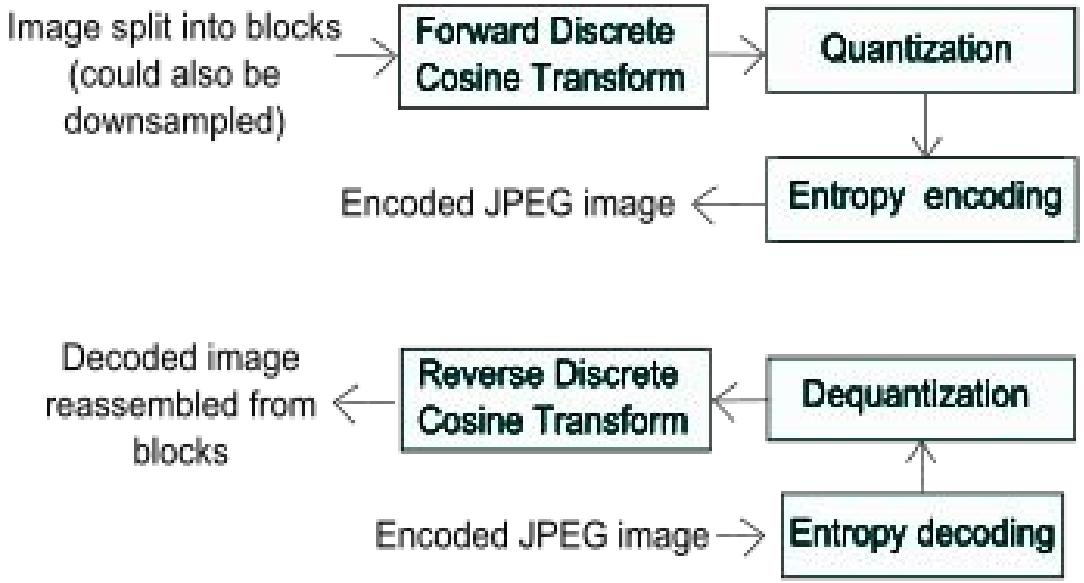
Parallel Image Compression

By -
Medhini G N(13IT118)
Ohileshwar Itagi(13IT251)
Shyam B S(13IT144)

Abstract

- Image compression is significant because of the **large image transfer** happening over the internet and advancements in photography
- Compression is useful because it helps **reduce the consumption of expensive resources** such as hard disk space or transmission bandwidth
- The problem is to compress image files whilst **maintaining the quality of the image**
- The JPEG algorithm takes advantage of the fact that humans can’t see colors at high frequencies. These **high frequencies are eliminated** during the compression.
- Our project uses techniques of parallelization to compress multiple images simultaneously

Flow Diagram



Methodology

1. Reading the image
    a. The pixel data is read from an image and is saved in 8 x 8 matrices
    b. As the pixel data for each block is independent of the other's, the data was read in parallel - SIMD
    c. The above is an application of **data parallelism**
2. The image compression algorithm
    a. Zero padding of the image
    b. Extracting image data (RGB Values)
    c. Parallel Discrete Cosine Transformation Algorithm
    d. Parallel Quantization of all the blocks
    e. Parallel Entropy coding

a. Zero Padding
   i. If the image cannot be divided into 8-by-8 blocks, then we add in empty pixels around the edges, essentially zero-padding the image
b. Extracting image data
   i. The algorithm works for grayscale images
   ii. The RGB data is read and the average is stored in another matrix
c. Parallel Discrete Cosine Transformation
   i. Conversion of matrix values using DCT function
   ii. Complexity is $O(n^2xm^2)$
   iii. DCT is applied parallel to all the 8 x 8 blocks of the image

