# PARALLEL AND DISTRIBUTED COMPUTING ASSIGNMENT - 1


## Yashas Kumar S & Shashank Upadhyay

## 2022BCS0145 & 2022BCS0088


## 1. Objective and description

**a) Objective Purpose:**

In parallelizing the pre-processing operations involved in an image, flipping an image, rotating it by 90°, and converting it into grayscale to enhance the performance as well as scalability of data augmentation during an image processing pipeline.

**b) Description**

**What:**

Image pre-processing is an important part of the computer vision and machine learning pipeline, transforming raw images into a format convenient for analysis or model training. Data augmentation, as a special case of pre-processing, generates more samples artificially from existing images through different applied transformations, thus enhancing the generalization of models.

**Why:**

- **Inefficiency:** You may need to process very large datasets sequentially in certain contexts. Processing time significantly reduces since you can parallelize tasks.

- **Scalability:** With growth in dimensions of datasets, parallel processing handles pre-processing amicably.

- **Performance:** The faster the pre-processing, the faster the model's training and experimentation cycles.

**Where:**

This parallelized image pre-processing can be used for following applications

- **Pipelines with Machine Learning**: Improvement of dataset to make it ready to train a model.

- **Real Time Systems:** Satisfying fast pre-processing in applications connected with autonomous vehicle or video real-time analysis.

- **Distributed System Environments of Cloud Computing:** Using distributed systems to process larger image datasets.

## 2. Algorithm:

1. Start

2. Load the image matrix (m x n for grayscale, m x n x 3 for colored).

3. Apply transformations sequentially or in parallel:

   a. Flip the image horizontally:

      For each row, reverse the column elements.

   b. Rotate the image by 90 degrees:

      Transpose the matrix, then reverse the rows (or columns).

   c. Convert the image to grayscale:

      For each pixel (R, G, B), compute luminance using:

      L = 0.2989 * R + 0.5870 * G + 0.1140 * B

4. Display the processed image.

5. End

## 3. Solution demonstration:

eg: Matrix =

$$\begin{bmatrix} (1,0,1) & (4,5,6) \\ (2,3,4) & (7,8,9) \end{bmatrix}$$

Image flip horizontally:

$$\begin{bmatrix} (4,5,6) & (1,0,1) \\ (7,8,9) & (2,3,4) \end{bmatrix}$$

Rotate image by 90°:
Transpose:

$$\begin{bmatrix} (1,0,1) & (2,3,4) \\ (4,5,6) & (7,8,9) \end{bmatrix}$$

Reverse rows:

$$\begin{bmatrix} (2,3,4) & (1,0,1) \\ (7,8,9) & (4,5,6) \end{bmatrix}$$

RGB to BW:

$$0 : 1 \times 0.2989 + 0 \times 0.5870 + 1 \times 0.1140 = 0.4129$$
$$1 : 4 \times 0.2989 + 5 \times 0.5870 + 6 \times 0.1140 = 4.8146$$
$$2 : 2 \times 0.2989 + 3 \times 0.5870 + 4 \times 0.1140 = 2.8148$$
$$3 : 7 \times 0.2989 + 8 \times 0.5870 + 9 \times 0.1140 = 7.8143$$

$$\begin{bmatrix} 0.4129 & 4.8146 \\ 2.8148 & 7.8143 \end{bmatrix}$$

$$\begin{bmatrix} (0.4129, 0.4129, 0.4129) & (4.8146, 4.8146, 4.8146) \\ (2.8148, 2.8148, 2.8148) & (7.8143, 7.8143, 7.8143) \end{bmatrix}$$

