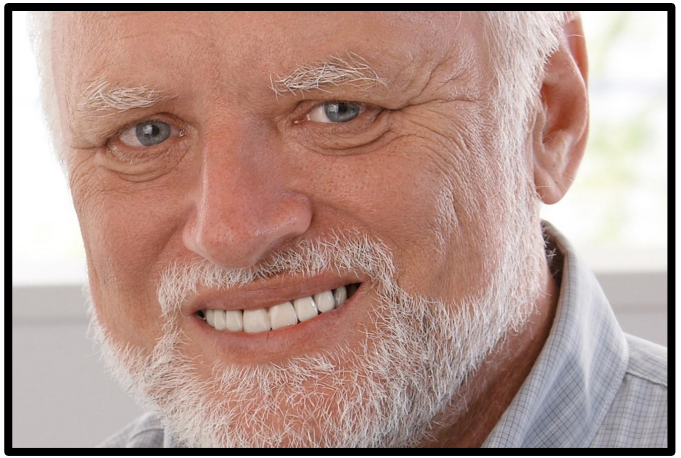
Gruppe 04

Grauwert, Histogramm und Helligkeit

Enrico Gamil Toros de Chadarevian & Sonja Albers

Gliederung

1. Aufgabenstellung
2. Berechnung
3. Parallelisierung
4. Auswertung
5. Code Ausführung
6. Fazit



1. Aufgabenstellung

Bild unter Parallelisierung …

… in **Grauwertbild** umwandeln

… **heller/ dunkler** machen

… als **Histogramm** darstellen

2. Parallelisierung

- Java mit ExecutorService statt OpenMP
- Jeweils Blocking und Non-Blocking

```java
public class BrightnessBlockingTask extends BrightnessTask implements Runnable
{
    int[][] imgRgbArray;
    @Override
    public void run() {
        int[] row = brightnessTask(imgRgbArray, rowIndex, brightness);
        synchronized (image) {
            ImageUtils.setRgbRow(image, rowIndex, row);
        }
    }
}
```

```java
public class BrightnessNonBlockingTask extends BrightnessTask implements Runnable {
    int[][] imgRgbArray;
    Map<Integer, int[]> results;
    @Override
    public void run() {
        int[] row = brightnessTask(imgRgbArray, rowIndex, brightness);
        results.put(this.rowIndex, row);
    }
}
```

2. Parallelisierung

```
// 1. Read Image
public static int[][] imageToRgbArray(@Nonnull BufferedImage image) {
        int width = image.getWidth();
        int height = image.getHeight();
        int[][] array = new int[width][height];
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                array[x][y] = image.getRGB(x, y);
            }
        }
        return array;
}
```

2. Parallelisierung

```
// 2. Make Tasks for each Row of the image
protected ExecutorService makeTask() {
   ExecutorService executor = Executors.newFixedThreadPool(getThreadPoolSize());
   for (int row = 0; row < getImage().getHeight(); row++) {
           executor.execute(new ImageTask(getImgRgbArray(), row, getImage()));
   }
   return executor;
}
```

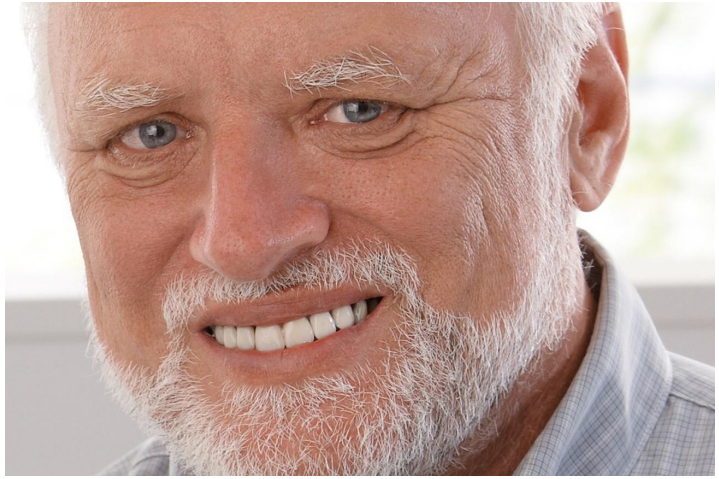
Blocking

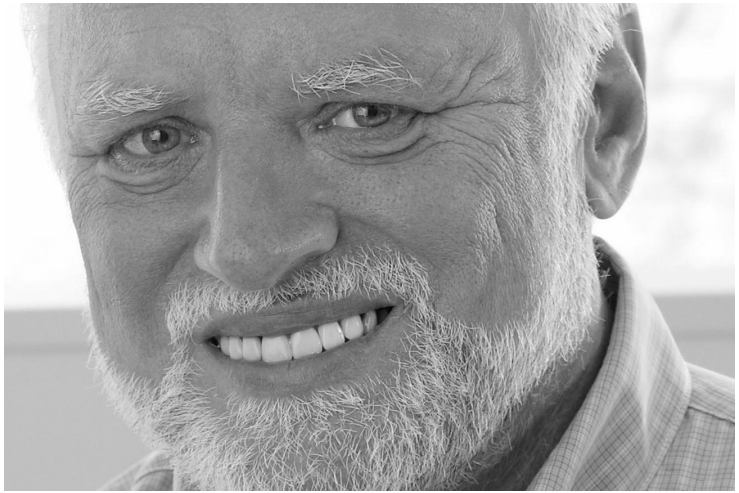
```
public class BrightnessBlockingTask extends BrightnessTask implements Runnable {
    int[][] imgRgbArray;

    @Override

    public void run() {

        int[] row = brightnessTask(imgRgbArray, rowIndex, brightness);

        synchronized (image) {

            ImageUtils.setRgbRow(image, rowIndex, row);

        }

    }
}
```

