Prirodno matematički fakultet
Univerzitet Crne Gore

Seminarski rad

Predmet:
Paralelno programiranje/Praralelni algoritmi

Tema rada:

Primjeri paralelizacije nad tehnikama obrade slika

Profesori:

Prof. dr. Igor Jovanović
mr. Nikola Pižurica

Studenti:

Milica Simović 10/24 D
Ognjen Tomčić 2/24 C

Sadržaj

Slika i paralelna obrada

Slika je dvodimenzionalna raspodjela malih tačaka slike koje se nazivaju pikseli. Može se posmatrati kao funkcija od dvije realne promjenljive, na primjer f(x, y), gdje je f amplituda (npr. osvijetljenost) slike na poziciji (x, y). Obrada slike je proces poboljšavanja slike i izdvajanja značajnih informacija iz nje. Obrada slike dobija sve veći značaj u različitim oblastima primjene. Široko se koristi u mnogim oblastima, uključujući filmsku industriju, medicinsko snimanje, industrijsku proizvodnju, vremensku prognozu itd. U nekim od ovih oblasti veličina slika može biti veoma velika, dok vrijeme obrade mora biti veoma kratko, a ponekad je potrebna i obrada u realnom vremenu.

Paralelna obrada predstavlja ključni dio svakog modela računanja visokih performansi. Ona uključuje korišćenje velikih količina računarskih resursa za rešavanje složenog zadatka ili problema. Resursi specifični za paralelnu obradu su procesor (CPU) i memorija.

Obrada slika uz pomoć paralelnog računanja predstavlja alternativni način za rješavanje problema obrade slika koji zahtevaju dugo vrijeme obrade ili rukovanje velikom količinom informacija u "prihvatljivom vremenu". Glavna ideja paralelne obrade slika jeste da se problem podijeli na jednostavne zadatke i da se oni rješavaju istovremeno, tako da se ukupno vrijeme podijeli između ukupnog broja zadataka (u najboljem slučaju). Paralelna obrada slika ne može se primijeniti na sve probleme, drugim riječima, ne mogu se svi problemi kodirati u paralelnoj formi.

Osobine paralelnog programiranja

Paralelno programiranje mora da zadovoljava određene karakteristike kako bi se operacije izvršavale efikasno. Neke od tih karakteristika su:

- Brojčanost (engl. granularity) - definisano je kao broj osnovnih jedinica operacije i dalje se dijeli na:
    - Krupne jednice (engl. coarse-grained) nekoliko jedinica zahtjevnijih za izvršavanje
    - Sitne jedinice (engl. fine grained) ogroman broj jedinica lakših za izvršavanje
- Sinhronizacija - sprečava preplitanje procesa
- Kašnjenje (engl. latency) - vrijeme koje je potrebno zahtjevu od slanja do stizanja rezultata
- Skalabilnost - mgućnost algoritma da održi efikasnost uz proporcijalno povećavanje procesorske moći i veličine zadatka
- Ubrzanje i efikasnost - metrika da se ocjeni kvalitet paralelne implementacije
- “Overheads” - dodatno neophodno vrijeme za računanje

Tipovi paralelizacije

Pipeline paralelizacija

Duge sekvence procedura ili zadataka su paralelne, ali dodatno postoji preklapanje uzastopnih procesa tokom kojih je moguće izvršavanje više paralelnih zadataka. Relacioni model se dobro uklapa u ovaj tip paralelizacije. Rezultat nekih relacijskih operatora postaje ulaz za druge operatore; zbog toga dolazi do određenog čekanja.

Nezavisni ili prirodni paralelizam

Ova paralelizacija, kao što joj samo ime kaže, nema preklapanja između procesa, zato je brža od pipeline paralelizacije.

