REPORT OF TERM PROJECT

Name: Zuheng Kang
MUID: 006133440

INTRODUCTION

This project is about the algorithm of JPEG graphics compression and decompression using CUDA. The detail of algorithm can be find in Wikipedia: https://en.wikipedia.org/wiki/JPEG. The sequential method is very straightforward, however in this project, I mainly focus on the parallelization of graphics coding and decoding with Fourier DCT (discrete cosine transform) algorithm, and zigzag transform algorithm with CUDA. However, the Huffman Tree compression part is not in this project since it can only be calculated with sequential algorithm.

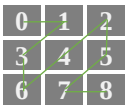
The project file is also posted in `github`: https://github.com/RobertBoganKang/JpegModelCUDA. The file also attached into the PDF.

PARALLEL IMPLEMENTATION

Since the mechanism of JPEG coding and decoding is already written in Wikipedia, here I will explain the special implementation of my algorithm of parallelization.

Zigzag Traversal

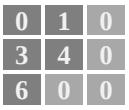
An example of zigzag array, if given a matrix.



And given a length of array that implemented the matrix zigzagged traversal of 5.

The result is an array {0, 1, 3, 6, 4}.

And if restore back to the matrix which is,



The restoration algorithm follows the original patterns of encoding with specific length, however other empty slots are filled with 0s. This algorithm will drop the high frequency wavelets of 2D image and preserve the low frequency property to achieve the image compression.

Since each image block (8 * 8 pixels) will all implement this zigzag traversal algorithm for encode and decode, it is better to use index array that guide CUDA to encode and decode the zigzag array.

Namely, the coordinate of the first element is {0, 0}, the second is {1, 0}, follow this patter, we can gather the index array: {{0, 0}, {1, 0}, {0, 1}, {0, 2}, {1, 1}}; the total length is 5. Then we flatten this array into {0, 0, 1, 0, 0, 1, 0, 2, 1, 1}, the $2i^{th}$ number is the i^{th} x's coordinate, and $2i+1$ is the y's coordinate.

These blocks ($G_{u,v}$ in Eq. (1)) will have a zigzagged traversal to generate an array, which will later be merged into a large array as compressed data. For example, we have 16 * 16 pixels, which is 2 * 2 blocks with encoding length 2, and each block zigzag encoding is {1, 1}, {2, 2}, {3, 3}, {4, 4} respectively, thus the encoding large array is {1, 1, 2, 2, 3, 3, 4, 4}.

Assume that the length of zigzag transform is l and there are m by n such pixel blocks, the time complexity of encode and decode for sequential algorithm is $O(m \cdot n \cdot l)$ reduced to $O(1)$ if implemented parallel with CUDA.

Fourier Discrete Cosine Transform (DCT) with CUDA

This project only deals with the image in gray scale color space; the color space of colored image will be in RGB (CMYK, YUV) color space will only implement this method of each color space attributes.

The image will be partitioned into small 2D squared blocks (the standard for JPEG is 8 * 8), and each image block will be implemented Fourier DCT algorithm in parallel with different blocks in GPU respectively.

The theoretical expression for Fourier DCT is:

$$G_{u,v} = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7}\left(g_{x,y}\cos\left(\frac{(2x+1)u\cdot\pi}{16}\right)\cos\left(\frac{(2y+1)v\cdot\pi}{16}\right)\right) \tag{1}$$

