### **Logistika**

* Predaja se vrši preko github classroom sistema:   
  <https://classroom.github.com/a/ubs0-1dr>
* Rok za predaju je 05.01.2025 do kraja dana.
* Prilikom odbrane obavezno je koristiti kod iz poslednjeg commit-a iz repozitorijuma
* Kod na osnovu kojeg se brani projekat sme da sadrži maksimalno 5 komentara, koji ne smeju biti duži od 100 karaktera (otprilike jedna duža linija koda).
* U toku izrade projekta je dozvoljeno držati neograničeno mnogo komentara u kodu, i takav kod se može upotrebiti i za konsultacije (tj. ne morate da “čistite” kod ako hoćete nešto da pitate).

# **Zadatak**

## **Uvod**

Obrada slika je jedna od najčešćih primena paralelnog programiranja, gde se iste operacije primenjuju na veliki broj piksela nezavisno. U ovom projektu implementiraćemo tri osnovna algoritma za obradu slike: sivljenje (grayscale), Gaussovo zamagljivanje (Gaussian blur) i podešavanje svetline (brightness adjustment). Implementacija treba da bude u CUDA / PyCUDA okruženju (čista CUDA sa C host kodom se takođe priznaje, ali je verovatno teže).

## **Ključni algoritmi**

### **1. Sivljenje (Grayscale) [6 bodova]**

* Konvertovanje RGB slike u sivu sliku
* Za svaki piksel računamo ponderisanu sumu: 0.299 \* R + 0.587 \* G + 0.114 \* B
* Rezultat je jednokanalna slika sa vrednostima između 0 i 255
* Za sve bodove voditi računa o gustim pristupima memoriji za različite dimenzije slike

### **2. Gaussovo zamagljivanje (Gaussian Blur) [8 bodova]**

* Primena Gaussian filtera na svaki kanal slike zasebno
  + Filter je matrica težina bazirana na Gaussovoj distribuciji
  + Za svaki piksel, nova vrednost je ponderisana suma okolnih piksela
* Rezultat je zamagljena slika istih dimenzija kao originalna
* Filter matricu možete izračunati unapred, u python kodu
* Obrada različitih kanala (R,G,B) može biti paralelizovana
  + Koristiti z dimenziju bloka za paralelnu obradu kanala
* Za sve bodove:
  + Dimenzije filter matrice bi trebalo da budu parametarizovane
  + Kod bi trebalo da ispravno obrađuje i slike čije su dimenzije veće od veličine jednog bloka. Delimično rešenje (samo slike koje je moguće obraditi jednom blokom se priznaje, uz manje bodova)

### **3. Podešavanje svetline (Brightness Adjustment) [6 bodova]**

* Modifikacija vrednosti piksela u odnosu na srednju vrednost intenziteta
* Skaliranje razlike od srednje vrednosti zadatim faktorom
* Očuvanje vrednosti u opsegu 0-255
* Rezultat je svetlija ili tamnija slika, zavisno od faktora
* Implementacija je kroz dva kernela:
  + Računanje srednje vrednosti (sume, dok se deljenje sa veličinom slike može uraditi u python kodu)
  + Skaliranje vrednosti piksela
* Za sve bodove računanje sume bi trebalo da minimizuje divergenciju niti i vodi računa o konfliktima memorijskih banki

## **Implementacija**

Okvirni principi rešenja:

* Niti duž x i y dimenzije bloka koriste se za obradu pojedinačnih piksela
* Svaka nit obrađuje jedan piksel slike
* Rezultat se upisuje u odgovarajuću poziciju izlazne matrice

## **Napomene**

* Slike učitavati kao numpy array-e, a zatim prebacivati u CUDA memoriju
* Implementirati funkcije za svaki od tri algoritma zasebno
* Testirati implementaciju na slikama različitih dimenzija

## 

## **Dodatak: Detaljan opis Gaussian Blur algoritma**

### **1. Kreiranje Gaussianfiltera**

* Kreira se matrica težina dimenzija N×N (obično 3×3, 5×5 ili 7×7)
* Vrednosti se računaju prema 2D Gaussian formuli:  
  G(x,y) = (1/(2\*π\*σ²)) \* e^(-(x² + y²)/(2\*σ²))
* Gde su x i y udaljenosti od centra kernela, a σ (sigma) kontroliše stepen zamagljenja

Primer 3×3 kernela sa σ=1:  
[0.075 0.124 0.075]

[0.124 0.204 0.124]  
[0.075 0.124 0.075]

### **2. Normalizacija kernela**

* Sve vrednosti u kernelu se dele njihovom ukupnom sumom
* Osigurava očuvanje ukupne osvetljenosti slike

### **3. Primena filtera**

* Za svaki piksel u slici:
  + Centriranje kernela na trenutni piksel
  + Množenje preklopljenih piksela sa vrednostima iz kernela
  + Sumiranje proizvoda za novu vrednost piksela
* Matematički, ovo je konvolucija matrice slike sa Gaussian kernelom

### **4. Tretiranje ivica**

Kada je kernel centriran na piksel blizu ivice, deo kernela "pada" van slike. Strategije rešavanja (odaberite jednu i navedite koju ste uzeli):

* Padding (dodavanje piksela oko slike)
* Mirror padding (preslikavanje ivičnih piksela)
* Edge padding (ponavljanje ivičnih piksela)

### **CUDA implementacija**

Za efikasnu implementaciju u CUDA-i, važno je:

* Kernel vrednosti izračunati na CPU i proslediti kao konstantu
* Svaka nit obrađuje jedan piksel
* Učitati u deljenu memoriju sve piksele potrebne za obradu jednog bloka
* Veličina deljene memorije mora uključiti padding za ivične piksele bloka