	Instytut Informatyki Politechniki Śląskiej **Zespół Mikroinformatyki i Teorii Automatów Cyfrowych**			
Rok akademicki	**Rodzaj studiów (SSI/NSI/NSM):**	**Przedmiot:**	**Grupa**	**Sekcja**
22 /23	**SSI**	**JA proj.**	**2**	**2**
Prowadzący Przedmiot:	mgr inż. Ernest Antolak			
Termin: (dzień)	**wtorek**	**Termin: (godzina)**	**10:00-11:30**	
Imię:	**Jakub**			
Nazwisko:	**Hoś**			
Email:	**jakuhos330@student.polsl.pl**			

Karta projektu

Temat projektu:

Mieszanie obrazów

Główne założenia projektu:

Stworzenie aplikacji pozwalające na „wymieszanie” dwóch obrazów. Użytkownik wybiera wagę obrazów, ilość wątków na jakich będą realizowane obliczenia oraz bibliotekę służącą do wykonania algorytmu – biblioteka języka wysokopoziomowego(C#) lub biblioteka języka asemblera.

Po wykonaniu użytkownik widzi obraz wynikowy, który zostaje napisany na dysku oraz czas wykonania algorytmu.

Opis problemu

Zagadnienie dotyczy łączenia dwóch obrazów w jeden. Aby algorytm zadziałał obrazy muszą posiadać taką samą rozdzielczość. Obraz wynikowy ma rozdzielczość obrazów początkowych. Istotny jest współczynnik u<0,1>, który mówi o wagach obrazów.
Każdy pojedynczy bajt obrazu wynikowego wyznaczamy za pomocą zależności:

$$\boldsymbol{P = Au + B(1 - u)},$$

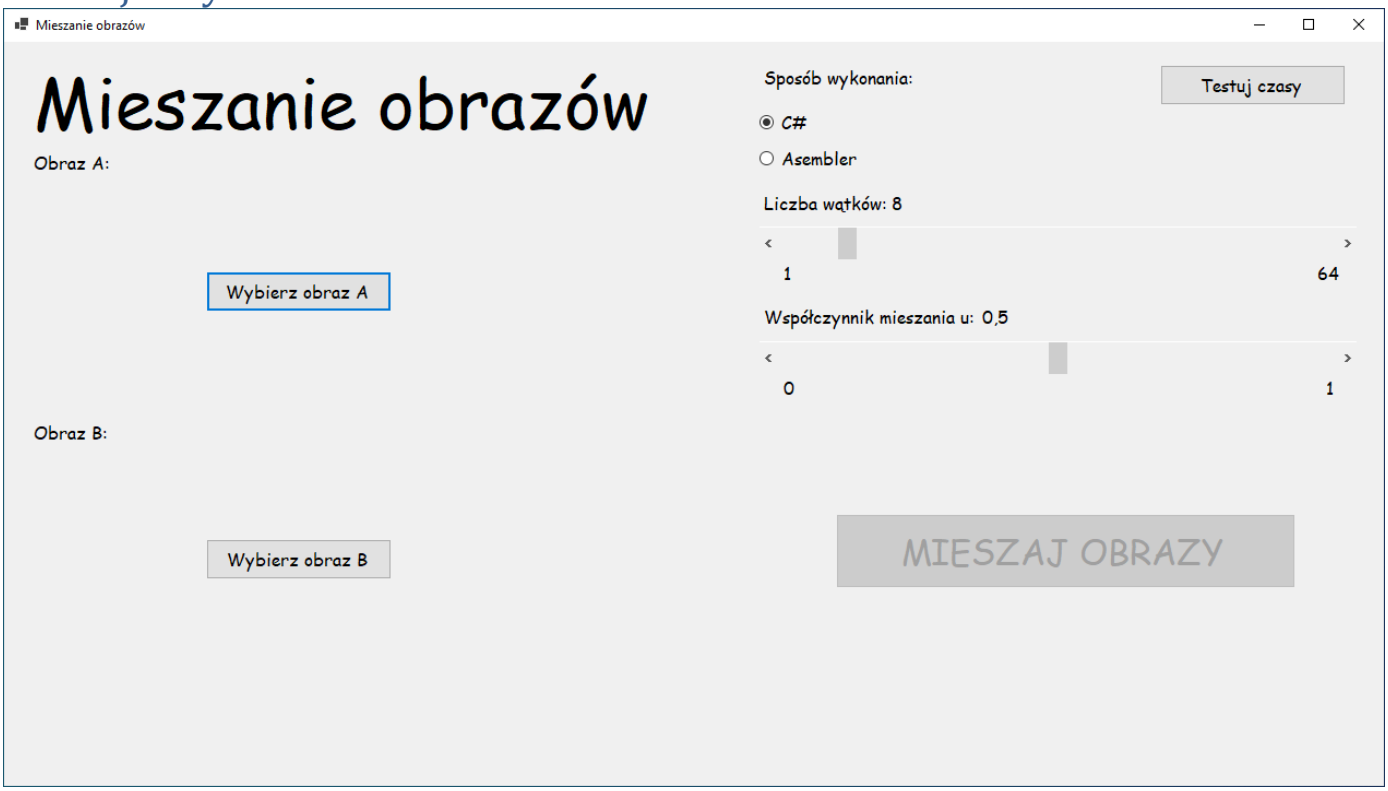
gdzie:
P – bajt wynikowy
A – bajt pierwszego obrazu
B – bajt drugiego obrazu
u – współczynnik mieszania
Do rozwiązania problemu można użyć analogicznego wzoru:

