



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

НОВИ САД

Департман за рачунарство и аутоматику

Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: Мја Гагић

Број индекса: 12579

Тема рада: Реформатирање слика у мултимедијалним системима и имплементација на таблет базираним платформама

Ментор рада: проф. др Миодраг Темеринац

Нови Сад, јун, 2012.



КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:			
Идентификациони број, ИБР:			
Тип документације, ТД:	Монографска документација		
Тип записа, ТЗ:	Текстуални штампани материјал		
Врста рада, ВР:	Завршни (Bachelor) рад		
Аутор, АУ:	Маја Гагић		
Ментор, МН:	Миодраг Темеринац		
Наслов рада, НР:	Реформатирање слика у мултимедијалним системима и имплементација на таблет базираним платформама		
Језик публикације, ЈП:	Српски / латиница		
Језик извода, ЈИ:	Српски		
Земља публиковања, ЗП:	Република Србија		
Уже географско подручје, УГП:	Војводина		
Година, ГО:	2012		
Издавач, ИЗ:	Ауторски репринт		
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6		
Физички опис рада, ФО: (поглавља/страница/цитата/табела/слика/графика/прилога)			
Научна област, НО:	Електротехника и рачунарство		
Научна дисциплина, НД:	Рачунарска техника		
Предметна одредница/Кључне речи, ПО:			
УДК			
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад		
Важна напомена, ВН:			
Извод, ИЗ:			
Датум прихватања теме, ДП:			
Датум одбране, ДО:			
Чланови комисије, КО:	Председник:		
	Члан:		
	Члан, ментор:	Потпис ментора	



KEY WORDS DOCUMENTATION

Accession number, ANO:		
Identification number, INO:		
Document type, DT:	Monographic publication	
Type of record, TR:	Textual printed material	
Contents code, CC:	Bachelor Thesis	
Author, AU:	Maja Gagić	
Mentor, MN:	Miodrag Temerinac	
Title, TI:	Image reformatting in multimedia systems and implementation on tablet based platforms.	
Language of text, LT:	Serbian	
Language of abstract, LA:	Serbian	
Country of publication, CP:	Republic of Serbia	
Locality of publication, LP:	Vojvodina	
Publication year, PY:	2012	
Publisher, PB:	Author's reprint	
Publication place, PP:	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, PD: (chapters/pages/ref./tables/pictures/graphs/appendices)		
Scientific field, SF:	Electrical Engineering	
Scientific discipline, SD:	Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, S/KW:		
UC		
Holding data, HD:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, N:		
Abstract, AB:		
Accepted by the Scientific Board on, ASB:		
Defended on, DE:		
Defended Board, DB:	President:	
	Member:	
	Member, Mentor:	



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



SADRŽAJ

1.	Uvod	1
2.	Analiza.....	2
2.1	Opis platforme	4
2.1.1	Programska podrška.....	4
2.1.2	Fizička arhitektura.....	4
3.	Algoritam.....	7
3.1	Unapređenje algoritma.....	11
3.2	Mere poređenja sa konkurencijom.....	16
3.2.1	PSNR.....	16
3.2.2	SSIM	16
3.2.3	Oštrina slike	17
3.3	Konkurentni algoritmi.....	17
3.3.1	Nearest neighbor	18
3.3.2	Bilinearna interpolacija	18
4.	Prilagođenje ciljnoj platformi.....	19
4.1	OpenMP	23
4.1.1	Rezultati	24
4.2	POSIX API.....	25
4.2.1	Podela obrade po linijama.....	26
4.2.2	Podela obrade po bloku linija.....	32
4.2.3	Podela obrade po bloku linija, korišćenjem prealocirane grupe niti.....	34
4.2.4	Uporedni rezultati svih POSIX modela paralelizacije	38
4.3	Celobrojna aritmetika.....	40

5.	Testno okruženje.....	42
6.	Zaključak i pravci daljeg razvoja.....	43
7.	Literatura	45



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



SPISAK SLIKA

Slika 1 - Struktura Sigma SMP8910 platforme	5
Slika 2 - Struktura MIPS32 1004K procesnog sistema	6
Slika 3 - Proces skaliranja za svaki piksel	8
Slika 4 - Domen interpolacije	8
Slika 5 - Četiri tipa ivica (a) vertikalna, (b) horizontalna, (c) JZ-SI, (d) SZ-JI	9
Slika 6 - Relativna udaljenost interpolirane tačke od susednih piksela	9
Slika 7 - Rezultat interpolacije bilinearnom metodom (a) i AISLED algoritmom (b) ...	10
Slika 8 - Vizualizacija razlike rezultata intrepolacije bilinearnom i AISLED metodom za sliku sa mnogo tekstura	11
Slika 9 - BDA proširenja AISLED algoritma koja uključuju precizniju detekciju ivica	12
Slika 10 - Među-koraci u procesu unapređenja detekcije ivica; Originalna slika (a), slika nakon primene Sobel operatora (b), slika nakon izbora regija od interesa (c)	13
Slika 11 - Uporedni rezultati interpolacije slike bilinearnom metodom i AISLED algoritmom pre i nakon unapređenja.....	14
Slika 12 - Uporedni rezultati nakon interpolacije slike bilinearnom metodom i AISLED algoritmom pre i nakon poboljšanja.....	14

Slika 13 - Slika interpolirana bilinearnom metodom (a) i AISLED tehnikom pre (b) i nakon (c) poboljšanja	15
Slika 14 - Uporedni rezultati nakon interpolacije slike bilinearnom metodom i AISLED algoritmom pre i nakon poboljšanja.....	15
Slika 15 - Detalj testne slike interpolirane AISLED metodom pre (a) i nakon (b) unapređenja	15
Slika 16 - Različita organizacija podataka u programskom jeziku C i MATLAB okruženju.....	20
Slika 17 - Uporedni prikaz ubrzanja dobijenog paralelizacijom korišćenjem OpenMP standarda sa i bez LUT težinskih faktora (Intel platforma)	24
Slika 18 - Uporedni prikaz vremena potrebnog za izvršenje algoritma na ciljnoj platformi korišćenjem OpenMP standarda za paralelizaciju (sa i bez LUT težinskih faktora)	25
Slika 19 - BDA obrade sa podelom posla nitima po linijama	27
Slika 20 - Način podele posla nitima u implementaciji obrade po linijama slike	28
Slika 21 - MSC algoritma sa dodelom posla nitima po linijama slike	29
Slika 22 - MSC dijagram u slučaju kada visina slike nije deljiva brojem niti.....	30
Slika 23 – Rezultati dobijeni na Intel platformi paralelizacijom obrade po linijama slike	31
Slika 24 - BDA obrade podelom posla na blokove linija	32
Slika 25 - Način podele posla nitima pri parelizaciji po bloku linija slike	33
Slika 26 - Vreme izvršavanja algoritma na Intel platformi za paralelizaciju obradom po bloku linija.....	33
Slika 27 - Ubrzanje postignuto na ciljnoj arhitekturi paralelizacijom obrade po bloku linija.....	34
Slika 28 - BDA paralelizacije obrade po bloku linija, korišćenjem preallocirane grupe niti.....	35
Slika 29 - MSC paralelizacije obrade po bloku linija, korišćenjem preallocirane grupe niti.....	36
Slika 30 - Uporedni rezultati dobijeni primenom OpenMp standarda i obrade po blokovima linija sa prelociranom grupom niti	37
Slika 31 - Rezultati obrade podelom posla na blokove linija, korišćenjem preallocirane grupe niti na Intel platformi	37
Slika 32 - Rezultati obrade podelom posla na blokove linija, korišćenjem preallocirane grupe niti na ciljnoj arhitekturi	38

Slika 33 - Udeo vremena potrebnog za stvaranje 16 niti u ukupnom vremenu izvršenja	38
Slika 34 - Uporedni prikaz ubrzanja dobijenih primenom POSIX niti za paralelizaciju na Intel platformi.....	39
Slika 35 - Uporedni prikaz ubrzanja dobijenih primenom POSIX niti za paralelizaciju na ciljnoj platform	39
Slika 36 - Rezultati obrada po bloku linija korišćenjem preallocirane grupe niti za izvršenje na Intel platformi	40
Slika 37 - Ubrzanje postignuto prelaskom na celobrojnu aritmetiku na Intel platform .	40
Slika 38 - Ubrzanje postignuto prelaskom na celobrojnu aritmetiku na ciljnoj platform	41
Slika 39 - Struktura testnog okruženja.....	42

SPISAK TABELA

Tabela 1 - Karakteristike korišćenog višejezgarnog procesora opšte namene	23
Tabela 2 - Karakteristike korišćenog višejezgarnog ugrađenog procesora	23
Tabela 3 - Ubrzanje postignuto paralelizacijom obrade po linijama na MIPS platformi	31

SKRAĆENICE

MATLAB:	<i>Matrix Laboratory</i> , Programsko okruženje za numeričke proračune i programski jezik četvrte generacije
MIPS:	<i>Microprocessor without Interlocked Pipeline Stages</i> , RISC arhitektura sa protočnom instrukcijskom strukturom
POSIX:	<i>Portable Operating System Interface</i> , Familija standarda
HW:	<i>Hardware</i> , Fizička arhitektura
SW:	<i>Software</i> , Programska podrška
AISLED:	<i>Adaptive Image Scaling based on Local Edge Directions</i> , Adaptivni algoritam za promenu frekvencije odabiranja slike, baziran na lokalizovanoj detekciji ivica
JIT:	<i>Just In Time</i> , Tehnika za optimizaciju izvršenja rutina u okviru virtualne mašine
RISC:	<i>Reduced Instruction Set Computer</i> , Računar sa smanjenim setom instrukcija
VoIP:	<i>Voice over Internet protocol</i> , Internet telefonija
STB:	<i>Set-Top Box</i> , Uređaj za prijem digitalnog TV signala
MC (system):	<i>Multi-Core system</i> , Višejezgarni sistem
Piksela:	<i>Pixel, Picture Element</i> , Element slike
NN:	<i>Nearest Neighbor</i> , Tehnika interpolacije ponavljanjem piksela
PSNR:	<i>Peak Signal-to-Noise Ratio</i> , Odnos signal-šum
SSIM:	<i>Structural Similarity</i> , Mera sličnosti između slika
MSE:	<i>Mean Squared Error</i> , Srednja kvadratna greška
FIR (filter):	<i>Finite Impulse Response</i> , Filtar sa konačnim impulsnim odzivom
LUT:	<i>Lookup Table</i> , Predefinisana tabela vrednosti

MEX:	<i>MATLAB Executable</i> , Sprega između MATLAB okruženja i podrutina pisanih u C/C++ jeziku
IDE:	<i>Integrated Development Environment</i> , Integrisano razvojno okruženje
LCC:	<i>Little C Compiler</i> , Kompaktni, višeplatformski C prevodilac
API:	<i>Application Programming Interface</i> , Programska sprega
DLL:	<i>Dynamic Link Library</i> , Deljena biblioteka
FPU:	<i>Floating-point Unit</i> , Jedinica za aritmetiku pokretnog zareza
BDA:	<i>Blok Dijagram Algoritma</i>
MSC:	<i>Message Sequence Chart</i> , Dijagram toka razmenjenih poruka u sistemu
DMIPS:	<i>Dhrystone Million Instructions Per Second</i> , Metrika za ocenu procesne moći ciljne arhitekture

1. Uvod

Cilj zadatka je bio razviti i implementirati metodu za interpolaciju slike pri proizvoljnem faktoru uvećanja ili umanjenja, uz očuvanje oštine.

Razvoj i verifikaciju je bilo potrebno vršiti nad standardnim testnim slikama, kao i uporediti algoritam sa već postojećim metodama za interpolaciju slike, koje su deo programskog paketa MATLAB [1].

Za implementaciju su preporučeni C ili Java programski jezici, kako bi kod bio fleksibilan za primenu na različitim platformama, a za verifikaciju neka od ugrađenih (engl. embedded) platformi (Android).

Implementaciju je trebalo prilagoditi savremenim mobilnim sistemima, tako da u potpunosti iskorišćava njihovu procesnu moć.

Nakon razvoja algoritma u MATLAB okruženju, za realizaciju je izabran C programski jezik. Radi postepenog i jednostavnijeg prelaska na C, iskorišćene su pogodnosti MEX datoteka, koje obezbeđuju spregu između MATLAB koda i podrutina pisanih u C/C++ ili Fortran jeziku.

Za ciljnu platformu, izabrana je Sigma SMP8910 sa MIPS32 1004K ugrađenom procesnom jedinicom, koja poseduje više jezgara sa mogućnošću paralelnog izvršavanja više niti, što je iskorišćeno upotreborom nekoliko standarda i modela za paralelizaciju, poput OpenMP [2] i POSIX [3].

2. Analiza

Razvoj tehnologije je doveo do toga da je moderan život nezamisliv bez računara, pametnih telefona, automatizovanih kućnih aparata i drugih elektronskih naprava koje nam olakšavaju svakodnevnicu, ili nam služe za razonodu, dok je internet učinio informacije dostupnim i olakšao komunikaciju među ljudima.

Multimedijalna komunikacija preko interneta i bežičnih mobilnih mreža dobija na sve većem značaju, dajući povoda proizvođačima da preplave tržište namenske elektronike najrazličitijim multimedijalnim sistemima. Tržište aparata i aplikacija za zabavu je oduvek bilo razvijeno, pa ne treba da čudi činjenica da televizija zauzima bitno mesto među potrošačkom elektronikom. Zahvaljujući intenzivnom razvoju televizije u poslednjoj deceniji i prelaska na tehnologije ravnih ekrana, došlo je do povećanja njihovih dimenzija, kao i rezolucije slike. Pored toga, televizijski servisi su počeli da se kombinuju sa internet i multimedijalnim sadržajima, što je dovelo do povećanja zahteva u pogledu prenosa digitalnih signala sa stanovišta količine podataka koji treba da se prenesu i obrade u realnom vremenu.

Digitalne slike i video zapisi sadrže veliki broj podataka, čiji obim raste sa napretkom tehnike, što dovodi do prepreka prilikom prenosa multimedijalnog sadržaja, koji mora biti prenet u odgovarajućem roku, kao i bez gubitka kvaliteta. Kako bi se zadovoljili zahtevi kvaliteta, brzine prenosa i prilagođenja prenetih podataka ciljnim sistemima, obradi slike se pridaje sve veći značaj.

Ušteda propusnog opsega mreže može se postići kodovanjem slike niske rezolucije na strani enkodera, koja se onda, na strani dekodera, pre samog prikazivanja krajnjem korisniku, uvećava do rezolucije modernih panela. Povećanje slike do željene visoke rezolucije vrši se nekom od tehnika interpolacije. Takođe, jedan od važnih primera rastuće potrebe za interpolacijom je i prikaz TV signala standardne definicije (engl. Standard Definition

Television, SD TV) na modernim panelima koji su mahom veće rezolucije a dosta često i različitih proporcija (prikaz standardnog 4:3 SD signala na 16:9 HD panelima).

Iako interpolacija skoro nikada nije glavni cilj, malobrojni su slučajevi u kojima manipulacija slikom ne uključuje ovu tehniku. Bilo da želimo da uveličamo sliku, smanjimo ili povećamo rezoluciju kako bi slika bila prilagođena rezoluciji ekrana na kom će biti prikazana, ili izvršimo bilo koju geometrijsku transformaciju slike (npr. rotiranje), neophodno je bar u nekoj meri primeniti interpolaciju.

Ukratko, interpolacija se svodi na određivanje vrednosti nedostajućih podataka, na osnovu onih već poznatih, i predstavlja vezu između diskretnog i kontinualnog domena. Glavne osobine bilo kog algoritma za obradu slike su kvalitet dobijene slike i kompleksnost proračuna. Osnovne metode interpolacije, poput interpolacije nultog reda (engl. Zero-order interpolation) ili bilinearne interpolacije, iako veoma jednostavne sa stanovišta proračuna i realizacije, nisu uvek dovoljne za ispunjenje savremenih zahteva jer često kao posledicu imaju pojavu neželjenih izobličenja kao što su zamućene ili nazubljene ivice u slici. Ova izobličenja značajno utiču na vizuelni ugođaj krajnjem korisniku, i kao takva predstavljaju važan kriterijum prilikom ocenjivanja karakteristika uređaja potrošačke elektronike.

U cilju smanjenja ovih štetnih efekata, razvijeni su brojni adaptivni algoritmi koji interpolaciju slike vrše na osnovu lokalnih osobina slike, ali za posledicu imaju veću kompleksnost obrade. Jedan od takvih algoritama je i adaptivni algoritam za promenu frekvencije odabiranja slike, baziran na lokalizovanoj detekciji ivica (engl. Adaptive Image Scaling based on Local Edge Directions, AISLED) [4], koji je ujedno i tema ovog rada.

Ovaj algoritam koristi četiri susedna elementa slike (engl. picture element, piksel) za otkrivanje lokalnih ivica. Ukoliko ivica postoji, pomoću osobina otkrivene ivice, određuje se adekvatan metod za interpolaciju. U suprotnom, koristi se bilinearna interpolacija.

Svako povećanje procesne moći modernih fizičkih arhitektura (engl. hardware, HW), mora biti praćeno unapređenjem programske podrške (engl. software, SW) korišćenjem novih programskih modela, a sa ciljem da se poveća stepen iskorišćenosti date platforme. Iako su u pitanju ugrađene arhitekture, moderna rešenja ovog tipa sustižu standardne desktop računare kako po pitanju procesne moći (takt procesora), stepena paralelizma (višejezgarne arhitekture), tako i po pitanju veličine i brzine memorijskih modula. Ovo omogućava da se kao ciljna platforma za napredne algoritme za obradu slike, kao što je AISLED, izabere i platforma koja pripada tržištu potrošačke elektronike.

2.1 Opis platforme

Kao ciljna platforma za rad, izabrana je MIPS procesorska arhitektura i Android operativni sistem. MIPS je izabran jer predstavlja dominantnu arhitekturu na tržištu ugrađenih, namenskih uređaja za domaćinstvo, prvenstveno digitalnih televizora. U poslednjih par godina, MIPS procesorska jezgra nude paralelni potencijal kroz implementaciju višejezgarnih (dvojezgarnih, četvorojezgarni, itd.) procesora koji, zadržavajući svoj ugrađeni karakter, nude mogućnost realizacije kompleksnih programskih rešenja kao što je napredna obrada slike u realnom vremenu.

