HORYZONTALNE ROZMYCIE OBRAZU

Daniel Cogiel

Algorytm rozmycia horyzontalnego

- Przy obliczaniu wartości przetworzonego obrazu brane są pod uwagę 4 punkty z otoczenia
- Używana maska – 5 x 1
- Wszystkie wagi punktów wynoszą 1

f(-2,0)	f(-1,0)	f(-0,0)	f(1,0)	f(2,0)

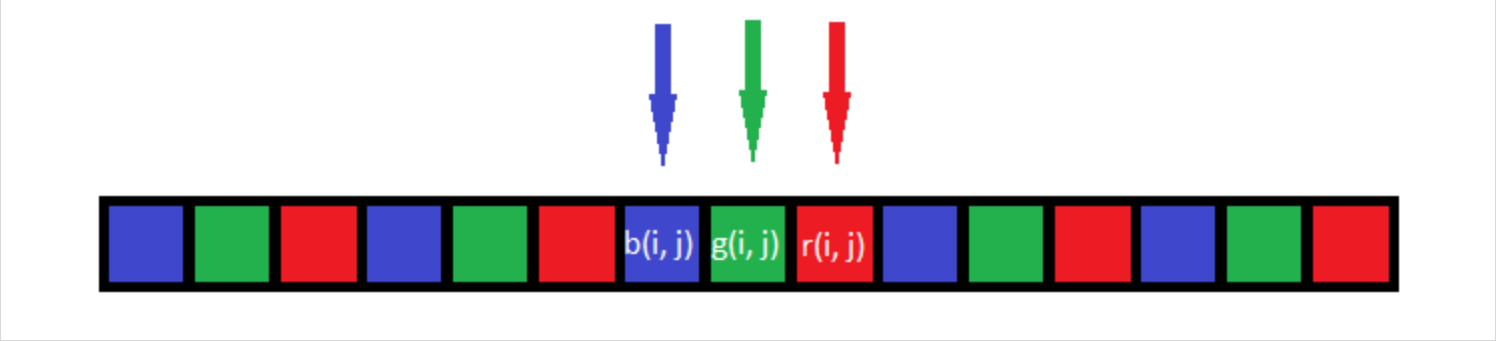
$$\grave{a}_{i,j} = \frac{f_{-2,0} \cdot a_{i-2,j} + f_{-1,0} \cdot a_{i-1,j} + f_{0,0} \cdot a_{i,j} + f_{1,0} \cdot a_{i+1,j} + f_{2,0} \cdot a_{i+2,j}}{f_{-2,0} + f_{-1,0} + f_{0,0} + f_{1,0} + f_{2,0}}$$

Po uproszczeniu:

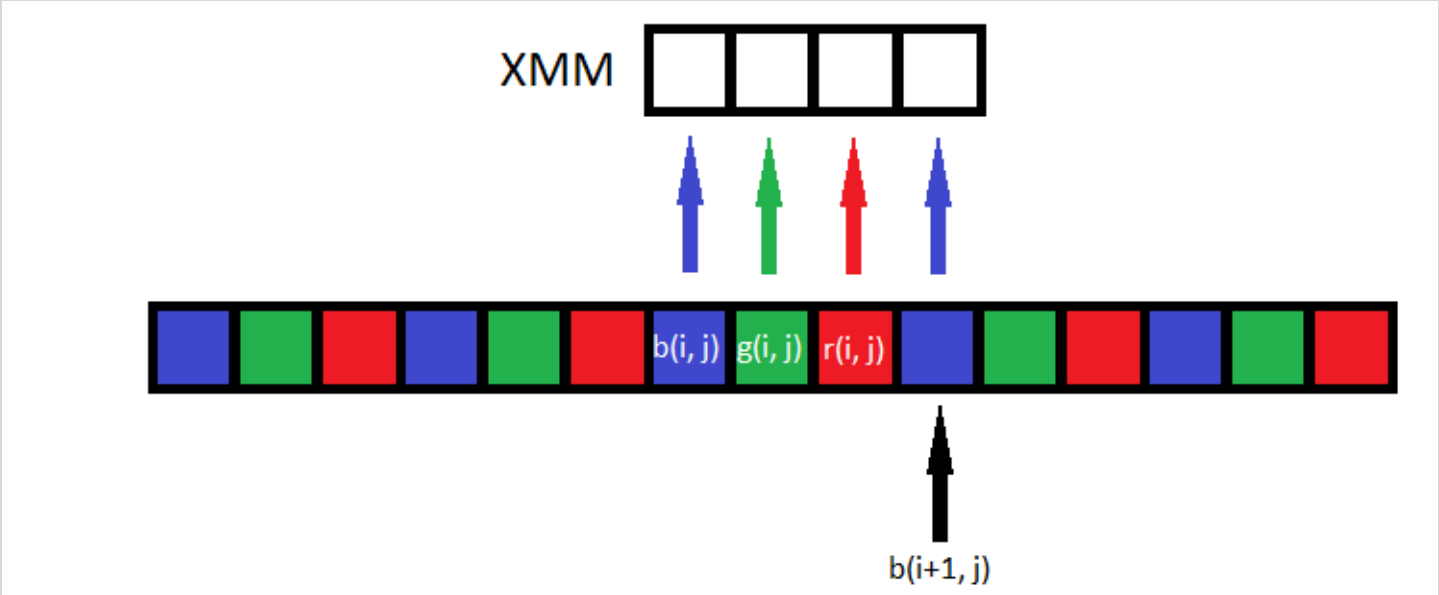
$$\grave{a}_{i,j} = \frac{a_{i-2,j} + a_{i-1,j} + a_{i,j} + a_{i+1,j} + a_{i+2,j}}{5}$$