$$\boldsymbol{P = A + u\ (B - A)}.$$

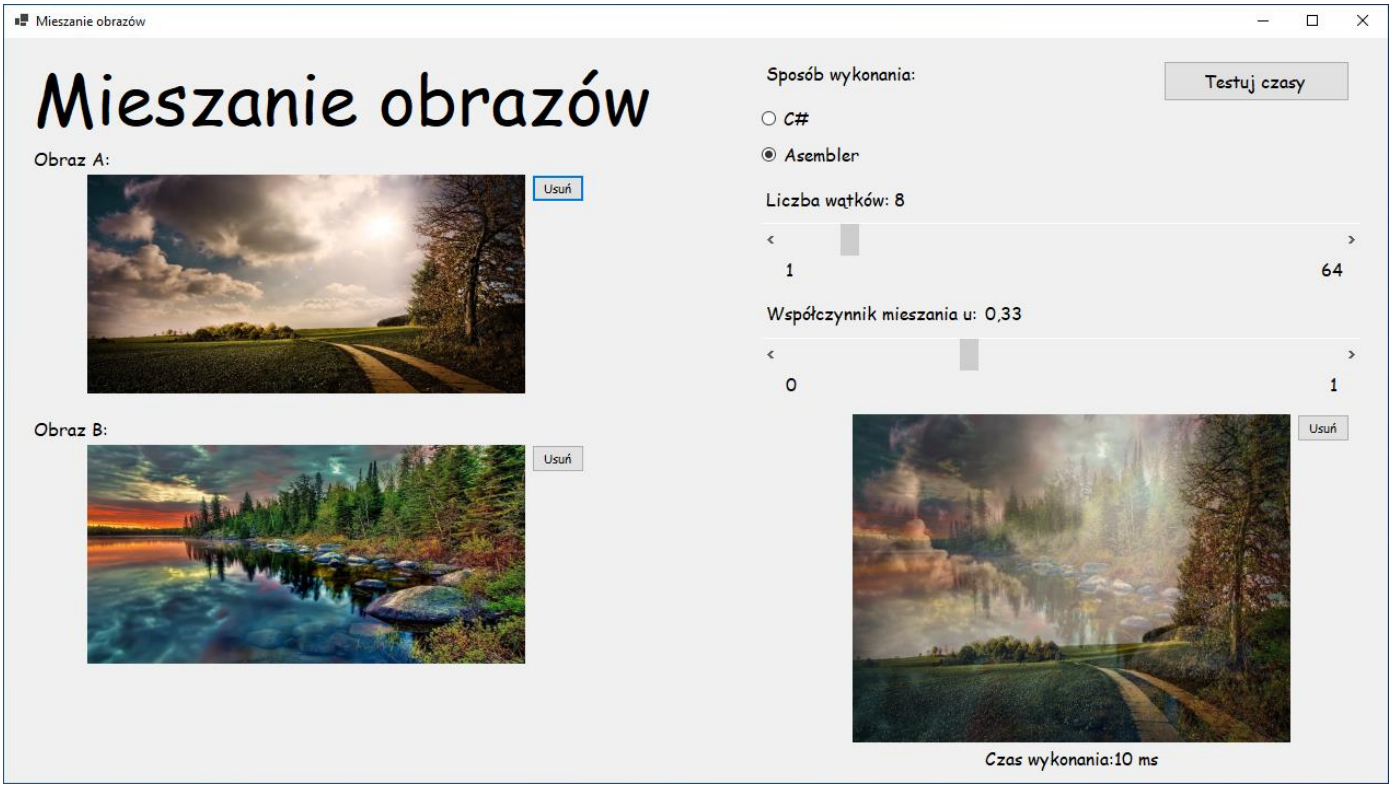
Rozwiązanie problemu

Wskazane przez użytkownika obrazy są konwertowane na tablicę zmiennych typu int o długości równej ilości pikseli. Użytkownik wybiera, czy chce skorzystać z algorytmu zaimplementowanego w języku wysokiego poziomu(C#) lub języku asemblera.
W implementacji algorytmu w języku asemblera celem zwiększenia wydajności wykorzystano rejestry XMM oraz operacje wektorowe.
Dodatkowo oba algorytmy mogą być wykonywane na wielu wątkach.
W programie współczynnik u został przeskalowany na zmienną typu int z zakresu <0,255>, co pozwala działać jedynie na liczbach całkowitych i upraszcza kod. Konsekwencją tej operacji jest konieczność przesunięcia bitowego wyniku o 8 bitów w prawo.