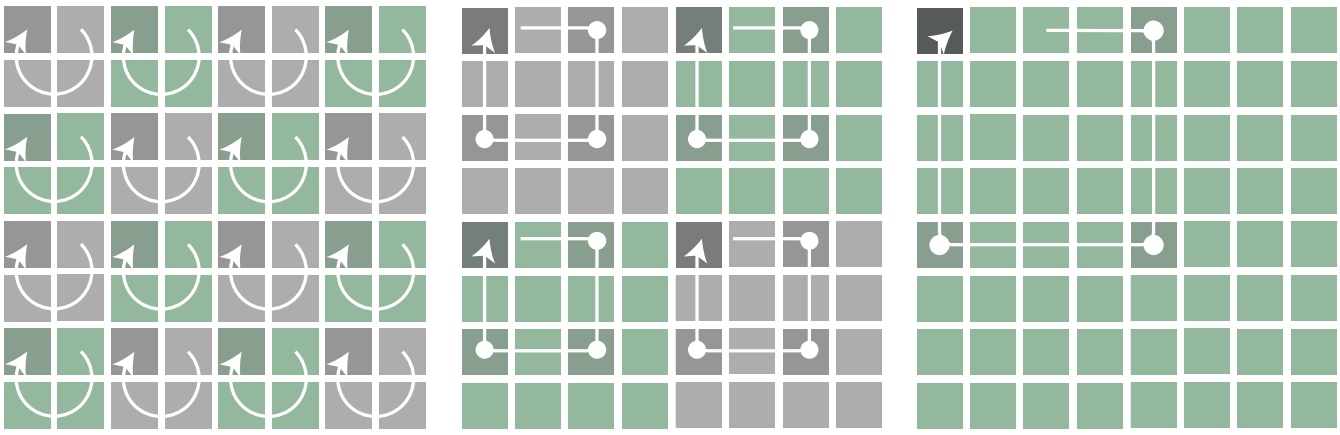
The inverse Fourier DCT is:

$$f_{x,y} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7}\left(\alpha(u)\alpha(v)F_{u,v}\cos\left(\frac{(2x+1)u\cdot\pi}{16}\right)\cos\left(\frac{(2y+1)v\cdot\pi}{16}\right)\right) \tag{2}$$

Where u is the horizontal spatial frequency and v is the vertical spatial frequency; x and y are the coordinate of image.

$$\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u \neq 0 \end{cases} \tag{3}$$

From Eq. (1), we can force each GPU blocks calculate the 8 * 8 pixels (the element inside two summations). However, two summations implement the summation of matrix, here I use another parallel algorithm as follows:



i = 1 ***i = 2*** ***i = 3***

Figure 1, Parallel Sum

We always make a summation of the element with $2i$ distance of coordinate x and y with 3 elements around them and stored in the left-top most element. Finally, the value of summation of matrix is the value of the first element (on the left top corner).

Assume that the image contains m by n such image blocks, the time complexity of the algorithm at this step is $O\left(8^2 \times \log_2 8\right) = O(192)$, however the sequential algorithm is $O\left(m \cdot n \cdot 8^2 \cdot 8^2\right) = O\left(4096 \times m \cdot n\right)$; which is originally 6 for loops reduced to just 3 for loops.

Edge Handling

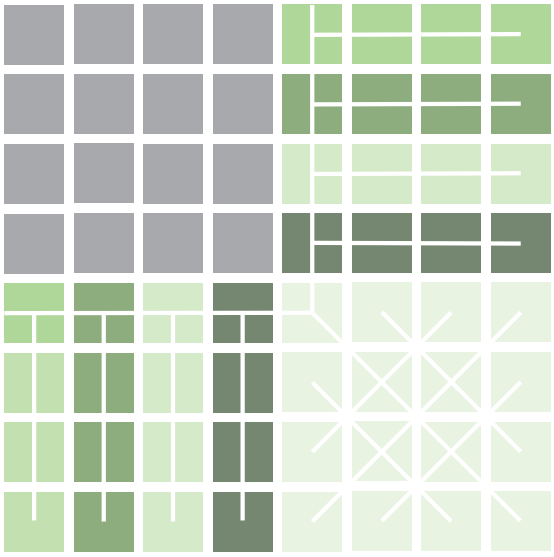


Figure 2, Edge Handling

If the last blocks are not filled up the 8 by 8 pixels, I use the algorithm that filled up the edge corners such that the Fourier transformations will not create noisy data. The algorithm patterns are shown above, the green pixel part shows the outside image pixels equals the value of edged pixels.

PROJECT FILE AND FUNCTION EXPLANATION

main.cu

`main.cu` stores the main function.