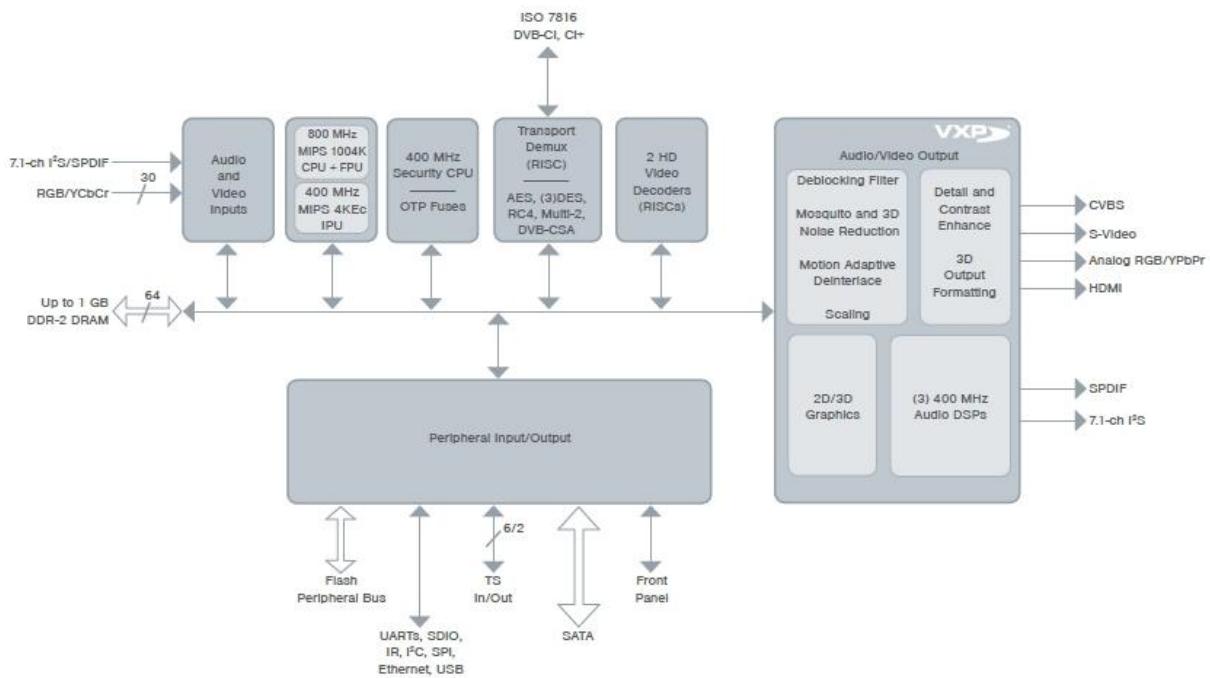
Android operativni sistem objedinjuje optimalno iskorišćenje platformskih resursa uz mogućnost izvršavanja modernih Java aplikacija korišćenjem proverenog jezgra Linux operativnog sistema i naprednih tehnika za optimizaciju izvršenja virtualnih mašina (engl. Just in time, JIT). Usled svojih skromnih HW zahteva i atraktivnosti aplikacija koje se izvršavaju na njemu, Android trenutno predstavlja najčešći izbor operativnog sistema na ugrađenim platformama i stoga predstavlja logičan izbor za ovaj rad.

2.1.1 Programska podrška

Android je platforma za mobilne uređaje razvijena od strane kompanije Google. Realizovan je u obliku SW steka koji uključuje jezgro operativnog sistema, srednji (engl. middleware) i aplikativni sloj programa. Osnovni razlog velikog interesovanja za Android je dostupnost izvornog koda. Samim tim, Android je pogodan za dalja poboljšanja i modifikacije, u skladu sa potrebama pojedinaca širom sveta. Ceo stek je, iako prvobitno namenjen uređajima zasnovanim na ARM (Advanced RISC Machine) arhitekturi, 2009. godine prilagođen i MIPS arhitekturi procesora, što je otvorilo mogućnosti za korišćenje u čitavom novom spektru uređaja, od televizora, VoIP telefona do digitalnih okvira za slike.

2.1.2 Fizička arhitektura

Za verifikaciju je odabrana Sigma SMP8910 platforma zbog MIPS32 1004K procesnog sistema na kom je zasnovana.

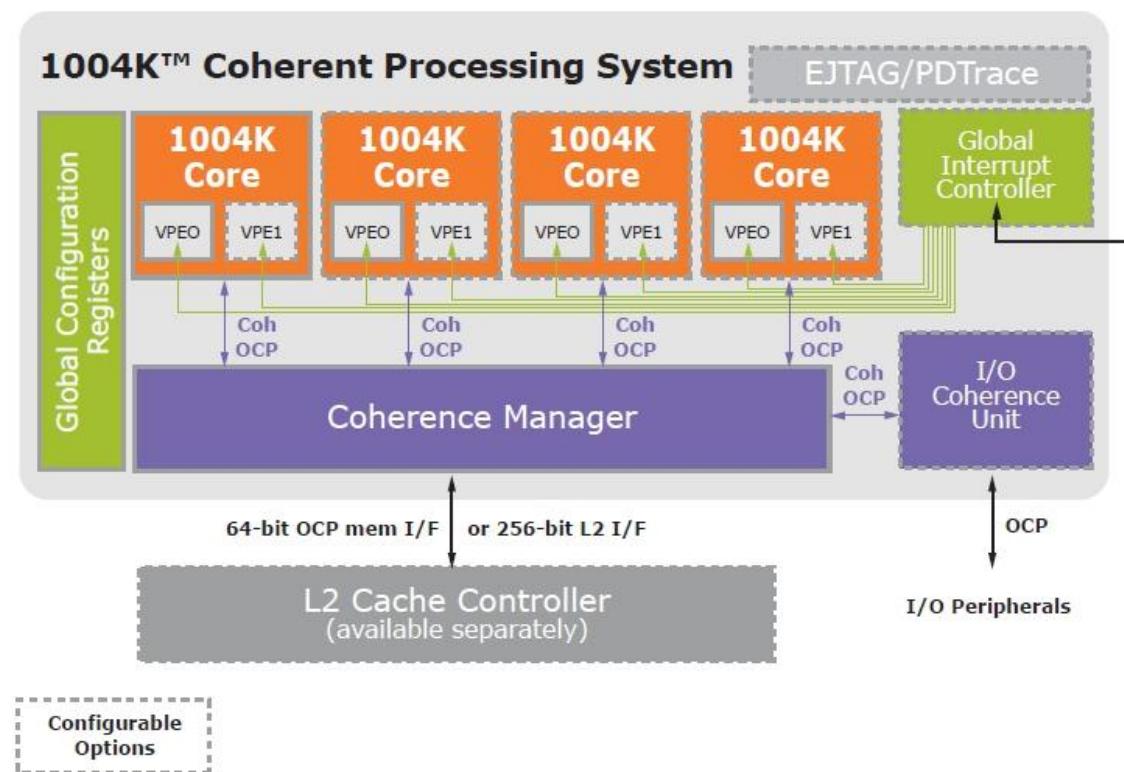


Slika 1 - Struktura Sigma SMP8910 platforme

MIPS arhitekture su arhitekture sa smanjenim setom instrukcija (engl. Reduced Instruction Set Computer, RISC) koje su zbog niske potrošnje i male dissipacije topline zauzele ključno mesto na tržištu namenskih sistema i osnova su sistema poput igračkih konzola, televizora i uređaja za prijem digitalnog TV signala (engl. Set-top Box, STB).

MIPS32 1004K se, od ostalih sličnih sistema, izdvaja po tome što je prva arhitektura sa više jezgara (engl. Multi-core, MC), od kojih svako poseduje podršku za istovremeno izvršavanje više niti, što mu omogućava da prevaziđe mogućnosti višejezgarnih sistema baziranih na jednonitnim procesorima. Ovaj sistem sadrži od jednog do četiri jezgra, povezanih jedinicom zaduženom za očuvanje usklađenosti skrivenih memorija svakog od njih. Pored toga, sadrži i rukovaoca globalnim prekidima, koji prihvata do 256 prekida i prosleđuje ih jezgrima ili pak nitima unutar jezgara.

Jezgra 1004K imaju 9-stepenu protočnu strukturu, koja omogućava obradu brzine i do 1.6 DMIPS/MHz po jezgru i pored celobrojne aritmetike, mogu podržavati i aritmetiku pokrednog zareza.



Slika 2 - Struktura MIPS32 1004K procesnog sistema

3. Algoritam

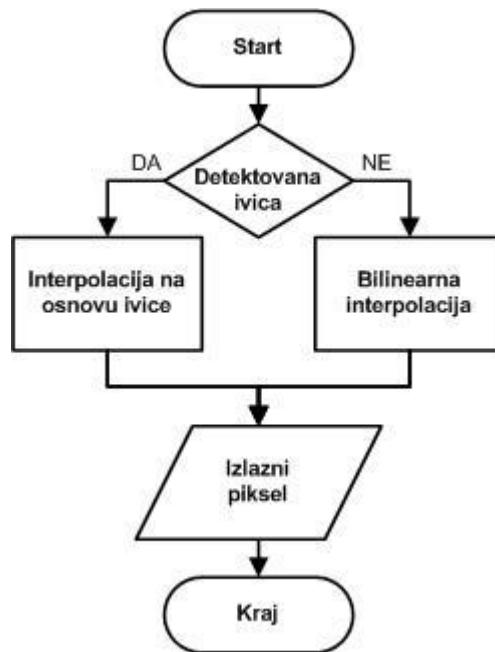
Interpolacione tehnike se mogu podeliti u dve glavne kategorije:

- Uobičajene metode interpolacije, koje koriste konstantne konvolucione kernele za celu sliku
- Adaptivne interpolacione metode, koje za interpoliranje koriste informacije o ivicama

U standardne metode spadaju i interpolacija ponavljanjem piksela (engl. Nearest Neighbor interpolation, NN) i bilinearna interpolacija, koje često ne daju rezultate potrebnog kvaliteta. Bilinearna interpolacija, na primer, zadržava niskofrekventne komponentne slike, ali nije u stanju da pojača visoke frekvencije i samim tim sačuva ivice pa dolazi do pojave preklapanja spektra (engl. aliasing), zamućenja ili drugih veštačkih izobličenja.

Mnogi adaptivni algoritmi su već razvijeni zarad otklanjanja ovih nepoželjnih pojava u slikama, a u cilju rešavanja problema oštine skalirane slike, razvijen je i AISLED algoritam, baziran na lokalizovanoj detekciji ivica.

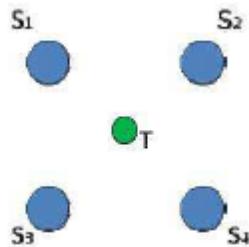
Algoritam koristi četiri susedna piksela za otkrivanje ivice. Ukoliko je lokalna ivica otkrivena, određuje se prikladan metod za interpolaciju, dok se u slučaju da ivica nije otkrivena, koristi bilinearna interpolacija. Slika 3 prikazuje proces skaliranja za svaki piksel.



Slika 3 - Proces skaliranja za svaki piksel

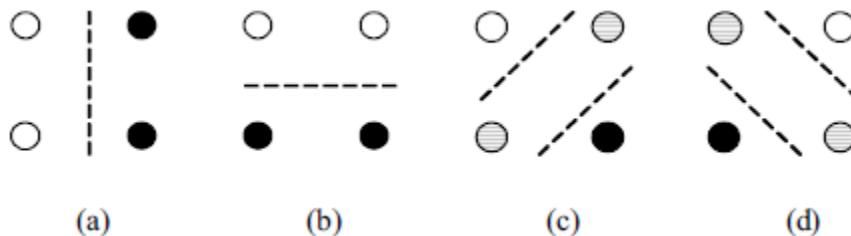
Prvi korak je detekcija ivice. Na početku, određe se razlike vrednosti između piksela oko obrađivanog piksela. Ako interpolirani piksel označimo sa T , a susedne piksele sa S_1, S_2, S_3 i S_4 , kao što prikazuje Slika 4, šest razlika su definisane kao:

$$\begin{aligned} D_1 &= |S_1 - S_2| & D_2 &= |S_1 - S_3| & D_3 &= |S_1 - S_4| \\ D_4 &= |S_2 - S_3| & D_5 &= |S_2 - S_4| & D_6 &= |S_3 - S_4| \end{aligned} \quad (1)$$



Slika 4 - Domen interpolacije

Nakon toga, određuje se minimalna vrednost iz skupa $D_1 - D_6$, koja služi za proveru da li ivica postoji ili ne. Sledeći korak je otkrivanje ivice i njena klasifikacija. Ukoliko ivica postoji, razvrstava se u jednu od četiri grupe, koje prikazuje Slika 5.



Slika 5 - Četiri tipa ivica (a) vertikalna, (b) horizontalna, (c) JZ-SI, (d) SZ-JI

U zavisnosti od vrednosti $D_1 - D_6$, i eksperimentalno određene konstante Th , koja predstavlja graničnu vrednost, određuje se da li je ivica prisutna ili ne. Menjanjem vrednosti Th , moguće je uticati na osetljivost algoritma na intenzitet ivice. Na primer, za prisustvo vertikalne ivice, moraju biti ispunjeni sledeći uslovi:

$$\min = D_2 \text{ i } D_3 > Th \text{ i } D_4 > Th \quad (2)$$

$$\min = D_5 \text{ i } D_3 > Th \text{ i } D_4 > Th \quad (3)$$

Ukoliko je ivica otkrivena, algoritam određuje odgovarajući način interpolacije na osnovu tipa date ivice. U suprotnom, piksel se interpolira korišćenjem bilinearne metode.

Za primer detektovane vertikalne ivice, težinski faktori w_1, w_2, w_3, w_4 računali bi se na sledeći način:

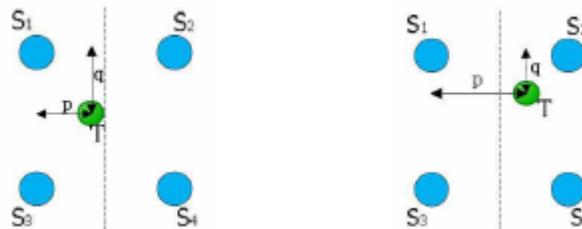
Ukoliko sa q i p označimo vrednosti u opsegu $[0, 1]$ koje predstavljaju koordinate relativne udaljenosti interpolirane tačke od četiri susedna piksela, kao što prikazuje Slika 6, udaljenost piksela od suseda računamo kao:

$$d_1 = q^2 + p^2 \quad (4)$$

$$d_2 = q^2 + (1-p)^2 \quad (5)$$

$$d_3 = (1-p)^2 + p^2 \quad (6)$$

$$d_4 = (1-q)^2 + (1-p)^2 \quad (7)$$



Slika 6 - Relativna udaljenost interpolirane tačke od susednih piksela

Pomoću toga dobijamo vrednosti težinskih faktora svakog susednog piksela za vertikalnu ivicu:

Ako je $p < 0.5$

$$w_1 = d_3 \quad (8)$$

$$w_2 = 0 \quad (9)$$

$$w_3 = d_1 \quad (10)$$

$$w_4 = 0 \quad (11)$$

U suprotnom:

$$w_1 = 0 \quad (12)$$

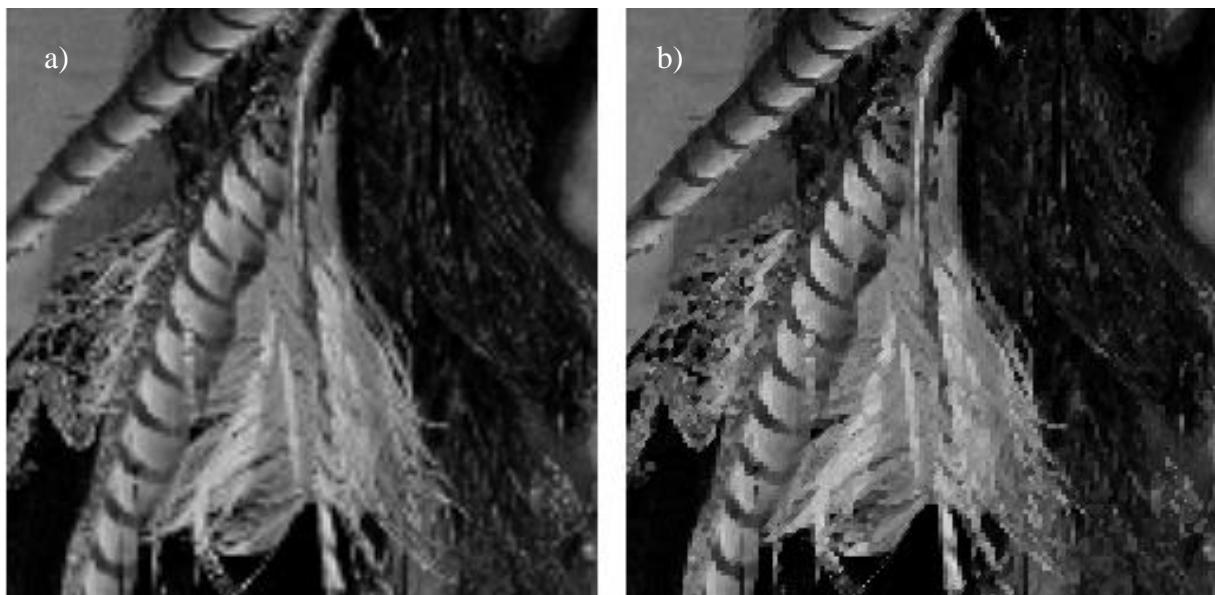
$$w_2 = d_4 \quad (13)$$

$$w_3 = 0 \quad (14)$$

$$w_4 = d_2 \quad (15)$$

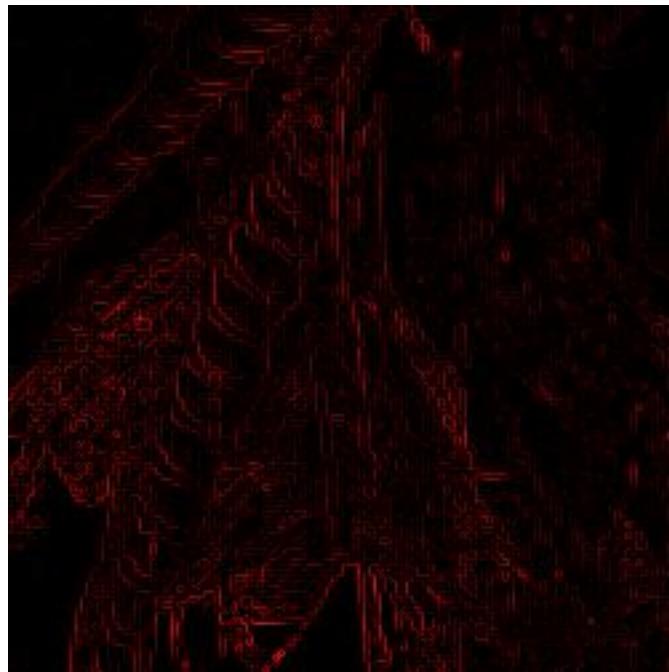
U cilju smanjenja kompleksnosti proračuna, algoritam koristi samo vrednosti četiri susedna piksela prilikom detekcije ivice, što se u eksperimentima pokazalo kao zadovoljavajuć kvalitet otkrivanja ivica. Više relativnih pravaca ivica (različitih uglova dijagonalnih ivica, a ne samo 45 i 135 stepeni, kao u trenutnoj implementaciji) bi se moglo detektovati ukoliko bi se koristio veći broj susednih piksela, ali bi se na taj način kompleksnost algoritma značajno povećala.