Interfejs użytkownika



Rys.1 "Interfejs aplikacji po uruchomieniu"



Rys.2 "Działanie aplikacji"

Użycie operacji wektorowych

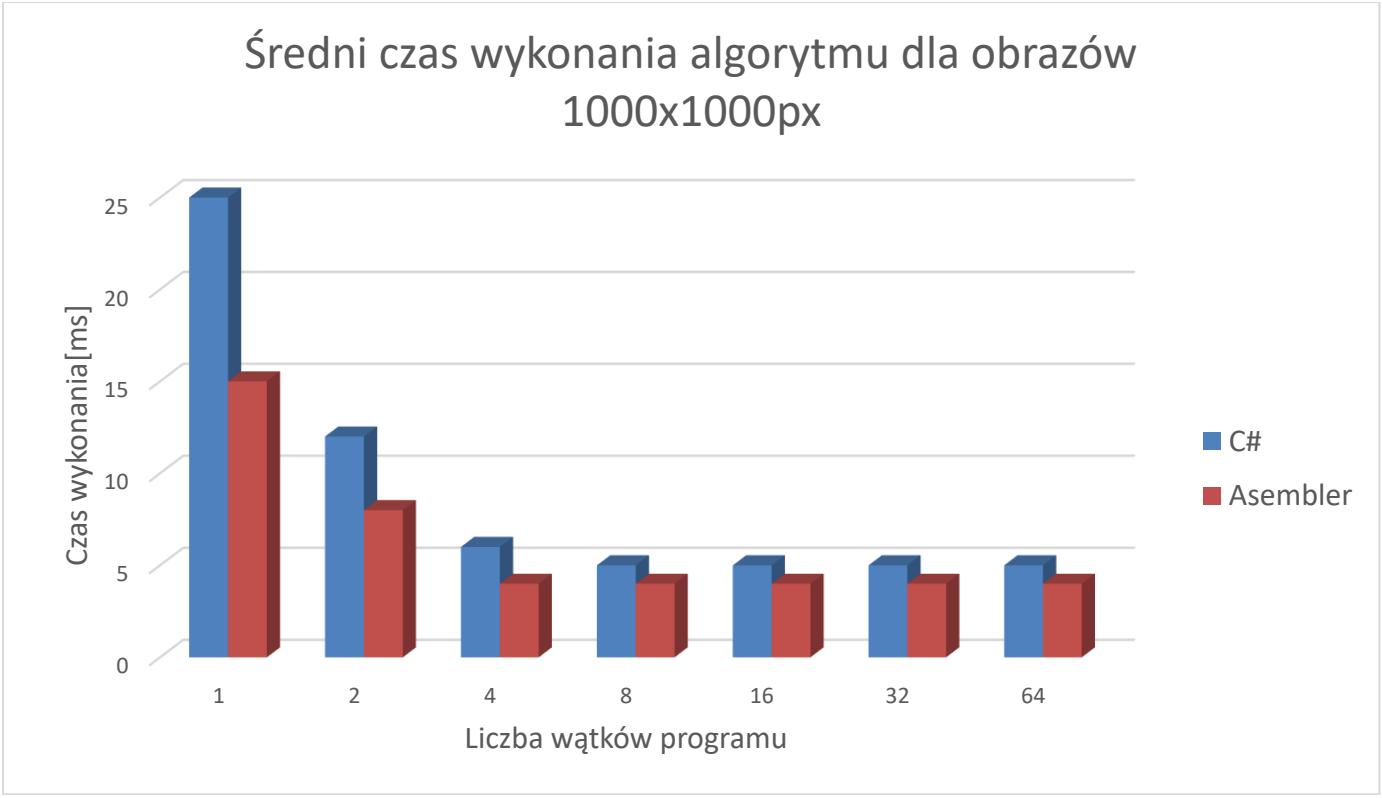
```
mov r10, rcx ;moving first procedure argument to r10(pixelA)
mov r11, rdx ;moving second procedure argument to r11(pixelB)
mov rbx, r8 ;moving third procedure argument to rbx(u)

mov rax, 256
sub rax, rbx
mov ah, bl  ;rax = |u|1-i|u|1-u|

movq xmm5, rax ;moving rax to xmm5

PMOVZXBW xmm5, xmm5
punpckldq xmm5, xmm5
punpckldq xmm5, xmm5

movq xmm0, r10  ;moving pixels to xmm registers ;xmm0 = |x|x|x|x|x|x|x|x|x|x|x|x|R1|G1|B1|
movq xmm1, r11  ;xmm1 = |x|x|x|x|x|x|x|x|x|x|x|x|R2|G2|B2|

PMOVZXBW xmm0, xmm0     ;xmm0 = |00 00|00 R1|00 G1|00 B1| ;Zero extend packed 8bit integers int the low 8bytes of xmm0 to packed 16bit
PMOVZXBW xmm1, xmm1     ;xmm1 = |00 B3|00 R2|00 G2|00 B2| ;Zero extend packed 8bit integers int the low 8bytes of xmm1 to packed 16bit

movups xmm2, xmm0
movups xmm3, xmm1

punpcklwd xmm0, xmm1    ;xmm0 = |G2 |00 G1|00 B2|00 B1| ;Interleave low-order words from xmm1 into xmm0
punpckhwd xmm2, xmm3    ;xmm2 = |00 00|00 00|00 R2|00 R1| ;Interleave low-order words from xmm3 into xmm2

pmaddwd xmm0, xmm5  ;xmm0 = |u*G2 + (1-u)*G1|u*B2 + (1-u)*B1| ;Multiplying and adding packed integers
pmaddwd xmm2, xmm5  ;xmm2 = |00|u*R2 + (1-u)*R1| ;Multiplying and adding packed integers

psrld     xmm0, 8 ;xmm0 = xmm0 >> 8
psrld     xmm2, 8 ;xmm2 = xmm0 >> 8

packssdw  xmm0, xmm2 ;Converts 2 packed signed doubleword integers
packuswb xmm0, xmm6 ;Converts 2 packed signed doubleword integers

movd rax, xmm0 ;Result left in eax

ret
mixAsm endp
end
```

