Politechnika Wrocławska

Dokumentacja
Platformy programistyczne .NET i Java

25 kwietnia 2024

Filip Mirski nr indeksu 264267

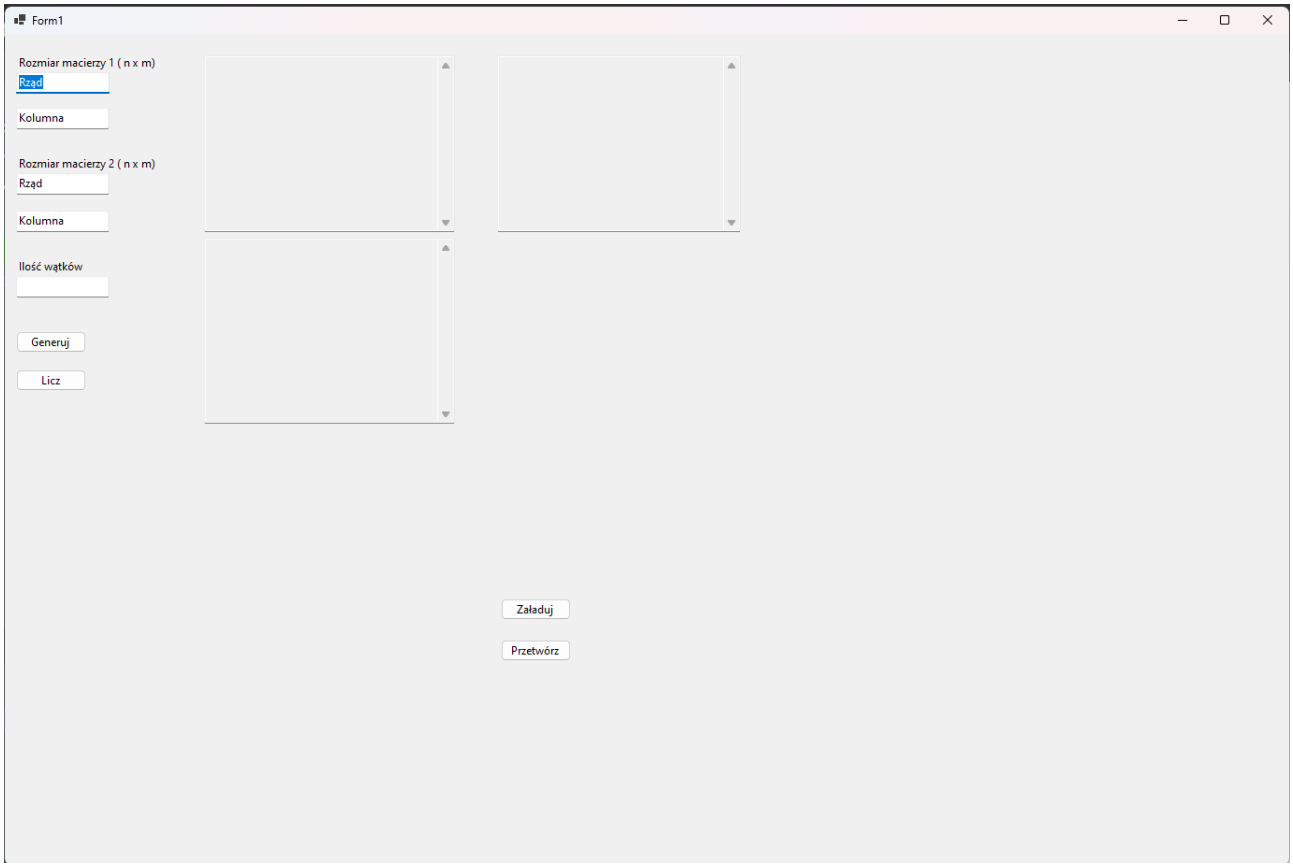
1 Cel zadania

Celem laboratorium było zapoznanie się z podstawami przetwarzania wielowątkowego w technologii .NET. W ramach zajęć należało stworzyć program w języku C#, który pozwoli na przyspieszenie obliczeń poprzez zrównoleglenie kodu.. Do tego została użyta funkcja wysokiego poziomu i niskiego w celu porównania wyników.

2 Zadania

2.1 Zadanie 1 - Wielowątkowe mnożenie macierzy z wykorzystaniem klasy Thread.

W GUI, które zostało stworzone specjalnie na potrzeby wykonania tego zadania należało wybrać jakiego rozmiaru macierze chcemy pomnożyć.



