
	Instytut Informatyki Politechniki Śląskiej			
	Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/SMiW):	Grupa	Sekcja
2019/2020	SSI	JA	3	6
Imię:	Paulina	Prowadzący:	AO	
Nazwisko:	Urbaś			
<h2 style="text-align: center;">Raport końcowy</h2>				
Temat projektu: <div style="text-align: center; margin-top: 100px;"> <h1>Negatyw bitmapy</h1> </div>				
Data oddania sprawozdania: dd/mm/rrrr		7/12/2019		

Temat projektu opis założeń

Tematem projektu było napisanie programu, który tworzy negatyw podanej przez użytkownika bitmapy.

Założenia części głównej

Główna część aplikacji napisana jest w języku C++. Użytkownik posiada możliwość wyboru, dzięki przełącznikom, za pomocą której biblioteki będzie przetwarzany obraz. Użytkownik posiada możliwość wyboru ilości wątków. Program jest włączany z konsoli. Po zakończeniu działania programu użytkownik otrzymuje przetworzoną bitmapę (zapisaną w pliku, którego nazwę podaje po przełączniku -o). Dodatkowo na konsoli pojawia się czas przetwarzania danego algorytmu na danej ilości wątków.

Założenia funkcji biblioteki

W obu bibliotekach DLL, jednej napisanej w assemblerze, drugiej napisanej w C, znajduje się funkcja, która wykonuje konwersję bitmapy na negatyw na wybranym fragmencie.

Analiza zadania

Działanie funkcji negatyw polega na wykonaniu odejmowania od maksymalnej dopuszczalnej wartości (255), wartości obecnie przetwarzanego piksela.

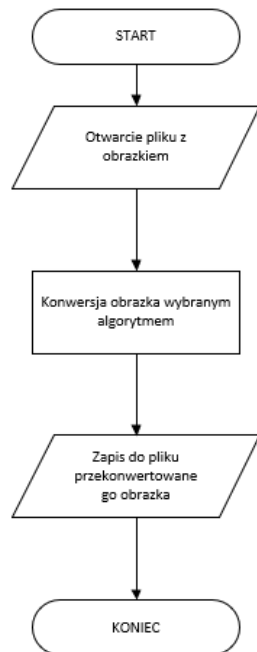
$$R_{\text{nowe}} = 255 - R_{\text{stare}}$$

$$B_{\text{nowe}} = 255 - B_{\text{stare}}$$

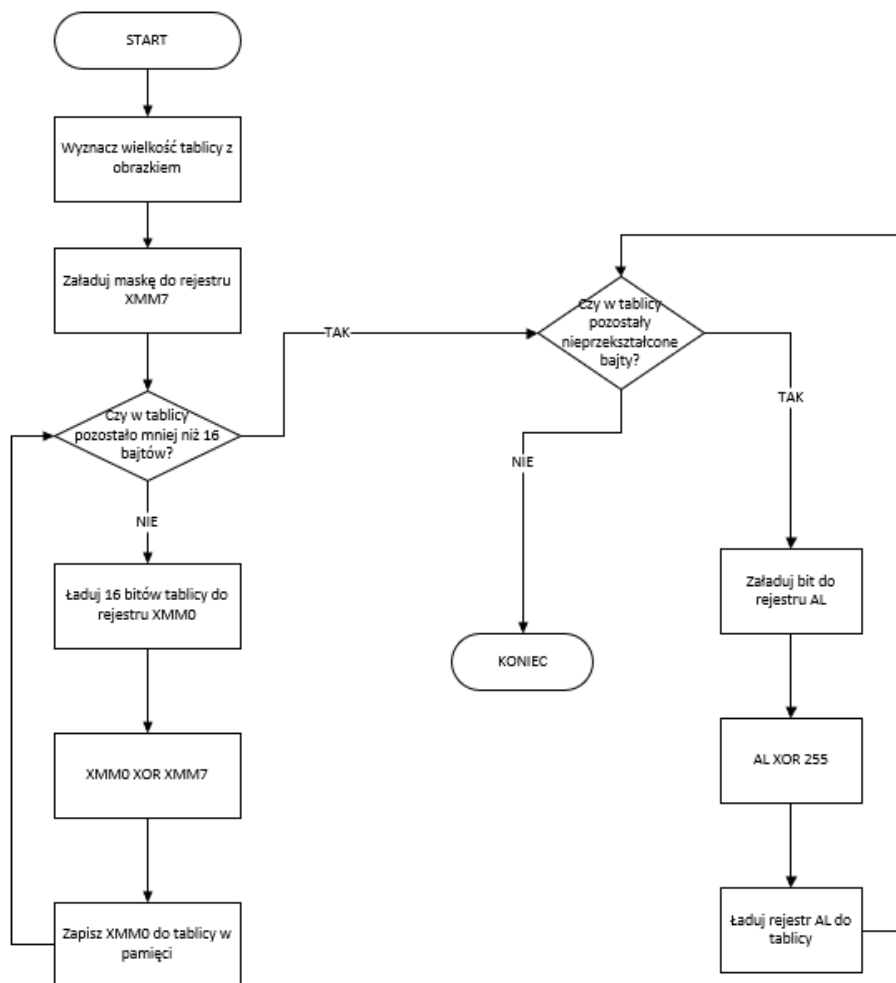
$$G_{\text{nowe}} = 255 - G_{\text{stare}}$$

