Politechnika Wrocławska

Platformy programistyczne .Net i Java

Obliczenia wielowątkowe w technologii .NET

Prowadzący: Mgr. inż. Michał Jaroszczuk

Grupa Środa, 18:55-20:40

Jan Klisowski 263485

14.04.2024

Spis treści

1. Opis Projektu - thread

Projekt dotyczy implementacji programu w języku C# służącego do mnożenia macierzy. Program umożliwia zarówno sekwencyjne, jak i równoległe mnożenie macierzy za pomocą wielu wątków przy użyciu Thread. Interfejs graficzny pozwala użytkownikowi na wprowadzenie rozmiaru macierzy, liczby wątków oraz ziarna dla generacji liczb losowych.

2. Opis Klas i Metod

2.1. Klasa `Program`

Klasa `Program` zawiera punkt wejścia dla aplikacji. Metoda `Main` uruchamia główne okno aplikacji.

2.2. Klasa `MatrixOperations`

Klasa `MatrixOperations` zawiera metody do manipulacji macierzami oraz operacji na nich.

2.2.1. Metoda `GenerateRandomMatrix`

Generuje losową macierz o określonych rozmiarach.

2.2.2. Metoda `MultiplyMatricesSequential`

Wykonuje sekwencyjne mnożenie dwóch macierzy.

2.2.3. Metoda `MultiplyMatricesParallel`

Wykonuje równoległe mnożenie dwóch macierzy przy użyciu wielu wątków.

2.2.4. Metoda `MultiplyMatricesPartial`

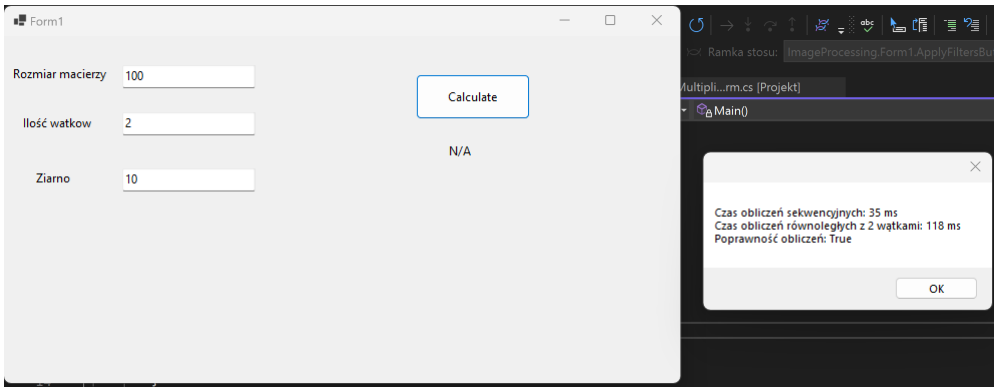
Wykonuje częściowe mnożenie dwóch macierzy w trybie równoległym..

2.3. Klasa `MatrixMultiplierForm`

Klasa `MatrixMultiplierForm` reprezentuje główne okno aplikacji. Umożliwia użytkownikowi wprowadzenie rozmiaru macierzy, liczby wątków i ziarna, a następnie przeprowadzenie operacji mnożenia macierzy.

