POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Zarządzanie w systemach i sieciach komputerowych - projekt

Optymalizacja algorytmu splotowego przetwarzania obrazu z wykorzystaniem wielowątkowości

AUTORZY:

Jakub Kapłonek 263908
Radosław Zimoch 263963
E-mail:
- 263908@student.pwr.edu.pl
- 263963@student.pwr.edu.pl

PROWADZĄCY ZAJĘCIA:

Dr inż. Robert Wójcik

OCENA PRACY:

Wrocław, 2024

Spis treści

1. Cel i zakres projektu

Celem tego projektu jest optymalizacja algorytmu przetwarzania obrazu za pomocą operacji splotu, z wykorzystaniem wielowątkowości. Zadaniem algorytmu jest przetwarzanie obrazów, gdzie operacja splotu [1] jest wykonywana przy użyciu macierzy rozmycia Gaussa [2]. Optymalizacja procesu polega na przyspieszeniu operacji przetwarzania dzięki podzieleniu zadań na mniejsze fragmenty, które są wykonywane równolegle na różnych wątkach [3]. Tego typu podejście może znaleźć zastosowanie w aplikacjach wymagających szybkiego przetwarzania dużych obrazów, np. w systemach wizyjnych czy analizie danych graficznych.

W ramach projektu zostanie opracowany kompletny system, począwszy od analizy problemu, poprzez projekt i implementację algorytmu, aż po testowanie i ocenę jakości jego działania. Implementacja obejmie wczytywanie danych z plików BMP oraz CSV, równoległe przetwarzanie obrazu z wykorzystaniem wielowątkowości oraz zapis przetworzonego obrazu do pliku wynikowego w formacie bitmapy. Testowanie obejmie zarówno sprawdzanie poprawności przetwarzania obrazu, jak i ocenę wydajności działania algorytmu, w tym porównanie efektywności wersji wielowątkowej z jednowątkową. Wyniki testów pozwolą na określenie skuteczności zastosowanej optymalizacji oraz możliwości jej zastosowania w praktycznych systemach przetwarzania obrazów.

2. Sformułowanie problemu

Algorytm przetwarzania obrazu realizuje operację splotu między obrazem a macierzą Gaussa, co prowadzi do efektu rozmycia obrazu. Splot ten polega na modyfikacji wartości każdego piksela obrazu na podstawie wartości jego sąsiadów, co wpływa na wygładzenie i rozmycie krawędzi. Problem, który należy rozwiązać, to efektywne wykonanie tej operacji dla dużych obrazów i macierzy, gdzie czas przetwarzania może być znaczny. W celu przyspieszenia obliczeń wykorzystana zostanie
wielowątkowość, co pozwoli na równoległe przetwarzanie wielu pikseli jednocześnie. Dzięki temu możliwe jest znaczne zmniejszenie czasu potrzebnego na wykonanie całej operacji.

Założenia:

- Zastosowanie operacji splotu do przetwarzania każdego piksela obrazu na podstawie sąsiadujących pikseli i wartości macierzy Gaussa.
- Podzielenie obrazu na mniejsze segmenty przetwarzane równolegle przez wiele wątków.
- Zastosowanie wielowątkowości w celu przyspieszenia obliczeń.
- Możliwość ustalenia liczby wątków przed uruchomieniem algorytmu.
- Wczytanie obrazu BMP z pliku.
- Wczytanie macierzy rozmycia Gaussa z pliku CSV.
- Końcowy obraz jest zapisywany w formacie BMP po zakończeniu przetwarzania.
- Testowanie poprawności obrazu oraz wydajności algorytmu w wersji wielowątkowej i jednowątkowej.

3. Analiza złożoności obliczeniowej problemu

W przypadku operacji splotu, złożoność obliczeniowa zależy głównie od rozmiaru obrazu oraz macierzy splotu. Dla każdego piksela obrazu wykonywane są operacje na sąsiadujących pikselach, co prowadzi do zależności czasowej liniowej względem liczby pikseli oraz elementów macierzy splotu. W prostszych przypadkach, gdzie rozmiary obrazu i macierzy są małe, czas obliczeń może być akceptowalny, jednak w przypadku dużych obrazów i większych macierzy złożoność obliczeniowa rośnie, co skutkuje znacznym wydłużeniem czasu przetwarzania.

