Parallel Cosine Discrete Transform Compression on JPEG Files

Lingxi Zhang
Carnegie Mellon University
Pittsburgh, USA
lingxiz@andrew.cmu.edu

Figure 1: Character Ranni from the Game Elden Ring

ABSTRACT

This paper explores the intrinsics of discrete cosine transform(DCT) and builds a parallel DCT visual codec system. Our system maintains a good balance between the compression ratio and quality preservation. Specifically, a compressed image of 50 % compressing rate is hardly distinguishable by human eyes. It also executes almost perfectly parallel as the speed up is close to linear with multi-threading in OpenMP mechanics.

KEYWORDS

Discrete Cosine Transform, Quantification, compression, matrix transform

ACM Reference Format:
Lingxi Zhang. 2022. Parallel Cosine Discrete Transform Compression on JPEG Files. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Since low-frequency content is predominant in images of the real world, and the human visual system is much less sensitive to high frequency sources of error, we can exploit the characteristics of human perception to build efficient image processing systems. In other words, we can simply identify and remove some of the high frequency components since humans are insensitive to these contents to maintain image quality and compress the image at the same time.

A good method of such conversion is provided by the discrete cosine transform (DCT) which helps separate the image into parts of differing frequencies, transforming scalar domain to frequency domain. Then we can remove high frequencies to compress the image. Finally we can use inverse DCT to transform the frequency domain to the original scalar domain to get the original image when we need, though with some loss of quality. One of the important benefits of such a compression process is reducing the burden of transmitting images limited by network latency and bandwidth.

2 RELATED WORKS

The origin of discrete cosine transform(DCT) can be traced all the way back to 1974 (N.Ahmed, et al). The paper provides a through and complex mathematical foundation for discrete cosine transform. It also points out that since there has been an growing interest with respect to using a class of orthogonal transforms in the general area of digital signal processing, and DCT stands out to be the optimal solution.

However, this paper did not address the implementation details of DCT that are specific to the image compression, such as quantization, methods to manipulate standard 8 bits pixels, and algorithms that compute DCT efficiently.

In the paper "Image Compression and the Discrete Cosine Transform" by Ken Cabben and Peter Gent, the implementation details are addressed as previously mentioned. It converts DCT transformation into a series of matrix multiplication and presents the experimental quantization matrix based on human perception. We will reference these details in this paper as well. Nevertheless, this paper does not discuss the serialization and deserialization protocols of compression. There is also no benchmark, parallelization, and analysis of the relationship between the quality factor and compression ratio.

Hence, in our paper, we are aiming to build a fast, robust, and parallel DCT system that addresses the problems aforementioned and conduct a through analysis of the relationship and trade-offs among different factors.

3 METHODOLOGY

We use the Cool Image (CImg), a small and open-source C++ library for image processing in our project. It provides very simple APIs and library linking for extracting and manipulating the image pixels, at the cost of slightly longer compile times. For matrix manipulation we write our own matrix library for better debugging purpose.

Although DCT also works for colored images, the focus of this paper will be solely on grayscale JPEG images. Colored images can be reconstructed from the grayscale ones, but we need to know the ratio of each color in the pixel, which hurts the compression rate.

3.1 Blocking

We decompose the image into $B \times B$ blocks of pixels. For image dimensions that are not a multiple of B, we simply pad with zeros on the boundary. Ordered from left to right, top to bottom, the blocks get their block ids and are assigned to different threads in OpenMP. For instance, when the image dimension is 40 x 40, block size is 8 x 8, and there are a total of 3 threads,

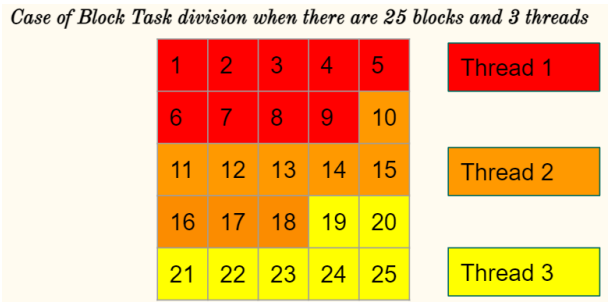


Figure 2: Blocking schemes

3.2 Matrix Form of DCT

The original DCT equation is

$$D_{ij} = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{B-1} \sum_{y=0}^{B-1} p_{xy} \cos\left(\frac{(2x+1)i\pi}{2B}\right) \cos\left(\frac{(2y+1)j\pi}{2B}\right) \tag{1}$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u > 0 \end{cases} \tag{2}$$

where D_{ij} is the (i, j) entry of the DCT matrix $\mathbf{D}$ and p_{xy} is the image pixel value at (x, y) inside the block.

