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	**Języki Asemblerowe**	**2**	**3**

Prowadzący przedmiot:	mgr inż. Krzysztof Hanzel	**Termin: (dzień tygodnia godzina)**
Imię:	**Kamil**	**18.02.2020**
Nazwisko:	**Susek**	**12:00**
Email:	**kamisus961@student.polsl.pl**	

Sprawozdanie z projektu

Temat projektu:

Wykrywanie krawędzi metodą Robertsa.

Główne założenia projektu:

Program posiada interfejs graficzny, obsługiwany przez użytkownika (implementacja w c#). Program posiada funkcję zaimplementowaną w c++ oraz asm. Czas realizacji funkcji przez powyżej wspomniane implementacje jest zliczany. Program umożliwia wykorzystanie wielowątkowości (1-64 wątków). Program realizuje algorytm wykrywania krawędzi, na wybranym przez użytkownika obrazie. Tworzony jest obraz wynikowy z zaznaczonymi krawędziami. Biblioteka 64-bitowa.

1.Analiza zadania.

Algorytm Robertsa wykorzystywany jest w procesie obróbki graficznej obrazów. Jego działanie pozwala na wykrycie krawędzi w danym obrazie. Algorytm Robertsa wykorzystuje okno o rozmiarze 2x2 piksele (macierz W). Na podstawie wartości pikseli w oknie obliczana jest nowa wartość piksela, która zapisywana jest w tablicy wynikowej (macierz Z).

$$W_{i,j} = \begin{pmatrix} a_{i,j} & a_{i+1,j} \\ a_{i,j+1} & a_{i+1,j+1} \end{pmatrix}$$

$$Z_{i,j} = |a_{i,j} - a_{i+1,j+1}| + |a_{i+1,j} - a_{i,j+1}|$$

2. Realizacja zadania.

Wykorzystanie rozkazów wektorowych w algorytmie Robertsa wymaga dopasowania zbioru danych. W tym celu tablica P zawierająca przetwarzany obraz została podzielona według następującej zależności:

*Dla każdego i = 0, 1, 2, …, 2 * WYSOKOŚĆ_OBRAZU * SZEROKOŚĆ_OBRAZU*

$$A_{2i} = P_i, A_{2i+1} = P_{i+1}$$

$$B_{2i} = P_{i+1+W}, B_{2i+1} = P_{i+W}$$

, gdzie W to szerokość obrazu w pikselach.

Dane podzielone na tablice A i B są przetwarzane w następujący sposób

$$Z_i = |A_i - B_i|$$

Co prowadzi do uzyskania wyjściowego obrazu zapisanego w tablicy Z.

3. Realizacja bibliotek dll.

Jednym z wymagań projektu było porównanie czasów działania bibliotek dll w c++ i asemblerze. Dlatego ważnym aspektem realizacji oprogramowania było, jak największe podobieństwo bibliotek, pod względem wykonywanych operacji. W tym celu wykorzystane zostały funkcje z biblioteki "nmmintrin.h", które są implementacją rozkazów z technologii SSE w języku c++.

3.1.Biblioteka w języku wysokiego poziomu (c++).

```
extern "C"  void  __declspec(dllexport)  operateOnPixelsCpp(unsigned char
*pixels, unsigned char *copy1, unsigned char *copy2)
{
	//vector a and b
	__m128i a, b;
	// loading 128 bits of tab copy1 and copy 2
	a = _mm_loadu_si128((__m128i*)copy1);
	b = _mm_loadu_si128((__m128i*)copy2);
	// subtract b from a and get absolute value, next store result in pixels
array
	_mm_storeu_si128((__m128i*)pixels, (_mm_abs_epi8(_mm_sub_epi8(a, b))));
}
```

3.2. Biblioteka w Asemblerze.

```
operateOnPixelsAsm PROC

	movdqu xmm1, [rbx + 0*SIZEOF BYTE]  ; load vector 1
	movdqu xmm2, [rdx + 0*SIZEOF BYTE]  ; load vector 2

	psubb xmm1,xmm2                     ; subtract vectors

	pabsb xmm1,xmm1                     ; get absolute value

	movdqu [rcx],xmm1          ; load absolute value to result array

	ret
operateOnPixelsAsm ENDP
```

3.3. Porównanie działania bibliotek.

Kolejno wykonywane operacje na danych, przez biblioteki opisuje wzór:

$$Z_i = |A_i - B_i|.$$

Na samym początku każdej funkcji pobierane są wskaźniki na tablicę przechowujące dane wejściowe. W przypadku funkcji w c++ wskaźniki te są podawane, jako parametry funkcji, natomiast w procedurze asemblerowej znajdują się one w rejestrach rbx i rdx. Odczytanie wskaźników pozwala na znalezienie obszaru w pamięci, z którego można pobrać 128 bitów, na których następnie wykonywana jest operacja odejmowania. Z wyniku odejmowania wyciągana jest wartość bezwzględna, która zapisywana jest do tablicy wynikowej (w asemblerze jej adres znajduje się w rejestrze rcx).