Złożoność obliczeniową operacji splotu można wyrazić jako funkcję rozmiaru obrazu oraz rozmiaru macierzy Gaussa. Zakładając, że:

- **n** to liczba pikseli obrazu (np. dla obrazu o rozmiarze ($w \times h$), gdzie (w) to szerokość, a (h) to wysokość, liczba pikseli wynosi ($n = w \times h$)),
- **m** to liczba elementów w macierzy Gaussa

złożoność obliczeniowa splotu wynosi:

$$O(n \times m)$$

Dla każdego piksela obrazu (n) wykonywane jest m operacji, czyli tyle, ile wynosi liczba elementów w macierzy Gaussa. W przypadku dużych obrazów i skomplikowanych macierzy Gaussa, złożoność obliczeniowa może sięgać nawet wartości kwadratowych, co wymaga zastosowania optymalizacji. W zależności od liczby dostępnych wątków i zasobów systemowych, wzrost wydajności może być znaczący, co sprawia, że wielowątkowość jest kluczowym elementem optymalizacji w przypadku tego typu obliczeń.

4. Metoda i algorytmy rozwiązywania problemu

Operacja splotu opiera się na klasycznej metodzie przetwarzania obrazu, gdzie wartość każdego piksela jest obliczana na podstawie wartości pikseli sąsiednich, z uwzględnieniem macierzy filtrującej, jaką w tym przypadku jest macierz Gaussa. W celu optymalizacji operacji przetwarzania algorytm dzieli obraz na mniejsze fragmenty, które mogą być równolegle przetwarzane przez różne wątki. Działanie to pozwala na lepsze wykorzystanie zasobów procesora, zwłaszcza w systemach z wieloma rdzeniami, gdzie każdy rdzeń może obsłużyć inny fragment obrazu.

Algorytm rozpoczyna się od wczytania obrazu BMP i macierzy CSV, po czym obraz jest dzielony na segmenty, które są przypisywane do poszczególnych wątków. Każdy wątek wykonuje operację splotu na przydzielonym fragmencie, po czym przetworzone fragmenty są łączone w jeden obraz wynikowy. Tego typu podejście pozwala na równoczesne przetwarzanie różnych części obrazu, co prowadzi do znacznego skrócenia czasu obliczeń. Dzięki takiej optymalizacji algorytm jest w stanie efektywnie przetwarzać nawet bardzo duże obrazy w rozsądnym czasie.

5. Technologie i narzędzia implementacji

Projekt zostanie zrealizowany w języku Java [4], który oferuje bogaty zestaw narzędzi do obsługi wielowątkowości oraz przetwarzania plików. Dzięki temu możliwe będzie optymalne zarządzanie zasobami systemowymi, co pozwoli na efektywne wykorzystanie wielordzeniowych procesorów. Do wczytywania plików BMP oraz CSV zostaną użyte standardowe biblioteki Java, takie jak BufferedImage oraz BufferedReader [5], które zapewniają łatwy dostęp do danych obrazowych i macierzy.

Wybrane przez nas środowisko programistyczne to IntelliJ IDEA, które oferuje wsparcie dla debugowania, zarządzania wątkami oraz optymalizacji kodu. Testowanie wydajności algorytmu odbędzie się przy użyciu narzędzi do pomiaru czasu wykonywania zadań, takich jak System.nanoTime(), co pozwoli na dokładne śledzenie efektywności działania wersji wielowątkowej. Implementacja zostanie zrealizowana w taki sposób, aby liczba wątków w algorytmie była możliwa do ustalenia przed uruchomieniem aplikacji.