Inter- query and intra-query parallelism

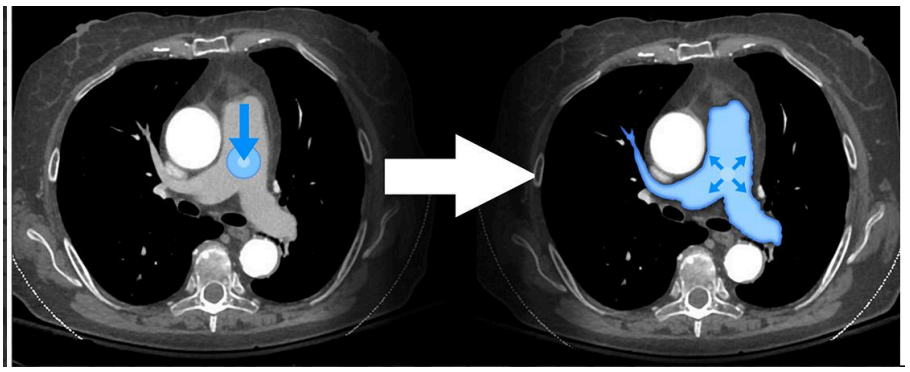
Transakcije su nezavisne. Više procesora (CPU-a) može biti aktivno tako što se svakom zadatku ili upitu dodijeli poseban procesor. Ovaj tip paralelizma, koji koristi više odvojenih i nezavisnih upita u isto vrijeme, naziva se među-upitni (inter-query) paralelizam.

Task parallelism

U strategiji paralelizma zadataka, preporuke za obradu slika i procedure niskog nivoa grupišu se u zadatke, pri čemu se svaki zadatak dodjeljuje posebnoj procesorskoj jedinici. Glavni izazov u ovoj strategiji je efikasna dekompozicija podataka i kompozicija rezultata.

Regional Growing Segmentation

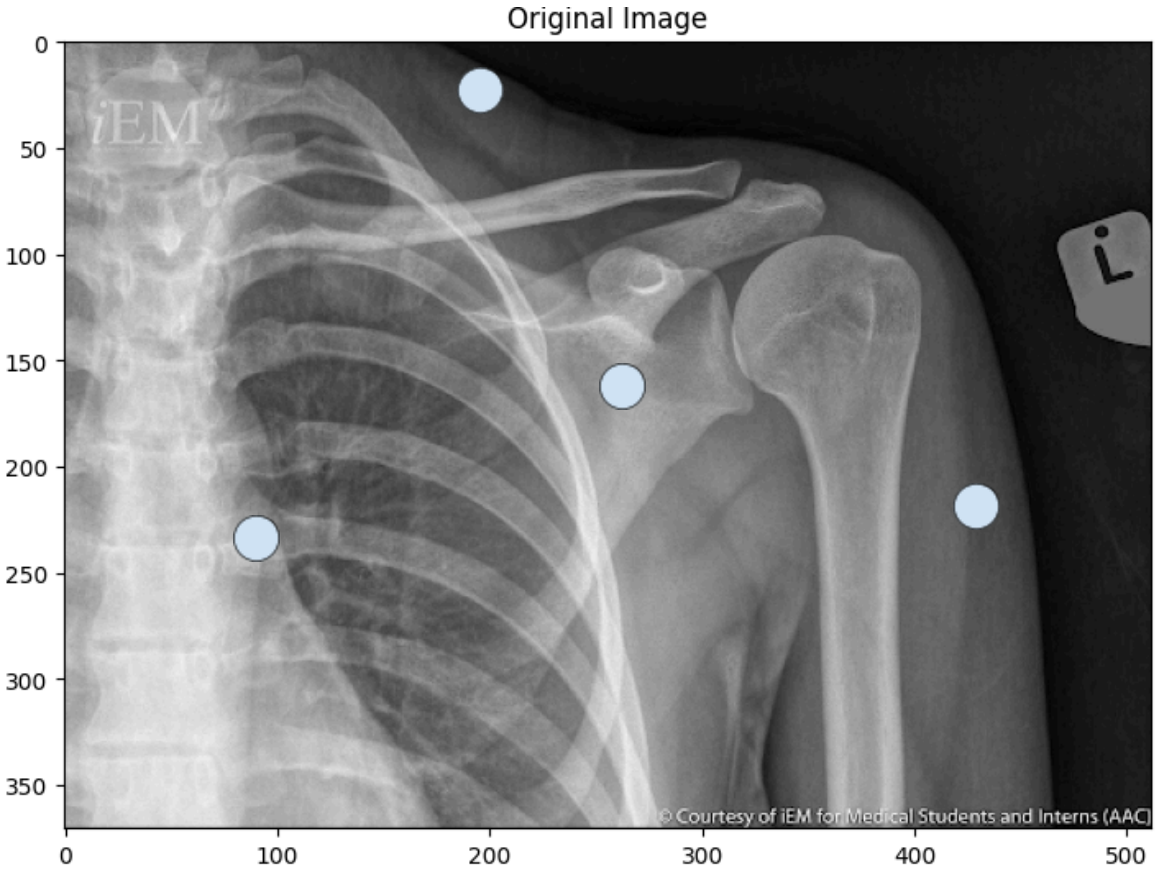
Regional Growing Segmentation je tehnika segmentacije koja grupiše susjedne piksele na osnovu određenog kriterijuma sličnosti. Počinje tako što se odaberu početni pikseli (engl. seed pixel) i grupa piksela se postepeno širi susjednim pikselima koji su zadovoljili uslov link.