3.4. Wykorzystane instrukcje wektorowe.

PSUBB XMM0, XMM1

128-bitowe rejestry XMM0 i XMM1 dzielone są na 8-bitowe “paczki”.Rozkaz odejmuje zawartość rejestru XMM1, od XMM0 traktując kolejne 8-bitów jako osobne liczby. Wynik operacji znajduje się w XMM0.

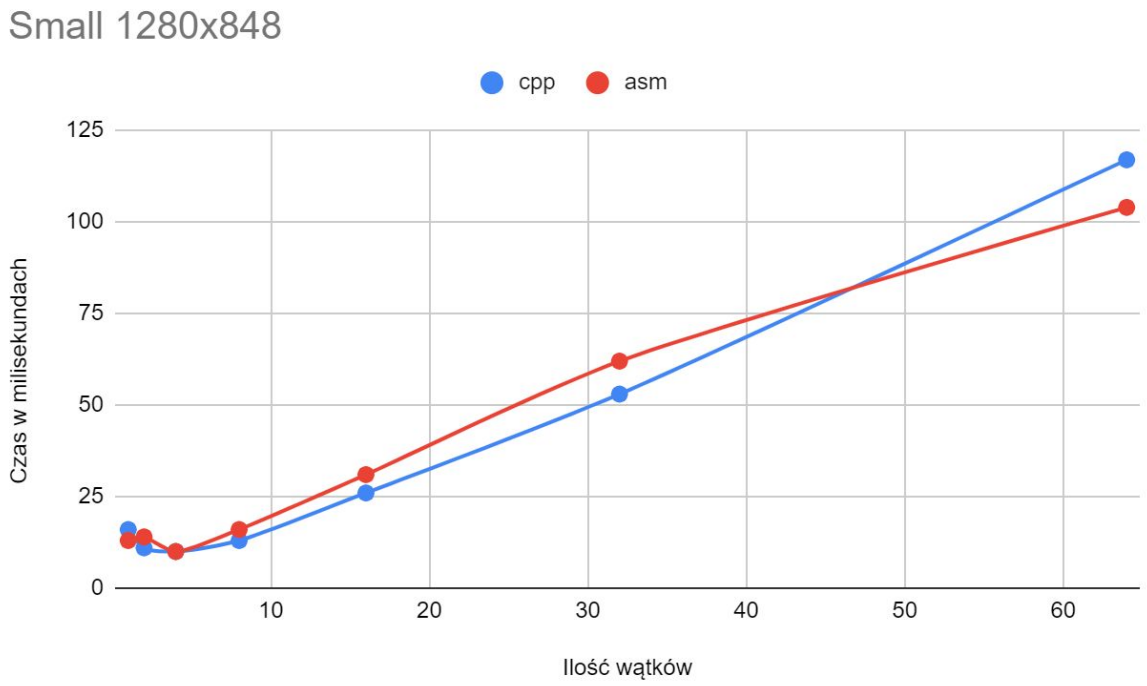
PABSB XMM0

Rejestr XMM0 dzielone jest na 8-bitowe “paczki”. Rozkaz oblicza wartość bezwzględną XMM0. Wynik operacji znajduje się w XMM0.

