Analiza jedne implementacije serijske i paralelne detekcije ivica unutar slike u programskom jeziku C++

Srđan Stjepanović

Sadržaj

1. Uvod 3
2. Implementacija 4
3. Ispitivanje 7
4. Analiza rezultata 11
5. Tabele 12

Uvod

U ovom projektnom zadatku potrebno je bilo proučiti dva algoritma za detekciju ivica unutar slike. Cilj je bio od početne slike u boji dobiti crno-bijelu sliku sa izraženim ivicama. Korištena su dva algoritma za detekciju ivica. Prvi algoritam implementiramo pomoću Prewitt operatora, gdje je sliku potrebno filtrirati filterima za horizontalne, odnosno vertikalne ivice, a zatim rezultujuću vrijednost izračunati zbirom apsolutnih vrijednosti horizontalne i vertikalne komponente. Na kraju je potrebno otkloniti sivu komponentu, postavljanjem vrednosti na bijelu ili crnu boju u zavisnoti da li je početna vrijednost manja ili veća od 128. Drugi implementirani algoritam radi tako što se prvo sve vrijednosti postave na 0 ili 1 odsijecanjem za 128, a zatim se provjerava okolina svake tačke i bilježi se 1 ako u okolini postoji vrijednost 1, a u suprotnom se bilježi vrijdnost nula. Isti postupak se koristi i za provjeru postojanja vrijednosti 0 u okolini tačke. Zatim se računa razlika apsolutnih vrijednosti zapamćenih informacija o okolini i rezultat se vraća u opseg 0-255.

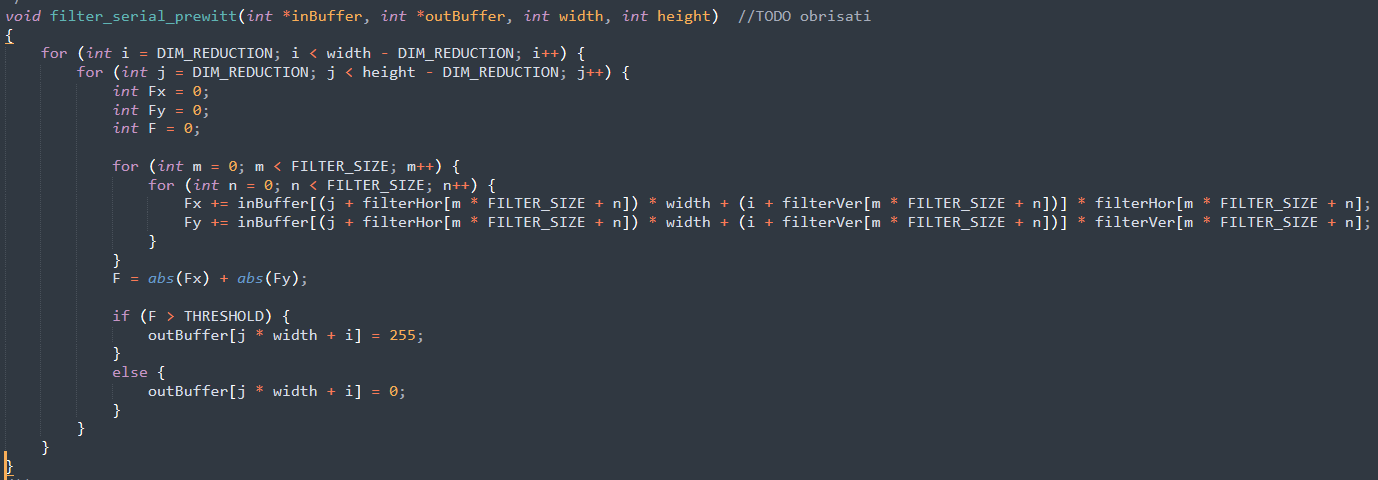
U daljem tekstu ćemo analizirati jednu implementaciju ovih algoritama u serijskoj i paralelnoj varijanti u programskom jeziku C++. Pri paralelizaciji algoritma korištena je apstrakcija zadataka, obezbeđena od strane Intel-ove Thread Building Blocks biblioteke. Sva mjerenja vršena su na AMD Ryzen 7 3700U četvorojezgarnom (sa 8 logičkih jezgara) procesoru.