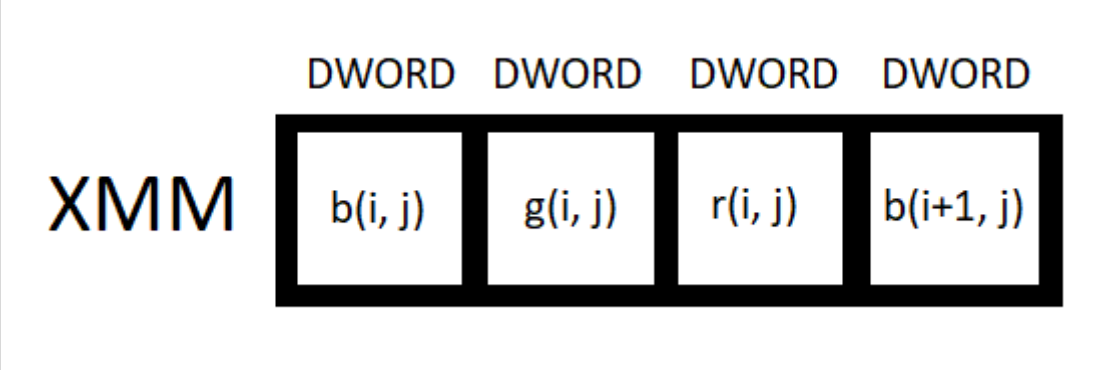
31.01.2023

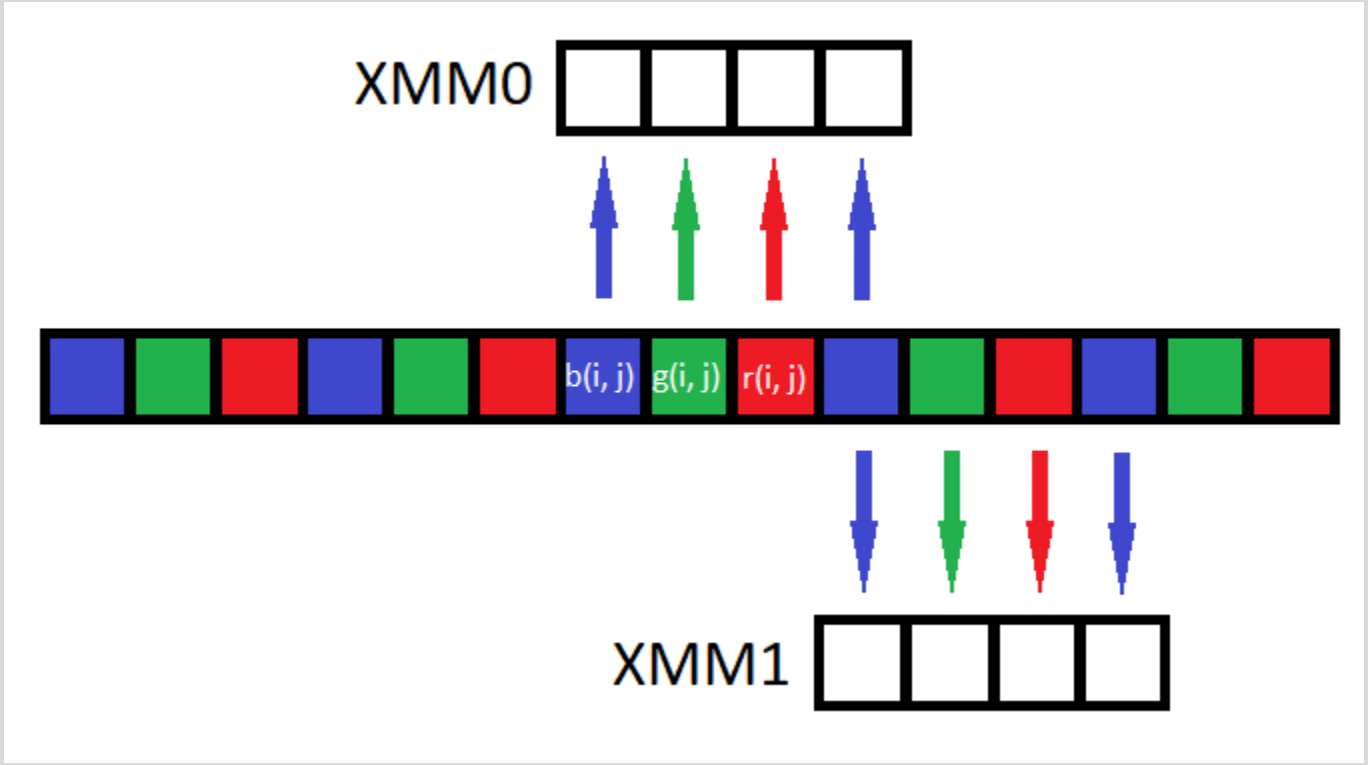
IMPLEMENTACJA



- $b_{i,j}^{`} = \frac{b_{i-2,j}+b_{i-1,j}+b_{i,j}+b_{i+1,j}+b_{i+2,j}}{5}$
- $g_{i,j}^{`} = \frac{g_{i-2,j}+g_{i-1,j}+g_{i,j}+g_{i+1,j}+g_{i+2,j}}{5}$
- $r_{i,j}^{`} = \frac{r_{i-2,j}+r_{i-1,j}+r_{i,j}+r_{i+1,j}+r_{i+2,j}}{5}$


XMM
b(i, j)
g(i, j)
r(i, j)
b(i+1, j)


DWORD
DWORD
DWORD
DWORD
XMM
b(i, j)
g(i, j)
r(i, j)
b(i+1, j)



- Aktualne sumy wartości piksela są przetrzymywane w rejestrze XMM0
- Rejestr XMM1 jest używany do dodawania kolejnych wartości pikseli

Operacje na przetwarzanych pikselach

- 4 bajty obrazu są wczytywane na koniec rejestru XMM0
- Wczytane bajty zamieniane są na DWORD-y
- Kolejne 4-bajtowe paczki są wczytywane do XMM1, w odstępach 3-bajtowych (-6, -3, +3, +6)
- Wartości są sumowane i zapisywane w XMM0, a następnie dzielone przez 5

Operacja dzielenia w XMM0

- Dobrą praktyką podczas wykonywania operacji arytmetycznych jest używanie mnożenia bądź przesunięcia bitowego zamiast dzielenia
- W omawianym algorytmie odbywa się dzielenie przez 5, nie można więc zastąpić go przesunięciem bitowym w prawo
- Można zastosować wzór:

$$\frac{x}{k} \approx \left(x \cdot \frac{2^n}{k} + \frac{2^n}{k} \right) \gg n$$

