Instytut Informatyki Politechniki Śląskiej
Zespół Mikroinformatyki i Teorii Automatów Cyfrowych

Rok akademicki	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/SMiW):	Grupa	Sekcja
2020/2021	**SSI**	**Języki Asemblerowe**	**1**	**2**

Imię	**Łukasz**	**Prowadzący**: OA/JP/KT/GD/BSz/GB	**JP**
Nazwisko	**Ważny**		

Raport końcowy

Temat projektu:

Przekształcenia obrazów binarnych

Data oddania: dd/mm/rrrr	**07/11/2020**

Główne założenia projektu:

1. Przekształcenie zadanego obrazu binarnego według algorytmu dylatacji.
2. Możliwość zobaczenia histogramu obrazu oryginalnego i przetworzonego.

Założenia części głównej projektu w języku wysokiego poziomu:

1. Wykonanie graficznego interfejsu użytkownika z wykorzystaniem WPF w języku C#.
2. Obsługa wątków, podzielenie obrazu na części oraz przydzielenie każdej części do innego wątku, połączenie obrazu w całość, po dokonanych przekształceniach.
3. Pomiar szybkości wykonanego algorytmu.
4. Stworzenie histogramu obrazu oryginalnego i przetworzonego.
5. Możliwość zapisania obrazu.

Założenia projektu dla funkcji biblioteki:

1. Biblioteka wysokiego poziomu zostanie napisana w języku C.
2. Będzie odpowiedzialna za sam algorytm dylatacji.
3. Użycie instrukcji wektorowych w bibliotece asemblerowej.

Analiza zadania oraz uzasadnienie wyboru rozwiązania:

Program wymaga napisania dwóch bibliotek w Asemblerze oraz w C. Obie biblioteki zostaną wykonane w architekturze 64 bitowej. Zgodnie z ogólnymi wymaganiami biblioteka asemblerowa będzie wykorzystywała instrukcje wektorowe.

W trakcie dogłębnej analizy doszedłem do wniosku, iż aby otrzymać efekt wywoływany przez algorytm dylatacji należy rozdzielić podany algorytm na dwie pętle.

Pierwsza pętla kopiuje bez zmian obraz źródłowy do tablicy przeznaczonej na obraz docelowy. Dzięki temu algorytm będzie badał otoczenie pikseli jedynie na obrazie źródłowym, a wynik zapisywał jedynie do obrazu docelowego, co z kolei spowoduje, że otoczenie piksela nie będzie badane w oparciu o przetworzone już piksele, co spowodowałoby błędne wyniki.

Druga pętla będzie odpowiadać za właściwy algorytm dylatacji. W jej obrębie zostaną stworzone 4 instrukcje warunkowe, z czego pierwsze 3 będą rozpatrywać szczególne przypadki, gdy po przyłożeniu do badanych pikseli elementu strukturalnego w punkcie centralnym, element strukturalny wychodziłby poza obszar obrazu z: 1. lewej i górnej strony jednocześnie, 2. tylko z lewej strony, 3. tylko z górnej strony. 4-ta instrukcja warunkowa rozpatruje wszystkie inne przypadki. Wewnątrz instrukcji warunkowych znajdują się pętle sprawdzające, czy w obrębie przyłożonego do badanego piksela elementu strukturalnego znajduje się przynajmniej jeden czarny piksel (kolejna instrukcja warunkowa wewnątrz pętli) i jeżeli tak jest - do badanego piksela przypisywana jest wartość 0 (czarny). Każda z pętli wewnątrz czterech instrukcji warunkowych jest podobna, jednak lekko zmodyfikowana, uwzględniając różnice, wynikające z rozpatrywanych przypadków szczególnych - podobnie z instrukcjami warunkowymi wewnątrz tych pętli.

Wspomniane wyżej modyfikacje pętli, wykonujących właściwy algorytm w przypadkach szczególnych, mają za zadanie poradzenie sobie z pikselami krańcowymi w następujący sposób - wiersze i kolumny pikseli, które normalnie byłyby potrzebne są lekceważone i otoczenie piksela jest wyznaczane w oparciu tylko o sąsiedztwo istniejących pikseli.

W projekcie graficzny interfejs użytkownika został zaimplementowany w języku C# z wykorzystaniem technologii WPF, ponieważ posiadam doświadczenie w tworzeniu aplikacji na tę właśnie technologię oraz jest ona bardzo intuicyjna w tworzeniu interfejsu użytkownika ze względu na możliwość przeciągania i umieszczania elementów w tzw. „designerze” za pomocą elementów z „toolboxa”.

Wykorzystuję również język C dla stworzenia biblioteki zgodnie z założeniami projektu. Zdecydowałem się na ten język z powodu łatwości implementacji algorytmu w C oraz łatwego eksportowania funkcji tego języka do pliku DLL (wystarczą dwie linijki w wierszu poleceń, korzystające z kompilatora gcc).

Wykorzystanie asemblera w projekcie było konieczne, natomiast ja wykorzystuje konkretnie MASM 64-bitowego w realizacji mojego projektu ze względu na doświadczenie nabyte w trakcie zajęć laboratoryjnych z przedmiotu Języki Asemblerowe.

Wprowadzenie

Przekształcenia morfologiczne są jednymi z ważniejszych operacji przetwarzania obrazów, gdyż pozwalają przeprowadzić zaawansowaną analizę kształtów poszczególnych obiektów oraz odległości między nimi. Podstawowe przekształcenia morfologiczne można ze sobą łączyć, co daje podstawę do budowania skomplikowanych systemów analizy obrazu.

Przetwarzanie obrazów binarnych odbywa się z wykorzystaniem morfologii matematycznej, teorii analizy i przetwarzania struktur geometrycznych opartej na teorii mnogości, kratach, topologii i funkcjach losowych.

Przekształceniem morfologicznym, które zostało zaimplementowane w ramach tego projektu, jest dylatacja, zwana również rozszerzeniem, ponieważ właściwie właśnie tego dokonuje z obrazem.

Konsekwencją dylatacji jest zwiększenie obiektu, zniknięcie detali i wypełnienie „dziur” w niespójnym obszarze. Często, aby uzyskać pożądany efekt stosuje się dylatacje wielokrotną.

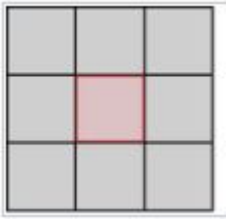
Pojęcie elementu strukturalnego

Pojęcie elementu strukturalnego jest kluczowe dla zrozumienia algorytmu dylatacji, jak i wszystkich algorytmów morfologicznych.