W assemblerze bajty definiujące poszczególne składowe kolorów są liczbami całkowitymi bez znaku. Aby uzyskać negatyw, czyli odwrotność tych wartości, wykonywana jest operacja XOR z wartością 255.

Schemat blokowy programu



Rysunek 1 Schemat blokowy działania programu głównego



Rysunek 2 Schemat blokowy funkcji bibliotecznej w assemblerze

Opis programu głównego

Główna część aplikacji została napisana w języku C++. Jest to aplikacja konsolowa. Aby ją uruchomić użytkownik musi podać napisane odpowiednio polecenie z odpowiednimi parametrami. W części głównej programu następuje podział programu na wątki oraz wywoływane są funkcje z bibliotek DLL.

```
void Help();
```

Funkcja wyświetlająca pomoc.

```
int TerminalCheck(int argc, char * argv[]);
```

Funkcja sprawdzająca parametry wywołania programu.

Jeśli parametry wywołania są nie poprawne zwraca wartość 1, jeśli są poprawne zwraca 0 lub 2:

- 0 w przypadku wywołania bez ilości podanych wątków
- 2 w przypadku podania ilości wątków.

```
bool CheckIfNumber(int number);
```

Funkcja sprawdzająca, czy podany numer jest większy od 0 i mniejszy od 65.

Przyjmuje jako parametr ilość wątków podanych przez użytkownika.

Gdy numer jest mniejszy od 0 lub większy od 64 funkcja zwraca wartość false, w przeciwnym razie wartość true

```
bool checkIfExist(const std::string & inputName);
```

Funkcja sprawdzająca, czy plik o podanej nazwie istnieje.

Przyjmuje jako parametr nazwę do pliku.

Zwraca true, jeśli plik może zostać otwarty, w przeciwnym wypadku zwraca wartość false.

```
class Image
```

Klasa przechowuje informacje o obrazku. Posiada metody umożliwiające odczyt i zapis bitmapy.

```
private: int height = 0;  
        int width = 0;  
        int offset = 0;
```

Zmienne przechowujące właściwości obrazka.

```
char * Data;  
BITMAPFILEHEADER* FileInfo;  
BITMAPINFOHEADER* PictureInfo;
```

Zmienne przechowujące tablicę pikseli, header oraz infoheader bitmapy.

```
char * makeBitmap(const char* inputName);
```

Funkcja wczytująca obraz z pliku do programu. Jako parametr przyjmuje nazwę pliku. Zwraca wskaźnik do tablicy z pikselami.

```
void saveBitmap(std::string inputName);
```

Funkcja zapisująca obrazek do pliku. Jako parametr przyjmuję nazwę pliku.

```
std::vector<std::thread*>threads;
```

Wektor wątków.

Opis funkcji biblioteki DLL C

W bibliotece znajduje się jedna funkcja:

```
void Negative(char* bmp, int width, int height)
```

Konwertuje podany fragment bitmapy na negatyw. Przechodzi kolejno po wszystkich pikselach z zakresu ustawionego przez argumenty. Dla 3 kolejnych bitów wykonuje operację odjęcia od wartości 255, wartości obecnie przetwarzanego piksela:

```
bmp[i] = 255 - bmp[i];  
bmp[i + 1] = 255 - bmp[i + 1];  
bmp[i + 2] = 255 - bmp[i + 2];
```

Argumenty:

- **bmp** – wskaźnik do tablicy z obrazkiem
- **width** – szerokość
- **height** – wysokość

Opis funkcji biblioteki DLL ASM, parametry

W bibliotece znajduje się jedna funkcja do przetwarzania bitmapy. Funkcja zaczyna działanie od obliczenia wielkości bitmapy według wzoru:

$$\text{wielkość tablicy} = \text{szerokość} * \text{wysokość} * 3 \text{ bajty RGB}$$

Zorganizowana jest w dwie pętle: dużą, która przekształca 16 bajtów oraz małą, która przekształca pozostałe bajty.

