NIE	Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
Rok akademicki:	**Rodzaj studiów*: SSI/NSI/NSM**	**Przedmiot (Języki Asemblerowe/SMiW):**	**Grupa**	**Sekcja**
2018/2019	**SSI**	**Języki Asemblerowe**	**5**	**9**
Imię:	**Paweł**	**Prowadzący**: OA/JP/KT/GD/BSz/GB	**AO**	
Nazwisko:	**Wawszczyk**			

Raport końcowy

Temat projektu:

Usuwanie danego koloru z obrazu

Data oddania: dd/mm/rrrr	**02/12/2018**

1. Temat projektu:

   Tematem projektu jest stworzenie bibliotek w C++/C# oraz w języku asemblerowym, umożliwiających wykrycie podanego koloru oraz zastąpienie go odpowiadającym mu kolorem w skali szarości.

2. Opis założeń:
   a. Założenia części głównej

      Program oblicza czas wykonania algorytmu znajdującego się w bibliotekach. Algorytm może zostać wykonany przy użyciu od 0 do 64 wątków, w zależności od ustawień wybranych przez użytkownika. W przypadku nie wybrania wartości z powyższego przedziału, algorytm uruchamiany jest przy użyciu liczby wątków zgodnej z liczbą rdzeni logicznych procesora.

