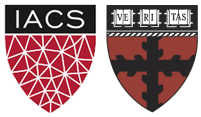
CS205, Fall 2013
Computing Foundations for Computational Science
Cris Cecka



HARVARD
School of Engineering
and Applied Sciences

Homework 5
Due November 22nd, 2013 at 11:59pm ET

You just got hired to work on videos for Youtube. Excited about that, but lacking the fundamentals of video processing, you decide to grab a book and study.... but then, your favorite class comes through with a homework that will definitely help and let you save tons of time!

In this homework you will manipulate videos using multiple GPUs. As such, you will combine your knowledge of CUDA programming on the GPUs with the power of MPI parallelization. You will also use Cheetah metaprogramming which allows you to generate code on the fly.

Download the code for HW5 here.

or

```
wget http://iacs-courses.seas.harvard.edu/courses/cs205/resources/hw5.zip
```

Problem 1 - Generalized linear filters for videos

Having coded a linear filter for images on a GPU in HW4, you are now an expert in this technique on the GPU. We are now interested in applying this method for videos, on a per-frame basis, and for generalized filters.

Executing

Execute the code from the headnode with the provided `run.qsub` which uses the queue and batch:

```
#$ -q gpubatch.q
#$ -pe ortegpu_reserve 1
```

CUDA

You are provided with a sample code that reads a video file, precomputes a function that operates on frames, applies that function to each frame of the video, and writes an output video. This processing is done per-frame and per-pixel in serial. That is, it computes the convolution

$$\text{out}[i][j][\ell] = \text{offset} + \sum_a \sum_b \text{F}[a][b] \times \text{in}[i + a - A][j + b - B][\ell]$$

where i, j is the pixel coordinate, ℓ is the color channel, and A and B are offsets so the kernel is centered. You may use any strategy for out-of-bounds pixels – e.g. treating out-of-bounds as zero, not computing the filter for border pixels, etc.

Your task is to modify `make_frame_processor` to precompute a function that uses CUDA to apply the filter. We could store the filter in global or `__constant__` global memory as in this example to implement this general linear filter in CUDA. However, accessing memory and using registers in a CUDA kernel can be costly: for instance, multiplying 5 variables is more costly than multiplying 5 hard coded constants (in the latter case, this is even done at compilation rather than at runtime). Instead of a classical approach, we will Cheetah to build a custom CUDA kernel for any filter of any size.

These linear filters are rather cheap to apply, so the overhead of CPU-GPU transfers will make it difficult to beat OpenCV's implementation used in `serial.py`, optimize!

HINT: In your CUDA kernel, it is easiest to use `uchar3*` for the input and output data. With `uchar3*` access the BGR channels with `.x`, `.y`, and `.z`. You may want to copy pixel data into floats and convert back with `make_uchar3(·,·,·)`. Take into account coalescing and instruction-level parallelism.

HINT: We want to perform as little work inside the `frame_processing` function as possible. Thus, we would like to do as much setup and precomputation in `make_frame_processor` as possible. Instead of using `to_gpu` and `get`, which perform allocations on the GPU and CPU before copying data, try using `pycuda.driver.memcpy_htod` and `memcpy_dtoh` with pre-alllocated memory on the host and device.

MPI + CUDA

Still not fast enough? Use multiple GPUs with MPI. See this example.

There are two GPUs per node on the resonance cluster. To make sure each process can initialize its own GPU, use a `$round_robin` MPI process allocation. The `ortegpu_reserve` parallel environment has been set up for this on resonance.

Since frames can only be loaded from a video stream consecutively, you have the option to have the master process load the whole video and scatter the necessary frames to the other processes, or have all the processes create the video stream and select a subset of the frames to work with.

Extra Credit

Generalize even further so that your frame operations can handle even more general operations. For example, consider the "Take the minimum/maximum of the pixels around me" filter. Can we parameterize more general operations than just multiplication and addition?

Document what modifications you made and what else your video processor can do.

Resources

Cheetah engine

CheetahSAXPY.py

CheetahMORE.py

HelloMultiGPU.py

Submission Requirements

- `P1.pdf`: Explanation and analysis of your results.
- `P1.py`: MPI+CUDA video general linear filter.