Paralelna realizacija metoda za obradu slike pomoću NVIDIA CUDA tehnologije

Marko Haralović, Ita Poklepović, Lana Barić, Dino Babić, Anabel Dautović, Valeria Javornik

2. lipnja 2024.

Sažetak

U ovom radu analizirana je razlika u performansi između centralnih procesorskih jedinica (CPU) i grafičkih procesorskih jedinica (GPU) prilikom izvršavanja nekih od odabranih metoda obrade slika koristeći CUDA tehnologiju. Ukratko će biti opisana teoretska razlika između funkcionalnosti CPU i GPU, a onda objašnjena i prikazana implementacija odabranih metoda za koju su korištene Numpy, Cupy i scikit-image open-source python biblioteke. Korištene metode odabrane su zbog svoje važnosti u području obrade digitalnih slika, a uključuju matrično množenje, detekciju rubova pomoću Sobel i Canny operatora, grupiranje s K srednjim vrijednostima, treniranje neuronske mreže za klasifikaciju rukom pisanih znamenaka, te ostale metode koje će kasnije biti predstavljene. Eksperimenti pokazuju da razlike u kvaliteti rezultata između CPU i GPU nema, ali u brzini izvođenja metoda, GPU je i do 10 puta veći.

1. Uvod

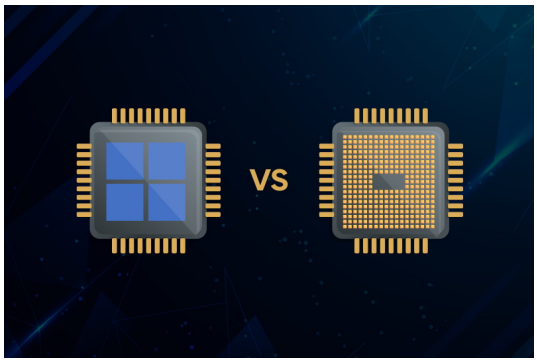
Područje obrade slika obuhvaća mnoge tehnike i metode za manipulaciju i analizu slika, koje se koriste za različite primjene u gotovo svim disciplinama. U ovom radu, fokus je na obradi informacija gdje je ulazni signal digitalna slika prikazana kao dvodimenzionalna matrica s jednim kanalom (slika bez boje - siva vrijednost) ili s tri kanala (slika u boji - RGB kanali).
Obrada slika je izuzetno važna zbog svoje široke primjene u mnogim područjima. Na primjer, koristi se za olakšavanje medicinske dijagnostike, industrijske automatizacije, forenzike, autonomnih vozila, poljoprivrede, geografije, građevine i mnogih drugih disciplina. Široka primjenjivost ovog područja zahtjeva da korištene metode budu ne samo točne, već i efikasne u pogledu korištenja resursa. U računarstvu, cilj je često da se metode izvode što brže, ali pritom treba uzeti u obzir i trošak bržeg izvođenja, posebno kada se ubrzanje postiže unapređenjem fizičke arhitekture računala.

2. GPU, CPU i aktivnost u svijetu

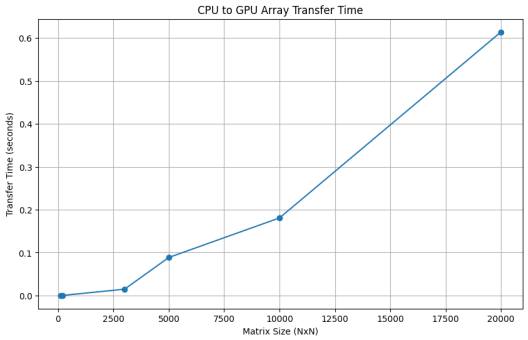
Grafički procesori (GPU) prvo su se počeli koristiti za poboljšanje performansi videoigara. Već 1970-ih godina, u arkadne igre su se ugrađivali posebni čipovi čiji je zadatak bio prikazivanje videoigre na ekranu. Međutim, trebalo je dugo vremena da se GPU-ovi počnu šire koristiti za ubrzavanje drugih procesa.

