Grupa 3	Albert Soczewski, Maciej Ząbek	02/05/2018

Zadanie 2 - Rozmycie Gaussa w MPI

Celem zadania było wykonanie programu rozmywającego zadane zdjęcie za pomocą algorytmu Gaussa z maską o wymiarach 5x5. Do zrównoleglenia operacji wykorzystany został interfejs MPI, natomiast funkcjonalości potrzebne do pracy z obrazami zapewnione zostały dzięki bibliotece OpenCV.

Do rozmycia została użyta maska o poniższych wartościach:

0	1	2	1	0
1	4	8	4	1
2	8	16	8	2
1	4	8	4	1
0	1	2	1	0

Rysunek 1: Zastosowany filtr wykorzystujący funkcje Gaussa

Dla wczytanego obrazu zostaje dodana ramka o rozmiarze 2 pikseli, które są kopiami pikseli brzegowych. Ma to na celu ułatwienie zastosowania maski dla pikseli na krawędziach obrazka.

Algorytm rozmycia został zaimplementowany podobnie jak w poprzednim zadaniu:

```
    int y, x, red, green, blue;
    for (y = 2; y < inputImage.rows - 2; y++) {
        for (x = 2; x < inputImage.cols - 2; x++) {
            red = 0; green = 0; blue = 0;
            for (int y_m = 0; y_m<5; y_m++) {
                for (int x_m = 0; x_m<5; x_m++) {
                    Vec3b intensity = inputImage.at <Vec3b>(y_m + y - 2,
                        x_m + x - 2);

                    red += intensity.val[2] * mask[x_m][y_m];
                    green += intensity.val[1] * mask[x_m][y_m];
                    blue += intensity.val[0] * mask[x_m][y_m];
                }
            }
            Vec3b masks = Vec3b();
            masks.val[2] = red / weight;
            masks.val[1] = green / weight;
            masks.val[0] = blue / weight;
            outputImage.at<Vec3b>(y - 2, x - 2) = masks;
        }
    }
    return outputImage;
```

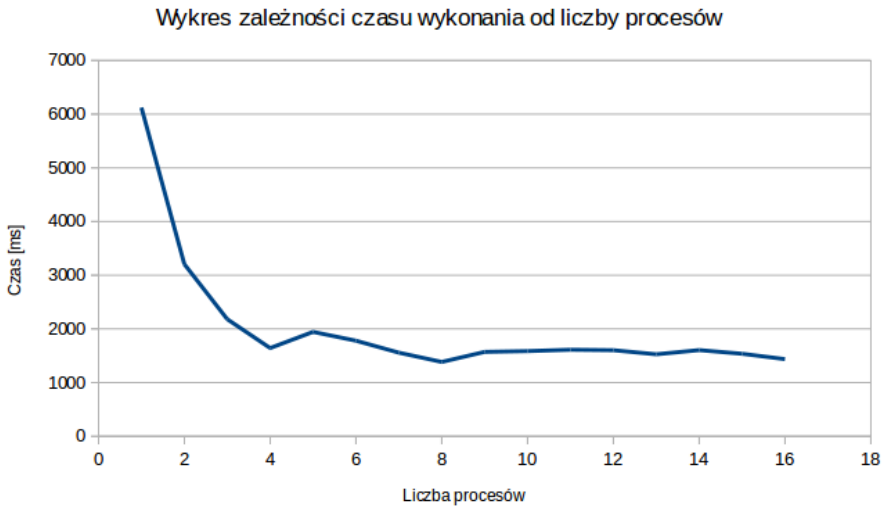
Algorytm dla każdego piksela (z pominięciem ramki) pobiera wartości kolorów składowych, wyznacza nową wartość piksela (na podstawie otaczających go sąsiadów) oraz dokonuje zapisu na nowym obrazie.

W celu uzyskania odpowiednich rezultatów, do wczytanego obrazu dodana zostaje ramka złożona z kopii pikseli brzegowych.

```
copyMakeBorder(inputImage, inputImage, 2, 2, 2, 2, BORDER_REPLICATE);
```

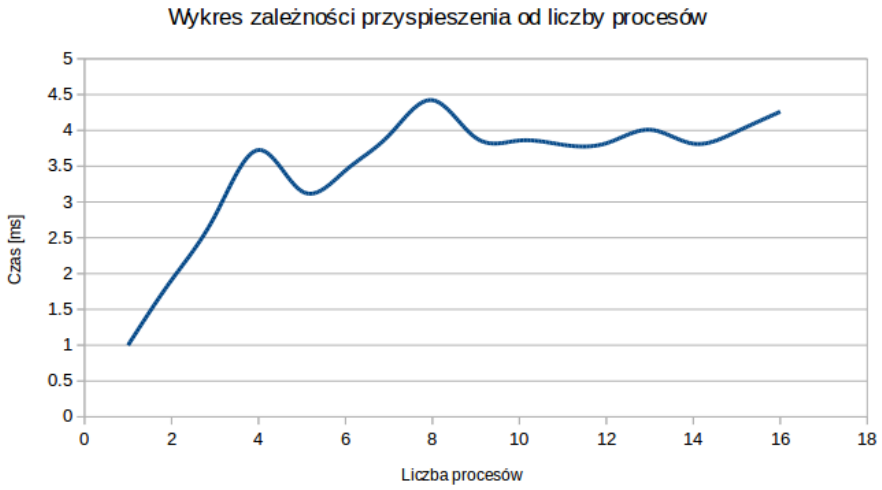
Zrównoleglenie zostało wykonane poprzez podział obrazka w procesie głównym, a następnie rozesłanie po 1 części do każdego z procesów (również dla procesu głównego). Wyodrębnione części obrazka zawierają dodatkowe 2 piksele z lewej i prawej strony, aby możliwe było obliczenie rozmycia dla każdego piksela obrazka. Natomiast 2 dodatkowe piksele u góry i na dole są pikselami wcześniej dodanej ramki.

Poniższy wykres przedstawia zależność czasu wykonywania programu od liczby wykorzysywanych procesów. Badany obraz miał wymiary 4032x2268px. Zrównoleglenie zostało wykonane dla od 1 do 16 procesów.



Rysunek 2: Wykres zależności czasu wykonania od liczby procesów

Na wykresie można zaobserwować, że w czas wykonania malał gdy liczba procesów się zwiększała. Poprawę zanotowano dla pierwszych 4 procesów. Nieco wolniej malał aż do wykorzystania 8 procesów, po osiągnięciu których ustabilizował się. Wynika to z faktu, że największy zysk przyspieszenia uzyskujemy uruchamiając pierwsze dodatkowe rdzenie. Korzystając z większej ilości rdzeni, zysk nie jest już tak znaczący. Potwierdza to wykres przyspieszenia umieszczony poniżej.



Rysunek 3: Wykres zależności przyspieszenia od liczby procesów

Drugi wykres przedstawia zależność przyspieszenia od liczby procesów.Podobnie jak na pierwszym wykresie widać, że przyspieszenie poprawia się do 4 procesów, następnie spada, by później osiągnąć maksymalną wartość dla 8 procesów.

Zadanie zostało w pełni zrealizowane, a otrzymane wyniki są zadowalające. Zrównoleglając operacje przy użyciu interfejsu MPI udało się uzyskać niemalże 4,5 krotne przyspieszenie, co jest satysfakcjonującym wynikiem. Warto zauważyć także nieoczekiwany wzrost czasu wykonania dla wykonywania operacji na 5,6 oraz 7 procesach. Wynika to z faktu, że do dyspozycji mamy 4 rdzenie procesora.