Platformy programistyczne .Net i Java

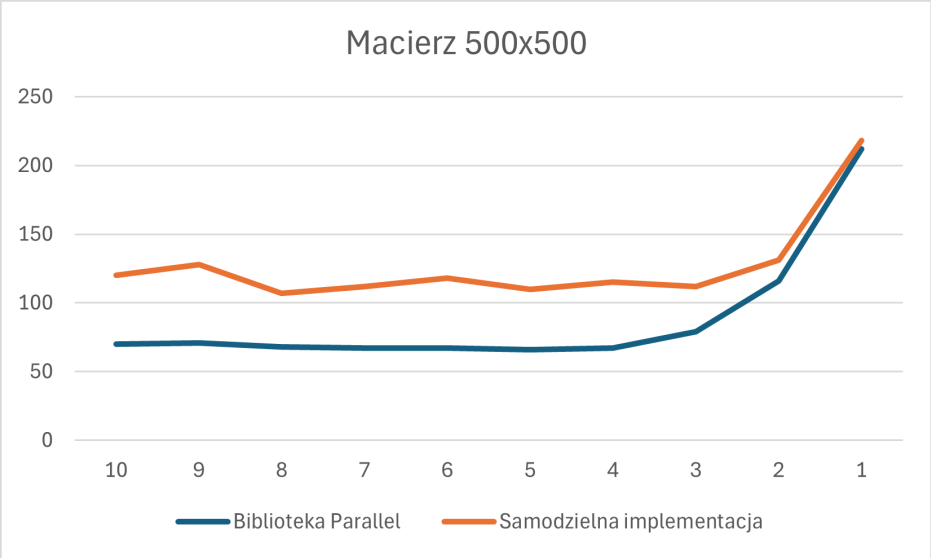
Laboratorium 3

Filip Zioło (272543)

1 Wstęp

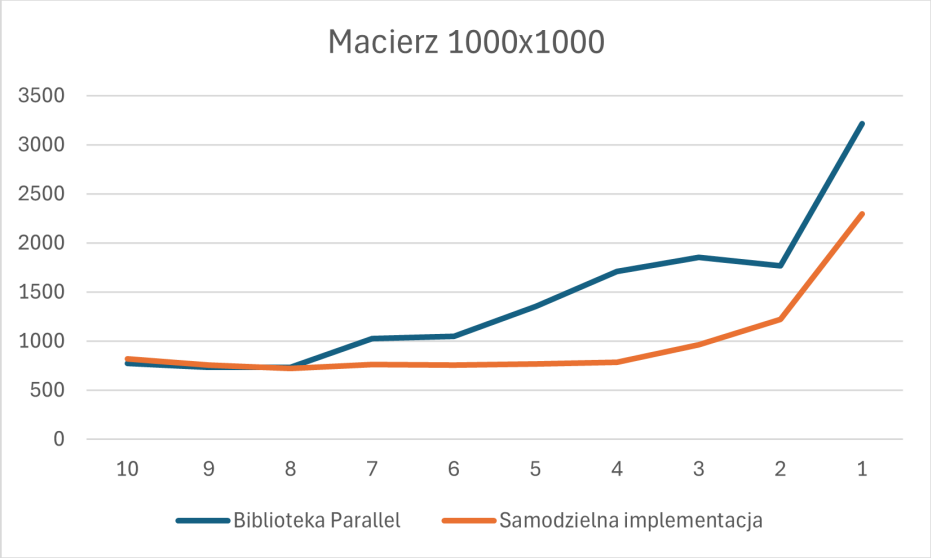
Zadanie polegało na utworzeniu programu który pozwalał nam na użycie funkcji z biblioteki parallel oraz utworzeniu wątków samemu. Następnie mielismy stworzyć aplikację okienkową która pozwalała nam przetwarzać obrazy 4 róznymi filtrami. Dodatkowym aspektem tego ćwiczenia było przeprowadzenie badań.