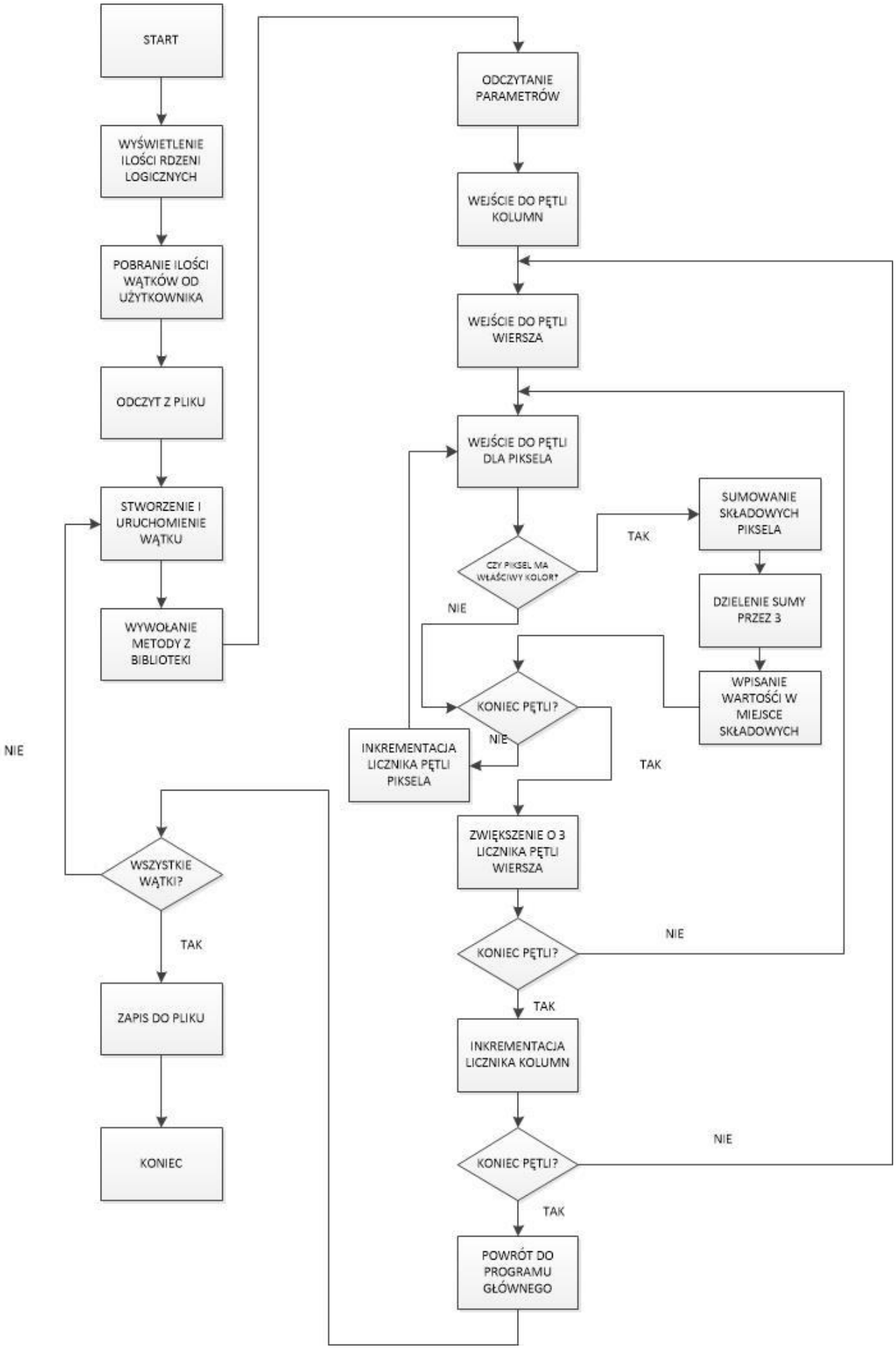
   b. Funkcje biblioteki

      Obie biblioteki wykonują identyczny algorytm, co umożliwia rzetelne porównanie czasów wykonania. Biblioteka asemblerowa wykorzystuje instrukcje wektorowe (SIMD) do wykonania algorytmu.

3. Analiza zadania:

   Temat projektu wymaga, aby wykrywać kolor każdego piksela i w przypadku wykrycia koloru, który ma zostać edytowany, zastąpić go kolorem w skali szarości. Po przenalizowaniu tematu, wybranym sposobem została analiza składowych każdego piksela zapisanego w modelu RGB i sprawdzanie, czy składowe te spełniają wymagania, by piksel mógł zostać zakwalifikowany do konkretnego koloru. Sposób ten wykrywania koloru piksela jest przejrzysty i nie wymaga dodatkowych konwersji, z racji faktu, że w języku C++ po wykonaniu odczytu z pliku i zapisania danych do tablicy, wartości kolejnych pikseli w tablicy są podane w modelu RGB.

4. Schemat blokowy:



5. Opis programu w języku wysokiego poziomu:

Część główna zajmuje się wykonaniem wszystkiego poza głównym algorytmem edycji obrazu. Znajduje się w niej klasa BMP zawierają wszystkie wymagane metody oraz pola, potrzebne do prawidłowego działania programu oraz metoda main. W klasie BMP znajdują się:

- readBMP - metoda zajmująca się odczytem z pliku BMP, posiada jeden parametr, który jest nazwą pliku, jaki zostać ma odczytany. Do tablicy info wpisany zostaje cały nagłówek pliku, natomiast do tablicy data wpisane zostają wszystkie piksele (w modelu RGB, w kolejności składowych niebieski, zielony, czerwony). Metoda dodatkowo rozwiązuje problem paddingu oraz wpisuje do tablicy size odpowiednie parametry obrazu. Metoda nic nie zwraca.
- saveBMP – metoda zajmująca się zapisem do pliku, posiada jeden parametr, który jest nazwą pliku, do którego ma zostać zapisany obraz. Nagłówek nowego pliku jest pobrany ze zmiennej, do której trafił podczas odczytu pliku, a następnie wpisane do pliku są dane pikseli.
- grayScaleAsm oraz grayScaleCpp - dwie metody zajmujące się podziałem na wątki oraz wywołaniem metody z biblioteki, przekazując jej wszystkie parametry, umożlwiające poprawne działanie dla konkretnego wątku. Otrzymują dwa parametry: ilość wątków, jaka ma zostać stworzona oraz kolor, jaki ma zostać edytowany.

W main znajduje się wyświetlenie ilości rdzeni logicznych oraz wywołanie metod z klasy BMP.

6. Opis funkcji bibliotek:

Biblioteki w C++ oraz asemblerze wykonują główny algorytm programu. W obu bibliotekach znajduje się jedna metoda o nazwie ChangeImage. Przyjmuje ona trzy parametry: tablicę size, dane wejściowe odczytane z pliku oraz kolor, jaki ma zostać edytowany. Metoda zawiera trzy pętle: pętlę poruszającą się po kolumnach, zagnieżdżoną pętlę poruszającą się po wierszach oraz zagnieżdżoną pętlę, która dla danego piksela sprawdza, czy piksel jest danego koloru, jaki został przekazany w parametrze wejściowym (czerwony, zielony lub niebieski). W bibliotece asemblerowej używane rejestry to:

- r15 – przechowujący tablicę size
- r14 – dane wejściowe
- r12 – kolor, jaki ma zostać edytowany
- r11 oraz xmm6 – licznik pętli kolumn
- r10 oraz xmm8 – licznik pętli wiersza
- r19 – rejestr używany do zapisywania edytowanych składowych piksela w określone miejsce w tablicy
- r8 – licznik pętli dla danego piksela
- rcx oraz xmm15 – rejestr, w którym przechowana jest szerokość obrazu
- rdx – rejestr, w którym przechowywana jest wysokość obrazu
- xmm10 – rejestr używany przejściowo podczas obliczania, gdzie ma być wpisana edytowana składowa piksela
- xmm5 oraz xmm3 – rejestry przejściowo używane podczas wykonywania skali szarości na pikselu
- xmm2, xmm1 oraz xmm0 – rejestry używane do rozpakowania danych do rejestru xmm2

Metoda sprawdza, czy wartości składowych piksela są odpowiednie, aby móc uznać go za piksel o kolorze takim, jaki ma być poddany edycji. Jeśli tak, to wszystkie składowe piksela są sumowe, a następnie suma ta jest dzielone przez 3, a ostateczny wynik wpisany w miejsce składowych piksela. Dzięki temu otrzymana zostaje skala szarości w miejsce zadanego koloru. Jeśli piksel nie zostaje rozpoznany jako piksel o kolorze do edycji, następuje skok o 3 miejsca w tablicy, do następnego. Po przejściu do końca tablicy, następuje zakończenie działania metody i powrót do programu głównego.

7. Dane wejściowe i testowe:

Dane wejściowe w postaci obrazu muszą być w formacie 24-bitowej bitmapy, również dane wyjściowe zostają stworzone w takim samym formacie. Do testowania używana jest również dowolny obraz, o ile jego format to 24-bitowa bitmapa.

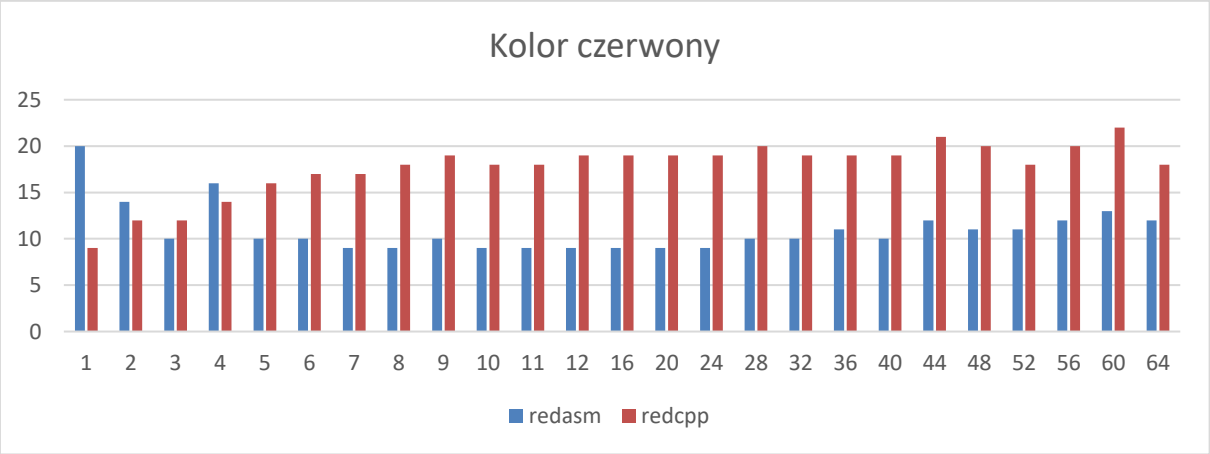
8. Opis uruchamiania programu i testowania:

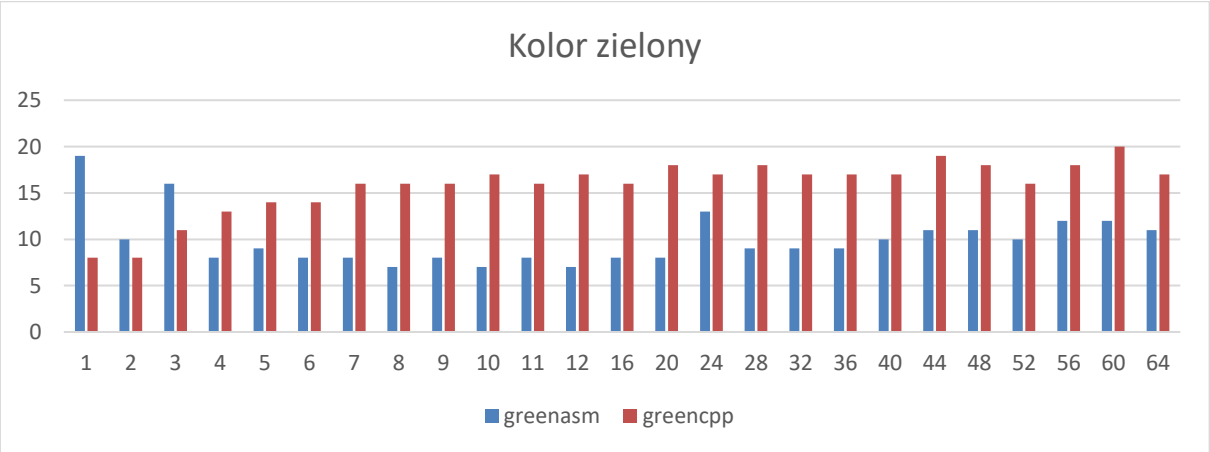
Program uruchamiany jest w konsoli. Następnie algorytm zostaje uruchomiony na podanej ilości wątków – każdy wątek obsługuje określoną ilość wierszy obrazu. Następnie wyświetlony jest w milisekundach czas wykonania algorytmu przy użyciu bibliotek w C++ oraz w asemblerze. Następnie po kliknięciu dowolnego przycisku program zostaje zamknięty.

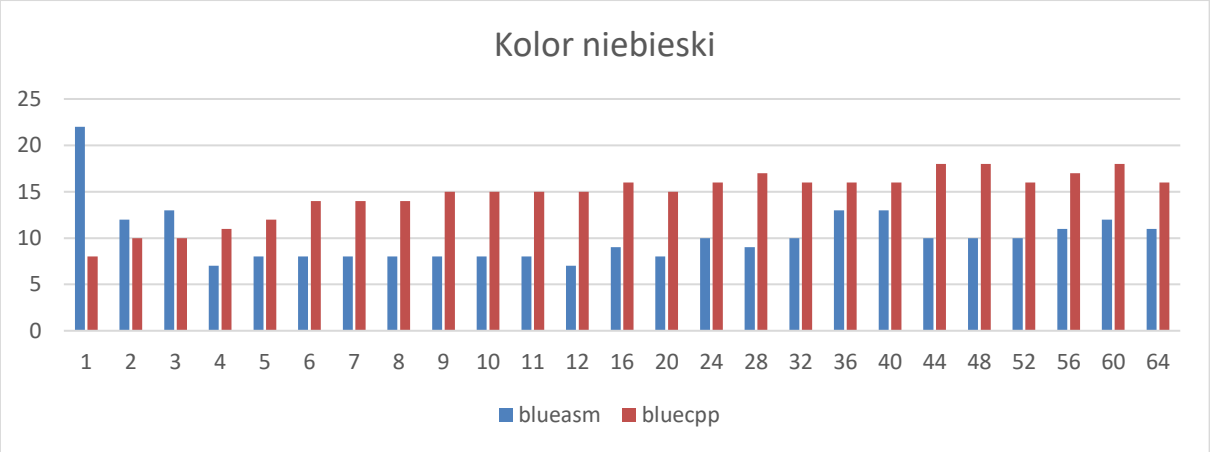
Aby wykonać testowanie, należy w metodzie main zakomentować fragment kodu nazwany „Uruchomienie” oraz odkomentować fragment kodu nazwany „Testowanie”. Wtedy po uruchomieniu programu, algorytm jest wykonywany przy użyciu od 1 do 64 wątków dla metod z obu bibliotek. Następnie aby zamknąć program, wystarczy wcisnąć dowolny przycisk.