Właściwości filtrów morfologicznych określane są przez tzw. element strukturalny, wykorzystywany jako ruchome okno. Określony jest względem wybranego piksela, tzw. punktu centralnego lub początkowego.

Element strukturalny może przybrać dowolną szerokość i wysokość, zaś jego punkt centralny może być w dowolnym miejscu.

Jednym z częściej stosowanych elementów strukturalnym jest kwadrat o boku 3 (wysokość i szerokość równa 3) z punktem centralnym w środku (współrzędne x=1 i y=1, licząc pierwszą współrzędną od 0).



Przykładowy element strukturalny

Algorytm Dylatacji

Dylatacja, czyli rozszerzanie, jest zastosowaniem sumy Minkowskiego do obrazu, co można przedstawić wzorem:

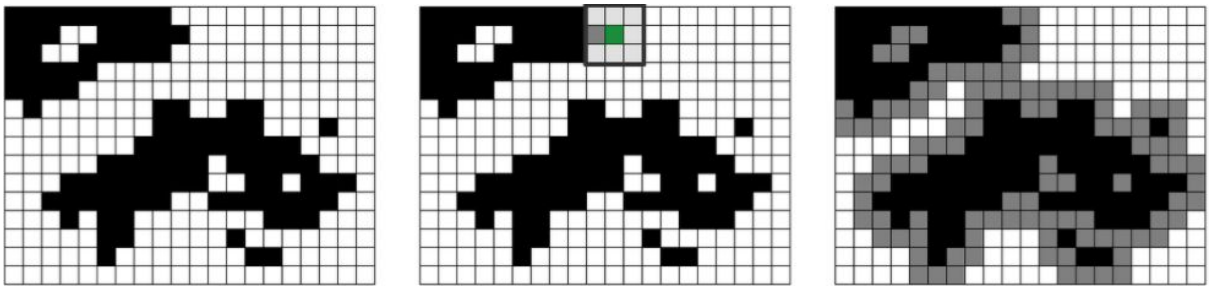
$$I \oplus B \equiv \{p \in U_1 | \tilde{B} \cap I \neq \emptyset\}$$

lub

$$I \oplus B \equiv \bigcup_{p \in B} (I + p).$$

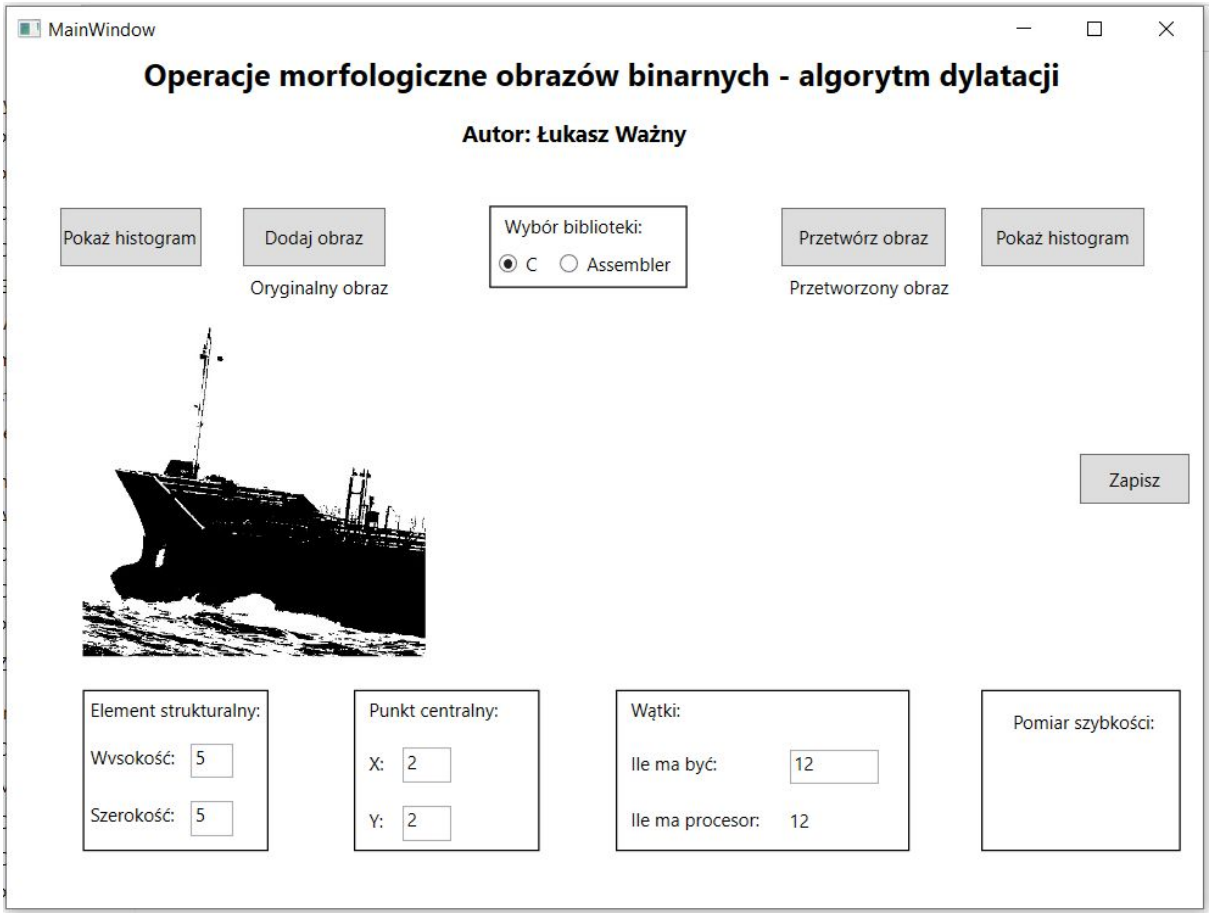
Operacja ta przykłada element strukturalny w punkcie centralnym do każdego piksela na obrazie. Jeżeli choć jeden piksel z sąsiedztwa objętego przez element strukturalny (*B*) ma wartość równą jeden (czarny), dany piksel również otrzymuje wartość jeden.

Istotny wpływ na wynik dylatacji ma wybór elementu strukturalnego. Przy niestandardowym kształcie rozrost obiektu jest kierunkowy (np. szerokość znacznie większa niż szerokość). Z kolei jeżeli punkt centralny nie leży w środku elementu, rozrost jest niesymetryczny. Najbardziej neutralny efekt daje wykorzystanie elementu kwadratowego o punkcie centralnym w środku (patrz: przykładowy piksel w poprzednim rozdziale).