Mali broj referentnih piksela se pokazao kao negativna strana algoritma u slučajevima interpolacije slike koja u sebi sadrži mnogo tekstura. Tada blok od četiri piksela nije dovoljan za precizno razlikovanje prostornih ivica od tekture u slici, koju takođe karakterišu visokofrekventne komponente, ali uz visoku prostornu lokalizovanost. Slika 7 prikazuje uporedne rezultate interpolacije slike sa mnogo tekstura bilinearnom metodom i AISLED algoritmom.



Slika 7 - Rezultat interpolacije bilinearnom metodom (a) i AISLED algoritmom (b)

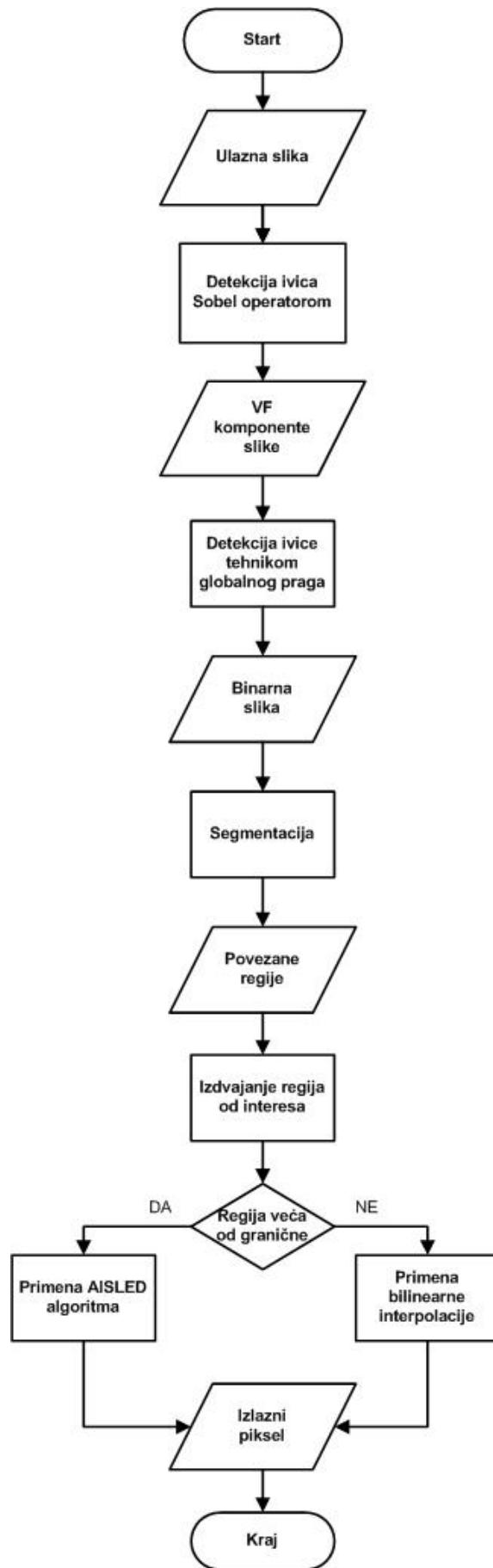
Slika 8 prikazuje mapu izmena AISLED metode u odnosu na bilinearnu metodu interpolacije. Sa slike se može videti da AISLED pojačava i deo slike koji predstavlja teksturu i samim tim unosi izobličenje u rezultujuću sliku.



Slika 8 - Vizualizacija razlike rezultata intrepolacije bilinearnom i AISLED metodom za sliku sa mnogo tekstura

3.1 Unapređenje algoritma

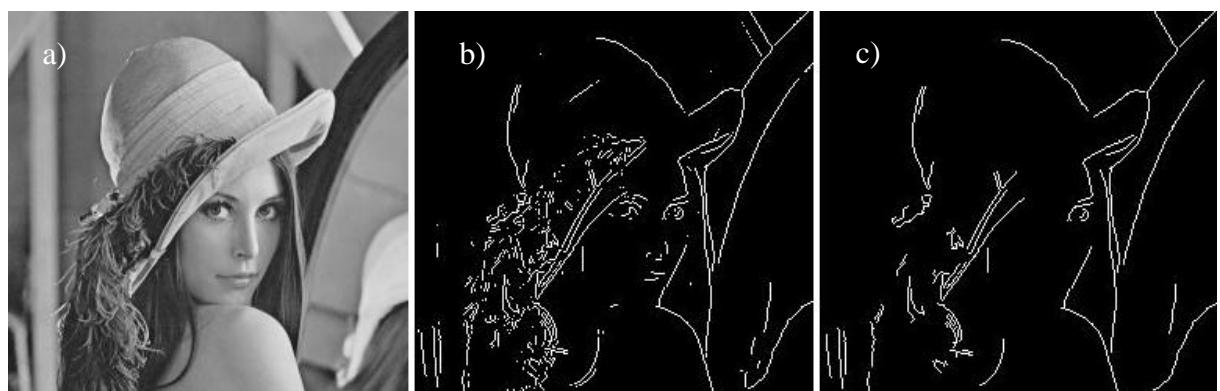
Kao najveća mana algoritma, pokazalo se nedovoljno precizno detektovanje ivica u slikama koje u sebi sadrže mnogo tekstura. Teksture, kao i ivice, karakterišu visokofrekventne komponente, ali prostorno lokalizovanje, pa blok od 2×2 susedna piksela nije dovoljan za precizno razlikovanje ivica od tekstura. Slika 9 prikazuje unapređen proces detekcije ivica, kojim je postignuto uspešnije razlikovanje tekstura od pravih ivica u slici.



Slika 9 - BDA proširenja AISLED algoritma koja uključuju precizniju detekciju ivica

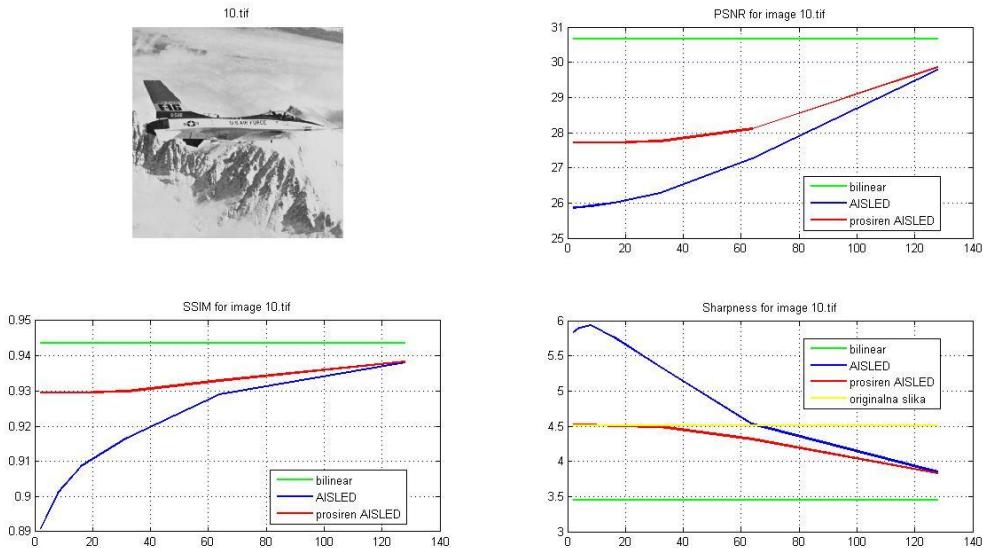
Nad ulaznom slikom, prvo se primenjuje Sobel operator, koji iz slike izdvaja visokofrekventne komponente. Nakon toga se, tehnikom globalnog praga izdvajaju samo potencijalne ivice iz slike, dok se ostale informacije zanemaruju. Time se dobija binarna slika koja ima vrednost nula za sve piksele osim onih koji čine detektovanu ivicu. Binarna slika prolazi kroz blok za segmentaciju, koji grupiše povezane ivice u regije, nakon čega sledi izdvajanje samo regija od interesa, tj. onih koje su veće od neke granične veličine (granica se može proizvoljno definisati). Na taj način, zanemaruju se male regije, stvorene pogrešnom detekcijom teksture kao ivice. Slika 10 prikazuje process unapređenja algoritma za detekciju ivica.

Dalji tok obrade je nepromenjen. Pikseli koji čine ivicu se obrađuju AISLED metodom, dok se ostatak slike interpolira bilinearnom interpolacijom.

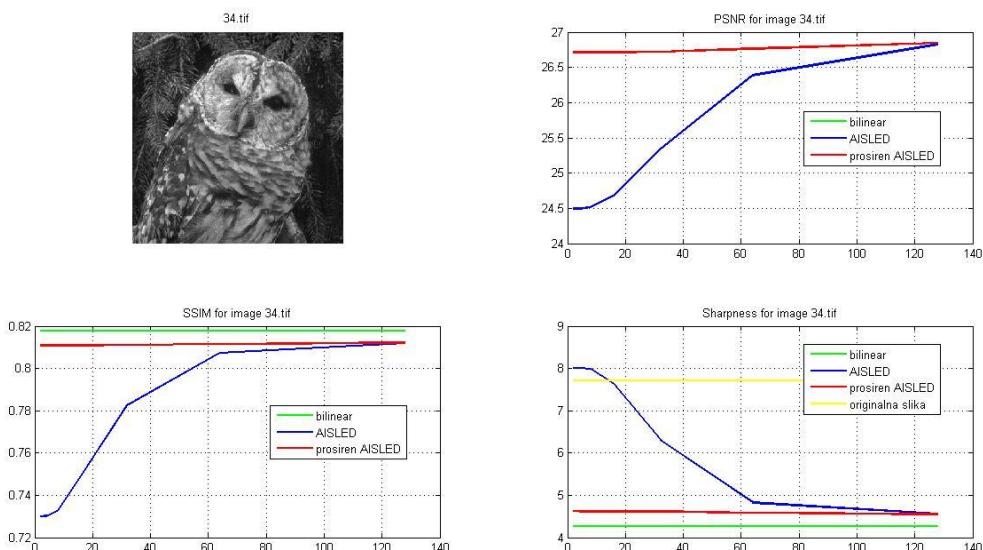


Slika 10 - Među-koraci u procesu unapređenja detekcije ivica; Originalna slika (a), slika nakon primene Sobel operatora (b), slika nakon izbora regija od interesa (c)

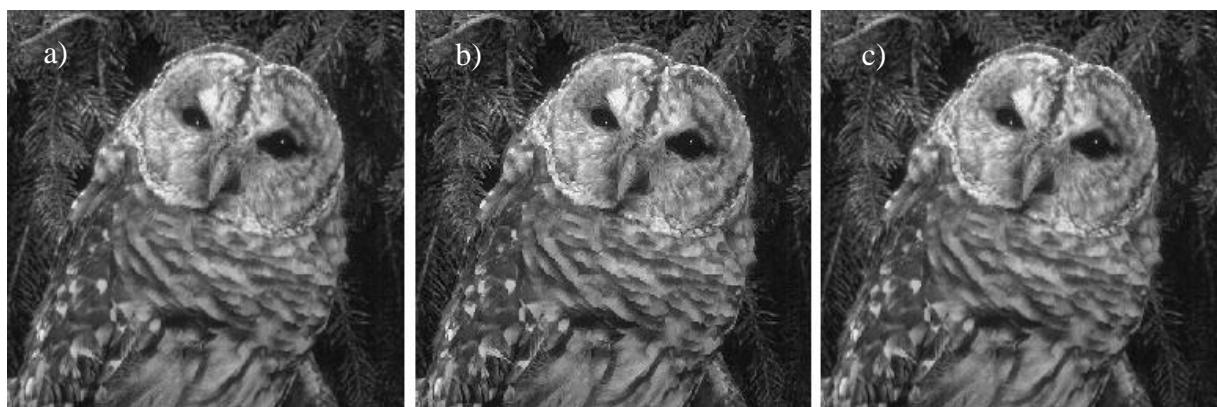
Slika 11, Slika 12 i Slika 14 prikazuju poboljšanja dobijena na ovaj način. Sa slike se može uočiti da bilinearna metoda zadržava najbolji PSNR rezultat pri čemu je oština slike značajno lošija u odnosu na AISLED metod, koji očekivano daje oštriju sliku, uz lošiji PSNR. Predložena poboljšanja AISLED algoritma čine da rezultujuća slika ima značajno bolji PSNR rezultat uz još uvek oštriju sliku u odnosu na bilinearnu metodu. Na svakom od grafika, apscisa prikazuje različite vrednosti Th, dok ordinata pokazuje odgovarajuću metriku (PSNR, SSIM, oština).



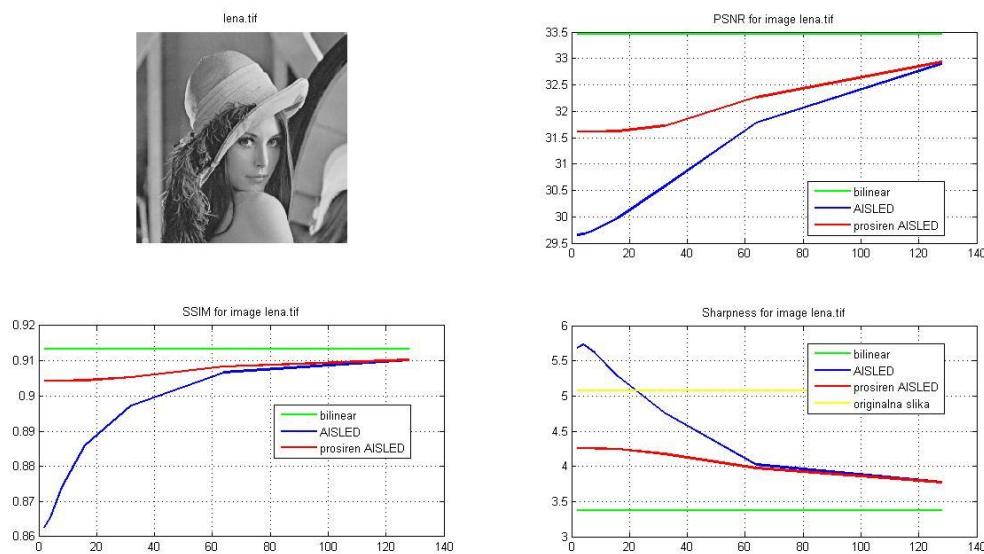
Slika 11 - Uporedni rezultati interpolacije slike bilinearnom metodom i AISLED algoritmom pre i nakon unapređenja



Slika 12 - Uporedni rezultati nakon interpolacije slike bilinearnom metodom i AISLED algoritmom pre i nakon poboljšanja



Slika 13 - Slika interpolirana bilinearnom metodom (a) i AISLED tehnikom pre (b) i nakon (c) poboljšanja



Slika 14 - Uporedni rezultati nakon interpolacije slike bilinearnom metodom i AISLED algoritmom pre i nakon poboljšanja

Slika 15 prikazuje detalj testne slike interpolirane AISLED metodom pre i nakon unapređenja algoritma za detekciju ivica. Vidi se da su, na ovaj način, uklonjena izobličenja u vidu nazubljenih ivica, kao i da slika izgleda prirodnije.



Slika 15 - Detalj testne slike interpolirane AISLED metodom pre (a) i nakon (b) unapređenja

3.2 Mere poređenja sa konkurencijom

Algoritam je poređen sa nekoliko uobičajenih algoritama korišćenjem sledećih objektivnih metrika:

- Odnos signal-šum (engl. Peak Signal-to-Noise Ratio, PSNR)
- Mera sličnosti između slika (engl. Structural Similarity, SSIM)
- Oštrina rezultujuće slike [5]

3.2.1 PSNR

PSNR je mera koja predstavlja odnos između najveće moguće vrednosti signala, i vrednosti šuma u signalu. Najčešće se koristi kao mera kvaliteta rekonstrukcije nakon kompresije sa gubicima, u kom slučaju signal predstavlja originalan podatak, a šum gubici usled kompresije.

