KOLEJ UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

Assignment

BMCS3003 DISTRIBUTED SYSTEMS AND PARALLEL COMPUTING

2021/2022

Student's name/ ID Number	:	RYAN KHO YUEN THIAN / 2204097
Student's name/ ID Number	:	ONG WENG KAI / 2203309
Student's name/ ID Number	:	YONG ZEE LIN / 2203770
Programme	:	Bachelor of Computer Science in Data Science
Tutorial Group	:	2
Date of Submission to Tutor	:	15/9/2024

Near Lossless Image Compression using Discrete Cosine Transform (DCT)

CHAPTER 1: ABSTRACT

This study presents a comprehensive approach to near lossless image compression using the Discrete Cosine Transform (DCT) and advanced parallel computing techniques to optimize both performance and image quality. Traditional image compression methods often struggle to balance compression efficiency with image fidelity. To address this, our research leverages OpenMP, CUDA and MPI parallel platforms to accelerate DCT and its inverse (IDCT) operations, significantly speeding up the compression and decompression processes. Our approach involved compressing and decompressing an image (small to large) to simulate the transmission of image data over a network. We used several techniques, which are DCT (transformation), Quantization (Truncation of DCT coefficient to 1 decimal place), Zig-zag scanning, Run-Length Encoding (RLE), LZMA2 (Lempel–Ziv–Markov chain algorithm), Run-Length Decoding, Reverse Zig-zag scanning and IDCT (inverse transformation). By analyzing the effects of these parallel platforms, we assess the benefits of accelerated compression and decompression workflows. Key variables include image fidelity (measured through visual and quantitative metrics), compression ratio and computational efficiency (evaluated based on processing speed and resource usage). The findings highlighted the significant speed improvements achieved through parallel computing. The CUDA implementation was the fastest, followed by MPI and OpenMP. This indicates that parallel computing can be a robust solution for compression and decompression and optimize compression and decompression techniques in various applications. This includes optimizing real-time image processing and streaming applications to ensure swift and efficient compression and decompression. This allows immediate feedback on image quality and integrity, while also verifying the correct transmission of compressed data in real-time across networks or between systems. Additionally, all 4 implementations produce approximately the same compression ratios, indicating consistent results. Lastly, this study revealed that each parallel platform demands a distinct approach to problem-solving. OpenMP focuses on managing threads and data dependencies within shared memory. CUDA optimizes GPU performance through efficient memory access and kernel configurations. MPI distributes tasks across multiple processes in a distributed memory setup, making communication between processes crucial to minimizing overhead. These differences emphasize the importance of adapting algorithms to leverage the unique strengths of each platform for maximum performance gains.

Keywords: compression, decompression, DCT, IDCT, OpenMP, CUDA, MPI, parallel, serial, time

CHAPTER 2: RESULTS (PERFORMANCE MEASUREMENTS)

To fully evaluate the performance of the three parallel approaches (OpenMP, CUDA and MPI), we used 3 .png images of increasing sizes & resolutions (771 KB, 801 KB, 2 MB). You may view the images under the following subheadings or from the google drive and github links below. They are named testimg.png (771 KB), dark.png (801 KB) and dnb_land_ocean_ice.2012.13500x13500.A1-0000.png (2 MB). Before any processing is performed, the input image is converted into .bmp first so that the image is initially uncompressed. After decompression, the image will also be .bmp.

Google drive & Github links to image files:
https://drive.google.com/drive/folders/1gE15xWWPXLrdZ5PVFF7f5gj3ualeTEp9?usp=drive_link
https://github.com/RyankTheDS/Near-Lossless-Image-Compression-using-DCT/tree/main/images

The following pages show the results for each of the 3 images. For each image result under each implementation, we show the sizes of the original image and the returned image (after compression and decompression), the Mean Squared Error (MSE) and Structured Similarity Index (SSIM) between the original and decompressed images, the compression ratios achieved and the time taken for the compression and decompression pipeline to complete. Additionally, we compare the images produced by different implementations and compute the speedup gained from using parallel computing methods.

After conversion to .bmp:

- testimg.png increases in size to around 2 MB (Dimensions: 889 x 894)
- dark.png increases in size to around 3 MB (Dimensions: 1420 x 763)

- dnb_land_ocean_ice.2012.13500x13500.A1-0000.png increases in size to around 66 MB (Dimensions: 6750 x 3375)

2.1 Result for testimg.png

The following content shows the results for each implementation. Their results are evaluated under 2.1.5 (Overall Results). Note: the term “returned image” refers to the decompressed image.