Implementacija

U uvodu smo se kratko dotakli Prewitt operatora koji smo koristili za detekciju ivica unutar slike. Prewitt operator možemo zamisliti kao klizanje matrice filtera preko svih podmatrica slike i obavljenje traženih operacija nad ovako stvorenim preklapajućim matricama. Prilikom implementacije ovog algoritma trebalo je voditi računa da kada bi se filter postavljao preko cijele matrice došlo bi do izlaska van opsega same matrice. Ovaj problem smo riješili smanjivanjem dimenzija matrice sa svih strana za određeni broj koji zavisi od veličine filtera koji je kvadratna matrica neparnih dimenzija.

U ovoj sekciji izdvojićemo samo najbitnije dijelove implementacije.

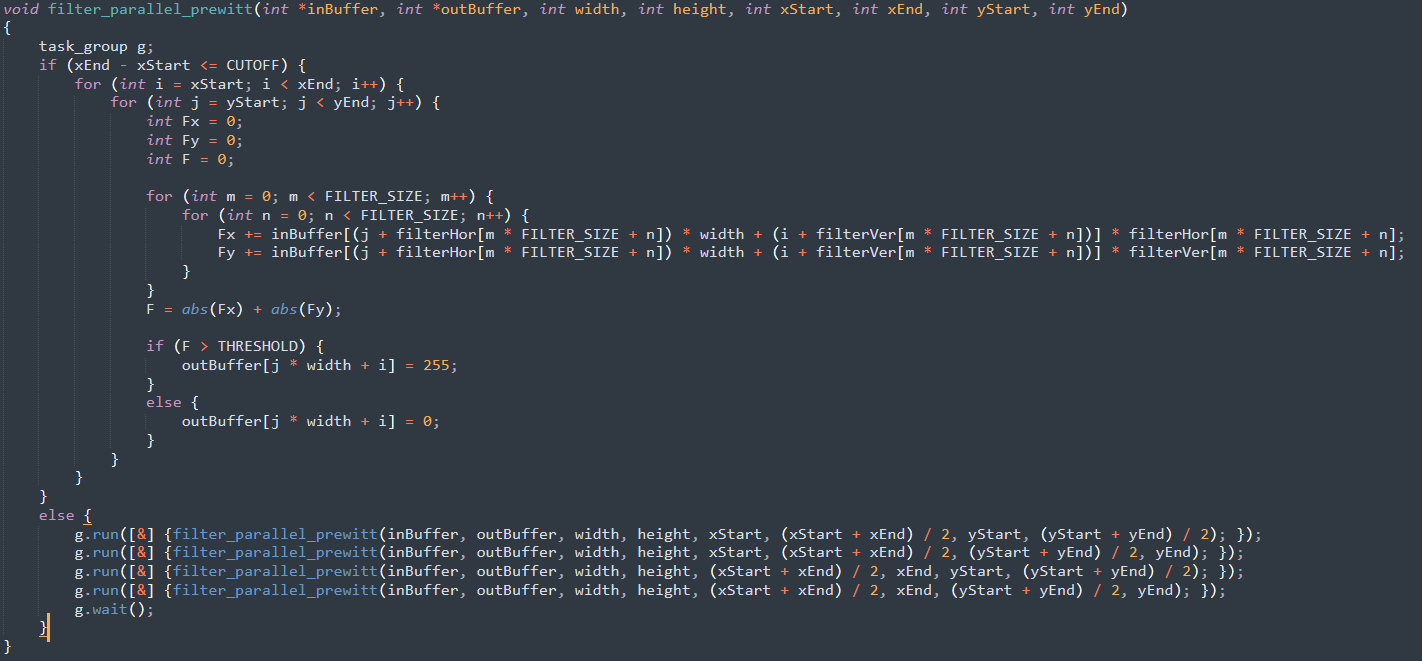
Serijska implementacija Prewitt operatora