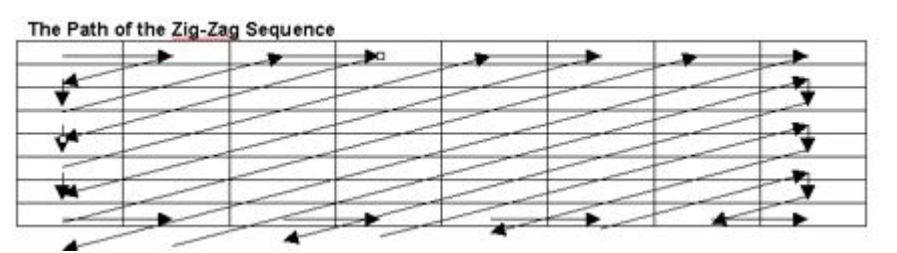
$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)\, C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(x) = \frac{1}{\sqrt{2}} \text{ if } x \text{ is } 0, \text{ else } 1 \text{ if } x > 0$$

d. Parallel Quantisation
   i. Process of reducing the number of bits needed to store an integer value by reducing the precision of the integer
   ii. Multiplying DCT matrix with separate quantization matrix removes irrelevant frequencies

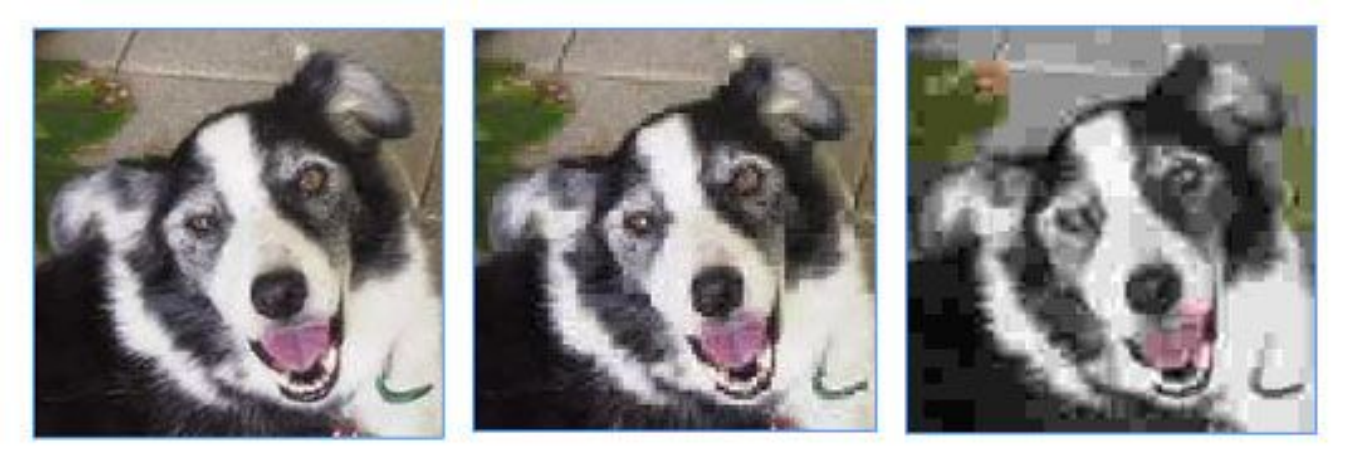
$$\text{Quantized Value}(i, j) = \frac{\text{DCT}(i, j)}{\text{Quantum}(i, j)} \text{ Rounded to the nearest integer}$$

e. Parallel Entropy Encoding
   i. Convert the DC coefficient to a relative value
   ii. Reorder the DCT block in a zig-zag sequence
   iii. Entropy Encoding : Run length encoding and Huffman encoding