2.1.1 Serial Implementation

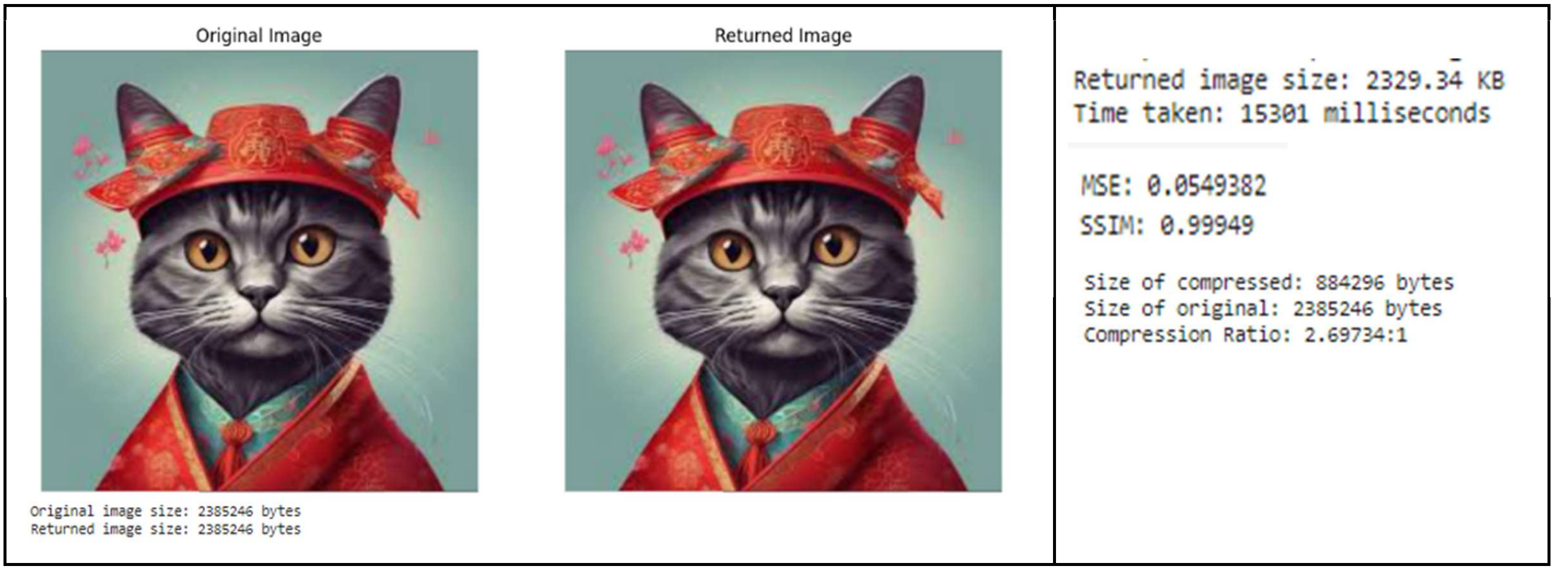


Figure 2.1: Result for Serial Implementation

2.1.2 OpenMP Implementation

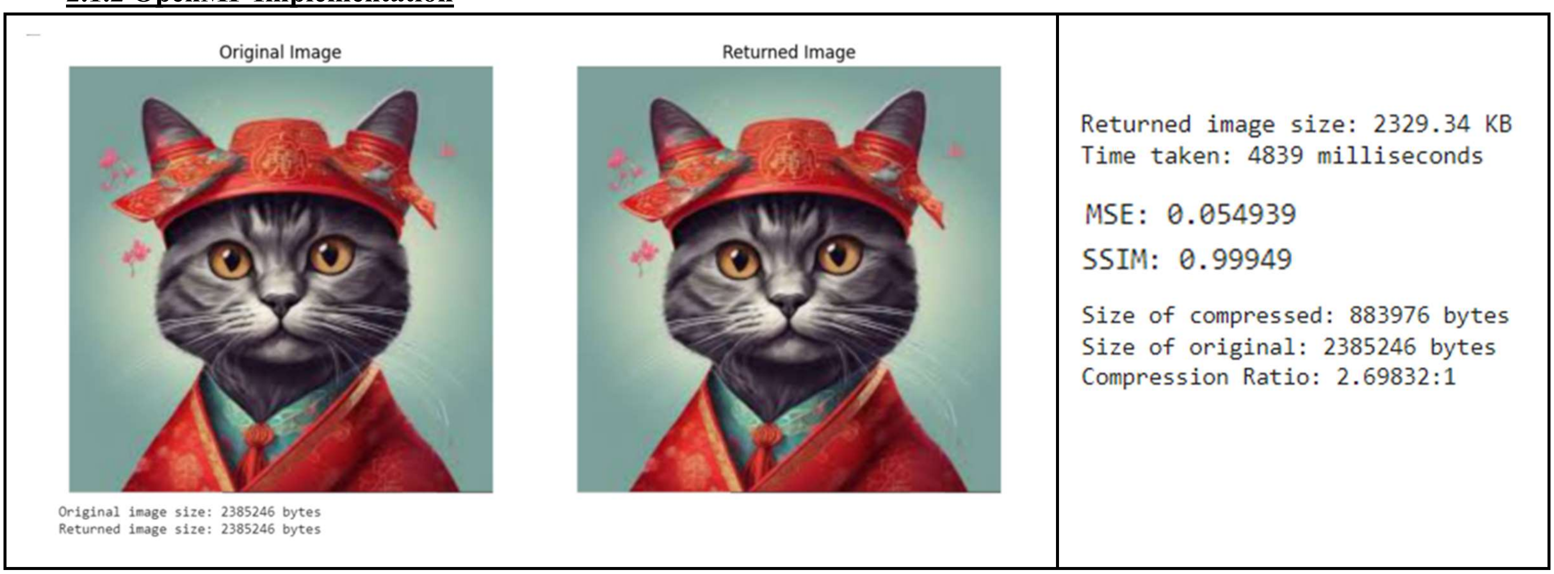


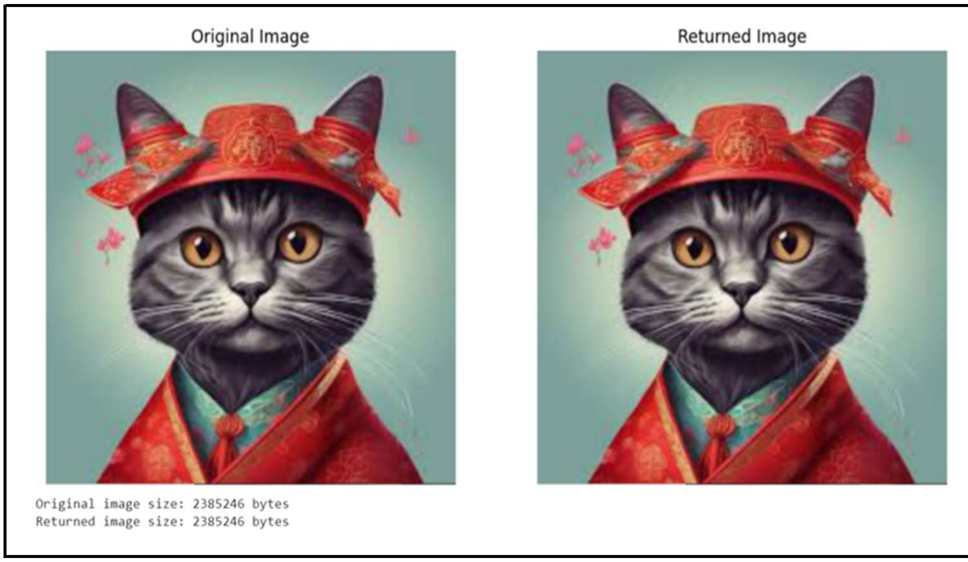
Figure 2.2: Result for OpenMP Implementation

2.1.3 MPI Implementation (with 2 processes)


```
The image will be divided into blocks of 1204224 bytes each
Number of rows: 896
Rows per process: 448
Columns per process: 896
Returned image size: 2329.34 KB
Time taken: 3718 milliseconds
MSE: 0.054939
SSIM: 0.99949
Size of compressed: 895640 bytes
Size of original: 2385246 bytes
Compression Ratio: 2.66317:1
```

Figure 2.3: Result for MPI Implementation

2.1.4 CUDA Implementation




```
Returned image size: 2329.34 KB
GPU Time: 933.579 milliseconds
Total Time (including I/O): 1356.49 milliseconds
MSE: 0.062662
SSIM: 0.999426
Size of compressed: 936156 bytes
Size of original: 2385246 bytes
Compression Ratio: 2.54792:1
```

Figure 2.4: Result for CUDA Implementation

```
Returned image size: 2329.34 KB
GPU Time: 939 milliseconds
Total Time (including I/O): 1217.26 milliseconds
==1959== Profiling application: ./dct_idct_zigzagrle testimg.bmp
==1959== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   44.59%  3.1855ms         7  455.07us     672ns  547.79us  [CUDA memcpy HtoD]
                   38.65%  2.7614ms         6  460.23us  395.80us  531.47us  [CUDA memcpy DtoH]
                    9.64%  688.88us         3  229.63us  229.27us  229.95us  idct8x8Kernel(float*, int, int, int)
                    7.12%  508.37us         3  169.46us  169.02us  170.01us  dct8x8Kernel(float*, int, int, int)
      API calls:   93.81%  179.69ms         1  179.69ms  179.69ms  179.69ms  cudaMalloc
                    4.36%  8.3527ms        12  696.06us  597.03us  760.91us  cudaMemcpy
                    0.77%  1.4741ms         6  245.69us  198.03us  277.35us  cudaDeviceSynchronize
                    0.72%  1.3793ms         1  1.3793ms  1.3793ms  1.3793ms  cudaMemcpyToSymbol
                    0.13%  241.48us         1  241.48us  241.48us  241.48us  cudaFree
                    0.10%  189.10us         6  31.517us  18.485us  44.213us  cudaLaunchKernel
                    0.08%  154.01us       114  1.3500us     131ns  59.603us  cuDeviceGetAttribute
                    0.01%  23.090us         2  11.545us  9.3010us  13.789us  cudaEventRecord
                    0.01%  11.463us         2  5.7310us  1.2210us  10.242us  cudaEventCreate
                    0.01%  11.319us         1  11.319us  11.319us  11.319us  cuDeviceGetName
                    0.00%  7.3560us         1  7.3560us  7.3560us  7.3560us  cuDeviceGetPCIBusId
                    0.00%  5.8810us         1  5.8810us  5.8810us  5.8810us  cudaEventSynchronize
                    0.00%  4.8120us         1  4.8120us  4.8120us  4.8120us  cuDeviceTotalMem
                    0.00%  3.4540us         1  3.4540us  3.4540us  3.4540us  cudaEventElapsedTime
                    0.00%  2.4190us         2  1.2090us     742ns  1.6770us  cudaEventDestroy
                    0.00%  1.9790us         3     659ns     214ns  1.4980us  cuDeviceGetCount
                    0.00%  1.0780us         2     539ns     193ns     885ns  cuDeviceGet
                    0.00%     371ns         1     371ns     371ns     371ns  cuModuleGetLoadingMode
                    0.00%     258ns         1     258ns     258ns     258ns  cuDeviceGetUuid
```