2 Badania



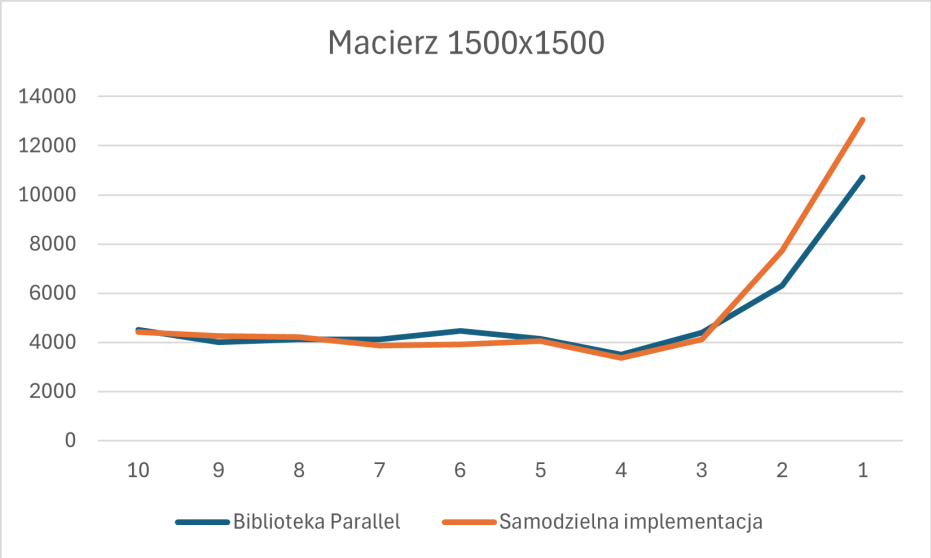
Rysunek 1: Wykres czasu od ilości wątków

Jak widzimy na powyższym wykresie samodzielna implementacją wątków jest metodą zajmującą średnio większą ilość czasu. Różnica ta wynika z tego, że inicjalizacja wątków trwa dłużej kiedy robimy to samemu. Macierz jest mała więc samodzielna implementacja nie daje rady być szybsza.



Rysunek 2: Wykres czasu od ilości wątków

Tutaj samodzielna implementacja jest szybsza ponieważ macierz jest już większa więc czas potrzebny na implementacje.



Rysunek 3: Wykres czasu od ilości wątków

Dla największej macierzy implementacje są bardzo podobne ponieważ różnica w szybkości wyrównuje się dlatego, że biblioteka Parallel lepiej dzieli kolumny/wiersze które przysługują danemu wątku.