Najlakše se definiše preko srednje kvadratne greške (engl. Mean Squared Error, MSE), koja je za dve monohromatske slike I i K , veličina $m \times n$, pri čemu se jedna smatra zašumljenom verzijom druge, definisana kao:

$$MSE = \frac{1}{m n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2 \quad (16)$$

PSNR se tada dobija kao:

$$\begin{aligned} PSNR &= 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \\ &= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20 \cdot \log_{10} (MAX_I) - 10 \cdot \log_{10} (MSE) \end{aligned} \quad (17)$$

3.2.2 SSIM

SSIM predstavlja metod procene sličnosti između dve slike, koji se u potpunosti oslanja na originalnu, nekompresovanu sliku bez grešaka. Razvijena je kao unapređenje mera poput PSNR i MSE, koje često nisu u skladu sa percepcijom ljudskog oka, i bazira se na pretpostavci da su pikseli međusobno čvrsto povezani, posebno kada su prostorno vrlo blizu.

Mera se računa na različitim prozorima u slici, a mera između prozora x i y , veličina $N \times N$ dobija se kao:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (18)$$

Gde su:

μ_x srednja vrednost x

μ_y srednja vrednost y

σ_x^2 varijansa x

σ_y^2 varijansa y

σ_{xy} kovarijansa x i y

$c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ L – dinamički opseg vrednosti piksela (obično $2^{\# \text{bita po pikselu}} - 1$)

$k_1 = 0.01$ i $k_2 = 0.03$

3.2.3 Oštrina slike

Odabrani algoritam za procenu oštine slike [5] primenjuje se u prostornom domenu, nad nekompresovanom slikom. Originalna slika se zamućuje pomoću dva različita Gausova šuma, koji služe kao model prirodnog šuma. Gausov šum je jedan tip filtra sa konačnim impulsnim odzivom (engl. Finite Impulse Response, FIR), koji koristi normalnu (Gausovu) raspodelu za računanje transformacije koja treba da se primeni na svaku tačku slike. Jednačina za N-dimenzionalnu Gausovu raspodelu je:

$$G(r) = \frac{1}{(2\pi\sigma^2)^{N/2}} e^{-r^2/(2\sigma^2)} \quad (19)$$

A za dvodimenzionalnu:

$$G(u, v) = \frac{1}{(2\pi\sigma^2)^2} e^{-(u^2+v^2)/(2\sigma^2)} \quad (20)$$

Gde r predstavlja prečnik zamućenja $r^2 = (u^2 + v^2)$, a σ standardnu devijaciju Gausove raspodele.

Rezultat primene ove jednačine na dvodimenzionalne podatke, kao što je slika, je površina čije su konture koncentrični krugovi sa Gausovom raspodelom u odnosu na centralnu tačku, tj. svaka tačka dobija usrednjenu vrednost tačaka u svojoj okolini. Na ovaj način se dobija vidno zamućenje jer se oštре ivice ovim postupkom ublažavaju.

Nakon dvostrukog zašumljenja Gausovim filtrom, zamućene slike se porede sa početnom. Ukoliko razlika između početne i zašumljene slike nije velika, zaključuje se da je početna slika najverovatnije već bila zamućena, tj. dodatno zamućenje na nju nije mnogo uticalo. Nasuprot tome, ako je razlika između originalne slike i slike dobijene nakon primene Gausovog šuma velika, dolazi se do zaključka da početna slika verovatno nije bila zamućena u velikoj meri. Rezultati merenja su vrednosti u opsegu [0.0, 1.0]. Ukoliko je izmereno slabo zamućenje, rezultat teži vrednosti 0.0, u suprotnom teži 1.0.

3.3 Konkurentni algoritmi

Za poređenje rezultata, odabrani su sledeći referentni algoritmi:

3.3.1 Nearest neighbor

Nearest-neighbor interpolacija je jednostavna metoda skaliranja slike, koja značaj daje samo tački najbližoj obrađivanom pikselu, zanemarujući ostale susede. Pošto je veoma brza, korisna je kada je brzina obrade jedini zahtev jer, iako zadržava oštrinu slike, daje nazubljene i neprirodne ivice.

3.3.2 Bilinearna interpolacija

Bilinearna interpolacija uzima u obzir vrednosti četiri dijagonalno susedna piksela, računa njihovu srednju vrednost, dajući značaj najbližoj tački, i dodeljuje je pikselu koji se obrađuje. Ova metoda daje mekše ivice, u odnosu na ivice dobijene nearest-neighbor algoritmom, ali uvodi brojne druge greške poput zamućenja (engl. blurring) ili efekta lažne ivice (engl. edge halo).

4. Prilagođenje ciljnoj platformi

Cilj zadatka je bio razviti i realizovati metodu za interpolaciju slike pri proizvoljnom faktoru uvećanja ili umanjenja, uz očuvanje oštine. Pošto je u pitanju algoritam za obradu slike, prirodan izbor za njegov razvoj i verifikaciju bio je MATLAB programski paket.

MATLAB je programsko okruženje namenjeno numeričkim proračunima, razvoju algoritama, analizi podataka i njihovoj vizualizaciji, a lakoća rukovanja čak i višedimenzionalnim matricama je ono što ga čini pravim izborom za razvoj jednog ovakvog algoritma. Programsко rešenje realizovano u MATLAB okruženju se sastoji od ulaznog modula, koji priprema sliku za obradu, i funkcionalnog modula, koji vrši obradu na bazi piksela.

Prvo prilagođenje algoritma je bio prelazak na korišćenje predefinisane tabele vrednosti težinskih faktora w_i (engl. Lookup Table, LUT), umesto njihovog računanja prilikom obrade svakog zasebnog piksela. Ovaj metod je doveo do ubrzanja algoritma u prvim fazama razvoja, dok u kasnijim rešenjima poboljšanja nije bilo, ili se čak brzina obrade smanjila. Bez obzira na to, sva rešenja su testirana u obe varijante radi prikupljanja potpunijih podataka.

Iako MATLAB ima prednosti u procesu razvoja algoritma, za implementaciju je bilo neophodno preći na C programski jezik jer je, pored unapređenja samog algoritma, cilj bio i njegovo prilagođenje za rad na platformi sa ograničenim resursima (MIPS). U tu svrhu, iskorišćene su prednosti MEX datoteka, koje su omogućile postepen prelazak na C, uz brzu i jednostavnu verifikaciju direktno iz MATLAB okruženja. MEX datoteke su dinamički povezane podrutine, koje MATLAB interpreter automatski učitava i izvršava, i predstavljaju spregu između MATLAB koda i podrutina pisanih u C/C++ programskom jeziku. Omogućavaju pozivanje i izvršavanje C/C++ funkcija iz MATLAB okruženja, kao da su u pitanju njegove ugrađene funkcije.

Najveći značaj MEX datoteka ogleda se u ubrzanju izvršavanja algoritma nakon izmeštanja izvršenja petlji izvan MATLAB interpretera, kao i u olakšanom otkrivanju i uklanjanju grešaka, usled deljenja koda na manje, nezavisne celine. Sa druge strane, njihova mana je potreba za realizacijom specifične funkcije, koja suštinski predstavlja MEX verziju `main` funkcije u C/C++ jeziku, koja povezuje MATLAB sa spoljašnjim programom i rukuje prenosom podataka između njih.

Deklaracija MEX specifične funkcije:

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
```

Glavna MEX funkcija sve MATLAB promenljive prenosi kao `mxArray` pa je potrebno voditi računa o konverziji tipa prilikom čitanja podataka iz memorije, kao i prilikom upisivanja. Dodatno, za razliku od matrica u C/C++ jeziku, MATLAB matrice su u memoriji smeštene po kolonama pa je, prilikom rukovanja podacima, neophodno voditi računa i o tome. Slika 16 prikazuje razliku između MATLAB i C/C++ načina čuvanja podataka u memoriji na primeru dvodimenzionalne matrice.

1	2	3
4	5	6
7	8	9

MATLAB

1	4	7	2	5	8	3	6	9
C								
1	2	3	4	5	6	7	8	9

Slika 16 - Različita organizacija podataka u programskom jeziku C i MATLAB okruženju

I pored nekolicine otežavajućih okolnosti, rukovanje MEX datotekama je prilično jednostavno i višestruko korisno. Ne sme se zanemariti ni činjenica da ih je moguće koristiti i pod Linux i pod Windows operativnim sistemom što omogućava verifikaciju algoritma u različitim orkuženjima. U našem slučaju, za ovaj među-korak, izabran je Windows operativni sistem i Microsoft Visual Studio 2008 (VS9) integrисано razvojno okruženje (engl. Integrated Development Environment, IDE). Pored toga što je, zbog korišćenja C++ sintakse prilikom realizacije algoritma, za prevodenje MEX datoteka bilo potrebno koristiti neki drugi prevodilac umesto LCC prevodioca, ugrađenog u MATLAB, grafičko okruženje VS9

omogućava olakšan pristup otkrivanju i otklanjanju grešaka, što je u ovom prelaznom koraku bilo od velike važnosti. Iz tog razloga, VS9 je poslužio i kao početno okruženje za paralelizaciju algoritma u cilju maksimalnog iskorišćenja fizičke arhitekture ciljne platforme.

Za ciljnu platformu, izabran je Sigma SMP8910 STB sa MIPS32 1004K procesnim sistemom, koji poseduje dva jezgra od kojih svaki ima mogućnost izvršavanja dve niti odjednom. Višejezgarni sistemi se, zbog toga što nude veću učinkovitost uz manju potrošnju, sve češće koriste u ugrađenim i namenskim sistemima, omogućujući im potpuno nove primene pa se fokus sa projektovanja fizičke arhitekture prebacuje na razvoj programske podrške koja, zbog mogućnosti realizacije više funkcija na istom uređaju, postaje sve kompleksnija.

Kao prvi korak u prilagođenju algoritma paralelnoj obradi, iskorišćen je industrijski standard OpenMP, koji predstavlja programsku spregu (engl. Application Programming Interface, API) namenjenju primeni na višeprocesorskim platformama sa deljenom memorijom. OpenMP podržava realizaciju programa u C/C++ i Fortran jezicima na većini procesorskih arhitektura, kao i većini operativnih sistema. Sastoji se od skupa direktiva namenjenih prevodiocu, biblioteka sa podrutinama kao i sistemskih konzolnih promenljivih (engl. Environment variable), koji utiču na ponašanje programa prilikom izvršenja.

Jednostavnost realizacije paralelizma je jedna od glavnih prednosti OpenMP standarda. Raščlanjivanje koda na paralelne delove je prepusteno prevodiocu, a uglavnom nije neophodno voditi računa ni o detaljima sinhronizacije niti. Nisu potrebne drastične izmene u kodu i moguće je postepeno paralelizovati deo po deo sekvencijalnog koda, dodavanjem direktiva ispred željenog dela, što omogućava jednostavniju verifikaciju paralelnog rešenja. Posebna korist je to što nema potrebe za odvojenim kodom za sekvencijalnu i paralelnu verziju rešenja, jer se ovakvo, paralelno, rešenje može prevoditi i prevodiocem koji ne podržava OpenMP standard, čime će pretprocesorske direktive biti zanemarene i dobiće se funkcionalan sekvencijalan kod. Izmenom sistemskih konzolnih promenljivih se, bez potrebe za ponovnim prevođenjem celog programa, može uticati na način izvršavanja, pa se tako, npr. postavljanjem promenljive OMP_NUM_THREADS na 1, paralelizam može isključiti, dok se postavljanjem na broj veći od jedinice zadaje broj niti koje treba uposlitи tokom izvršenja programa.

Pošto je realizacija paralelizacije i rukovanja nitima sakrivena od programera, postoji rizik od pojave nekorektnog izvršavanja programa usled nekontrolisanog nadmetanja niti za resurse (engl. Race conditions) koje bi bilo teško ukloniti. Dodatne mane su to što je standard trenutno efikasan samo na višeprocesorskim platformama sa deljenom memorijom, kao i to što je za prevođenje paralelizovanog koda neophodan prevodilac koji podržava OpenMP.

I pored svih prednosti koje nudi OpenMP i zavidnog nivoa paralelizacije koji se njime može postići, algoritam je paralelizovan korišćenjem i drugog popularnog industrijskog standarda za paralelizaciju – POSIX Threads (PThreads), koji definiše programsku spregu za stvaranje i rukovanje nitima. Ovime je, prvenstveno, htelo da se otkrije nešto više o načinu na koji OpenMP standard funkcioniše, kao i da li se korišćenjem POSIX niti može postići bolje iskorišćenje arhitekture koju nudi ciljna platforma.

Iako postoji više sličnih standarda, PThreads je izabran zbog toga što ciljna platforma ima podršku za distribucije Linux operativnog sistema prilagođenog MIPS arhitekturama. Za razliku od OpenMP, koji nudi jednostavnost i visok nivo apstrakcije pri paralelizaciji, PThreads zahteva veći napor pri pisanju koda, ali obezbeđuje apsolutnu kontrolu nad izvršavanjem paralelizovanih delova. U početnim fazama realizacije pristupa sa nitima, korišćeno je PThreads proširenje za Windows operativni sistem u vidu deljene biblioteke (engl. Dynamic Link Library, DLL), kako bi se razvoj vršio u Visual Studio okruženju i pojednostavio proces otklanjanja grešaka. Zarad lakšeg testiranja implementacija koje koriste POSIX API, nastavljeno je korišćenje sistemskih konzolnih promenljivih OpenMP standarda, pa se broj niti koje treba da budu uposlene na obradi podataka definiše se preko OMP_NUM_THREADS sistemske konzolne promenljive. Na taj način, jedna implementacija se može pokrenuti vise puta i testirati za različiti broj niti, bez ponovnog prevođenja koda.

Kao poslednji korak u optimizovanju algoritma, izvršen je prelazak sa aritmetike pokretnog zareza na celobrojnu aritmetiku. U aritmetici pokretnog zareza, koja služi za predstavljanje i manipulaciju racionalnim brojevima, brojevi su predstavljeni pomoću mantise i eksponenta (npr. $A \times 2^B$, gde je A mantisa, a B eksponent). Nasuprot tome, celobrojna aritmetika služi za rukovanje pozitivnim i negativnim celobrojnim vrednostima. U celobrojnoj aritmetici, mogu se predstaviti i razlomljeni brojevi, ali se oni, skaliranjem, pretvaraju u celobrojne vrednosti, što može dovesti do gubitka preciznosti i netačne reprezentacije broja.

Iako MIPS32 1004K ugrađeni sistem, koji je odabran za verifikaciju, poseduje jedinicu za aritmetiku pokretnog zareza (engl. Floating-Point Unit, FPU), zbog kašnjenja koje se javlja prilikom izvršenja ovakvih instrukcija, usled potrebe da se podaci prenose do izdvojenog koprocesora namenjenog njihovoј obradi, ali i nižeg radnog takta ovog koprocesora, bolji učinak je dobijen nakon prelaska na celobrojnu aritmetiku.

Verifikacija je vršena nad slikama u YUV domenu, veličine 512x512 piksela. Tabela 1 i Tabela 2 sadrže karakteristike opštenamenskog i ugrađenog procesora koji su korišćeni prilikom implementacije i testiranja. Sva merenja na Intel platformi vršena su u 400, a na Sigma8910 u 20 iteracija, kako bi se dobilo preciznije vreme potrebno za izvršenje algoritma na jednoj slici.

Tabela 1 - Karakteristike korišćenog višejezgarnog procesora opšte namene

Informacija	Procesor opšte namene
Ime	Intel (compute server, ICS)
Oznaka	Xeon
Broj fizičkih procesora	4
Broj fizičkih jezgara (po procesoru)	2
Broj logičkih jezgara (po procesoru)	4
Ukupan broj logičkih jezgara	16
Takt procesorskog jezgra	3200 MHz
Veličina skrivene memorije (L2/L3)	1024KB/8192KB

Tabela 2 - Karakteristike korišćenog višejezgarnog ugrađenog procesora

Informacija	Ugrađeni procesor
Ime	Sigma 8910
Oznaka	MIPS 1004Kf
Broj fizičkih procesora	1
Broj fizičkih jezgara (po procesoru)	2
Broj logičkih jezgara (po procesoru)	4
Ukupan broj logičkih jezgara	4
Takt procesorskog jezgra	800 MHz
Veličina skrivene memorije (L1/L2)	64KB/512KB

4.1 OpenMP

OpenMP standard je, zbog pojednostavljenog pristupa koji nudi, bio početna tačka u paralelizaciji algoritma. Najveće povećanje učinkovitosti OpenMP postiže prilikom paralelizacije na nivou petlji, što se pokazalo kao idealna osobina za paralelizaciju algoritma za obradu slike, koji svoju obradu vrši nad svakim pikselom u slici.

Korišćenjem preprocesorske direktive `#pragma omp parallel for` označava se deo koda koji je potrebno podeliti među nitima. Da bi se izbeglo nekontrolisano nadmetanje niti za resurse, direktivi se dodaje i atribut `private` koji označava da su ti podaci u okviru paralelnog regiona privatni za svaku nit, tj. da će svaka nit imati svoju kopiju podatka i koristiti je kao lokalnu promenljivu.