CPU (centralna procesorska jedinica) i GPU općenito se razlikuju u svojoj arhitekturi i primjeni. CPU se sastoji od manjeg broja jezgri (najčešće od 2 do 16), što ograničava njegovu sposobnost za istovremeno izvršavanje zadataka. Kada se zadaci paraleliziraju na CPU-u, koriste se njegove jezgre, no zbog malog broja jezgri, samo se nekoliko zadataka može izvršavati istovremeno. Često se koristi prividna paralelizacija gdje se više zadataka brzo prebacuje s jedne jezgre na drugu. Ovakva paralelizacija nije najefikasnija jer je promjena konteksta dretvi vremenski neučinkovita.

S druge strane, GPU se sastoji od velikog broja jezgri, koje u boljim grafičkim karticama mogu brojati desetke tisuća. Ovo ga čini idealnim za paralelno izvođenje velikog broja jednostavnih operacija. Na slici 1. može se vidjeti razlika u arhitekturi dviju procesorskih jedinica.



Slika 1.: Pojednostavljeni prikaz CPU (lijevo) i GPU (desno) [1]



Slika 2.: Prikaz vremena prijenosa matrica različitih veličina na GPU.

Trenutno je NVIDIA vodeća tvrtka u razvoju GPU tehnologije. Osnovana je 1993. godine, a svoju popularnost stekla je lansiranjem prvog grafičkog procesora, nazvanog GeForce. CUDA je također tehnologija koju je razvila NVIDIA, a omogućuje programerima da pišu kod i pritom koriste funkcije koje će se izvršavati paralelno, na grafičkom procesoru.

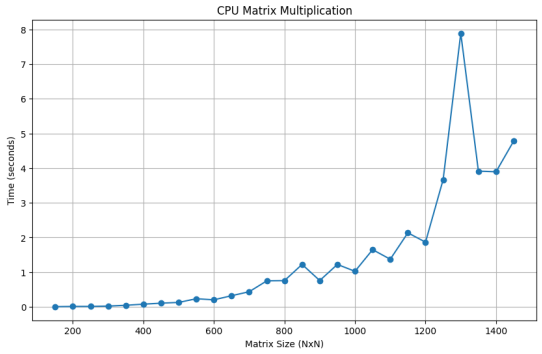
3. Prijenos podataka između CPU i GPU

Prijenos podataka s CPU na GPU je vremenski zahtjevan. Da bi korištenje GPU-a doista bilo isplativo, potrebno je baratati s velikom količinom podataka. Ako je podataka malo, tada će CPU obaviti istu operaciju brže nego li GPU, upravo zbog prijenosa podataka na GPU. Na slici 2. prikazana su vremena prijenosa matrica na GPU. Najmanja matrica je veličine 100x100, a najveća veličine 20000x20000.

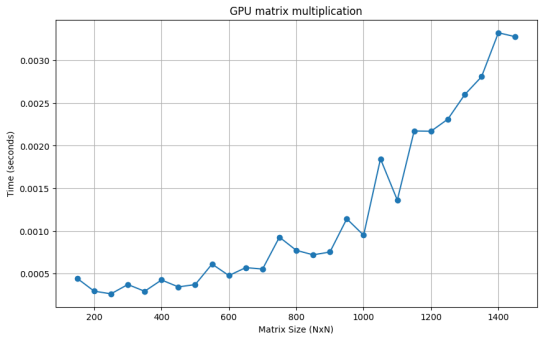
4. Pregled implementacija metoda za obradu slika pomoću NVIDIA CUDA tehnologije

4.1. Usporedba matričnog množenja na CPU i GPU te primjena nad transformacijama slike

Matrično množenje vrlo je često korištena operacija te je nužno osigurati njezino učinkovito izvođenje čak i u slučaju vrlo velikih matrica. Razmotrimo sljedeći primjer. Kreirane su kvadratne matrice veličina od 150x150 do 1500x1500. Svaka matrica pomnožena je sama sa sobom te je vrijeme obavljanja te operacije zabilježeno. Na slici 3. prikazana su vremena izvođenja u slučaju korištenja CPU-a i u slučaju korištenja GPU-a. Na slici možemo uočiti da je izvođenje značajno brže na GPU, čak i za relativno male veličine matrica.



