Instytut Informatyki Politechniki Śląskiej
Zespół Mikroinformatyki i Teorii Automatów Cyfrowych

Projekt JA

Rok akademicki	Rodzaj studiów*: SSI/NSI/NSM	Numer ćwiczenia:	Grupa	Sekcja
2019/2020	**SSI**	**-**	**2**	**3**

Prowadzący: **KH**

Sprawozdanie

Temat Projektu:

Zmiana kontrastu obrazu

AUTOR

Wojciech Waleszczyk

1.OPIS PROJEKTU

Projekt polegał na stworzeniu programu polegającego na zmianie kontrastu obrazu.
Projekt miał być napisany za pomocą 3 języków programowania. C# do ogólnego działania programu(GUI i obsługa DLL oraz wątków), C++ jako programu w pliku DLL, który będzie wykonywał pewną funkcjonalność tą samą co plik DLL w języku Assemblerowym.

2.ZAŁOŻENIA PROJEKTOWE

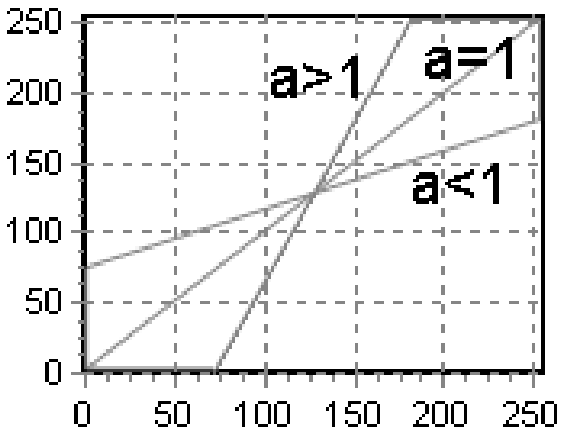
- Program będzie porównywał zmianę kontrastu obrazu w języku wysokiego poziomu(C++) i niskiego(Assembler)
- Program będzie posiadał GUI wykonane w C#
- Program będzie wykonywany na wątkach(1-64)
- 64B architektura programu

3.ALGORYTM UŻYTY W PROGRAMIE

Algorytm, który użyłem w programie polega na wypełnienie tak zwanej tablicy LUT odpowiednimi wartościami na podstawie algorytmu przedstawionego poniżej:

$$LUT(i) = \begin{cases} 0 & \text{jeżeli } a(i - \frac{i_{max}}{2}) + \frac{i_{max}}{2} < 0 \\ a(i - \frac{i_{max}}{2}) + \frac{i_{max}}{2} & \text{jeżeli } 0 \le a(i - \frac{i_{max}}{2}) + \frac{i_{max}}{2} \le i_{max} \\ i_{max} & \text{jeżeli } a(i - \frac{i_{max}}{2}) + \frac{i_{max}}{2} > i_{max} \end{cases}$$

Algorytm ten wypełnia tą tablice na podstawie liczby wybranej przez użytkownika programu od -255 do 255. Tablica wypełniana jest według tego algorytmu a następnie wartości wszystkich pikseli w obrazie są podmieniane na jej podstawie na inne zawarte w tej tablicy.



4.PLIKI DLL

4.1 C++

W plikach DLL postanowiłem zaimplementować przepisywanie tablicy LUT od wartości tablicy z pikselami do wynikowej tablicy pikseli. Dzięki tej operacji możemy zobaczyć już wynikowy obraz, ponieważ jest to ostatni operacja w algorytmie, która trzeba wykonać.

```
void __declspec(dllexport)  BitmapConvert(unsigned char* pixels, unsigned char* LUT)
{
    __m128i tab; //vector tab
    tab = _mm_loadu_si128((__m128i*)LUT); // loading 128 bits of tab LUT
    _mm_storeu_si128((__m128i*)pixels, tab); //  Store tab(LUT) in pixels tab
}
```

