Parallel Optimization of JPEG Compression Algorithm

Team 6 PP Final Project Proposal

Kun Yao Lin
311551170
IOC, NYCU
Hsinchu, Taiwan
enk857@gmail.com

Pei Chi Chen
312553008
ILE, NYCU
Hsinchu, Taiwan
a0909205568@gmail.com

Shun Lin
312553027
ILE, NYCU
Hsinchu, Taiwan
linsapphire@gmail.com

1 INTRODUCTION

JPEG compression is one of the most widely used image compression algorithms, enabling efficient storage and transmission of digital images. However, with the rise of high-definition imagery and the demand for real-time processing, speeding up JPEG compression has become crucial. This project aims to optimize the performance of JPEG compression using parallel programming techniques, specifically focusing on parallelizing computationally expensive steps such as the Discrete Cosine Transform (DCT) and quantization. By leveraging multi-core processors through OpenMP and CUDA for GPU acceleration, we can divide the image into independent blocks and process them in parallel. This will enhance key operations like color space conversion, DCT, quantization, and frequency table building, significantly reducing processing time while maintaining image quality.

2 STATEMENT OF PROBLEM

In this project, we aim to optimize the JPEG compression algorithm using parallel programming techniques. JPEG compression consists of several computational steps, which we will break down and analyze to identify potential areas for parallelization. The key stages of the JPEG compression algorithm are as follows: **Color Space Conversion, Discrete Cosine Transform (DCT), Quantization, ZigZag Scanning, Run-Length Encoding (RLE), and Huffman Encoding**, as shown in the Figure 1 below.

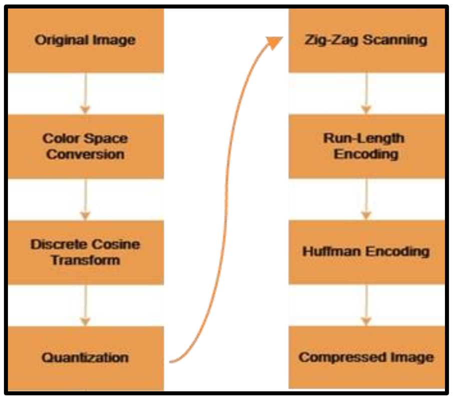

Figure 1: The flow chart of JPEG compression.

Below, we provide a detailed explanation of each step and analyze which parts can be parallelized for performance improvement.

2.1 Color Space Conversion

In JPEG compression, the input image is first converted from the RGB color space to the YCbCr color space. This transformation is applied to each pixel independently using the following formulas:

$$\mathrm{Y} = 0.299\mathrm{R} + 0.587\mathrm{G} + 0.114\mathrm{B}$$
$$Cb = 128 - 0.168736R - 0.331264G + 0.5B$$
$$Cr = 128 + 0.5R - 0.418688G - 0.081312B$$

2.2 Discrete Cosine Transform (DCT)

The DCT is applied to each 8×8 block of the Y, Cb, and Cr components to convert spatial image data into frequency data. The DCT for each block can be expressed as:

$$F(u,v) = \frac{1}{4}\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

Where $f(x,y)$ is the pixel value in the block, and $F(u,v)$ is the DCT coefficient.

2.3 Quantization

In the quantization stage, the DCT coefficients are divided by a quantization matrix to reduce their precision and discard less important data. The quantized value $Q(u,v)$ can be expressed as:

$$Q(u,v) = round\left(\frac{F(u,v)}{Q_{table}(u,v)}\right)$$

Where $Q_{table}(u,v)$ is the corresponding entry in the quantization table.

2.4 ZigZag Scanning

After quantization, the 8×8 block of DCT coefficients is converted into a one-dimensional array using ZigZag scanning. This step reorders the coefficients in a way that clusters low-frequency components (which are more likely to be significant) before high-frequency ones (which are often zeros). The ZigZag scan follows this pattern:

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

2.5 Run-Length Encoding (RLE)

Once the coefficients are reordered using ZigZag, Run-Length Encoding (RLE) is applied to compress sequences of consecutive zeros in the frequency data. This reduces the size of the data by representing multiple zeros with a single count and value. For example, the sequence $[15, 0, 0, 0, 6, 0, 0, 0]$ would be encoded as $[(15), (0,3), (6), (0,3)]$.