Ova tehnika segmentacije ima primjenu u medicini kod analize rendgenskih snimaka, gdje naglašava određene anatomske strukture. Može da segmentira strukture koje na prvi pogled nisu uočljive.

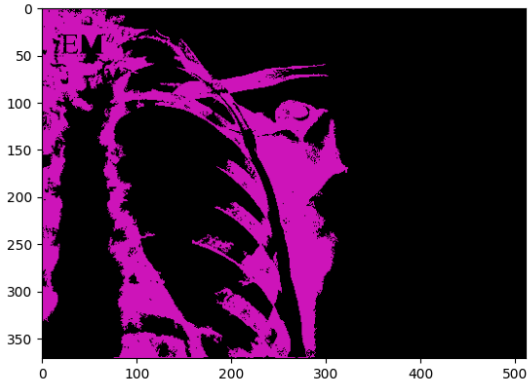
Serilizacioni pristup

U narednom dijelu objasniće se sekvencijalna realizacija ovog algoritma na primjeru slike 1, rendgenskog snimka ramena. Kao što je navedeno prije, prvo se izaberu početni pikseli na slici, a oni su uveličani plavim tačkama.

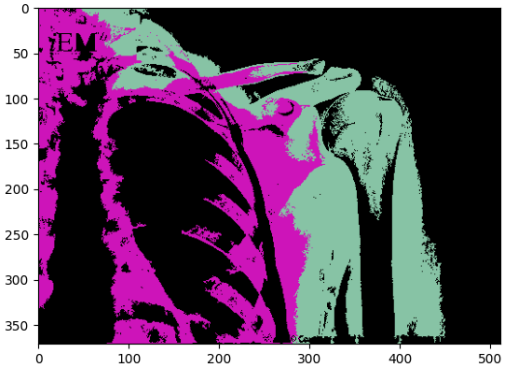


Slika 1. Slika rendgenskog snimka ramena

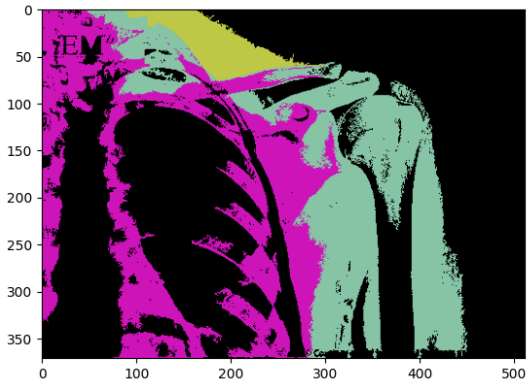
Nakon selekcije piksela iterativno se prolazi svaki odabrani piksel i konstantno se dodaju okoline piksela koji zadovoljavaju određeni uslov, u ovom slučaju uslov je mala razlika u osvetljenosti.