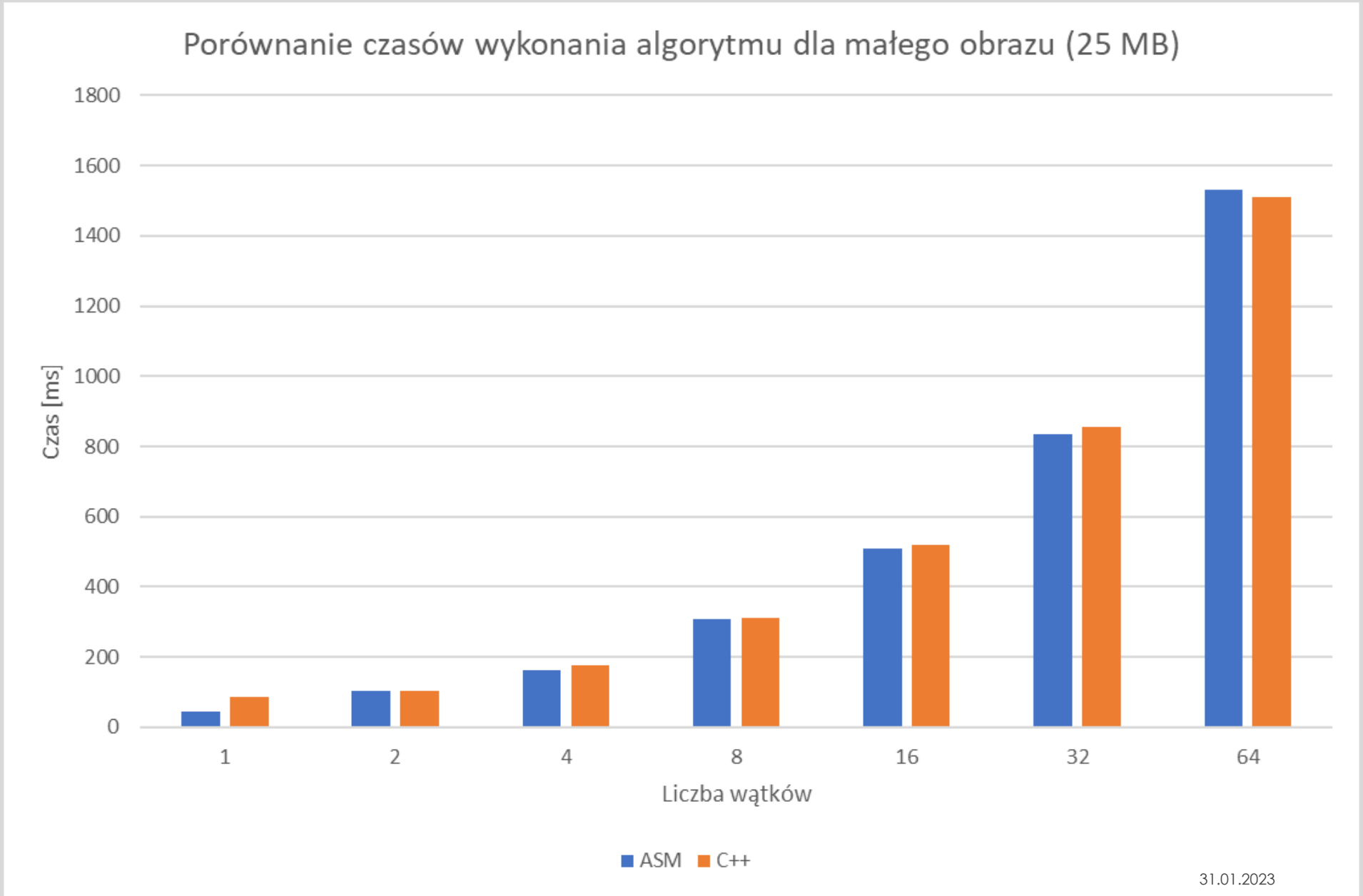
- W implementowanym algorytmie wybrano wartość n=4:

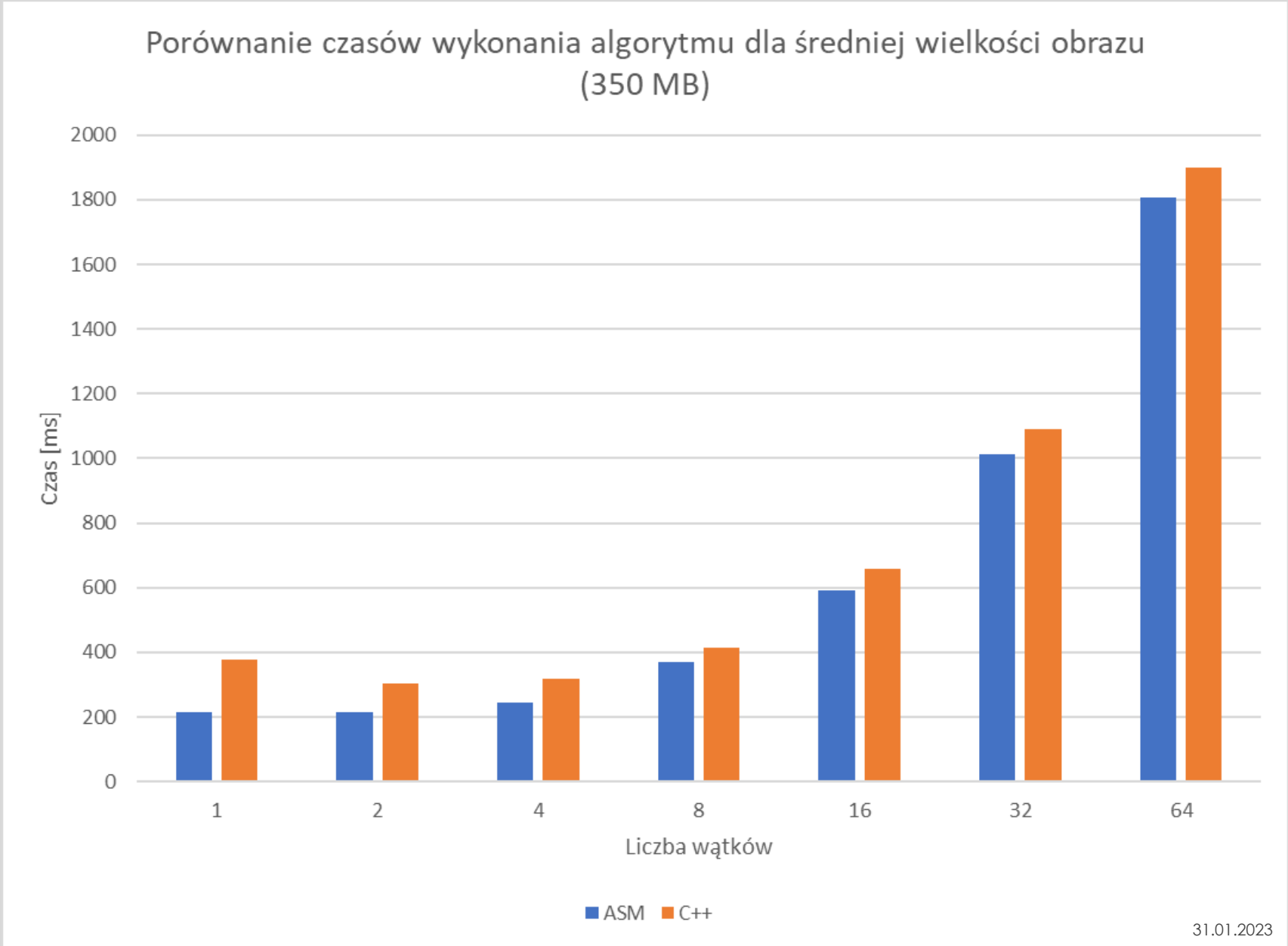
$$\frac{2^n}{k} = \frac{2^4}{5} \approx 3$$