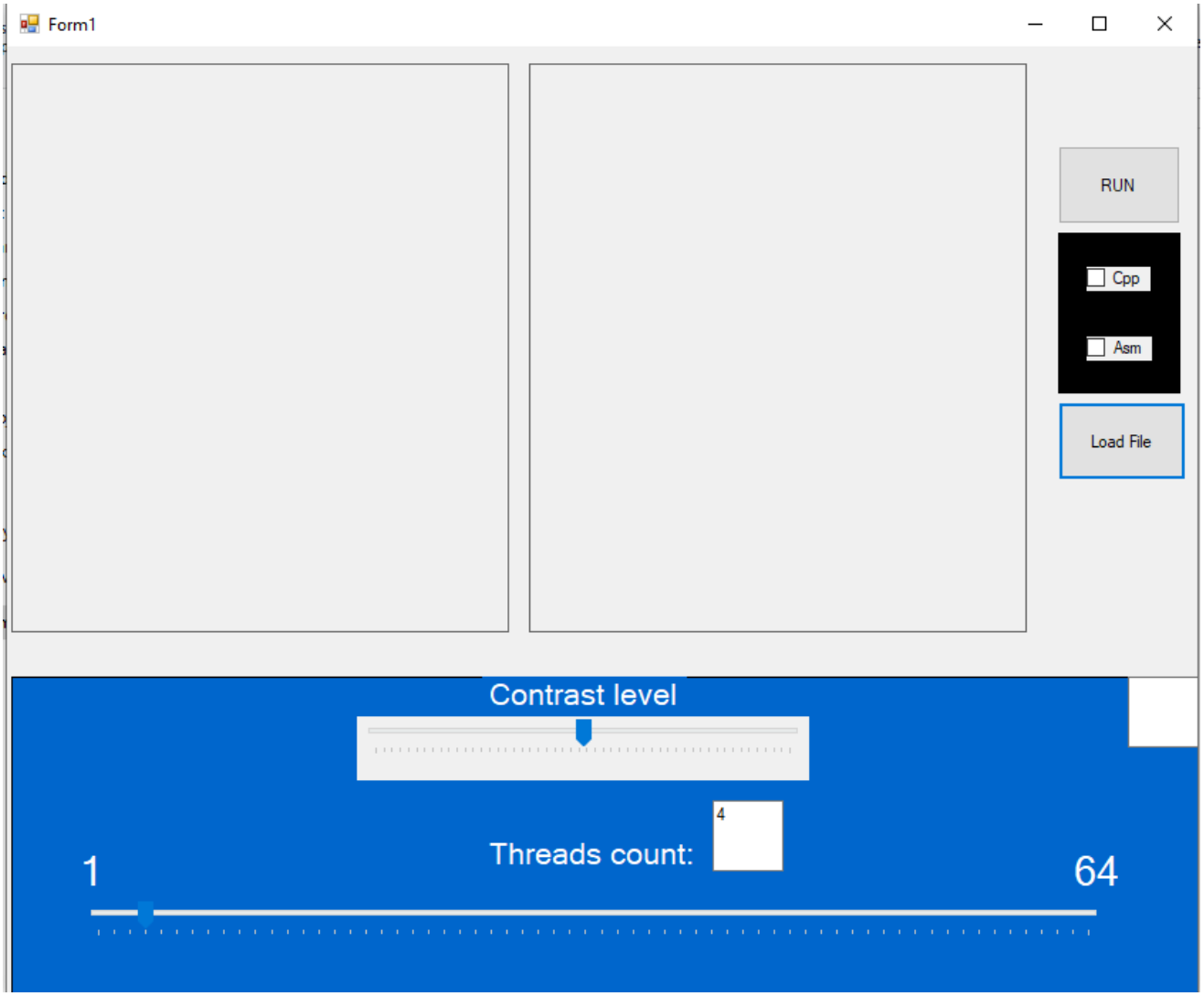
4.2 ASM

W pliku DLL z assmeblerem wykonałem tą samą funkcjonalność co w pliku C++, ponieważ było to jednym z założeń projektowych.

```
operateOnPixelsAsm PROC

    movdqu xmm1, [rbx + 0*SIZEOF BYTE]  ;load vector LUT

    movdqu [rcx],xmm1    ; Store vector LUT in pixels array

    ret
operateOnPixelsAsm ENDP
END
```

