# Projekat iz predmeta Paralelno programiranje

## Detekcija ivica unutar slike

Hristina Adamović SV32/2020

**Analiza problema**

Detekcija ivica je tehnika obrade slike za pronalaženje granica objekata unutar slika. Radi tako što otkriva diskontinuitet u osvjetljenosti i koristi se za segmentaciju slike i ekstrakciju podataka u oblastima kao što su obrada slike, kompjuterski vid i mašinski vid. U datom projektu korištena su dva algoritma obrade slike: detekcija ivica korištenjem prewitt operatora i detekcija ivica analizom okoline tačke.

**Koncept rješenja**

U slučaju oba algoritma, ulazna slika je predstavljenja 2-dimenzionalnom matricom koja sadrži podatke o vrijednostima boja pojedinih tačaka/piksela te slike. U nastavku će biti opisani serijska implementacija oba algoritma, a potom način njihove paralelizacije.

*Detekcija ivica korištenjem prewitt operatora* je algoritam koji funkcioniše tako što se ulazna slika/matrica filtrira prewitt operatorima odnosno operatorima za horizontalne i vertikalne ivice. Operatori su filteri predstavljeni kvadratnim matricama neparnih dimenzija (>=3). Filtriranje se vrši tako što se za svaku tačku ulazne matrice kreira podmatrica, koja mora biti iste dimenzije kao prewitt operator a čine je vrijednosti iz okoline odabrane tačke, i množi se sa odgovarajućim filterom. Množenje se vrši za oba filtera (vertikalnim i horizontalnim). Rezultati ova dva množenja se potom sabiraju po aposlutnim vrijednostima i zavisno od rezultata sabiranja određuje se vrijednost tačke na crnu ili bijelu boju. Cijeli algoritam se može posmatrati kao klizanje filtera preko svih podmatrica slike i obavljanja odgovarajućih aritmetičkih operacija nad svakim ovako kreiranim parom. Završetkom ovog algoritma dobija se filtrirana slika koja jasno prikazuje ivice date slike.

*Detekcija ivica analizom okoline tačke* je drugi algoritam koji rješava ovaj problem na način da za svaku tačku ulazne matrice provjerava njenu okolinu do određene udaljenosti, vrši poređenja tačaka i na osnovu toga donosi zaključke. Prvi korak je da se sve vrijednosti ulazne matrice promijene na 0 ili 1, zavisno od praga odsijecanja. Ovo će rezultovati crno-bijelom slikom, ali bez izraženih ivica. Dalje se za svaku tačku ovako promijenjene matrice provjerava okolina tačke, gdje se pod okolinom tačke podrazumijevaju sve tačke koje su lijevo, desno, gore, dole i dijagonalno od nje do određene udaljenosti. Provjera okoline se vrši na način da se pamte dva rezultata, jedan koji provjerava da li se u okolini tačke nalazi bar jedna jedinica, a drugi koji provjerava da li se u okolini tačke nalazi bar jedna nula. Zavisno od ove pretrage, navedeni rezultati određuju svoju vrijednost na 1 ili 0. Posljednji korak se ogleda u tome da se vrši apsolutna razlika ova dva rezultata i na osnovu razlike se vrijednost tačke postavlja na crnu ili bijelu boju. Ovim postupkom dobija se filtrirana slika.

*Paralelizacija* navedena dva serijska algoritma postiže se korištenjem oneTBB zadataka. Princip paralelizacije isti je za oba algoritma. Funkcioniše tako što se ulazna matrica podijeli na 4 manje matrice i kreiraju se 4 zadatka kojima se svakom proslijeđuje po jedna od podmatrica. Ovi zadaci se paralelno izvršavaju i postupak dijeljena matrica na podmatrice i kreiranja novih paralelnih zadataka se nastavlja sve dok se ne dođe do unaprijed definisane granice (određena visina ili širina) od koje se počinje sa izvršavanjem nekog od opisanih serijskih algoritama. Na ovaj način postiže se dosta brže i efikasnije izvršenje algoritma, što će detaljnije biti objašnjeno u nastavku.

**Programsko rješenje**

Za implementaciju projekta korišteno je nekoliko fajlova. *Parameters.h* fajl čuva vrijednosti parametara ključnih za podešavanje efikasnosti rada samih algoritama. *Range.cpp* je fajl koji sadrži implementaciju klase Range koja sadrži atribute koji pamte moguće vrijednosti odnosno opsege za x i y koordinate od kojih se dalje kreiraju tačke. Klasa takođe pamti vrijednosti cjelokupne dužine i širine slike i sadrži funkcije koje podešavaju vrijednosti prethodno navedenih atributa. Glavne funkcije za izvršenje programa nalaze se u *main.cpp* fajlu, a objašnjena tih funkcija biće data u nastavku.