Rys.3 "Operacje wektorowe"

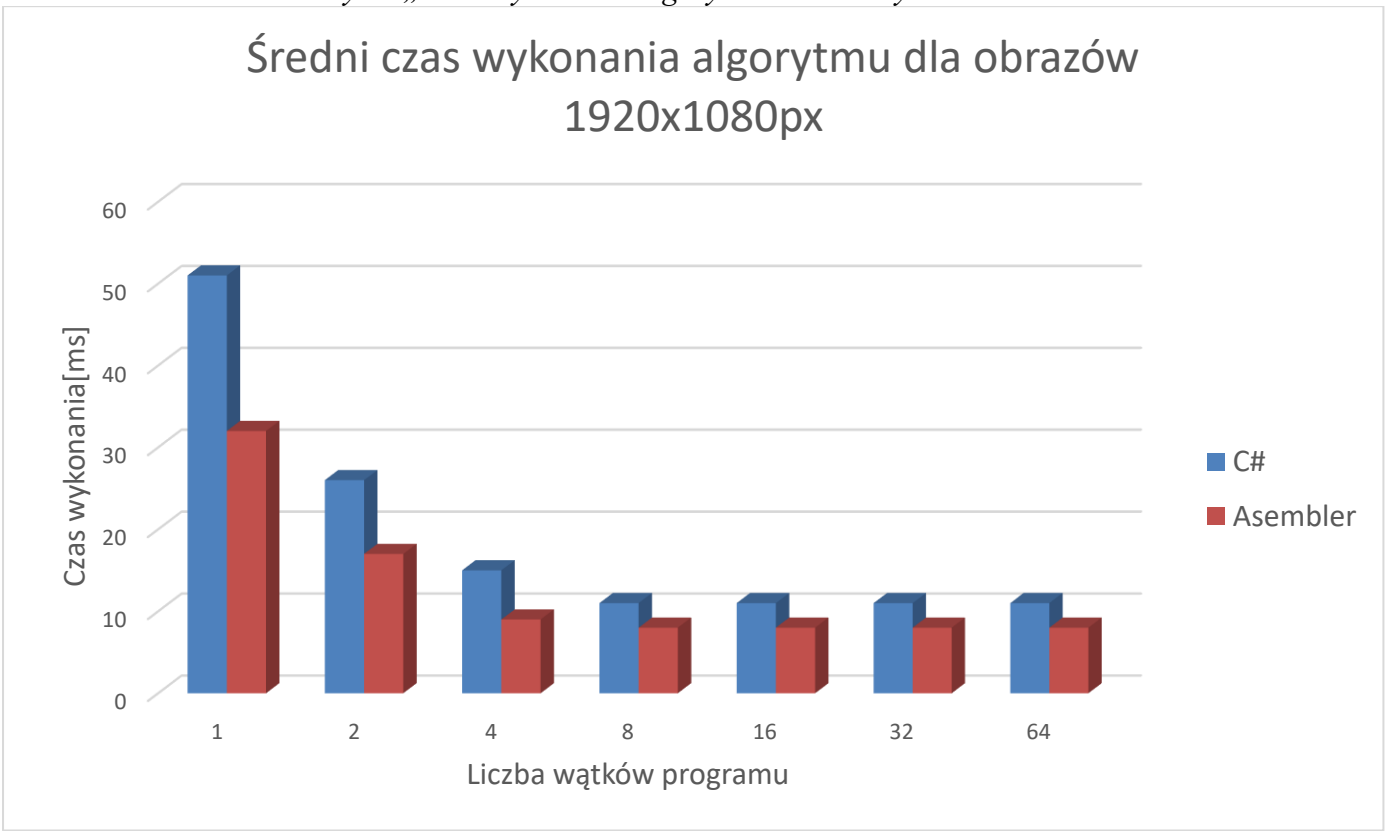
Wykorzystany rozkaz pmaddwd pozwala na wymnożenie bajtów obu obrazów przez odpowiednią wartość współczynnika u oraz następnie dodanie tych bajtów do siebie i zapis w jednym z rejestrów.