Parametri same funkcije su ulazni i izlazni bafer i dimenzije ulazne fotografije. Prve dvije ugnježdene petlje koristimo za prolazak kroz sve tačke matrice, s tim da koristimo konstantu DIM\_REDUCTION definisanu na globalnom nivou kako bi smanjili matricu u zavisnosti od veličine filtera i izbjegli izlazak van opsega. Zatim sa dvije unutrašnje petlje množimo odgovarajuće elemente filtera i podmatrice na kojoj se trenutno nalazimo. Nakon računanja vertikalne i horizontalne komponente sumiramo apsolutne vrijednosti rezultata i tačku postavljamo na odgovarajuću vrijednost u zavisnoti od odnosa rezultata i konstante THRESHOLD.

Paralelna implemetacija Prewitt algoritma

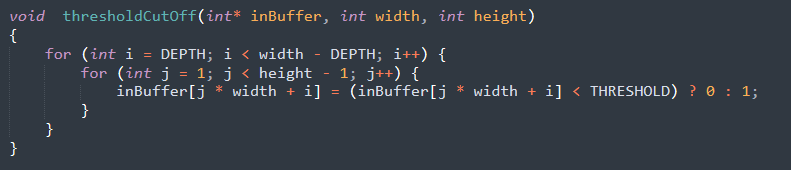
Za implementaciju paralelizma korištena je podjela na zadatke task\_group-a obezbeđenog od stane Intel-ovog TBB-a.



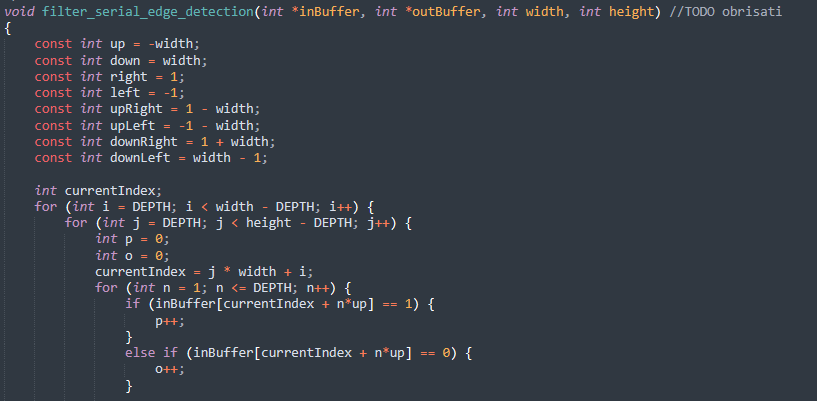
Ideja je da se posao prolaženja kroz cijelu matricu podijeli na manje dijelove koji će raditi u paraleli. Ovo postižemo tako sto matricu dijelimo da 4 jednaka dijela koja se obrađuju u paraleli, ovo dijeljenje se odvija rekurzivno sve dok širina proslijeđenog dijela matrice ne postane manja od unaprijed zadane veličine CUTOFF, nakon čega se obrada zadatka odvija identično kao i serijski. Na početku prilikom poziva same funkcije pored standardnih parametara kao i kod serijske verzije dodatno prosleđujemo indekse matrice po širini(x osi) i visini(y osi) koji predstavljaju oblast zaduženja zadatka. Ako je proslijeđena udaljenost od početnog i krajnjeg indeksa po x manja od CUTOFF serijski obrađujemo zadani dio matrice, u suprotnom pomoću task\_group-a kreiramo 4 nova rekurzivna poziva sa umanjenim zadacima za obradu. Na kraju pozivamo funkciju g.wait() koja predstavlja teorijsku naredbu „sync“ kako bi se sačekalo da se svi rekurzivni pozivi vrate prije povratka same roditeljske funkcije.

Serijska implementacija detekcije ivica

Za implementaciju ovog algoritma potrebno je prvo vrijdenosti cijele matrice postaviti na 1 ili 0 u zavisnosti od odnosa tačke i TRESHOLD-a.