Non-Blocking

```
public class BrightnessNonBlockingTask extends BrightnessTask implements Runnable {
    int[][] imgRgbArray;

    Map<Integer, int[]> results;

    @Override

    public void run() {

        int[] row = brightnessTask(imgRgbArray, rowIndex, brightness);

        results.put(this.rowIndex, row);

    }
}
```

3. Berechnung - Grauwertbild

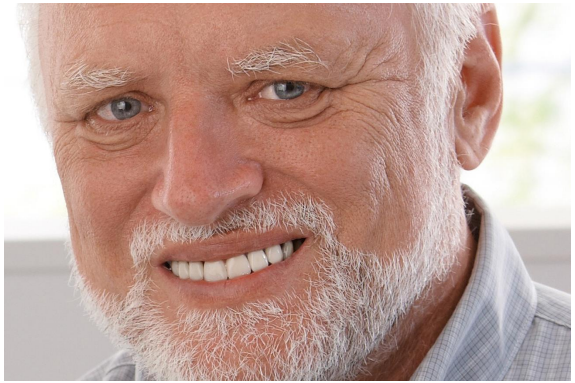


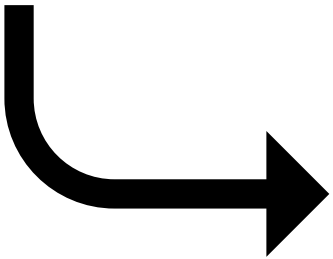


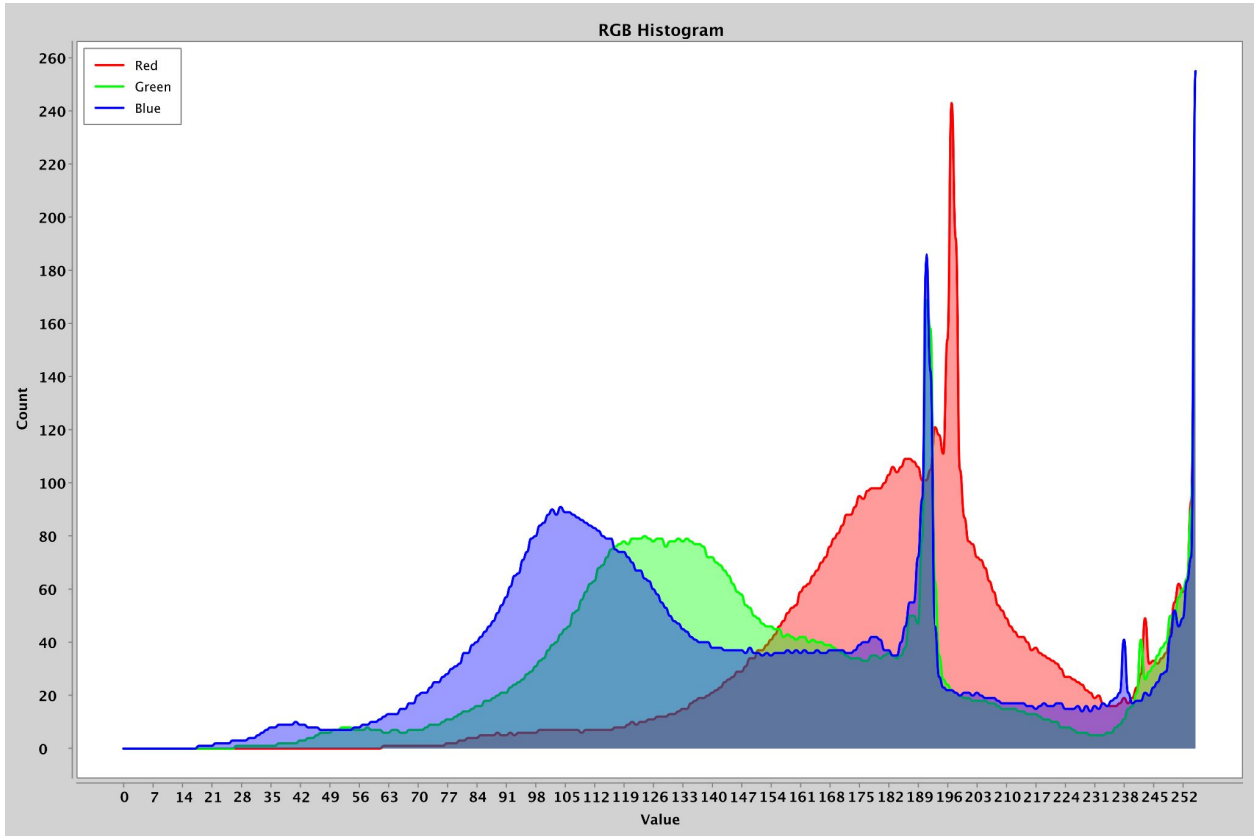
3. Berechnung – Grauwertbild

```
public int[] greyscaleTask(int[][] imgRgbArray, int rowIndex) {
        int[] row = new int[imgRgbArray.length];
        for (int i = 0; i < row.length; i++) {
            int pixel = imgRgbArray[i][rowIndex];
            int red = (pixel >> 16) & 0xFF;
            int green = (pixel >> 8) & 0xFF;
            int blue = (pixel) & 0xFF;
            int rgbRes = (int) (red * 0.21 + green * 0.72 + blue * 0.07);
            row[i] = ((0xFF) << 24) | // alpha not needed
                    ((rgbRes & 0xFF) << 16) |
                    ((rgbRes & 0xFF) << 8) |
                    ((rgbRes & 0xFF));
        }
        return row;
}
```