Steps After Compression

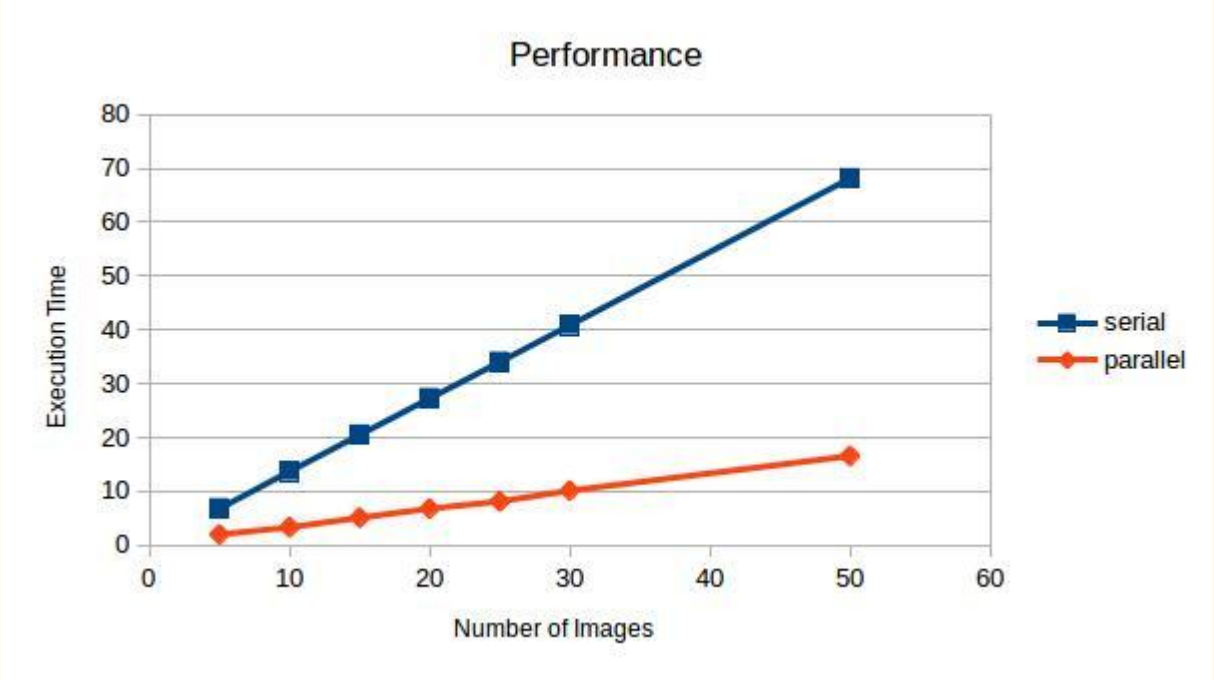
1. Applying inverse quantisation
2. Applying inverse discrete cosine transformation
3. Retrieving the image from the pixel data



Experimental Setup

1. OpenCV was used for reading and writing of image data
2. OpenMP was used to parallelize the program
3. Data parallelism technique was used to parallelize the different blocks
4. Parallellizing the “for” loops : #pragma omp parallel for
5. Different scheduling techniques such as Static, Dynamic, Runtime and Guided were tried
6. The program was run for “n” images and the serial v/s parallel results for varying n values were obtained
7. The time taken was calculated using “get time of the day” and the values were written to a CSV file where a graph was plotted