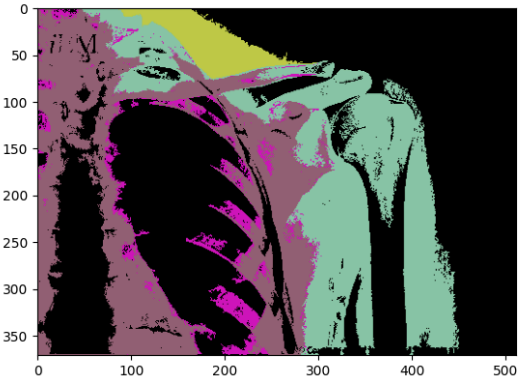
Slika 2. Iteracija 1.



Slika 3. Iteracija 2.

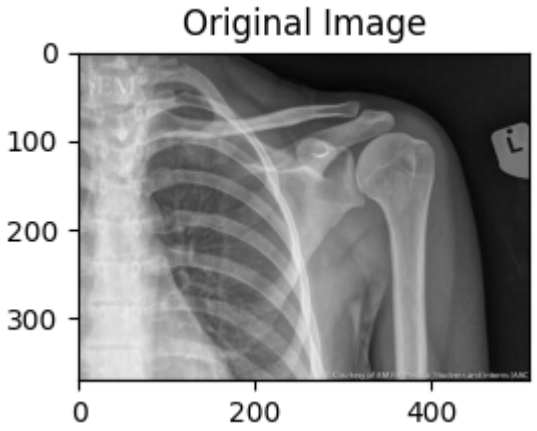


Slika 4. Iteracija 3.

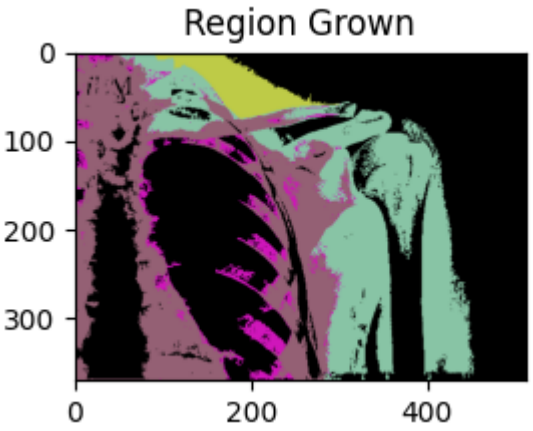


Slik 5. Iteracija 4.

Nakon završetka algoritma na slikama 6 i 7 možemo da vidimo rezultate segmentacije.



Slika 6. Originalna slika



Slika 7. Nakon algoritma

Kod koji radi sledecu operaciju segmentacije pomoću “regiona growing” segmentacije.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
import time

def region_growing(img, seed, threshold=5):
    h, w = img.shape
    segmented = np.zeros((h, w), np.bool_)
    visited = np.zeros((h, w), np.bool_)

    stack = [seed]
    seed_value = img[seed]

    while stack:
```

```
        x, y = stack.pop()
        if visited[x, y]:
            continue
        visited[x, y] = True
        if abs(int(img[x, y]) - int(seed_value)) < threshold:
            segmented[x, y] = True
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < h and 0 <= ny < w and not visited[nx,
ny]:
                        stack.append((nx, ny))
    return segmented

def apply_color_to_mask(mask, color):
    h, w = mask.shape
    output = np.zeros((h, w, 3), dtype=np.uint8)
    output[mask] = color

