UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA

Odsek za elektrotehniku i računarstvo

Institut za računarstvo i automatiku

Katedra za računarsku tehniku i računarske komunikacije

Optimizacija HuffYUV enkodera za Intel bazirane platforme

Diplomski - Bachelor rad

Mentor: prof. dr Miodrag Temerinac

Student: Vladimir Živkov 11547

Novi Sad, Jun 2009.

UNIVERZITET U NOVOM SADU • **FAKULTET TEHNIČKIH NAUKA**

21000 NOVI SAD, Trg Dositeja Obradovića 6

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, **RBR**:			
Identifikacioni broj, **IBR**:			
Tip dokumentacije, **TD**:		Monografska publikacija	
Tip zapisa, **TZ**:		Tekstualni štampani materijal	
Vrsta rada, **VR**:		Diplomski-bachelor rad	
Autor, **AU**:		Vladimir Živkov	
Mentor, **MN**:		prof. dr Miodrag Temerinac	
Naslov rada, **NR**:		Optimizacija HuffYUV enkodera za Intel bazirane platforme	
Jezik publikacije, **JP**:		Srpski	
Jezik izvoda, **JI**:		Srpski	
Zemlja publikovanja, **ZP**:		Srbija	
Uže geografsko područje, **UGP**:		Vojvodina	
Godina, **GO**:		2009	
Izdavač, **IZ**:			
Mesto i adresa, **MA**:			
Fizički opis rada, **FO**: (poglavlja/strana/citata/tabela/slika/grafika/prilo)			
Naučna oblast, **NO**:		Elektrotehnika i računarska tehnika	
Naučna disciplina, **ND**:			
Predmetna odrednica/Ključne **PO**:			
UDK			
Čuva se, **ČU**:		U biblioteci FTN, Novi Sad,Trg Dositeja Obradovića 6	
Važna napomena, **VN**:			
Izvod, **IZ**:			
Datum prihvatanja teme, **DP**:			
Datum odbrane, **DO**:			
Članovi komisije, **KO**:	Predsednik:		
	Član:		Potpis mentora
	Član, mentor:		

UNIVERSITY OF NOVI SAD • **FACULTY TECHNICAL SCIENCES**

21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic's publication	
Type of record, **TR**:		Word printed record	
Contents code, **CC**:		Bachelor thesis	
Author, **AU**:		Vladimir Živkov	
Mentor, **MN**:		prof. dr Miodrag Temerinac	
Title, **TI**:		Optimization of HuffYUV video encoder for Intel based platform	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2009	
Publisher, **PB**:			
Publication place, **PP**:			
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appis)			
Scientific field, **SF**:		Electrical engineering and computer science	
Scientific discipline, **SD**:			
Subject/Key words, **S/KW**:			
UC			
Holding data, **HD**:		Library of FTN, Novi Sad, Trg Dositeja Obradovića 6	
Note, **N**:			
Abstract, **AB**:			
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:		
	Member:		Mentor's sign
	Member, Mentor:		

SADRŽAJ:

1. Zadatak

Potrebno je prilagoditi HuffYUV algoritam Intel arhitekturi i optimizovati ga (korišćenjem SSE instrukcionog seta) u cilju postizanja rada u realnom vremenu (kompresija polja visoke rezolucije za vreme manje od 16 ms). Potrebno je particionisati podatke i vektorizovati obradu. Prilagoditi obradu izvršavanju na više procesorskih jezgara (dva jezgra, četiri jezgra) i ispitati skalabilnost vremena obrade sa brojem procesora na kojima se obrada izvršava.

2. Analiza problema

Potrebno je realizovati HuffYUV enkoder, upotrebom SSE instrukcionog seta, u cilju kompresije polja visoke rezolucije (1920x1080 tačaka, pri učestanosti od 60 Hz). Video kompresija se vrši u cilju smanjenja količine podataka u odnosu na ulaznu video sekvencu. Prilikom izbora video enkodera jednako je važno imati sliku visokog kvaliteta (nakon dekodovanja) i imati visok stepen kompresije. Ove dve stvari su, svakako, u koliziji.

Ulaz u aplikaciju je polje visoke rezolucije, dok je izlaz tok podataka.

Prilikom analiziranja problema, potrebno je u potpunosti upoznati se sa algoritmom HuffYUV enkodera i SSE instrukcionim setom. Takođe, treba razmotriti razlog HuffYUV kodovanja video sekvenci, kada postoje kartice za hvatanje (eng. capture card) video sekvenci, i u kolikoj meri je to isplativo sa stanovišta potrošnje računarskih resursa i materijalnih sredstava.

U nastavku će svaki od problema biti objašnjen u posebnom pod-poglavlju.