3. Berechnung – Histogramm



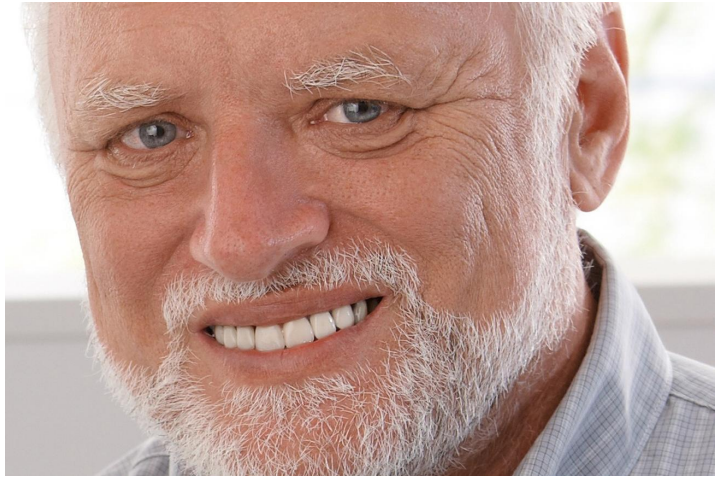


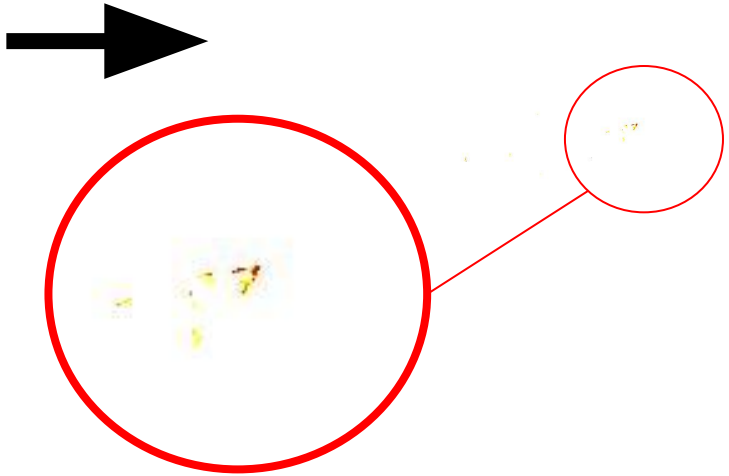


3. Berechnung - Histogramm

```
public int[] histogramTask(int[][] imgRgbArray, int rowIndex,
                int[] redBucket, int[] greenBucket, int[]
                blueBucket) {
        for (int[] rows : imgRgbArray) {
            Color c = new Color(rows[rowIndex]);
            this.redBucket[c.getRed()]++;
            this.greenBucket[c.getGreen()]++;
            this.blueBucket[c.getBlue()]++;
        }
}
```

3. Berechnung - Helligkeit





60.00 als Eingabewert

3. Berechnung – Helligkeit

```java
public int[] brightnessTask(int[][] imgRgbArray, int rowIndex, double brightness) {
        int[] row = new int[imgRgbArray.length];
        for (int i = 0; i < row.length; i++) {
            int pixel = imgRgbArray[i][rowIndex];
            int red = (pixel >> 16) & 0xFF;
            int green = (pixel >> 8) & 0xFF;
            int blue = (pixel) & 0xFF;
            // increase brightness
            red *= brightness;
            green *= brightness;
            blue *= brightness;
            // truncate
            red = Math.min(Math.max(red, 0), 255);
            green = Math.min(Math.max(green, 0), 255);
            blue = Math.min(Math.max(blue, 0), 255);
            row[i] = (red << 16) | (green << 8) | blue;
        }
        return row;
}
```

4. Auswertung

- jeweils 1000 Ausführungen
- 1 bis 8 Threads
- nur IntelliJ geöffnet
- Ausführung auf:

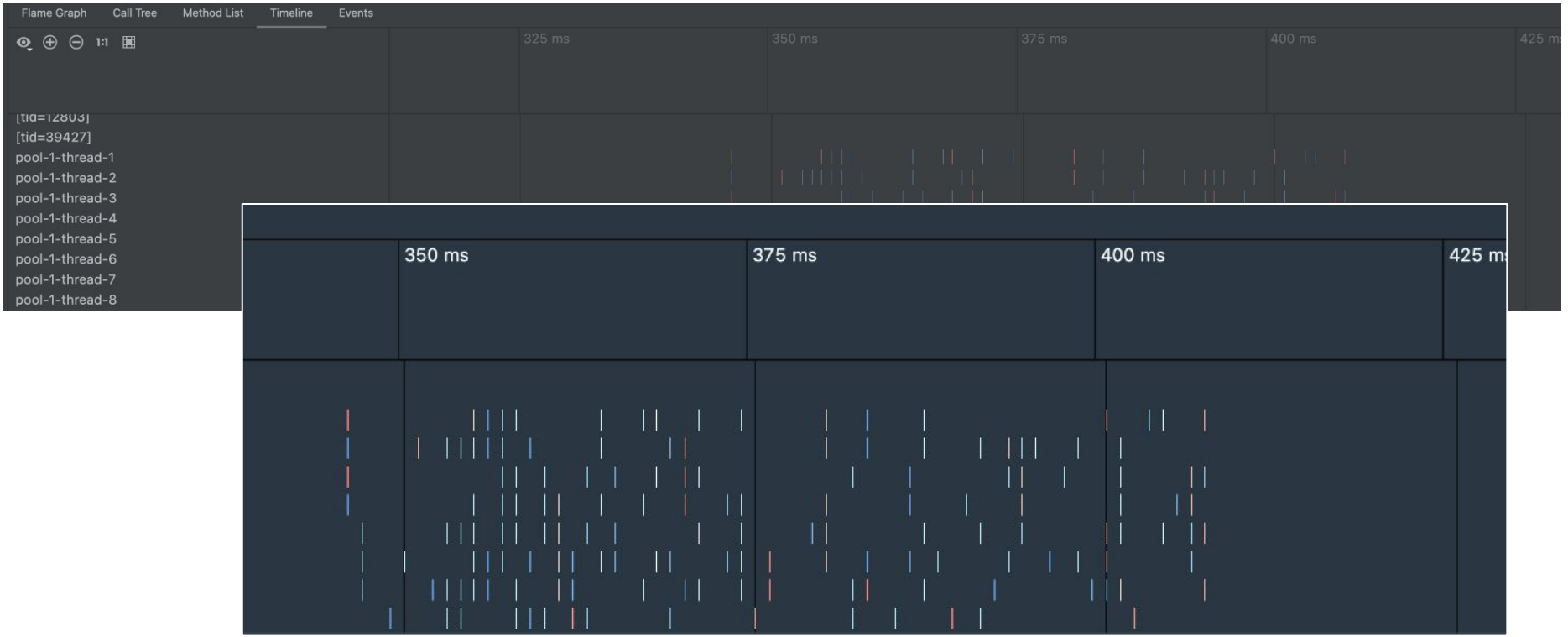
MacBook Pro (2015)

2,9 GHz Dual-Core Intel Core i5 Prozessor
8 GB Speicher

MacBook Pro (2020)