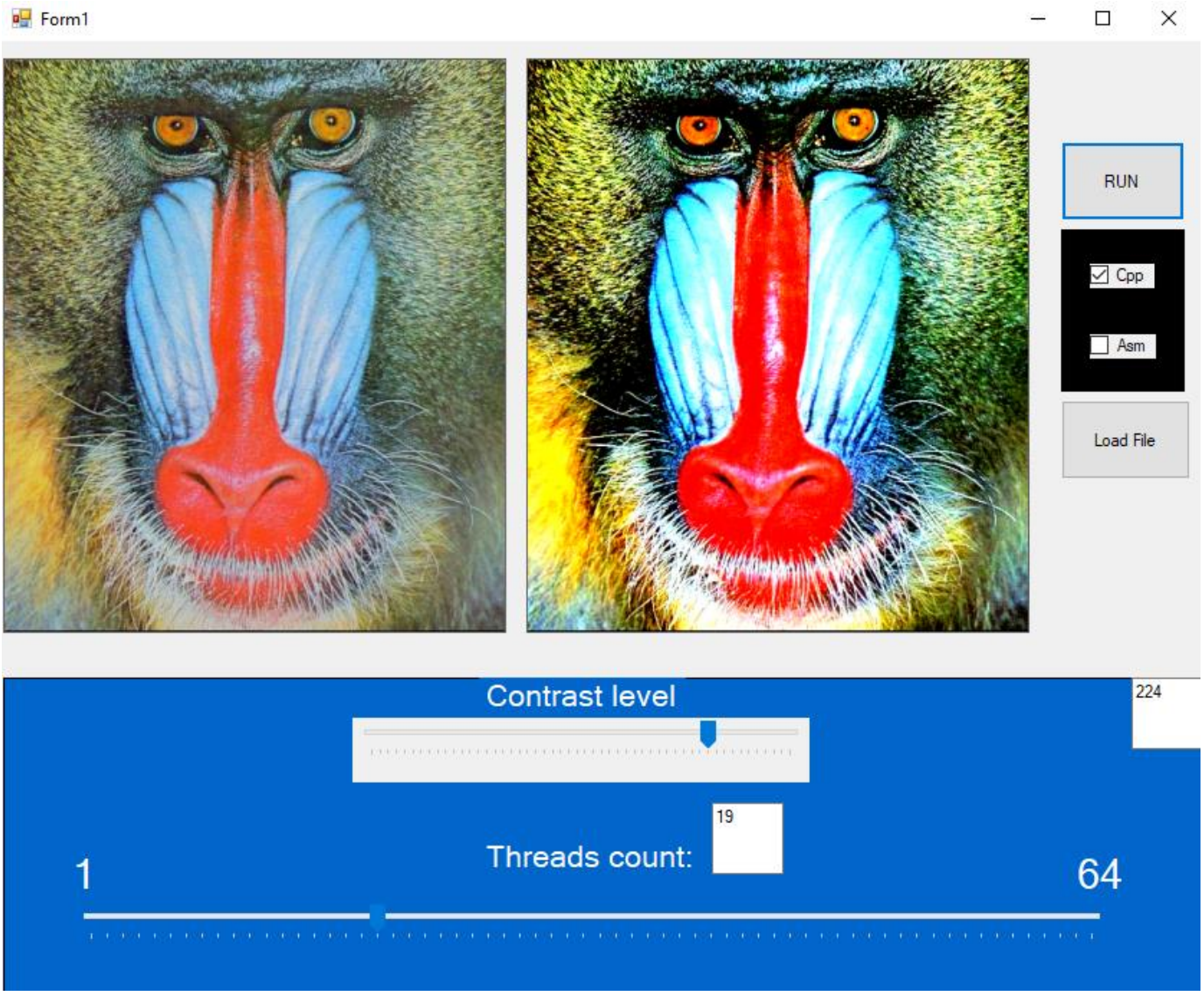
Plik ten nie jest zbyt obszerny ponieważ algorytm, którego postanowiłem użyć nie jest skomplikowany. W tej DLL przypisuje wartość tablicy przekazywanej w 2 parametrze do tablicy wynikowej która znajduje się w 1 parametrze. Aby wykonać tą operację posługuję się rejestrami, w których umieszczam te tablice. Następnie po prostu przypisuje wektor xmm1 do rejestru wynikowego, którym jest rejestr rcx.