2.1 HuffYUV enkoder

HuffYUV enkoder je video enkoder bez gubitaka (eng. *lossless*), što znači da je izlaz iz dekodera bit-identičan originalnom ulazu u enkoder. Naime, algoritam se zasniva na računanju razlike susednih tačaka. Izračunata razlika susednih tačaka se koristi kao indeks tabele enkodovanja. Tabela enkodovanja zatim vraća kod izračunate razlike i njenu dužinu.

Problem HuffYUV enkodera može biti sledeći: ukoliko je razlika između susednih tačaka uvek najveća, dolazi se u situaciju da je tok podataka na izlazu enkodera veći od ulazne video sekvence koja se kompresuje (što je u suprotnosti sa celokupnom idejom kompresije)! Ovakva situacija se dešava ukoliko je tabela enkodovanja napravljena da favorizuje male razlike.

2.2 SSE instrukcioni set

SSE[1] (eng. *Streaming SIMD Extension*) je instrukcioni set za vektorsku obradu podataka. On predstavlja nadgradnju x86 arhitekturi, dizajniranu od strane Intela. Predstavljen je 1999. godine na Pentium III seriji procesora.

SSE podržava aritmetiku u pokretnom zarezu, kao i celobrojnu aritmetiku. Uvedeno je osam 128-bitnih registara. Svaki od registara može da se interpretira kao dva 64-bitna broja, četiri 32-bitna broja, osam 16-bitnih brojeva ili šesnaest 8-bitnih vrednosti.

SSE2 instrukcioni set se uvodi pojavom Pentium IV procesora. Predstavljene su nove instrukcije koje omogućavaju efikasniju vektorsku obradu. Zatim se pojavljuju SSE3, SSSE3, SSE4 instrukcioni setovi.

2.3 Kartice za hvatanje video sekvenci nasuprot HuffYUV enkodera

Postavlja se pitanje zašto pisati aplikaciju koja radi HuffYUV kompresiju, kada postoje kartice za hvatanje video sekvenci?

Deo ovakvih kartica direktno može da hvata video sekvence, potpuno oslobađajući procesor za druge procese. Pored analognog signala, ove kartice mogu da hvataju i digitalni signal (polje visoke rezolucije), sve naravno, u realnom vremenu. Njihova cena je relativno visoka (od nekoliko stotina, pa do hiljadu dolara, u zavisnoti od karakteristika).

Sa druge strane, postoje kartice za hvatanje video sekvenci, koje su niže cene (od 40$, pa na više), ali one zahtevaju procesorske resurse i potreban im je visok takt procesora prilikom obrade. Neke od jeftinijih verzija kartica ne mogu hvatati sekvence visoke rezolucije.