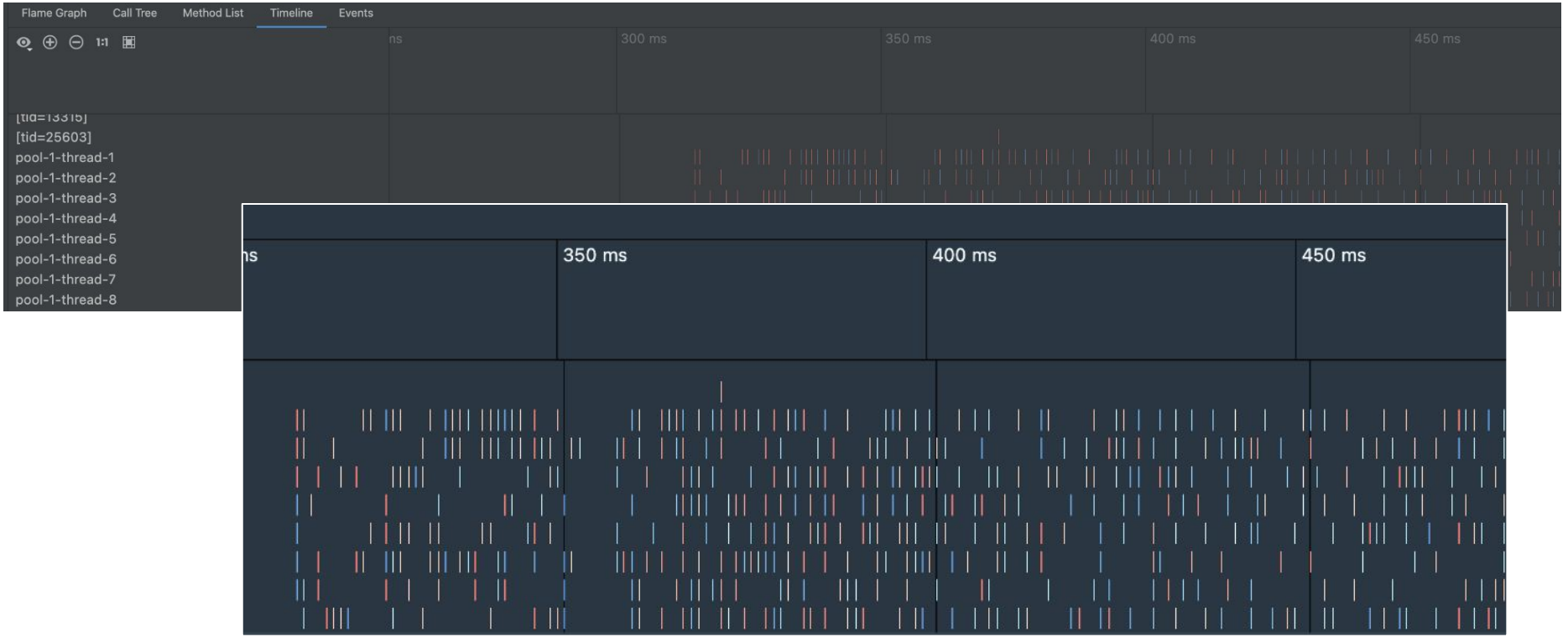
Apple M1 Chip
16 GB Speicher

4. Auswertung – Non-Blocking

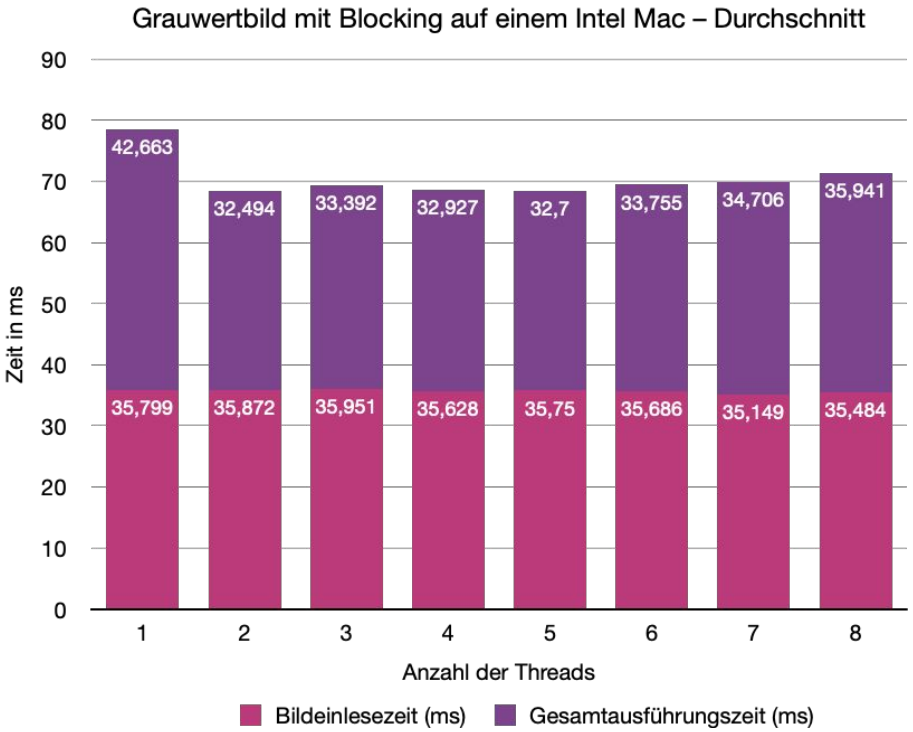


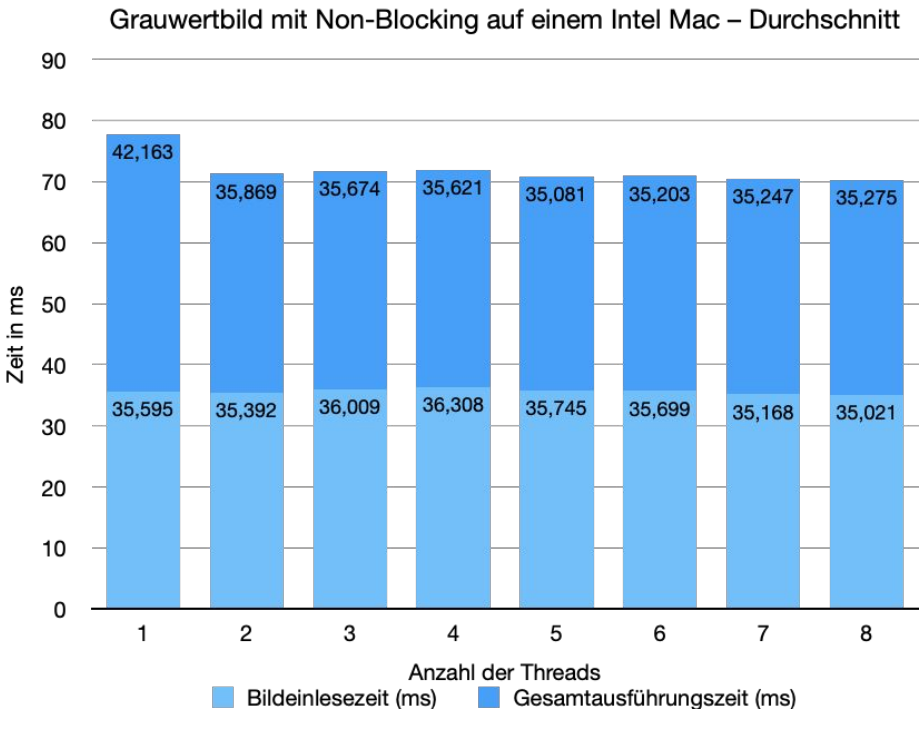
https://www.jetbrains.com/help/idea/2022.3/read-the-profiling-report.html#profiler-timeline