9. Wyniki pomiarów czasu wykonania programu:

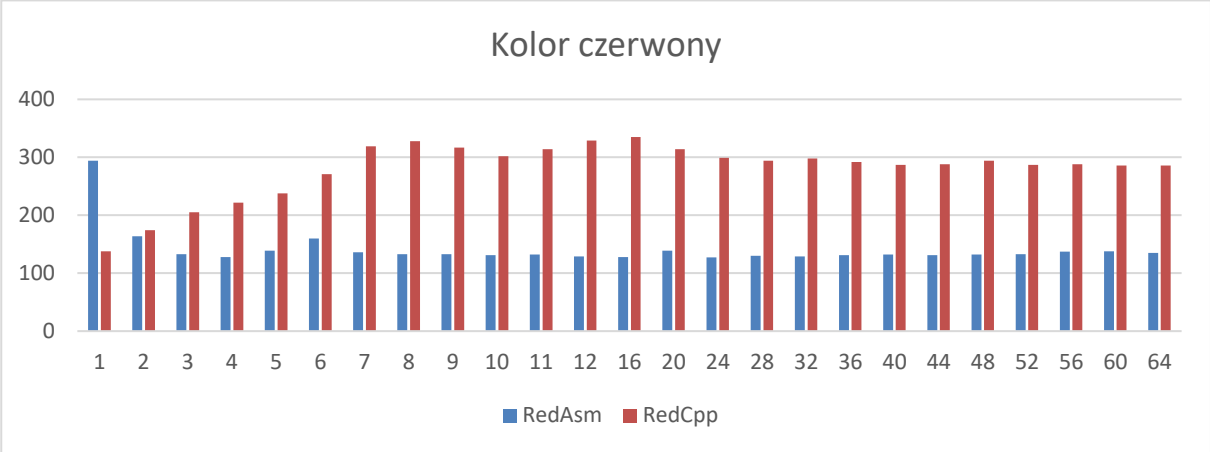
- Wykresy czasowe dla obrazu o małej rozdzielczości:

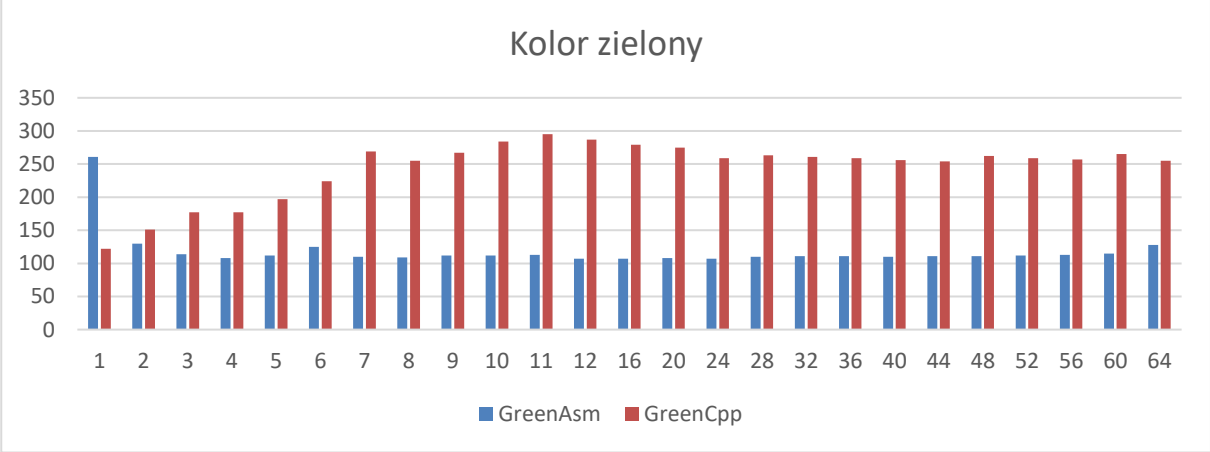


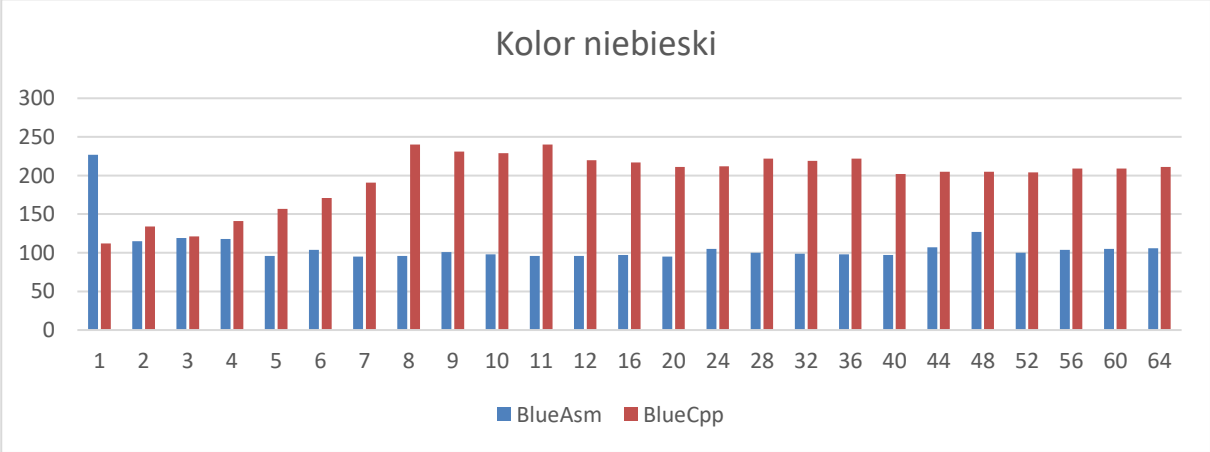




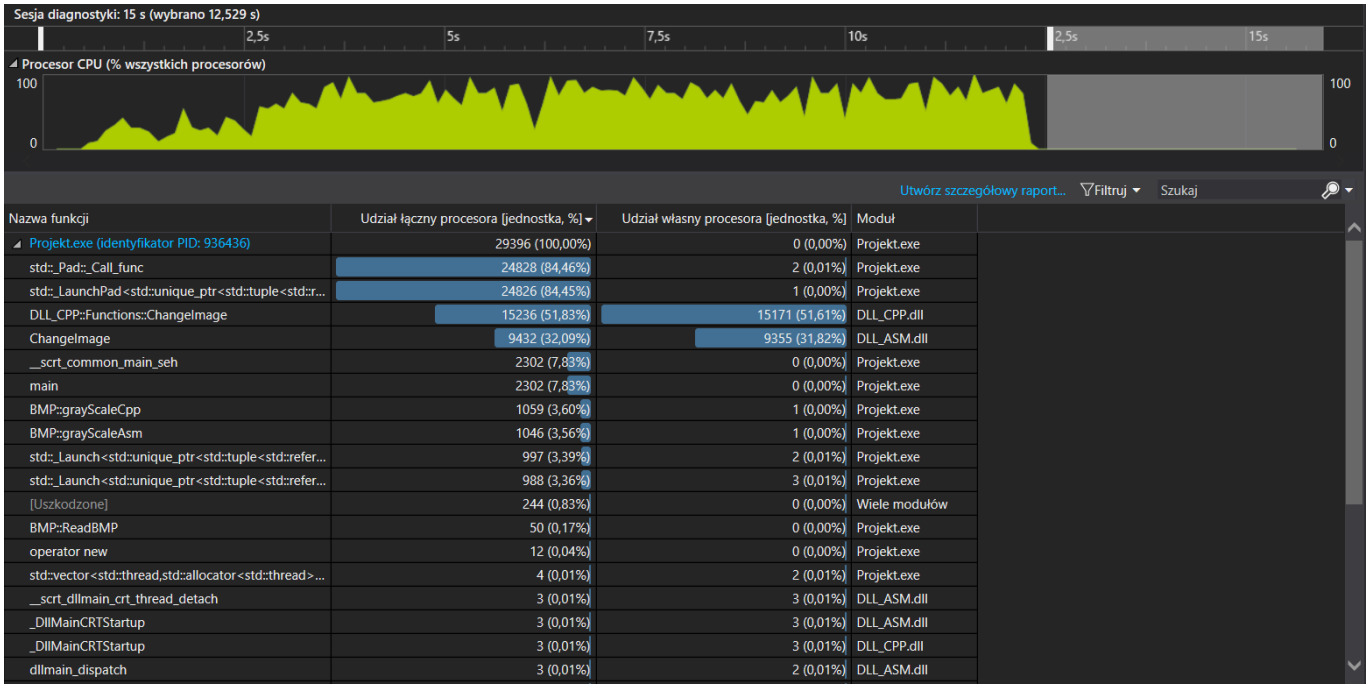
- Wykresy czasowe dla obrazu o dużej rozdzielczości:







10. Analiza działania z wykorzystanie profilera:



Nazwa funkcji	Udział łączny procesora [jednostka, %]	Udział własny procesora [jednostka, %]	Moduł
Projekt.exe (identyfikator PID: 936436)	29396 (100,00%)	0 (0,00%)	Projekt.exe
std::_Pad::_Call_func	24828 (84,46%)	2 (0,01%)	Projekt.exe
std::_LaunchPad<std::unique_ptr<std::tuple<std::r...	24826 (84,45%)	1 (0,00%)	Projekt.exe
DLL_CPP::Functions::ChangeImage	15236 (51,83%)	15171 (51,61%)	DLL_CPP.dll
ChangeImage	9432 (32,09%)	9355 (31,82%)	DLL_ASM.dll
__scrt_common_main_seh	2302 (7,83%)	0 (0,00%)	Projekt.exe
main	2302 (7,83%)	0 (0,00%)	Projekt.exe
BMP::grayScaleCpp	1059 (3,60%)	1 (0,00%)	Projekt.exe
BMP::grayScaleAsm	1046 (3,56%)	1 (0,00%)	Projekt.exe
std::_Launch<std::unique_ptr<std::tuple<std::refer...	997 (3,39%)	2 (0,01%)	Projekt.exe
std::_Launch<std::unique_ptr<std::tuple<std::refer...	988 (3,36%)	3 (0,01%)	Projekt.exe
[Uszkodzone]	244 (0,83%)	0 (0,00%)	Wiele modułów
BMP::ReadBMP	50 (0,17%)	0 (0,00%)	Projekt.exe
operator new	12 (0,04%)	0 (0,00%)	Projekt.exe
std::vector<std::thread,std::allocator<std::thread>...	4 (0,01%)	2 (0,01%)	Projekt.exe
__scrt_dllmain_crt_thread_detach	3 (0,01%)	3 (0,01%)	DLL_ASM.dll
_DllMainCRTStartup	3 (0,01%)	3 (0,01%)	DLL_ASM.dll
_DllMainCRTStartup	3 (0,01%)	3 (0,01%)	DLL_CPP.dll
dllmain_dispatch	3 (0,01%)	2 (0,01%)	DLL_ASM.dll

11. Instrukcja obsługi programu:

Po włączeniu programu w trybie „Uruchomienie”, wyświetlana zostaje ilość rdzeni logicznych. Następnie należy podać ilość wątków, przy użyciu jakiej algorytm ma się wykonać. Należy podać wartość z przedziału od 0 do 64, w innym przypadku program zostanie uruchomiony przy użyciu liczby wątków zgodnej z liczbą rdzeni logicznych procesora.

Po włączeniu programu w trybie „Testowanie”, brak jakiejkolwiek interakcji
W kodzie w metodzie main w dwóch miejscach można dokonywać korekt, aby zmodyfikować działanie programu.

Aby zmienić kolor, jaki ma zostać edytowany, należy zmienić drugi parametr wywoływanych w main metod grayScaleAsm oraz grayScaleCpp. „1” pozwala na usunięcie koloru czerwonego, „2” zielonego, a „3” czerwonego.

Aby zmienić obraz, jaki zostanie zmodyfikowany, należy w każdym wywołaniu metody ReadBMP zmienić pierwszy parametr na nazwę pliku (włącznie z rozszerzeniem, wszystko w cudzysłowie) jaki ma zostać edytowany. Aby zmienić nazwę pliku, do jakiego wyjściowy obraz ma zostać zapisany, należy zmodyfikować pierwszy parametr w każdym wywołaniu metody SaveBMP.

12. Wnioski:

Napisanie tego projektu pozwoliło wyciągnąć kilka wniosków:

- Czas wykonywania metody w asemblerze nie zawsze jest krótszy niż jego odpowiednika w C++.
- Wzrost użytej ilości wątków nie jest równoważny z krótszym czasem wykonania algorytmu.
- Wielkość danych testowych jest kluczowa – w zależności od wielkości danych czas wykonywania asemblera wraz ze wzrostem ilości wątków może szybciej lub wolniej rosnąć.

13. Literatura:

Tworzenie skali szarości: www.algorytmy.org