Accelerating the WebP Decoding Pipeline

Kevin Geng
Emma Liu

URL: https://emmaloool.github.io/15418-Final-Project/

SUMMARY:

We intend to implement one of the stages of the WebP image decoding pipeline on NVIDIA GPUs, to take advantage of the parallelism offered by GPUs. To do so, we will study the algorithms used and the reference implementation of WebP decoding, and rewrite one portion of the pipeline to best take advantage of GPU computing.

BACKGROUND:

Image compression involves the reduction of image data without degrading the quality of its visual perception. It is a canonical problem in the intersection of computational photography and high-performance computing as it relates to fast computation of large datasets because of the redundant nature of image representation (i.e., pixels) among colors and similarity among pixels.

There are two main categories of compression algorithms for images: lossless compression, which is able to recover the original image data with no loss of quality, and lossy compression, which sacrifices the possibility to exactly reconstruct of the image for smaller file sizes.

If we are able to develop fast image encoding and decoding routines for GPUs, this will aid in cases where large amounts of image data need to be converted. Additionally, such optimized routines can be used to achieve better compression ratios for images before publishing them on a web site, which is important for reducing the bandwidth consumed by visitors to the web site.

Finally, a fast implementation of image encoding/decoding can aid in the practical implementation of other GPU-based, compute-intensive image processing algorithms. With such algorithms, only the compressed image is to be stored in main memory; the image could be encoded and decoded on the fly after it arrives in GPU device memory, allowing greater flexibility by reducing the amount of main memory required for temporary storage.

NVIDIA has released a library, nvJPEG, which serves the purpose of a GPU-accelerated JPEG decoder (of the JFIF image encoding format), with one of their main goals being to efficiently ingest image data in order to efficiently train machine learning models.

We hope to target the lossy WebP format (based upon VP8 key frame encoding), which serves a similar purpose: to compress files when it is okay to lose some accuracy. Since WebP is a much

more modern format, it is able to compress images more efficiently than the well-known JPEG format. However, this also means that it uses several new techniques for which GPU primitives may not have yet been implemented, leaving room for us to implement them.

CHALLENGE:

The challenge is to implement an image encoding / decoding algorithm in a way that is able to perform efficiently on GPU devices, and can surpass optimized implementations that have been developed on CPUs.

The benefit from using GPUs comes from being able to exploit a large amount of parallelism, both through parallel threads of execution, and SIMD vectorized primitives. The ability to do so requires specialized algorithms: for instance, code that requires a large amount of branching will not perform well due to issues with warp divergence that we discussed in Assignment 1. Furthermore, as with any algorithm, parallelizing code that is written in a serial manner is not trivial. Thus, it is necessary to examine how to implement algorithms in a fashion that performs well when GPU computing is used.

Of course, image formats contain quite a bit of complexity, and it would be very difficult for us to implement end-to-end the entire image encoding / decoding pipeline. As such, we will likely choose a specific part of the pipeline to improve on using GPU computing.

Since the WebP format is relatively new, it incorporates several features that are not used by the JPEG format. Namely, it takes advantage of prediction coding, of adaptive block quantization, and boolean arithmetic encoding. Most likely, we will focus on one of these aspects of WebP encoding to implement in a GPU-accelerated fashion.

RESOURCES:

For our preliminary research, we found several good papers that lay out general motivations for image compression and algorithms.

- A sample chapter of "Multimedia Systems Technology: Coding and Compression"
- Sarang Bansod and Shweta Jain. *Recent Image Compression Algorithms: A Survey* (2013). https://www.ijarcce.com/upload/2013/december/IJARCCE8A-s-Sarang_Bansod_RECENT_IMAGE.pdf

Papers examining the implementation of lossless data compression algorithms on the CUDA architecture for general-purpose GPU computing, and the associated challenges that others have encountered in doing so:

- Adam Ozsoy and Martin Swany. *CULZSS: LZSS Lossless Data Compression on CUDA* (2011). https://web.cs.hacettepe.edu.tr/~aozsoy/papers/2011-ppac.pdf
- Keh Kek Yong, Meng Wei Chua, and Wing Kent Ho. *CUDA Lossless Data Compression Algorithms: A Comparative Study* (2016). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7881980&tag=1

We are targeting the WebP lossy compression algorithm, so we would be able to use the reference implementation of the encoding/decoding procedures as a reference. In part, we chose the WebP format because despite being relatively recently developed, it has gained popularity due to its adoption by major web browsers, and it has been standardized. As such, it is more likely than other modern image formats to have good documentation and multiple well-tested implementations.

- VP8 Data Format and Decoding Guide (RFC 6386)

PLATFORM CHOICE:

We plan to use the GHC cluster machines, in order to take advantage of the GPUs available for use on those machines. We believe that GPUs provide the best platform for studying image compression. Existing code for encoding and decoding images is written for and optimized for CPU workloads. However, the compression of a single image is usually not significant enough to benefit from parallelization over multiple machines.

Using GPU technology strikes the perfect balance between efficiency in that it can speed up image encoding / decoding by taking advantage of parallelism, and practicality in that it does not require the use of a compute cluster for something which should be a fast and efficient operation.

GOALS AND DELIVERABLES:

At the poster session, we should be able to present a live demo of image encoding or decoding running in real-time, even if only part of the process is GPU-accelerated. (However, this probably won't be very interesting, since encoding is generally very fast!) We would also be able to display images describing the particular part of the WebP pipeline that we focused on, and why it helps improve image compression.

We **plan to achieve** a GPU-accelerated implementation of one part of the WebP decoding pipeline. This part of the implementation should be measurably faster than the corresponding serial implementation.

We **hope to achieve** a GPU-accelerated implementation of the same part of the pipeline, but for *encoding* instead of decoding.

SCHEDULE:

Monday, November 4, 2019
Read description of WebP image encoding / decoding algorithm in detail.

Monday, November 11, 2019
Set up project space (migrate source code from libwebp to our repository).
Investigate initial viable opportunities to express parallelism in encoding pipeline.
Decide on which aspect(s) of the pipeline we should study for parallelization.

- Vp8l_enc.c: ApplySubtractGreen
- The rest of the transforms
- Compare different CUDA implementations of the same transforms?
- AnalyzeHistogram

Wednesday, Nov 13, 2019

- **Reconfigure build system to support CUDA**
- **Implement a basic direct CUDA translation of the pipeline stage that compiles**
- **Revisit & refine list of opportunities for parallelization in this stage**
- **Consolidate test cases (what are good images to compress?)**
- **Determine how to display analysis**
    - **Speedup**
    - **Computation time**
    - **Four different transforms - investigate how one is chosen?**
- **Integrate timing code before/after function to parallelize, compare to sequential**

Monday, November 18, 2019
Test out at least one opportunity for parallelization
Submit the **checkpoint report** by midnight

Monday, November 25, 2019
Compile performance measurements for different parallelization opportunities
Test out at least one other opportunity for parallelization

Monday, December 2, 2019
Complete deliverable
Prepare for poster session and documentation

Monday, December 9, 2019
Complete writing and submit the **final report** by midnight.