Figure 2.5: Result of nvprof command

2.1.5 Overall Results

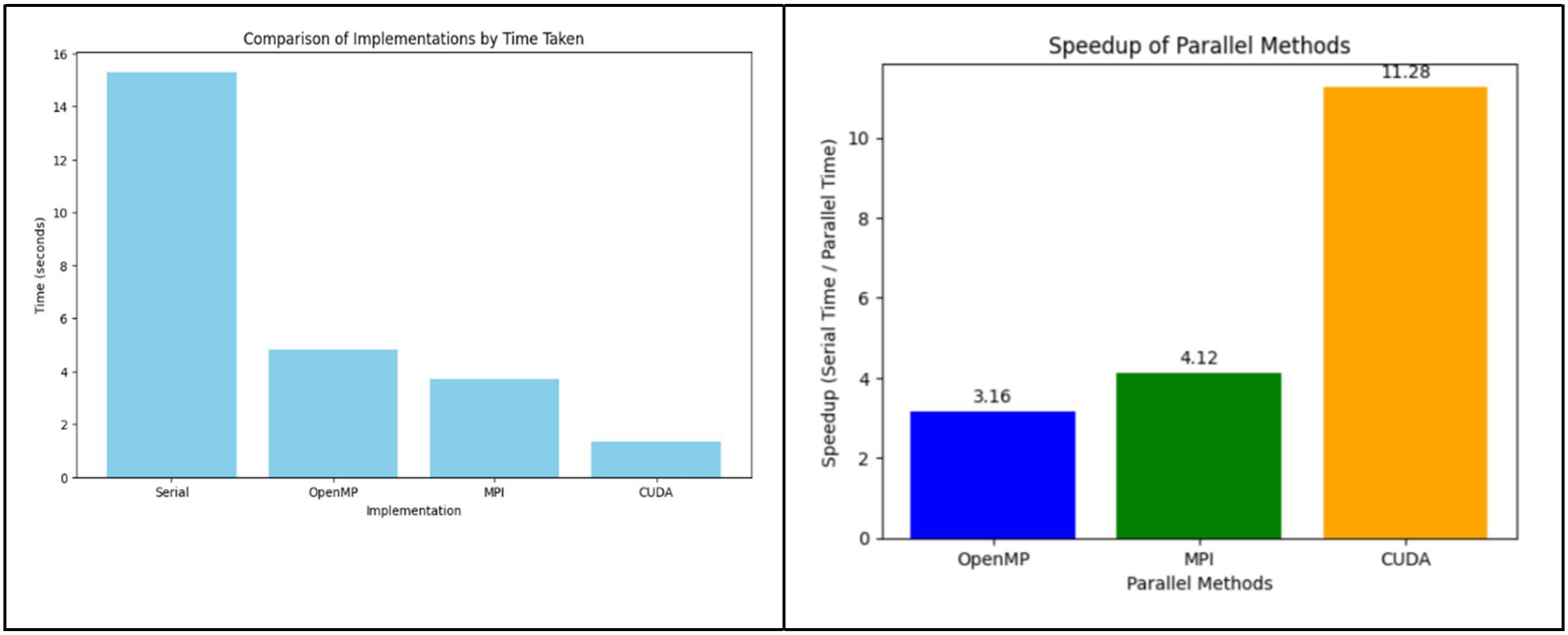


Figure 2.6: Performance Gains with OpenMP, MPI and CUDA

Figures 2.1 to 2.4 show the individual results for each implementation. Each of those figures shows that the values for MSE and SSIM between the returned (decompressed) and original images are very close to 0 and 1 respectively. This indicates that the images are nearly identical. Moreover, there are no visual differences between the original and returned images. The sizes of the returned images are the same as the original images as well. The compression achieved for each implementation is around 60%. Figure 2.7 shows that there are very minor differences between the returned images of the 4 implementations since the MSE values are generally close to 0 and that the SSIM values are generally close to 1.

```
Comparing Image 1 and Image 2:
[ WARN:0] global ./modules/core/
[ WARN:0] global ./modules/core/
MSE: 1.50988e-05, SSIM: 1

Comparing Image 1 and Image 3:
MSE: 0.0237818, SSIM: 0.999765

Comparing Image 1 and Image 4:
MSE: 1.50988e-05, SSIM: 1

Comparing Image 2 and Image 3:
MSE: 0.0237743, SSIM: 0.999765

Comparing Image 2 and Image 4:
MSE: 0, SSIM: 1

Comparing Image 3 and Image 4:
MSE: 0.0237743, SSIM: 0.999765
```

Figure 2.7: Comparison of the Returned Images by all implementations

Referring to Figure 2.6, the results clearly show that OpenMP, MPI and CUDA significantly speed up the compression-decompression pipeline. On one hand, CUDA demonstrated the greatest performance improvement, likely due to its superior ability to parallelize the DCT and IDCT operations. Based on Figure 2.5 under the GPU activities, very minimal time was spent on DCT and IDCT operations. Most of the time was spent on CUDA memcpy from device to host and from host to device. On the other hand, both OpenMP and MPI achieved similar performance levels, which are much lower than that of CUDA's.

2.2 Result for dark.png

The following content shows the results for each implementation. Their results are evaluated under 2.2.5 (Overall Results). Note: the term "returned image" refers to the decompressed image. Since dark.png/dark.bmp is an image of a starry night sky, you may want to increase the brightness to compare the original and decompressed images visually.