Funkcja wykorzystuje następujące rejestry jako argumenty funkcji:

- **RCX** – pierwszy argument funkcji, przechowuje adres pierwszego bitu tablicy bitmapy
- **RDX** – drugi argument funkcji, przechowuje szerokość bitmapy
- **R8** – trzeci argument funkcji, przechowuje wysokość bitmapy

Funkcja wykorzystuje następujące rejestry do funkcji przetwarzania bitmapy:

- **RAX** – przechowuje szerokość bitmapy, następnie przechowuje licznik przejść dużej pętli
- **RBX** – przechowuje wysokość bitmapy
- **RCX** – przechowuje licznik przejść małej pętli
- **RDX** – przechowuje wielkość obrazka w bajtach
- **RSI** – przechowuje wskaźnik do tablicy z bitmapą
- **XMM0** – przechowuje 16 bajtów z tablicy pikseli
- **XMM7** – przechowuje wartość 255

Opis struktury danych wejściowych/testowych programu

Daną wejściową jest plik bitmapy 24-bitowej. Wymagane rozszerzenie to .bmp

Opis uruchamiania programu i jego testowania

Program uruchamiany jest z konsoli lub wiersza poleceń. Po zakończeniu konwersji obrazu na negatyw pojawia się informacja o czasie przetwarzania. Program przetestowano pod kątem próby otwarcia nieistniejącego pliku, źle podanych parametrów, zbyt dużej ilości wątków, zbyt małej ilości wątków. Testowanie algorytmów napisanych w bibliotekach DLL polegało na konwersji różnych bitmap na różnej ilości wątków.

Wszystkie testy wykonano na komputerze z procesorem Intel Core i5-7200U.

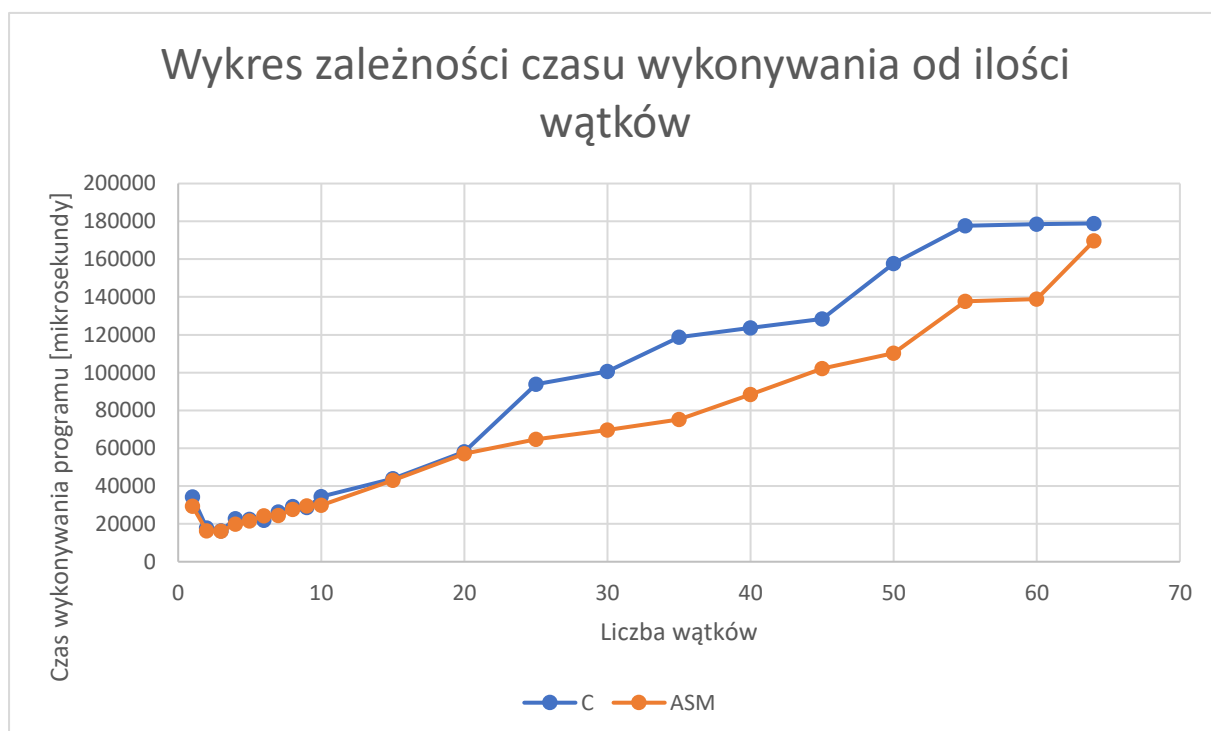
Wyniki pomiarów czasu wykonania programu dla różnych danych testowych i obu wersji bibliotek