Algoritam je implementiran tako da se prolazi kroz cijelu matricu i provjerava okolina u svim pravcima. Na početku inicijalizujemo vrijdnosti koje trebamo dodati na trenutni indeks kako bi dobili tačku koja je susjedna od tekuće u određenom pravcu, tako da inicijalizujemo 8 tačaka koliko ima i susjeda(pravaca).



Zatim sa dvije ugnježdene petlje prolazimo kroz matricu, tako da prilikom svake iteracije postavimo vrijdnost p i o na nulu. Vrijednost p predstavlja broj susjeda koji su 1, a o predstavlja susjede koji su 0. Zatim provjeravamo u svim pravcima da li je susjed 1 ili 0, i u zavisnosti od toga povećavamo p i o. Na slici je prikazana provjera samo za susjeda koji se nalazi iznad tekuće tačke, analogno se provjerava i u drugim pravcima. Koristimo još jednu petlju koja nam omogućava da provjeravamo susjede koji su više udaljeni odnosno koristimo promjenljivu dubinu analize slike. Nakon provjere svih susjeda, stavljamo p na 1 ako je p veće od nula i analogno radimo za o. Zatim rezultat postavljamo na 0 ili 255 u zavisnoti od razlike absolutnih vrijednosti p i o.

Paralelna implementacija detekcije ivica

Paralelna implementacija ovog algoritma realzivoana je na identičan način kao i paralelizacija prewitt algoritma, gdje obradu ulazne matrice dijelimo na manje zadatke koji se rekurzivno odvijaju u paraleli. Korištena je ista biblioteka TBB-a.

Ispitivanje

Kao što je navedeno u uvodu testiranja su vršena na AMD Ryzen 7 3700U četvorojezgarnom procesoru sa 8 logičkih jezgara. Analiza je vršena na ulaznim slikama različitih dimenzija, tako što smo varirali vrijednost CUTOFF konstate koja nam služi da natjeramo paralelni algoritam da radi serijski kada ga podijelimo na dovoljno dijelova. U samom algoritmu implementirano je zapisivanje rezultata rada programa u odgovarajućim fajlovima koji se nalaze u folderu test\_data. Zabilježene rezultate smo koristili u python skriptama za crtanje dijagrama. Za analizu rezultata smo koristili python biblioteke matplotlib i numpy. Takođe smo vršili analizu ubrzanja u odnosu na različite dimenzije filtera kod Prewitt operatora, gdje smo koristili 250 kao CUTOFF, koji je izabran eksperimentalnom metodom.

Analiza Prewitt operatora

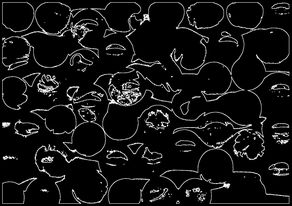
Prvo smo analizirali sliku dimenzija 1499x1001. Izgled slike i rezultati obrade prikazani su na sledećim fotografijama.