Przykład wykonania dylatacji na obrazie binarnym z wykorzystaniem elementu strukturalnego opisanego w rozdziale powyżej.

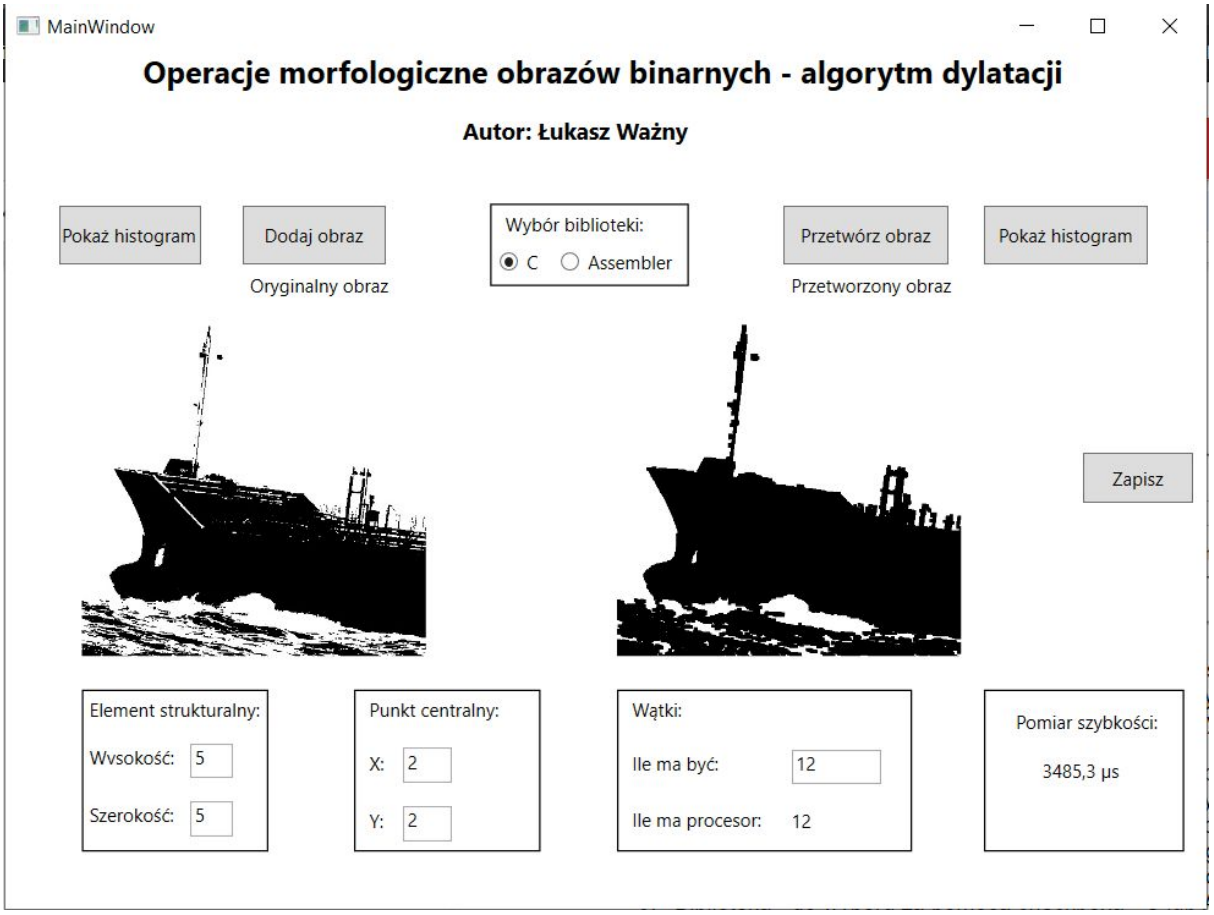
Opis działania programu



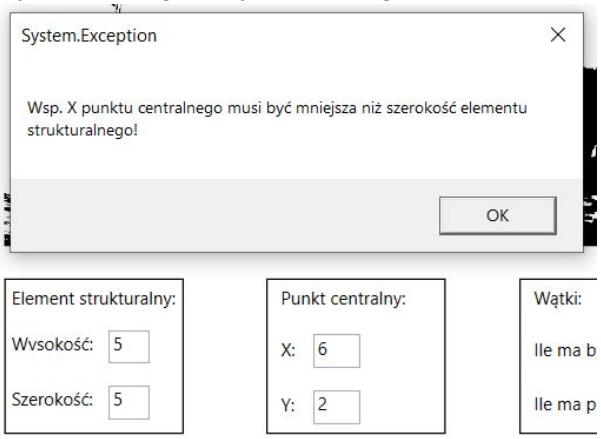
Parametry wejściowe wczytywane są poprzez GUI, a są nimi:

1. Obraz binarny - wybierany z dysku po naciśnięciu przycisku dodaj obraz
2. Wysokość i szerokość elementu strukturalnego - wpisywana w przeznaczonych do tego polach - zakres, to dowolna liczba nieujemna.
3. Współrzędne X,Y punktu centralnego - wpisywane w przeznaczonych do tego polach - zakres, to dowolna liczba nieujemna, mniejsza od wysokości lub szerokości elementu strukturalnego (tak aby punkt nie wyszedł poza obszar elementu)
4. Liczba wątków - wpisywana w przeznaczonego do tego polu - zakres, to dowolna liczba naturalna, mniejsza od szerokości obrazu źródłowego.
5. Biblioteka - do wyboru za pomocą checkboxa - C lub Asembler.

Po naciśnięciu przycisku przetwórz obraz, dokonywana jest operacja dylatacji, a jej efekt jest wyświetlany w okienku po prawej:



