Grupa 4	Barbara Nowak, Piotr Tomaszewski	18/01/2017

Zadanie 6 - Rozmycie Gaussa w CUDA

Naszym zadaniem było zaimplementowanie programu, który rozmyje zadane zdjęcie (podobnie jak w zadaniach 4 i 5) ale stosując kartę graficzną i technologię CUDA.

Program uruchamiany jest z dwoma argumentami: pierwszy - to wybrany obrazek, a drugi - miejsce do zapisu zmodyfikowanego obrazka. Sprawdzamy czy program uruchamiany jest z wymaganą liczą argumentów wejściowych. Następnie deklarujemy maskę w postaci 2-wymiarowej tablicy liczb całkowitych oraz za pomocą funkcji imread(), wczytujemy wybrany plik do zmiennej zdjecie, a dzięki parametrowi CVxLOADxIMAGExCOLOR konwertujemy obrazek do wersji kolorowej. Kolejnym krokiem jest podział zadania na gridy(ilość bloków) oraz bloki(ilość wątków), który wyliczamy dzieląc odpowiednio kolumny i wiersze siatki zdjęcia przez ustalony rozmiar bloku równy 32. Następnie deklarujemy krotki dim3 do określenia wyliczonej wcześniej siatki oraz bloków.

```
1  int rozmiarBlok = 32;
2  int rozmiarSiatkaSzer = zdj_we.cols/rozmiarBlok;
3  int rozmiarSiatkaDlug = zdj_we.rows/rozmiarBlok;
4  dim3 siatka(rozmiarSiatkaSzer, rozmiarSiatkaDlug);
5  dim3 blok(rozmiarBlok, rozmiarBlok);
```

Tworzymy kopię wcześniej wczytanego zdjęcia do zmiennej zdjxwy. Alokujemy pamięć na karcie graficznej dla wczytanego zdjęcia oraz dla jego kopii:

```
1  cudaMalloc(&zdj_gpu_we, zdj_we.rows*zdj_we.step*sizeof(unsigned char));
2  cudaMalloc(&zdj_gpu_wy, zdj_we.rows*zdj_we.step*sizeof(unsigned char));
```

Kolejnym krokiem jest kopiowanie danych między kartą graficzną pamięcią RAM, gdzie obszar pamięci źródłowej należy do komputera (RAM), natomiast docelowy obszar pamięci należy do pamięci karty graficznej.

```
1  cudaMemcpy(zdj_gpu_we, zdj_we.ptr(), zdj_we.rows*zdj_we.step,
       cudaMemcpyHostToDevice);
```

Następnie rozpoczynamy pomiar czasu obliczeń pamiętając o użyciu cudaDeviceSynchronize(), która blokuje bieżący wątek aplikacji do czasu zakończenia wszystkich oczekiwanych obliczeń na karcie graficznej.

```
1  cudaEventRecord(czas_start, 0);
2  rozmycie<<<siatka,blok>>>(zdj_we.channels(), sumaMaski(maska,0,0,0),
       zdj_gpu_we, zdj_gpu_wy, zdj_we.rows, zdj_we.step);
3  cudaEventRecord(czas_stop, 0);
4  cudaEventSynchronize(czas_stop);
5  float czas = 0.0;
6  cudaEventElapsedTime(&czas, czas_start, czas_stop);
7  cudaDeviceSynchronize();
```

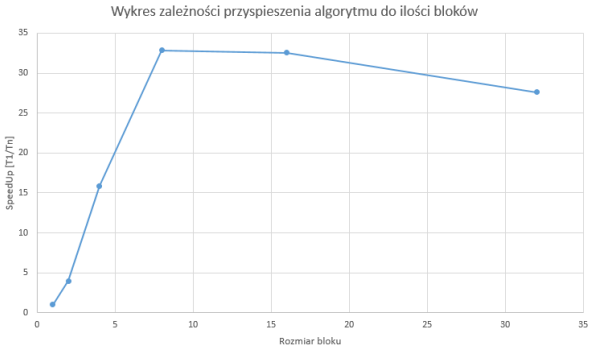
Ponownie kopiujemy dane między kartą graficzną a pamięcią RAM, jednak teraz obszar pamięci źródłowej należy do pamięci karty graficznej, natomiast docelowy obszar pamięci należy do komputera (RAM).

```
1  cudaMemcpy(zdj_wy.ptr(), zdj_gpu_wy, zdj_we.rows*zdj_we.step,
       cudaMemcpyDeviceToHost);
```