2.6 Huffman Encoding

Huffman encoding relies on building a frequency table for unique coefficients. Each coefficient is assigned a variable-length binary code based on its frequency, with more common symbols receiving shorter codes. The process involves analyzing run-length encoded data to build the frequency table and then constructing a binary tree for encoding

2.7 Why We Chose This Problem

The JPEG compression algorithm is applied to image processing and requires calculations on each pixel, resulting in relatively long execution times. However, a careful analysis of the algorithm's various steps reveals that many of them involve traversing all pixels in the image, with each pixel's calculations being independent. For example, steps like the Discrete Cosine Transform (DCT), and Quantization can all be executed using loops for traversal. This indicates that we can parallelize these steps to accelerate the execution of the algorithm. Therefore, the next chapter will discuss our plans for parallelizing these steps.

3 PROPOSED APPROACHES

This section outlines key steps in our approach, emphasizing their potential for parallelization. We analyze which processes can be performed concurrently to improve efficiency and explain the limitations for those requiring sequential execution.

3.1 Color Space Conversion

Since this process is performed independently for each pixel, it can be easily parallelized. By dividing the image into smaller chunks, we can assign multiple threads to handle the conversion of different regions of the image simultaneously.

3.2 Discrete Cosine Transform (DCT)

Since the DCT is computed on a block-by-block basis, each 8×8 block can be processed independently, making it highly parallelizable. Threads can be assigned to perform DCT on multiple blocks concurrently, reducing overall computation time.

3.3 Quantization

Like the DCT, quantization is performed independently for each 8×8 block, making it amenable to parallelization. Multiple threads can handle quantization for different blocks in parallel.

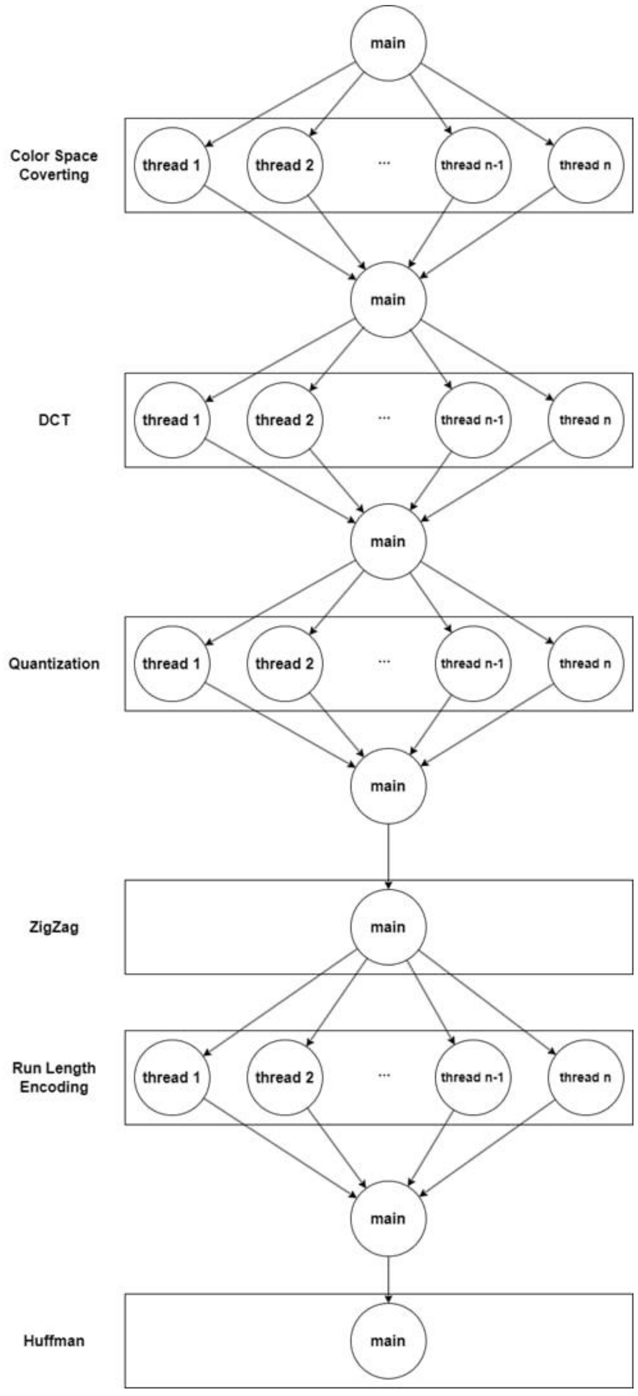

Figure 2: Parallel Strategies and Flow Chart of JPEG compression Parallelism.

3.4 ZigZag Scanning

ZigZag scanning is the process of converting the DCT coefficients in an 8×8 block into a one-dimensional sequence. Since this scanning process must be performed in a specific order, it cannot be parallelized. The data within each block must be sequentially converted according to a fixed path, meaning each block must be scanned step by step, and multiple data points cannot be processed simultaneously.