6. Testowanie i ocena jakości rozwiązań

Testowanie algorytmu będzie obejmować zarówno sprawdzanie poprawności działania, jak i ocenę wydajności. Testy poprawności mają na celu zweryfikowanie, czy obraz po przetworzeniu przez algorytm splotu jest poprawnie wygładzony, a efekt rozmycia odpowiada oczekiwanym rezultatom. Obraz testowy o ustalonym rozmiarze i stopniu szczegółowości będzie porównywany przed i po przetwarzaniu, aby sprawdzić, czy algorytm działa zgodnie z założeniami. Wynikiem działania algorytmu powinien być obraz o zmniejszonej ostrości, gdzie krawędzie zostały wygładzone zgodnie z macierzą. Ocenione zostaną również czasy realizacji operacji wygładzania w wariancie sekwencyjnym i wielowątkowym w zależności od rozmiaru obrazu i macierzy. Testowanie przebiegało na 4 rdzeniowym procesorze Intel i5 7600K.

7. Opis implementacji

Implementacja algorytmu splotowego z wykorzystaniem wielowątkowości została podzielona na kilka kluczowych klas, z których każda odpowiada za specyficzny aspekt projektu. Poniżej przedstawiono opis poszczególnych klas i ich funkcji w systemie.

7.1 Klasa `Main`

Klasa `Main` pełni rolę punktu startowego programu. Jej zadaniem jest:

- Walidacja parametrów wejściowych podanych w linii poleceń.
- Inicjalizacja odpowiedniej wersji algorytmu splotowego (jednowątkowego lub wielowątkowego) na podstawie liczby wątków.
- Wczytanie obrazu w formacie BMP i macierzy filtrującej z pliku CSV za pomocą klas `BitmapManager` i `CsvReader`.
- Pomiar czasu wykonania algorytmu splotowego.
- Zapisanie przetworzonego obrazu do pliku wynikowego BMP.

Przykład uruchomienia:

```
java -jar convolution.jar input.bmp filter.csv 4
```

W powyższym przykładzie program wczytuje obraz `input.bmp` oraz filtr `filter.csv` i wykonuje operację splotu z użyciem 4 wątków.

7.2 Klasa `SingleThreadConvolution`

`SingleThreadConvolution` implementuje interfejs `Convolution` i realizuje algorytm splotowy w wersji jednowątkowej.

- Główna metoda `calculateConvolution` iteruje po każdym pikselu obrazu i oblicza jego wartość na podstawie macierzy sąsiadujących pikseli oraz wartości z macierzy filtrującej.
- Obsługiwane są przypadki brzegowe obrazu, gdzie brakujące piksele są traktowane jako wartość zerowa.

```
for (int y = 0; y < matrixHeight; y++) {
    for (int x = 0; x < matrixWidth; x++) {
        double sum = 0.0;
        for (int ky = 0; ky < filterHeight; ky++) {
            for (int kx = 0; kx < filterWidth; kx++) {
                int matY = y + ky - filterHeight / 2;
                int matX = x + kx - filterWidth / 2;
                if (
                       matX >= 0 && matX < matrixWidth &&
                       matY >= 0 && matY < matrixHeight
                )
                    sum += filter[ky][kx] * matrix[matY][matX];
                }
            }
        }
        output[y][x] = sum;
    }
```

7.3 Klasa `MultiThreadedJavaConvolution`

`MultiThreadedJavaConvolution` również implementuje interfejs `Convolution`, ale wykorzystuje wielowątkowość do równoległego przetwarzania obrazu.

- W konstruktorze przyjmuje liczbę wątków, które będą używane do przetwarzania.
- W metodzie `calculateConvolution` obraz jest dzielony na segmenty wierszy przypisywane poszczególnym wątkom.
- Każdy wątek przetwarza przypisany fragment obrazu za pomocą metody pomocniczej `applyConvolution`.
- Zastosowano klasę `ExecutorService` do zarządzania pulą wątków oraz synchronizacji ich zakończenia.