3 Dokumentacja

```
1
2 namespace Imgg_proccesing
3 {
```

```
    public partial class Form1 : Form
    {
        private Bitmap img;

        public Form1()
        {
            InitializeComponent();
            pictureBoxOriginal.SizeMode = PictureBoxSizeMode.Zoom;
            pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;
            pictureBox2.SizeMode = PictureBoxSizeMode.Zoom;
            pictureBox3.SizeMode = PictureBoxSizeMode.Zoom;
            pictureBox4.SizeMode = PictureBoxSizeMode.Zoom;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "jpg files (*.jpg)|*.jpg|All files
               (*.*)|*.*";
            openFileDialog1.FilterIndex = 1;

            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                img = new Bitmap(openFileDialog1.FileName);
                pictureBoxOriginal.Image = img;
            }
        }

        private async void button2_ClickAsync(object sender, EventArgs e
           )
        {
            if (img == null) return;

            Bitmap originalCopy1 = new Bitmap(img);
            Bitmap originalCopy2 = new Bitmap(img);
            Bitmap originalCopy3 = new Bitmap(img);
            Bitmap originalCopy4 = new Bitmap(img);

            var task1 = Task.Run(() => ApplyGrayscale(originalCopy1));
            var task2 = Task.Run(() => ApplyNegative(originalCopy2));
            var task3 = Task.Run(() => ApplyThreshold(originalCopy3));
            var task4 = Task.Run(() => ApplyMirror(originalCopy4));

            await Task.WhenAll(task1, task2, task3, task4);

            pictureBox1.Image = task1.Result;
            pictureBox2.Image = task2.Result;
            pictureBox3.Image = task3.Result;
            pictureBox4.Image = task4.Result;
        }
        private Bitmap ApplyGrayscale(Bitmap bmp)
```

```
        {
            for (int y = 0; y < bmp.Height; y++)
            {
                for (int x = 0; x < bmp.Width; x++)
                {
                    Color c = bmp.GetPixel(x, y);
                    int avg = (c.R + c.G + c.B) / 3;
                    bmp.SetPixel(x, y, Color.FromArgb(avg, avg, avg));
                }
            }
            return bmp;
        }

        private Bitmap ApplyNegative(Bitmap bmp)
        {
            for (int y = 0; y < bmp.Height; y++)
            {
                for (int x = 0; x < bmp.Width; x++)
                {
                    Color c = bmp.GetPixel(x, y);
                    bmp.SetPixel(x, y, Color.FromArgb(255 - c.R, 255 - c
                       .G, 255 - c.B));
                }
            }
            return bmp;
        }

        private Bitmap ApplyThreshold(Bitmap bmp)
        {
            for (int y = 0; y < bmp.Height; y++)
            {
                for (int x = 0; x < bmp.Width; x++)
                {
                    Color c = bmp.GetPixel(x, y);
                    int avg = (c.R + c.G + c.B) / 3;
                    bmp.SetPixel(x, y, avg < 128 ? Color.Black : Color.
                       White);
                }
            }
            return bmp;
        }

        private Bitmap ApplyMirror(Bitmap bmp)
        {
            int w = bmp.Width;
            int h = bmp.Height;

            for (int y = 0; y < h; y++)
            {
                for (int x = 0; x < w / 2; x++)
```

```
                {
                    Color left = bmp.GetPixel(x, y);
                    Color right = bmp.GetPixel(w - x - 1, y);
                    bmp.SetPixel(x, y, right);
                    bmp.SetPixel(w - x - 1, y, left);
                }
            }

            return bmp;
        }
    }
}
```

Klasa nakładająca filtry na obraz.

```
public class MatrixMultiplier
    {
        public static Matrix Multiply(Matrix A, Matrix B, int numThreads
            )
        {
            int size = A.Size;
            Matrix result = new Matrix(size);

            ParallelOptions options = new ParallelOptions()
            {
                MaxDegreeOfParallelism = numThreads
            };

            Parallel.For(0, size, options, i =>
            {
                for (int j = 0; j < size; j++)
                {
                    int sum = 0;
                    for (int k = 0; k < size; k++)
                        sum += A.Data[i, k] * B.Data[k, j];
                    result.Data[i, j] = sum;
                }
            });

            return result;
        }
    }
```

Główna funkcja do mnożenia macierzy.

```
public class Matrix
    {
        public int[,] Data { get; private set; }
        public int Size { get; private set; }

        public Matrix(int size, bool randomize = false)
        {
```

```
8               Size = size;
9               Data = new int[size, size];
10              if (randomize)
11                  FillRandom();
12          }
13
14          private void FillRandom()
15          {
16              Random rand = new Random();
17              for (int i = 0; i < Size; i++)
18                  for (int j = 0; j < Size; j++)
19                      Data[i, j] = rand.Next(1, 10);
20          }
21
22          public override string ToString()
23          {
24              var sb = new System.Text.StringBuilder();
25              for (int i = 0; i < Size; i++)
26              {
27                  for (int j = 0; j < Size; j++)
28                      sb.Append(Data[i, j] + "\t");
29                  sb.AppendLine();
30              }
31              return sb.ToString();
32          }
33      }
```

Funkcja tworząca dwie macierze i zwracająca je do stringa