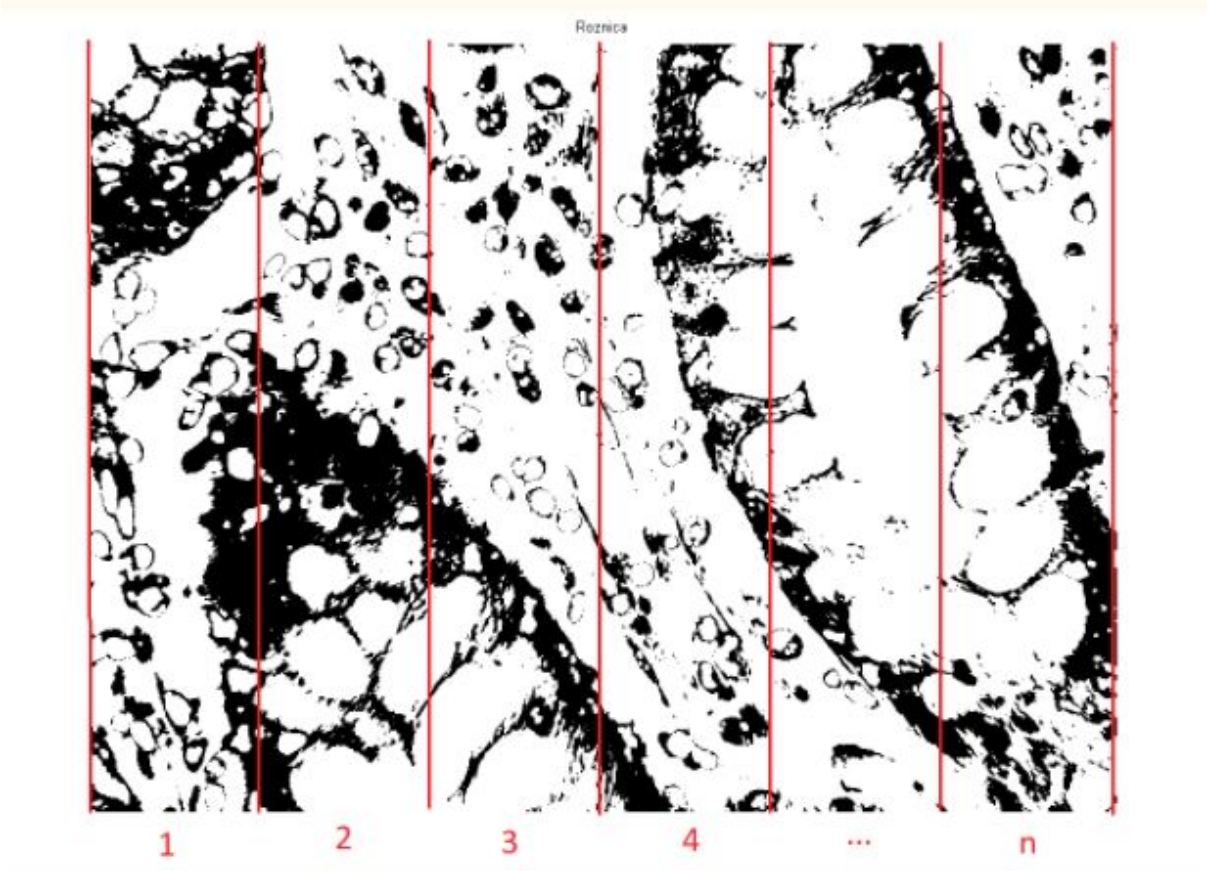
Program również posiada zabezpieczenia przed wczytaniem dowolnego parametru spoza zakresu, jak i przed przetworzeniem obrazu bez uprzedniego wczytania go z dysku. Wszystkie błędy wyświetlane są w osobnym okienku *MessageBox*, które wyświetla się po wystąpieniu błędu, wraz z opisem błędu. Okienko można zamknąć i kontynuować pracę z programem, bez potrzeby ponownego włączania programu:



Przykład błędu - współrzędna X jest poza obszarem elementu strukturalnego.

Obsługa wątków

Obraz dzielony jest na “paski” pionowe według następującego sposobu (dla *n* wątków):



Odpowiedzialna jest za to metoda w języku C#, która dzieli w ten sposób obraz, uwzględniając przypadki, gdy szerokość obrazu jest niepodzielna przez liczbę wątków - wtedy do ostatniego “paska” dodawane są pozostałe kolumny, więc jest on szerszy od pozostałych. Ponadto do “pasków” dodawane są kolumny pikseli nadmiarowe po lewej i prawej, aby nie było przekłamań na krańcach “pasków” po dokonaniu algorytmu. Po przetworzeniu “pasków”, obraz jest łączony z pominięciem nadmiarowych kolumn. Każdy z “pasków” jest przydzielany do osobnego wątku, które następnie są przeprowadzają równolegle operacje.