4. Auswertung - Blocking

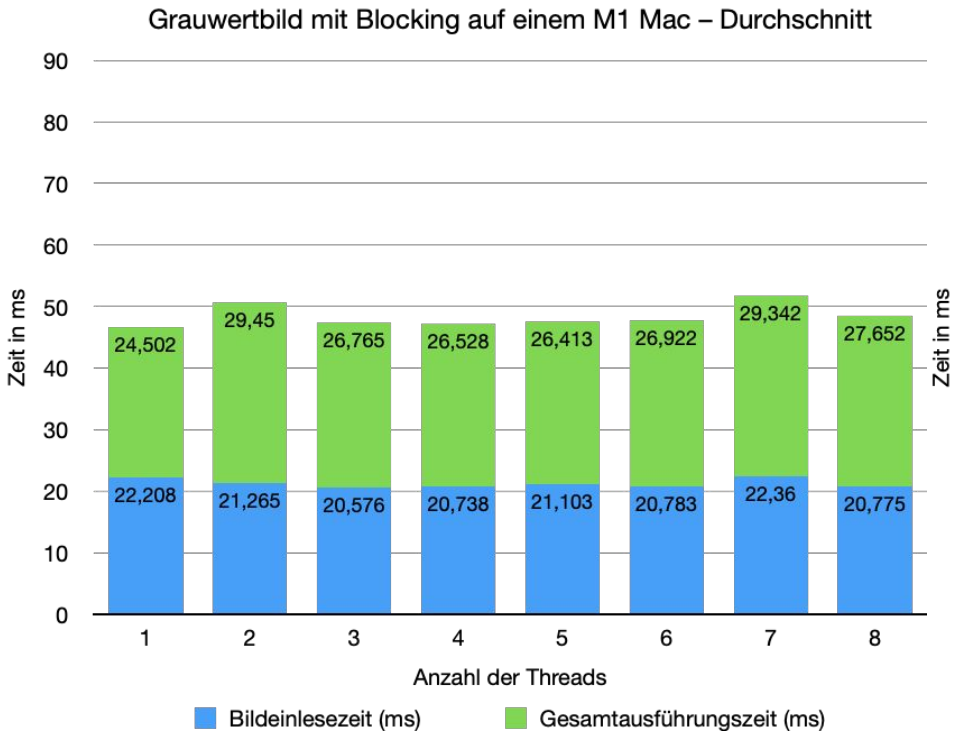


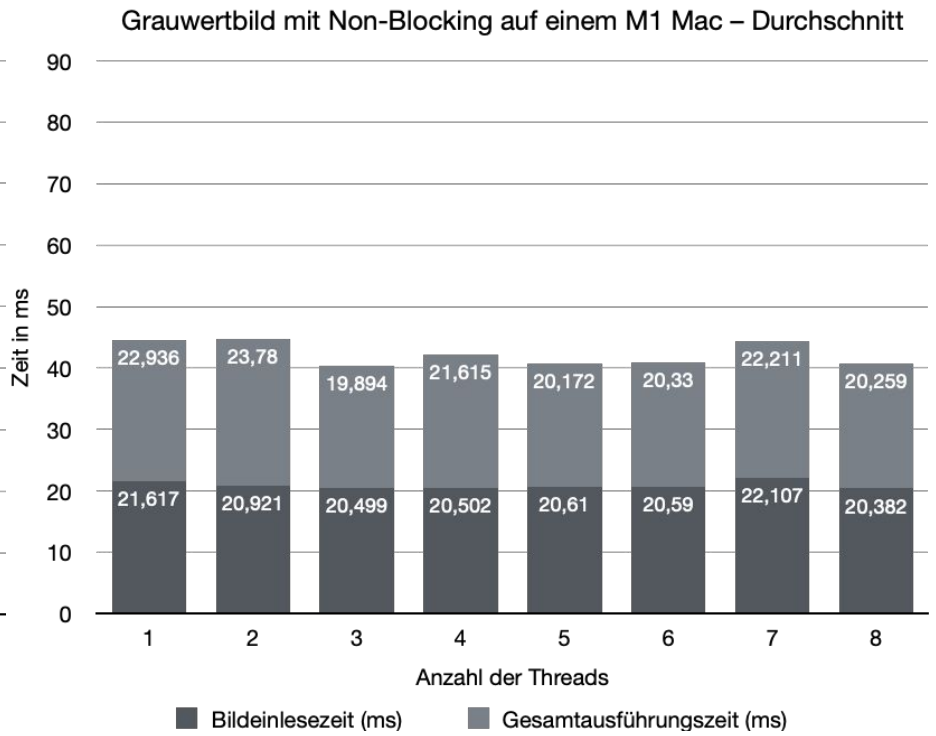
4.1 Grauwert – (Non-)Blocking



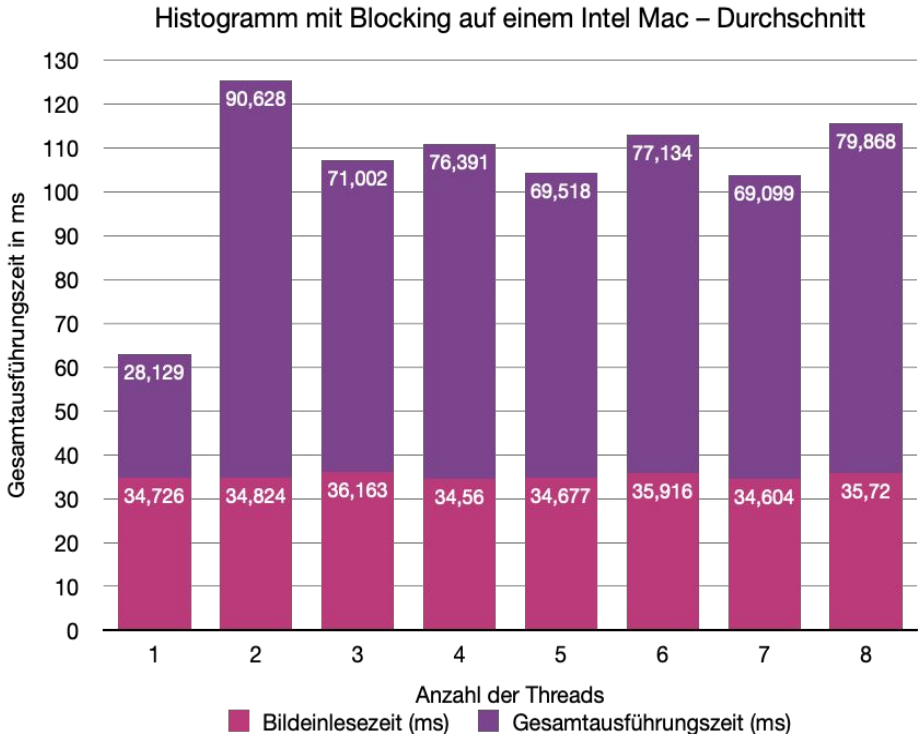


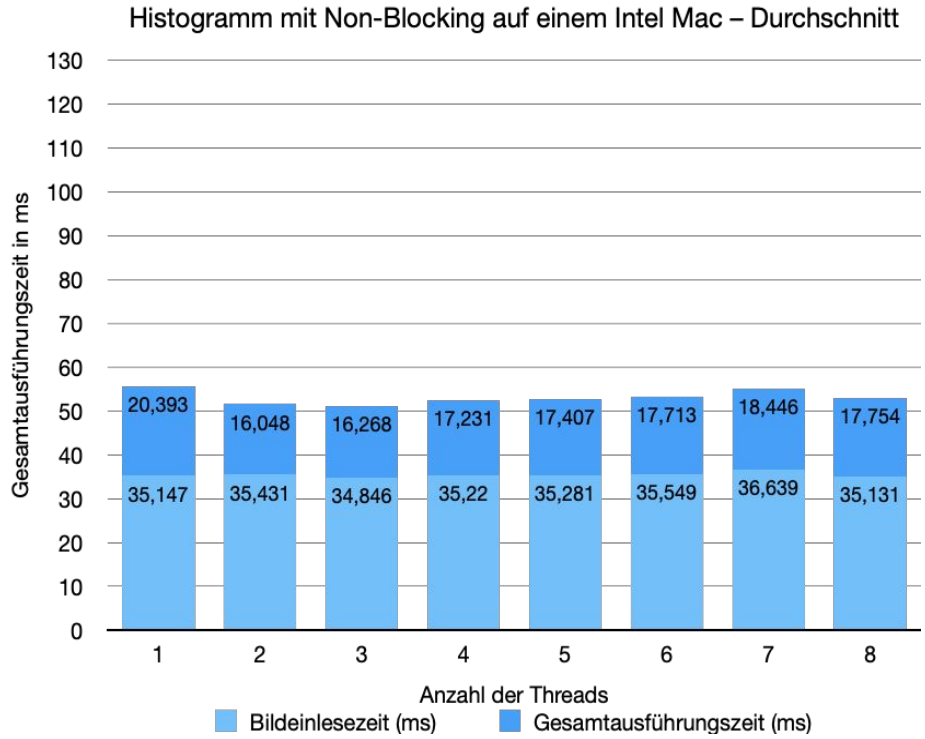