2.2.1 Serial Implementation

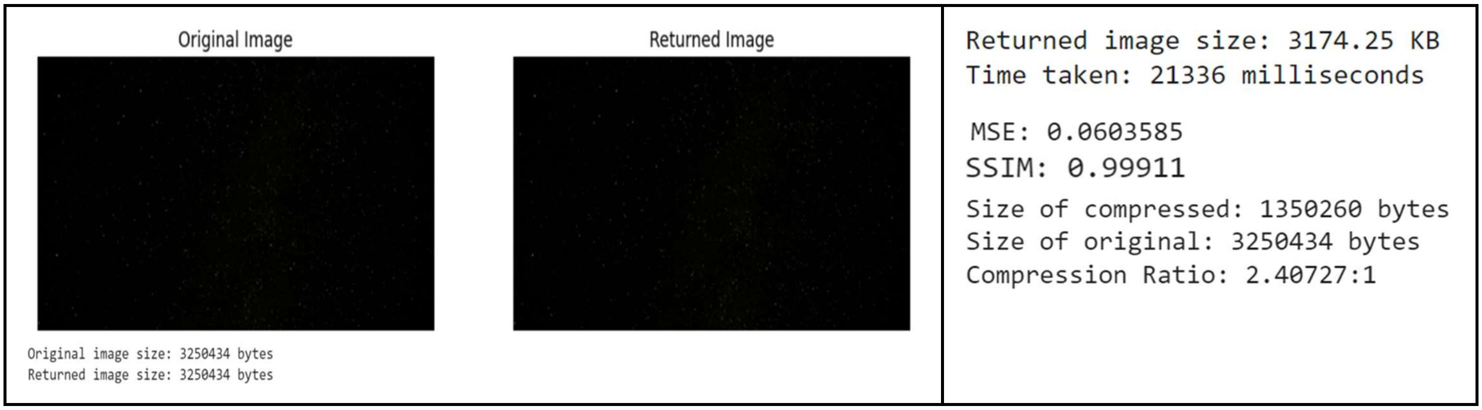


Figure 2.8: Result for Serial Implementation for dark.png

2.2.2 OpenMP Implementation

Figure 2.9: Result for OpenMP Implementation for dark.png

2.2.3 MPI Implementation (with 2 processes)

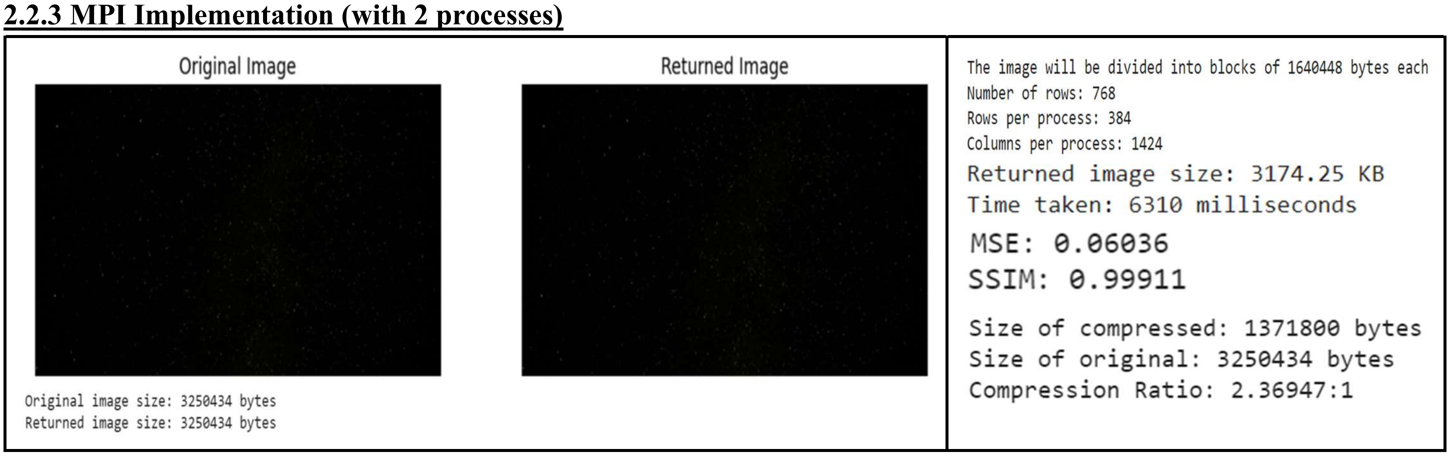
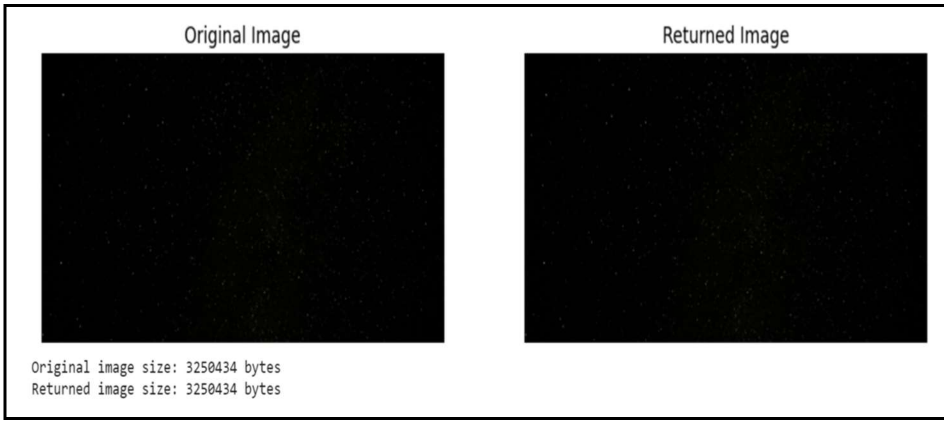


Figure 2.10: Result for MPI Implementation for dark.png

2.2.4 CUDA Implementation



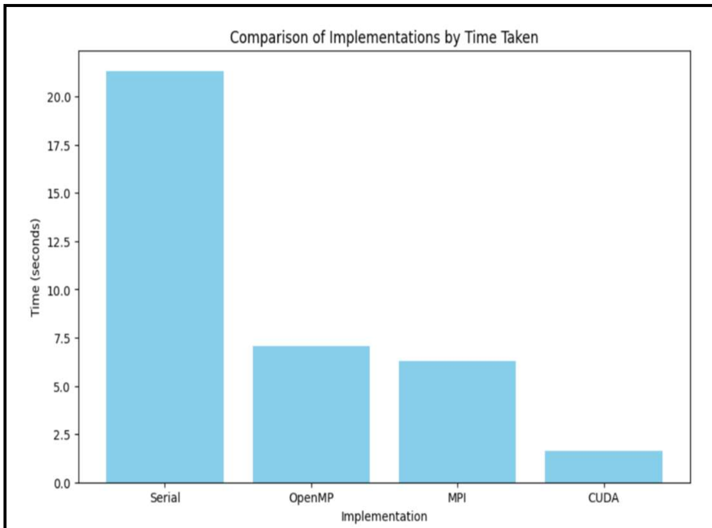

```
Returned image size: 3174.25 KB
GPU Time: 1307.65 milliseconds
Total Time (including I/O): 1648.01 milliseconds
MSE: 0.0604225
SSIM: 0.99911
Size of compressed: 1391148 bytes
Size of original: 3250434 bytes
Compression Ratio: 2.33651:1
```