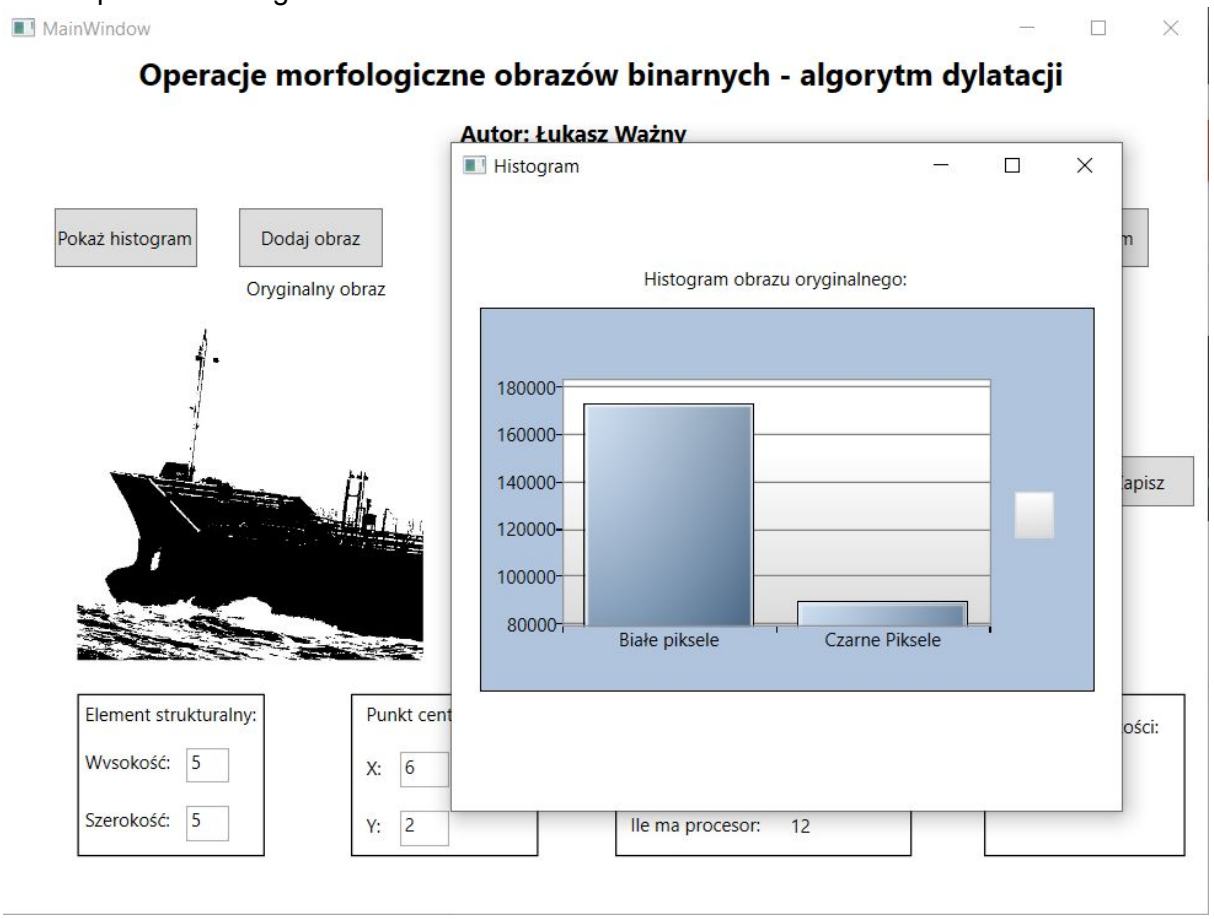
Pomiar szybkości

Pomiar szybkości dokonania operacji realizowany jest za pomocą klasy *Stopwatch* w języku C#, w następujący sposób:

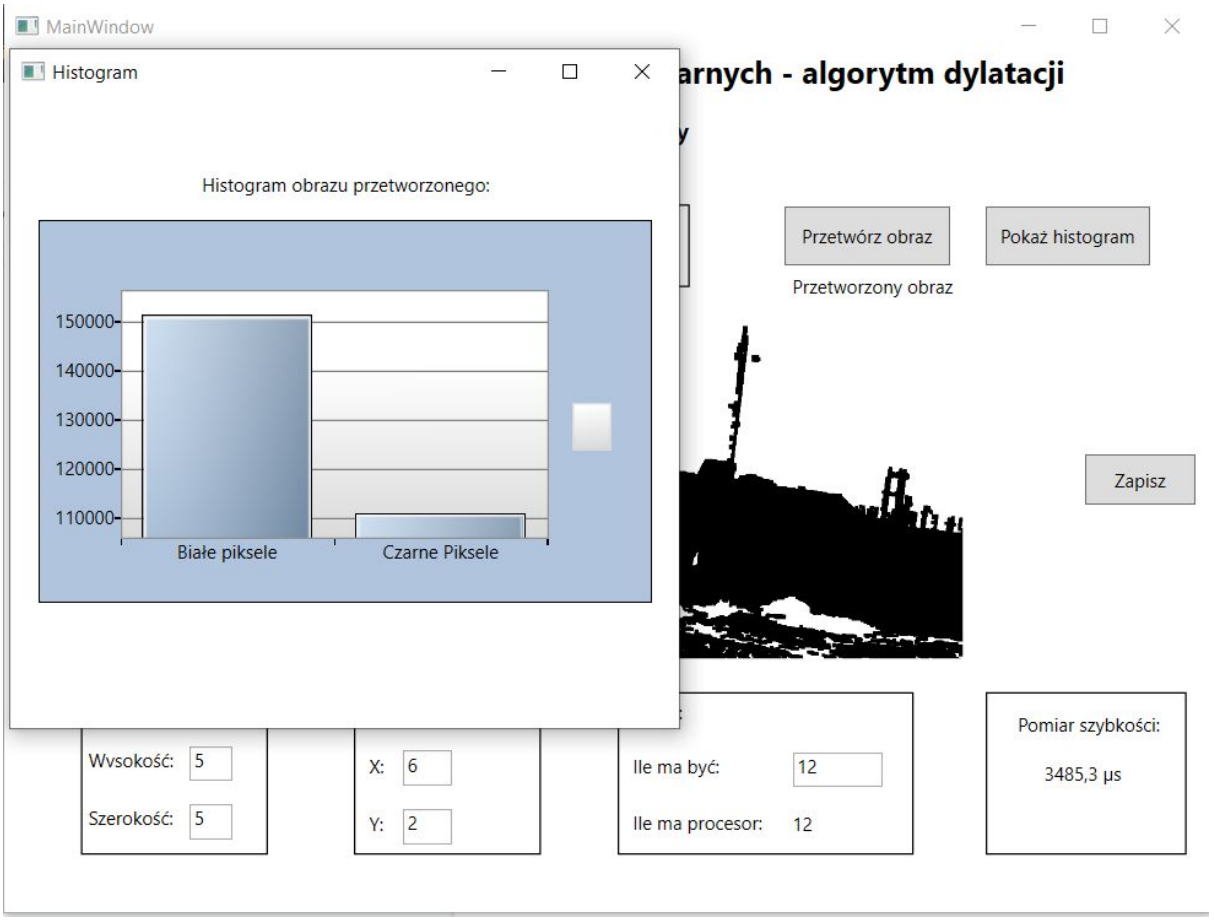
```
//pomiar czasu
Stopwatch watch = new Stopwatch();
watch.Start();
//operation
action.start();
watch.Stop();
speed.Text = ((((double) watch.ElapsedTicks) / ((double) Stopwatch.Frequency)) * 1000 * 1000) +
             " \u00b5s";
```

Histogram

Histogram obliczany jest w języku C#, poprzez proste zliczenie pikseli białych i czarnych, a następnie wyświetlenie ich na wykresie słupkowym, korzystając z biblioteki *System.WIndows.Controls.DataVisualization*. Histogram wyświetlany jest po naciśnięciu przycisku *Pokaż histogram* z lewej strony dla obrazu oryginalnego lub z prawej strony dla obrazu przetworzonego.



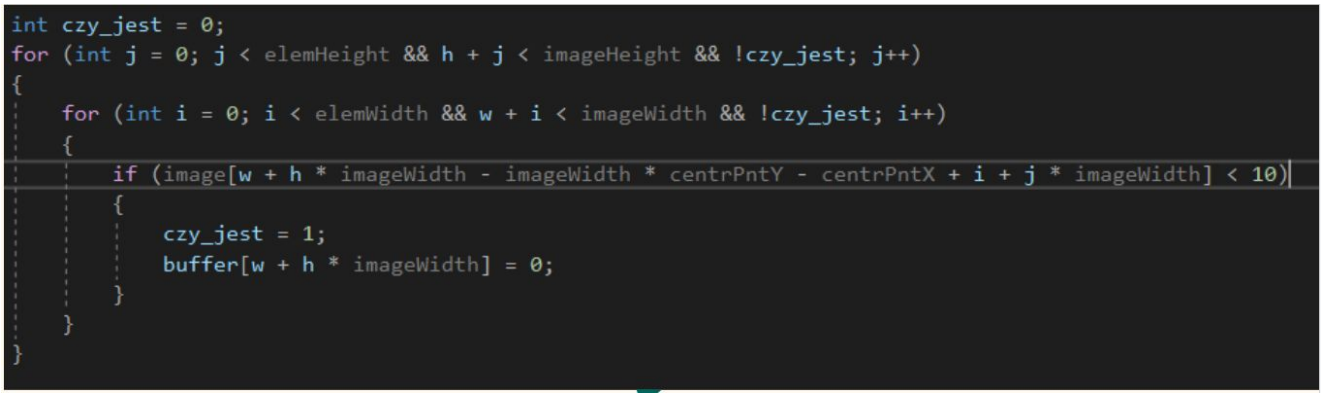
Przykładowy histogram dla obrazu oryginalnego