Porównanie czasów

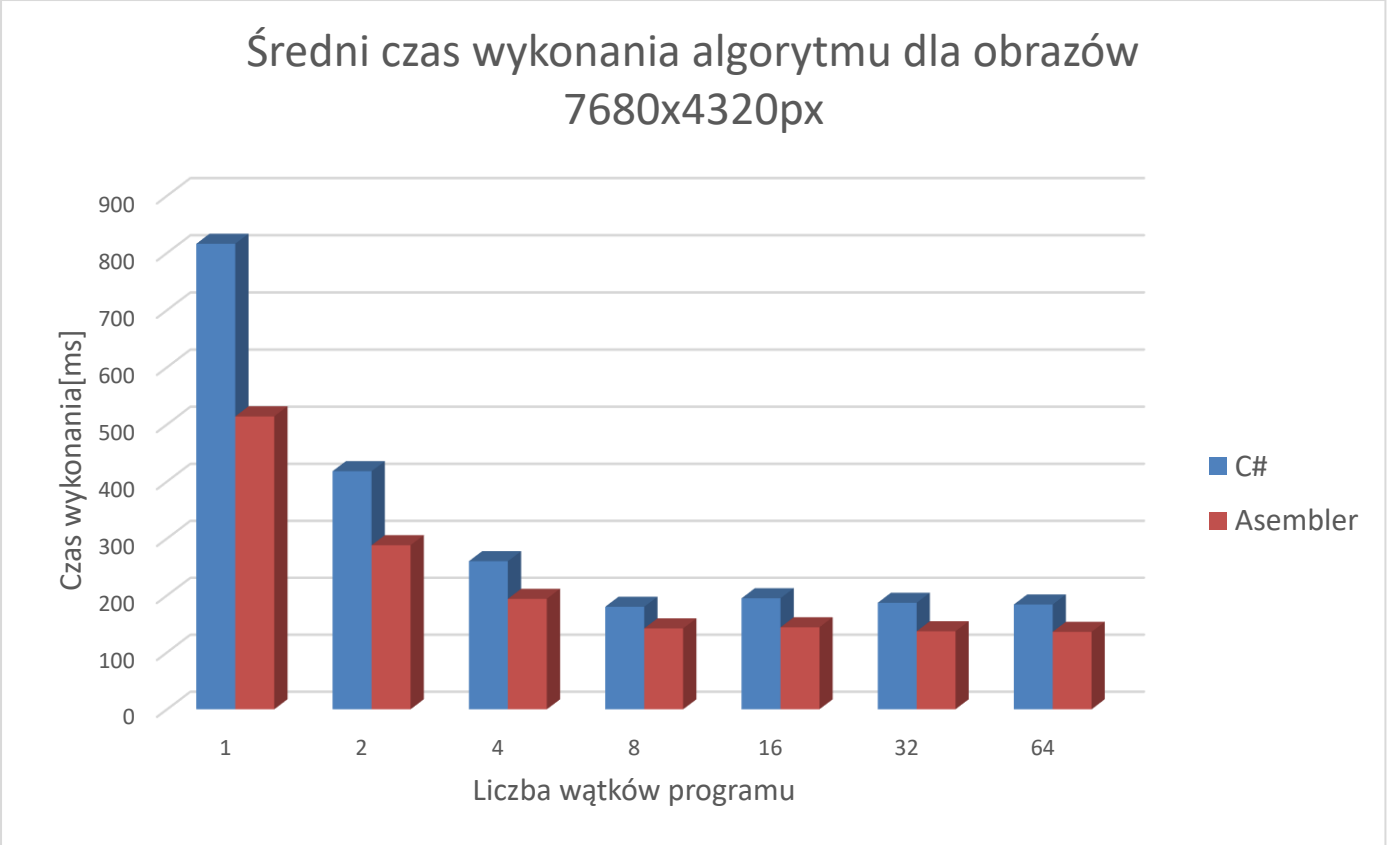
Każdy z czasów stanowi średnią z 100 pomiarów. Poniższe pomiary zostały wykonane dla obrazu podzielonego na 64 części.



