Grupa 6	Dariusz Szczupak, Kamil Wanat	25/10/2018

Zadanie 2 - Rozmycie Gaussa w MPI

Celem zadania jest program oparty na algorytmie Gaussa mający na celu rozmycie danego zdjęcia. Zadanie wykonane z wykorzstaniem standardu MPI. "Gauss 2- to użyta wersja filtru danego w dołączonym do treści zadania linku. Maska w danym filtrze wygląda następująco:

```
int mask[5][5] = {{1,1,2,1,1}, {1,2,4,2,1}, {2,4,8,4,2}, {1,2,4,2,1},
    {1,1,2,1,1}
};
```

Do obliczenia wartości z których składa się każdy piksel(Red, Green, Blue), potrzebne są punkty otaczające go. Każdy taki piksel ma jakąś wage, która z kolei zapisana jest w masce filtra. Liczymy sume ważoną kolejnych pixeli i jego sąsiadów, kolejnie dzielimy taką sumę przez sumę całej maski. Filtrowanie obrazu następuje oddzielnie dla każdej wartości z której składa się pixel. W metodzie filtrowania Gaussa znaczenie wartości pixela jest największe w punkcie, który jest liczony i zmniejsza się wraz z oddalaniem się od niego. Poniżej przesyłanie czesci danych pomiedzy procesami:

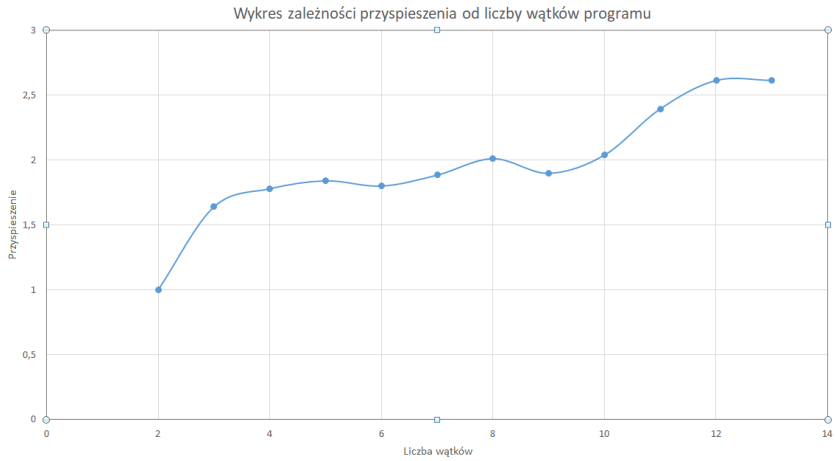
```
comm = MPI_COMM_WORLD;
    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(comm, &size);
    if (rank == 0) {
for (int i = 1; i < size; i++) {
            if (i == (size - 1))
                width = img.cols - start;
            slice = Mat(width, img.rows, CV_8UC3);
            slice = img(Rect(start, 0, width, img.rows)).clone();
            start = end - 4;
            end = start + width;

            int sliceColumn = slice.cols;
            int sliceRows = slice.rows;

            MPI_Send(&sliceColumn, 1, MPI_INT, i, 0, comm);
            MPI_Send(&sliceRows, 1, MPI_INT, i, 1, comm);
            MPI_Send(slice.data, sliceColumn * sliceRows * 3, MPI_BYTE, i,
                2, comm);
        }

for (int i = 1; i < size; i++) {
            int tempcols, temprows;
            MPI_Recv(&tempcols, 1, MPI_INT, i, 0, comm, MPI_STATUS_IGNORE);
            MPI_Recv(&temprows, 1, MPI_INT, i, 1, comm, MPI_STATUS_IGNORE);
            Mat tempimg = Mat(temprows, tempcols, CV_8UC3);
            MPI_Recv(tempimg.data, tempcols * temprows * 3, MPI_BYTE, i, 2,
                comm, MPI_STATUS_IGNORE);
            if (i == 1) {
                imgScore = tempimg.clone();
            } else {
                hconcat(imgScore, tempimg, imgScore);
            }
          }
        }
```

Na wstępie proces macierzysty (rank == 0) dzieli obrazek na paski, a następnie rozsyła je równomiernie do wszystkich pozostałych procesów. Po przetworzeniu dane w postaci pasów pikseli zostają poskładane w nowy obraz. Rozwiązanie to pozwala na maksymalne wykorzystanie dostepnych procesów.

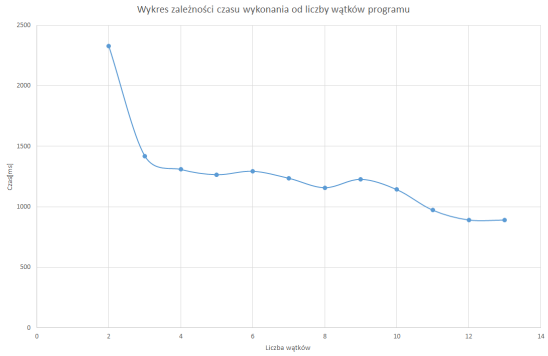


Rysunek 1: Wykres przyspieszenia

Z powyższych wykresów można wywnioskować że czas obliczeń spada gwałtownie miedzy 2 a 3 procesem po czym następuje stabilizacja.

Dane przeprowadzanych testów:

- Wzięto pod uwagę średnią pomiarów czasów wykonania programu dla liczby wątków z przedziału od 2 do 13. Program był uruchamiany dla obrazka ściągniętego z przestworzy internetów w rozdzielczośći 1920x1200px.
- Do mierzenia czasu wykorzystano MPIWtime.
- Do wczytania obrazu wykorzystano bibliotekę OpenCV .
- Testy zostały wykonane na serwerze cuda.iti.pk.edu.pl .



Rysunek 2: Wykres zależności czasu od liczby wątków

Zapoznaliśmy się z protokołem MPI. Zadanie udało się wykonać w całości. Wniosek nasuwa się taki, iż za pomocą MPI można przyspieszyć czas obliczeń algorytmu rozmycia Gaussa. Najlepsze wyniki osiągnięto dla ilości wątków zbliżonej do ilości rdzeni procesora na serwerze CUDA.