Grupa 3	Albert Soczewski, Maciej Ząbek	24/05/2018

Zadanie 3 - Rozmycie Gaussa w CUDA

Celem zadania było wykonanie programu rozmywającego zadane zdjęcie za pomocą algorytmu Gaussa z maską o wymiarach 5x5. Do zrównoleglenia operacji wykorzystana została technologia CUDA, 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:

```
__global__ void gaussianBlur(uchar * inputImage, uchar * outputImage, long
    width, long height)
{
    long x = (blockIdx.x * blockDim.x) + threadIdx.x;
    long y = (blockIdx.y * blockDim.y) + threadIdx.y;

    if (x < width-2 && y < height-2 && x>1 && y>1)
    {
        long r=0, g=0, b=0;
        long pixelIn, pixelOut;

        for (int y_m = 0; y_m<5; y_m++)
        {
            for (int x_m = 0; x_m<5; x_m++)
            {
                pixelIn = width*(y + y_m - 2) * 3 + (x + x_m - 2) * 3;

                r += inputImage[pixelIn + 2] * dev_mask[x_m][y_m];
                g += inputImage[pixelIn + 1] * dev_mask[x_m][y_m];
                b += inputImage[pixelIn] * dev_mask[x_m][y_m];
            }
        }
        pixelOut = (width - 4)*(y - 2) * 3 + (x - 2) * 3;
        outputImage[pixelOut + 2] = r / dev_weight;
        outputImage[pixelOut + 1] = g / dev_weight;
        outputImage[pixelOut] = b / dev_weight;
    }
}
```

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);
```

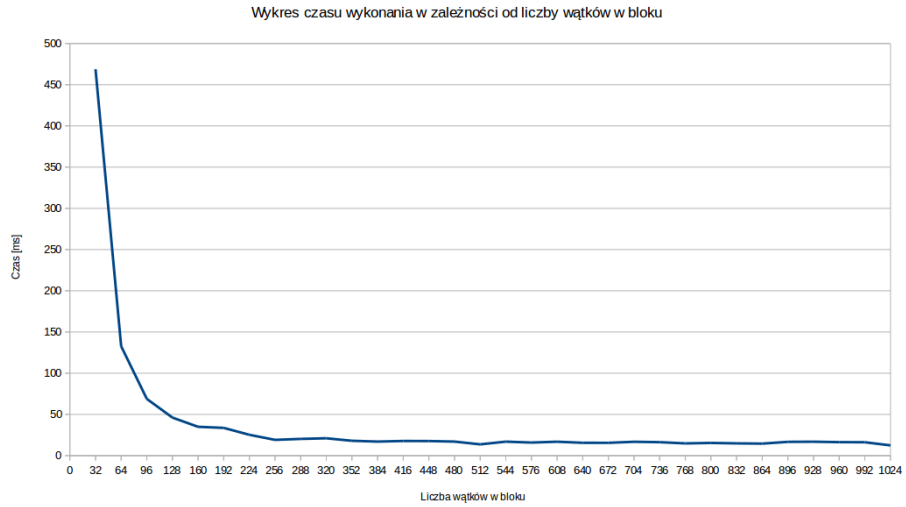
Stosując się do wskazówek autorów książki 'CUDA w przykładach' wybraliśmy rozwiązanie problemu polegające na podzieleniu obrazu na tyle wątków z ilu składa się pikseli. W tym celu zdefiniowaliśmy najpierw dwie dwuwymiarowe zmienne o nazwach gridSize oraz threadsPerBlock. Pierwsza reprezentuje liczbę bloków, a druga - liczbę wątków na blok. Dzięki temu każdy wątek posiada unikalny indeks (x,y) dzięki któremu jest możemy powiązać go z pixelem w obrazie wejsciowym. Po przeprowadzeniu testów uznaliśmy, że najbardziej optymalny rozmiar bloku wynosi 32 x 32(1024 wątki per blok). Aby przy takim rozmiarze bloku otrzymać po jednym wątku na piksel należy uruchomić odpowiednią ilość bloków. Cały proces przedstawiony jest na listingu poniżej.

```
1   int gridWidth, gridHeight;
2    gridWidth = inputImage.cols / blockSize + ((inputImage.cols % blockSize
        ) == 0 ? 0 : 1);
3    gridHeight = inputImage.rows / blockSize + ((inputImage.rows %
        blockSize) == 0 ? 0 : 1);
4
5    dim3 gridSize(gridWidth, gridHeight);
6    dim3 threadsPerBlock(blockSize, blockSize);
```

Po deklaracjach zmiennych do przechowywania wymiarów uruchamiamy jądro które będzie obliczało wartości pikseli po zastosowaniu rozmycia Gaussa.

```
1    gaussianBlur <<< gridSize,threadsPerBlock >>> (dev_inputImage,
        dev_outputImage, inputImage.cols, inputImage.rows);
```

Poniższy wykres przedstawia zależność czasu wykonywania programu od liczby wątków w bloku. Badany obraz miał wymiary 4032x2268px. Zrównoleglenie zostało wykonane dla rozmiaru bloku od 1 do 32. Wynika to z faktu, że maksymalna liczba wątków w bloku to 1024 (32x32), dlatego w celu znalezienia optymalnej liczby przebadaliśmy wszystkie możliwe konfiguracje siatki oraz liczby wątków w bloku zachowując sumaryczną ilość wątków równą liczbie pikseli w obrazie.



Rysunek 2: Wykres czasu wykonania w zależności od liczby wątków w bloku

Na wykresie można zaobserwować, że w czas wykonania malał gdy liczba wątków się zwiększała. Największy zysk jest zauważalny do uruchomienia około 256 wątków per blok. Po osiągnięciu tego progu czas ustabilizował się. Najmniejszy czas operacji zanotowano dla 1024 wątków.

Zadanie zostało zrealizowane, a otrzymane wyniki są zadowalające. Zrównoleglając operacje przy technologii CUDA udało się uzyskać satysfakcjonującą poprawe czasu wykonania programu. Warto zauważyć że uzyskane przyspieszenie jest większe niż w przypadku zrównoleglania przy pomocy biblioteki OpenMP a także interfejsu MPI.