    return output

def add_region(B, A):
    non_black_mask = np.any(A != 0, axis=2)
    B[non_black_mask] = A[non_black_mask]


def run(file_path):
    img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)

    h, w = img.shape
    region_grown = np.zeros((h,w,3),dtype=np.uint8)

    t_1 = time.time()
    for _ in range(4):
        color =
(random.randrange(255),random.randrange(255),random.randrange(255))
        seed_point = (random.randrange(h), random.randrange(w))

        one_region = region_growing(img, seed_point, threshold=30)
        one_region_color = apply_color_to_mask(one_region,color)
        add_region(region_grown,one_region_color)
    t_2 = time.time()

    plt.subplot(1, 2, 1)
    plt.title("Originalna slika")
    plt.imshow(img,cmap='gray')
```

```
    plt.subplot(1, 2, 2)
    plt.title("Region Grown")
    plt.imshow(region_grown)
    plt.show()
    return t_2-t_1

run('snimak.jpg')
```

Paralelizacija “Region Growning Segmentation”

Regional Growing Segmentation se često koristi. Glavna mana ovog pristupa je da zahtjeva mnogo vremena da se izvrši.

Ovaj algoritam je stvoren da smanji vrijeme potrebno za izvršavanje gornjeg algoritma. Postupak izvršavanja je sledeći:

1. Učitavanje slike na procesor
2. Odabir početnih piksela (može da zavisi od izbora klijenta, odabira posebnih raspona nijansi sive, ravnomjerne raspoređenosti piksela po rešetki i sl.)
3. Aktiviranje proporcionalnog broja kernela u zavisnosti od broja početnih piksela
4. Slanje kopije slike i početnog piksela na odgovarajjući novonastali kernel procesor
5. Pripajanje susjednih pikselia regionu ako zadovoljavaju kriterijum
6. Računanje da li je u pitanju region od interesa (engl. region of interest/ROI)
7. Slanje regiona sa ROI početnom procesoru
8. Rekonstrukcija slike i prikazivanje informacija

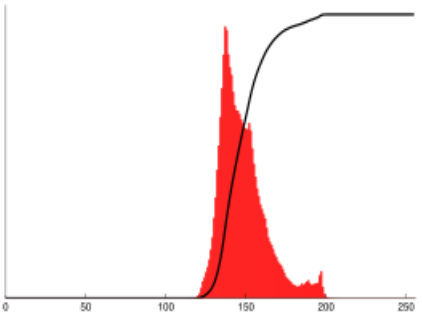