Przykładowy histogram dla obrazu przetworzonego

Opis funkcji biblioteki języka C

Poniżej znajduje się screen fragmentu kodu funkcji języka C



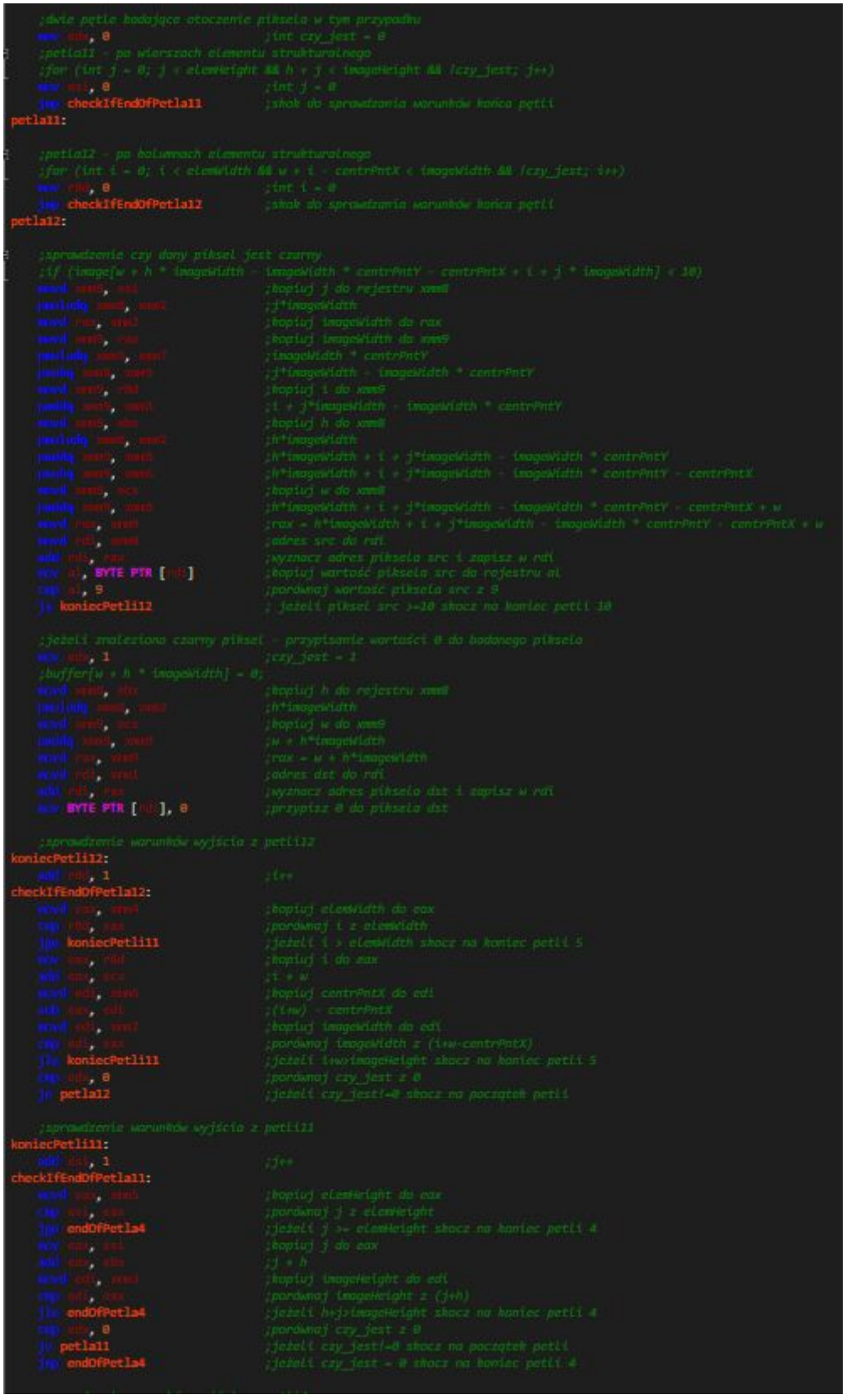
```
int czy_jest = 0;
for (int j = 0; j < elemHeight && h + j < imageHeight && !czy_jest; j++)
{
    for (int i = 0; i < elemWidth && w + i < imageWidth && !czy_jest; i++)
    {
        if (image[w + h * imageWidth - imageWidth * centrPntY - centrPntX + i + j * imageWidth] < 10)
        {
            czy_jest = 1;
            buffer[w + h * imageWidth] = 0;
        }
    }
}
```

Jest to fragment z ostatniej instrukcji warunkowej, który rozpatruje przypadek, gdy piksel nie jest w miejscu “krańcowym”. Cały kod nie będzie omawiany ponieważ jest on po prostu implementacją w C podejścia, które przedstawiłem dosyć szczegółowo na samym początku raportu w *Analizie zadania*.

Jak widać powyższy fragment za pomocą dwóch pętli oraz instrukcji warunkowej bada, czy jakikolwiek piksel w obrębie elementu strukturalnego ma wartość mniejszą niż 10 (jeżeli jakieś piksele nie byłyby “idealnie” białe, to mogą mieć wartość trochę większą niż 0, stąd badamy, czy jest mniejsze niż 10, a nie równe 0). Jeżeli tak jest - przypisywana jest do badanego piksela wartość 0.

Opis funkcji biblioteki w języku Asembler

Poniżej znajduje się screen fragmentu kodu, który odpowiada fragmentowi powyżej w języku C.



Fragment jest dosyć szczegółowo skomentowany w pliku *.asm*, więc nie ma sensu go tutaj szczegółowo omawiać. Realizuje on dokładnie to samo, co przedstawiony wcześniej fragment w języku C. Jak widać jest on znacznie bardziej rozbudowany (80 linijek Asemblera vs 12 linijek w C). Jest to spowodowane faktem, iż pętle oraz instrukcja warunkowa w języku

C, posiadają wiele warunków do sprawdzenia oraz wymagają wielu operacji arytmetycznych w tym celu, a każdy warunek, operacja, wymaga przynajmniej jednej dodatkowej linijki w Asemblerze, a często nawet kilku linijek.

