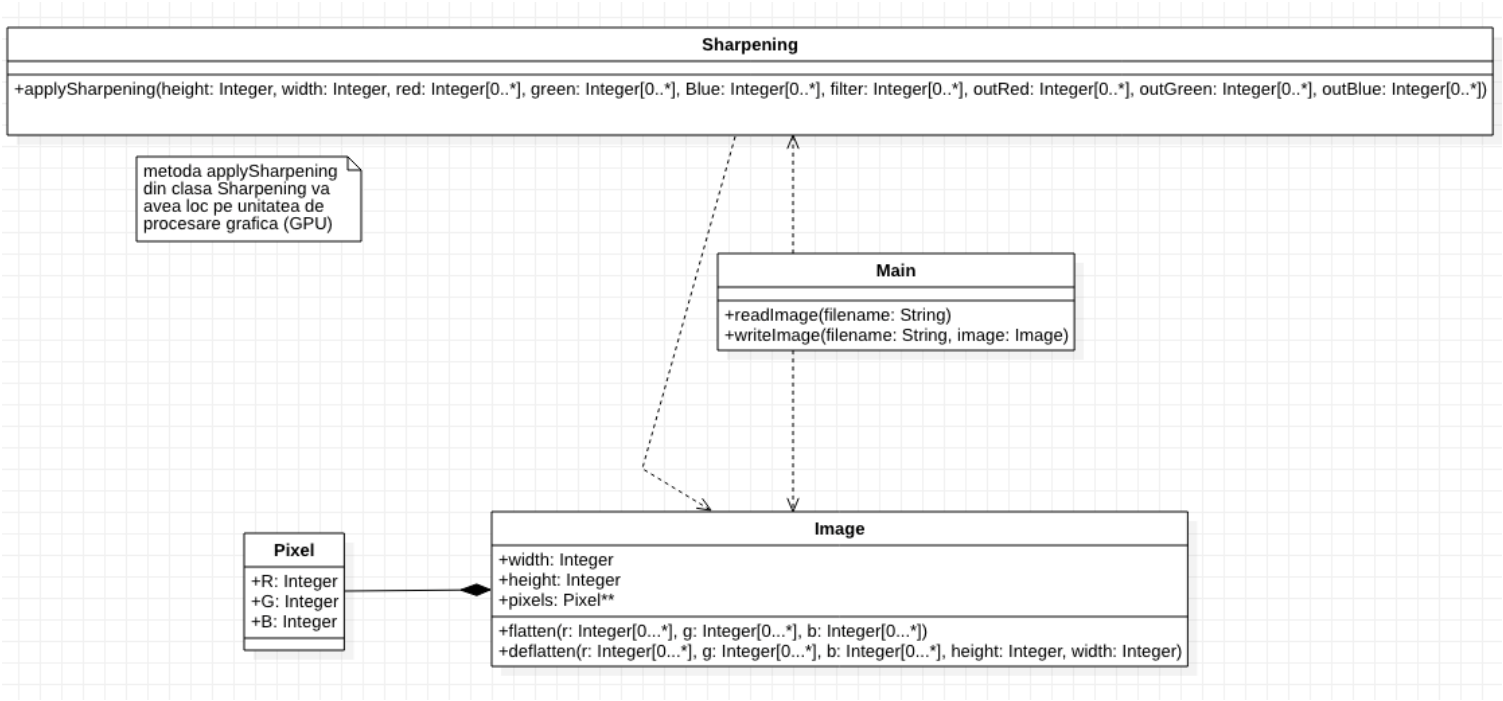
Picus Oskar, Popovici Denisa

P2 – DOCUMENTATIE

1. ANALIZA

Avand ca date de intrare o imagine sub forma unui fisier cu extensie .ppm, versiunea P3, se doreste o accentuare a trasaturilor si calitatii imaginii, printr-un proces de “sharpening” al acesteia. Dupa prelucrare, se asteapta ca imaginea sa fie mai bine conturata, iar detaliile ei mai bine evidentiate. Prelucrarea in sine va fi realizata asupra fiecarui pixel al imaginii in parte. Rezultatul final va fi reprezentat de un alt fisier .ppm, corespunzator imaginii obtinute.