Početni izgled

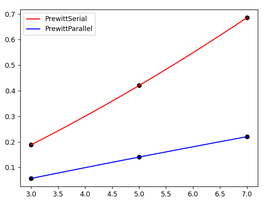


Izlazna slika nakon detekcije Prewitt operatorom

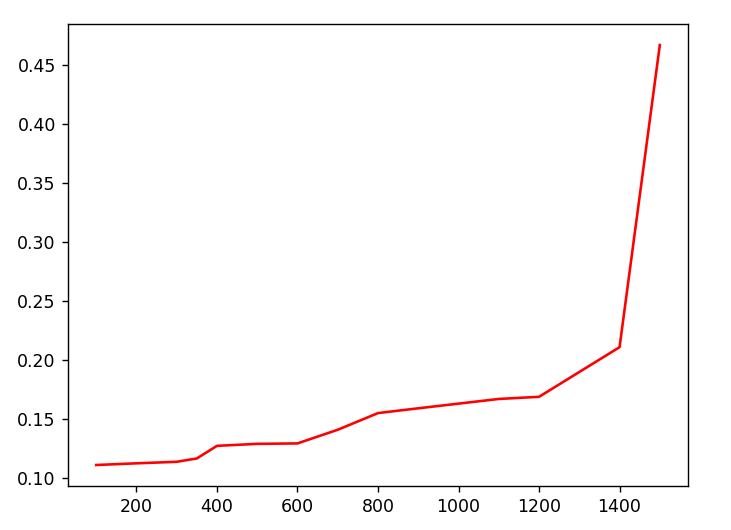


Izlazna slika nakon detekcije ivica drugom metodom

U nastavku su prikazani grafici analize performane za prethodne slike. Na sledećem grafiku je prikazan odnos brzine izvršavanja serijskog i paralelnog algoritma detekcijom ivica Prewitt operatorom u zavisnosti od dimenzija filtera (3, 5, 7), a korišten je CUTOFF 350, izabran eksperimentalnom metodom.(dim. 1499x1001)

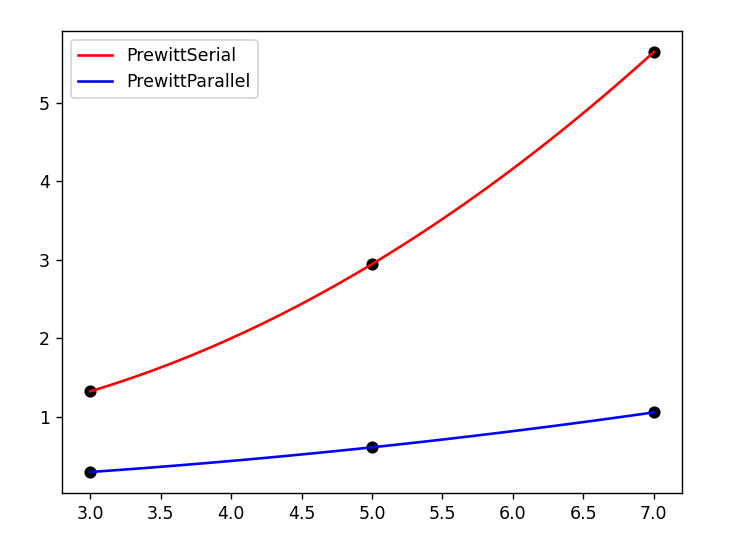


Na sledećem grafiku je prikazan odnos ubrzanja u odnosu na različite vrijdnosti za CUTOFF, s tim da je korišten filter dimenzija 5x5 za svaku različitu vrijdnost CUTOFF-a.(dim. 1499x1001)

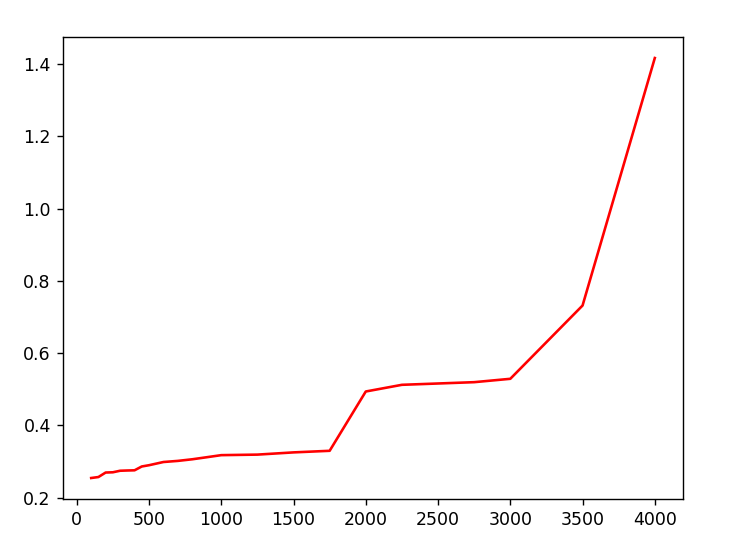


Nakon analize prve slike, za primjer smo uzeli sliku vecih dimenzija kako bi uporedili ubrazanje u odnosu na različite dimenzije slika. Dimenzije druge slike su 3888x2592, izgled ove slike je prikazan u specifikaciji projekta s toga u ovoj dokumentaciji nece biti prikazan, fokus će biti na samim performansama.

