15-418 Parallel Computer Arch., Spring 2021

Project Proposal

Due Date: Friday April 7, 2021 23:59 EST

Name(s): Elan Biswas, Gustavo Silvera

AndrewID(s): ebiswas, gsilvera

Summary:

For our project, we plan on implementing and optimizing a video frame interpolation algorithm using `NVidia Cuda` GPUs and other parallel techniques from 15-418. We will provide metrics from the resulting optimization and measure the speedup compared to reference sequential implementations.

Background:

Interpolation algorithms primarily use adjacent frames in a video to construct an intermediate frame that could be interleaved in the video, increasing the number of frames-per-second relative to the original. We should have multiple avenues for parallelism that we can explore similarly to assignment2: across pixels and across frames.

The computer vision algorithms heavily rely on matrix operations that operate on entire frames of pixels at once, for an $(n \times m)$ resolution image this could likely be parallelized heavily. Additionally, since interpolation in different parts of the video would be completely independent (for simple algorithms) we should also be able to parallelize our algorithm accross multiple different frames concurrently.

Finally, if we accomplish the above `cuda` implementation we'd also like to parallelize our host code with `OpenMP`, and eventually parallelize our algorithm further across several machines working independently and communicating through `MPI`.

The Challenge:

Within the frame-computation in most computer vision algorithms for frame interpolation, it is difficult to parallelize completely across pixels because neighbouring pixels are often used. Communication across pixels would limit our parallelism due to memory accesses and adding some sequential bottlenecks. To counteract this, we'll need to make sure that we have enough work to do in our `CUDA` kernels so that thread pipelining can take advantage of doing more work while waiting on the slow (GPU) device memory.

We also will be limited in the number of frames we can parallelize because individual frames could require large amounts of memory ($(n \times m \times 3)$ `float`'s for an rgb image) which could easily overflow the device memory for long videos. This may be an avenue where we could split up videos and process them in parallel (across multiple GPU-accelerated computers) with `MPI` communication at the beginning and end.

Resources:

In particular, we'll start by looking at a *Phase-Based Frame Interpolation* algorithm[1] for its ease of implementation and potential for parallelism. This algorithm makes use of phase-based methods to represent motion of individual pixels instead of common methods such as Lagrangian optical flow correspondences.

We won't be starting from a useful codebase because the only open-source implementations available are written in `python` and are cpu-based and "highly un-optimized"[2] so we will be implementing the core algorithm from scratch. We will use this codebase[2] as a benchmark once ours is completed, and ideally ours is magnitudes faster since it leverages the GPU

Access to the Gates machines would be useful because we would like to use the `RTX 2080`'s onboard the machines. We may also consider using multiple (different) Gates machines at once in order to have multiple gpu devices working at once on separate `MPI` process instances.

1) studios.disneyresearch.com/wp-content/uploads/2019/03/Phase-Based-Frame-Interpolation-for-Video.pdf
2) github.com/justanhduc/Phase-based-Frame-Interpolation

Goals and Deliverables:

Plan to achieve: Ideally we could get a fully functional and correct implementation of the video interpolation algorithm working with GPU acceleration. We do not have metrics on our initial sequential implementation (since we have not started it yet), but should hope to achieve near-linear speedup in comparison.

Hope to achieve: Once the above is completed, we'd like to implement the `MPI` optimization to obtain further speedup across multiple GPU devices.

Presentation: By the end of the semester, we'd like to present our project by showcasing some example videos of an original source (playing at 15fps for example) side by side to our interpolated/smoother render (ideally playing at 30+fps).

Platform Choice:

We will be working with `C++` and `CUDA` on the Gates machines and `gsilvera`'s own `goosinator` machine which has an RTX 2080 super 8GB. If we get to the `MPI` implementation, we'd like to also have these machines work on the same video concurrently.

`C++` is a good choice for our project since it has clean integration with `CUDA`, `MPI`, and has many built-in data structures through the `std` libraries that we can make use of. Both group members are comfortable with `C++` and `CUDA` and `MPI` so we should not have many problems getting started.

Schedule:

Week	Discussion
1	...
2	...
3	...
4	...
5	...
6	..
7	...