Rysunek 1: GUI do wykonanego zadania

Po wybraniu rozmiarów macierzy należało nacisnąć przycisk "Generuj", którego logika jest przedstawiona poniżej:

```
private void buttonMatrixGenerate_Click(object sender, EventArgs e)
{
    int row1 = int.Parse(textBoxMatrixRow1.Text);
    int column1 = int.Parse(textBoxMatrixColumn1.Text);
    int row2 = int.Parse(textBoxMatrixRow2.Text);
    int column2 = int.Parse(textBoxMatrixColumn2.Text);

    matrixMultiplier = new MatrixMultiplier(row1, column1, row2, column2
        );

    matrixMultiplier.GenerateMatrix(row1, column1, row2, column2);

    this.matrix1 = matrixMultiplier.GetMatrix1();
    this.matrix2 = matrixMultiplier.GetMatrix2();
```

Funkcja generowania macierzy jest natomiast następująca:

```
public void GenerateMatrix(int row1, int column1, int row2, int column2)
{
    this.matrix1 = new int[row1, column1];
    this.matrix2 = new int[row2, column2];

    this.resultMatrix = new int[row1, column2];

    for (int i = 0; i < row1; i++)
    {
        for (int j = 0; j < column1; j++)
        {
            matrix1[i, j] = random.Next(1, 100);
            matrix2[i, j] = random.Next(1, 100);
        }
    }
}
```

Po wygenerowaniu można było przejść do zrównoleglania funkcji i wykonania obliczeń, ta logika została przedstawiona poniżej:

```
private void buttonCalculate_Click(object sender, EventArgs e)
{
    if (textBoxThreads.Text != "")
    {
        this.threads = int.Parse(textBoxThreads.Text);
    }
    else
    {
        this.threads = 1;
    }

    if (!matrixMultiplier.IsMultiplicationPossible())
    {
        MessageBox.Show("Nie mo na pomno y tych macierzy");
        return;
    }

    int size = matrix1.GetLength(0);
    int rowsPerThread = size / threads;

    this.threadsArray = new Thread[threads];
    for (int i = 0; i < threads; i++)
    {
        int startRow = i * rowsPerThread;
        int endRow = (i == threads - 1) ? size : (i + 1) * rowsPerThread
            ;

        threadsArray[i] = new Thread(() => matrixMultiplier.
            CalculateMatrix(startRow, endRow));
    }

    var watch = System.Diagnostics.Stopwatch.StartNew();
    foreach (var thread in threadsArray)
    {
        thread.Start();
    }
    foreach (var thread in threadsArray)
    {
        thread.Join();
    }
    watch.Stop();
```

```
var elapsedMs = watch.ElapsedMilliseconds;
labelThread.Text = "Czas niskopoziomowy: " + elapsedMs + " ms";
```

Sama funkcja mnożąca macierze jest prosta i wygląda następująco:

```
public void CalculateMatrix(int startRow, int endRow)
{
    for (int i = startRow; i < endRow; i++)
    {
        for (int j = 0; j < this.column2; j++)
        {
            for (int k = 0; k < this.column1; k++)
            {
                this.resultMatrix[i, j] += this.matrix1[i, k] * this.
                    matrix2[k, j];
            }
        }
    }
}
```

2.2 Zadanie 2 - Biblioteka Parallel oraz analiza przyśpieszenia.

W dalszej cześci zadania należało powtórzyć mnożenie tych samych macierzy, ale teraz z użyciem funkcji wysokiego poziomu tj. ParallelFor. W głównej części programu zostały podane opcje jako maksymalna ilość wątków, które należało użyć i zostało to uzyskane za pomocą klasy ParallelOptions:

```
ParallelOptions options = new ParallelOptions();
options.MaxDegreeOfParallelism = threads;
```

A nastepnie zostało wykonane mnożenie zrównoleglone:

```
public void CalculateMatrixParallel(ParallelOptions opt)
{
    Parallel.For(0, this.row1, opt, i =>
    {
        for (int j = 0; j < this.column2; j++)
        {
            for (int k = 0; k < this.column1; k++)
            {
                this.resultMatrix[i, j] += this.matrix1[i, k] * this.
                    matrix2[k, j];
            }
        }
    });
}
```