Fragment kodu:

```
ExecutorService executor = Executors.newFixedThreadPool(numThreads);
for (int i = 0; i < numThreads; i++) {
    final int threadId = i;
    executor.submit(() -> {
        int startRow = (matrixHeight * threadId) / numThreads;
        int endRow = (matrixHeight * (threadId + 1)) / numThreads;
        applyConvolution(matrix, filter, output, startRow, endRow);
    });
}
executor.shutdown();
executor.awaitTermination(Long.MAX_VALUE, TimeUnit.NANOSECONDS);
```

7.4 Klasa `BitmapManager`

`BitmapManager` odpowiada za wczytywanie i zapisywanie obrazów w formacie BMP.

- Metoda `readBMPToDoubleArray` konwertuje obraz na dwuwymiarową tablicę wartości typu `Double`, przeliczając wartości RGB na skalę szarości.

- Metoda `saveDoubleArrayToBMP` zapisuje dwuwymiarową tablicę `Double` jako obraz BMP, przekształcając wartości szarości na odpowiednie wartości RGB.

7.5 Klasa `CsvReader`

`CsvReader` jest odpowiedzialna za wczytywanie macierzy filtrującej z pliku CSV.

- Metoda `readCSVToDoubleArray` wczytuje dane z pliku CSV do dwuwymiarowej tablicy `Double`, obsługując pliki z wartościami oddzielonymi średnikami.
- Obsługiwane są błędy podczas wczytywania pliku, takie jak brak pliku lub błędny format danych.

7.6 Interfejs `Convolution`

Interfejs `Convolution` definiuje metodę `calculateConvolution`, którą muszą zaimplementować wszystkie klasy obsługujące algorytm splotowy. Dzięki temu możliwa jest łatwa wymiana implementacji (np. między wersją jednowątkową a wielowątkową) w klasie `Main`.

8. Testowanie i analiza wyników

W celu oceny efektywności zaimplementowanego algorytmu splotowego, przeprowadzono kompleksowe testy, uwzględniające różne rozmiary obrazów, wielkości filtrów oraz liczby wątków. Celem testów było zbadanie wydajności, skalowalności oraz wpływu parametrów na czas przetwarzania.

8.1 Cel testów

Testy zostały zaprojektowane tak, aby odpowiedzieć na kluczowe pytania dotyczące działania algorytmu:

- Jak zmienia się czas wykonania w zależności od rozmiaru obrazu?
- Jakie znaczenie ma wielkość filtra dla wydajności algorytmu?
- W jaki sposób wzrost liczby wątków wpływa na czas przetwarzania i kiedy osiągana jest granica wydajności?

8.2 Zestawy danych i konfiguracja

Do testów wybrano trzy obrazy w formacie BMP, które różnią się rozmiarem i poziomem szczegółowości:

- **Sample1.bmp**
  Rozmiar: 480x360, Wielkość: 500 KB
- **Sample2.bmp**
  Rozmiar: 1280x720, Wielkość: 2.6 MB
- **Sample3.bmp**
  Rozmiar: 3840x2160, Wielkość: 23.7 MB

Dla każdego obrazu algorytm przetwarzano przy użyciu filtrów o rozmiarach:

- **3x3**: Mały filtr, szybkie przetwarzanie.
- **5x5**: Typowy dla wygładzania.
- **9x9**: Duży filtr, intensywne obliczeniowo.

Liczba wątków została zwiększana stopniowo od 1 do 2,4,8 oraz dalej, aby zaobserwować granice skalowalności. Każdy test wykonywano 10-krotnie, a czas wykonania dla każdej kombinacji parametrów był uśredniany, aby wyeliminować wpływ ewentualnych fluktuacji w wydajności systemu. Dodatkowo testy przeprowadzono z użyciem flagi **`-Xint`**, co powoduje wyłączenie kompilatora Just-In-Time (JIT) w środowisku JVM. Decyzja ta została podjęta w celu uniknięcia wpływu dynamicznych optymalizacji wykonywanych przez JIT, które mogą prowadzić do niestabilnych wyników i trudności w porównywaniu wydajności w różnych konfiguracjach. W efekcie wszystkie operacje były wykonywane w trybie interpretowanym, co zapewniło bardziej przewidywalne i powtarzalne wyniki testów.

