Grupa 3	Albert Soczewski, Maciej Ząbek	14/04/2018

Zadanie 1 - Rozmycie Gaussa w OpenMP

Celem zadania było wykonanie programu rozmywającego zadane zdjęcie za pomocą algorytmu Gaussa z maską o wymiarach 5x5. Do zrównoleglenia operacji wykorzystana została biblioteka OpenMP, natomiast funkcjonalości potrzebne do pracy z obrazami zapewnione zostały dzięki bibliotece OpenCV.
W programie została wykorzystana przedstawiona poniżej maska:

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

Poniżej zaprezentowano algorytm rozmycia. Zmiennymi wspólnymi dla wątków są obrazy (wejściowy oraz wyjściowy) oraz liczba wątków.

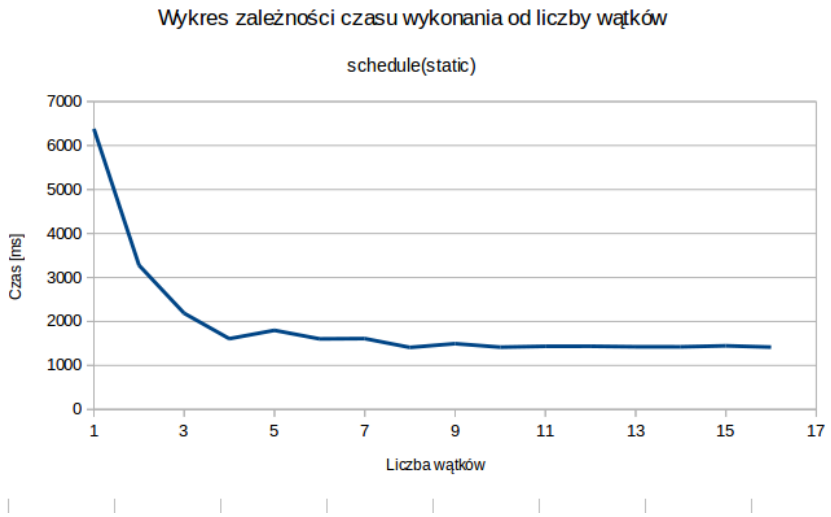
```
#pragma omp parallel for shared(inputImage, outputImage, weight)
    num_threads(threads) private(red,green,blue,y,x) schedule(static)
    for (y = 2; y < inputImage.rows - 2; y++) {
        for (x = 2; x < inputImage.cols - 2; x++) {
            red = 0; green = 0; blue = 0;
            for (int 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;
        }
    }
```

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.

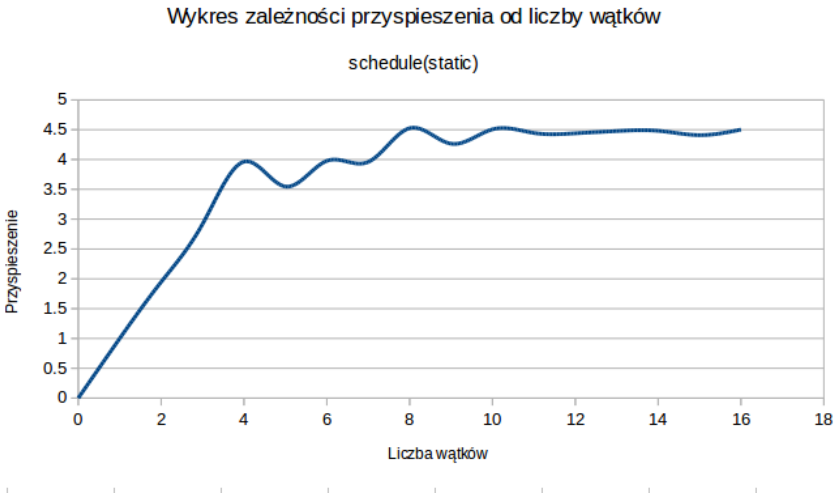
```
copyMakeBorder(inputImage, inputImage, 2, 2, 2, 2, BORDER_REPLICATE);
```

Poniższy wykres przedstawia zależność czasu wykonywania programu od liczby wykorzysywanych wątków. Badany obraz miał wymiary 4032x2268px.



Rysunek 2: Wykres zależności czasu wykonania od liczby wątków

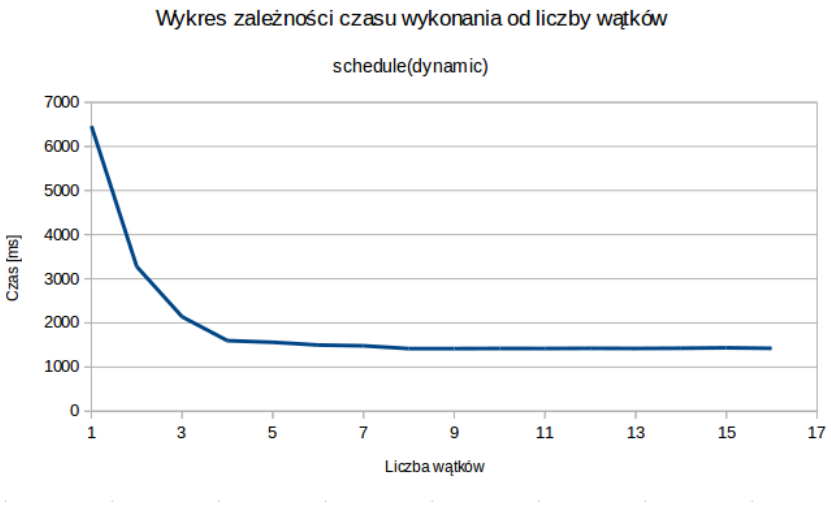
Na wykresie można zaobserwować, że w czas wykonania malał gdy liczba wątków się zwiększała. Najwiekszą poprawę zanotowano dla pierwszych 4 wątków. Nieco wolniej malał aż do wykorzystania 8 wątków, po osiągnięciu których ustabilizował się. Wynika to z faktu, że największy zysk przyspieszenia uzyskujemy uruchamiając pierwsze dodatkowe wątki. Korzystając z wielu wątków procesora, zysk nie jest już tak imponujący. Potwierdza to wykres przyspieszenia umieszczony poniżej.



Rysunek 3: Wykres zależności przyspieszenia od liczby wątków

Zrównoleglając operacje 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 wątkach. Wynika to z faktu, że do dyspozycji mamy 4 rdzenie procesora - 4 wątki. Przy użyciu dyrektywy schedule z argumentem static w wspomnianych wyżej przypadkach wątki oczekują na możliwość przeprowadzenia operacji. Problem ten rozwiązuje użycie dyrektywy dynamic, dzięki której wątki otrzy-

mują kolejną porcje iteracji do wykonania od razu po zakończeniu obliczen. Tę sytuację przedstawia wykres poniżej.



Rysunek 4: Wykres zależności przyspieszenia od liczby wątków