Figure 2.11: Result for CUDA Implementation for dark.png

```
Returned image size: 3174.25 KB
GPU Time: 1325.08 milliseconds
Total Time (including I/O): 1531.84 milliseconds
==2526== Profiling application: ./dct_idct_zigzagrle testimg.bmp
==2526== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   43.71%  4.7861ms         7  683.73us     704ns  813.91us  [CUDA memcpy HtoD]
                   41.51%  4.5453ms         6  757.55us  704.02us  796.76us  [CUDA memcpy DtoH]
                    8.57%  938.64us         3  312.88us  312.73us  313.05us  idct8x8Kernel(float*, int, int, int)
                    6.21%  679.41us         3  226.47us  226.17us  226.97us  dct8x8Kernel(float*, int, int, int)
      API calls:   86.44%  100.28ms         1  100.28ms  100.28ms  100.28ms  cudaMalloc
                   10.36%  12.025ms        12  1.0020ms  957.71us  1.0295ms  cudaMemcpy
                    1.50%  1.7354ms         6  289.23us  228.63us  335.29us  cudaDeviceSynchronize
                    1.01%  1.1662ms         1  1.1662ms  1.1662ms  1.1662ms  cudaMemcpyToSymbol
                    0.25%  289.91us         1  289.91us  289.91us  289.91us  cudaFree
                    0.19%  216.96us         6  36.160us  22.632us  56.094us  cudaLaunchKernel
                    0.18%  207.64us       114  1.8210us     281ns  83.399us  cuDeviceGetAttribute
                    0.03%  34.285us         2  17.142us  9.6990us  24.586us  cudaEventRecord
                    0.01%  14.344us         1  14.344us  14.344us  14.344us  cuDeviceGetName
                    0.01%  12.431us         2  6.2150us  1.1190us  11.312us  cudaEventCreate
                    0.01%  8.0180us         1  8.0180us  8.0180us  8.0180us  cudaEventSynchronize
                    0.01%  7.9610us         1  7.9610us  7.9610us  7.9610us  cuDeviceGetPCIBusId
                    0.01%  6.5760us         1  6.5760us  6.5760us  6.5760us  cuDeviceTotalMem
                    0.00%  3.3910us         1  3.3910us  3.3910us  3.3910us  cudaEventElapsedTime
                    0.00%  2.6020us         2  1.3010us     716ns  1.8860us  cudaEventDestroy
                    0.00%  2.2030us         3     734ns     348ns  1.5030us  cuDeviceGetCount
                    0.00%  1.2800us         2     640ns     302ns     978ns  cuDeviceGet
                    0.00%     472ns         1     472ns     472ns     472ns  cuModuleGetLoadingMode
                    0.00%     399ns         1     399ns     399ns     399ns  cuDeviceGetUuid
```

Figure 2.12: Result of nvprof command

2.2.5 Overall Results



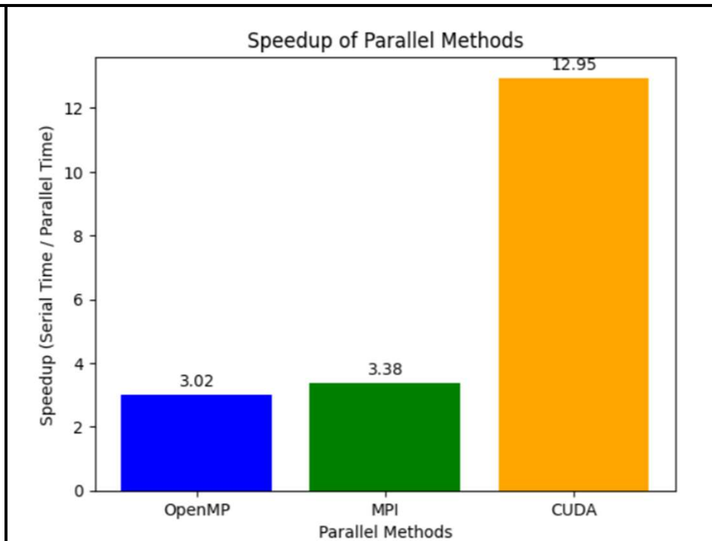


Figure 2.13: Performance Gains with OpenMP, MPI and CUDA

For all 4 implementations, they yielded MSE and SSIM values that are close to 0 and 1 respectively, showing consistency in their results. Visually, all 4 returned images look the same. Figure 2.14 demonstrates that there are no visual differences between the final returned images and the original images from all four implementations. This indicates that the final output from each implementation is nearly identical to the original image. Moreover, the sizes of the returned images are the same and the compression achieved for each implementation is around 57%.

```
Comparing Image 1 and Image 2:
[ WARN:0] global ./modules/core/src
[ WARN:0] global ./modules/core/src
MSE: 6.46078e-06, SSIM: 1

Comparing Image 1 and Image 3:
MSE: 0.000881435, SSIM: 0.999989

Comparing Image 1 and Image 4:
MSE: 6.46078e-06, SSIM: 1

Comparing Image 2 and Image 3:
MSE: 0.000887896, SSIM: 0.999989

Comparing Image 2 and Image 4:
MSE: 0, SSIM: 1

Comparing Image 3 and Image 4:
MSE: 0.000887896, SSIM: 0.999989
```

Figure 2.14: Comparison of the Returned Images by all implementations

Referring to Figure 2.13, the results clearly show that OpenMP, MPI and CUDA continue to significantly speed up the compression-decompression pipeline despite a larger image size. On one hand, CUDA demonstrated the greatest performance improvement, likely due to its superior ability to parallelize the DCT and IDCT operations. Based on Figure 2.12 under the GPU activities, very minimal time was spent on DCT and IDCT operations. Most of the time was spent on CUDA memcpy from device to host and from host to device. On the other hand, both OpenMP and MPI achieved similar performance levels, which are much lower than that of CUDA's.

2.3 Result for dnb_land_ocean_ice.2012.13500x13500.A1-0000.png

The following content shows the results for each implementation. Their results are evaluated under 2.3.5 (Overall Results). Note: the term "returned image" refers to the decompressed image.

2.3.1 Serial Implementation


```
Original image size: 68350554 bytes
Returned image size: 68350554 bytes
```

```
Returned image size: 66748.6 KB
Time taken: 398787 milliseconds
MSE: 0.0155246
SSIM: 0.999729
Size of compressed: 5591512 bytes
Size of original: 68350554 bytes
Compression Ratio: 12.224:1
```