Observe that the summation part from equation (1) looks a lot like matrix multiplication. Actually, we can convert equation (1) to a matrix form:

$$T_{ij} = \begin{cases} \frac{1}{\sqrt{B}} & i = 0 \\ \sqrt{\frac{2}{B}} \cos\left(\frac{(2j+1)i\pi}{2B}\right) & i > 0 \end{cases} \tag{3}$$

Suppose the original 8 bit grayscale matrix are composed of pixel values in $[0, 255]$, since DCT is designed to work on a symmetrical domain, we want to subtract 128 from each entry first.

$$M_{ij} = P_{ij} - 128, (0 \le i, j < B) \tag{4}$$

We are now ready to perform the Discrete Cosine Transform, which is accomplished by two matrix multiplication,

$$\mathbf{D} = \mathbf{TMT}^T \tag{5}$$

Notes that T is actually a orthogonal matrix, which has a nice property that

$$\mathbf{T}^{-1} = \mathbf{T}^T$$

Therefore we have another formula that we can used in later decompression as well,

$$\mathbf{M} = \mathbf{T}^T\mathbf{DT} \tag{6}$$

3.3 Quantization

Quantization is the key step to actually achieve compression, basically, the result compressed matrix $\mathbf{C}$ is computed as

$$C_{ij} = round\left(\frac{D_{ij}}{Q_{ij}}\right) \tag{7}$$

A standard quality 50 quantization matrix, generated by scientific experiment on human perception is Notice that the entry is gen-

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 3: Standard 8 x 8 Quantization Matrix

erally increasing from the low frequency top left corner to high frequency bottom right corner. That is because this entry by entry division and rounding will penalize high frequency count more since human perceptions are less sensitive to high frequency, and in general low frequency are more prevalent in majority of pictures.

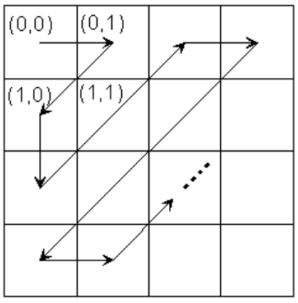
How do we scale the quantization matrix with different quality factor then? In practice there are two separate equations for quality factor above and below 50,

$$\mathbf{Q}_i = \begin{cases} \frac{100-i}{50}\mathbf{Q}_{50} & i > 50 \\ \frac{50}{i}\mathbf{Q}_{50} & i < 50 \end{cases} \tag{8}$$

We can see that the higher the quality is, the smaller the quantization matrix entry is, so the result matrix **C** will have more non-zero entries that results in lower compression rate.

3.4 Serialization

We access entries diagonally in a zigzag manner, from the top left corner to the right bottom corner. We store the results into a vector and can remove all the trailing zeros. This makes sense because by accessing diagonally we can have the most amount of trailing zeros and higher compression rate, since high frequency counts are more likely to be zero.



$$\begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 1 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 1 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For instance, in our result vector it will be
10, 4, 3, -7, 9, 2, 5, 1, -5, -3, -2, -5, 1, 2, 1, 0, 1, -2, 0, 1 ,0, 0, 0, 0, -1, -1, a total of 26 entries. This achieves an ideal compression rate of $\frac{26}{64} = 40.63\%$

Finally, our program writes the $26 + 1(size)$ bytes into disk(file ending with .cps) for each block.

3.5 Decompression

We use the same access pattern to decompress the .cps file into the reconstructed result matrix. As we mentioned earlier, since the DCT transform matrix is orthogonal, inverting the process is trivial,

$$D_{ij} = C_{ij} \times Q_{ij} \tag{9}$$

$$\mathbf{M} = \mathbf{T}^T\mathbf{D}\mathbf{T} \tag{10}$$

$$P_{ij} = M_{ij} + 128 \tag{11}$$

3.6 Time Complexity

Suppose the block size is B, and the image size is $M \times N$ then there are a total of $\frac{MN}{B^2}$ blocks. Time per block is just the cost of matrix multiplication, which is $O(B^3)$ instead of $O(B^4)$ originally, where there is no matrix form optimization. Although this can be further improved by faster matrix multiplication algorithms, they tend to have a very large constant and block size is generally small. Therefore, the overall time complexity is $\frac{MN}{B^2} \times B^3 = O(BMN)$, for $B = 8$ in our case this is not bad.

The extra memory overhead is also pretty small, since there is only temporal matrix we generated , which is $O(B^2)$

4 RESULTS

We benchmarked our results based on the 1024x768 image of Ranni with different quality factors, ranging from 5 to 95 with a step size of 5.

4.1 Image Showcase

Figure 4: Quality factor 5 and 10

Figure 5: Quality factor 50 and 90

Figure 6: Original Grayscale Image

We can see that it is hard to perceive the difference between compressed image with quality 50 with the original grayscale image, and the main artifacts that are more perceivable only show up with very low quality factor. (But low quality images sometimes have their ambient beauty and artistic values, isn't it?)

4.2 Compression Rate

We measured the ideal compression rate, which is the average percentage of non-trailing zero entries in each 8×8 block, the real compression rate, which is the compressed file size divided by the raw image file size. Here is a summary of the result.