TIME TRACKING

In main function, I set the clock on the function of CPU part and GPU part.

JpegGPU.cu

`JpegGPU.cu` contains all CUDA function needed for parallel computing for Jpeg encoding and decoding implementation that explained above.

ERRORCUDA

`errorCUDA` is a simple error handling function for CUDA functions, the error position is the position with global counter.

SUMMATRIX

`SumMatrix` is the summation of all element for matrix in parallel.

EDGES

`Edges` is the simple edge handling method function, which will reduce the wavelet encoding noise.

FOURIERDCT

`FourierDCT` is the Fourier DCT that convert pixel matrix to spatial frequency matrix; the method equation is shown in Eq. (1); and then divided the quantization matrix.

ZIGZAGENCODE

`ZigzagEncode` is the encoding function that flattened the encoded array into large array.

ZEROS

`Zeros` function will set all numbers in matrix to be 0.

ZIGZAGDECODE

`ZigzagDecode` function convert large zigzag encoding array back to matrix (spatial frequencies matrix).

INVERSEFOURIERDCT

`InverseFourierDCT` function is to convert spatial frequency matrix back to pixel values as the JPEG compression result.

CUDA_MAIN

`CUDA_MAIN` function stores the all other device method to implement JPEG compression and de-compression in parallel.

JpegDimension.cpp

`JpegDimension.cpp` stores the dimension of block size, and the percentage of zigzag compression.

BLOCK

`BLOCK` is the image blocks of compression unit, in JPEG standard is 8 * 8, however I set this value here easy for future development if change the quantize matrix other than 8 * 8. Nevertheless, the algorithm can only handle the number which is power of 2 (such as, 8, 16, 32…) due to the `SumMatrix` function.

PERCENT

`PERCENT` is the percentage of length of zigzag traversal of image compression blocks.

JpegCPU.cpp

`JpegCPU.cpp` contains all CPU function that is the same function of implementation on single core CPU; the organize of this function package is the same as JpegGPU.cpp.

FUNCTIONS EXPLAIN

All functions are similar with methods in the file `JpegGPU.cu`. In order to separate the name of functions, I put a letter “S” in front of function names; if one function contains several methods together, I just put “_” to connect the function names.

JpegCommon.cu

`JpegCommon.cu` stores the common functions for both CPU and GPU code, including the file handling (with .pgm files), JPEG standard quantization matrix, zigzag index array, etc.

STOI

`STOI` is the function that only exist in C++11 (`stoi` function). This function convert the string to the number.

PGMOPEN

`pgmOpen` is the function that could import the image file with format .`pgm` to be a matrix and dimensions of image.

PGMWRITE

pgmWrite is to write the image data matrix back to the `.pgm` file

QMATRIX8

`Qmatrix8` is the JPEG standard quantization matrix for 8 * 8 pixel blocks.

SATURATION8BIT

`saturation8bit` handles the pixel values that will not greater than 255 or smaller than 0.

ZIGZAGENCODEARRAY

`zigzagEncodeArray` calculate the zigzag index array that to guide traverse the matrix into zigzag array in the image blocks.

USAGE

Prepare a `.pgm` image with string numbers coding (see the sample image file; however you cannot put #metadata in the second line of file); and name it to be `i.pgm`. Then compile the program with `nvcc`, and run it, it will automatically create `ocpu.pgm` and `ogpu.pgm`, which created by CPU and GPU algorithms respectively.

RESULT

Image Test

The test input image (120 * 128 pixels; large image could see pixels clear):

Figure 3, i.pgm

The output image with 30% of compression:

Figure 4, o.pgm

Speedup Test

Result

There is a speedup indicator that will print out in the terminal (single running time):

```
!! Breaking News !!
My GPU is 6269 times faster than CPU!

Details:
GPU: 26 ms
CPU: 163007 ms
```

The average speedup comparing paralleled CUDA algorithm with single core sequential CPU algorithm will be more than five thousand times.

Processor Details

CPU: Intel i7 – 6700HQ;

GPU: NVidia GeForce – 960M;