Wyniki pomiarów czasu dla trzech bitmap o następujących wielkościach:

a. 880x520

Liczba wątków	C	ASM
1	34325	29287
2	17774	16323
3	16377	16139
4	22725	19904
5	22502	21585
6	21946	24321
7	26348	24460
8	29192	27728
9	28736	29612
10	34414	29858
15	43850	43067
20	58158	57102
25	93805	64741
30	100663	69639
35	118698	75179
40	123619	88432
45	128331	102217
50	157633	110256
55	177647	137726
60	178560	138933
64	178901	169696

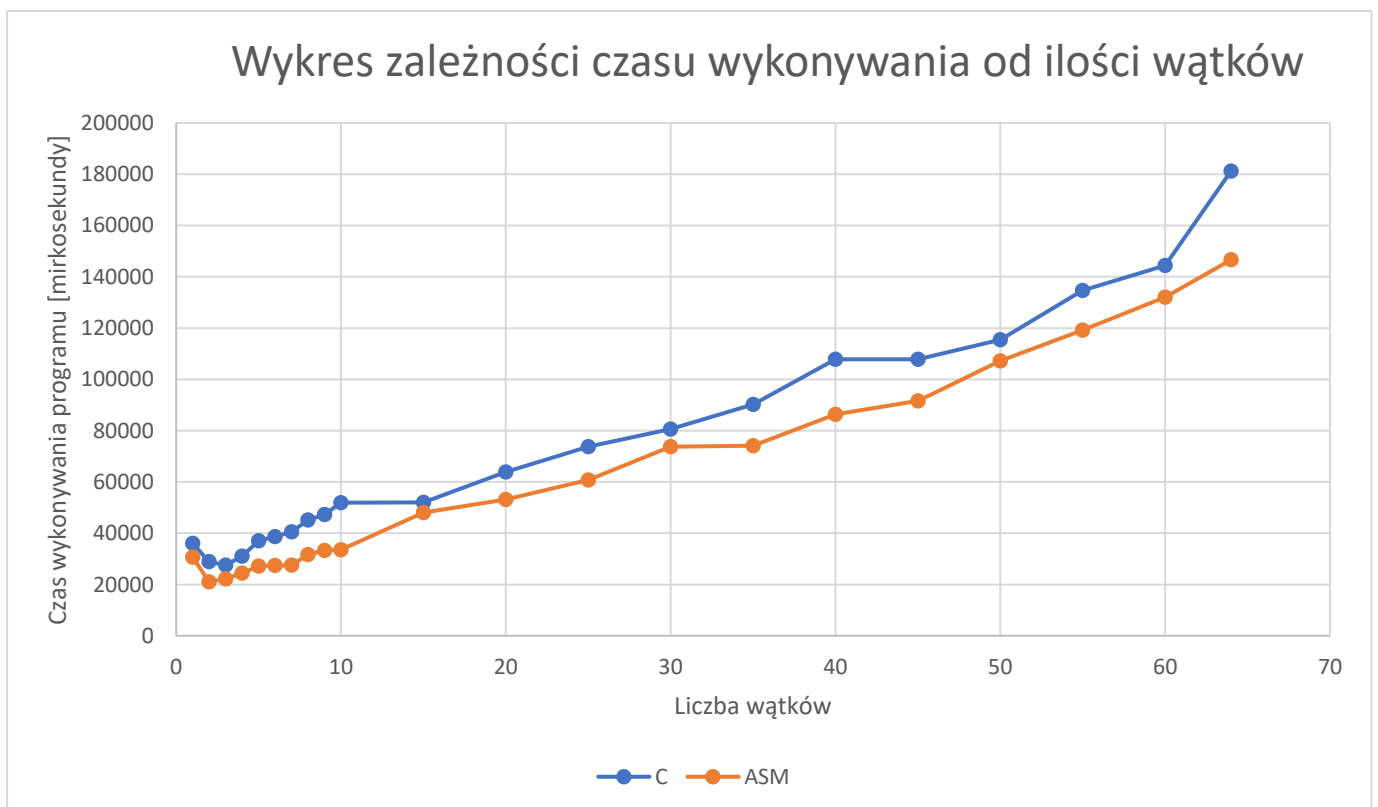
Tabela 1 Czas wykonywania funkcji w C i assemblerze w zależności od ilości wątków dla bitmapy o rozmiarze 880x520