U ovom postupku, svakom početnom pikselu dodijelio bi se kernel, što znači da će ukupno vrijeme za nekoliko početnih piksela biti svedeno kao na računanje jednog početnog piksela. Odnosno za n početnih piksela biće n-ostruko brže, pod uslovom da imamo isti broj kernela. Ukoliko je broj kernela p koji je manji od broja početnih piksela, tad će algoritam biti n/p-ostruko puta brži.

Histogram ekvalizacije

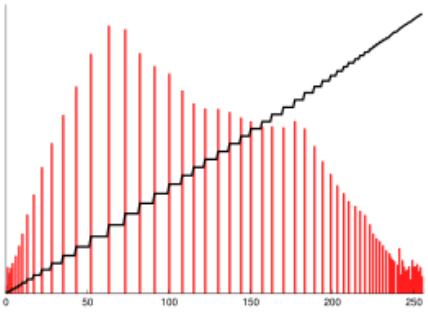
Radi jednostavnosti razmatraćemo historame sivoskaliranih slika. Histogram slike predstavlja ukupan broj pojavljivanja određene boje (inteziteta sive skale) u slici. Ima brojne primjene a jedna od njih je ocjena kvaliteta slike. Iako slika može da ima veliku rezoluciju to ne znači da će biti jasna, recimo, usled velike osvetljenosti ili zatamnjenja. Za takvu sliku kažemo da nema optimalan kvalitet slike. Određena istraživanja pokazuju da je optimalan kvalitet slike prisutan ako je broj pojavljivanja svih boja jednako raspoređen, odnosno ako histogram ima uniformnu raspodjelu. Postoji transformacija koja svaku sliku transformiše tako da histogram dobije uniformniju rasopdjelu koja se naziva ekvalizacija histograma.

Slika 8. Originalna slika

Slika 9. Ekvivalizovana slika



Slika 10. Histogram slike

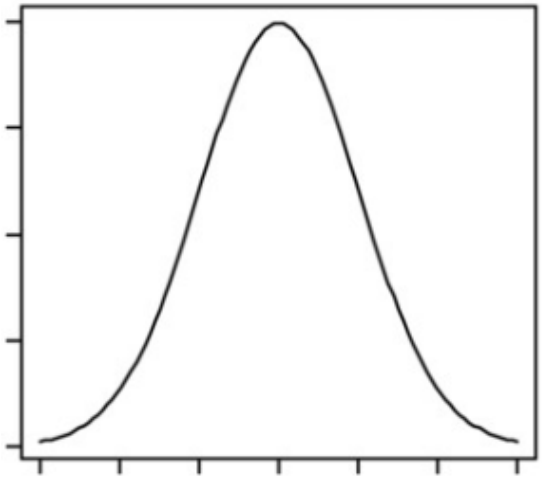


Slika 11. Histogram ekvivalizovane slike

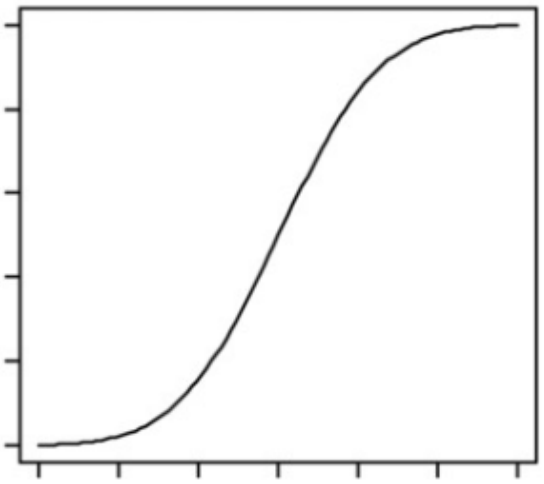
Slika 8 predstavlja originalnu sliku, a njen histogram je prikazan na slici 10.. Slika 9. je slika nad kojom se primjenila metoda histogram ekvalizacije, a njen odgovarajući histogram je na slici 11. Možemo da uočimo da je kvalitet slike 9 bolji i da je na njenom histogramu raspodjela uniformnija u odnosu na prethodnu sliku.

Kulmulativan histogram

Kulmulativan histogram (slika 13.) opisuje histogram slike (slika 12.). On predstavlja sumu pojavljivanja piksela čije su vrijednosti boje manje ili jednake od nezavisne promjenjive. Radi boljeg razumijevanja napraviće se paralela sa vjerovatnoćom. Histogram predstavlja funkciju gustine raspodjele, a kulmulativni histogram predstavlja funkciju raspodjele.