5.OGÓLNY WYGLĄD PROGRAMU

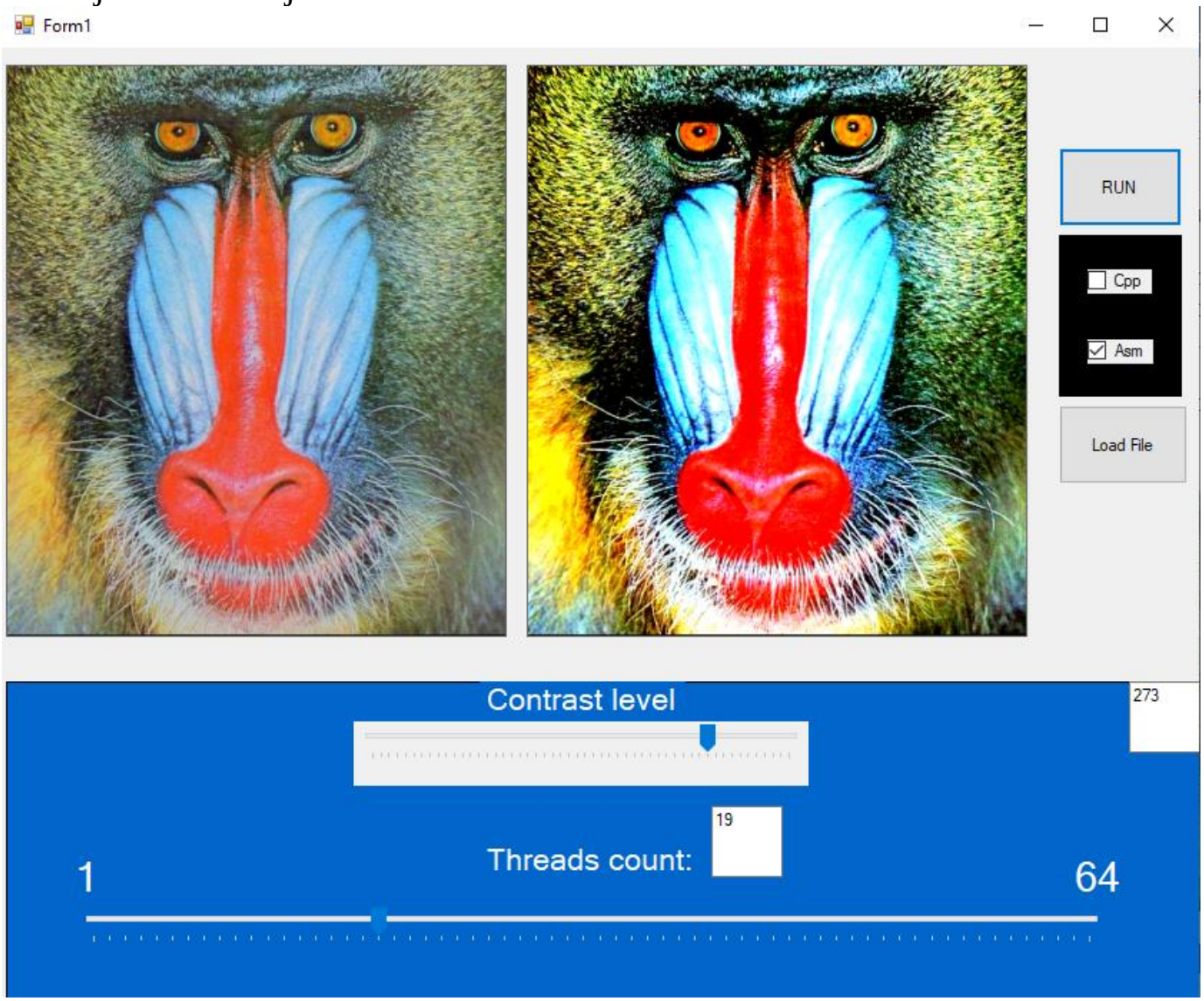


Zdjęcie przedstawione powyżej jest moim GUI, które zostało wykonane w „formsie”. Znajdują się tu 2 miejsca na zdjęcia. Jedno na oryginalne 2 na przerobione.
Suwak do ustawiania poziomu kontrastu oraz liczby wątków(od 1 do 64).
2 klawisze, jeden do załadowania obrazka i drugi do rozpoczęcia programu.
2 check boxy do wybrania DLL na, której program ma pracować.