Sledeća fotografija prikazuje odnos brzine izvršavanja serijskog i paralelnog algoritma detekcijom ivica Prewitt operatorom u zavisnosti od dimenzija filtera (3, 5, 7), a korišten je CUTOFF 350, izabran eksperimentalnom metodom. (dim. 3888x2592)



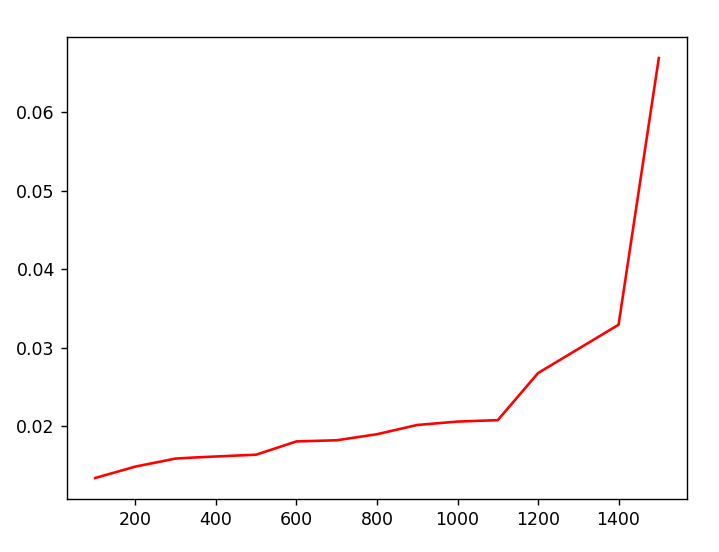
Na sledećem grafiku je prikazan odnos ubrzanja u odnosu na različite vrijdnosti za CUTOFF, s tim da je korišten filter dimenzija 3x3 za svaku različitu vrijdnost CUTOFF-a.(dim. 3888x2592)



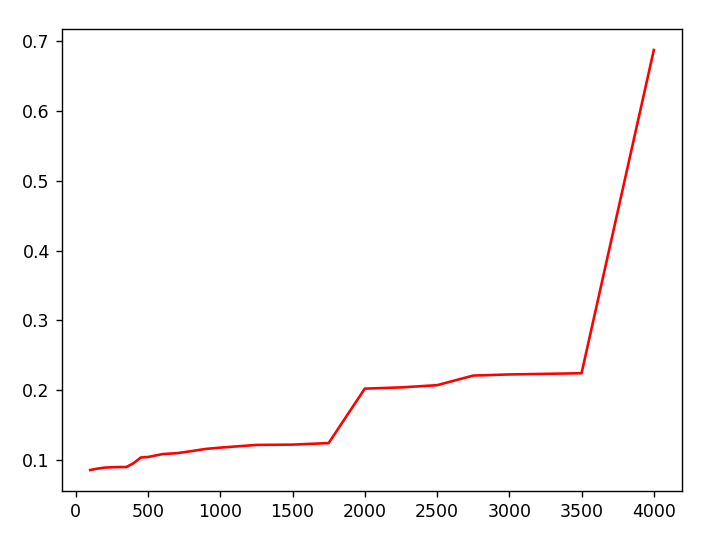
Analiza drugog algoritma

Za analizu drugog algoritma sa susjedima koristilo smo iste slike kao i za prethodnu analizu. Varirali smo vrijednost CUTOFF-a i gledali brzinu izvršavanja paralelnog algoritma. Algoritam je vršen samo za neposredne susjede, odnosno dubina je postavljena na jedan.

Sledeći grafik prikazuje odnos brzine paralelnog algoritma za sliku dimenzija 1499x1001



Sledeći grafik prikazuje odnos brzine paralelnog algoritma za slika dimenzija 3888x2592



Možemo primijetiti da je brzina izvršavanja veoma slična za veliki opseg vrijednosti CUTOFF-a s toga eksperimentalno trebamo izabrati odgovarajuću vrijednost, kako ona ne bi bila suviše mala gdje bi bilo preskupo kreirati zadatke i serijska impelementacija bi radila brže.

