Platformy programistyczne .Net i Java

Obliczenia wielowątkowe w technologii.NET

Mateusz Marciak 272599

Opis projektu

Tematem laboratorium było wykonanie obliczeń na przykładzie mnożenia dwóch macierzy A i B, z użyciem zrównoleglenia wątkowego **wysoko poziomo** oraz **nisko poziomowo**. Następnie należało wykonać projekt aplikacji GUI do przetwarzania obrazów równolegle. W projekcie użyto:

- Platformę .NET 8.0
- Bibliotekę Parallel (obliczenia wysoko poziomowe)
- Bibliotekę Thread (obliczenia nisko poziomowe)
- Windows Forms

Zrównoleglenie obliczeń

W ramach testów przyjęto macierze kwadratowe o rozmiarach 100 x 100, 300 x 300 oraz 600 x 600. Generowane w nich wartości są losowymi danymi typu **float**.

Wysoko poziomowe

Na tym etapie wykorzystaliśmy bibliotekę wysokiego poziomu **Parallel**.

```
public static double[,] Multiply(double[,] A, double[,] B, int threads)
{
    int rowsA = A.GetLength(0);
    int colsA = A.GetLength(1);
    int colsB = B.GetLength(1);
    var result = new double[rowsA, colsB];
    if (colsB != rowsA)
        throw new ArgumentException("Liczba kolumn A musi być równa liczbie wierszy B");

    var options = new ParallelOptions { MaxDegreeOfParallelism = threads };

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

    return result;
}
```

Rysunek 1 Metoda mnożenia macierzy przy pomocy Parallel

W implementacji należało ograniczyć liczbę jednocześnie uruchamianych wątków. Służy do tego obiekt **ParallelOptions**. Z kolei **MaxDegreeOfParallelism** pozwala wykorzystać dokładnie tyle wątków, ile mu podano. Następnie wykonywała się równoległa wersja pętli **for**, w które dla każdego wiersz **i** macierzy uruchamiano osobny wątek.

Nisko poziomowe

W tym etapie z kolei wykorzystaliśmy bibliotekę niskopoziomową **Threads**.

```
public static double[,] MultiplyLowLevel(double[,] A, double[,] B, int threads)
{
    int rowsA = A.GetLength(0);
    int colsA = A.GetLength(1);
    int rowsB = B.GetLength(0);
    int colsB = B.GetLength(1);

    if (colsA != rowsB)
        throw new ArgumentException("Liczba kolumn A musi być równa liczbie wierszy B");

    double[,] result = new double[rowsA, colsB];
    Thread[] threadArray = new Thread[threads];
    int chunkSize = rowsA / threads;
    object locker = new object();

    for (int t = 0; t < threads; t++)
    {
        int start = t * chunkSize;
        int end = (t == threads - 1) ? rowsA : start + chunkSize;

        threadArray[t] = new Thread(() =>
        {
            for (int i = start; i < end; i++)
            {
                for (int j = 0; j < colsB; j++)
                {
                    double sum = 0;
                    for (int k = 0; k < colsA; k++)
                    {
                        sum += A[i, k] * B[k, j];
                    }
                    result[i, j] = sum;
                }
            }
        });

        threadArray[t].Start();
    }

    foreach (Thread thread in threadArray)
        thread.Join();

    return result;
}
```

Rysunek 2 Metoda mnożenia macierzy przy pomocy Threads

W implementacji tworzyliśmy tablicę wątków **threadArray**, a następnie tworzyliśmy chunki. Chunki służyły do podziału pracy na wątki.

W tym przypadku każdy wątek przetwarza określony zakres wierszy (od start do end).

Synchronizacja zakończenia obliczeń odbywała się za pomocą metody **Join()**.

Porównanie wyników

Obliczenia dla obu podjeść zostały zaimplementowane w głównych programach dla obu podejść analogicznie. Opierały się na zmierzeniu czasu pracy w milisekundach dla 1, 2, 4 oraz 8 wątków, a następnie wykonaniu **5** powtórzeń tych operacji oraz uśrednieniu wyników. Dodatkowo dla obu podejść zaimplementowano metodę **obliczania sekwencyjnego** (z użyciem tylko głównego wątku) oraz metodę **porównania wyników** dla mnożenia równoległego oraz sekwencyjnego.

```
1 reference
public static double[,] MultiplySequential(double[,] A, double[,] B)
{
    int rowsA = A.GetLength(0);
    int colsA = A.GetLength(1);
    int rowsB = B.GetLength(0);
    int colsB = B.GetLength(1);

    if (colsA != rowsB)
        throw new ArgumentException("Liczba kolumn A musi być równa liczbie wierszy B");

    double[,] result = new double[rowsA, colsB];

    for (int i = 0; i < rowsA; i++)
        for (int j = 0; j < colsB; j++)
        {
            double sum = 0;
            for (int k = 0; k < colsA; k++)
                sum += A[i, k] * B[k, j];
            result[i, j] = sum;
        }

    return result;
}
```

Rysunek 3 Mnożenie sekwencyjne

```
1 reference
public static bool Compare(double[,] A, double[,] B, double epsilon = 1e-6)
{
    if (A.GetLength(0) != B.GetLength(0) || A.GetLength(1) != B.GetLength(1))
        return false;

    for (int i = 0; i < A.GetLength(0); i++)
        for (int j = 0; j < A.GetLength(1); j++)
            if (Math.Abs(A[i, j] - B[i, j]) > epsilon)

                return false;

    return true;
}
```