## 4. Serial code in C:

```c
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to flip the image horizontally
void flipImageHorizontally(unsigned char *img, int width, int height, int channels) {
   for (int i = 0; i < height; i++) {
      for (int j = 0; j < width / 2; j++) {
         for (int c = 0; c < channels; c++) {
            int left_idx = (i * width + j) * channels + c;
            int right_idx = (i * width + (width - j - 1)) * channels + c;

            // Swap pixels
            unsigned char temp = img[left_idx];
            img[left_idx] = img[right_idx];
            img[right_idx] = temp;
         }
      }
   }
}

// Function to rotate the image by 90 degrees clockwise
unsigned char* rotateImage90(unsigned char *img, int width, int height, int channels) {
   unsigned char *rotatedImg = (unsigned char*)malloc(width * height * channels);

   // Perform 90-degree clockwise rotation
```

```
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            for (int c = 0; c < channels; c++) {
                // Map the original pixel (i, j) to the new position (j, height - i - 1)
                rotatedImg[(j * height + (height - i - 1)) * channels + c] = img[(i * width + j) * channels + c];
            }
        }
    }

    return rotatedImg;
}

// Function to convert the image to grayscale (assuming 3 channels RGB)
void convertToGrayscale(unsigned char *img, int width, int height, int channels) {
    for (int i = 0; i < width * height; i++) {
        unsigned char r = img[i * channels];
        unsigned char g = img[i * channels + 1];
        unsigned char b = img[i * channels + 2];

        // Calculate grayscale value using luminosity method
        unsigned char gray = (unsigned char)(0.2989 * r + 0.5870 * g + 0.1140 * b);

        // Set all RGB channels to the grayscale value
        img[i * channels] = gray;
        img[i * channels + 1] = gray;
        img[i * channels + 2] = gray;
    }
}

int main() {
    srand(time(NULL)); // Seed for random number generation
```

```
int width, height, channels;

// Load image
unsigned char *img = stbi_load("download.jpeg", &width, &height, &channels, 0);
printf("Image loaded: %dx%d, %d channels\n", width, height, channels);

int choice;
printf("Choose an operation:\n");
printf("1. Flip Image Horizontally\n");
printf("2. Rotate Image 90 Degrees\n");
printf("3. Convert Image to Grayscale\n");
printf("Enter your choice (1-3): ");
scanf("%d", &choice);

unsigned char *resultImg = NULL;

switch (choice) {
  case 1:
    flipImageHorizontally(img, width, height, channels);
    resultImg = img;
    printf("Image flipped horizontally.\n");
    break;
  case 2:
    resultImg = rotateImage90(img, width, height, channels);
    printf("Image rotated by 90 degrees.\n");
    // Swap width and height after rotation
    int temp = width;
    width = height;
    height = temp;
    break;
  case 3:
```

```
            convertToGrayscale(img, width, height, channels);
            resultImg = img;
            printf("Image converted to grayscale.\n");
            break;
        default:
            printf("Invalid choice.\n");
            return -1;
    }

    // Save the processed image
    if (choice == 2) {
        stbi_write_jpg("processed_image.jpg", height, width, channels, resultImg, 100);
    } else {
        stbi_write_jpg("processed_image.jpg", width, height, channels, resultImg, 100);
    }

    printf("Image processing complete. Saved as 'processed_image.jpg'.\n");

    return 0;
}
```

**Input:**

**Output:**

**Test 1:**

```
PS D:\College\Sem5\Parallel computing> ./a.exe
Image loaded: 254x180, 3 channels
Choose an operation:
1. Flip Image Horizontally
2. Rotate Image 90 Degrees
3. Convert Image to Grayscale
Enter your choice (1-3): 1
Image flipped horizontally.
Image processing complete. Saved as 'processed_image.jpg'.
```

**Test2:**

```
PS D:\College\Sem5\Parallel computing> ./a.exe
Image loaded: 254x180, 3 channels
Choose an operation:
1. Flip Image Horizontally
2. Rotate Image 90 Degrees
3. Convert Image to Grayscale
Enter your choice (1-3): 3
Image converted to grayscale.
Image processing complete. Saved as 'processed_image.jpg'.
```

**Test 3:**

```
PS D:\College\Sem5\Parallel computing> ./a.exe
Image loaded: 305x180, 3 channels
Choose an operation:
1. Flip Image Horizontally
2. Rotate Image 90 Degrees
3. Convert Image to Grayscale
Enter your choice (1-3): 1
Image flipped horizontally.
Image processing complete. Saved as 'processed_image.jpg'.
```

## 5. Time complexity analysis:

### 1. Flipping the Image Horizontally

- **Operation**: The image is flipped horizontally by swapping pixels across the vertical axis.
- **Time Complexity**:
  - Outer Loop: Iterates over the height of the image, m.
  - Inner Loop: Iterates over half the width of the image, n/2.
  - Innermost Loop: Iterates over the color channels, c.