(a) Vrijeme izvođenja na CPU.



(b) Vrijeme izvođenja na GPU.

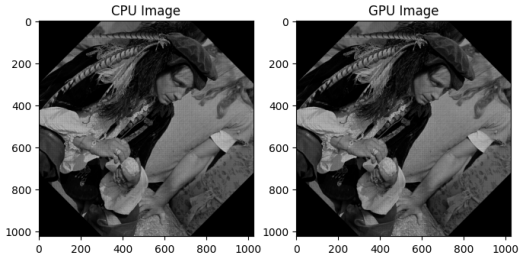
Slika 3.: Usporedba vremena množenja matrica na CPU i GPU.

4.2. Linearna geometrijska transformacija

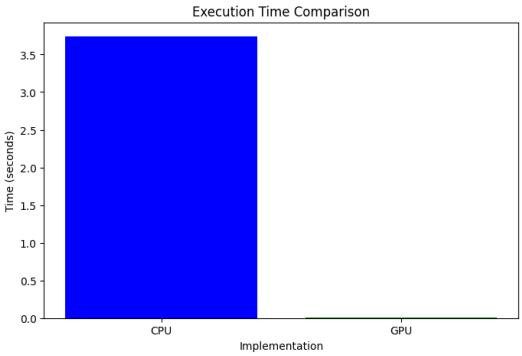
Linearne geometrijske transformacije slika uključuju translaciju, skaliranje i rotaciju. Prikazana matrica transformacije rotacije (1) određuje koja je nova pozicija jednog pixela slike. Ova je metoda savršen primjer za paralelizaciju jer je račun nove pozicije svakog pixela u potpunosti neovisan o ostalima pa se teoretski mogu svi računati u isto vrijeme. Naravno to neće uvjek biti moguće zbog ograničenog broja jezgri GPU-a.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{1}$$

Na slici 4. je vidljivo da paralelizaciju pomoću GPU-a nikako ne utječe na ispravnost transformacije, ali zato kao što je vidljivo iz grafa 5. znatno utječe na brzinu jer se koristeći CPU pozicija svakog piksela treba slijedno računati. Korištena slika je veličine 1024 x 1024 i rotirana je za 45°.



Slika 4.: Rezultati rotacije računate na CPU i GPU



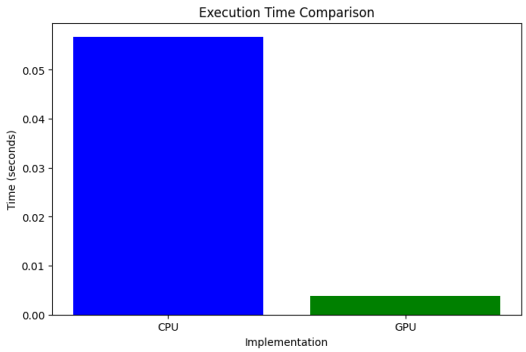
Slika 5.: Razlika u vremenu izvođenja transformacije rotacije na CPU (3.738 s) i GPU (0.005 s)