Na koniec pamiętamy o zwolnieniu pamięci wcześniej zaalokowanej na karcie graficznej, za pomocą funkcji cudaFree() oraz zapisujemy zmodyfikowany obrazek w miejsce zadane jako drugi argument.

Na powyższym rysunku (Rysunek1) widzimy, że na przedziale 1 - 8 bloków czas obliczeńalgorytmu wraz ze wzrostem liczby bloków zmniejsza się parabolicznie aż do osiągnięcia stabilnego poziomu. Wykres zależności przyspieszenia algorytmu do ilości bloków obrazuje, że na przedziale 1 - 8 bloków CUDA ma bardzo duże przyspieszenie, następnie lekko się stabilizuje by na koniec nieco zwolnić.

Rysunek 1: Wykres zależności czasów obliczeń do ilości bloków



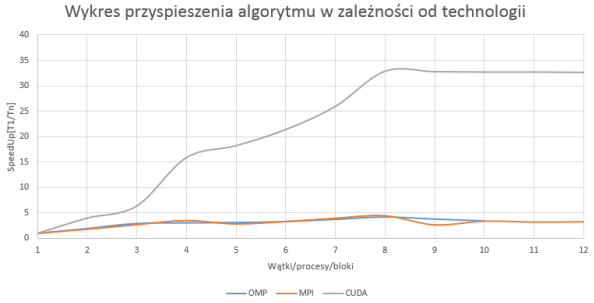
Rysunek 2: Wykres przyspieszenia

Powyższa tabela przedstawia porównanie czasów przyspieszenia algorytmu do ilości bloków, dla trzech technologii: OpenMP, MPI oraz CUDA. Zarówno z wyników w tabelce jak i na wykresie widzimy, że technologia CUDA ma bardzo duże przyspieszenie i osiąga najlepszy wynik w porównaniu do dwóch pozostałych technologii, a więc jest najbardziej

Nr	1	2	3	4	5	6	7	8	9	10	11	12	16
OMP	429,9453	223,969	148,7227	143,0743	139,8773	132,031	116,9497	104,237	115,3907	126,9913			
MPI	371,453	204,835	137,522	105,645	130,748	111,736	93,774	83,296	139,421	109,367	115,464	113,661	
CUDA	1619,1	405,64	253,82	102	88,815	75,63	62,445	49,26	49,375	49,49	49,49	49,605	49,72

Rysunek 3: Tabela czasów dla technologii OpenMP, MPI i CUDA

Podsumowanie: CUDA (ang. Compute Unified Device Architecture) jest uniwersalną architekturą wielordzeniowych procesorów (głównie kart graficznych). Umożliwia wykorzystanie mocy obliczeniowej karty graficznej do rozwiązania problemów numerycznych w sposób wydajnieszy niż w tradycyjnych, sekwencyjnych procesorach ogólnego zastosowania. Jednak zestawiając CUDA z dwoma pozostałymi



Rysunek 4: Wykres przyspieszenia

technologiami MPI oraz OpenMP, czasowo wypada o wiele lepiej co świadczy o tym, że w przypadku rozmywania obrazka jest ona najbardziej wydajna. Algorytm Gaussa pozwolił nam w prosty i łatwy sposób dokonać rozmycia zadanej fotografii, a dzięki wcześniejszym laboratoriom z technologii CUDA, zastosowanie jej w niniejszym zadaniu nie stanowiło już dla nas większego problemu.