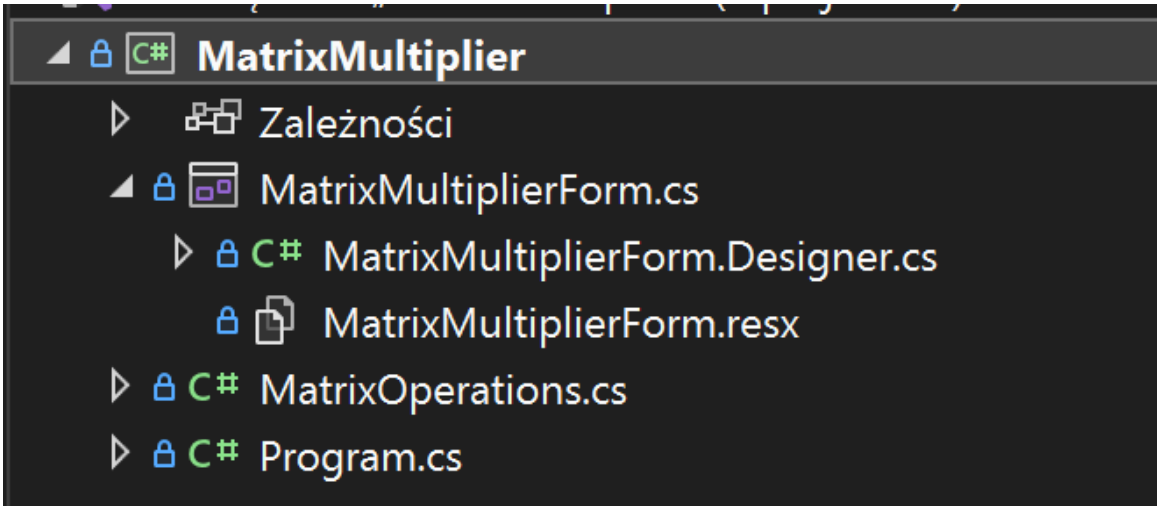
3. Zdjęcia

3.1. Interfejs Graficzny Aplikacji



Rys. 1: Interfejs Graficzny Aplikacji

3.2. Drzewo Projektu



Rys. 2: Drzewo Projektu

4. Opis Projektu - parallel

Projekt dotyczy implementacji programu w języku C# służącego do mnożenia macierzy. Program umożliwia zarówno sekwencyjne, jak i równoległe mnożenie macierzy za pomocą technologii wysokopoziomowej Parallel z biblioteki .NET. Interfejs graficzny pozwala użytkownikowi na wprowadzenie rozmiaru macierzy, liczby wątków oraz ziarna dla generacji liczb losowych.

5. Opis Klas i Metod

5.1. Klasa `MatrixOperations`

Klasa `MatrixOperations` zawiera metody do manipulacji macierzami oraz operacji na nich.

5.1.1. Metoda `GenerateRandomMatrix`

Generuje losową macierz o określonych rozmiarach.

5.1.2. Metoda `MultiplyMatricesSequential`

Wykonuje sekwencyjne mnożenie dwóch macierzy.

5.1.3. Metoda `MultiplyMatricesParallel`

Wykonuje równoległe mnożenie dwóch macierzy za pomocą technologii Parallel.

5.2. Klasa `MatrixMultiplierForm`

Klasa `MatrixMultiplierForm` reprezentuje interfejs użytkownika.

5.2.1. Metoda `MultiplyButton`$_{C}$*lick*

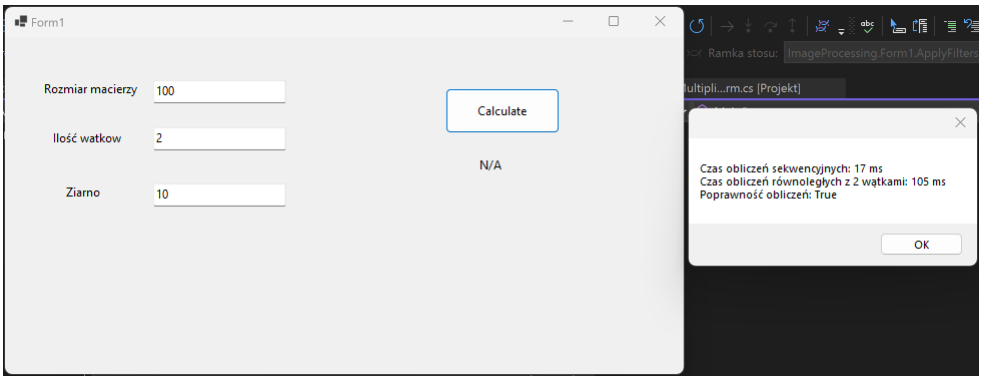
Wywoływana po kliknięciu przycisku "Calculate". Przetwarza macierze i wyświetla wyniki w oknie dialogowym.

5.2.2. Metoda `ShowMatrix`

Wyświetla macierz w oknie dialogowym.