Slika 12. Histogram



Slika 13. Kulmulativni histogram

Metod ekvalizacije

Da bi se primjenio histogram ekvalizacije neophodno je da znamo kulmulativni histogram slike. Formula kojom se preslikava vrijednosti piksela dat je sledećom formulom.

$$T(r_k) = (L-1) \cdot \sum_{i=0}^{k} \frac{h_i}{n}, \qquad k = 0, 1, ..., L-1$$

Gdje je T funkcija transformacije, L je najveći intezitet boje, n ukupan broj piksela, h_i broj piksela inteziteta, a k intezitet piksela koji se razmatra r_k. Suma je zapravo kulmulativni

Pristup serijalizacije

Ovaj pristup bii zahtjevao primjenu ekvalizacije histograma čitave slike.

Paralelni pristup

Paralelni pristup bi se odnosio na sledeće korake:

1. Izračunati kulmulativni histogram slike i funkciju ekvalizacije histograma
2. Podijeliti sliku na nekoliko uzastopnih redova ili kolona
3. Za svaki red ili kolonu aktivirati kernel koji će uzeti odgovarajući red ili kolonu
4. Kernel računa kulmulativni histogram reda ili kolone
5. Pikseli se preslikavaju u nove vrijednosti pomoću histograma ekvivalizacije
6. Redovi ili kolone se objedinjuju na klijentovom procesu

Paralelni pristup bi ubrzao ovaj algoritam n-ostruko puta, gdje je n broj redova, pod uslovom da na raspolaganju imamo isti broj procesa. Ukoliko imamo p procesa, p≤n, ubrzanje bi bilo n/p-ostruko.

Paralelna segmentacija slike pomoću globalnog određivanja praga

"Tresholding" ili (neki dobar prevod) je jedna od najjednostavnijih vidova obrade slike koji se može koristiti za konvertovanje sivoskaliranih slika u binarne.

Slika 14. Originalna slika

Slika 15. Binarna slika

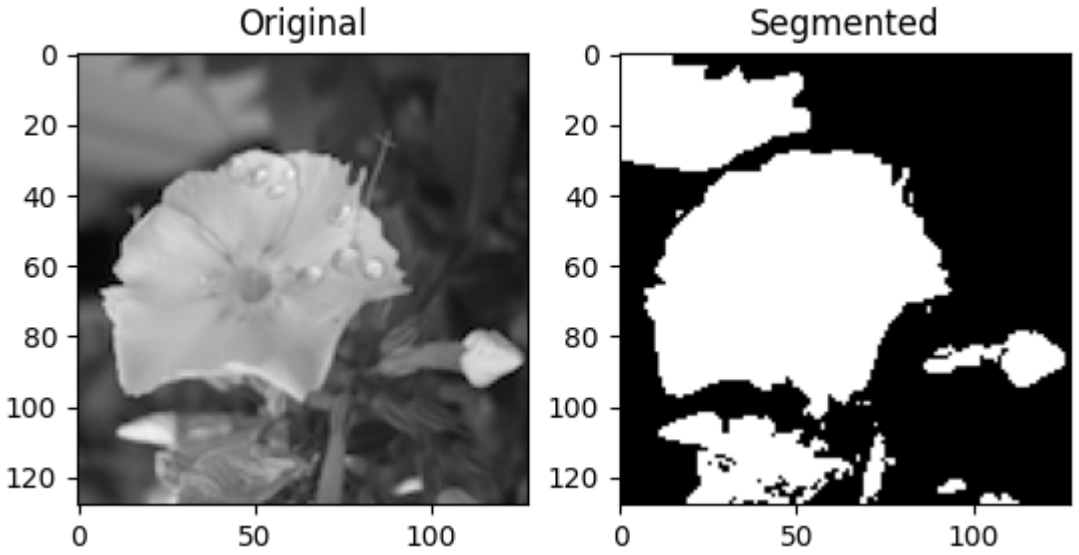
```python
def global_thresholding(image, threshold=127, max_iter=100, tol=0.5):
    """global thresholding iterativnim metodom"""
    prev_threshold = threshold