Figure 2.15: Result for Serial Implementation

2.3.2 OpenMP Implementation

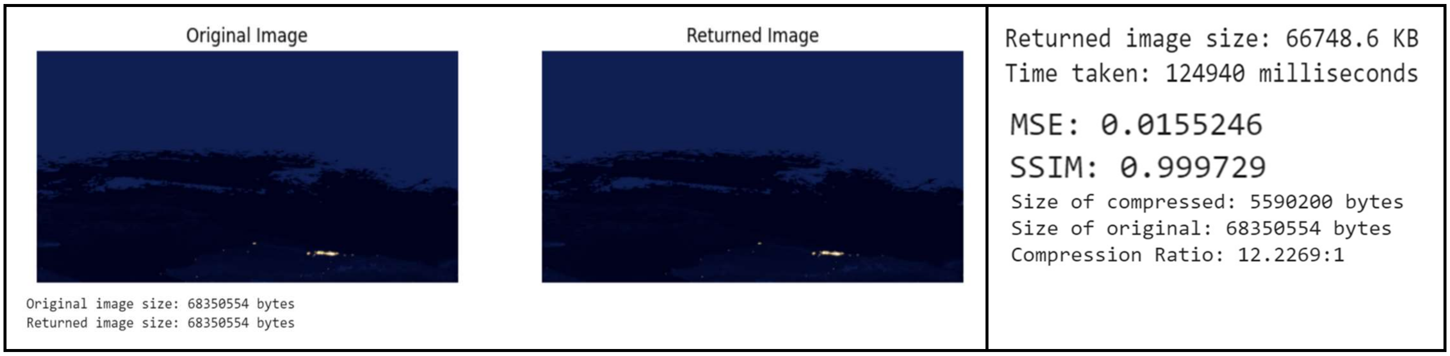


Figure 2.16: Result for OpenMP Implementation

2.3.3 MPI Implementation (with 2 processes)

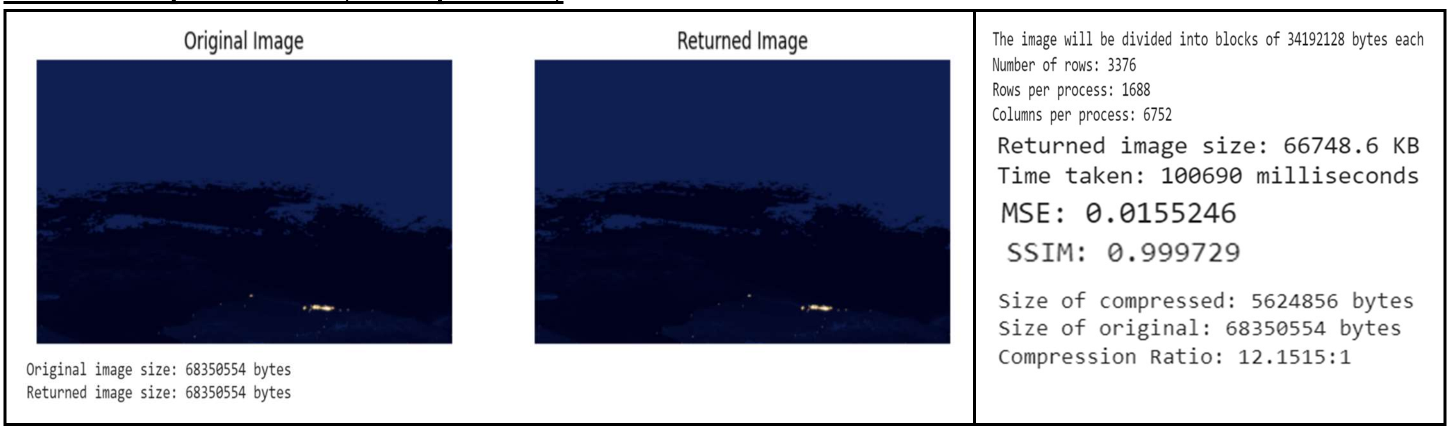


Figure 2.17: Result for MPI Implementation

2.3.4 CUDA Implementation

Figure 2.18: Result for CUDA Implementation

```
Returned image size: 66748.6 KB
GPU Time: 15693.2 milliseconds
Total Time (including I/O): 16397.2 milliseconds
==10211== Profiling application: ./dct_idct_zigzagrle testimg.bmp
==10211== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   46.25%  118.94ms         7  16.992ms     704ns  21.104ms  [CUDA memcpy HtoD]
                   40.88%  105.13ms         6  17.522ms  16.506ms  18.905ms  [CUDA memcpy DtoH]
                    7.50%  19.279ms         3  6.4262ms  6.4253ms  6.4273ms  idct8x8Kernel(float*, int, int, int)
                    5.37%  13.818ms         3  4.6062ms  4.6059ms  4.6063ms  dct8x8Kernel(float*, int, int, int)
      API calls:   64.92%  227.31ms        12  18.942ms  16.811ms  21.339ms  cudaMemcpy
                   24.81%  86.874ms         1  86.874ms  86.874ms  86.874ms  cudaMalloc
                    9.51%  33.307ms         6  5.5512ms  4.6145ms  6.4666ms  cudaDeviceSynchronize
                    0.44%  1.5410ms         1  1.5410ms  1.5410ms  1.5410ms  cudaMemcpyToSymbol
                    0.18%  613.64us         1  613.64us  613.64us  613.64us  cudaFree
                    0.07%  250.90us         6  41.815us  29.019us  65.444us  cudaLaunchKernel
                    0.04%  132.46us       114  1.1610us     145ns  51.818us  cuDeviceGetAttribute
                    0.01%  34.370us         2  17.185us  8.0320us  26.338us  cudaEventRecord
                    0.00%  12.882us         2  6.4410us  1.2170us  11.665us  cudaEventCreate
                    0.00%  12.608us         1  12.608us  12.608us  12.608us  cuDeviceGetName
                    0.00%  8.0240us         1  8.0240us  8.0240us  8.0240us  cudaEventSynchronize
                    0.00%  5.3640us         1  5.3640us  5.3640us  5.3640us  cuDeviceGetPCIBusId
                    0.00%  4.5150us         1  4.5150us  4.5150us  4.5150us  cuDeviceTotalMem
                    0.00%  3.5400us         1  3.5400us  3.5400us  3.5400us  cudaEventElapsedTime
                    0.00%  2.7390us         2  1.3690us     864ns  1.8750us  cudaEventDestroy
                    0.00%  1.5580us         3     519ns     209ns  1.0350us  cuDeviceGetCount
                    0.00%  1.1150us         2     557ns     183ns     932ns  cuDeviceGet
                    0.00%     607ns         1     607ns     607ns     607ns  cuModuleGetLoadingMode
                    0.00%     275ns         1     275ns     275ns     275ns  cuDeviceGetUuid
```