2. PROIECTARE



3. DETALII DE IMPLEMENTARE

Vom utiliza doua struct-uri menite sa ajute la memorarea datelor corespunzatoare unei imagini: Pixel si Image. Structura unui pixel este urmatoarea: 3 valori intregi, R, G, B, reprezentand valorile concentratiilor culorilor rosu, verde si albastru in pixelul nostru. Structura Image va fi corespundenta noastra “in cod” a unei imagini si va contine latimea si inaltimea imaginii, valori intregi, precum si o matrice de pixeli, intocmai unei imagini reale.

Pentru a transpune o imagine cu extensia .ppm, versiunea P3, in entitatea Image, vom utiliza functia de readImage, care primeste drept parametru calea catre fisierul care contine imaginea. Formatul unui astfel de fisier .ppm este urmatorul: primele 2 linii ale sale contin 2 headere care pot fi ignorate, cea de-a treia linie ne ofera latimea si inaltimea imaginii, cea de-a patra linie va avea intotdeauna valoarea 255, valoarea maxima a R,G,B, iar urmatoarele linii vor avea succesiv valorile R, G, B pentru fiecare pixel al imaginii.

*Parcurgerea pixelilor este pe linii. Astfel, obtinand mai intai inaltimea si latimea, putem crea pe rand cate un Pixel, citind mereu cele 3 valori, si punandu-l intr-o matrice de dimensiune latime*inaltime.*

*Totusi, pentru o mai buna transmitere a datelor intre unitatea centrala de prelucrare (CPU/host) si unitatea de procesare grafica (GPU/device), va trebui ca noua matrice obtinuta sa fie prelucrata in 3 vectori distincti care sa contina valorile R, G, B pentru pixelii imaginii, corespunzatori matricii liniarizate. Acesti 3 vectori se vor declara pe partea de CPU si vor purta nume sugestive: h_red, h_green, h_blue. Ei vor fi transmisi functiei de flatten care ii va popula in felul urmator: se realizeaza o parcurgere a matricii pe linii, stiind ca pozitia [linie][coloana] din matrice corespunde pozitiei linie*latime_imagine+coloana din vectorii doriti. Avand pozitia la care se face inserarea, luam valorile R, G, B din pixelul curent si le adaugam. In final, vom obtine cei 3 vectori de dimensiune latime_imagine*lungime_imagine.*

Pentru partea de GPU, vom declara alti 3 vectori, d_red, d_green, d_blue, folosind functia cudaMalloc, care ne aloca memorie direct pe device. Folosind cudaMemcpy, vom copia de pe partea de host (CPU) pe partea de device (GPU) vectorii h_red, h_green, h_blue in vectorii d_red, d_green, d_blue. Nu uitam sa dam ca parametru functiei argumentul cudaMemcpyHostToDevice, mentionand ca facem o copiere de pe CPU pe GPU. Pe langa acesti 3 vectori pe GPU, vom mai avea nevoie de alti 3 pentru a retine rezultatele obtinute dupa prelucrarea pixelilor si de unul care sa memoreze matricea kernel liniarizata care va fi folosita in formulele matematice. Acesti vectori se vor numi d_outRed, d_outGreen, d_outBlue, d_filter si vor fi alocati folosind cudaMalloc. In d_filter vom copia matricea filter care a fost declarata anterior pe CPU. Aceasta ia valorile [[0, -1, 0], [-1, 5, -1], [0, -1, 0]]si corespunde procesului de sharpening.