Analiza rezultata

U prethodnom poglavlju mogli smo vidjeti grafike koji prikazuju različita testiranja performansi algoritama. Kada je riječ o paralelizaciji Prewitt operatora prva dva grafika nam prikazuju odnos brzine izvršavanja serijskog i paralelnog algoritma za različite dimenzije. Možemo primijetiti da je ubrazanje veoma dobra gdje se slika dimenzija 3888x2592 filterom dimenzija 3x3 serijski obrađuje za 1.32s dok paralelno za 0.29s što je odlično ubrzanje. Povećanjem dimenzija filtera odnos serijskog i paralelnog vremena se još više poboljšava. Ubrzanje kod slike manjih dimenzija je prema grafiku istog reda veličine kao i kod veće slike, ali zbog svojih manjih dimenzija i serijski algoritam se izvršava relativno brzo, s toga se paralelizacija vidi više na slikama većih dimenzija. Za paralelizaciju pomoću zadataka veoma je bitna vrijdnost na osnovu koje prekidamo dijeljenje matrice i ostatak izvršvamo serijski, jer ako izaberemo suviše veliku vrijednost zadaci će biti i dalje veliki i nećemo dobiti traženo ubrzanje. Sa druge strane treba paziti da nam vrijednost za CUTOFF ne bude previše mala, jer u tom slučaju kreiranje zadatak će biti skuplje od same serijske obrade što takođe želimo izbjeći. Vrijdenost CUTOFF-a je birana eksperimentalnom metodom, tako što je prvo izabrana velika vrijednost i zatim smanjivala dok nismo dobili odgovarajuću vrijednost. Ovo dosta zavisi od samih performansi mašine na kojoj se izvršava algoritam, odnosno procesora, propusnog opsega itd. Paralelizacijom druge metode za detekciju ivica smo takođe dobili zadovoljavajuće ubrzanje, gdje se za sliku dimenzija 3888x2592 sa CUTOFF-om od 350 serijski kod izvršava za 0.76s, a paralelni za 0.08s. Analizom grafika možemo vidjeti da dosta veliki spektar vrijednosti za CUTOFF-a daje slične rezultate, gdje se optimalnu vrijednost takođe tražili eksperimentalnom metodom. Samo serijsko izvršavanje ove druge metode je dosta brže od izvršavanja Prewitt operatora, s toga su i paralelna izvršenja dosta brža za istu vrijednost CUTOFF-a.

Na kraju zaključujemo da u vrijeme višejezgarnih procesora paralelizacija veoma bitna ako želimo da naši projekti zadovolje određene performanse. U konkretno ovom projektu koristili smo Intel-ovu TBB biblioteku i njena šablone koji su nam omogućili poboljšanje performansi obrade slike.

Tabele

|  |  |  |  |
| --- | --- | --- | --- |
| Dimenzije slike | Dimenzije filtera | Serijska verzija | Paralelna verzija |
| 1499x1001 | 3x3 | 0.188 | 0.057 |
| 1499x1001 | 5x5 | 0.420 | 0.140 |
| 1499x1001 | 7x7 | 0.68 | 0.220 |

Tabela za grafik 1

|  |  |  |
| --- | --- | --- |
| Dimenzije slike | CUTOFF | Paralelna verzija |
| 1499x1001 | 100 | 0.113 |
| 1499x1001 | 200 | 0.116 |
| 1499x1001 | 300 | 0.128 |
| 1499x1001 | 350 | 0.140 |
| 1499x1001 | 400 | 0.128 |
| 1499x1001 | 500 | 0.110 |
| 1499x1001 | 600 | 0.126 |
| 1499x1001 | 700 | 0.112 |
| 1499x1001 | 800 | 0.154 |
| 1499x1001 | 900 | 0.166 |
| 1499x1001 | 1000 | 0.158 |
| 1499x1001 | 1100 | 0.210 |
| 1499x1001 | 1200 | 0.168 |
| 1499x1001 | 1400 | 0.162 |
| 1499x1001 | 1500 | 0.466 |