Figure 2.19: Result of nvprof command

2.3.5 Overall Results

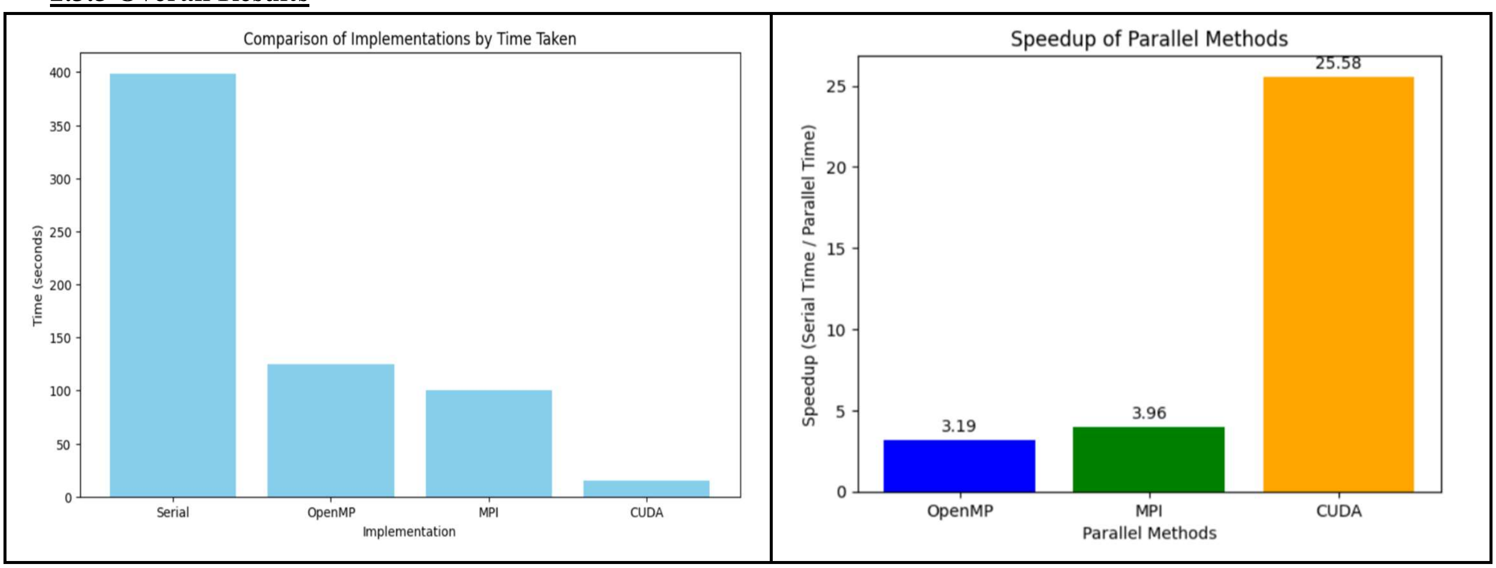


Figure 2.20: Performance Gains with OpenMP, MPI and CUDA

Referring to Figure 2.20, the serial method continues to be the slowest, taking nearly 400 seconds to complete, while the parallel methods (OpenMP, MPI and CUDA) are still much faster. MPI continues to perform slightly better than OpenMP, but CUDA stands out as the fastest to complete the task in a fraction of the time. In terms of speedup, CUDA significantly improves performance, making the process around 25 times faster than the serial method. Referring to Figure 2.19, most of the time was spent on CUDA memcpy from device to host and from host to device. MPI and OpenMP also offer improvements with speedups of 3.96 and 3.19 times. Overall, CUDA continues to be the most efficient method both in time saved and speed increase. Moreover, the sizes of the returned images are the same. The compression achieved by each implementation is around 90%.

```
Comparing Image 1 and Image 2:
[ WARN:0] global ./modules/core/
[ WARN:0] global ./modules/core/
MSE: 6.58436e-07, SSIM: 1

Comparing Image 1 and Image 3:
MSE: 0.00263778, SSIM: 0.999966

Comparing Image 1 and Image 4:
MSE: 6.58436e-07, SSIM: 1

Comparing Image 2 and Image 3:
MSE: 0.002638, SSIM: 0.999966

Comparing Image 2 and Image 4:
MSE: 0, SSIM: 1

Comparing Image 3 and Image 4:
MSE: 0.002638, SSIM: 0.999966
```

Figure 2.21: Comparison of the Returned Images by all implementations

Referring to Figure 2.21, the results show that all the returned images are almost the same as each other. For every comparison, the pixel difference (MSE) is 0 or close to 0 and the similarity score (SSIM) is 1 or close to 1. This means there are no major differences between the images.

2.4 Evaluation of Results for all 3 Images

The impact of parallel computing on both compression and decompression processes is observable regardless of the image file size. Across the three different images tested—varying in resolution and complexity—the performance improvements offered by each parallel platform (OpenMP, CUDA and MPI) showed consistent patterns. Based on the results of the 3 images, as the image size increases, the performance gain provided by each parallel platform also increases gradually. This suggests that the parallel algorithms implemented are well-optimized to handle diverse data sizes, mitigating bottlenecks that typically arise with larger image files. The ability of the parallel algorithms to maintain performance across different image sizes suggests potential for applying these methods to other data types or domains where large file sizes or complex data structures are a factor.

Referring to the results of `nvprof`, the CUDA program is bottlenecked by memory transfers between the host (CPU) and device (GPU), as `cudaMemcpy` operations consume the highest percentage of time, while the DCT and IDCT kernels show low execution times. This suggests that, although the kernels are efficient, the frequent or large data transfers are slowing down overall performance. To optimize the program, the user should minimize memory transfers, utilize pinned memory, and overlap data transfers with kernel execution using asynchronous operations. These steps will improve GPU utilization and reduce transfer-related bottlenecks.

For all 4 implementations (including the serial implementation), the returned image sizes, MSE and SSIM values were the same across the three test images. The consistency in these values highlights that while each platform leverages different techniques to speed up the process, the overall outcome in terms of the returned image remains comparable.

CHAPTER 3: DISCUSSION AND CONCLUSION

3.1 Significance of Results and System Performance

The results of using CUDA, OpenMP, and MPI to parallelize near lossless image compression with DCT demonstrated significant performance improvements across all platforms, effectively speeding up the compression process. All 3 parallel platforms provided more than 2 times performance gain. Each platform contributed unique strengths to different areas of the process, while also posing certain challenges. CUDA was the fastest, followed by MPI and then OpenMP.

CUDA: As a GPU-based platform, CUDA excelled in performing the computationally intensive tasks of the DCT & IDCT, providing the fastest execution times for large matrix transformations. The parallel processing capabilities of CUDA enabled efficient handling of multiple image blocks simultaneously, significantly reducing compression and decompression times. However, CUDA's reliance on a specific GPU architecture posed a hardware dependency, and ensuring optimal task distribution across the GPU threads was challenging.

OpenMP: OpenMP showed strong performance in parallelizing the individual for-loops within the DCT and IDCT functions, effectively distributing tasks across CPU cores. It provided an easy-to-implement solution for multi-core parallelism. However, race conditions were a challenge when data dependencies were not properly managed, particularly when accessing shared resources. Despite this, OpenMP achieved good speedup for smaller to medium-sized image data, particularly in regions where thread-level parallelism was most effective.

MPI: In this implementation, MPI divided the image among processes, allowing each process to handle specific portions of the image. This approach worked well for distributing the workload evenly across processes, particularly for large images with many blocks. However, communication overhead between processes and ensuring that tasks were properly balanced remained challenges. As the number of processes increased, managing synchronization and avoiding bottlenecks in data transfer became more difficult, which could impact scalability. Furthermore, it was important to set the correct number of processes to process any image. This is because since we are performing 8×8 block processing, the number of rows of the image assigned to each process must be evenly distributed and a multiple of 8. Despite these challenges, MPI provided effective parallelism for handling distributed image blocks in the compression process.

Using **hardcoded cosine values** in DCT or IDCT enhances performance by avoiding repeated computation of costly trigonometric functions. This is especially important in parallel environments like OpenMP, MPI and CUDA, where efficiency is key. Precomputing these values ensures consistency across threads or processes, reducing variability in the results. In CUDA, leveraging hardcoded values stored in constant memory further optimises access speed during kernel execution. Overall, this approach significantly boosts the speed of image compression tasks.

Although not shown in the report and code, **combining MPI and OpenMP** to parallelize the image compression-decompression program—by using MPI for distributing image blocks and OpenMP for parallelizing computations within each process—did not necessarily result in a significant speedup. The complexity of managing both inter-process communication and thread-level parallelism introduced additional overhead, which sometimes negated the performance gains. Moreover, ensuring proper synchronization between processes and threads increased the risk of inefficiencies, limiting the overall effectiveness of the combined approach.

In conclusion, an image compression-decompression pipeline can be sped up with the use of parallel computing platforms, such as OpenMP, MPI and CUDA. Moreover, the performance gain provided by each parallel platform increases as the dataset size increases. With reference to the results, CUDA seems to be the most attractive option to speed up an image compression-decompression pipeline.

3.2 Limitations

- **Exclusion of Other Platforms and Limited Testing Scope:** The project was limited to CUDA, OpenMP, and MPI, excluding other parallel platforms like OpenCL, OpenGL, and TBB. This limitation may affect the generalizability of results and overlook potential optimization opportunities provided by these additional platforms.
- **Hardware Dependency:** Performance improvements are constrained by the available hardware. The project utilized a T4 GPU and a 2-Core CPU from Google Colab, which may limit potential gains. Testing on a broader range of hardware could provide more comprehensive performance insights.
- **Learning Curve:** The need to master different optimization strategies and debugging approaches for CUDA, OpenMP, and MPI increases the learning curve and development time.
- **MPI Scalability:** MPI scalability may be challenged by communication overhead and system limitations as the number of processes increases, potentially affecting performance.
- **Single-Computer MPI Implementation:** The MPI version of the project was implemented on a single computer, limiting the ability to fully explore distributed computing advantages across multiple nodes.