4.1 Grauwert – (Non-)Blocking



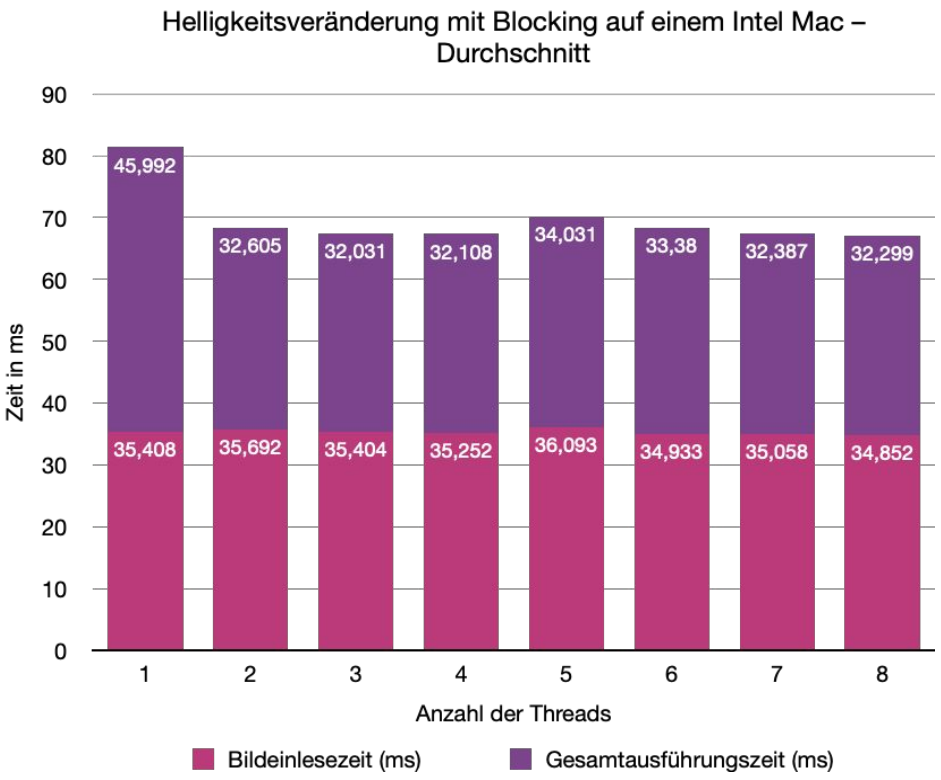


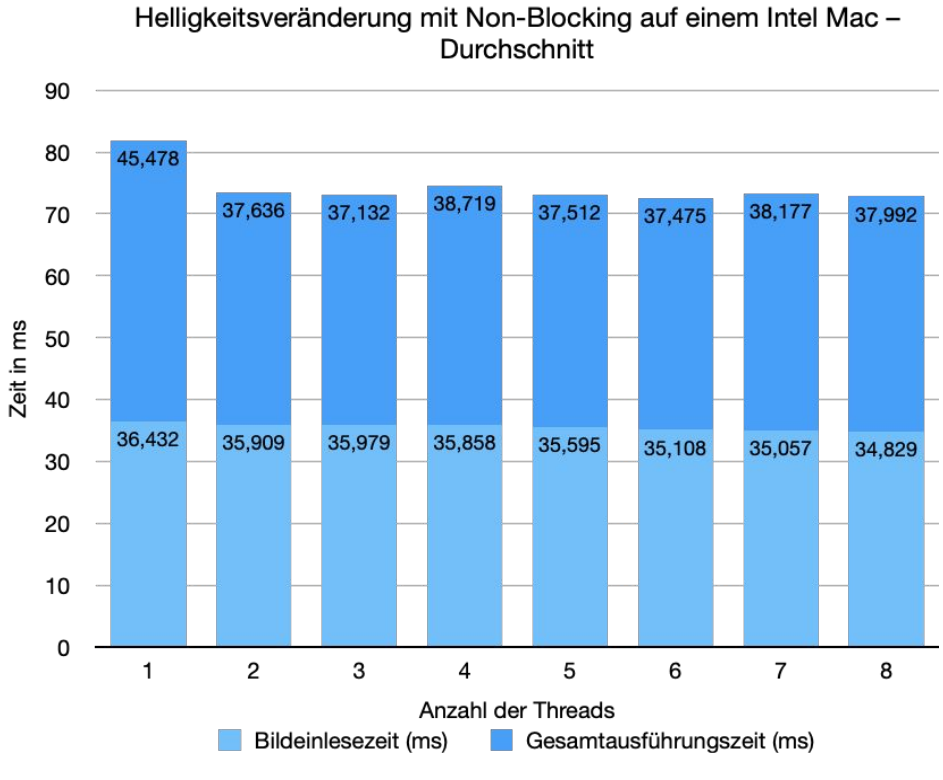
4.2 Histogramm – (Non-)Blocking





4.3 Helligkeit – (Non-)Blocking





Zeit für Code

6. Fazit

- M1 schneller als Intel i5
- Non-Blocking bei Histogramm schneller
- allgemein keine großen Unterschiede zwischen Non-Blocking und Blocking
- Versuche mit mehr Threads nötig!

Danke für Eure Zeit :)