8.3 Przykład działania programu

Poniżej zaprezentowano działanie programu

a) b)

Figura 1: Przykład przetworzonego przez program obrazu a) Oryginalny obraz b) Obraz z rozmyciem Gausa

Program po procesie przetwarzania obrazu zwracał następujące dane. Dane te następnie zebrano i na ich podstawie obliczono wyniki przedstawione w następnej sekcji.

```
> ./run.sh imgs/sample1.bmp kernels/gaussian_blur.csv 4
Starting program...
Using multi-threaded convolution with 4 threads.
Elapsed time:                   39.245773 ms
Elapsed time:                   39245us
File   name:                   imgs/sample1.bmp
Filter name:                   kernels/gaussian_blur.csv
Matrix height:                 360
Matrix width:                  480
Filter height:                 3
Filter width:                  3
```

Figura 2: Uruchomienie programu i wynik działania

8.4 Wyniki testów

Tabela 1: Wyniki dla obrazu Sample1.bmp (480x360)

Rozmiar filtra	Wątki	Czas wykonania (ms)
3x3	1	140
3x3	2	75
3x3	4	43
3x3	8	48
3x3	16	47
3x3	32	45
5x5	1	345
5x5	2	185
5x5	4	98
5x5	8	100
5x5	16	102
5x5	32	98
9x9	1	1030
9x9	2	535
9x9	4	281
9x9	8	280
9x9	16	282
9x9	32	281

