Video Analysis using CUDA and OpenCV

Sam Radhakrishnan
Alphonso Labs

Session Overview

This session will cover introducing video analysis using CUDA, using a simple problem of detecting scene transitions. We will talk about

- Video Encoding and Decoding.
- Using FFMPEG to decode videos on the GPU.
- Using OpenCV to perform pixel computations on a frame in parallel.

Detecting scene changes

The content-aware scene change detection algorithm finds all areas in a video where the difference between two subsequent frames exceeds a threshold value. The proposed algorithm has the following steps -

- Decode the video.
- For each frame -
  - Convert the frame to HSV colour space.
  - For each channel, find the average pixel by pixel difference with the previous frame
  - Average the difference over the number of channels.
  - If greater than a given threshold declare the frame as a scene change

All these steps can be executed on the GPU

Video Codec

A video codec is an electronic circuit or software that compresses or decompresses digital video. It converts uncompressed video to a compressed format or vice versa.

A typical example is MPEG.

Video files often need to be decoded/decompressed before they can be used for analysis.

Hardware vs Software Video Decoding

Video Decoding is the process of converting video frames from the compressed codec format to raw video frames. This is a necessary step before one starts using the video frames to do anything.

There are two types of video decoders available - hardware and software.

Hardware decoders are dedicated multimedia chipsets, specifically meant to handle this task.

Software decoders use the CPU for decoding the video.

Hardware Decoders

According to the NVIDIA website - "*NVIDIA GPUs contain one or more hardware-based decoder and encoder(s) (separate from the CUDA cores) which provides fully-accelerated hardware-based video decoding and encoding for several popular codecs.* "

One can use ffmpeg to decode video frames using NVIDIA's decoders.

FFMPEG

FFmpeg popular open source library for transcoding, encoding and decoding video frames. FFmpeg is written in C and FFmpeg's integration of NVIDIA Video Codec SDK enables high performance hardware accelerated video pipelines.

One must build it from source to use NVIDIA Video Codec

Build ffmpeg for GPU - https://developer.nvidia.com/FFmpeg

General Purpose GPU - CUDA

CUDA is NVIDIA's parallel computing architecture that enables one to use a GPU for performing computations.

CUDA provides with C-like extensions that provide with fine-grained task and data parallelism.

CUDA is specifically suited for programs that have large matrices.

Analysing Video frames - OpenCV

OpenCV(Open Source Computer Vision Library) is a free and open source library for computer vision. OpenCV has plugins in C++, python and Java. The library supports multi-core processing and hardware acceleration.

How does a GPU help?

Video frames are essentially images. Images are represented as a matrix of intensity values on the pixel. Operations on matrices benefit from parallel execution.

For example:

Adding two vectors with 64 elements.

Using OpenCV with CUDA

OpenCV can be compiled with CUDA to take advantage of hardware acceleration. OpenCV provides with namespaces for cuda that provides with wrappers to run on top of CUDA. It uses a special data format called GpuMat to store “GPU” matrices

cv::cudacodec - OpenCV namespace for video encoding and decoding

Cv::cudarithm - OpenCV namespace for operations on matrices

Compiling OpenCV with CUDA
https://www.pyimagesearch.com/2016/07/11/compiling-opencv-with-cuda-support/

Combining it all

For building the scene detection module, we need the following

Use cv::codec::VideoReader to decode the video frames on GPU.
(https://docs.opencv.org/3.4.0/db/ded/classcv_1_1cudacodec_1_1VideoReader.html)

Use cv::cuda to perform individual computations on the decodec frames.
(https://docs.opencv.org/3.4.0/de/d09/group__cudaarithm__core.html)

For every frame -

```
cv::cuda::subtract( hsv_frame, prev_hsv_frame, difference );

sum = cv::cuda::absSum(difference);
```

Find the average across all channels

```
avg = (sum[ 0 ] + sum[ 1 ] + sum[ 2 ])/3.0;
avg /= ( double ( this->num_pixels_per_frame ) );
```

If the avg > threshold then declare a scene transition

Some experimental Stats

The time taken for execution for running GPU vs CPU versions of the above algorithm on a 1 min, HD mp4 file on a ubuntu 14 machine running OpenCV version 3.4 and CUDA 10.

Software decoder without CUDA - 18s

Software decoder with CUDA - 11s

Hardware decoder with CUDA - 2s

Limitations

Nvidia Video Codec SDK does not work with all container formats, even if the underlying codec is supported.

At most two encoding sessions at a time for most GPU architectures.

Alternatives

Nvvl - A library that uses hardware acceleration to load sequences of video frames to facilitate machine learning training.

Thank you

Appendix

OpenCV with CUDA - https://www.pyimagesearch.com/2016/07/11/compiling-opencv-with-cuda-support/

FFMpeg with CUDA - https://developer.nvidia.com/FFmpeg

OpenCV docs - https://docs.opencv.org/3.4.0/db/ded/classcv_1_1cudacodec_1_1VideoReader.html

Nvvl - https://github.com/NVIDIA/nvvl