchar4* image)
2) An integer describing the width of the image (int width)
3) An integer describing the height of the image (int height)

This kernel should be used to take the pixel values and change the values so that the image appears as a gray scale image.

NOTE:

Our eye is not equally sensitive to the different color components Red (R), Green (G) and Blue (B). A very common formula to calculate the gray scale value from R, G and B is to use the following formula:

Gray = 0.299*R + 0.587*G + 0.114*B

Each pixel in the image now uses the uchar4 type. The uchar4 type is a struct having 4 unsigned chars x, y, z, w as members. They correspond to R, G, B and A.

What is A in this context? _____

For our application right now, is it of any use? _____

The TASK:

Convert the RGB values to gray scale in your own kernel using the GPU as efficiently as possible. How is a gray scale color represented in RGB? Before you start, make sure you have copied ex1.cu to ex2.cu and make your changes in the new file.

Your challenge is to ensure you access the memory the kernel in a smart way, and that you launch the kernel with a good setup during launch (how many threads and blocks should you use?)
Try to use the techniques you learned in lab 2.

Also try to implement the formula that calculates the gray scale value into a device function, that will be called by your kernel. This way it is easier to reuse code.

Use nvprof to measure the average time used by your kernel, and so how you can optimize your application to minimize this average time by considering the challenges mentioned above.

What is your best performance? _____

Exercise 3

So far, we have only modified the pixels in place (just overwriting the pixels with new values). If we want to do more advanced algorithms this might be a bad idea. Copy ex2.cu to ex3.cu

Modify your kernel so that it takes a uchar4* pointer for the input buffer, and one uchar4* pointer for the output buffer.
You have to extend your application so memory is allocated for the output buffer (will you use separate allocations for host and device buffer or use unified memory allocation?).

Also add code so both the original image and the converted image (gray scale) are shown in different windows.

Good work!!! Show the lab assistant your work and results. Lab OK: _____