Tabela za grafik 2

|  |  |  |  |
| --- | --- | --- | --- |
| Dimenzije slike | Dimenzije filtera | Serijska verzija | Paralelna verzija |
| 3888x2592 | 3x3 | 1.325 | 0.298 |
| 3888x2592 | 5x5 | 2.945 | 0.612 |
| 3888x2592 | 7x7 | 5.646 | 1.057 |

Tabela za grafik 3

|  |  |  |
| --- | --- | --- |
| Dimenzije slike | CUTOFF | Paralelna verzija |
| 3888x2592 | 100 | 0.274 |
| 3888x2592 | 150 | 0.269 |
| 3888x2592 | 200 | 0.269 |
| 3888x2592 | 250 | 0.251 |
| 3888x2592 | 300 | 0.289 |
| 3888x2592 | 350 | 0.298 |
| 3888x2592 | 400 | 0.254 |
| 3888x2592 | 450 | 0.275 |
| 3888x2592 | 500 | 0.301 |
| 3888x2592 | 600 | 0.274 |
| 3888x2592 | 700 | 0.311 |
| 3888x2592 | 800 | 0.285 |
| 3888x2592 | 900 | 0.305 |
| 3888x2592 | 1000 | 0.329 |
| 3888x2592 | 1250 | 0.317 |
| 3888x2592 | 1500 | 0.325 |
| 3888x2592 | 1750 | 0.318 |
| 3888x2592 | 2000 | 0.512 |
| 3888x2592 | 2250 | 0.493 |
| 3888x2592 | 2500 | 0.515 |
| 3888x2592 | 2750 | 0.731 |
| 3888x2592 | 3000 | 0.519 |
| 3888x2592 | 3500 | 0.528 |
| 3888x2592 | 4000 | 1.422 |

Tabela za grafik 4

|  |  |  |  |
| --- | --- | --- | --- |
| Dimenzije slike | Dubina | CUTOFF | Paralelna verzija |
| 1499x1001 | 1 | 100 | 0.016 |
| 1499x1001 | 1 | 200 | 0.013 |
| 1499x1001 | 1 | 300 | 0.018 |
| 1499x1001 | 1 | 350 | 0.014 |
| 1499x1001 | 1 | 400 | 0.016 |
| 1499x1001 | 1 | 500 | 0.018 |
| 1499x1001 | 1 | 600 | 0.016 |
| 1499x1001 | 1 | 700 | 0.015 |
| 1499x1001 | 1 | 800 | 0.032 |
| 1499x1001 | 1 | 900 | 0.020 |
| 1499x1001 | 1 | 1000 | 0.020 |
| 1499x1001 | 1 | 1100 | 0.020 |
| 1499x1001 | 1 | 1200 | 0.018 |
| 1499x1001 | 1 | 1400 | 0.026 |
| 1499x1001 | 1 | 1500 | 0.066 |

Tabela za grafik 5

|  |  |  |  |
| --- | --- | --- | --- |
| Dimenzije slike | Dubina | CUTOFF | Paralelna verzija |
| 3888x2592 | 1 | 100 | 0.103 |
| 3888x2592 | 1 | 200 | 0.090 |
| 3888x2592 | 1 | 300 | 0.095 |
| 3888x2592 | 1 | 400 | 0.097 |
| 3888x2592 | 1 | 500 | 0.109 |
| 3888x2592 | 1 | 600 | 0.116 |
| 3888x2592 | 1 | 700 | 0.108 |
| 3888x2592 | 1 | 800 | 0.104 |
| 3888x2592 | 1 | 900 | 0.112 |
| 3888x2592 | 1 | 1000 | 0.122 |
| 3888x2592 | 1 | 1250 | 0.117 |
| 3888x2592 | 1 | 1500 | 0.124 |
| 3888x2592 | 1 | 1750 | 0.121 |
| 3888x2592 | 1 | 2000 | 0.204 |
| 3888x2592 | 1 | 2250 | 0.207 |
| 3888x2592 | 1 | 2500 | 0.222 |
| 3888x2592 | 1 | 2750 | 0.224 |
| 3888x2592 | 1 | 3000 | 0.221 |
| 3888x2592 | 1 | 3500 | 0.202 |
| 3888x2592 | 1 | 4000 | 0.722 |

Tabela za grafik 6