4.3. Sobel operator za detekciju rubova

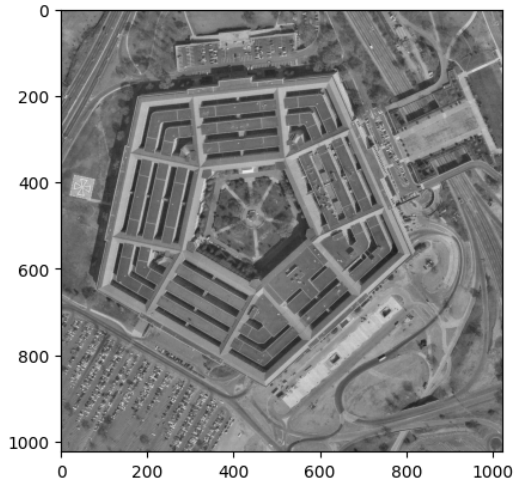
Sobel operator jedan je od poznatijih algoritama za detekciju rubova u slikama. Za aproksimaciju prve derivacije intenziteta slike koristi dvije matrice veličine 3x3.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \text{i} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Izradili smo dvije implementacije Sobel operatora, jednu koristeću biblioteku NumPy i drugu koristeći biblioteku CuPy koja omogućava izvođenje na GPU. Na slici 6. prikazana je usporedba vremena izvoženja Sobel implementacije koja se izvodi na GPU i one koja se izvodi na CPU.



Slika 6.: Implementacija Sobel operatora koja se izvodi na GPU značajno je brža od one koja se izvodi na CPU.



Slika 7.: Originalna slika dimenzija 1024 x 1024

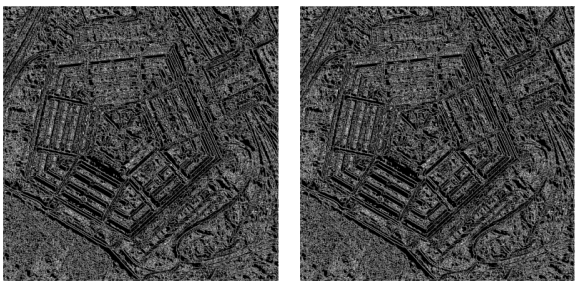
4.4. Detekcija rubova koristeći Canny detektor

Iako je Canny detektor rubova John F. Canny predstavio još u 1986. u članku [1] on je i dan danas ostao jedan od najpouzdanijih algoritama detekcije rubova u slikama.

Canny detektor radi u više koraka. Prvi korak je redukcija šuma korištenjem Gaussovog filtra koji blago zamagljuje sliku kako bi se umanjio efekt nasumičnog šuma. Nakon uklanjanja šuma koristi se Sobel operator (objašnjen u prošlom poglavlju) kako bi se izračunao intezitet promjene vrijednosti piksela slike i njen smjer. Ovaj način detekcije rubova vraća dosta neodređene rubove pa u sljedećem koraku Canny detektor njih stanjuje eliminirajući sve piksele koji se ne nalaze na lokalnom maksimumu u pravcu gradijenta što rezultira preciznijim rubovima. Kako bi se dalje izbacili rubovi koji su potencijalno nastali zbog šuma ili varijacije koriste se dva praga: visoki i niski. Dalje se gledaju pikseli rubova. Ako je gradijent piksela manji od niskog praga odbacuje se. Ako je veći od visokog praga označuje se kao jaki rub, dok se ostali pikseli rubova označuju kao slabi rub.

U zadnjem koraku se izbacuju pikseli slabih rubova u slučaju kada se ne mogu povezati s jakim rubom.

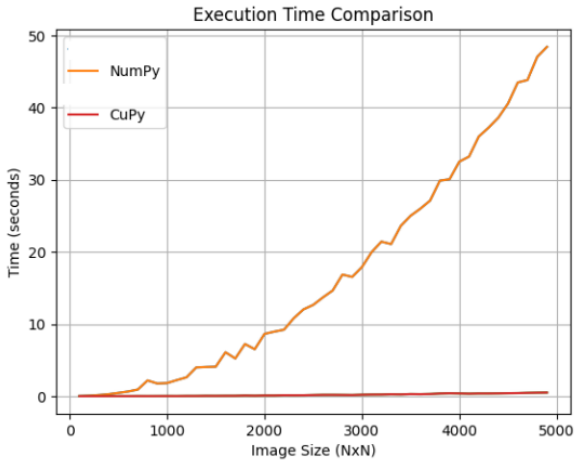
Na slikama ispod mogu se vidjeti originalna slika 7., dobiveni rezultat Canny detektora na CPU i GPU 8..