b. 4712x3092

Liczba wątków	C	ASM
1	36028	30707
2	28947	21036
3	27502	22138
4	31108	24425
5	37032	27165
6	38704	27469
7	40506	27513
8	45191	31714
9	47259	33270
10	51850	33514
15	52046	48093
20	63946	53147
25	73703	60728
30	80598	73790
35	90203	74132
40	107797	86345
45	107862	91557
50	115409	107248
55	134615	119255
60	144455	132072
64	181237	146606

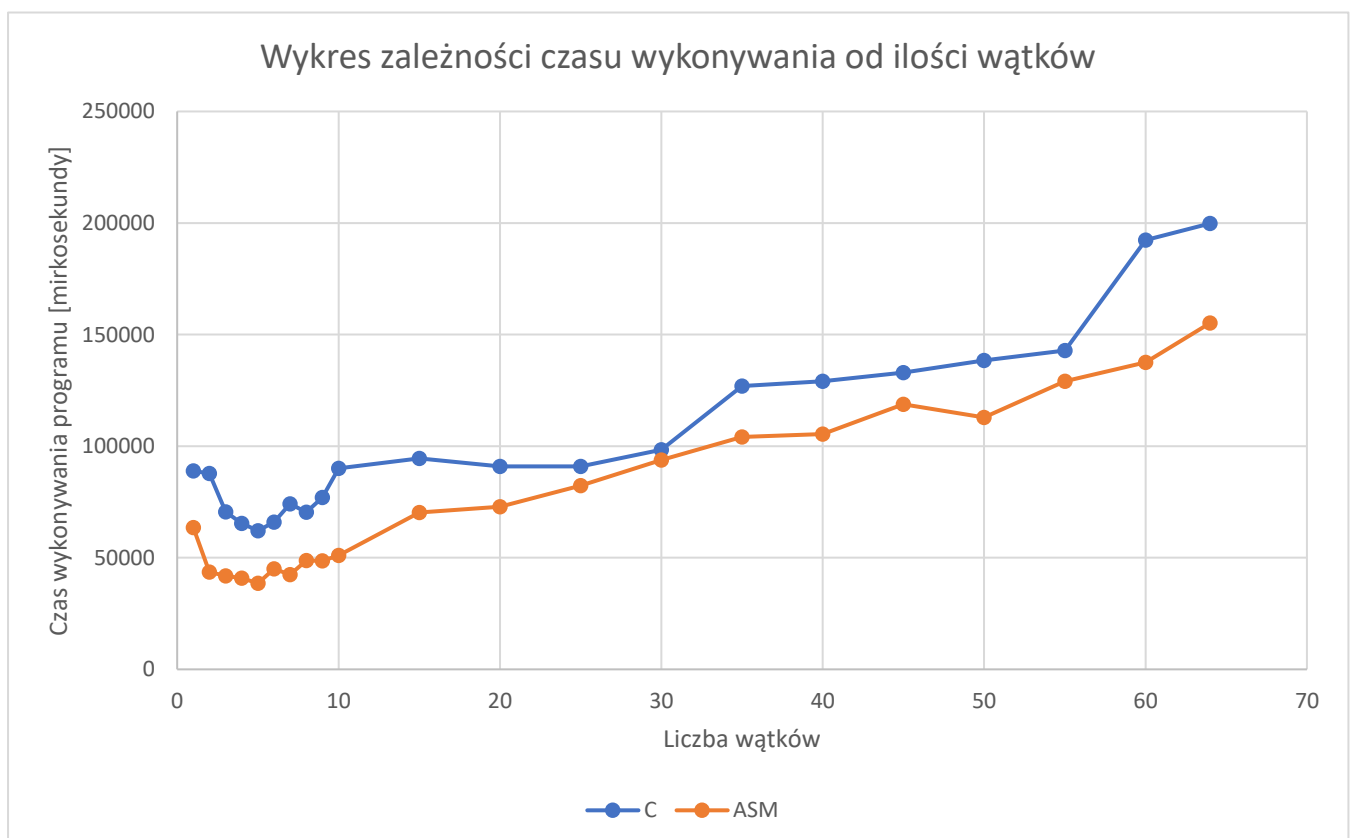
Tabela 2 Czas wykonywania funkcji w C i assemblerze w zależności od ilości wątków dla bitmapy o rozmiarze 4712x3092