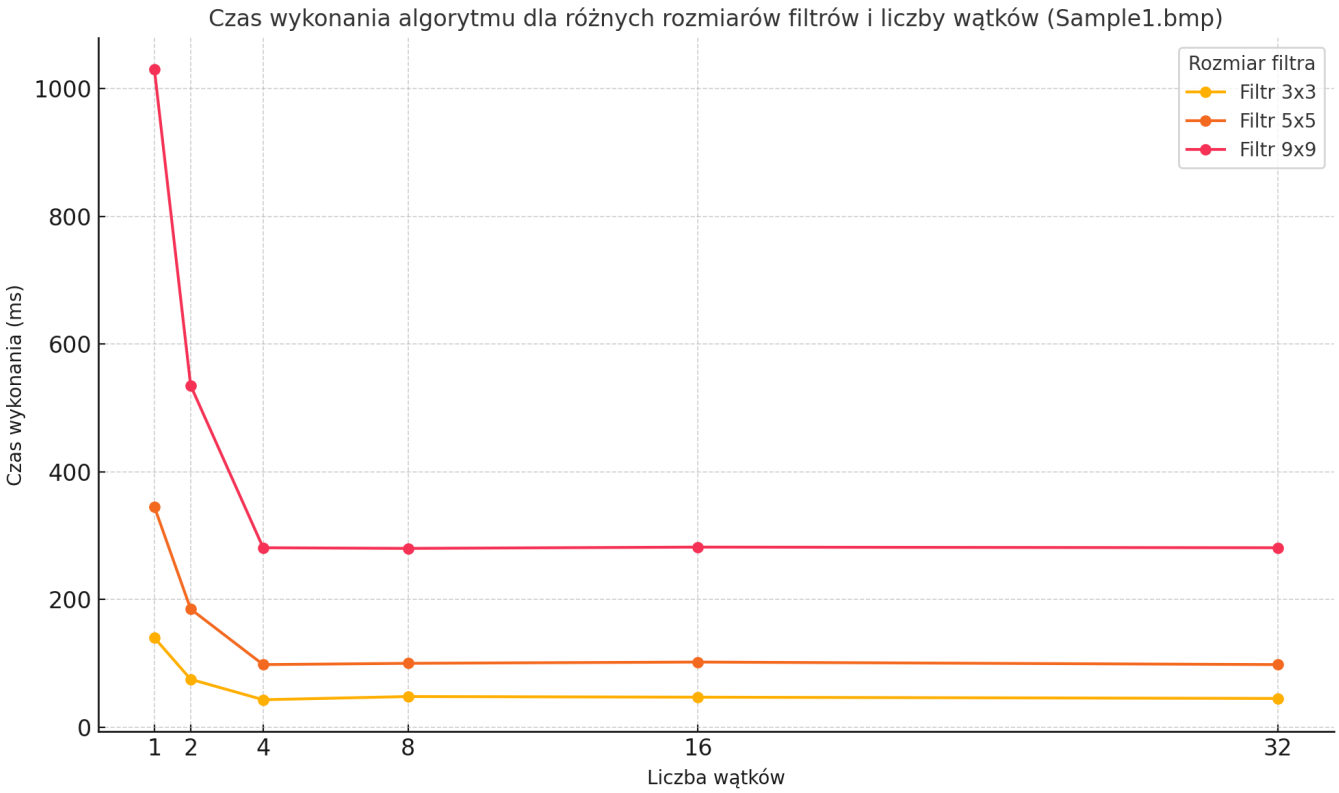

Figura 3: Wykres czasu wykonania dla Sample1.bmp

Tabela 2: Wyniki dla obrazu Sample2.bmp (1280x720)

Rozmiar filtra	Wątki	Czas wykonania (ms)
3x3	1	740
3x3	2	366
3x3	4	218
3x3	8	220
3x3	16	219
3x3	32	222
5x5	1	715
5x5	2	341
5x5	4	208
5x5	8	210
5x5	16	211
5x5	32	209
9x9	1	5375
9x9	2	2855
9x9	4	1492
9x9	8	1512
9x9	16	1508
9x9	32	1518