3.5 Run-Length Encoding (RLE)

After ZigZag scanning, the scanned results undergo RLE encoding. The core of RLE is compressing consecutive occurrences of the same number into a value and count pair (e.g., if the number 0 appears 5 times consecutively, it is recorded as (0, 5)). This encoding process is performed independently for each block, so multiple threads can handle RLE encoding for different blocks in parallel, improving efficiency.

3.6 Huffman Encoding

Huffman encoding is used to further compress the data after RLE. By constructing a Huffman tree, shorter codes are assigned to frequent symbols, while longer codes are assigned to less frequent symbols. Huffman encoding is typically performed at the whole image level, so unlike previous steps, its parallelization is limited.

4 Language Selection

Our programming language is C++. Since C++ is a compiled language, it is converted to machine code before execution, making it faster and more efficient. C++ also provides ways to manage memory, offering a certain degree of flexibility and design space. Additionally, C++'s Standard Template Library (STL) provides various APIs to support parallelization, including std::future, std::atomic, and others. Moreover, other frameworks such as OpenMP, CUDA, and OpenCL can be used to further support parallelization..

5 Related Work

JPEG compression is widely used for reducing image file size while maintaining acceptable quality. The process consists of several stages, including color space conversion, block-based transformations, quantization, and entropy encoding. Several studies have explored methods to accelerate these stages through parallel processing on multi-core CPUs and GPUs, focusing on improving efficiency for real-time applications.

D. Liu and X. Y. Fan outlines a structured approach to parallelizing JPEG compression in [1], primarily focusing on multi-core CPUs. They break down the JPEG encoding process into independent tasks, emphasizing two types of parallelism: data parallelism, where different 8x8 pixel blocks are processed in parallel, and pipeline parallelism, where stages like DCT and quantization operate concurrently on different parts of the image. This design takes advantage of the block-based nature of JPEG, with dynamic task scheduling for efficient load balancing across multiple cores. Liu and Fan demonstrate significant speedups, achieving 5x to 7x performance gains on multi-core processors.

On the other hand, [2] addresses the specific stage of color space conversion, a crucial preprocessing step in JPEG compression. Most images are stored in RGB format but need to be converted to a more efficient color space (such as YCbCr) for compression. The study emphasizes the computational intensity of this transformation, particularly for high-resolution images, and proposes using GPUs for acceleration. The authors take advantage of the massive parallelism of GPUs, where each pixel's RGB values are independently converted to YCbCr in parallel, significantly speeding up the process.

6 Statement of Expected Result

Parallelizing key tasks in the image processing pipeline is expected to yield significant performance improvements, particularly for computationally intensive operations. This section outlines the anticipated results, focusing on both the advantages of parallel execution and the constraints posed by sequential processes.

6.1 Parallelization Efficiency

The operations that can be parallelized, such as color space conversion, DCT, quantization, and RLE, are expected to show significant reductions in computation time. Since these steps operate on independent units (pixels or 8×8 blocks), assigning multiple threads to handle different parts of the image simultaneously should lead to substantial speed improvements.

6.2 Impact on Large vs. Small Images

Larger images are expected to benefit more from parallelization compared to smaller images. With more data to process, parallel threads can work more effectively on large images, distributing the workload more evenly and efficiently across the available resources. In contrast, smaller images may not see as much speedup due to the limited amount of data and less efficient use of parallel resources.

6.3 Sequential Steps and Limitations

ZigZag scanning and Huffman encoding, due to their inherently sequential nature, are expected to offer little to no improvement from parallelization. ZigZag scanning requires a strict data order for each 8×8 block, while Huffman encoding involves a global frequency analysis that must be processed as a whole, limiting the potential for concurrency in these steps.

6.4 Overall System Performance

Even though some steps cannot be parallelized, the majority of the computationally intensive tasks are parallelizable. Therefore, the overall system performance should still benefit significantly from parallelization, with the greatest gains in image processing tasks where block-based operations dominate, such as the DCT and quantization steps.

7 Timetable

Plan	Nov				Dec		
	W1	W2	W3	W4	W1	W2	W3
Serial Program of JPEG Compression	▨	▨					
Pthread and OpenMP Parallelism		▨	▨				
CUDA Parallelism			▨	▨	▨		
Experiment & Evaluation				▨	▨	▨	
Final Report and Presentation Preparation						▨	▨

REFERENCES

[1] D. Liu and X. Y. Fan, "Parallel program design for JPEG compression encoding," 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery, Chongqing, China, 2012, pp. 2502-2506, doi: 10.1109/FSKD.2012.6234221..

[2] SHEN, Meng; RUAN, Shengdong. Accelerating Color Space Conversion using Graphic Processing Units.