Total time complexity for flipping the image:

- T_flip = O(m * (n/2) * c) = O(m * n * c)

## 2. Rotating the Image by 90 Degrees

- **Operation**: The image is rotated by 90 degrees by mapping each pixel to its new position.
- **Time Complexity**:
  - Outer Loop: Iterates over the width of the image, n.
  - Inner Loop: Iterates over the height of the image, m.
  - Innermost Loop: Iterates over the color channels, c.

Total time complexity for rotating the image:

- T_rotate = O(n * m * c) = O(m * n * c)

## 3. Converting to Black and White (Grayscale)

- **Operation**: The luminance for each pixel of a colored image is computed to convert it to black and white.
- **Time Complexity**:
  - Outer Loop: Iterates over the height of the image, m.
  - Inner Loop: Iterates over the width of the image, n.
  - Color channels involved for RGB, so the computation is done for each pixel.

Total time complexity for converting to grayscale:

- T_bw = O(m * n * c)

## Overall Time Complexity

For a single image of size m x n with c color channels, the overall time complexity for all operations when performed sequentially is:

- T_total = T_flip + T_rotate + T_bw
- T_total = O(m * n * c) + O(m * n * c) + O(m * n * c)

Assuming that m' x n' is smaller than or equal to m x n, we can simplify the overall complexity:

- T_total = O(m * n * c)

## Asymptotic Complexity

- Sequential: O(m * n * c)

Results (Serial):

| Image size | Function | Time (in secs) |
|---|---|---|
| 297 x 169 | Flip image horizontally | 0.001191 |
| | Rotate by 90° | 0.002385 |
| | Convert image to grayscale | 0.001195 |
| 1000 x 1000 | Flip image horizontally | 0.013568 |
| | Rotate by 90° | 0.018363 |
| | Convert image to grayscale | 0.011990 |

# MODULE – 2

# Yashas Kumar S & Shashank Upadhyay

# 2022BCS0145 & 2022BCS0085

## 6. Identification of Parallelizable Blocks:

1. **Flip Image Horizontally (flipImageHorizontally Function):**
   - **Outer Loop (for (int i = 0; i < height; i++)):** Each iteration processes a different row of the image independently.
   - **Inner Loop (for (int j = 0; j < width / 2; j++)):** Swapping pixels within a row can be done in parallel since each swap is independent of others.
2. **Rotate Image 90 Degrees Clockwise (rotateImage90 Function):**
   - **Outer Loop (for (int i = 0; i < height; i++)):** Each iteration handles a different row of the original image.
   - **Inner Loop (for (int j = 0; j < width; j++)):** Each pixel in a row is mapped to a new position in the rotated image independently.
3. **Convert Image to Grayscale (convertToGrayscale Function):**
   - **Single Loop (for (int i = 0; i < width * height; i++)):** Each pixel's RGB values are processed independently to compute the grayscale value.

**Rationale for Parallelization:**

- **Data Independence:** Each pixel operation is independent, ensuring no data races or dependencies.
- **Large Data Size:** Images typically contain a large number of pixels, providing ample opportunities for parallelism.
- **Uniform Workload:** Similar processing tasks across pixels allow for balanced workload distribution across threads.

