Grupa 1	Kamil Sacha, Konrad Szwedo	24/11/2016

Zadanie 4 - Rozmycie Gaussa w OpenMP

Naszym zadaniem laboratoryjnym było napisanie programu wykonującego rozmycie Gaussa, w oparciu o środowisko do zrównoleglania kodu: OpenMP. Algorytm Gaussa polega na zastąpieniu pikseli obrazu wejściowego, średnią z obszaru otaczającego ten piksel. W naszym przypadku rozpatrywaliśmy maskę wielkości 5×5, czyli do obliczenia musieliśmy znać wszystkie 25 wartości trzech kanałów rozpatrywanego piksela (kanały R - Czerwony, G - Zielony, B - Niebieski). Jako wartość poszczególnego kanału (R, G, B) ustawiamy średnią z całości rozpatrywanego obszaru.

Podział obrazu pomiędzy wątki

Podział wejściowego obrazu jest realizowany automatycznie poprzez zastosowanie poniższej dyrektywy **#pragma omp parallel for...**, biblioteka OpenMP rozdziela ilość iteracji pętli odpowiednio do rozmiaru obrazu wejściowego.

```
1 #pragma omp parallel for private(row, col, i, minus1Row, minus2Row,
2 currentRow, plus1Row, plus2Row) num_threads(threads)
```

Listing kodu 1: Główna dyrektywa OpenMP zewnętrznej pętli iterującej po liczbie wierszy.

Jako zmienne prywatne, czyli te dla których każdy wątek tworzy swoją kopię, zostały ustawione zmienne iteracyjne row, col, i, czyli zmienne oznaczające następująco numer bieżącego wiersza, kolumny, indeks pomocniczy do algorytmu Gaussa. Dodatkowo jako zmienne prywatne oflagowane zostały wskaźniki na 5 wierszy potrzebnych do przyśpieszenia obliczeń, czyli: po dwa wskaźniki na wiersze poprzedzające i następujące oraz wskaźnik na obecny wiersz. Jako obecny wiersz rozumiemy ten wiersz dla którego obliczamy wartość piksela.

Algorytm obliczeń

Korzystając z biblioteki OpenCV wczytujemy obraz wejściowy, który w pamięci przechowywany jest jako tablica składająca się z wartości każdego kanału piksela. Piksel składa się z 3 kanałów zapisywanych w kolejności odpowiednio (R, G, B). Każdy kanał reprezentowany jest jako liczba typu **unsigned char** (w skrócie uchar), czyli liczba typu char bez znaku, która może przechowywać liczby z zakresu [0 - 255]. Aby przyśpieszyć odczyt pikseli obrazu wejściowego skorzystaliśmy z metody **ptr** która zwraca nam wskaźnik na cały wiersz o jaki pytamy. Dzięki temu ograniczyliśmy korzystanie z kosztownej obliczeniowo metody **at** która przelicza za każdym razem wartości wiersza i kolumny na indeks odpowiedniego kanału.

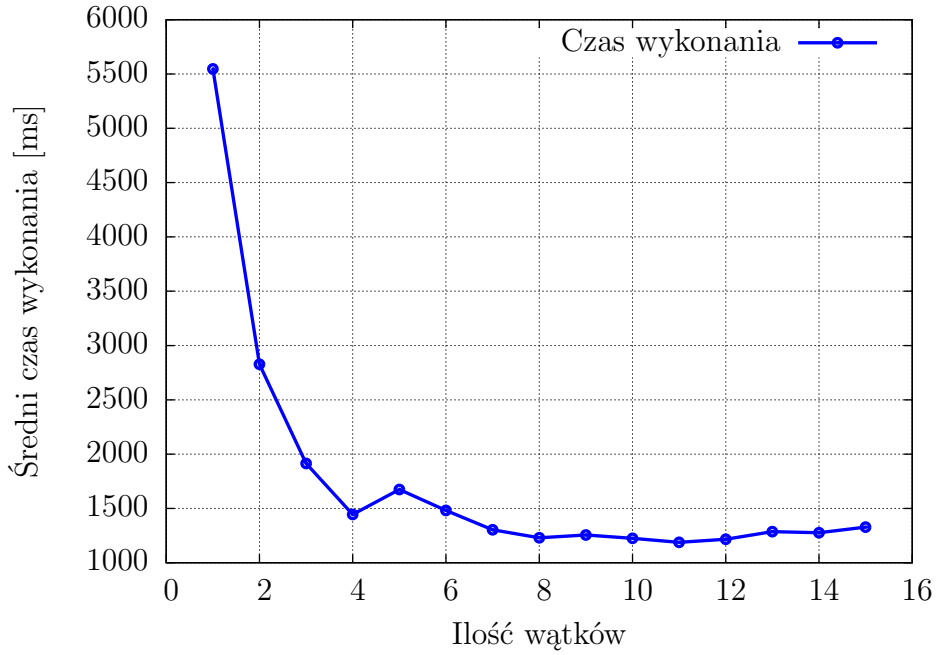
```
minus2Row  = my_input_img->ptr<uchar>( row - 2 );
minus1Row  = my_input_img->ptr<uchar>( row - 1 );
currentRow = my_input_img->ptr<uchar>( row     );
plus1Row   = my_input_img->ptr<uchar>( row + 1 );
plus2Row   = my_input_img->ptr<uchar>( row + 2 );
...
blueTotal  += minus2Row [ blueColIndex ] + minus1Row[ blueColIndex ] +
              currentRow[ blueColIndex ] + plus1Row [ blueColIndex ] +
              plus2Row  [ blueColIndex ];
...
blueTotal  = round(blueTotal  / 25.0f);
...
my_output_image->at<cv::Vec3b>(row - 2, col - 2) = cv::Vec3b(blueTotal,
    greenTotal, redTotal);
```