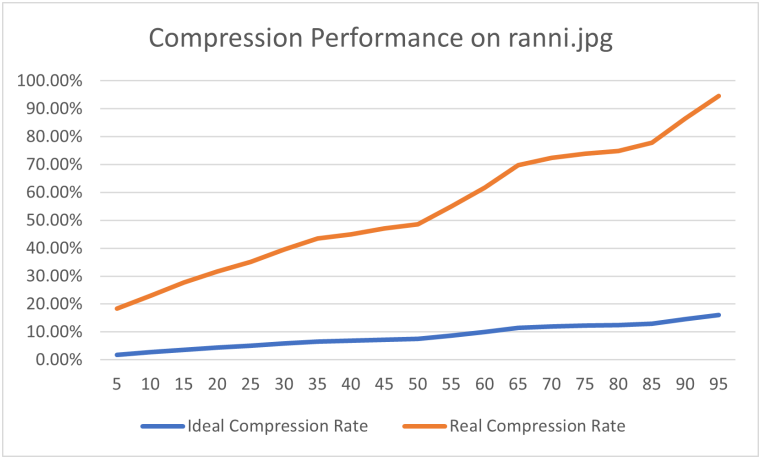


Figure 7: Compression Rate vs Quality Factor

We can see that both ideal and real compression rate scales almost linearly with the quality factor. Even with the huge gap between the ideal and real compression rate due to serialization protocols and binary file overhead, the real compression rate is still pretty good for practical purpose.

4.3 Parallelism

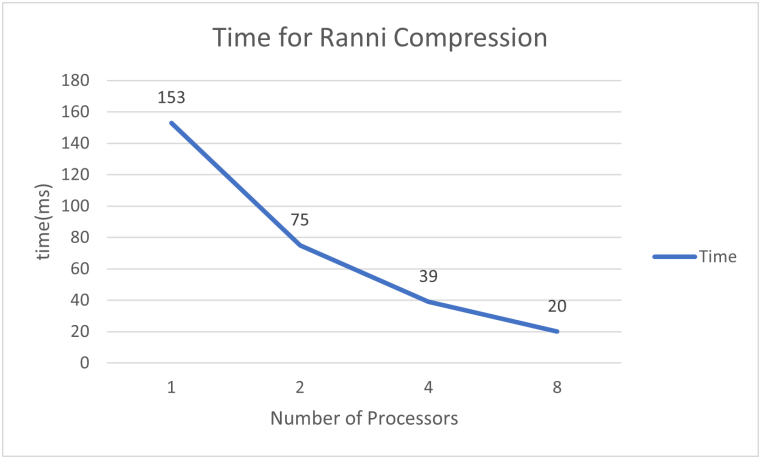


Figure 8: Time vs Number of Threads

DCT, compression, and inverse DCT is applied on each block separately amd there is no need for communication between threads. We can see OpenMP is the best choice of parallelism since it is based on shared memory model and avoids complexity of pthreads series. There are also no worries of cache coherence problems because we can copy the image data to local matrix first. As a result, the entire computation is almost perfectly parallel. As we can see from the graph, the speedup scales almost perfectly linearly with the number of processors due to the highly parallel nature of our computation.

5 REMARK AND FUTURE WORKS

Our system maintains a good balance between the compression ratio and quality preservation. It also executes almost perfectly parallel as the speed up is close to linear with multi-threading in OpenMP mechanics.

Our system is also convenient for further expansion with new settings. We can increase the block size to have higher precision for the cosine transform at the cost of higher processing time, as we discussed in the time complexity section. We can change the DCT matrix easily with different block size by just reapplying the formula with new parameter. The quantization matrix, on the other hand, is not quite easily expandable for different block size since the 8x8 block is conducted by previous scientific experiments of human visual system. Alternating mathematical solutions exist but they do not perform as good as the experimental one.

We can also expand our system to 16 bit HDR image with just more nuance considerations in the serialization part, as there is no much difference in the transformation part. **The result frequency is not necessary within 16 bit since the low frequency count can dominate very much in some cases**. A robust and well-designed serialization protocol is both important in increasing compressing ability and ensuring compressing correctness.

Additionally, originally we thought that since colored images have to be reconstructed from the grayscale ones, we need to know the ratio of each color in the pixel, which hurts the compression rate. That is why we only discuss grayscale images for simplicity in this paper. However, in the very end of this research we found out that actually each color channel can be processed separately and recombined together in the end, instead of converting back and forth with grayscale.

ACKNOWLEDGMENTS

Thanks Professor Oscar Dadfar for his guidance in accomplishing this project and lectures about discrete cosine transform in visual computing system class and thanks my parents and friends for their emotional support during this period.

REFERENCES

.

[1] N. Ahmed, T. Natarajan and K. R. Rao, "Discrete Cosine Transform," in IEEE Transactions on Computers, vol. C-23, no. 1, pp. 90-93, Jan. 1974, doi: 10.1109/T-C.1974.223784.

[2] Ken Cabeen and Peter Gent, Image Compression and the Discrete Cosine Transform

[3] Visual Computing System lectures about JPEG and DCT Compression from Carnegie Mellon University in Spring 2022