```

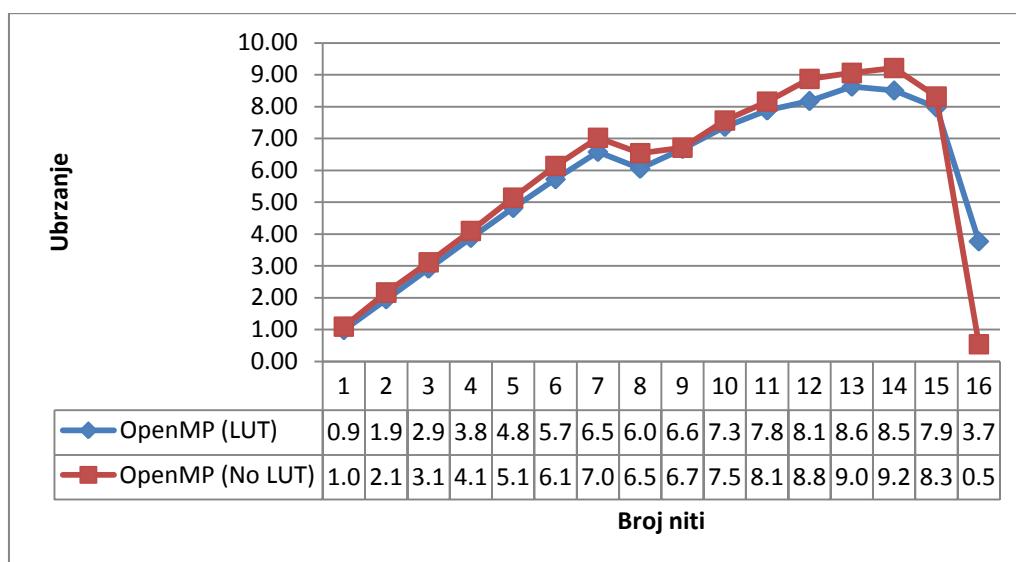
#pragma omp parallel for private(y, dy, hi, yo)
for(yo = 0; yo < hOut; yo++) {
    y = ((double)yo)/scaleFactV;
    hi = (int)floor(y);
    dy = y-hi;