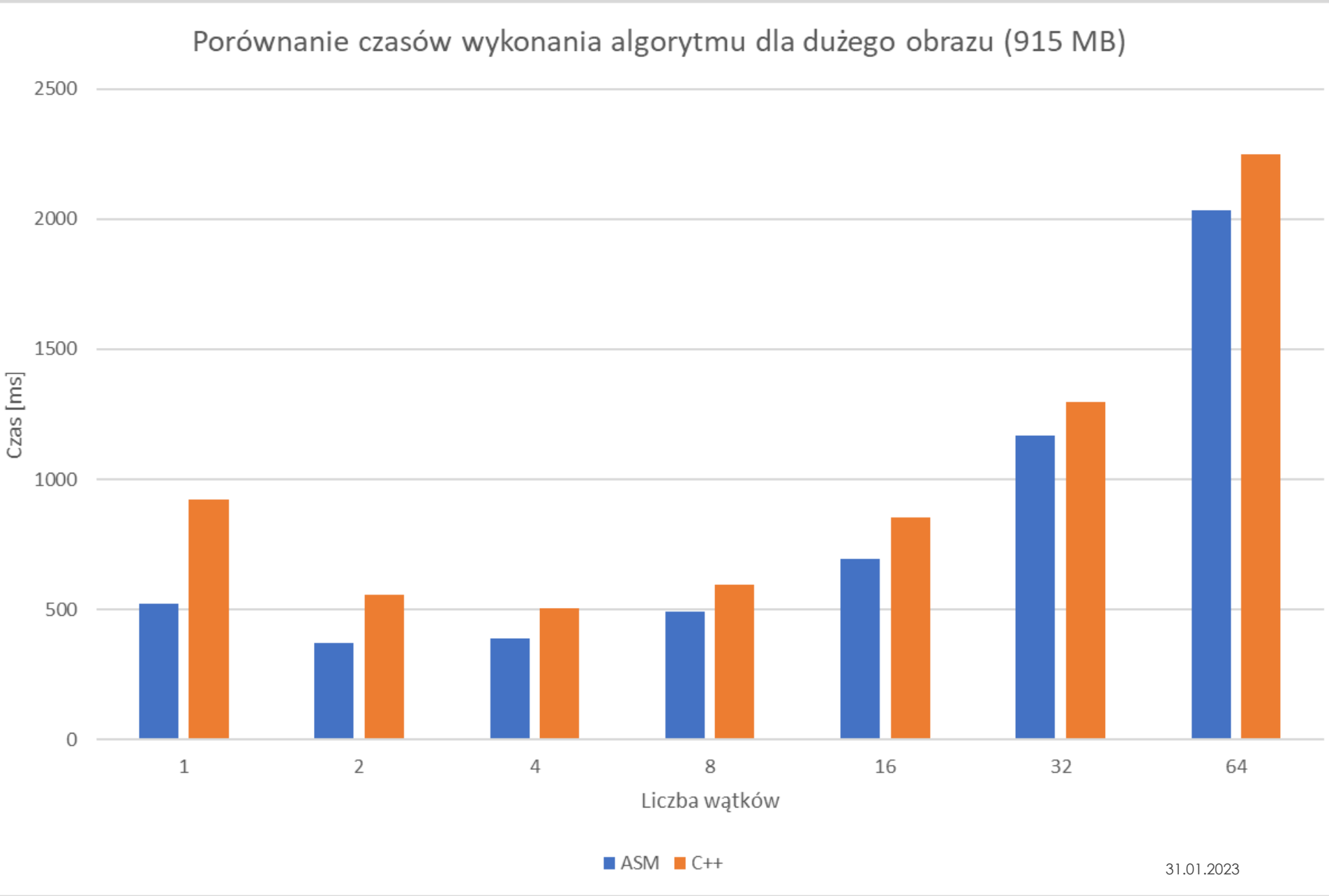
$$\frac{x}{5} \approx (x \cdot 3 + 3) \gg 4$$

31.01.2023

WYNIKI CZASOWE


Porównanie czasów wykonania algorytmu dla małego obrazu (25 MB)
1800
1600
1400
1200
1000
800
600
400
200
0
Czas [ms]
1
2
4
8
16
32
64
Liczba wątków
ASM
C++
31.01.2023


Porównanie czasów wykonania algorytmu dla średniej wielkości obrazu (350 MB)
2000
1800
1600
1400
1200
1000
800
600
400
200
0
Czas [ms]
1
2
4
8
16
32
64
Liczba wątków
ASM
C++
31.01.2023


Porównanie czasów wykonania algorytmu dla dużego obrazu (915 MB)
2500
2000
1500
1000
500
0
Czas [ms]
1
2
4
8
16
32
64
Liczba wątków
ASM
C++
31.01.2023

31.01.2023

EFEKTY DZIAŁANIA ALGORYTMU

Wnioski

- Pisanie algorytmów w języku asemblerowym zapewnia przewagę w kwestii szybkości wykonania nad innymi znanymi językami (np. z rodziny C)
- W trakcie pisania algorytmów w języku asemblerowym należy zwracać szczególną uwagę na naruszenia dostępu do pamięci
- Przewagę szybkości algorytmu asemblerowego nad algorytmem w języku C++ najlepiej widać, gdy rozmycie wykonywane jest w pojedynczym wątku
- Przetwarzanie algorytmu w wielu wątkach zapewnia korzyści przy dużych rozmiarach obrazów
- Instrukcje wektorowe zapewniają sprawniejsze przetwarzanie danych w tzw. paczkach

31.01.2023

DZIĘKUJĘ ZA UWAGĘ!

Daniel Cogiel