Na szczególną uwagę zasługuje wiele użytych instrukcji wektorowych, które bardzo przyspieszają działanie algorytmu, takich jak:

- pmuludq xmm8, xmm2 - mnożenie liczb całkowitych dodatnich DWORD
- psubq xmm8, xmm9 - odejmowanie liczb całkowitych QWORD
- paddq xmm9, xmm8 - dodawanie liczb całkowitych QWORD

Opis testowania i uruchamiania programu

Program uruchamiany jest przez zwykłe dwukrotne kliknięcie pliku *.exe* dostępnego w folderze *Release*, należy pamiętać, aby w tym samym folderze znalazły się 4 pliki DLL:

- *asm_function.dll* - biblioteka z funkcją asemblerową
- *CFunction.dll* - biblioteka z funkcją w języku C
- *System.Windows.Controls.DataVisualization.Toolkit.dll* - używana w programie biblioteka zewnętrzna
- *WPFToolkit.dll* - używana w programie biblioteka zwenętrzna

Ponadto na komputerze powinien być zainstalowany pakiet redystrybucyjny Microsoft Visual C++ procesora x64 oraz platforma programistyczna .NET Framework Microsoft w wersji przynajmniej 4.7.2 - jest to niezbędny warunek uruchomienia programu.

Cała reszta informacji niezbędnych do testowania programu znajduje się w rozdziale *Opis działania programu*, tak więc nie będzie tu powtarzana.

Program został przetestowany na wielu parametrach, czego wyniki są przedstawione w rozdziale *Raporty szybkości*.

Pliki do testowania znajdują się w folderze *Release\obrazy_do_testowania*.

Raporty szybkości

Program został przetestowany na obrazie 512 x 512 pikseli na komputerze o specyfikacji :

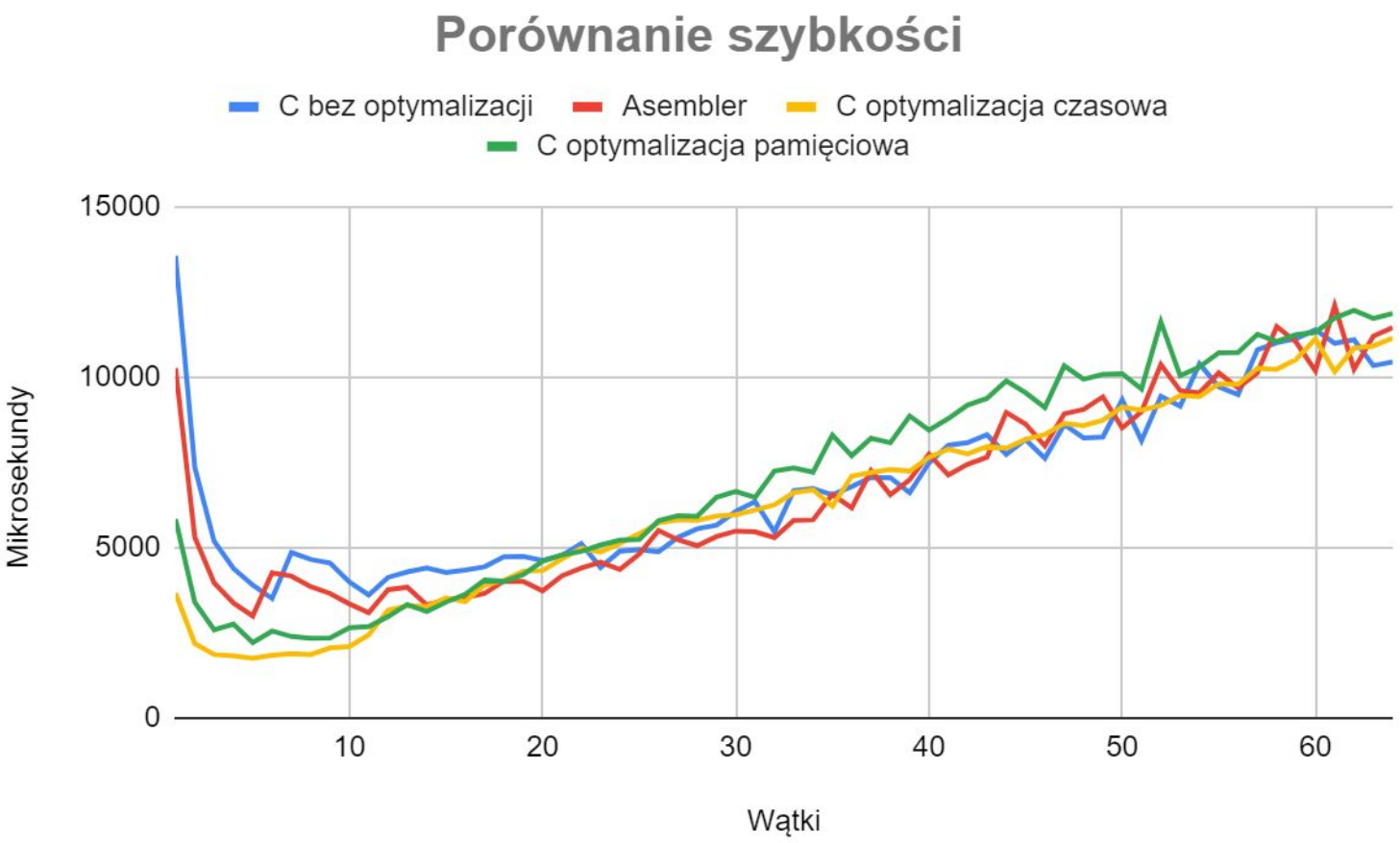
- Intel Core i7-8750H 2.2 GHz (12 wątków),
- 8 GB DDR4 Ram,
- 1 TB HDD,
- Windows 10 64-bit.