Poniżej przedstawiam przykładowe działanie programu dla realizacji w C++



A tutaj dla realizacji w ASM



Jak widać w białym polu po lewej, te 2 przykładowe działania (jedno z wykorzystaniem DLL C++ a drugie ASM) różnią się tylko czasem wykonania, który jest minimalnie lepszy w programie, który użył DLL w C++.(Czas podany jest z ms)

6.PORÓWNANIE CZASOWE

Tabela poniżej dla obrazu wielkości S

DLL \ Wątki	4	16	28	36	48	56	64
C++	10ms	38ms	67ms	86ms	110ms	133ms	150ms
ASM	12ms	40ms	65ms	85ms	112ms	129ms	149ms

Tabela poniżej dla obrazu wielkości M

Wątki / DLL	4	16	28	36	48	56	64
C++	10ms	27ms	66ms	82ms	110ms	129ms	151ms
ASM	9ms	37ms	66ms	85ms	110ms	129ms	146ms

Tabela poniżej dla obrazu wielkości L

Wątki / DLL	4	16	28	36	48	56	64
C++	13ms	36ms	58ms	79ms	109ms	126ms	156ms
ASM	9ms	36ms	61ms	61ms	107ms	125ms	144ms

Porównania czasów były wykonywane na 3 różnych obrazach wielkości S,M oraz L. Kontrast był ustawiony na jedną wartość równą 255. Algorytm ten jeżeli chodzi o czas jest nie miarodajny, ponieważ dla różnych obrazów tej samej wielkości może być inny gdyż algorytm ten opiera się na podmianie pikseli. Analizując dane w tabelach powyżej możemy wywnioskować, że assembler w obrazach większej wielkości niż S jest bardziej wydajny. Praca na więcej niż 4 wątkach była dłuższa niż dla mniejszej ilości. Jest to związane z tym, że mój komputer pracuje na 4 wątkach wiec załóżmy gdy chcemy uruchomić program dla 36 wątków to 4 wątki wykonują maksymalnie swoje zadanie a 32 pozostałe czekają na zwolnienie się jakiegoś procesora logicznego. Dlatego właśnie najbardziej optymalne jest uruchomienie programu na 4 wątkach.

7.INSTRUKCJE WEKTOROWE

W programie użyłem tylko kilku instrukcji wektorowych a były nimi:
_m128i – użyłem jej aby zadeklarować wektor zawierający liczby całkowite (w moim przypadku były to wartości z tablicy LUT). Wektor ten zawiera 16 liczb jednobajtowych lub 8 słów.
_mm_storeu_si128 – instrukcja ta pozwala na zdeponowanie wektora _m128i który wcześniej zadeklarowałem w wynikowym wektorze, który znajduje się w jego pierwszym argumencie.

_mm_storeu_si128(wektor wynikowy, wektor do zdeponowania)

8.WNIOSKI

Przy pracy nad tym projektem udoskonaliłem swoje umiejętności w programowaniu w języku niskiego poziomu jakim jest assembler jak również nauczyłem się pracy z C#. Poznałem również czasową różnice między tym samym programem w c++ oraz assemblerze. Różnice pomiędzy czasami wykonania programu w ASM i C++ nie są zbyt duże dla danych S,M i L jednak gdyby porównywać obrazy bardzo dużych rozmiarów różnica mogła by być widoczna. Podczas wykonywania projektu poznałem również instrukcje wektorowe C++ i są one dość przydane jeżeli pracujemy z obrazami lub innego tego typu danymi.

Do zrealizowania problemu zawartego w projekcie jakim była edycja kontrastu obrazu zrealizowałem na algorytmie polegającym na wypełnianiu odpowiedniej tablicy. Jednak sposobów na jego realizacje jest więcej. Ja wybrałem ten bo wydał mi się odpowiedni do jego zaimplementowania.

Projekt wymagał ode mnie dużego poświęcenia czasowego, ponieważ dopiero poznawałem świat języka niskiego poziomu jakim jest assembler oraz instrukcji wektorowych, które używałem dopiero pierwszy raz,