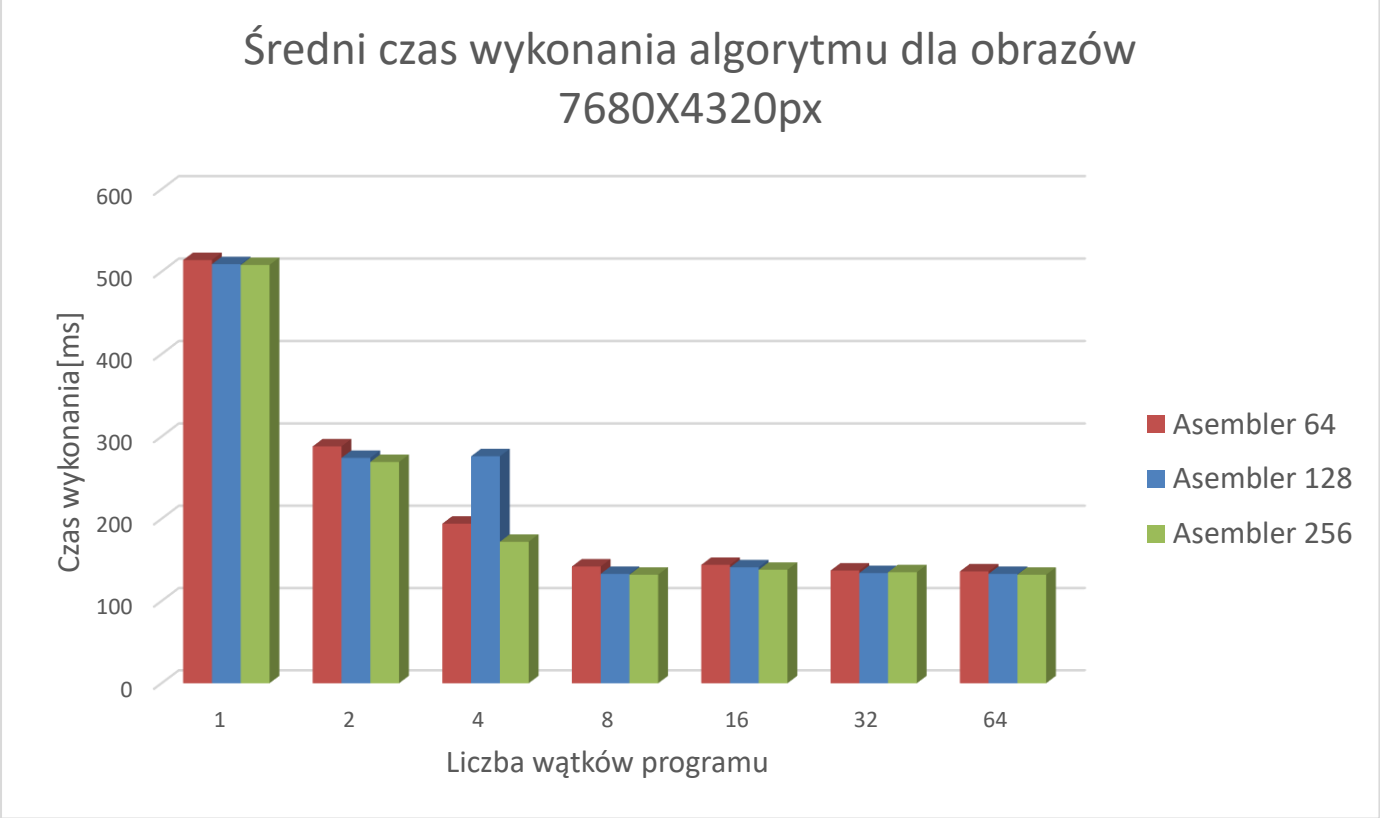
Rys.4 „Czas wykonania algorytmów dla małych obrazów”



Rys.5 „Czas wykonania algorytmów dla średnich obrazów”



Rys.5 „Czas wykonania algorytmów dla dużych obrazów”



Rys.6 „Porównanie czasu wykonania dla różnych podziałów obrazu - asembler”