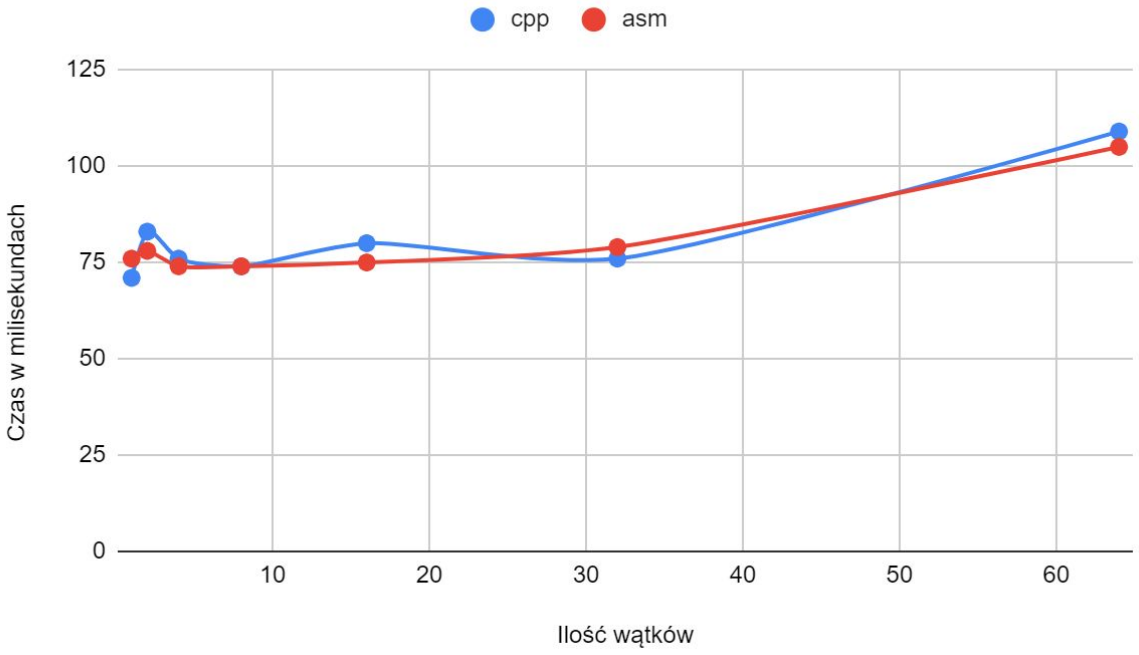
MOVDQU a, b

Rozkaz pozwala na załadowanie 128-bitowej liczby całkowitej, z rejestru b do a. Adres a w pamięci nie musi dzielić się na 16, tak jak w przypadku rozkazu MOVDQA.

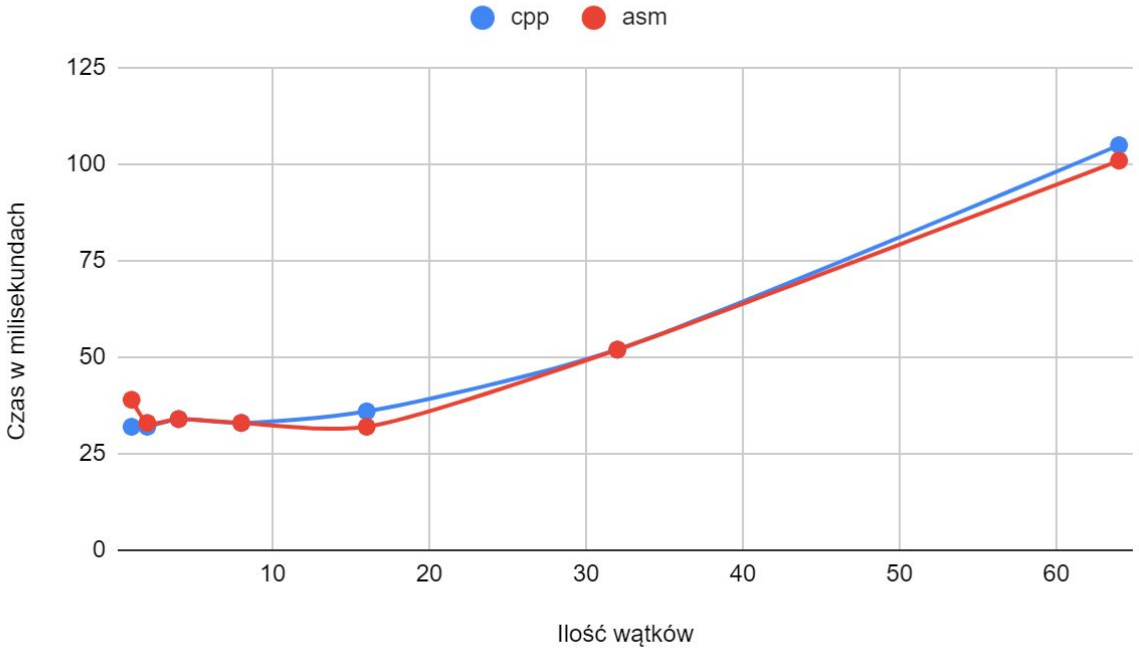
4.Porównanie czasów działania.



Medium 2560x1440



Large 2668x3181



5.Podsumowanie.

Efektywne wykorzystanie rozkazów wektorowych wymaga dopasowania zbioru danych. Koszt dopasowania danych czasami może być większy, niż zysk wynikający z wykorzystania rozkazów SIMD.

Najbardziej optymalne wyniki czasowe zostały uzyskane, gdy liczba aktywnych wątków nie przekraczała maksymalnej liczby wątków procesora.

Testy przeprowadzane były na 4 wątkowym procesorze.