Od biblioteka korištena je EasyBMP biblioteka koja služi za konverziju slike iz .bmp formata u matricu piksela sa RGB rasponom boja. Iz ove biblioteke korištena je klasa BitmapRawConverter koja vrši prethodno spomenuto.

U nastavku biće data objašnjena funkcija korištenih u kodu po sljedećem šablonu:

kratko objašnjenje funkcije

deklaracija funkcije

parametri funkcije

Funkcija koja za svaku tačku izvršava filtriranje za serijski kod algoritma detekcije ivica korištenjem prewitt operatora.

void multiply\_prewitt(int\* inBuffer, int\* outBuffer, int x, int y, int width)

*inBuffer* – bafer ulazne slike

*outBuffer* – bafer izlazne odnosno filtrirane slike

*x* – horizontalna koordinata tačke

*y* - vertikalna koordinata tačke

*width* – širina slike

Funkcija koja za svaku tačku izvršava filtriranje za serijski kod algoritma detekcije ivica analizom okoline tačke.

void search\_neighbour\_points(int\* inBuffer, int\* outBuffer, int x, int y, int width, int height)

Parametri identični kao u prethodnom primjeru.

*height* – visina slike

Funkcija koja za proslijeđeni opseg kreira tačke i poziva multiply\_prewitt funkciju za svaki kreirani par.

void filter\_serial\_prewitt(int\* inBuffer, int\* outBuffer, Range r)

Prva dva parametra identična kao u prethodnim primjerima.

*r* – instanca klase Range. Klasa Range sadrži atribute koji pamte moguće vrijednosti odnosno opsege za x i y koordinate od kojih će se kreirati tačke.

Funkcija koja za proslijeđeni opseg kreira tačke i poziva search\_neighbour\_points funkciju za svaki kreirani par.

void filter\_serial\_edge\_detection(int\* inBuffer, int\* outBuffer, Range r)

Parametri identični kao u prethodnom primjeru.

Funkcija koja filtrira cijelu ulaznu sliku/bafer i zavisno od *serial* parametra poziva funkciju za izvršenje serijskog ili paralelnog algoritma za detekciju ivica analizom okoline tačke.

void init\_edge\_detection(int\* inBuffer, int\* outBuffer, Range r, bool serial)

Parametri identični kao u prethodnom primjeru.

*serial* – ako sadrži vrijednost “*true“* poziva se funkcija za izvršenje serijskog algortima tj. filter\_serial\_edge\_detection, a ako sadrži vrijednost „*false“*, poziva se funkcija za izvršenje paralelnog algoritma tj. filter\_parallel

Funkcija koja predstavlja paralelnu implementaciju oba navedena serijska algoritma.

void filter\_parallel(int\* inBuffer, int\* outBuffer, Range r, Function\_name fn\_serial)

Parametri identični kao u prethodnim primjerima.

*fn\_serial* – predstavlja funkciju koja se poziva za serijsko izvršenje algoritma

**Ispitivanje**

Karakteristike računara na kojem je rađen projekat i na kojem su rađeni testni slučajevi iz ovog odjeljka su sljedeće:

Procesor: AMD Ryzen 3 3200U sa Radeon Vega Mobile Gfx, 2.60 GHz

RAM: 12.0 GB (9.95 GB iskoristivo)

Tip sistema: 64-bit operativni sistem, x64-baziran procesor

Operativni Sistem: Windows 10 Pro

Broj jezgara: 4

U nastavku biće prikazane tabele koje prikazuju zavisnost brzine izvršenja algoritama u odnosu na dimenzije ulazne slike, prag odsijecanja, veličinu prewitt operatora i udaljenost za okolinu tačke.