Użyty procesor

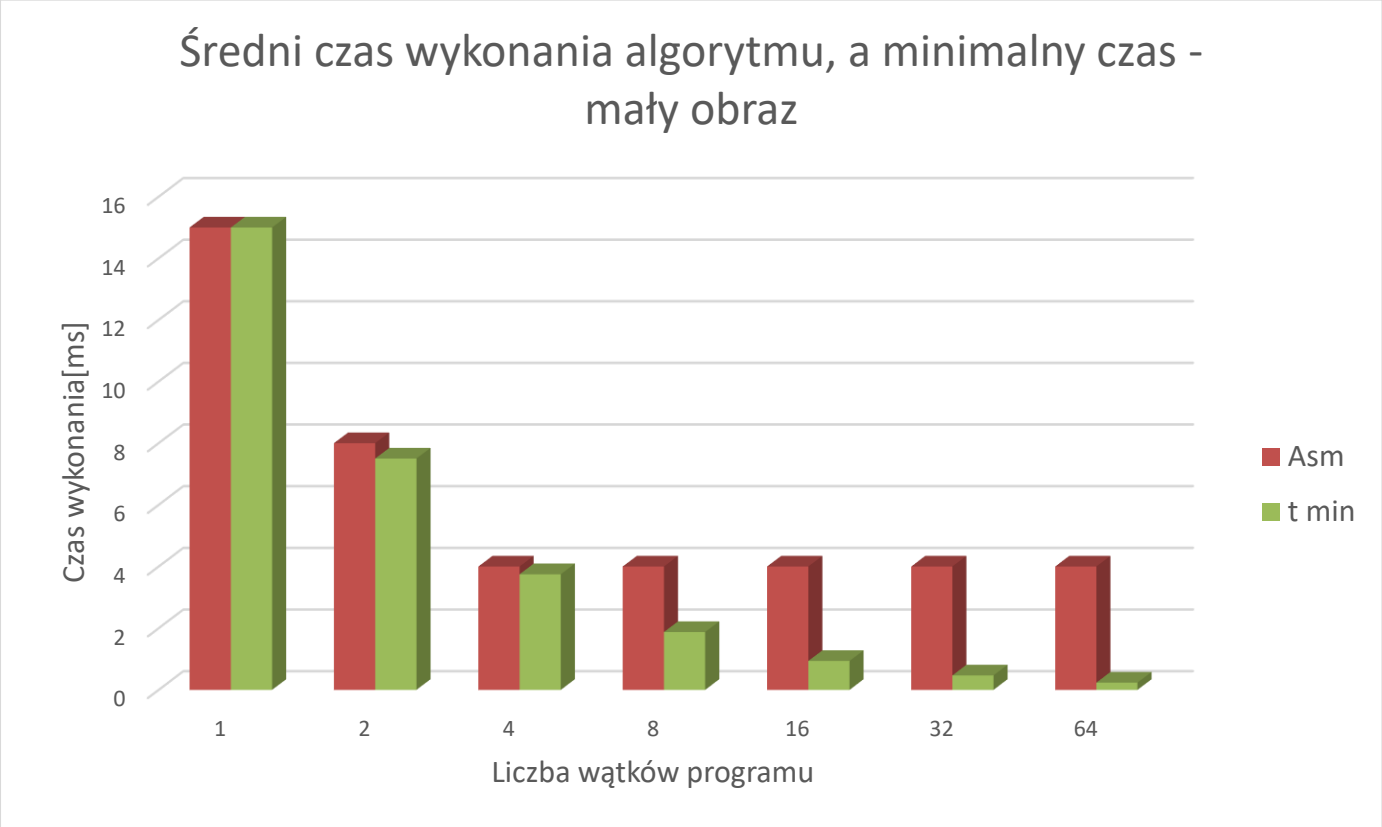
Model: i7-3770
Liczba rdzenie: 4
Liczba wątków: 8
Bazowa częstotliwość procesora: 3.4 GHz
Rozszerzony zestaw instrukcji: Intel® SSE4.1, Intel® SSE4.2, Intel® AVX

Testowanie i uruchamianie programu

Podczas testów okazało się, że program działa bardzo niestabilnie – przyczyną okazało się wychodzenie poza przydzieloną pamięć w procedurze napisanej w języku asemblera.
Po rozpoczęciu testowanie aplikacji w trybie Release pojawił się kolejny problem – algorytm w wersji niskopoziomowej wyliczał tylko 1/64 obrazu wynikowego. Co istotne w trybie Debug wyliczał całość. Rozwiązaniem okazało się wyłączenie optymalizacji kodu w trybie Release(w trybie Debug domyślnie wyłączona).
Pewien problem sprawiło również efektywne wykorzystanie wątków w programie. Dopiero użycie pętli *Parallel.For* z biblioteki *System.Threading.Tasks* pozwoliło na uzyskanie satysfakcjonujących wyników. Obraz jest dzielony na 64 części. Wartość tę można zmienić – jest ona polem klasy.
Program był testowany również dla podziału obrazu na 128 i 256 części, lecz otrzymane wyniki nie różniły się znacząco od poprzednich, z niewielką przewagą dla podziału na 256 części(co widać na rys.6).