*A venit timpul sa ne declaram blocurile de thread-uri care se vor ocupa de procesarea imaginii, fiecare bloc ocupandu-se de cate un pixel. Astfel, vom declara nr_pixeli=latime_imagine*inaltime_imagine blocuri de thread-uri, fiecare bloc avand o dimensiune prestabilita de 24x24.*

Pentru procesarea propriu-zisa se va apela functia de applySharpening, care primeste drept parametrii latimea si inaltimea imaginii, cei 3 vectori cu valorile R, G, B d_red, d_green, d_blue, vectorul corespunzator kernel-ului d_filter si vectorii de output d_outRed, d_outGreen, d_outBlue. Functia va lua indicii pixelului pe care dorim sa il prelucram in matricea de pixeli a imaginii tinand cont de indicii blocului de thread-uri in care ne aflam si ai thread-ului curent. Verificam ca indicii obtinuti sa se afle in intervalele impuse de dimensiunile imaginii. Vom crea o bordura imaginara a pixelului curent, pentru a obtine o matrice ale carei dimensiuni sa fie egale cu cele ale matricii kernel. Fiecare element din matricea kernel va trebui inmultit cu elementul corespunzator din matricea obtinuta in urma bordarii pixelului, iar aceste produse vor fi adunate. Daca elementul din matricea bordata este in afara matricii corespunzatoare imaginii, vom utiliza elementele care se afla chiar la margine. Suma obtinuta va reprezenta noua valoare a pixelului. Cum noi nu avem un pixel propriu-zis, ci doar vectorii cu valorile R, G, B, acesti pasi se vor realiza pentru fiecare dintre cele 3, obtinand 3 valori noi, pe care le punem in vectorii de output.

Dupa apelul functiei de applySharpening, vom apela cudaDeviceSynchronize, care ii spune gazdei sa astepte pana in momentul in care device-ul isi termina de executat toate thread-urile si sa nu treaca mai departe la celelalte linii de cod. Dupa obtinerea rezultatului, il copiem pe CPU, din d_outRed, d_outGreen, d_outBlue in h_red, h_green,

h_blue, de data aceasta de pe device pe host, folosind cudaMemcpy cu argumentul cudaMemcpyDeviceToHost.

In acest moment avem cei 3 vectori R, G, B corespunzatori pixelilor prelucrati. Dorim sa transpunem acesti vectori inapoi intr-o structura de tip Image, motiv pentru care vom apela functia de deflatten, care primeste latimea si inaltimea unei imagini, respectiv cei 3 vectori si returneaza un obiect de tip Image. Aceasta functioneaza in aceeasi maniera precum cea de flatten, tinand cont de aceeasi corespondenta intre indici.
In final, apeland functia de writeImage, la calea aleasa de noi si data ca parametru vom scrie un fisier in format .ppm tinand cont de imaginea obtinuta. Dupa inserarea antetelor si a dimensiunilor, vom insera pe rand cate o valoare R, G, B pentru fiecare pixel. Astfel, rezultatul final va fi reprezentat tot de un fisier .ppm, care contine imaginea mai bine conturata. Nu uitam sa eliberam memoria alocata pe partea de GPU, folosind cudaFree, si pe partea de CPU, folosind functia free.

4. TESTARE

Latime imagine	Inaltime imagine	Timp de executie
640	426	184.09340000000003
800	600	281.7232
801	1200	459.6361

5. ANALIZA REZULTATELOR

Pentru fiecare set de date au fost executate 10 rulari, rezulatul final reprezentand o medie aritmetica a timpilor de executie obtinuti. Toate testele au fost executate pe cluster-ul facultatii, nodul compute065. Pentru fiecare pixel al imaginii procesate a fost atribuit un bloc de thread-uri cu dimensiuni 24x24, deoarece se doreste o modificare individuala la nivelul fiecaruia. Astfel, vom avea in total atatea blocuri de thread-uri cat numarul total de pixeli ai imaginii (latime imagine x inaltime imagine). Observam, bineinteles, ca dimensiunile imaginii influenteaza puternic timpul de procesare al acesteia. O imagine mai mare va aduce in mod asteptat si timpi mai mari, tocmai din cauza acestei procesari la nivel de pixel si a direct proportionalitatii dintre numarul de pixeli si numarul de thread-uri care se creeaza.