    int xo, wi;
    double x, dx;
    for(xo = 0; xo < wOut; xo++) {
        x = ((double)xo)/scaleFactH;
        wi = (int)floor(x);
        dx = x-wi;
        // block processing
        interpolateNew2(&inputImg[hi*wIn + wi], wIn, dy, dx, threshold, xStep,
                        yStep, lut, &outputImg[yo*wOut + xo]);
    }
}

```

4.1.1 Rezultati

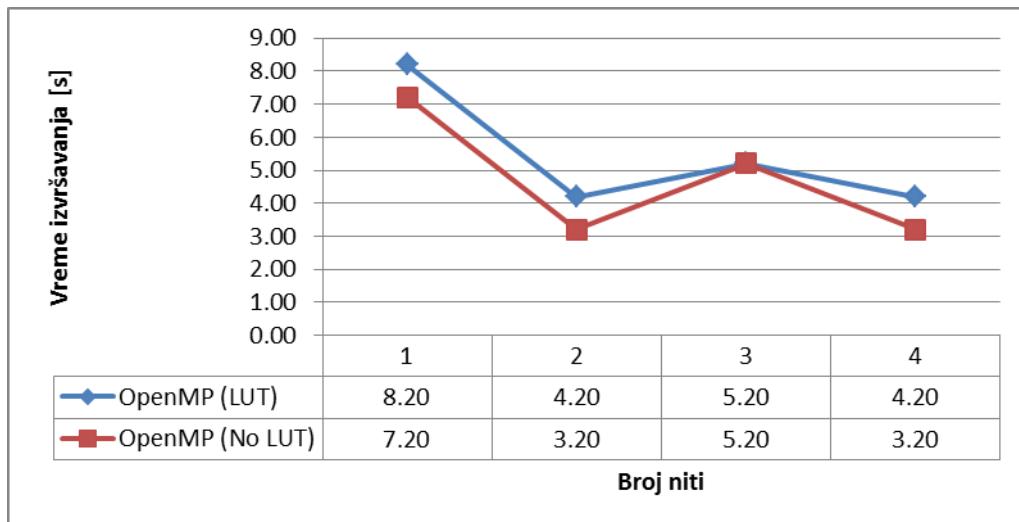
Slika 17 prikazuje ubrzanje postignuto na Intel platformi paralelizacijom OpenMP standardom, sa i bez tabele težinskih faktora. Vidi se da sa povećanjem broja niti do sedam, ubrzanje raste skoro linearno, nakon čega, na osam niti, neznatno opadne, da bi nastavilo skoro linearno da raste. Razlog za to je osam fizičkih jezgara Intel platforme. Do osam niti, svaka nit se izvršava na posebnom jezgru, nezavisno od drugih i iskorišćenje resursa je maksimalno. Kada broj niti pređe osam, počinju da se koriste logička jezgra i deo procesorskog vremena se troši na sinhronizaciju dve niti, koje rade na istom fizičkom jezgru.



Slika 17 - Uporedni prikaz ubrzanja dobijenog paralelizacijom korišćenjem OpenMP standarda sa i bez LUT težinskih faktora (Intel platforma)

OpenMP paralelizacija pokazuje najbolje rezultate za 14 niti, nakon čega učinak drastično opada. Paralelizacija korišćenjem maksimalnog broja dostupnih niti, u slučaju kada nije korišćena LUT, dovela je do usporenja u odnosu na sekvensijalno izvršenje što se može objasniti trošenjem procesorskog vremena na usklađivanje rada velikog broja niti.

Kao i na Intel platformi, i na ciljnoj arhitekturi se prelazak sa nezavisnog izvršavanja na odvojenim jezgrima na izvršavanje po dve niti na jednom fizičkom jezgru odrazilo na vreme izvršavanja algoritma. Slika 18 pokazuje izmerena vremena izvršenja na Sigma 9810 platformi.



Slika 18 - Uporedni prikaz vremena potrebnog za izvršenje algoritma na ciljnoj platformi korišćenjem OpenMP standarda za paralelizaciju (sa i bez LUT težinskih faktora)

4.2 POSIX API

PThreads je izabran zbog toga što ciljna arhitektura ima podršku za distribucije Linux operativnog sistema prilagođenog MIPS arhitekturama. Za razliku od OpenMP, koji nudi jednostavnost i visok nivo apstrakcije pri paralelizaciji, PThreads zahteva veći napor pri pisanju koda, ali obezbeđuje suptilniju kontrolu nad izvršavanjem paralelizovanih delova. Zarad lakšeg testiranja implementacija koje koriste POSIX programske spregu, nastavljeno je korišćenje sistemskih konzolnih promenljivih OpenMP standarda, pa se broj niti koje treba da budu uposlene na obradi podataka definiše se preko OMP_NUM_THREADS sistemske konzolne promenljive. Na taj način, jedna implementacija se može pokrenuti više puta i testirati za različiti broj niti, bez ponovnog prevođenja koda.

Paralelizacija korišćenjem POSIX niti tekla je u nekoliko faza i implementirana su četiri modela paralelizacije:

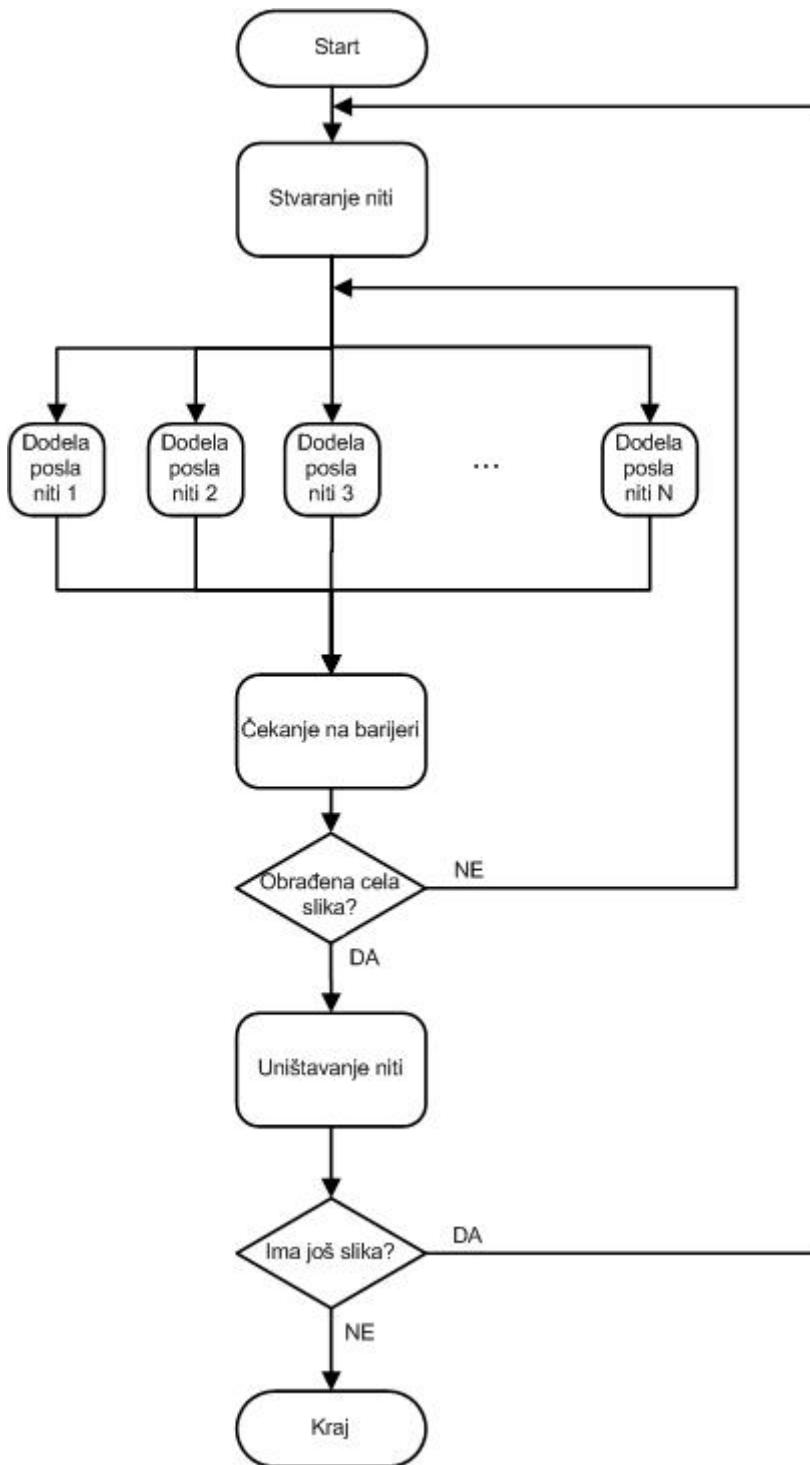
- Paralelizacija podelom obrade po linijama slike
- Paralelizacija podelom obrade po bloku linija slike

- Paralelizacija podelom obrade po bloku linija slike, uz korišćenje prealocirane grupe niti
- Paralelizacija podelom obrade po bloku linija slike, uz korišćenje prealocirane grupe niti i celobrojne aritmetike

Poslednji model ne predstavlja unapređenje paralelnog modela korišćenjem POSIX standarda (korišćen je isti paralelni model kao u trećoj fazi implementacije), već je optimizovan originalni algoritam, zamenom aritmetike pokretnog zareza celobrojnom aritmetikom.

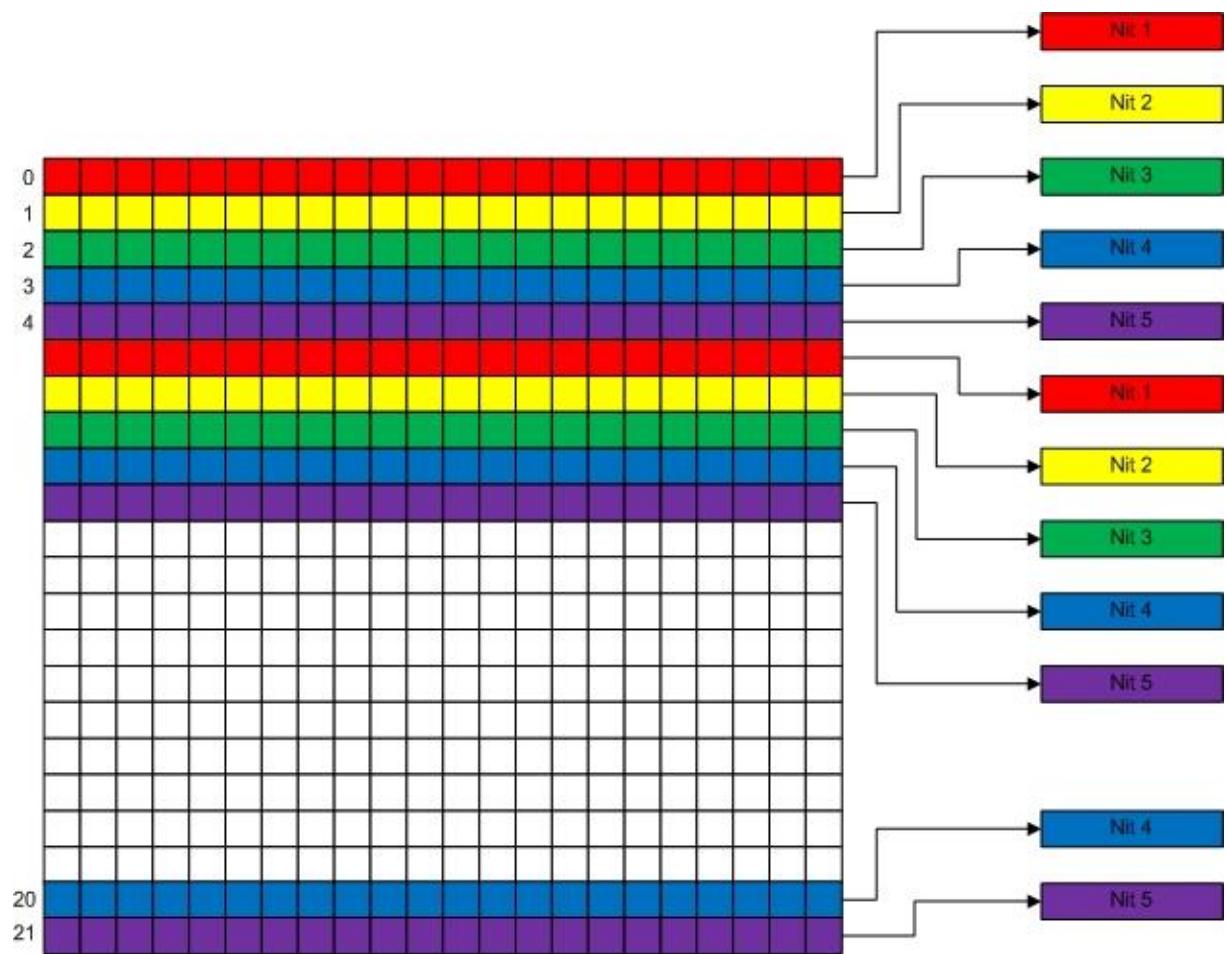
4.2.1 Podela obrade po linijama

Prilikom obrade po linijama, svakoj niti se dodeljuje samo po jedna linija slike po ciklusu obrade. Nakon što sve niti završe obradu svoje linije, dobijaju novu i proces se ponavlja sve dok se ne obradi cela slika. Slika 19 prikazuje blok dijagram algoritma (BDA) realizacije paralelne obrade podelom posla po linijama slike.

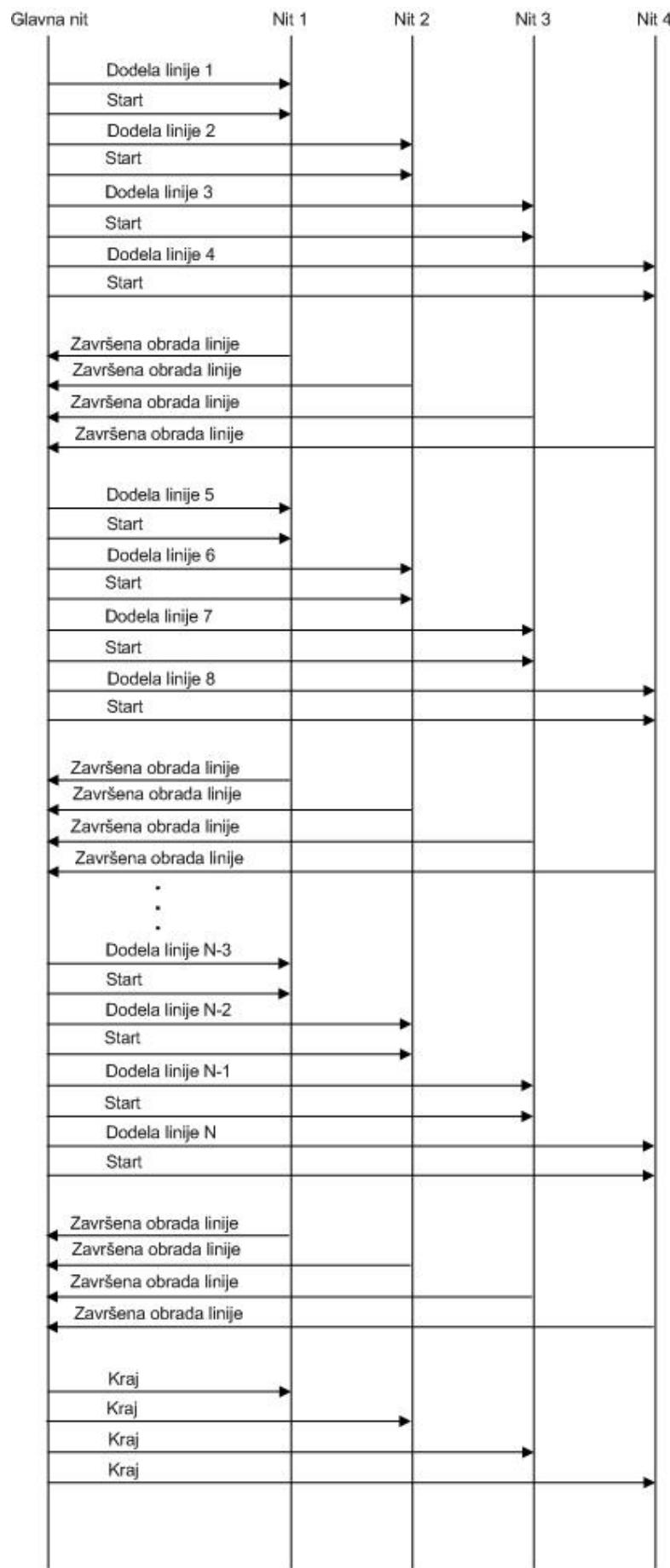


Slika 19 - BDA obrade sa podelom posla nitima po linijama

Pre početka same obrade, glavna nit stvara sve niti radilice, koje onda čekaju signal za početak obrade. Ukoliko je preostali broj redova slike veći od broja niti, svaka nit dobija po jedan red i signal da krene sa radom. U suprotnom, nitima za koje nema podataka glavna nit signalizira da preskoče obradu i sačekaju ostale niti da završe, dok ostalim dodeljuje po jedan red slike i signalizira im da počnu sa obradom. Sve radilice se uništavaju nakon obrade cele slike. Slika 20 prikazuje način dodele podataka nitima, a Slika 21 dijagram toka poruka (engl. Message Sequence Chart, MSC).

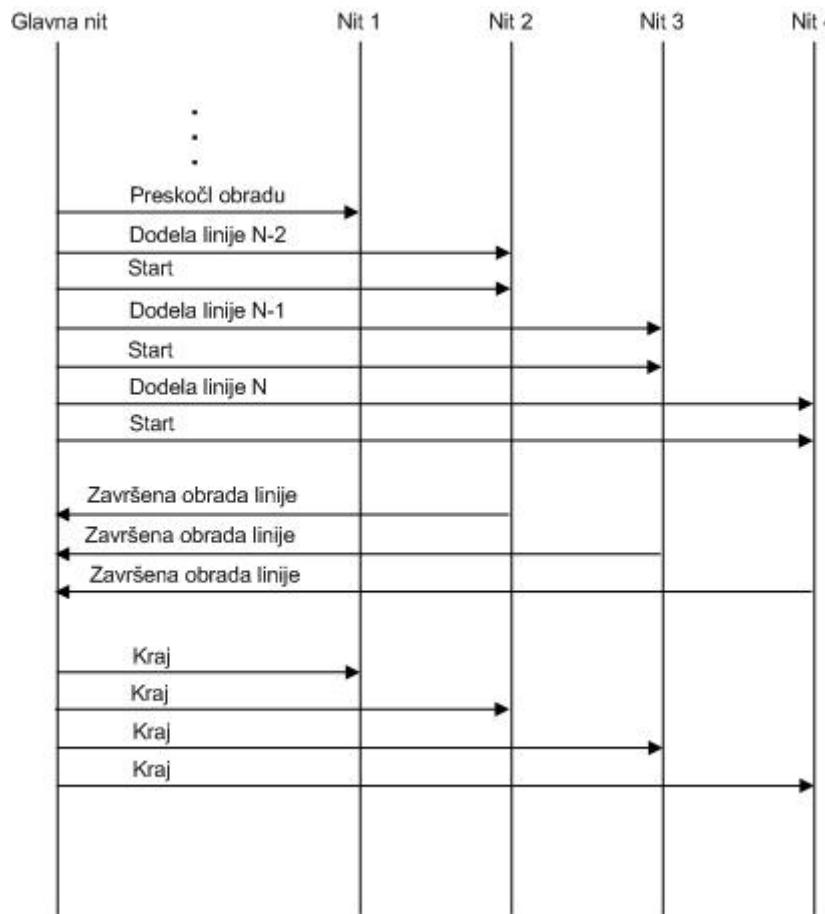


Slika 20 - Način podele posla nitima u implementaciji obrade po linijama slike



Slika 21 - MSC algoritma sa dodelom posla nitima po linijama slike

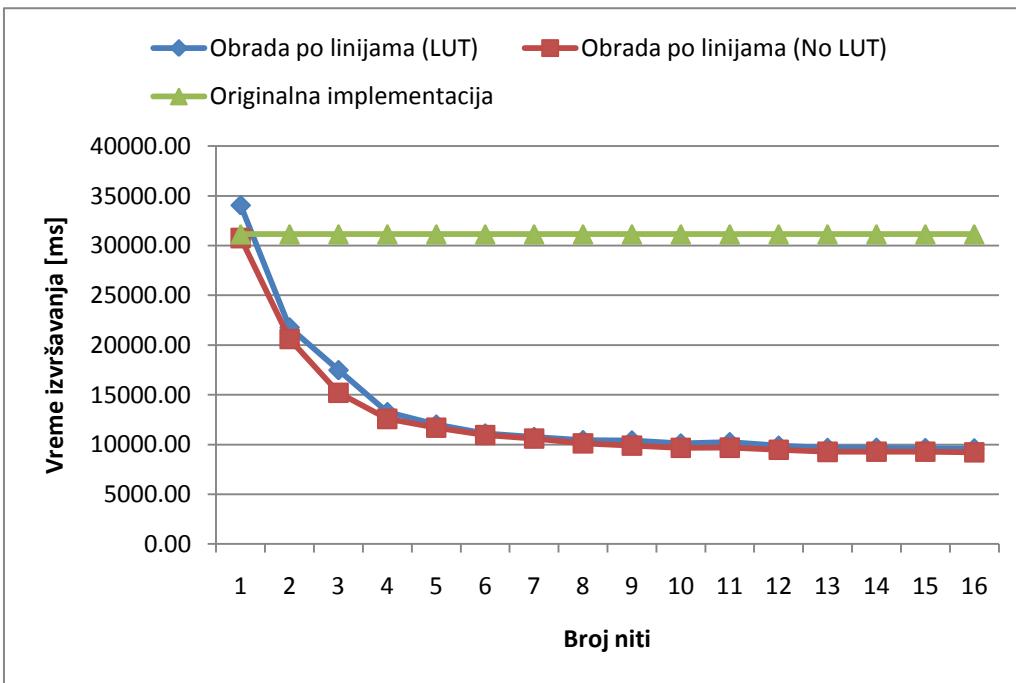
Slika 22 prikazuje razmenu poruka u slučaju kada visina slike nije deljiva brojem niti. Tada, u poslednjoj iteraciji, glavna nit ima manji broj linija za obradu, u odnosu na broj postojećih niti radilica. Nitima radilicama za koje nema podataka za obradu šalje se posebna poruka kojom im se signalizira da preskoče obradu i sačekaju signal za završetak programa.



Slika 22 - MSC dijagram u slučaju kada visina slike nije deljiva brojem niti

4.2.1.1 Rezultati

Slika 23 prikazuje rezultate obrade po linijama, dobijene na Intel platformi, za slučajeve korišćenja tabele sa težinskim faktorima, kao i bez nje.



Slika 23 – Rezultati dobijeni na Intel platformi paralelizacijom obrade po linijama slike

Vidi se da ovakva implementacija za jednu nit, u slučaju realizacije sa LUT, dovodi do sporijeg izvršenja, dok je bez LUT trajanje obrade približno jednako sekvencijalnoj realizaciji. Obrada se, u odnosu na sekvencijalno izvršavanje, ubrzava povećanjem broja niti, ali već nakon devet niti ulazi u zasićenje i ubrzanje postaje zanemarljivo. Kako nam procesorska jedinica nudi paralelno izvršavanje do 16 niti, vidimo da ovakva realizacija algoritma ne koristi dostupne resurse u potpunosti. Iako na Intel platformi istovremeno može raditi 16 niti, ona poseduje samo osam fizičkih jezgara pa se povećanjem broja niti iznad tog broja deo vremena izvršavanja troši na sinhronizaciju svih niti.