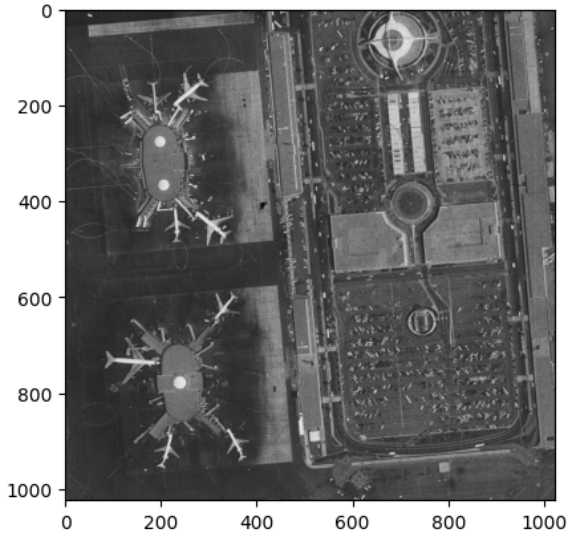
(a) CPU (4.807 s) (b) GPU (0.028 s)

Slika 8.: Rezultati Canny detektora rubova nad slikom 7.

Na slici 9. vidi se razlika između vremena potrebnog za izvođenje funkcije na CPU i GPU ovisno o veličini slike koju procesiramo. Možemo vidjeti da se manje slike do otprilike 500 x 500 veličine zapravo ne isplati prebacivati na GPU, ali zato sa sve većim slikama brzina računanja na CPU raste s veličinom slike dok na GPU jedva primijeti razlika za slike veličine od čak 5000 x 5000.



Slika 9.: Usporedba CPU i GPU izvedbe Canny detektora u ovisnosti o veličini slike



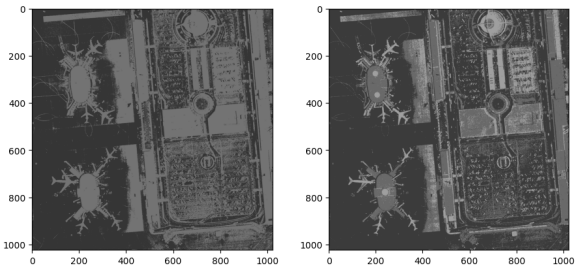
Slika 10.: Originalna slika dimenzija 1024 x 1024

4.5. Implementacija algoritma grupiranje s K srednjih vrijednosti za segmentaciju slike

Grupiranje s K srednjih vrijednosti ili K-means grupiranje (engl. K-means clustering) najjednostavniji je i najpoznatiji algoritam grupiranja. Algoritam ima složenost O(KnT), gdje je K broj grupa (engl. cluster), n broj podataka (u slučaju analize slika to je broj piksela), a T broj iteracija algoritma. U kodu je broj iteracija predstavljen varijablom *attempts*.

Ovaj postupak pripada segmentaciji grupiranjem (engl. clustering segmentation methods). K predstavlja broj (čvrstih) grupa u koje se pikseli grupiraju na temelju sličnosti. Svaka grupa ima srednju vrijednost (centroid) kojom je predstavljena.