## 7. Pseudocode:

```
Function flipImageHorizontally(img, width, height, channels):
  Parallelize the outer loop over 'i' using OpenMP
    For each row 'i' from 0 to height-1:
      Parallelize the inner loop over 'j' using OpenMP
        For each column 'j' from 0 to width/2 -1:
          For each channel 'c' from 0 to channels-1:
            Compute left_idx and right_idx
            Swap img[left_idx] with img[right_idx]


Function rotateImage90(img, width, height, channels):
```

```
    Allocate memory for rotatedImg with size width * height * channels
    Parallelize the outer loop over 'i' using OpenMP
        For each row 'i' from 0 to height-1:
            Parallelize the inner loop over 'j' using OpenMP
                For each column 'j' from 0 to width-1:
                    For each channel 'c' from 0 to channels-1:
                        Compute source index and destination index
                        Set rotatedImg[destination index] = img[source index]
    Return rotatedImg

Function convertToGrayscale(img, width, height, channels):
    Parallelize the loop over 'i' using OpenMP
        For each pixel 'i' from 0 to (width * height -1):
            Extract R, G, B values
            Compute grayscale value using luminosity method
            Set all RGB channels of img[i] to grayscale value
```

## 8. Solution demonstration:

Matrix =

$$\begin{bmatrix} (1,2,1) & (1,1,0) & (1,0,0) \\ (1,1,1) & (0,1,1) & (0,0,1) \\ (0,1,0) & (1,0,1) & (1,1,0) \end{bmatrix}$$

### Image flipping horizontally:

Thread 3 works on (1,0,1) & (1,0,0) & swaps them

Thread 1 works on (1,1,0) & swaps them with itself

Similarly, in all rows happens parallelly.

$$\begin{bmatrix} (1,0,0) & (1,1,0) & (1,0,1) \\ (0,0,1) & (0,1,1) & (1,1,1) \\ (1,1,0) & (1,0,1) & (0,1,0) \end{bmatrix}$$

### Rotate image by 90°:

Transpose reverse rows:

The swaps are done parallel & independently.

$$\begin{bmatrix} (0,1,0) & (1,1,1) & (1,0,1) \\ (1,0,1) & (0,1,1) & (1,1,0) \\ (1,1,0) & (0,0,1) & (1,0,0) \end{bmatrix}$$

### RGB to BW:

Compute grayscale value:

Set all RGB channels of img[i,j] to grayscale value

$$\begin{aligned}
&\text{Thread 3:} && 1 \times 0.2989 + 0 \times 0.5870 + 1 \times 0.1140 = 0.4129 \\
&\text{Thread 5:} && 1 \times 0.2989 + 1 \times 0.5870 + 0 \times 0.1140 = 0.8859 \\
&\text{Thread 1:} && 1 \times 0.2989 + 0 \times 0.5870 + 0 \times 0.1140 = 0.2989 \\
&\text{Thread 4:} && 1 \times 0.2989 + 1 \times 0.5870 + 1 \times 0.1140 = 0.9999 \\
&\text{Thread 2:} && 0 \times 0.2989 + 1 \times 0.5870 + 1 \times 0.1140 = 0.701 \\
&\text{Thread 6:} && 0 \times 0.2989 + 0 \times 0.5870 + 1 \times 0.1140 = 0.1140 \\
&\text{Thread 8:} && 0 \times 0.2989 + 1 \times 0.5870 + 0 \times 0.1140 = 0.5870 \\
&\text{Thread 7:} && 1 \times 0.2989 + 0 \times 0.5870 + 1 \times 0.1140 = 0.4129 \\
&\text{Thread 9:} && 1 \times 0.2989 + 1 \times 0.5870 + 0 \times 0.1140 = 0.8859
\end{aligned}$$

$$\begin{bmatrix} (0.4129, 0.4129, 0.4129) & (0.8859, 0.8859, 0.8859) & (0.2989, 0.2989, 0.2989) \\ (0.9999, 0.9999, 0.9999) & (0.701, 0.701, 0.701) & (0.1140, 0.1140, 0.1140) \\ (0.5870, 0.5870, 0.5870) & (0.4129, 0.4129, 0.4129) & (0.8859, 0.8859, 0.8859) \end{bmatrix}$$

