Tema 4 – ASC
CUDA: 2D Convolution

1. Introducere

Cerinţa temei este să implementaţi eficient Blocked Matrix Convolution, un algoritm folosit spre exemplu în procesarea de imagini pentru aplicarea de filtre precum blur şi corectarea pozelor. Veţi aplica prin convoluţie o matrice de dimensiune constantă (5x5) denumită “convolution kernel matrix” (matricea kernel A), asupra unor imagini de mărime arbitrară (matricea B).

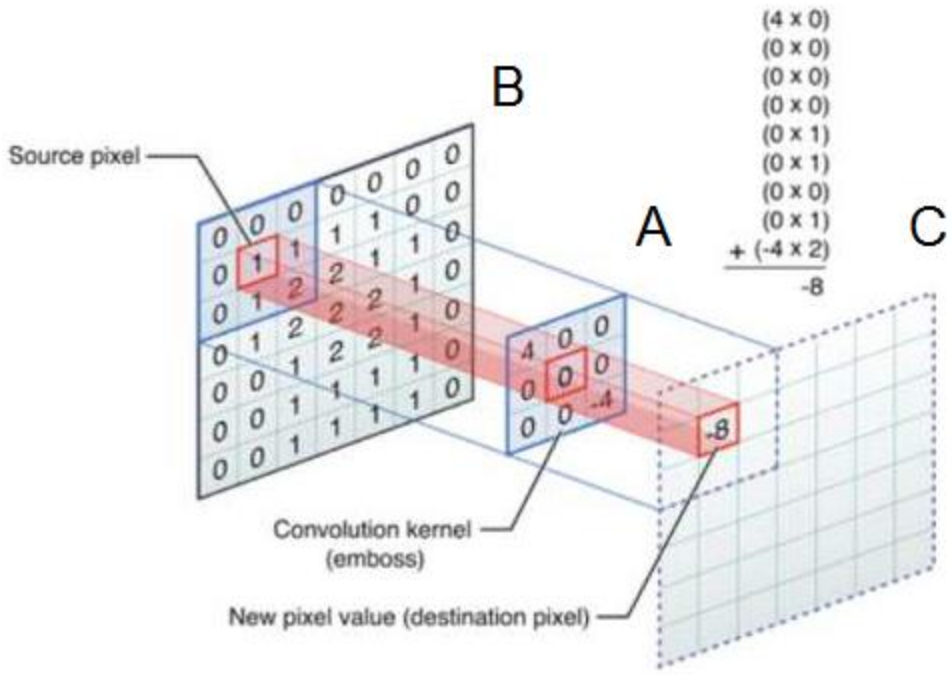
Aplicarea unei matrice kernel A de dimensiune 5x5 asupra unei matrice B constă de fapt în calculul fiecărui element din matricea rezultat C după această formulă:

$$\mathbf{C}[i][j] = \sum_{m=0}^{4} \sum_{n=0}^{4} \mathbf{A}[m][n] \cdot \mathbf{B}[i+m-2][j+n-2]$$

unde 0 ≤ i < B.height si 0 ≤ j < B.width.

2. Exemplu

Iată un exemplu de aplicare a unei matrice kernel A de 3x3 asupra unei matrice B de 7x7 (matricea kernel din figură are ca efect realizarea filtrului emboss). Funcţionarea cu o matrice kernel de 5x5 se face analog:



https://wiki.engr.illinois.edu/display/ece190sp12/Machine+Problem+4

Vizual, operaţia de convoluţie se referă la suprapunerea elementului din centrul matricei kernel peste fiecare dintre elementele matricei B pentru a obţine elementul corespunzător matricei C conform formulei. Observaţi că la suprapunerea matricei kernel A peste elementele de pe marginea matricei B este necesar să o bordăm pe aceasta din urma cu linii/coloane de “0”.

3. Enunţ

Se doreşte implementarea algoritmului Blocked Matrix Convolution folosind nVidia CUDA în două forme: **fără memorie partajată** şi **cu memorie partajată** (shared memory) pornind de la scheletul de cod ataşat temei. De asemenea, se doreşte compararea performanţelor obţinute de cele două variante ale programului.

Observaţii:

- a nu se confunda *matricea kernel* (de ex. matricea A din acest enunţ) cu *funcţia kernel* (de ex. funcţia VecAdd din exemplul din laborator)
- pentru a putea aplica prin convoluţie matricea kernel A de 5x5 în toate punctele matricei B (inclusiv cele de pe margini), aceasta din urmă va trebui să se comporte ca şi cum ar fi bordată cu 2 linii/coloane de elemente de valoare 0 (nu faceţi bordarea propriu-zis, ci doar consideraţi că aveţi zerouri pe poziţiile care ies în afara matricei B)
- pentru a măsura performanţa programului, trebuie măsurat timpul petrecut de acesta în funcţia kernel, care, în principiu, se execută asincron; prin urmare, este necesar să apelăm `cudaThreadSynchronize()` înainte de a măsura timpul de terminare
- pentru a vă reaminti variantele de cod cu şi fără memorie partajată, urmăriţi exemplul din laborator referitor la înmulţirea de matrice în cele două variante

4. Testare și notare

Scheletul de cod are implementat un mecanism de verificare a rezultatului vostru comparând rezultatul obţinut de codul vostru cu cel obţinut prin calculul pe CPU (funcţia `ComputeGold()`). Programul va afişa "Test PASSED" dacă cele două rezultate corespund.

Această verificare a corectitudinii rezultatului trebuie să rămână în cod pentru a putea fi punctaţi!

Scheletul de cod conţine un script care va testa programul pe mai multe teste. Fiecare test va genera random matricele A şi B pentru o dimensiune a matricei B specifică testului. Scriptul este gândit pentru a urmări performanța algoritmului pe matrice B de mărimi din ce în ce mai mari, pornind de la 5x5 până la 512x512.

Punctajele de mai jos se acordă proporţional cu numărul de teste pentru care programul vostru dă rezultatul corect ("Test PASSED") .

Punctajul pentru implementarea cu memorie partajată se acordă doar dacă în memoria partajată se păstrează atât matricea kernel A, cât și o parte din matricea B (din exemplul de mai sus). Scrieți în README cât este dimensiunea sub-matricei din B păstrată în memoria partajată și de ce ați ales această dimensiune.

Punctajele se acordă astfel:

A. **3.5p** - Implementarea corectă fără memorie partajată a algoritmului
B. **3.5p** - Implementarea corectă cu memorie partajată a algoritmului
C. **1p** - Realizarea unui grafic cu timpii de rulare pentru fiecare dintre teste, pe cele trei variante:
    a. varianta de pe CPU (existentă în schelet)
    b. varianta de pe GPU fără memorie partajată (punctul A)
    c. varianta de pe GPU cu memorie partajată (punctul B)
D. **2p** - Readme (în care, pe lângă detaliile de implementare, trebuie analizate rezultatele obținute la punctul anterior și date explicațiile cu privire la dimensiunea sub-matricei din memoria partajată) şi calitatea codului