    for i in range(max_iter):
        below = image[image < prev_threshold]
        above = image[image >= prev_threshold]

        if len(below) == 0 or len(above) == 0:
            break

        mean1 = np.mean(below)
        mean2 = np.mean(above)

        new_threshold = (mean1 + mean2) / 2

        if abs(new_threshold - prev_threshold) < tol:
            break

        prev_threshold = new_threshold

    segmented = np.where(image >= prev_threshold, 255, 0).astype(np.uint8)
    return segmented, prev_threshold
```

Slika 16. Iterativni kod za globalni tresholding

Za iterativni metod algoritam je sljedeći:

1) Inicijalizujemo neki prag (konkretno u primjeru 127, jer predstavlja sredju vrijednost opsega pri sivoskaliranju)

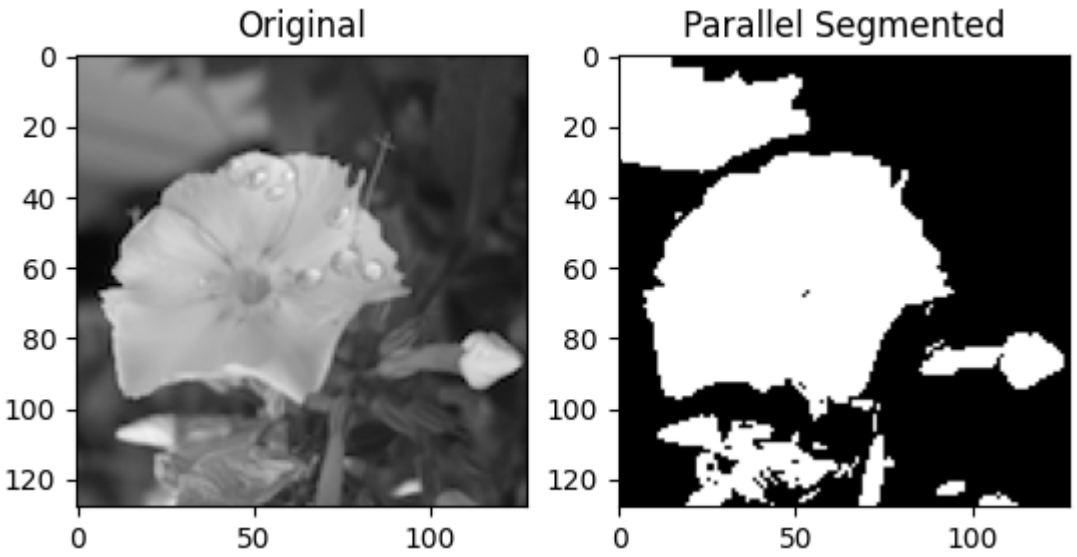
2) Ponavljanje sljedećih koraka dok ne dođemo do maksimalnog broja iteracija (ovdje 100):
    a) razdvajanje slike na 2 grupe: ispod i iznad praga
    b) izračunavanje srednjeg intenziteta obije grupe
    c) postavljanje praga na novi po formuli $T = (\mu_1 + \mu_2) / 2$
    d) ukoliko se novi prag od starog razlikuje za manje od zadate tolerancije zaustavljamo iteriranje
    e) ukoliko algoritam nije zaustavljen, uzimamo novi prag i nastavljamo iteriranje
3) Primjenjujemo prag nad čitavom slikom i vraćamo konačan rezultat



Slika 17. Originalna i binarna slika dobijena primjenom algoritma

Koraci za paralelizaciju ovog algoritma su sljedeći:

1) U klijentskom procesoru, slika se trasformiše u oblik kvadratnog stabla
2) Šalje se flag od klijentskog ka random procesoru i aktiviraju se 4 radna procesora
3) Svi djelovi slike se šalju na različite kernel ili radne procesore
4) Biraju se preliminarni pragovi T0-T3 zasebno za svaki od 4 procesora
5) Procjenjuju se srednje vrijednosti μ za svaki piksel ispod i iznad praga
6) Računa se novi prag po formuli $T = (\mu_1 + \mu_2) / 2$ za svaki procesor
7) Ponavljaju se koraci 5) i 6) sve dok prestane da dolazi do promjene u pragovima kod procesora
8) Klijentskom procesoru se šalje segmentisani region
9) Deaktiviraju se radni procesori
10) Rekonstruiše se segmentisana slika



Slika 18. Original i binarna slika dobijeni primjenom algoritma

Slika 19. Segmenti binarne slike prije konačnog spajanja