Listing kodu 2: Fragment wewnętrznej pętli iterującej po wierszach.

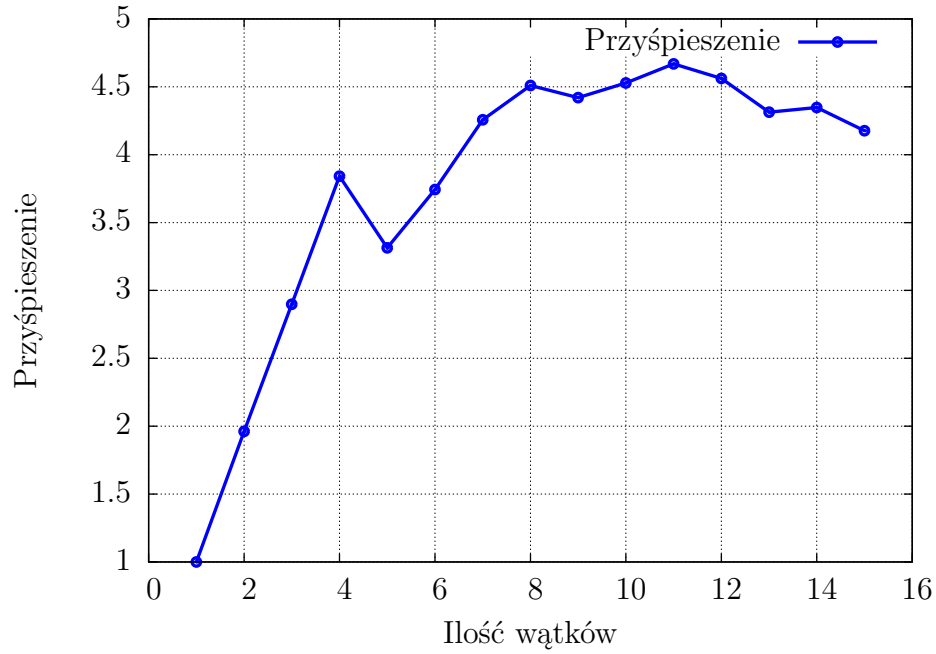
Wyniki i wnioski

Program zrównoleglony został poprawnie o czym świadczą wykresy czasu wykonania oraz przyśpieszenia w zależności od liczby wątków. Testy zostały przeprowadzone na serwerze **cuda.iti.pk.edu.pl**, przy użyciu dostępnego tam 4 rdzeniowego procesora. Istnieje jeszcze szybsza metoda, dostępu do wierszy obrazu wejściowego, a mianowicie bezpośredni dostęp w którym sami zajmujemy się indeksowaniem wskaźników. Według znalezionych informacji [1] wynika, że bezpośredni dostęp poprzez wskaźniki jest szybszy od zastosowanej przez nas metody o 18%. Postanowiliśmy jednak zostawić program w postaci operowania na całych wierszach, gdyż ważniejsze było bezpieczeństwo oraz czytelność kodu.

[1] Which way of accessing pixels in OpenCV is the fastest?



Rysunek 1: Wykres zależności czasu wykonania od ilości wątków. Dla obrazu wejściowego '1.jpg' o wymiarach $24107 \times 4491 \approx 10^9$ pikseli.



Rysunek 2: Wykres zależności przyśpieszenia od ilości wątków. Dla obrazu wejściowego '1.jpg' o wymiarach $24107 \times 4491 \approx 10^9$ pikseli.