Tabela 3 prikazuje ubrzanje postignuto na ciljnoj platformi. Iz tabele se vidi da ovakva realizacija paralelizacije datog algoritma na ciljnoj platformi nije dala željene rezultate, osim u slučaju podele obrade na dve niti, uz korišćenje LUT.

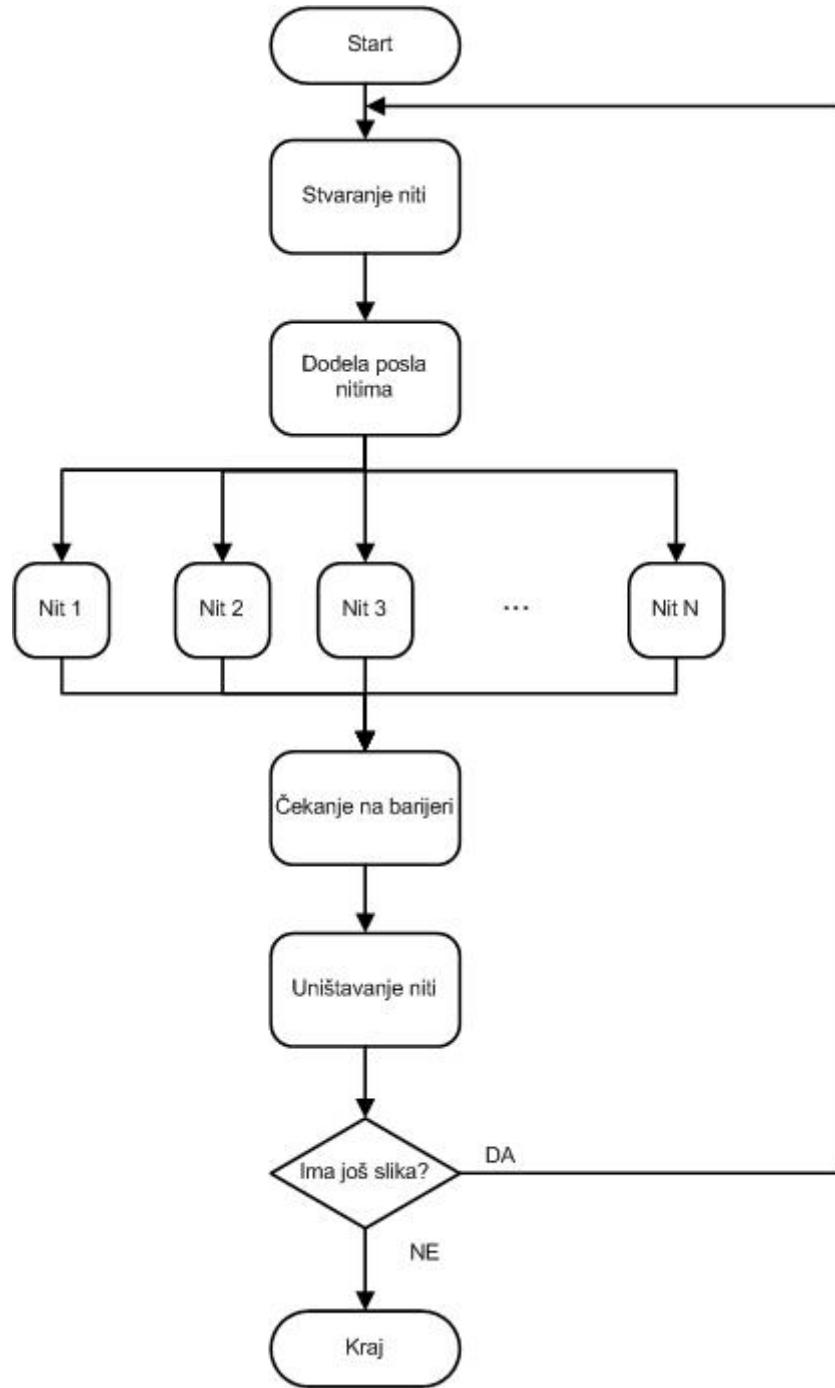
Tabela 3 - Ubrzanje postignuto paralelizacijom obrade po linijama na MIPS platformi

C/A	Originalna implementacija	Obrada po linijama (LUT)	Obrada po linijama (No LUT)
1	7.20	9.20	8.20
2	7.20	5.20	24.20
3	7.20	35.20	43.20
4	7.20	53.20	58.20

Podela posla po linijama nije efikasna jer podrazumeva trošenje mnogo procesorskog vremena na sinhronizaciju niti, koje se međusobno čekaju pre početka obrade, kao i po završetku, pre nego što dobiju nove podatke.

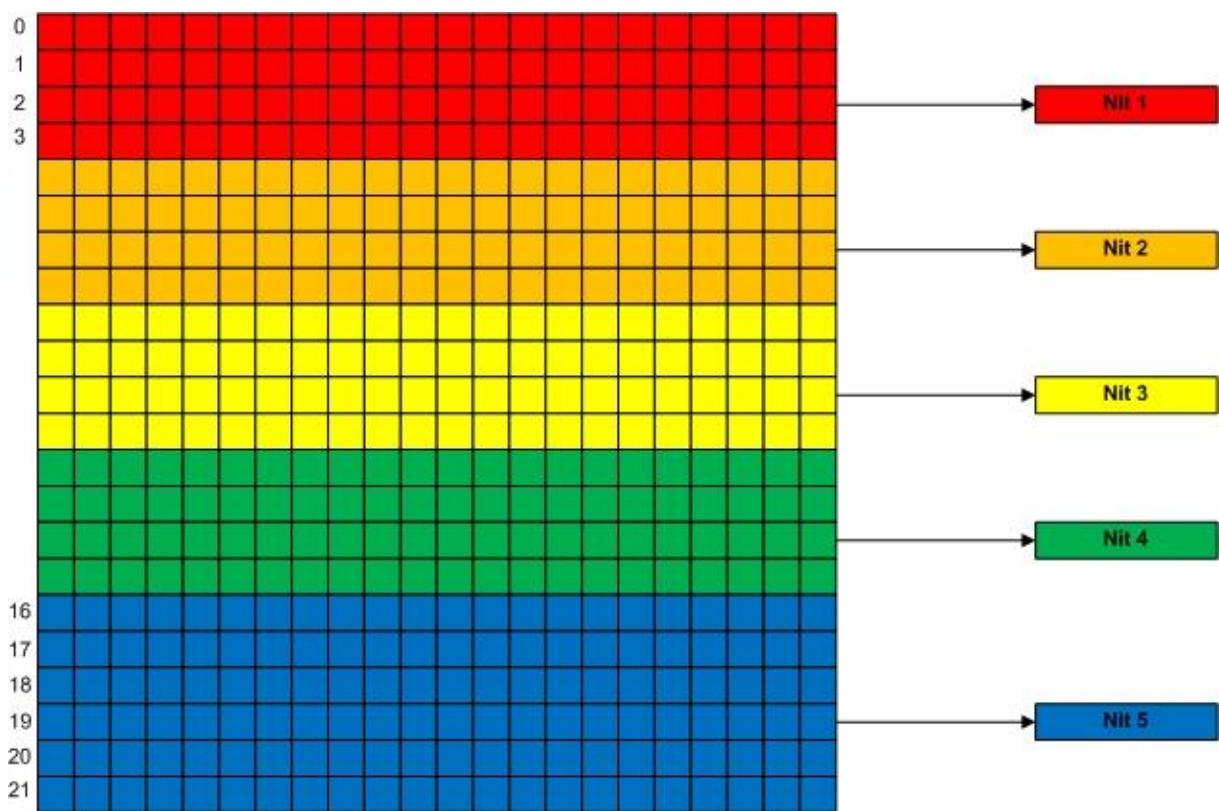
4.2.2 Podela obrade po bloku linija

Za razliku od prethodnog načina paralelizacije, ovde je na početku obrade svakoj niti bio dodeljivan blok linija slike. Slika 25 prikazuje način podele posla nitima pri ovakvoj paralelizaciji, a Slika 24 blok dijagram samog algoritma.



Slika 24 - BDA obrade podelom posla na blokove linija

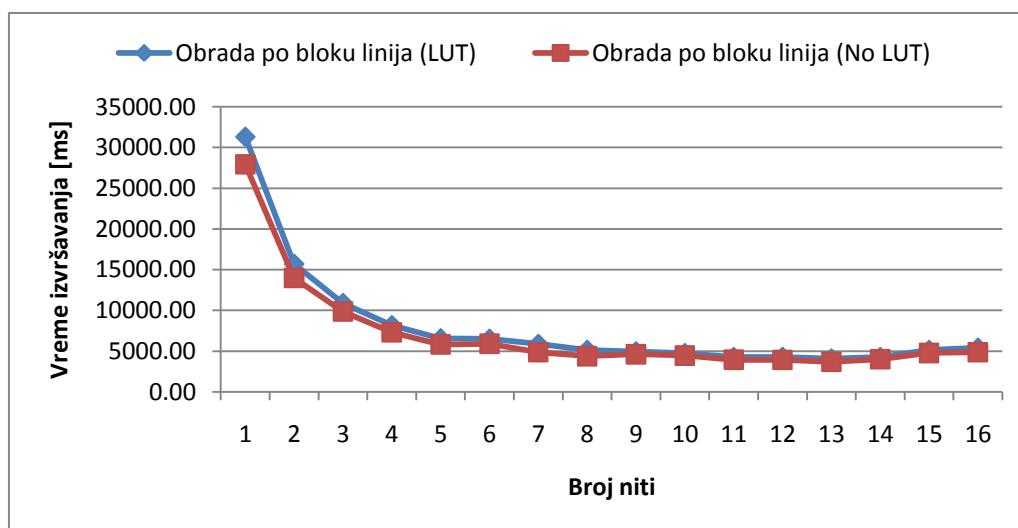
Nakon što glavna nit stvori niti radilice, svaka nit dobija određeni deo slike na obradu. Ukoliko visina slike nije deljiva brojem niti, poslednja nit će dobiti i sve linije koje ne mogu biti ravnopravno raspodeljene. Glavna nit čeka da sve radilice završe obradu, nakon čega ih uništava. Ukoliko se uzastopno obrađuju više slika, ceo postupak se ponavlja za svaku sliku.



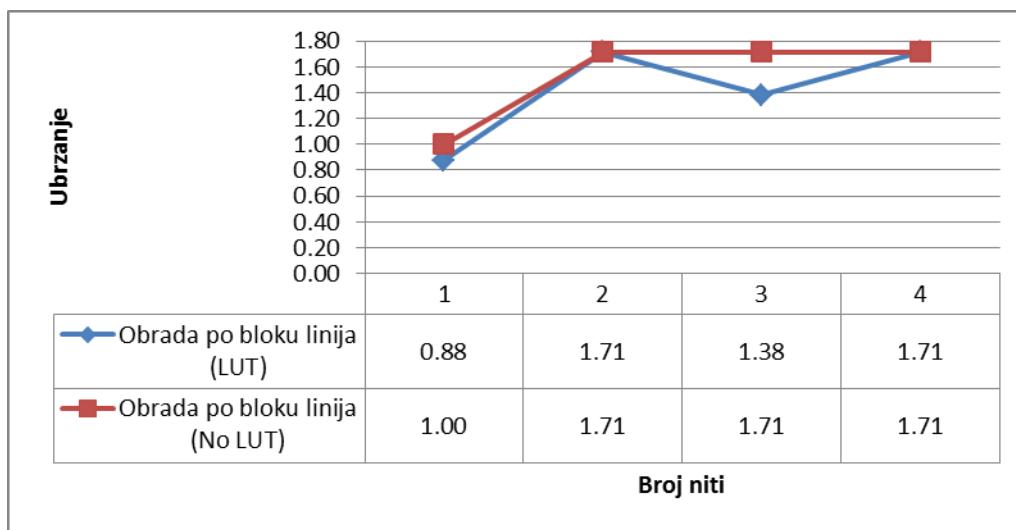
Slika 25 - Način podele posla nitima pri parelizaciji po bloku linija slike

4.2.2.1 Rezultati

Iako je ovakvim načinom paralelizacije postignuto bolje srednje vreme izvršenja, u odnosu na model obrade po linijama, javlja se isti problem zasićenja kada je broj niti radilica veći od osam. Teorijski maksimum se ne dostiže jer se srazmerno ukupnom vremenu izvršavanja programa značajno vreme potroši na stvaranje niti radilica.



Slika 26 - Vreme izvršavanja algoritma na Intel platformi za paralelizaciju obradom po bloku linija

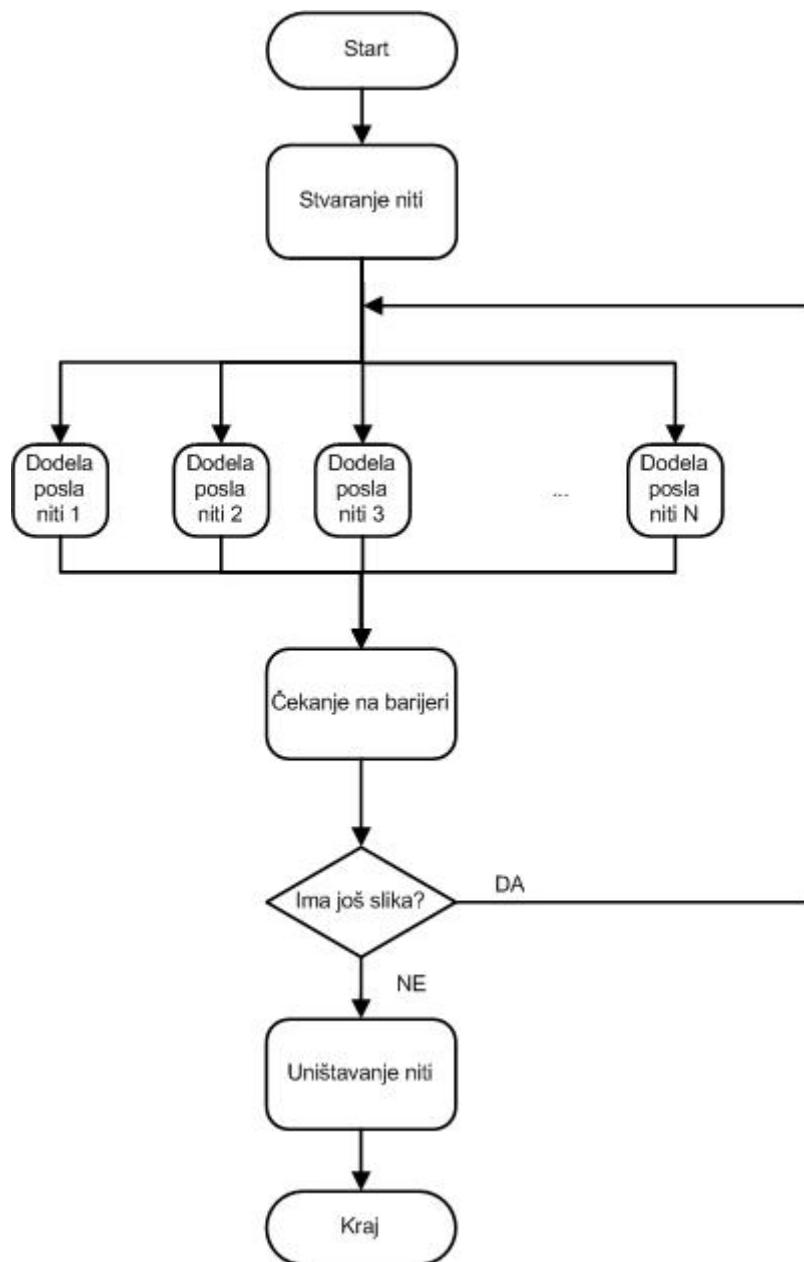


Slika 27 - Ubrzanje postignuto na ciljnoj arhitekturi paralelizacijom obrade po bloku linija

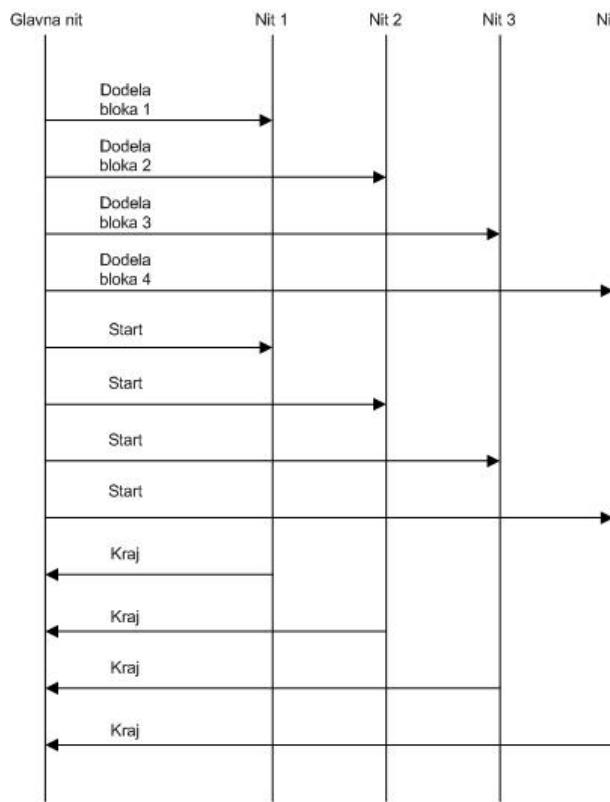
4.2.3 Podela obrade po bloku linija, korišćenjem preallocirane grupe niti

U modelu obrade po bloku linija korišćenjem preallocirane grupe niti, glavna nit stvara radilice pre početka ukupne obrade koju treba izvršiti. Na taj način je smanjena količina vremena koja se trošila na stvaranje i uništavanje niti u prethodnim modelima paralelizacije korišćenjem POSIX API. Slika 28 prikazuje BDA ovakvog modela.

Pošto su niti stvorene, svaka nit dobija blok linija slike na obradu, kao i u prethodnom modelu, što prikazuje Slika 25. Nakon toga, sinhronizacija niti se vrši samo po završenoj obradi cele slike. Ukoliko je potrebno interpolirati još slika, postupak podele posla se ponavlja, a u suprotnom se niti radilice uništavaju. Slika 29 prikazuje dijagram toka poruka između glavne niti i niti radilica u ovakovom modelu paralelizacije.



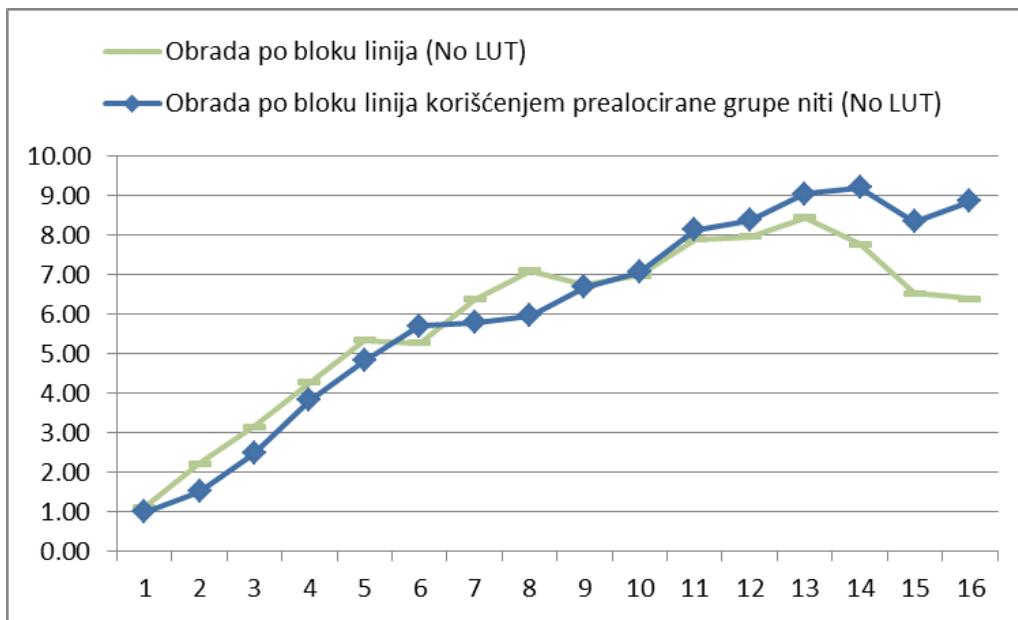
Slika 28 - BDA paralelizacije obrade po bloku linija, korišćenjem prealocirane grupe niti



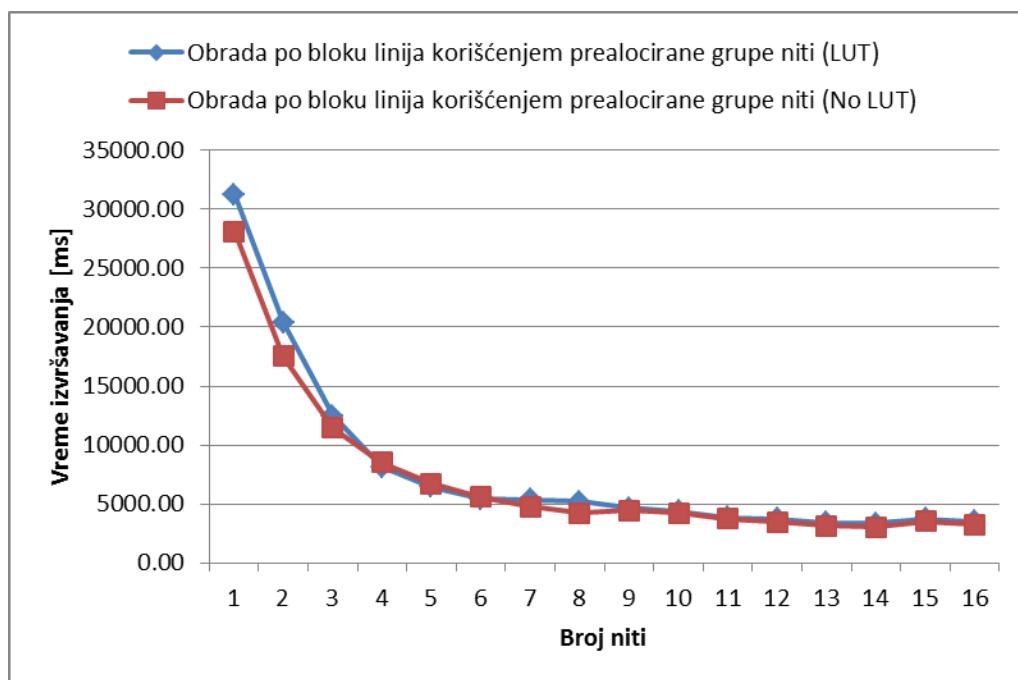
Slika 29 - MSC paralelizacije obrade po bloku linija, korišćenjem prealocirane grupe niti

4.2.3.1 Rezultati

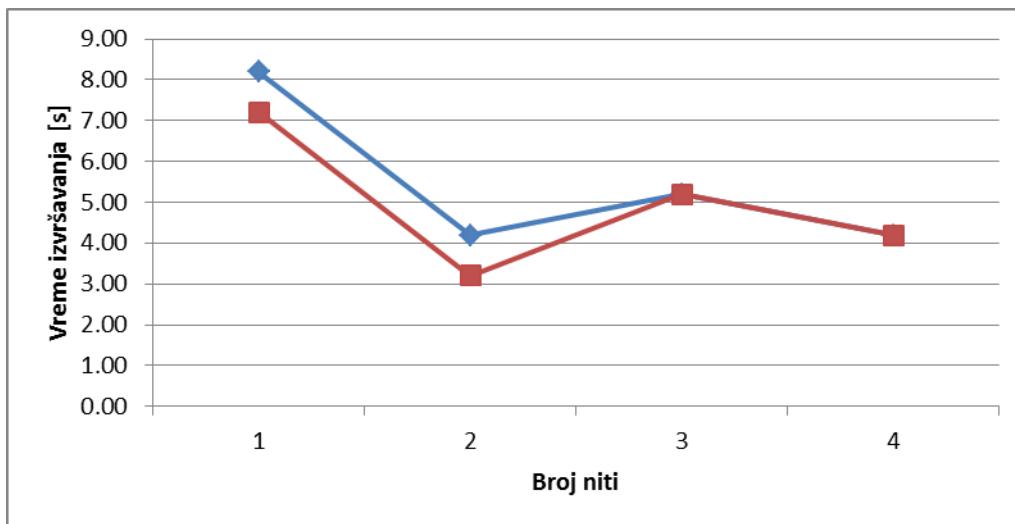
Kao i u prethodnim modelima paralelizacije POSIX nitima, nakon prelaska na logička jezgra i potrebe za sinhronizacijom niti, vreme izvršavanja postaje približno konstantno. Slika 31 prikazuje vremena izvršenja za Intel platformu, a Slika 32 za MIPS arhitekturu. Procenat ukupnog vremena koje se troši na stvaranje i uništavanje niti, za 16 niti, prikazuje Slika 33. Vidimo da se više od trećine vremena izvršenja zapravo ne troši na samu obradu. Slika 30 prikazuje rezultate dobijene primenom OpenMP standarda i ovog modela. Vidi se da su vremena izvršenja približna, s tim da ovakav model paralelizacije POSIX nitima donosi bolje rezultate sa porastom broja niti radilica.



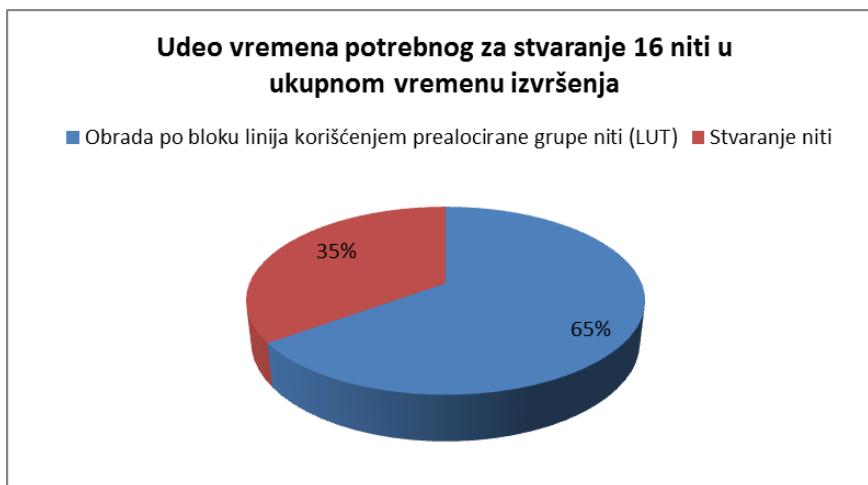
Slika 30 - Uporedni rezultati dobijeni primenom OpenMp standarda i obrade po blokovima linija sa prealociranom grupom niti



Slika 31 - Rezultati obrade podelom posla na blokove linija, korišćenjem prealocirane grupe niti na Intel platformi



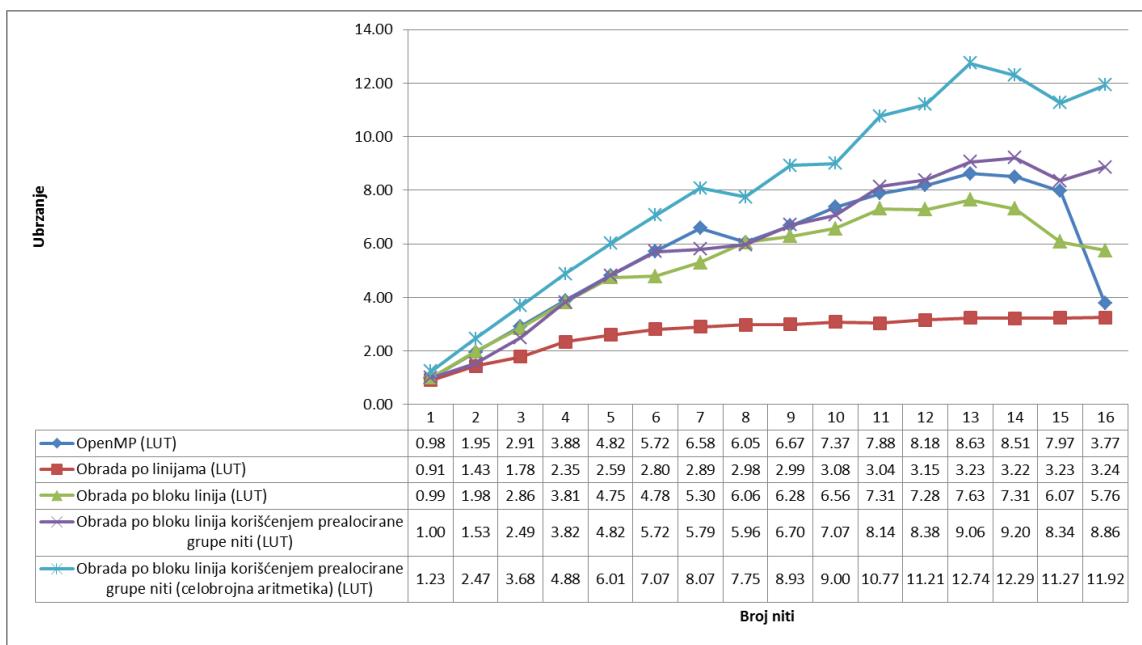
Slika 32 - Rezultati obrade podelom posla na blokove linija, korišćenjem prealocirane grupe niti na ciljnoj arhitekturi



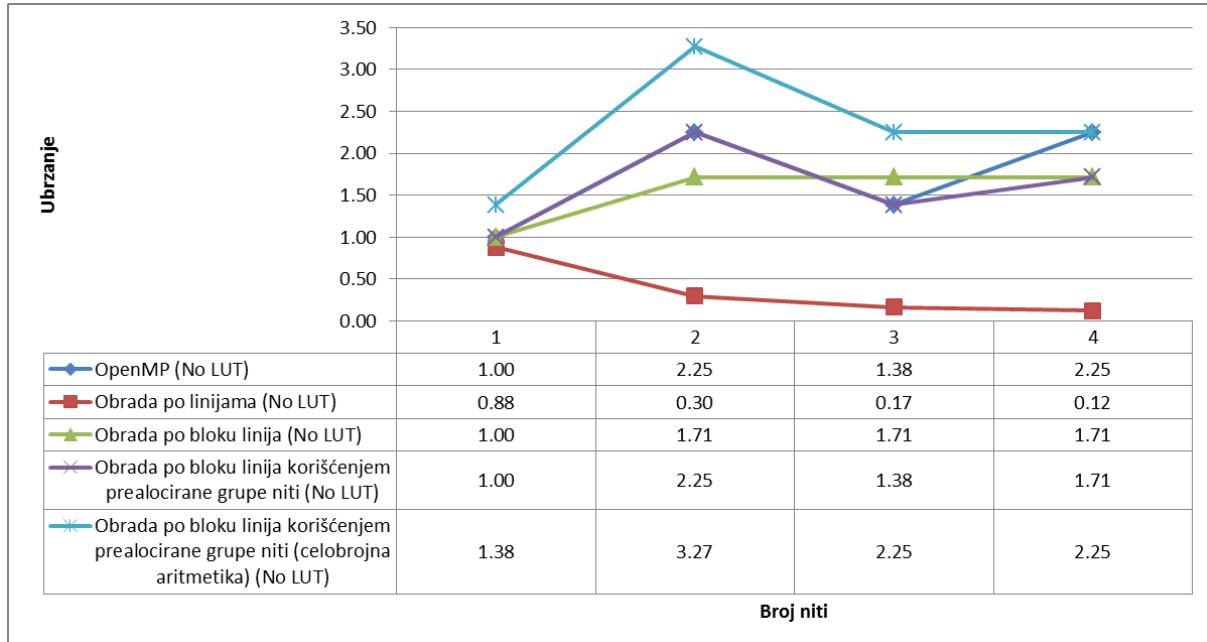
Slika 33 - Udeo vremena potrebnog za stvaranje 16 niti u ukupnom vremenu izvršenja

4.2.4 Uporedni rezultati svih POSIX modela paralelizacije

Iz uporednih rezultata svih primjenjenih modela paralelizacije POSIX nitima, koje prikazuju Slika 34 i Slika 35, vidimo da je obrada podelom posla po linijama ubedljivo najlošiji model jer se veći deo procesorskog vremena troši na sinhronizaciju niti pre i nakon obrade podataka. Paralelizacija OpenMP standardom i oba modela obrade podelom posla po bloku linija daju približne rezultate, s tim da model u kom glavna nit koristi prealociranu grupu niti radilica za celokupnu obradu daje nešto bolje rezultate za veći broj niti. Ubedljivo najefikasnija realizacija postignuta je kombinacijom paralelizacije obrade po bloku linija, uz korišćenje prealocirane grupe niti, i celobrojne aritmetike.



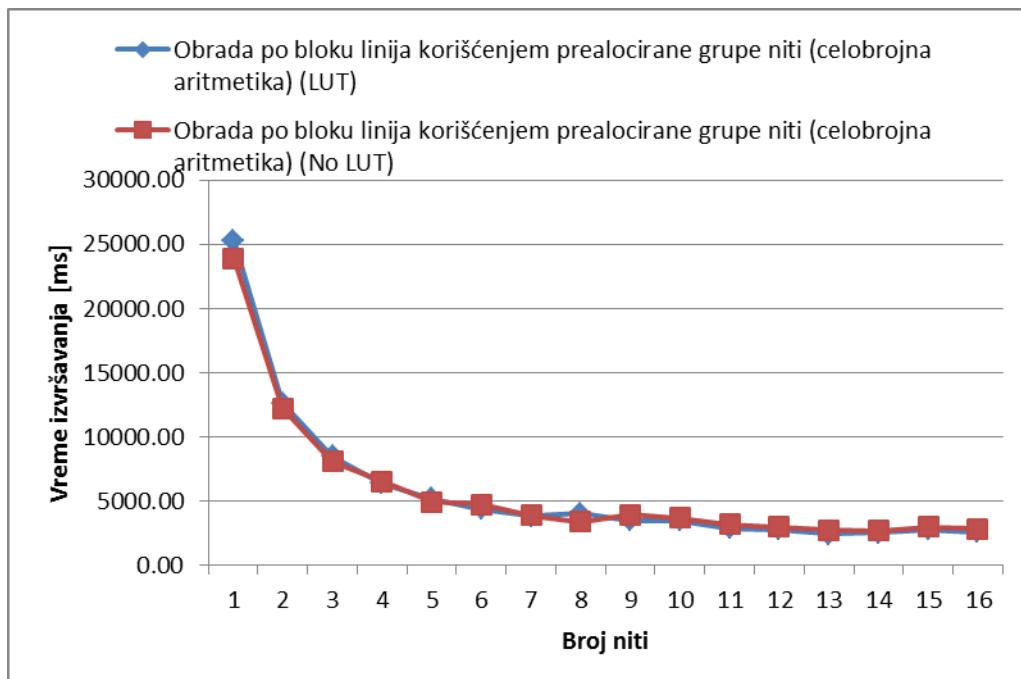
Slika 34 - Uporedni prikaz ubrzanja dobijenih primenom POSIX niti za paralelizaciju na Intel platformi



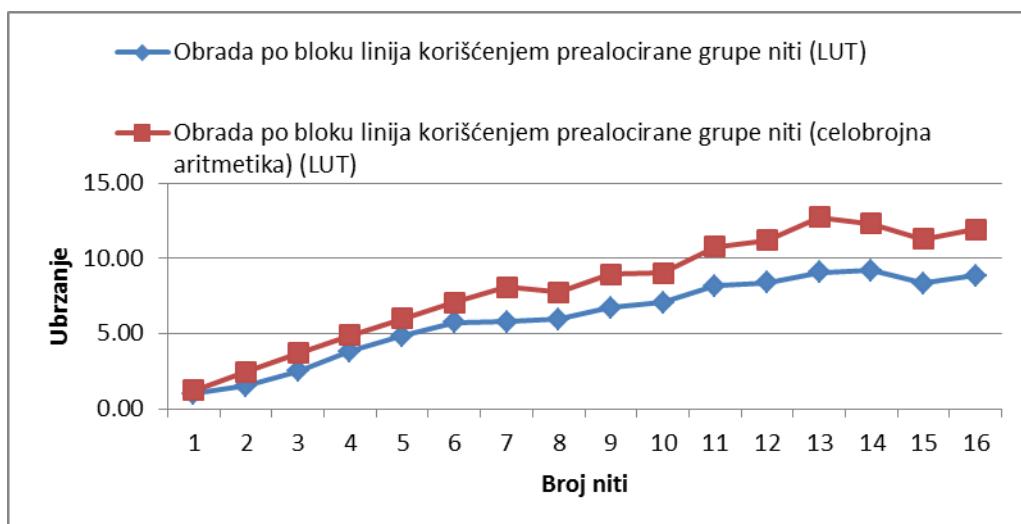
Slika 35 - Uporedni prikaz ubrzanja dobijenih primenom POSIX niti za paralelizaciju na ciljnoj platformi

4.3 Celobrojna aritmetika