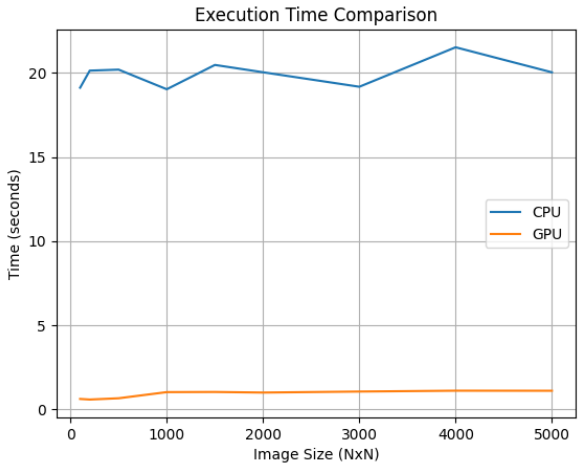
Algoritam započinje nasumičnim odabirom K centroida, nakon čega se izračunava udaljenost između svakog piksela i centroida pomoću njihovih HSV vrijednosti. Pikseli se dodjeljuju grupi najbližeg centroida. Zatim izračunava novu srednju vrijednost grupe i tu vrijednost postavlja kao centroid, nakon čega ponovno računa udaljenost svakog piksela do novo izračunatih centroida. Taj se postupak ponavlja sve dok se ne dogodi da ni jedan piksel nije promijenio grupu kojoj pripada nakon izračuna udaljenosti. Algoritam se ponavlja zadani broj puta T i odabire pokušaj s najuravnoteženijom varijacijom između grupa.



(a) CPU (20.013 s) (b) GPU (0.795 s)

Slika 11.: Rezultati K-means grupiranja nad slikom 10.

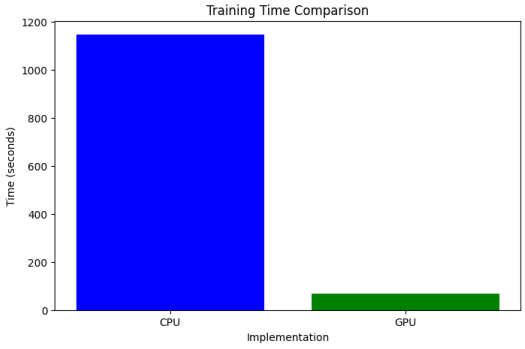
Rezultati grupiranja se vide na slikama 11., a vremena provođenja metode u odnosu na veličinu slika se mogu vidjeti na slici 12.. Iz rezultata mjerenja vidimo da je k-means grupiranje jako povoljna metoda za paralelizaciju. Različite jezgre GPU-a mogu biti zadužene za izračunavanje udaljenosti između jednog podatka i svih centroida i paralelno se mogu ažurirati centroide.



Slika 12.: Usporedba CPU i GPU izvedbe K-means grupiranja u ovisnosti o veličini slike

4.6. Implementacija neuronske mreže za klasifikaciju rukom pisanih znamenaka

U ovom poglavlju razmotrit ćemo implementaciju neuronske mreže za klasifikaciju rukom pisanih brojeva. [2] Obje implementacije napisane su u programskom jeziku Python. Verzija koja se izvodu na CPU zasniva se na biblioteci NumPy, dok se verzija koja se izvodi na GPU zasniva na biblioteci CuPy.
Za treniranje modela korišten je MNIST skup podataka, koji sadrži 60,000 slika rukom pisanih znamenaka. Svaka slika je veličine 28x28 piksela. Prilikom učitavanja skupa podataka, slike su povećane na veličinu 128x128 piksela jer je originalna veličina slika premala da bi se uočile prednosti izvođenja na grafičkim karticama. Naime, treniranje modela s originalnom veličinom slika bilo je sporije na grafičkoj kartici nego na procesoru. Iako se matrične operacije daleko brže izvode na grafičkim karticama, prenošenje podataka na grafičku karticu vremenski je skupo. U slučaju kada imamo malo podataka, vremenski je isplativije izvršiti operacije na procesoru nego prenositi podatke na grafičku karticu i natrag.
Model u ulaznom sloju ima 16,384 neurona. Skriveni sloj sadrži 30 neurona, dok izlazni sloj ima 10 neurona jer toliko imamo mogućih klasa. Za treniranje modela koristili smo stohastički gradijentni spust, dok smo kao aktivacijsku funkciju koristili sigmoidnu funkciju. Oba modela trenirali smo 10 epoha te su im točnosti na testnom skupu podataka bile približno jednake. Treniranje modela koji se izvršavao na procesoru je trajalo 1145 sekundi, dok je treniranje modela koji se izvršavao na grafičkoj kartici trajalo 67 sekundi. Grafička usporedba prikazana je na slici 13..
Vremenska ušteda je značajna. Verzija modela koja se izvršava na grafičkoj kartici također radi brže predikcije, te predikciju za istu sliku izvršava gotovo deset puta brže nego model koji se izvršava na procesoru.