c. 7352x5528

Liczba wątków	C	ASM
1	89003	63523
2	87823	43682
3	70606	41918
4	65454	40944
5	62142	38562
6	65959	45002
7	74162	42519
8	70413	48707
9	77093	48567
10	90024	51059
15	94579	70360
20	90938	72819
25	91002	82334
30	98441	93872
35	126917	104114
40	129073	105381
45	132934	118775
50	138475	112962
55	142868	129050
60	192422	137515
64	199801	155198

Tabela 2 Czas wykonywania funkcji w C i assemblerze w zależności od ilości wątków dla bitmapy o rozmiarze 735x5528



Funkcja napisana w assemblerze jest szybsza od funkcji napisanej w C, jednak dla małej bitmapy nie jest to aż tak widoczne. Dla dużych bitmap różnica pomiędzy assemblerem, a C jest większa.

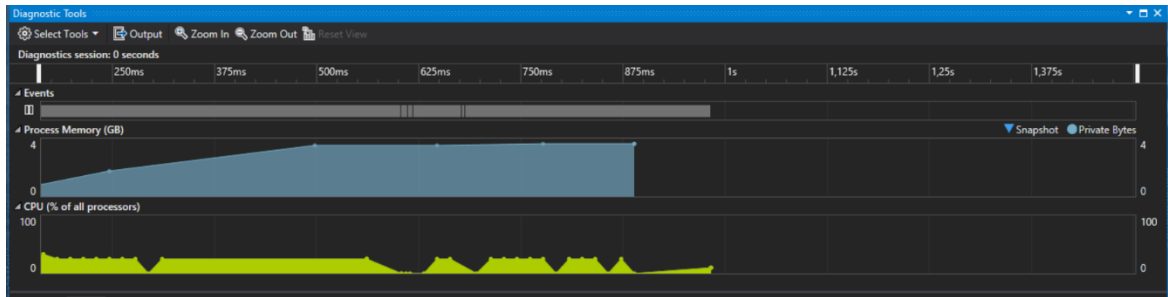
Przy mniejszych bitmapach nie jest opłacalne zwiększanie ilości wątków, ponieważ uruchamianie ich trwa dłużej niż przetworzenie całej małej bitmapy. Dla małej bitmapy optymalna ilość wątków dla C i assemblera to 3.

Dla dużej bitmapy optymalna ilość wątków dla assemblera to 12 wątków, dla C wynosi 9. Dla większej bitmapy opłaca się zwiększyć liczbę wątków. Jednak powyżej 12 wątków czas wykonywania rośnie znacząco, co spowodowane jest specyfikacją procesora, na którym były testowane dane.