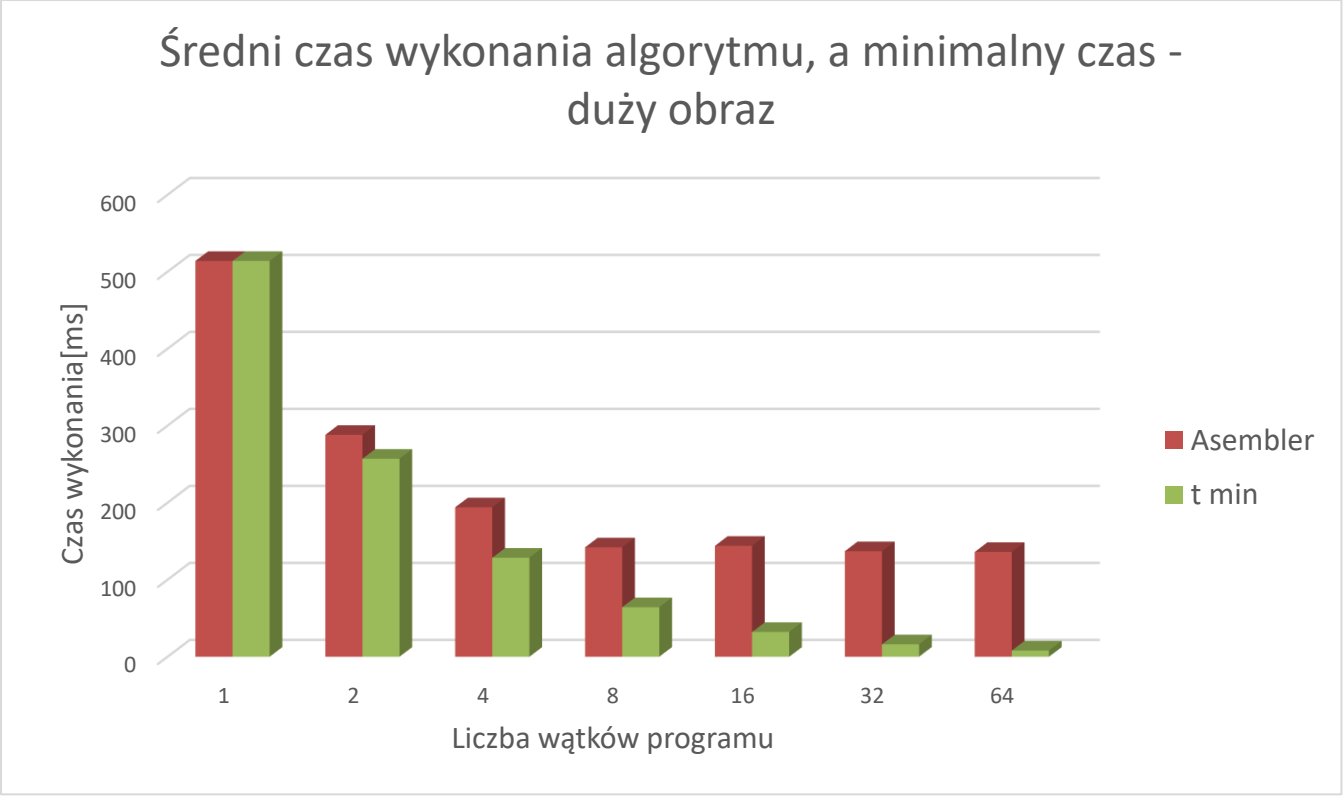
Wnioski

Algorytm napisany w języku C# okazał się być wolniejszy. Użycie wielu wątków w programie pozwoliło osiągnąć duże przyspieszanie, w niektórych przypadkach bliskie przyspieszeniu maksymalnemu:



Rys.7 „Porównanie czasów – obraz o rozdzielczości 1K”

Doskonale widać to na powyższym wykresie – uzyskujemy bardzo duże przyspieszenie do momentu zastosowania 4 wątków, większa liczba wątków nie wpływa już znacząco na przyspieszanie aplikacji. Wykres ten przedstawia wyniki dla małego obrazu 1K.



Rys.8 „Porównanie czasów – obraz o rozdzielczości 8K”

Dla obrazu 8K widzimy, że przyśpieszenie uzyskujemy do momentu zastosowania 8 wątków. Jest to wynik oczekiwany(użyty procesor i7-3770 posiada 8 wątków logicznych).