Slika 13.: Prikaz usporedbe vremena treniranja modela koji se izvršavaju na procesoru i na grafičkoj kartici.

4.7. Ostale metode obrade slika

Na slici 14. uspoređena su vremena izvođenja na CPU i na GPU Fourierove transformacije, inverzne Fourierove transformacije, Valćine transformacije, konvolucije, uklanjanja šuma, registracija slika, računanja prosječne vrijednosti i standardne devijacije.

Operation	Time (s)	Device	Number of Images	Image Shape	Time / Image (s)
FFT	1.7714591026306 2	CPU	200	(512, 512, 3)	0.00886
IFFT	1.7662076950073 2	CPU	200	(512, 512, 3)	0.00883
Wavelet Transform	2.0714282989502	CPU	200	(512, 512, 3)	0.01036
Convolution	2.3117394447326 7	CPU	200	(512, 512, 3)	0.01156
NL-means Denoising	639.968256473541	CPU	2	(512, 512, 3)	319.98413
Image Registration	9.5524961948394 8	CPU	200	(512, 512, 3)	0.04776
Inpainting	1.2878184318542 5	CPU	5	(512, 512, 3)	0.25756
Mean and Std	0.6386420726776 12	CPU	200	(512, 512, 3)	0.00319
FFT	0.3778769969940 19	GPU	200	(512, 512, 3)	0.00189
IFFT	0.1591351032257 08	GPU	200	(512, 512, 3)	0.00080
Wavelet Transform	2.1821579933166 5	GPU	200	(512, 512, 3)	0.01091
Convolution	0.1889572143554 69	GPU	200	(512, 512, 3)	0.00094
NL-means Denoising	641.026808977127	GPU	2	(512, 512, 3)	320.51340
Image Registration	9.7183928489685 1	GPU	200	(512, 512, 3)	0.04859
Inpainting	1.3010134696960 5	GPU	5	(512, 512, 3)	0.26020
Mean and Std	0.2891852855968 237	GPU	200	(512, 512, 3)	0.00145

Slika 14.: Implementacija metoda za obradu slike. Za svaku metodu tablica prikazuje vrijeme izvođenja na CPU i na GPU.

5. Zaključak

Ovaj rad je detaljno analizirao razlike u performansama između centralnih procesorskih jedinica (CPU) i grafičkih procesorskih jedinica (GPU) prilikom izvršavanja različitih metoda obrade slike koristeći NVIDIA CUDA tehnologiju. Prikazane su brojne implementacije algoritama za obradu slike, kao i implementacija neuronske mreže za klasifikaciju rukom pisanih brojeva. U svim slučajevima, implementacije koje su se izvodile na GPU nadmašile su one koje su se izvodile na CPU u pogledu brzine izvršavanja.

S obzirom na to da nas u budućnosti očekuje daljnji napredak grafičkih procesora i popratnih tehnologija, gotovo je sigurno da će s vremenom postupci obrade slike biti nezamislivi bez grafičkih procesora. GPU-ovi su pokazali izuzetnu efikasnost, naročito kod obrade velikih skupova podataka i složenih algoritama, što ih čini neophodnim za napredne aplikacije u medicini, industriji, forenzici, autonomnim vozilima i drugim disciplinama.

Buduća istraživanja mogla bi biti usmjerena na analizu prijenosa podataka između CPU-a i GPU-a te na potencijalna ubrzanja tog neizbježnog dijela.

Literatura

[1] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[2] Michael Nielsen. Neural networks and deep learning, 2022.