# Concurrency and Parallelism Project

Guilherme Fernandes
Vladyslav Mikytiv
60045
60735

## ABSTRACT

This report provides the explanation and the implementation of parallelisation of the algorithm of equalization of images. This is not the final report it's just a middle checkpoint.

## KEYWORDS

OpenMP Parallelism C++

## 1 CRITICAL CODE ANALYSIS AND PARALLELIZATION

In this section we will analyse the behaviour of our code and identify the critical hotspots.

### 1.1 The first steps

At first we ran the code without doing any modifications and used the profiler in order to identify the hot spots. The zones of the code that took a long time to perform the computation and in overall slow down the algorithm are: the **`normalization`** of the image, the **`correction of color`**, the **`convertion to the greyscale`** and some other not so computational heavy functions.

To reduce the impact on processing time caused by these functions within the algorithm, we integrated OpenMP directives into these specific parts of the code to accelerate computation. Prior to this optimization, we structured the code by creating separate functions for each step of the algorithm to maintain organization.

It's important to highlight the **thread management**. Since we are working with images we will have to do some simples calculus to determine the amount of work that each thread will be given. For that we must perform two calculations (one for RGB and one for the grey scale). For the RGB we will calculate `WIDTH * HEIGHT * 3` and to obtain the `CHUNK_SIZE_RGB` we just divide `WIDTH * HEIGHT * 3` by `N_THREADS`. For the grey scale it's similar but we don't multiply by 3. Now we will have the work that will be balanced between threads.

### 1.2 Function parallelization

The **normalization** function will be improved with the following code:

```
void normalize(//omitting for space) {
    #pragma omp parallel for schedule(dynamic, chunk_size_channels) num_threads(n_threads)
    for (int i = 0; i < size_channels; i++)
        uchar_image[i] = (unsigned char) (255 * input_image_data[i]);
}
```

**Listing 1: Normalization Function**

The **grey scale conversion** will be improved in two ways. We will mix the `fill_histogram` function with this one in order to do everything in the same function. Besides that we also apply OpenMP directives.

```
void convertoToGrayScale(//omitting for space) {
    // filling the histogram with zeroes
    #pragma omp parallel for reduction(+:histogram) num_threads(n_threads)
    for (int i = 0; i < size; i++){
            auto r = uchar_image[3 * i];
            auto g = uchar_image[3 * i + 1];
            auto b = uchar_image[3 * i + 2];
            gray_image[i] =
            static_cast<unsigned char>
            (0.21 * r + 0.71 * g + 0.07 * b);
            histogram[gray_image[i]]++;
        }
}
```

**Listing 2: Grey Conversion Function**

The function that **calculated the CDF** doesn't require any type of parallelization. It's a simple for loop that executes 256 iterations every time. Another function that we managed to simplify was the `cdf_min_loop`. Since we minimum will always be on the first position we just return it and that's how we compute the minimum of the CDF.

The **`correct_color_loop`** function was parallelized with the following directives:

```
void correct_color_loop(//omitting for space) {
    #pragma omp parallel for schedule(static, chunk_size_channels) num_threads(n_threads)
    for (int i = 0; i < size_channels; i++)
        uchar_image[i] = correct_color
        (cdf[uchar_image[i]], cdf_min);
  }
```

**Listing 3: Color Correction Function**

The last function to be parallelized is **`rescale`**. And it was also changed accordingly with the OpenMP directives.

```
void rescale(//omitting for space) {
    #pragma omp parallel for schedule(dynamic, chunk_size_channels) num_threads(n_threads)
        for (int i = 0; i < size_channels; i++)
            output_image_data[i] = static_cast<float>
            (uchar_image[i]) / 255.0f;
}
```

**Listing 4: Rescale Function**

## 2 METRIC ANALYSIS

Now, let's assess the impact of these changes on the runtime of our program. For that we will use **speed up** and **efficiency** and we will execute every executing multiple times in order to get the mean value of the execution times.

The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below.

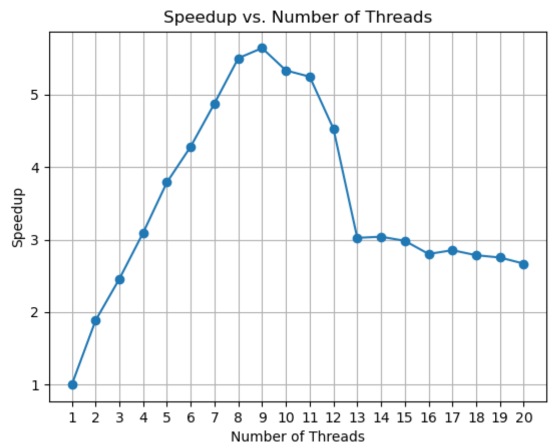


**Figure 1: 1907 Franklin Model D roadster. Photograph by Harris & Ewing, Inc. [Public domain], via Wikimedia Commons. (https://goo.gl/VLCRBB).**

Your figures should contain a caption which describes the figure to the reader. It's important to highlight the **thread management**. Since we are working with images we will have to do some simples calculus to determine the amount of work that each thread will be given. For that we must perform two calculations (one for RGB and one for the grey scale). For the RGB we will calculate `WIDTH * HEIGHT * 3` and to obtain the `CHUNK_SIZE_RGB` we just divide `WIDTH * HEIGHT * 3` by `N_THREADS`. For the grey scale it's similar but we don't multiply by 3. Now we will have the work that will be balanced between threads. It's important to highlight the **thread management**. Since we are working with images we will have to do some simples calculus to determine the amount of work that each thread will be given. For that we must perform two calculations (one for RGB and one for the grey scale). For the RGB we will calculate `WIDTH * HEIGHT * 3` and to obtain the `CHUNK_SIZE_RGB` we just divide `WIDTH * HEIGHT * 3` by `N_THREADS`. For the grey scale it's similar but we don't multiply by 3. Now we will have the work that will be balanced between threads. It's important to highlight the **thread management**. Since we are working with images we will have to do some simples calculus to determine the amount of work that each thread will be given. For that we must perform two calculations (one for RGB and one for the grey scale). For the RGB we will calculate `WIDTH * HEIGHT * 3` and to obtain the `CHUNK_SIZE_RGB` we just divide `WIDTH * HEIGHT * 3` by `N_THREADS`. For the grey scale it's similar but we don't multiply by 3. Now we will have the work that will be balanced between threads. It's important to highlight the **thread management**. Since we are working with images we will have to do some simples calculus to determine the amount of work that each thread will be given. For that we must perform two calculations (one for RGB and one for the grey scale). For the RGB we will calculate `WIDTH * HEIGHT * 3` and to obtain the `CHUNK_SIZE_RGB` we just divide `WIDTH * HEIGHT * 3` by `N_THREADS`. For the grey scale it's similar but we don't multiply by 3. Now we will have the work that will be balanced between threads. It's important to highlight the **thread management**. Since we are working with images we will have to do some simples calculus to determine the amount of work that each thread will be given. For that we must perform two calculations (one for RGB and one for the grey scale). For the RGB we will calculate `WIDTH * HEIGHT * 3` and to obtain the `CHUNK_SIZE_RGB` we just divide `WIDTH * HEIGHT * 3` by `N_THREADS`. For the grey scale it's similar but we don't multiply by 3. Now we will have the work that will be balanced between threads.

The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below. The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below. The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below. The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below. The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below. The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below. The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below. The "figure" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below.

### 2.1 Q&A

`TODO` Questions to ask the teacher.