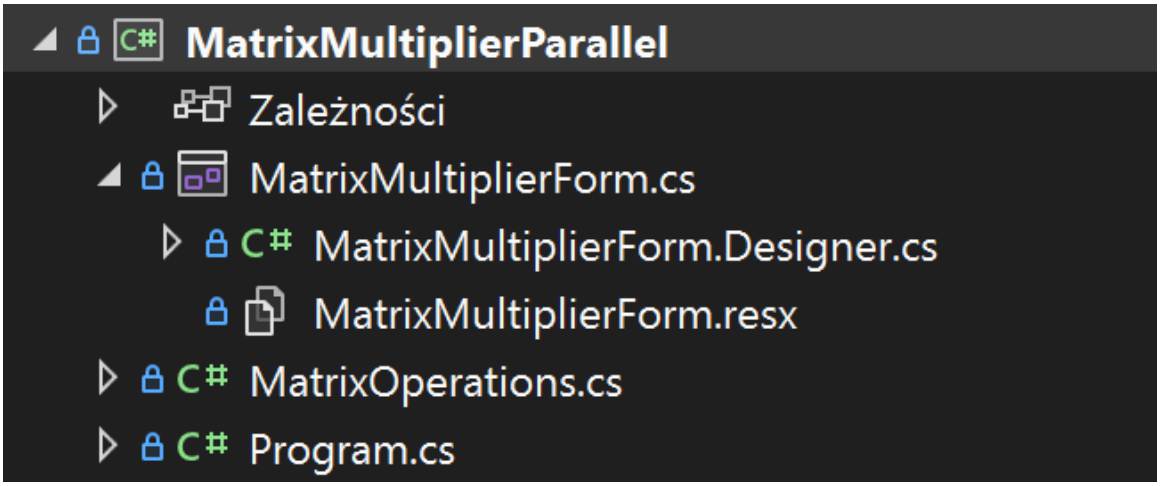
6. Zdjęcia

6.1. Interfejs Graficzny Aplikacji



Rys. 3: Interfejs Graficzny Aplikacji

6.2. Drzewo Projektu



Rys. 4: Drzewo Projektu

6.3. Kluczowy Fragment Programu

```
public static int[,] MultiplyMatricesParallel(int[,] matrixA, int[,] matrixB, int threadCount)
{
    int rowsA = matrixA.GetLength(0);
    int columnsB = matrixB.GetLength(1);
    int[,] result = new int[rowsA, columnsB];

    Parallel.For(0, rowsA, i =>
    {
        for (int j = 0; j < columnsB; j++)
        {
            for (int k = 0; k < matrixA.GetLength(1); k++)
            {
                result[i, j] += matrixA[i, k] * matrixB[k, j];
            }
        }
    });

    return result;
}
```

Rys. 5: Kluczowy Fragment Programu

7. Tabela porównawcza

Czasy są wysokie i nie występuje przyspieszenie. Wydaję mi się, że może byc to kwestia maszyny wirtualnej, na której pracuję.

Tabela 1: Czasy mnożenia macierzy dla ziarna 10

Typ	Macierz	Liczba wątków	Czas (ms)
Sekwencyjnie	800	-	7540
Sekwencyjnie	900	-	10569
Sekwencyjnie	1000	-	14671
Thread	800	2	2981
Thread	900	2	4506
Thread	900	4	4930
Thread	900	6	4618
Thread	1000	2	5620
Thread	1000	4	6820
Thread	1000	6	6373
Thread	1000	8	6281
Parallel	800	2	3882
Parallel	900	2	5451
Parallel	900	4	5569
Parallel	900	6	5498
Parallel	1000	2	7500
Parallel	1000	4	7431
Parallel	1000	6	7529
Parallel	1000	8	7574

8. Opis Projektu - obrazy

Projekt dotyczy implementacji aplikacji do przetwarzania obrazów, która umożliwia użytkownikowi otwarcie obrazu, a następnie zastosowanie na nim różnych filtrów graficznych. Aplikacja została napisana w języku C# z wykorzystaniem platformy .NET Framework.

9. Opis Klas i Metod

9.1. Klasa `Form1`

Klasa `Form1` reprezentuje główne okno aplikacji. Umożliwia użytkownikowi otwarcie obrazu, zastosowanie filtrów graficznych oraz wyświetlenie przetworzonych obrazów.

9.1.1. Metoda `OpenImageButton_Click`

Metoda `OpenImageButton_Click` jest wywoływana po kliknięciu przycisku "Załaduj". Otwiera okno dialogowe umożliwiające wybór pliku z obrazem, następnie wczytuje ten obraz i wyświetla go w głównym oknie aplikacji.