2.3 Zadanie 3 - Przetwarzanie obrazów.

W ostatniej części zadania należało wykonać 4 przekształcenia obrazów, ale każdy algorytm miał się policzyć na osobnym wątku. W tym celu po wybraniu zdjęcia w GUI po naciśnięciu przycisku "Przetwórz" zostawała uruchomiona funkcja:

```
private void buttonProcess_Click(object sender, EventArgs e)
{
    Thread[] processes = new Thread[4];
    processes[0] = new Thread(() =>
    {
        Bitmap negativeImage;
        lock (imageLock)
        {
            negativeImage = ImageProcessing.Negative(image);
        }
        pictureBox2.Image = negativeImage;
```

```
        });

        processes[1] = new Thread(() =>
        {
            Bitmap thresholdedImage;
            lock (imageLock)
            {
                thresholdedImage = ImageProcessing.ApplyThreshold(image,
                    128);
            }
            pictureBox3.Image = thresholdedImage;
        });

        processes[2] = new Thread(() =>
        {
            Bitmap mirroredImage;
            lock (imageLock)
            {
                mirroredImage = ImageProcessing.MirrorEffect(image);
            }
            pictureBox4.Image = mirroredImage;
        });

        processes[3] = new Thread(() =>
        {
            Bitmap edgesImage;
            lock (imageLock)
            {
                edgesImage = ImageProcessing.DetectEdges(image);
            }
            pictureBox5.Image = edgesImage;
        });

        foreach (var process in processes)
        {
            process.Start();
        }
        foreach (var process in processes)
        {
            process.Join();
        }
    }
```

Tworzy ona listę czteroelementową tak, aby każda funkcja mogła być wykonana przez osobny wątek, do tego celu użyto jeszcze blokowania zasobów, aby zapobiec zakleszczeniu oraz użyto klasy, która miała zaimplementowane cztery algorytmy w sobie:

1. Negatyw

```
        public static Bitmap Negative(Bitmap image)
        {
            int w = image.Width;
            int h = image.Height;
            BitmapData srcData = image.LockBits(new Rectangle(0, 0, w, h),
                ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);
            int bytes = srcData.Stride * srcData.Height;
            byte[] buffer = new byte[bytes];
            byte[] result = new byte[bytes];
            Marshal.Copy(srcData.Scan0, buffer, 0, bytes);
            image.UnlockBits(srcData);
            int current = 0;
            int cChannels = 3;
```

```
            for (int y = 0; y < h; y++)
            {
                for (int x = 0; x < w; x++)
                {
                    current = y * srcData.Stride + x * 4;
                    for (int c = 0; c < cChannels; c++)
                    {
                        result[current + c] = (byte)(255 - buffer[current +
                            c]);
                    }
                    result[current + 3] = 255;
                }
            }
            Bitmap resImg = new Bitmap(w, h);
            BitmapData resData = resImg.LockBits(new Rectangle(0, 0, w, h),
                ImageLockMode.WriteOnly, PixelFormat.Format32bppArgb);
            Marshal.Copy(result, 0, resData.Scan0, bytes);
            resImg.UnlockBits(resData);
            return resImg;
        }
```

2. Progowanie

```
        public static Bitmap ApplyThreshold(Bitmap inputBitmap, int
            threshold)
        {
            Bitmap outputBitmap = new Bitmap(inputBitmap.Width, inputBitmap.
                Height);

            for (int y = 0; y < inputBitmap.Height; y++)
            {
                for (int x = 0; x < inputBitmap.Width; x++)
                {
                    Color pixelColor = inputBitmap.GetPixel(x, y);
                    int grayscaleValue = (int)(pixelColor.R * 0.3 +
                        pixelColor.G * 0.59 + pixelColor.B * 0.11);

                    Color newColor = (grayscaleValue < threshold) ? Color.
                        Black : Color.White;

                    outputBitmap.SetPixel(x, y, newColor);
                }
            }

            return outputBitmap;
        }
```

3. Odbicie lustrzane

```
        public static Bitmap MirrorEffect(Bitmap image)
        {
            Bitmap resultBitmap = new Bitmap(image.Width, image.Height);

            for (int i = 0; i < image.Width; i++)
            {
                for (int j = 0; j < image.Height; j++)
                {
                    resultBitmap.SetPixel(i, j, image.GetPixel(image.Width -
                        i - 1, j));
                }
            }
```

```
            return resultBitmap;
        }
```

4. Wykrywanie krawędzi

```
        public static Bitmap DetectEdges(Bitmap inputBitmap)
        {
            // Convert Bitmap to Mat
            Mat inputImage = BitmapConverter.ToMat(inputBitmap);

            // Convert input image to grayscale
            Mat grayImage = new Mat();
            Cv2.CvtColor(inputImage, grayImage, ColorConversionCodes.
                BGR2GRAY);

            // Gaussian blur to reduce noise
            Mat blurredImage = new Mat();
            Cv2.GaussianBlur(grayImage, blurredImage, new OpenCvSharp.Size
                (5, 5), 0);

            // Canny edge detection
            Mat edges = new Mat();
            Cv2.Canny(blurredImage, edges, 50, 150);

            // Convert Mat to Bitmap
            Bitmap resultBitmap = edges.ToBitmap();

            return resultBitmap;
        }
```

Repozytorium github: `https://github.com/MrSpokeMan/.Net-and-Java/tree/master/Laboratorium3`