Rysunek 4 Porównanie wyników mnożenia

Wykonane badanie zestawione są w poniższych tabelkach:

Badania wysoko poziomowe

Rozmiar	Wątki	Sredni czas Watki [ms]	Sekwencyjnie [ms]	Poprawny wynik
100x100	1	7,6	5	True
100x100	2	2,8		True
100x100	4	1,6		True
100x100	8	15,4		True
300x300	1	131	141,2	True
300x300	2	65,4		True
300x300	4	39,6		True
300x300	8	43,2		True
600x600	1	1092,4	1080,2	True
600x600	2	554,4		True
600x600	4	323,4		True
600x600	8	239,8		True

Badania nisko poziomowe

Rozmiar	Wątki	Sredni czas Watki [ms]	Sekwencyjnie [ms]	Poprawny wynik
100x100	1	12,6	4,8	True
100x100	2	17,2		True
100x100	4	33,6		True
100x100	8	53,4		True
300x300	1	146,2	131,8	True
300x300	2	93		True
300x300	4	87,2		True
300x300	8	129,8		True
600x600	1	1200,6	1021,8	True
600x600	2	674,2		True
600x600	4	379,4		True
600x600	8	350,8		True

Aplikacja okienkowa do przetwarzania zdjęć

Ostatnim etapem było stworzenie aplikacji do przetwarzania zdjęć równolegle. Aplikacje utworzono w **windows forms**. Zastosowano zrównoleglenie wysoko poziomowe używając biblioteki Parallel.

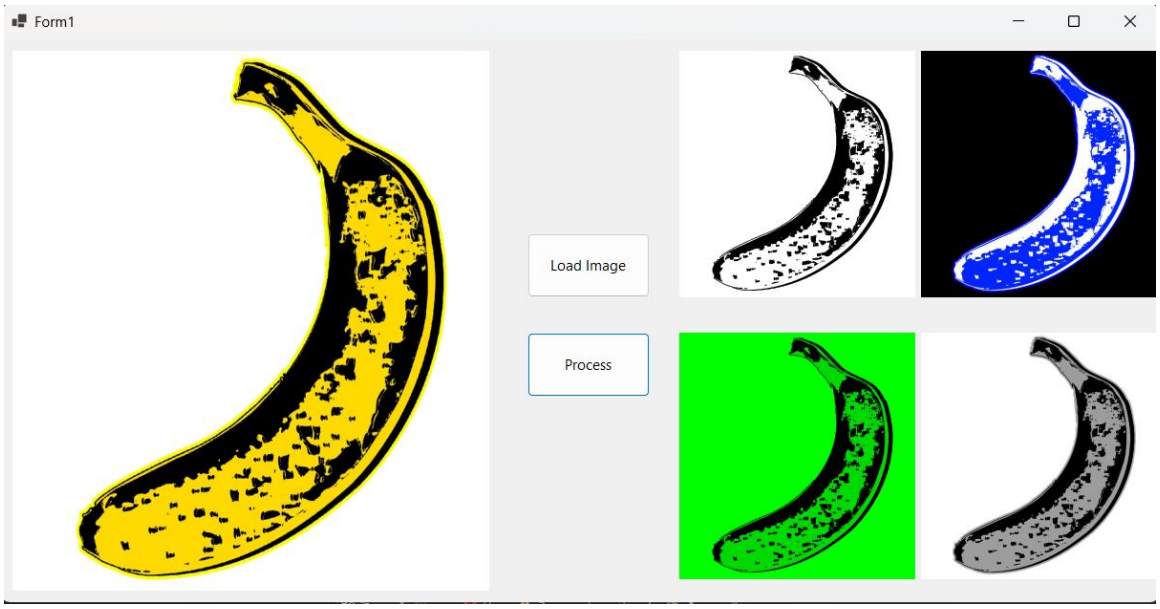
```
private void buttonProcess_Click(object sender, EventArgs e)
{
    if (originalImage == null) return;


    Bitmap image1 = (Bitmap)originalImage.Clone();
    Bitmap image2 = (Bitmap)originalImage.Clone();
    Bitmap image3 = (Bitmap)originalImage.Clone();
    Bitmap image4 = (Bitmap)originalImage.Clone();

    Bitmap gray = null;
    Bitmap negative = null;
    Bitmap binary = null;
    Bitmap green = null;

    Parallel.Invoke(
        () => { gray = ApplyGrayScale(image1); },
        () => { negative = ApplyNegative(image2); },
        () => { binary = ApplyThreshold(image3); },
        () => { green = ApplyGreenTint(image4); }
    );

    pictureBox2.Image = gray;
    pictureBox3.Image = negative;
    pictureBox4.Image = binary;
    pictureBox5.Image = green;
}
```

Rysunek 5 Implenetacja zrównoleglenia

Aplikacja składa się z dwóch przycisków oraz 5 pól dla obrazów (1orginalny 4 przetworzone). Pierwszy przycisk wczytywał obraz, a drugi inicjował przetwarzanie.

Dla wczytanego obrazu wykonano:

- progowanie
- negatyw
- zabarwienie na zielono
- przetworzenie na obraz w skali szarości



Rysunek 6 Rezultat