Wynik testowania w formie tabelarycznej (jednostka - mikrosekundy):

Wątki	C bez optymalizacji	Asembler	C optymalizacja czasowa	C optymalizacja pamięciowa
1	13558,9	10264,7	3660,6	5833,1
2	7348,1	5301,5	2186,5	3389,7
3	5184,5	3965,4	1864,1	2589,9
4	4387,8	3377,4	1826,7	2757,7
5	3898,3	2995,1	1753,7	2217,3
6	3507,2	4257,4	1844,2	2551,7
7	4854,5	4164,3	1884,6	2391,2
8	4651,5	3853,6	1863,5	2344,6
9	4548,9	3653,6	2059,2	2348
10	3991,9	3349,5	2094,4	2650
11	3611,5	3091,7	2437,4	2684,4
12	4125,6	3764,8	3163,5	2977,9
13	4284,6	3835,7	3291,6	3331,3
14	4403,6	3321,7	3268,7	3122,7
15	4268,4	3466,7	3522,6	3398,5
16	4341,6	3536,3	3413,6	3633,2
17	4436,3	3654,6	3907,9	4049,2
18	4728	4018,3	4037,9	4010,9
19	4740,1	4005,6	4306,5	4211
20	4620,2	3729,6	4326,8	4599,5
21	4764,2	4172,3	4666,8	4785,2
22	5111,9	4399,6	4976,6	4886,3
23	4418,2	4575,2	4873	5074,4
24	4890,6	4358,3	5124,7	5217,4
25	4938,6	4808,5	5404,2	5234,9
26	4880,8	5497,1	5723,1	5785,7
27	5298,3	5229,6	5812,6	5935,2
28	5547,8	5055	5801,2	5917,5
29	5655,2	5320	5927,1	6476
30	6059,1	5484	5964,4	6648,3
31	6347,9	5459,6	6099,5	6470,6
32	5462,3	5294,7	6252,3	7244,7
33	6667,7	5795,2	6611,3	7337,1

34	6732,1	5811	6692,3	7208,5
35	6546,3	6554,5	6214,3	8299,7
36	6790,6	6163,1	7093,9	7695
37	7054,6	7254,8	7202,6	8203,8
38	7048,3	6546,1	7293,1	8075,5
39	6607,9	6985,9	7242,9	8857,1
40	7481,4	7742,9	7647,6	8442,3
41	7997,5	7126,2	7887,3	8781,3
42	8077,1	7440,7	7750	9179,2
43	8306,1	7650,4	7961,7	9377
44	7735,2	8963,3	7913,4	9890,3
45	8168,4	8623,5	8172,3	9541,9
46	7617,9	7979,7	8314,3	9107,9
47	8606	8924,3	8636,1	10330,5
48	8217	9051,4	8576,5	9938,1
49	8242,6	9420,1	8731,1	10075,7
50	9335	8511,7	9121,6	10097,1
51	8134,4	8985	9029,3	9647,4
52	9435,5	10367	9165,3	11619,4
53	9144,5	9600,5	9455,1	10039,2
54	10386,5	9536	9431,1	10291,1
55	9717,2	10122,1	9801,9	10715,5
56	9488	9679,1	9790,9	10720,5
57	10800,9	10123,4	10260,9	11250,6
58	11005,3	11486,4	10227,3	11039,6
59	11124,6	11028,2	10513,8	11244
60	11385,7	10184,9	11115,8	11311,3
61	10990,2	12100,6	10159,4	11732,6
62	11092,5	10230,2	10850,7	11956,2
63	10335,2	11200	10912,5	11721,9
64	10445,2	11448,1	11140,7	11865

Wynik testowania w formie wykresu:



Wyniki testowania, nie są dużym zaskoczeniem. W każdym z przypadków widzimy charakterystyczne minimum dla liczby wątków bliskiej liczbie dostępnych wątków w procesorze.

Najszybsze okazało się C z optymalizacją czasową kompilatora *gcc*, co więcej najniższa szybkość okazała się naprawdę imponująca - 1826,7 mikrosekund. Jednakże przy większej ilości wątków - najlepsza szybkość optymalizacji czasowej, nie jest już tak oczywista.

Ten aspekt - czyli bardzo różne, wahające cię, niestabilne wyniki przy dużej ilości wątków (widzimy to we wszystkich przypadkach) - wymaga szerszego komentarza.

Na pomiar szybkości wykonywania algorytmu, składa się nie tylko szybkość samego wykonanego w wątkach algorytmu, ale również wiele innych czynników, jak chociażby procesy kolejkowania w systemie operacyjnym i wszystkie inne procesy związane z obsługą wątków, jak i programy działające w tle, które są aktywne z różnym natężeniem. To wszystko stanowi dosyć duży czynnik losowy, który - w przypadku małej liczby wątków - nie jest aż tak istotny, ponieważ do wątków przekazywane są dosyć duże fragmenty obrazu, powstałe po jego podzieleniu, w związku z tym algorytm w pojedynczym wątku wykonuje się stosunkowo dłużej niż wszystkie procesy wymienione na początku. Natomiast przy dużej liczbie wątków, do pojedynczego wątku, w przypadku testowanego obrazu, mogły być przekazywane fragmenty o szerokości nawet tylko 8 pikseli. Tak więc sam algorytm był wykonywany błyskawicznie, a procesy związane z obsługą wątków nasilały się ze względu na dużą ich ilość, co właśnie powoduje zauważalną na końcu wykresu niestabilność.

Ze względu na tę właśnie niestabilność, nie można wyciągać żadnych wniosków na podstawie pomiarów dokonanych na dużej ilości wątków. Dlatego też fakt, iż przy większej ilości wątków prawie każdy przypadek (oprócz C z optymalizacją pamięciową), miał szansę być najszybszy - o niczym nie świadczy.

Warto też zwrócić uwagę na fakt, iż Asembler w porównaniu z C bez optymalizacji wyszedł znacznie lepiej. Prawdopodobnie jest to związane z wielokrotnym używaniem przeze mnie instrukcji wektorowych, które znacznie wpłynęły na szybkość algorytmu.

Wnioski

Projekt pozwolił mi na dogłębne zapoznanie się z działaniem MASM w wersji 64 bitowej poprzez wykorzystywanie instrukcji 64 bitowych oraz sposobu przyjmowania parametrów poprzez rejestry rcx, rdx, r8, r9 oraz przez stos. Nauczyłem się tworzyć biblioteki programu oraz łączyć wykorzystanie ich w innym języku programowania niż zostały one same stworzone. Nabyłem istotną wiedzę w programowaniu w języku C#, C oraz Asembler. Sam projekt przypomniał mi jak zachodzą procesy morfologiczne przetwarzania obrazów - dylatacja, ale również przypomniał algorytmiczne zasady rozwiązywania zadań. Największą trudnością podczas tworzenia projektu było podłączeniu obu bibliotek programu i ich poprawna komunikacja z językiem wysokiego poziomu oraz szukanie błędów w algorytmie - ponieważ pierwsza jego wersja niestety nie działała dobrze dla wszystkich pikseli i należało dokonać kilku korekt. Pod koniec mogę stwierdzić, iż program sprostał moim początkowym założeniom i działa zgodnie z moim zamyśleniem.