Comparison - Serial vs Parallel

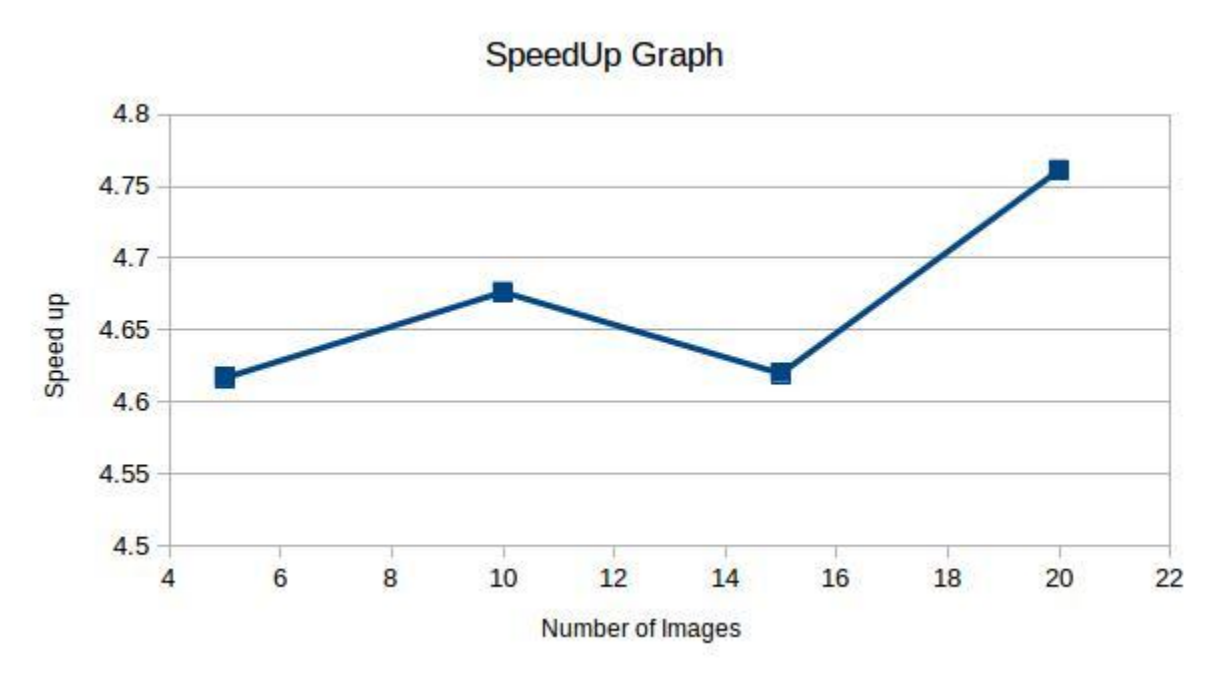


Speed up graph

We considered different sets of images of sizes 5, 10, 15, 20 and plotted the Speed up graph. Speed Up = (Time for serial)/(Time for Parallel)

Number of Images	Parallel Execution Time				Serial Execution Time			
5	1.5041	1.4431	1.5051	1.5047	6.8409	6.8861	6.8776	6.8985
10	2.8885	2.9143	2.9360	2.981	13.6876	13.6062	13.7483	13.766
15	4.3617	4.3529	4.4049	4.5325	20.3858	20.37	20.399	20.393
20	5.8147	5.8225	5.8446	5.8146	27.8359	27.6329	27.9314	27.519

Speed up was calculated as (Average $T_{Parallel}$/Average T_{Serial})



Average Speed up = $T_{serial}/T_{parallel}$ = 4.7

Conclusion

- Varying the block size helped us improve performance. The block size is 8 x 8 because
  i. Smaller block sizes make the compression more arduous
  ii. Larger block sizes would result in data loss due to color gradients
- Using runtime scheduling gave the best performance
- The speedup increases as we increase the number of images
- Through this project we were able to apply the concepts of parallel computing to a real world scenario. It helped us gain practical knowledge in the field.

References

1. JPEG image compression (Matt Marcus)
2. Stanford Image Compression techniques
3. Basic image compression and introduction to JPEG standard(Pao Yen Lin)

Individual Contribution

1. Medhini G Narasimhan - 13IT118
    a. Extraction of 8 x 8 blocks from the n x m image
    b. Applying Discrete Cosine Transformation algorithm on each block (both serial and parallel)
    c. Applying Inverse Discrete Cosine Transformation algorithm on each block (both serial and parallel)
    d. Updating the n x m matrix with the new 8 x 8 block values
2. Ohileshwar Itagi - 13IT251
    a. Parallel reading of pixel data from the image and calculation of grayscale values
    b. Huffman encoding
    c. Retrieval of image from the pixel data
3. Shyam S B - 13IT144
    a. Parallel quantization of 8 x 8 blocks
    b. Encoding quantized data using Run Length Encoding(RLE)
    c. Decoding the data using Reverse RLE
    d. Parallel dequantization of 8 x 8 blocks