Grupa 1	Kordian Kurdziel, Mateusz Maciejak	17/10/2018

Zadanie 1 - Rozmycie Gaussa w OpenMP

Zadanie programu było rozmycie obrazu podanego na wejściu za pomocą algorytmu Gaussa z maską 5x5. W celu poprawy wydajności programu do zrównoleglenia jego działania należało wykorzystać OpenMP.

Poniższy fragment kodu w pierwszej pętli for jest wykonywany równolegle w oddzielnych wątkach, dzięki zastosowaniu dyrektywy pragma omp parallel for. W programie użyto dyrektywę schedule z opcją static. Oznacza ona, że każdemu wątkowi zostanie przypisana taka sama ilość wykonywanych zadań. Zadania te powinny wykonywać się w tym samym czasie, więc taki przydział wydaje się tu odpowiedni - nie będzie stwarzał dodatkowych narzutów związanych z dynamicznym przydziałem zadań. Wartość każdego kanału RGB jest liczona oddzielnie w funkcji calculateNewPixelChannelValue()

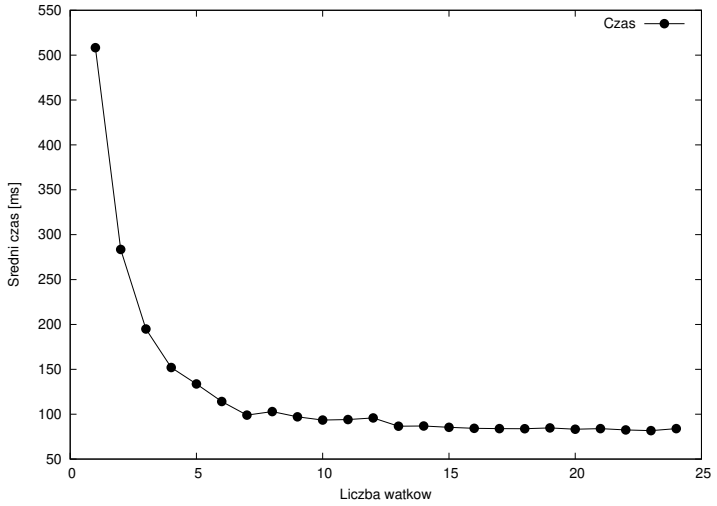
```
1 #pragma omp parallel for default(shared) private(i,j) schedule(static)
      num_threads(threadsNumber)
2 for (i = margin; i < inputImg.rows - margin; i++) {
3     for (j = margin; j < inputImg.cols - margin; j++) {
4         rgbOutputChannels[0].at<uchar>(i,j) = calculateNewPixelChannelValue(
              rgbInputChannels[0], i, j);
5         rgbOutputChannels[1].at<uchar>(i,j) = calculateNewPixelChannelValue(
              rgbInputChannels[1], i, j);
6         rgbOutputChannels[2].at<uchar>(i,j) = calculateNewPixelChannelValue(
              rgbInputChannels[2], i, j);
7     }
8 }
```

Funkcja wylicza wartość dla każdego kanału na podstawie wagi poszczególnych pikseli maski oraz wartości tych pikseli

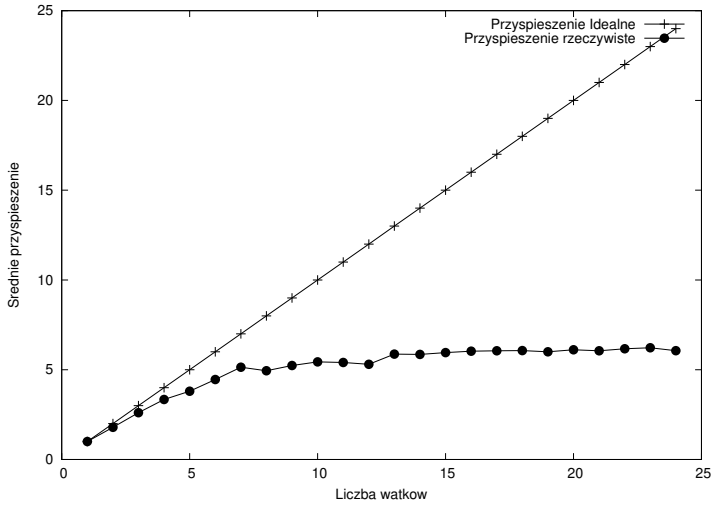
```
1 int calculateNewPixelChannelValue(Mat channel, int row, int col) {
2     int sum = 0;
3     for (int i = 0; i < maskSize; ++i) {
4         for (int j = 0; j < maskSize; ++j) {
5             sum+= mask[i][j] * ((int) channel.at<uchar>(row + i - 2, col +
                  j - 2));
6         }
7     }
8     return (int) (sum / maskWeight);
9 }
```

Poniższe wykresy 1 i 2, przedstawiające zależność czasową oraz przyspieszenia zostały oparte na średnich wynikach programu uruchamianych lokalnie na maszynie wirtualnej. Wykorzystany został 6 rdzeniowy procesor Intel z technologią Hyperthreadingu.

Jak widać dzięki zastosowaniu technologii OpenMP, wykorzystując dostępne wątki, udało się znacząco przyspieszyć wykonywanie programu. Uzyskane rzeczywiste przyspieszenie jest bliskie idealnemu, co może świadczyć o tym, że powyższa klasa problemów nadaje się całkiem dobrze do wykorzystywania obliczeń wielowątkowych. Jak można zauważyć na wykresie przyspieszenie działania programu wzrastało wraz z liczbą wykorzystywanych wątków jedynie do 6 wątku. Potem wraz ze wzrostem liczby wątków program nie wykazywał już aż tak dużego przyspieszenia, a nawet potrafił minimalnie zwolnić. Można więc wyciągnąć wniosek, że technologia Hyperthreadingu, wykorzystywana w pro-



Rysunek 1: Wykres zależności czasu wykonywania obliczeń od liczby wątków



Rysunek 2: Wykres przyspieszenia działania programu w zależności od liczby wątków

cesorach Intela, nie działa aż tak wydajnie przy takich obliczeniach. Do takiego zrównoleglania niewątpliwie lepszy byłby procesor z większą ilością rdzeni fizycznych.