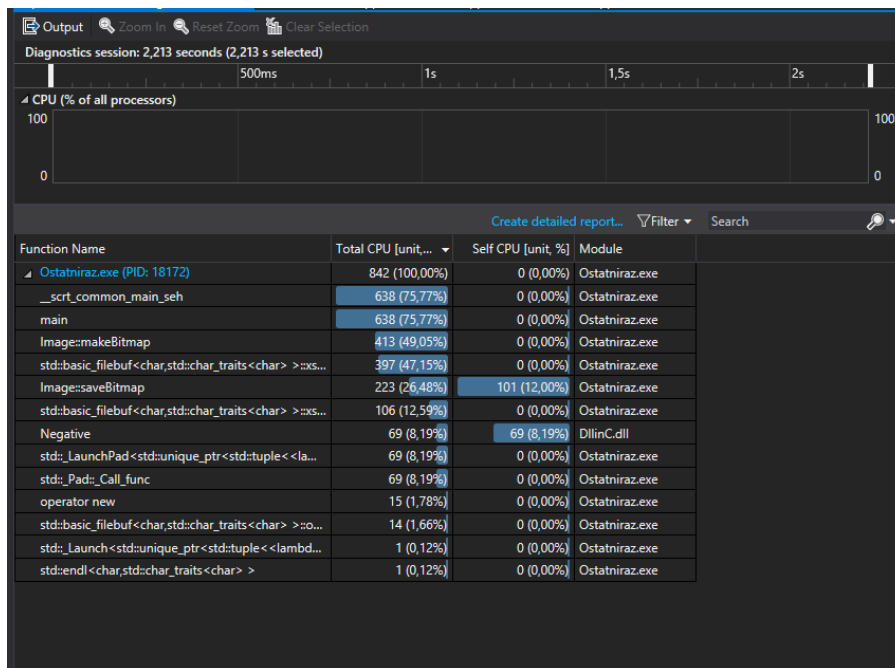
Analizę działania programu z wykorzystaniem modułu profilera

Analizę działania programu z wykorzystaniem modułu profilera przeprowadzono dla bitmapy o wielkości 7352x5528.