## 9. Parallel code:

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

// Function to flip the image horizontally
void flipImageHorizontally(unsigned char *img, int width, int height, int channels) {
   #pragma omp parallel for collapse(2)
   for (int i = 0; i < height; i++) {
      printf("The thread processing pixels in column %d is %d\n", i, omp_get_thread_num());
      for (int j = 0; j < width / 2; j++) {
         for (int c = 0; c < channels; c++) {
            int left_idx = (i * width + j) * channels + c;
            int right_idx = (i * width + (width - j - 1)) * channels + c;

            // Swap pixels
            unsigned char temp = img[left_idx];
            img[left_idx] = img[right_idx];
            img[right_idx] = temp;
         }
      }
   }
}

// Function to rotate the image by 90 degrees clockwise using two-pixel swaps
```

```
unsigned char* rotateImage90(unsigned char *img, int width, int height, int channels) {
  unsigned char *rotatedImg = (unsigned char*)malloc(width * height * channels);

  #pragma omp parallel for collapse(2)
  for (int i = 0; i < height; i++) {
    printf("The thread processing pixels in column %d is %d\n", i, omp_get_thread_num());
    for (int j = 0; j < width; j++) {
      for (int c = 0; c < channels; c++) {
        // Map the original pixel (i, j) to the new position (j, width - i - 1)
        rotatedImg[(j * height + (height - i - 1)) * channels + c] = img[(i * width + j) * channels + c];
      }
    }
  }
  return rotatedImg;
}

// Function to convert the image to grayscale (assuming 3 channels RGB)
void convertToGrayscale(unsigned char *img, int width, int height, int channels) {
  #pragma omp parallel for collapse(2)
  for (int i = 0; i < width * height; i++) {
    unsigned char r = img[i * channels];
    unsigned char g = img[i * channels + 1];
    unsigned char b = img[i * channels + 2];

    // Calculate grayscale value using luminosity method
    unsigned char gray = (unsigned char)(0.2989 * r + 0.5870 * g + 0.1140 * b);

    // Set all RGB channels to the grayscale value
    img[i * channels] = gray;
    img[i * channels + 1] = gray;
    img[i * channels + 2] = gray;
```

```
    }
}

int main() {
    srand(time(NULL));
    int width, height, channels;

    // Load image
    unsigned char *img = stbi_load("download1.jpeg", &width, &height, &channels, 0);
    printf("Image loaded: %dx%d, %d channels\n", width, height, channels);

    int choice;
    printf("Choose an operation:\n");
    printf("1. Flip Image Horizontally\n");
    printf("2. Rotate Image 90 Degrees\n");
    printf("3. Convert Image to Grayscale\n");
    printf("Enter your choice (1-3): ");
    scanf("%d", &choice);

    unsigned char *resultImg = NULL;
    double start, end;

    switch (choice) {
        case 1:
            start = omp_get_wtime();
            flipImageHorizontally(img, width, height, channels);
            end = omp_get_wtime();
            resultImg = img;
            printf("Image flipped horizontally.\n");
            break;
        case 2:
```

```
        start = omp_get_wtime();
        resultImg = rotateImage90(img, width, height, channels);
        end = omp_get_wtime();
        printf("Image rotated by 90 degrees.\n");
        // Swap width and height after rotation
        int temp = width;
        width = height;
        height = temp;
        break;
      case 3:
        start = omp_get_wtime();
        convertToGrayscale(img, width, height, channels);
        end = omp_get_wtime();
        resultImg = img;
        printf("Image converted to grayscale.\n");
        break;
      default:
        printf("Invalid choice.\n");
        return -1;
    }
    printf("Processing Time: %f seconds\n", end-start);
    stbi_write_jpg("processed_image.jpg", width, height, channels, resultImg, 100);

    printf("Image processing complete. Saved as 'processed_image.jpg'.\n");

    return 0;
}
```

**Test cases:**

**Input:**

## Output:

### Test 1:

```
PS D:\College\Sem5\Parallel computing> ./a.exe
Image loaded: 254x180, 3 channels
Choose an operation:
1. Flip Image Horizontally
2. Rotate Image 90 Degrees
3. Convert Image to Grayscale
Enter your choice (1-3): 1
The thread processing pixels in column 158 is 7
The thread processing pixels in column 159 is 7
The thread processing pixels in column 160 is 7
The thread processing pixels in column 161 is 7
The thread processing pixels in column 162 is 7
The thread processing pixels in column 163 is 7
The thread processing pixels in column 164 is 7
The thread processing pixels in column 165 is 7
The thread processing pixels in column 166 is 7
The thread processing pixels in column 92 is 4
The thread processing pixels in column 136 is 6
The thread processing pixels in column 137 is 6
The thread processing pixels in column 138 is 6
The thread processing pixels in column 139 is 6
The thread processing pixels in column 140 is 6
The thread processing pixels in column 141 is 6
```

### Test 2:

```
PS D:\College\Sem5\Parallel computing> ./a.exe
Image loaded: 254x180, 3 channels
Choose an operation:
1. Flip Image Horizontally
2. Rotate Image 90 Degrees
3. Convert Image to Grayscale
Enter your choice (1-3): 2
The thread processing pixels in column 158 is 7
The thread processing pixels in column 92 is 4
The thread processing pixels in column 0 is 0
The thread processing pixels in column 46 is 2
The thread processing pixels in column 136 is 6
The thread processing pixels in column 137 is 6
The thread processing pixels in column 69 is 3
The thread processing pixels in column 70 is 3
The thread processing pixels in column 159 is 7
The thread processing pixels in column 93 is 4
The thread processing pixels in column 1 is 0
The thread processing pixels in column 2 is 0
The thread processing pixels in column 3 is 0
The thread processing pixels in column 4 is 0
The thread processing pixels in column 23 is 1
The thread processing pixels in column 24 is 1
```

**Test 3:**

```
PS D:\College\Sem5\Parallel computing> ./a.exe
Image loaded: 254x180, 3 channels
Choose an operation:
1. Flip Image Horizontally
2. Rotate Image 90 Degrees
3. Convert Image to Grayscale
Enter your choice (1-3): 3
Image converted to grayscale.
Image processing complete. Saved as 'processed_image.jpg'.
```

## 10. Time complexity analysis:

- **Flip Image Horizontally:**
  - **Parallel Complexity:** O(height * (width / 2) * channels / P)
    - Where P is the number of available processors/threads.
  - **Speedup:** Ideally, linear speedup with P threads, reducing the processing time by a factor of P.
  - T_flip = O(m * (n/2) * c) = O(m * n * c / p)

- **Rotate Image 90 Degrees Clockwise:**
  - **Parallel Complexity:** O((height * width * channels) / P)
    - Each thread handles a portion of the pixel mappings.
  - **Speedup:** Close to linear speedup with P threads, as each pixel operation is independent.
  - T_rotate = O(m * (n/2) * c) = O(m * n * c / p)

- **Convert Image to Grayscale:**
  - **Parallel Complexity:** O((width * height * channels) / P)
    - Each thread processes a subset of the pixels.
  - **Speedup:** Near-linear speedup with P threads due to independent pixel processing.
  - T_gray = O(m * (n/2) * c) = O(m * n * c / p)

## Overall Time Complexity

For a single image of size m x n with c color channels, the overall time complexity for all operations when performed sequentially is:

- T_total = T_flip + T_rotate + T_gray

- T_total = O(m * n * c / p) + O(m * n * c / p) + O(m * n * c / p)

We can simplify the overall complexity:

- T_total = O(m * n * c / p)

## Asymptotic Complexity

- Parallel: O(m * n * c / p)

Results(Parallel):

| Image size | Function | Time (seconds) |
|---|---|---|
| 297x169 | Flip image horizontally | 0.001647 |
| | Rotate by 90° | 0.005666 |
| | Convert image to grayscale | 0.001529 |
| | Flip image horizontally | 0.012228 |
| | Rotate by 90° | 0.023003 |
| | Convert image to grayscale | 0.003872 |