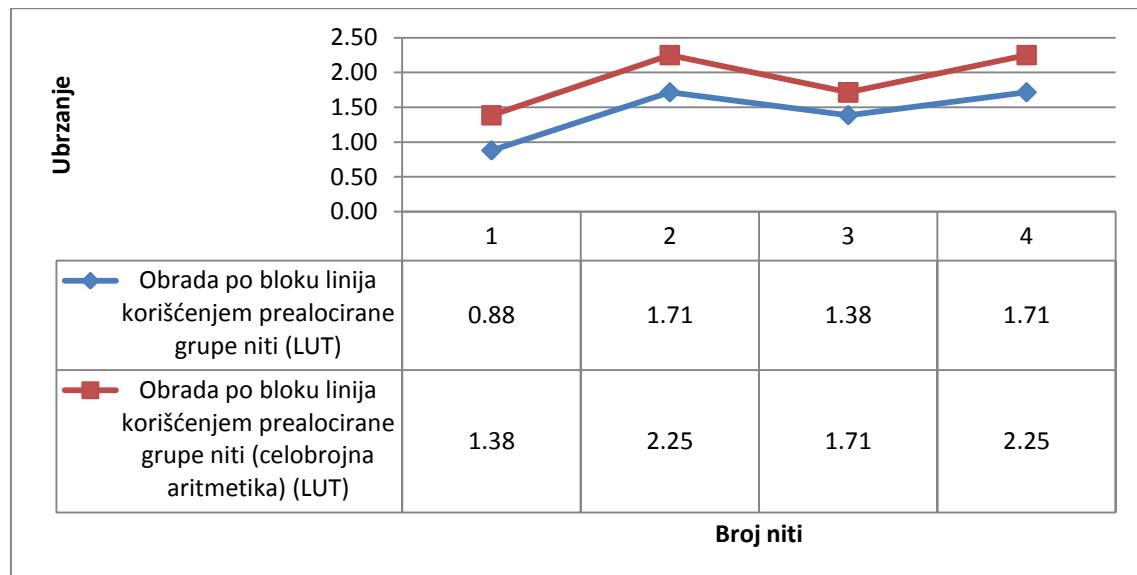
Ovaj metod paralelizacije se funkcionalno ne razlikuje od obrade po bloku linija korišćenjem prealocirane grupe niti. Jedina razlika je prelazak sa aritmetike pokretnog zareza na celobrojnu aritmetiku, čime se postiglo dodatno ubrzanje. Slika 36, Slika 37 i Slika 38 prikazuju poboljšanja dobijena na ovaj način.



Slika 36 - Rezultati obrada po bloku linija korišćenjem prealocirane grupe niti za izvršenje na Intel platformi



Slika 37 - Ubrzanje postignuto prelaskom na celobrojnu aritmetiku na Intel platform

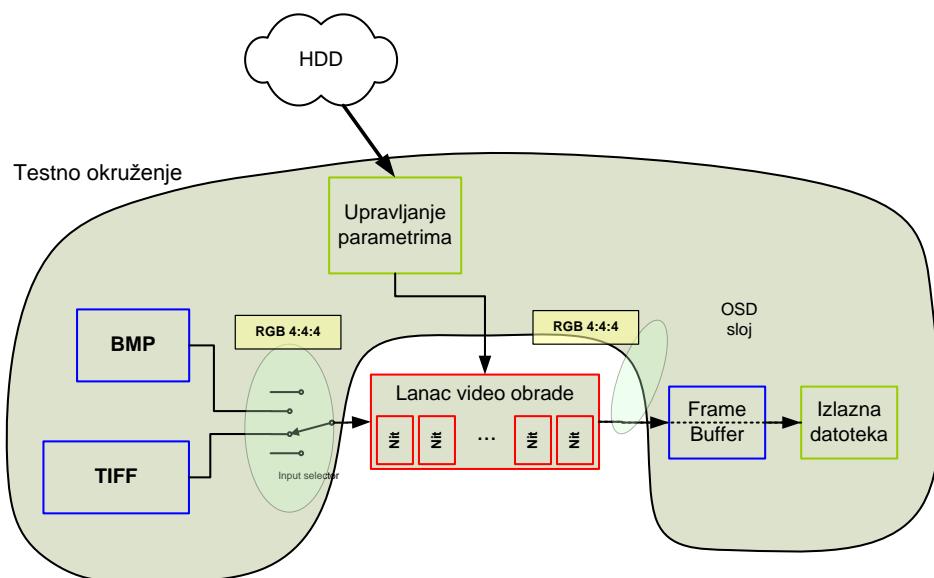


Slika 38 - Ubrzanje postignuto prelaskom na celobrojnu aritmetiku na ciljnoj platformi

5. Testno okruženje

Testno okruženje predstavlja Android konzolna aplikacija, koja kao parametar prima ulaznu sliku ili direktorijum sa listom ulaznih slika. Nad svakom slikom, primenjuju se bilinearna i AISLED interpolacija, a rezultati se uporedo prikazuju na ekranu. Osim za prikazivanje rezultata obrade, koji se mogu koristiti za analizu algoritma, uočavanje nastalih izobličenja itd., aplikacija služi i za merenje vremena izvršenja svake implementacije algoritma. Izlaz se ispisuje ili na ekran, ili u datoteku, u zavisnosti od konfiguracionog parametra koji se prosleđuje testnom okruženju.

Slika 39 prikazuje strukturu implementiranog radnog okruženja za testiranje realizacije kao i razvoj samog AISLED algoritma. Za proces razvoja algoritma, podržan je i automatski režim izvršavanja u kojem se rezultujuća slika ne prikazuje korisniku, već se smešta u izlaznu datoteku za kasniju analizu.



Slika 39 - Struktura testnog okruženja

6. Zaključak i pravci daljeg razvoja

Cilj zadatka je bio razviti i implementirati metodu za interpolaciju slika pri proizvoljnem faktoru uvećanja ili umanjenja, uz očuvanje oštine, i realizaciju prilagoditi savremenim mobilnim sistemima, tako da u potpunosti iskorišćava njihovu procesnu moć.

Intenzivan razvoj tehnologije poslednjih godina doveo je do povećanja zahteva u pogledu prenosa digitalnih signala sa stanovišta količine podataka koji treba da se prenesu i obrade u realnom vremenu. Napredak koji je postignut u domenu fizičkih arhitektura modernih uređaja potrebno je ispratiti i razvojem programske podrške, koja će korišćenjem novih programskih modela povećati stepen iskorišćenosti date platforme.

Predloženi algoritam je razvijan i implementiran inkrementalno u cilju poboljšanja njegovih već postojećih karakteristika i prilagođenju modernoj ugrađenoj platformi Sigma SMP8910.

Algoritam je pokazao dobre rezultate kod očuvanja oštine ivica u slici, s tim da su rezultati nešto lošiji prilikom interpolacije slike sa mnogo tekstura u sebi. Zbog malog bloka referentnih piksela pri detekciji ivice (ivica se određuje na osnovu četiri najbliža suseda), dešava se da se tekstura, koju, kao i ivice, karakterišu visokofrekventne komponente, detektuje kao prava ivica. Iz istog razloga, algoritam je ograničen na detektovanje samo četiri vrste ivica (horizontalna, vertikalna, i dve dijagonalne). Povećanjem broja referentnih piksela pomoću kojih se detektuje ivica, mogao bi se povećati i skup tipova ivica koje je moguće otkriti.

Kako bi se rešili neki od pomenutih problema, algoritam je proširen blokom koji pre interpolacije traži visokofrekventne komponente slike i izdvaja regije za koje se na osnovu veličine prepostavlja da su prave ivice. Prag za određivanje granične veličine regija je moguće menjati i na taj način podešiti preciznost kojom se ivice detektuju i dodatno poboljšati razlikovanje tekstura od ivica u slici. Rezultati pokazuju vidno poboljšanje

algoritma kako prilikom poređenja pomoću objektivnih mera, tako i prilikom subjektivnog ocenjivanja.

Dodatna mogućnost za unapređenje algoritma je povećanje bloka referentnih piksela, čime bi se, pored preciznije detekcije ivica, poboljšala identifikacija tipa ivica i proširio njihov skup, s tim da bi se kompleksnost algoritma povećala.

Pomoću nekoliko metoda paralelizacije, pokušano je prilagođenje algoritma izvršenju na ciljnoj, ugrađenoj, višejezgarnoj platformi. Algoritam interpolaciju vrši za svaki piksel posebno, bez zavisnosti od rezultata interpolacije ostalih piksela, pa je bio idealan za paralelizaciju na nivou podataka. Takva paralelizacija je postignuta korišćenjem OpenMP standarda, prilagođenog upravo paralelizaciji na nivou petlji. Odnos jednostavnosti realizacije paralelizma i rezultata dobijenih korišćenjem OpenMPa pokazuje da je za većinu primena ovaj standard sasvim dovoljan.

Sledeći korak je bio paralelizacija korišćenjem POSIX programske sprege, čime je, po cenu kompleksnije i duže implementacije, dobijena veća kontrola nad izvršenjem obrade i, kroz nekoliko modela raspodele posla među nitima, bolja iskorišćenost procesne moći ciljne platforme. Dobijeni rezultati su blizu teorijskog maksimuma kako na arhitekturi opšte namene, tako i na ugrađenim arhitekturama kao što je MIPS.

Dalja unapređenja su moguća primenom HW specifičnih optimizacija za ciljnu platformu. U te optimizacije spadaju i implementacija AISLED algoritma korišćenjem asemblerских instrukcija, kao i mehanizama za sinhronizaciju niti na nivou fizičke arhitekture. Takva rešenja garantuju optimalni programski kod za ciljnu paltformu, ali implementaciju čine zavisnom od konkretnе, izabrane arhitekture, što razvoj i održavanje programskog koda čini značajno kompleksnijim.

7. Literatura

- [1] MathWorks, <http://www.mathworks.com/>
- [2] OpenMP, <http://openmp.org/wp/>
- [3] Posix, <http://pubs.opengroup.org/onlinepubs/9699919799/>
- [4] Y. Lao, C. Tzeng, H. Wu, “Adaptive Image Scaling Based on Local Edge Directions”, Intelligent and Advanced Systems (ICIAS), 2010 International Conference on, pp. 1-4, 2010, Kuala Lumpur, Malaysia
- [5] M. Bratić, N. Lukić, M. Temerinac, D. Ačanski, “Jedno rešenje paralelizacije i vektorizacije algoritma za merenje zamućenosti pokretne slike na CBE platformi”, Zbornik radova 52. Konferencije za ETRAN, 2008, Palić, Srbija