a. Język C

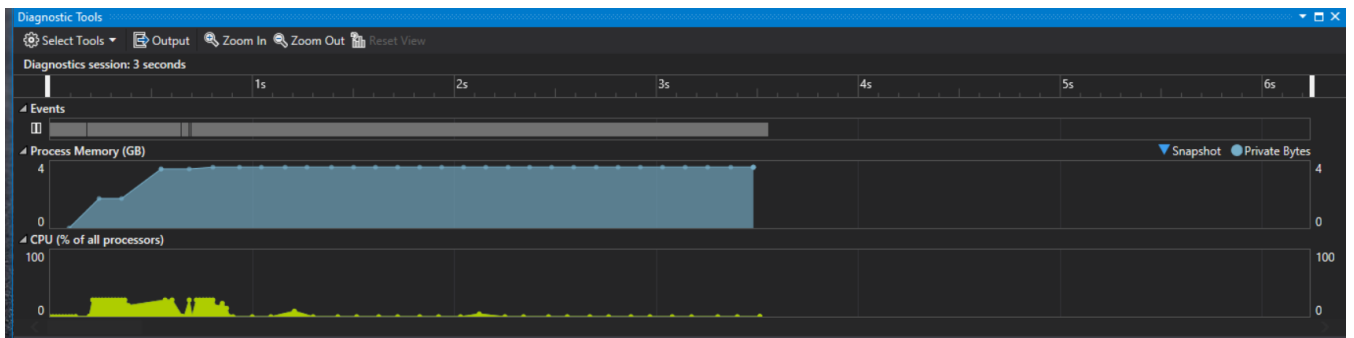


Rysunek 3 Analiza CPU i pamięci w trybie debug

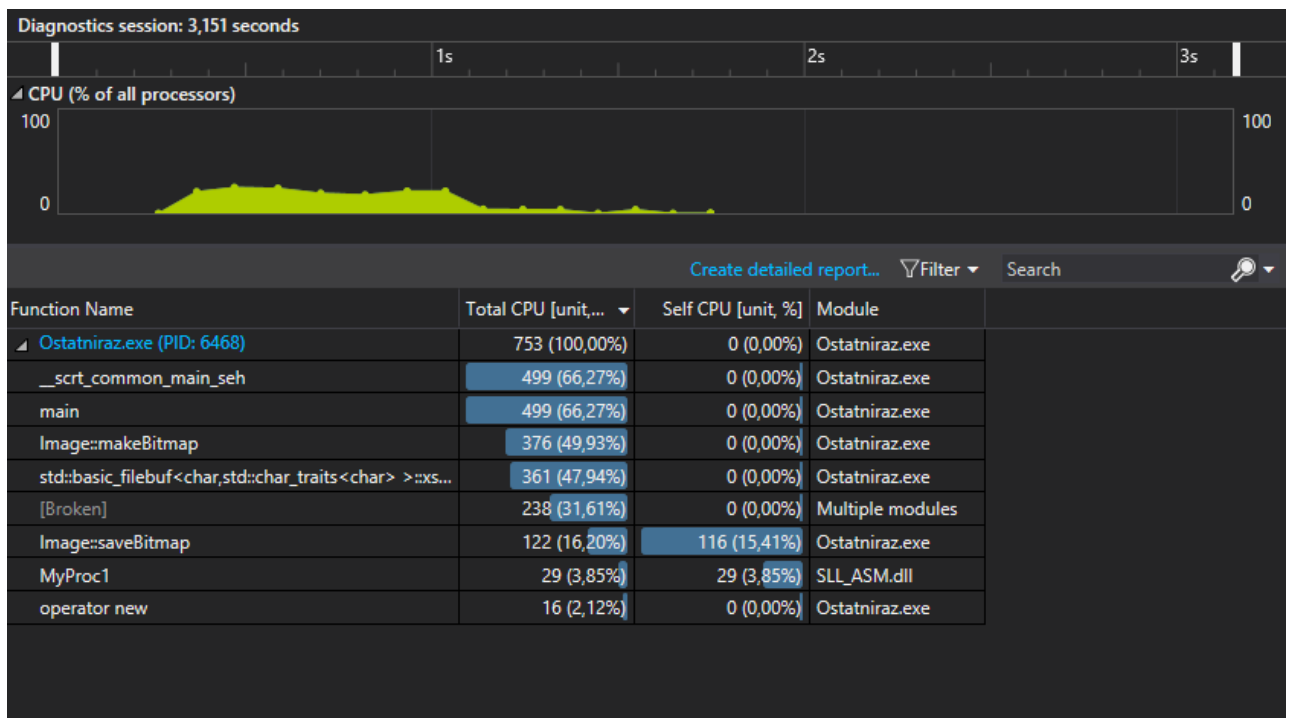


Rysunek 4 Analiza CPU bez debuggowania

b. ASM



Rysunek 5 Analiza CPU i pamięci w trybie debug



Rysunek 6 Analiza CPU bez debugowania

W raporcie wygenerowanym przez Profiler Visual Studio 2017 największa aktywność procesora następuje podczas otwarcie maina. Następnie podczas odczytu bitmapy z pliku, kolejno zapisu bitmapy do pliku. Aktywność procesora dla konwersji bitmapy jest mała, dla asemblera wynosi 3,85%. Dla C jest większa i wynosi 8,19%.

Instrukcja obsługi programu

Program można uruchomić z konsoli z następującymi parametrami:

- a. Bez podania liczby wątków:
`-i Charles.bmp -o Wyjście.bmp -2`
- b. Z podaniem liczby wątków:
`-i Charles.bmp -o Wyjście.bmp -2 -t 64`

Inne wywołania niż podane powodują pojawienie się pomocy oraz nazwy błędu:

```
Niepoprawna liczba wątków!  
Liczba wątków powinna być z zakresu od 1-64!
```

```
Nie poprawnie podana ścieżka  
Pomoc!  
Aby podać nazwę pliku wejściowego należy użyć komendy -i, a następnie podać nazwę pliku wejściowego  
Aby podać nazwę pliku wyjściowego należy użyć komendy -o, a następnie podać nazwę pliku wyjściowego  
Aby podać rodzaj przetwarzanego obrazu :  
-1 - C  
-2 - assembler  
Następnie -t - oznacza liczbę wątków do przetwarzania, a po przeczniku liczbę wątków z zakresu 1-64  
Przykładowe poprawne wywołania programu  
Przykładowe wywołanie -i SwietoHoli.bmp -o Wyjście -1  
Przykładowe wywołanie -i SwietoHoli.bmp -o Wyjście -1 -t 64
```

Wnioski

1. Udało się napisać szybszy program w assemblerze niż w C.
2. Dzięki projektowi nauczyłam się:
 - tworzyć biblioteki DLL;
 - działania rejestrów z rozszerzenia SSE;
 - implementować algorytmy w assemblerze z użyciem instrukcji wektorowych;
 - podziału programu na wątki.
3. Dowiedziałam się o wielu narzędziach pomagających podczas debugowania, analizy kodu oraz optymalizacji.

Literatura

- „Praktyczny kurs assemblera”, E. Wróbel, Helion, Gliwice 2004
- „Assembler. Sztuka programowania”, Randall Hyde, Helion, Gliwice 2004
- <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture?redirectedfrom=MSDN>