1. Tabela – prikaz vremena izvršavanja (s) serijskih algoritama za

različite dimenzije slika

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dimenzija slike | Prewitt operator dim = 3 | Prewitt operator dim = 5 | Udaljenost okoline tačke = 1 | Udaljenost okoline tačke = 2 | Udaljenost okoline tačke = 3 |
| 3888 x 2592 | 2.704 | 6.473 | 0.562 | 1.845 | 3.584 |
| 1200 x  800 | 0.282 | 0.614 | 0.068 | 0.187 | 0.367 |
| 1800 x 1200 | 0.573 | 1.347 | 0.129 | 0.483 | 0.946 |
| 2448 x 3264 | 2.120 | 5.020 | 0.463 | 1.234 | 2.797 |

Na osnovu prikazanih vrijednosti, zaključuje se da je serijski algoritam detekcije ivica analizom okoline tačke dosta brži, do skoro 5 puta, u odnosu na serijski algoritam detekcije ivica korištenjem prewitt operatora. Takođe, što su dimenzije slika veće, potrebno je više vremena za izvršenje ovih algoritama.

1. Tabela – prikaz vremena izvršavanja (s) paralelnih algoritama

za različite vrijednosti praga odsijecanja slike dimnezije 3888x2592

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prag odsijecanja | Prewitt operator dim = 3 | Prewitt operator dim = 5 | Udaljenost okoline tačke = 1 | Udaljenost okoline tačke = 2 | Udaljenost okoline tačke = 3 |
| 100 | 1.157 | 2.865 | 0.278 | 0.978 | 1.671 |
| 200 | 1.188 | 2.914 | 0.274 | 0.999 | 1.608 |
| 400 | 1.149 | 2.863 | 0.276 | 0.954 | 1.663 |
| 600 | 1.216 | 2.865 | 0.279 | 1.009 | 1.707 |
| 800 | 1.264 | 2.866 | 0.282 | 0.997 | 1.704 |

Na osnovu testiranja prikazanih u tabeli iznad, može se zaključiti da je najveće ubrzanje postignuto za prag odsijecanja = 400, iako su rezultati međusobno dosta slični. Ovim načinom paralelizacije dostiže se ubrzanje od približno 2 puta.

1. Tabela – prikaz vremena izvršavanja (s) paralelnih algoritama

za različite vrijednosti praga odsijecanja slike dimnezije 1200x800

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prag odsijecanja | Prewitt operator dim = 3 | Prewitt operator dim = 5 | Udaljenost okoline tačke = 1 | Udaljenost okoline tačke = 2 | Udaljenost okoline tačke = 3 |
| 100 | 0.098 | 0.256 | 0.027 | 0.066 | 0.152 |
| 200 | 0.123 | 0.256 | 0.030 | 0.077 | 0.160 |
| 400 | 0.100 | 0.319 | 0.031 | 0.070 | 0.180 |
| 600 | 0.116 | 0.266 | 0.027 | 0.092 | 0.163 |
| 800 | 0.100 | 0.794 | 0.036 | 0.178 | 0.382 |

Na osnovu testiranja prikazanih u prethodnoj tabeli, može se zaključiti da je najveće ubrzanje postignuto za prag odsijecanja = 100, iako su ponovo većina rezultata dosta slični. S obzirom da se radi nad slikom dosta manjih dimenzija, očekivano je da prag odsijecanja bude niži u ondosu na prethodni primjer. U ovom primjeru, paralelizacijom se dostiže ubrzanje koje iznosi približno 2.5 puta.

1. Grafički prikaz podataka iz tabele 2
2. Grafički prikaz podataka iz tabele 3

**Analiza rezultata**

Kao što je već prikazano u prethodnim primjerima, paralelizacijom serijskih algoritama postiže se ubrzanje od približno 2-2.5 puta, što je zadovoljavajući rezultat s obzirom da se radi sa 4 jezgra. Slike većih dimenzija zahtijevaju više vremena za izvršavanje, i u slučaju serijskog i paralelnog algoritma, što je logično s obzirom da se nad njima vrši više operacija. Algoritam detekcije ivica analizom okoline tačke, kako serijska tako i paralelna verzija, izvršavaju se vidno brže (skoro 5 puta) u odnosu na algoritam koji koristi prewitt operatore. Povećanjem dimenzije filtera, kao i povećanjem udaljenosti za okolinu tačke, povećava se vrijeme izvršenja algoritama do čak 3.5 puta.

Najbolji rezultati što se tiče samog izgleda rezultujuće slike postižu se korištenjem prewitt operatora dimenzije 3, kao i udaljenošću okoline tačke 1. U nastavku dati su primjeri rezultujućih slika sa ovim parametrima.

1. Originalna slika (dimenzija 1200x800)



1. Slika dobijena korištenjem prewitt operatora dimenzije 3



1. Slika dobijena korištenjem udaljenosti okoline tačke 1