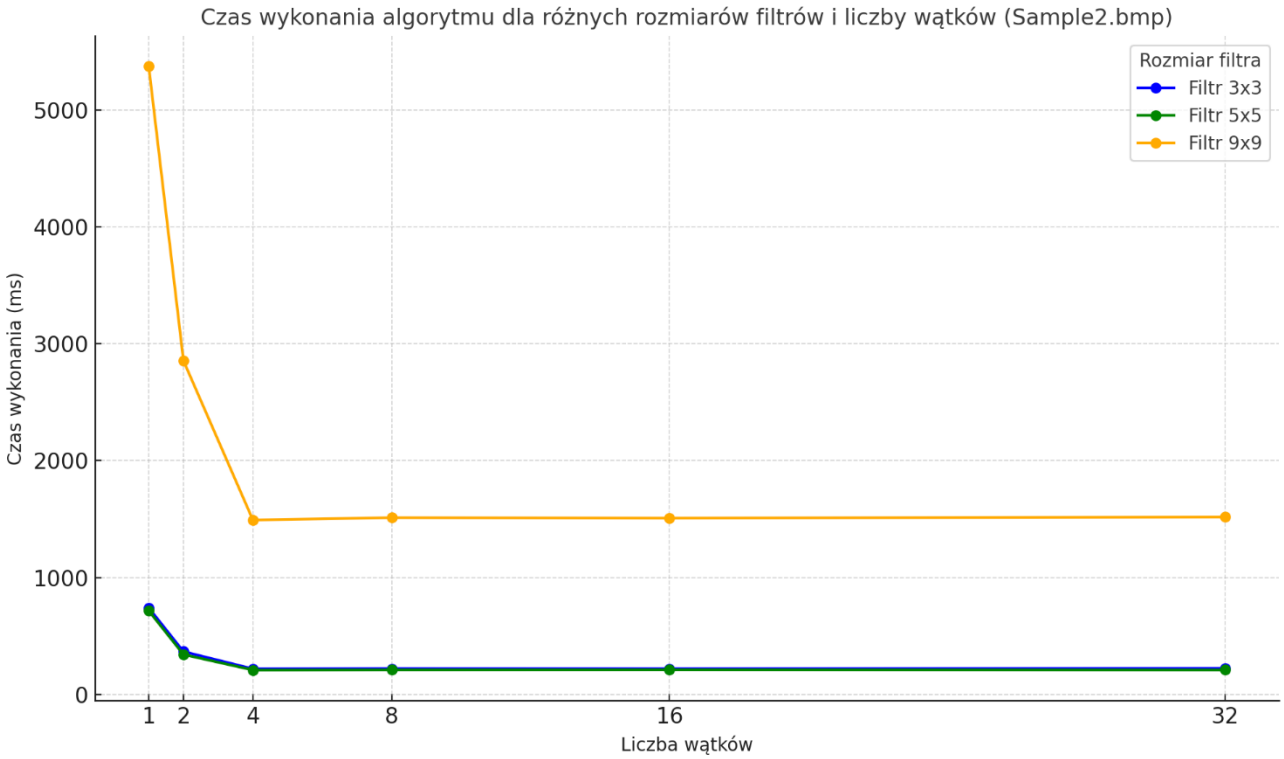

Figura 4: Wykres czasu wykonania dla Sample2.bmp

Tabela 3: Wyniki dla obrazu Sample3.bmp (3840x2160)

Rozmiar filtra	Wątki	Czas wykonania (ms)
3x3	1	6380
3x3	2	3330
3x3	4	1835
3x3	8	1849
3x3	16	1853
3x3	32	1840
5x5	1	6232
5x5	2	3390
5x5	4	1869
5x5	8	1902
5x5	16	1891
5x5	32	1915
9x9	1	47009
9x9	2	24616
9x9	4	13001
9x9	8	12736
9x9	16	13171
9x9	32	12957