9.1.2. Metoda `ApplyFiltersButton_Click`

Metoda `ApplyFiltersButton_Click` jest wywoływana po kliknięciu przycisku "Zastosuj". Przetwarza wczytany obraz przy użyciu czterech różnych filtrów graficznych: przekształcenie do skali szarości, progowanie, negatyw oraz lustrzane odbicie. Następnie wyświetla przetworzone obrazy w osobnych oknach.

9.1.3. Metoda `ApplyGrayscale`

Metoda `ApplyGrayscale` przekształca obraz na obraz w skali szarości.

9.1.4. Metoda `ApplyThreshold`

Metoda `ApplyThreshold` stosuje na obrazie progowanie.

9.1.5. Metoda `ApplyNegative`

Metoda `ApplyNegative` generuje negatyw obrazu.

9.1.6. Metoda `ApplyMirror`

Metoda `ApplyMirror` wykonuje lustrzane odbicie obrazu.

9.2. Klasa `Program`

Klasa `Program` zawiera punkt wejścia dla aplikacji. Uruchamia główne okno aplikacji.

10. Repozytorium

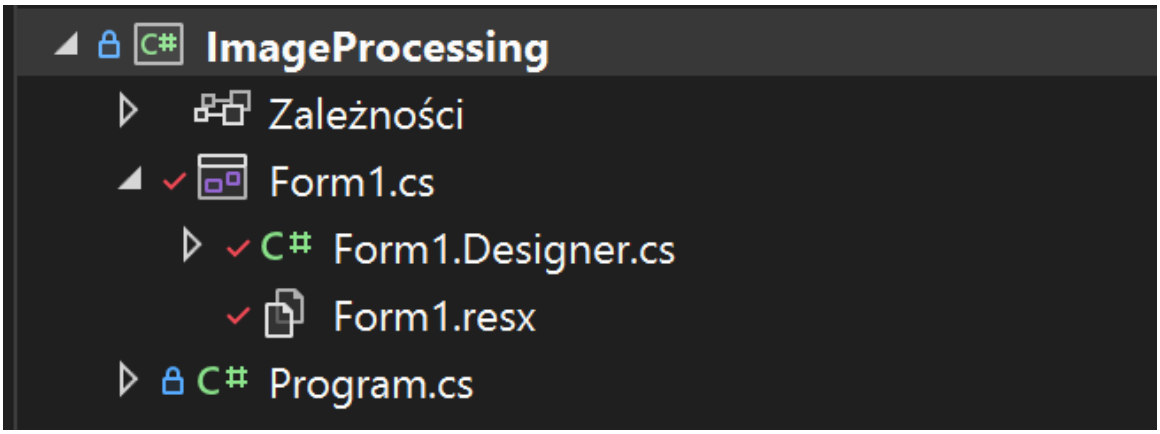
Link do repozytorium projektu: *https://github.com/jachoofrachoo/netjavaKlis*

11. Zdjęcia

11.1. Interfejs Graficzny Aplikacji

Rys. 6: Interfejs Graficzny Aplikacji

11.2. Drzewo Projektu



Rys. 7: Drzewo Projektu

11.3. Kluczowy Fragment Programu

```
Parallel.For(0, 4, i =>
{
    switch (i)
    {
        case 0: // szarosc
            processedImages[i] = ApplyGrayscale(new Bitmap(img));
            break;
        case 1: // progowanie
            processedImages[i] = ApplyThreshold(new Bitmap(img));
            break;
        case 2: // Negatyw
            processedImages[i] = ApplyNegative(new Bitmap(img));
            break;
        case 3: // lustrzane
            processedImages[i] = ApplyMirror(new Bitmap(img));
            break;
    }
```

Rys. 8: Kluczowy Fragment Programu

```
private Bitmap ApplyGrayscale(Bitmap image)
{
    Bitmap processedImage = new Bitmap(image.Width, image.Height);

    for (int y = 0; y < processedImage.Height; y++)
    {
        for (int x = 0; x < processedImage.Width; x++)
        {
            Color originalColor = image.GetPixel(x, y);
            int grayValue = (int)(originalColor.R * 0.3 + originalColor.G * 0.59 + originalColor.B * 0.11);
            Color grayColor = Color.FromArgb(grayValue, grayValue, grayValue);
            processedImage.SetPixel(x, y, grayColor);
        }
    }

    return processedImage;
}
```

Rys. 9: Kluczowy Fragment Programu