3.3 Future Improvements

- Future work could explore other parallel computing platforms such as OpenCL, OpenGL, or TBB. This would provide a more comprehensive analysis and reveal additional optimization opportunities across a broader range of environments, making the solution more adaptable and robust.
- To address hardware limitations, future projects could test on a wider variety of hardware setups, including more advanced GPUs and multi-core CPUs. This would provide deeper insights into performance scalability and allow optimization across different hardware architectures.
- To improve MPI scalability, future efforts could focus on optimizing communication patterns or exploring hybrid models that combine MPI with shared memory parallelism (e.g., OpenMP) to reduce overhead. Additionally, testing on larger clusters and more advanced network configurations could provide insights into optimizing large-scale MPI applications.
- Future work could explore alternative encoding methods that achieve true lossless compression without significant quantization. This might include adaptive quantization techniques or hybrid approaches that combine DCT with other compression algorithms, balancing precision with compression ratio more effectively.
- Testing the MPI version on a multi-node cluster would provide a better understanding of performance scaling in a true distributed environment. This could reveal insights into network latency, inter-node communication overhead, and overall scalability in a real-world distributed system setup.
- The project focused solely on DCT, limiting the exploration of other transform algorithms. Future work could investigate other transform algorithms, such as the Haar Wavelet Transform (HWT) or the Discrete Wavelet Transform (DWT), and parallelize them using similar techniques. This would allow a comparison of performance, compression efficiency and scalability across different algorithms.
- As a future improvement, the project could integrate actual network protocols to simulate the full transfer of image data. While the current focus was on parallelization for speeding up compression and decompression, incorporating real network communication would offer a more comprehensive assessment of performance in real-world scenarios.
- Future work could also focus on gray scale images (1 channel) and RGBA images (4 channels) instead of solely on coloured images (3 channels).

Name(s): Ryan Kho Yuen Thian, Ong Weng Kai, Yong Zee Lin Program: RDS3S1 Group: G2 Date: 17/9/2024

Program (60%) - CLO1

No	Item	Criteria			Final Marks
		Poor	Accomplished	Good	
1	Output (10)	Inadequate information/outputs needed are generated. Most of the information/outputs generated are less accurate. Results visualization is overly cluttered or the design seems inappropriate for problem area. Lack of information that are useful for the user 0-4	Adequate information/outputs needed are generated. The information/output generated are accurate but some with errors. Pleasant looking, clean, well-organized results visualization The information displayed are useful for the user, but some details are omitted. 5-7	All the necessary information/outputs are generated. All or most of the information/outputs generated are accurate. Minor errors can be ignored. The results are visually pleasing and appealing. Great use of colors, fonts, graphics and layout. The information displayed are useful to the users and complete with necessary details. 8-10	
2	Programming (10)	The end product fails with many logic errors, many actions lacked exception handling. Solutions are over-simplified. Programming skill needs improvement. 0-4	Major parts are logical, but some steps to complete a specific job may be tedious or unnecessarilycomplicated. Program algorithm demonstrates acceptable level of complexity. The student is qualified to be a programmer 5-7	Correct and logical flow, exceptions are handled well. Demonstrates appropriate or high level of complex algorithms and programming skills. 8-10	
3	Degree of completion (10)	Too much still remain to be done. Basic requirements are not fulfilled. The end product produces enormous errors, faults or incorrect results. 0-4	All required features present in the interface within the required scope, but some are simplified. Or one or two features are missing. The system is able to run with minor errors. 5-7	All required features present in the interface within or beyond the required scope. No bugs apparent during demonstration. 8-10	
4	Program Model Optimization (10)	The model is not optimized. Most of the processes are executed in serial. Only 1 parallel program model is used. 0-4	The model is optimized by using more than 1 parallel program model, i.e. SPMD, loop parallelism. 5-7	The model is optimized by using more than 1 parallel program model, i.e. SPMD, loop parallelism. The model is tested on different parallel platform, i.e. OpenMP (Homogenous), CUDA, OpenCL (Heterogenous). 8-10	

No	Item	Criteria			Final Marks
		Poor	**Accomplished**	**Good**	
5	System implementation (10)	The end product is produced with different system design or approach, which is not related to the initial proposal. 0-4	The end product conforms to most of the system design, but some are different from the specification. 5-7	The end product fully conforms to the proposed system design. 8-10	
6	Presentation (10)	The student is unclear about the work produced, sometimes not even knowing where to find the source code. 0-4	The student knows the code whereabouts, but sometimes may not be clear why the work was done in such a way. 5-7	The student is clear about every piece of the work done. 8-10	
				Sum of Score	

Final Report (40%) – CLO3

No	Item	Criteria				Final Marks
		Missing or Unacceptable	**Poor**	**Accomplished**	**Good**	
1	Title and abstract (10)	Title or abstract were omitted or inappropriate given the problem, research questions and method 0-2	Title or abstract lacks relevance or fails to offer appropriate details about the education issue, variables, context, or methods of the proposed study. 3-4	Title and abstract are relevant, offering details about the proposed research study. 5-7	Title and abstract are informative, succinct, and offer sufficiently specific details about the educational issue, variables, context, and proposed methods of the study. 8-10	
2	Results (Performance measurement) (10)	Analytical methods were missing or inappropriately aligned with data and research design. Results were confusing. 0-2	Analytical method was identified but the results were confusing, incomplete or lacked relevance to the research questions, data, or research design. 3-4	The analytical methods were identified. Results were presented. All were related to the research question and design. Sufficient metric or measurement is applied. 5-7	Analytical methods and results presentation were sufficient, specific, clear, structured and appropriate based on the research questions and research design. Extra metric or measurement is applied. 8-10	
3	Discussion and Conclusion (10)	Discussions or answers to the research question and system performance were omitted or confusing. No or very little conclusion could be yielded. 0-2	Little discussions were presented. Answers to the research question and system performance were unclear or confusing. 3-4	Discussions of the results were presented. The research question and system performance were answered and identified. 5-7	The significance of the results of the work was discussed, sufficiently inclusive of the information that concluded and answered the research question and system performance is evaluated comprehensively. Limitations and future improvements of the studies were identified. 8-10	
4	Organization (5)	The structure of the paper was incomprehensible, irrelevant, or confusing. Transition was awkward. 0-1	The structure of the paper was weak. Transition was weak and difficult to understand. 0-2	A workable structure was presented for presenting ideas. Transition was smooth and clear. 3-4	Structure was intuitive and sufficiently inclusive of important information of the research. Transition from one to another was smooth and organized. 5	
5	Spelling, Grammar and Writing Mechanics (5)	There were so many errors that meaning was obscured, make the content became difficult to understand 0-1	Some grammar or spelling errors were spotted. Some sentences were awkwardly constructed so that the reader was occasionally distracted. 0-2	There were occasional errors, but they did not represent a major distraction or obscure meaning. 3-4	Sentences were well-phrased. The writing was free or almost free of errors. 5	
					Sum of Score	
					Final score = sum of scores/100*60 (base 60%)	