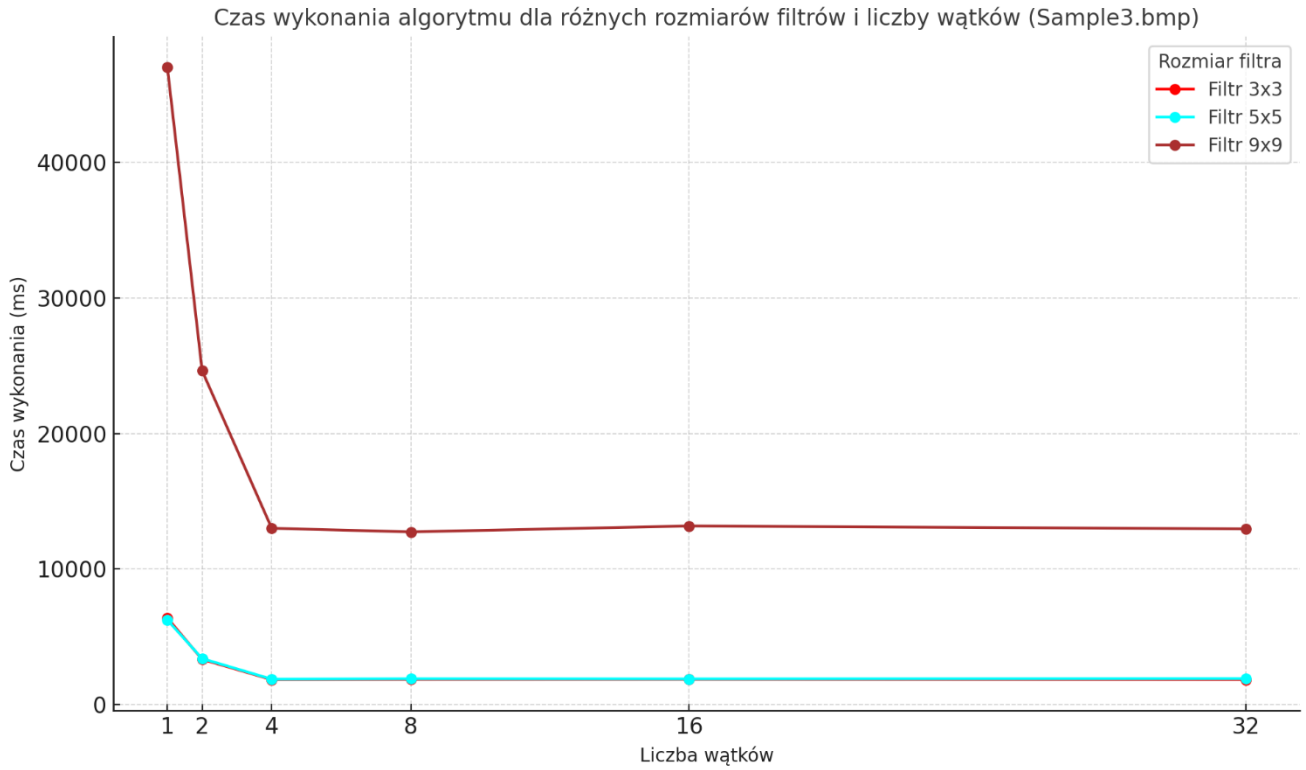

Figura 5: Wykres czasu wykonania dla Sample3.bmp

8.5 Wnioski

Na podstawie przeprowadzonych testów dla różnych kombinacji rozmiarów obrazów i wielkości filtrów zaobserwowano następujące kluczowe wnioski:

1. **Podobieństwo wykresów dla różnych konfiguracji:**
    - Niezależnie od kombinacji wielkości obrazu i rozmiaru macierzy filtrującej, wykresy przebiegu czasu wykonania w zależności od liczby wątków mają podobny kształt.
    - Punkt początkowy (czas dla jednego wątku) zależy wyłącznie od poziomu skomplikowania przetwarzania, czyli rozmiaru obrazu oraz filtra.
2. **Znaczący zysk z wielowątkowości:**
    - W każdym przypadku zastosowanie wielowątkowości przyniosło znaczące zmniejszenie czasu przetwarzania.
    - Zysk z wielowątkowości wynosił nawet kilkukrotne skrócenie czasu wykonania algorytmu w porównaniu do wersji jednowątkowej.
3. **Optymalna liczba wątków:**
    - Dla każdego testowanego przypadku największe korzyści osiągano przy liczbie wątków ustawionej na **4**, taka liczba wątków może wynikać z liczby dostępnych rdzeni procesora.
    - Po przekroczeniu tej liczby wykres wyraźnie się wypłaszczał, co oznacza, że dalsze zwiększanie liczby wątków nie przynosiło istotnych korzyści czasowych.
4. **Brak opłacalności dużej liczby wątków:**
    - Choć w niektórych przypadkach dalsze zwiększanie liczby wątków (np. do 8, 16 czy 32) mogło prowadzić do minimalnego skrócenia czasu wykonania, zyski były marginalne.
    - Dodatkowe wątki generowały narzut związany z synchronizacją i zarządzaniem zasobami, co czyniło ich użycie nieopłacalnym w kontekście efektywności zasobów.
5. **Ogólna rekomendacja:**
    - Dla dowolnej kombinacji obrazu i filtra zaleca się ustawienie liczby wątków na poziomie **4**, czyli liczbie równej liczbie dostępnych rdzeni procesora, co zapewnia najlepszy kompromis między czasem przetwarzania a efektywnym wykorzystaniem zasobów systemowych.

Podsumowując, wielowątkowość jest kluczowym elementem optymalizacji algorytmu splotowego, ale jej skuteczność zależy od dostosowania liczby wątków do rozmiaru obrazu i poziomu złożoności obliczeń.

Literatura

[1] Artur Gramacki: Operacja splotu (ang. convolution). Uniwersytet Zielonogórski Instytut Sterowania i Systemów Informatycznych, 2020
[2] Piotr Dalka: Filtracja obrazu – wykład. Katedra Systemów Multimedialnych, Politechnika Gdańska
[3] Jacek Czaja: Parallelization of Convolution using OpenMP, Intel Corporation January 14, 2022
[4] Oracle: Java documentation https://docs.oracle.com/en/java/, dostęp 24.11.2024
[5] Oracle: Java documentation – java.io packet https://docs.oracle.com/javase/8/docs/api/?java/io, dostęp 24.11.2024