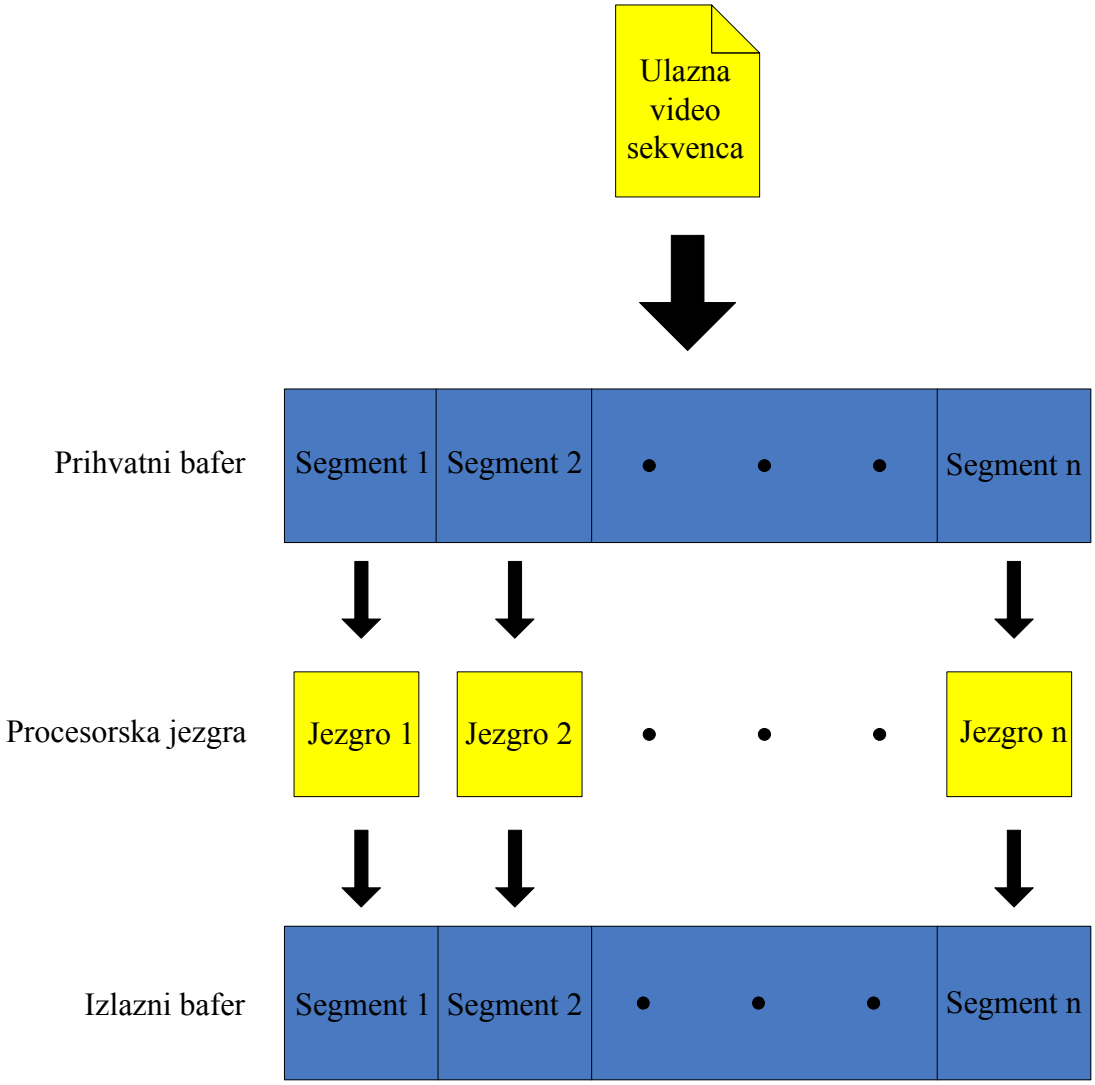
HuffYUV algoritam predstavlja programsko rešenje koje se oslanja na resurse procesora. Rešenje ne zateva nikakve dodatne uređaje koji bi potpomagali obradu.

3. Koncept rešenja

Kao okruženje za razvoj aplikacije koristi se *Microsoft Visual Studio 2008*, pod *Windows* operativnim sistemom.

Pri realizaciji programske podrške, treba voditi računa da obrada bude omogućena kako na jednom, tako i na više procesorskih jezgara, sve u cilju ispitivanja skalabilnosti vremena obrade u zavisnosti od broja procesorskih jezgara. Pre svega, cilj je realizovati obradu polja visoke rezolucije u realnom vremenu na jednom procesorkom jezgru. Slika 3.1. ilustruje podelu obrade na različit broj procesorskih jezgara. Ulazna video sekvenca se deli na željeni broj procesorskih jezgara za obradu. Zatim se određeni deo učitane sekvence prosleđuje određenom procesorskom jezgru, koje ga po obradi upisuje u deo izlaznog bafera, predviđenog za to jezgro.

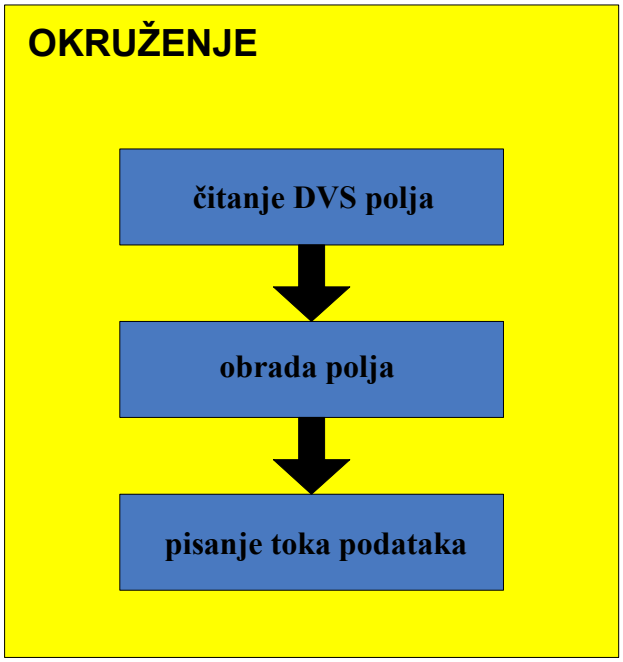
Pri realizaciji okruženja mora se voditi računa da svako procesorsko jezgro dobije svoj deo podataka za obradu i da se dobijeni tok podataka zapiše na pravo mesto u izlaznom baferu. Obrada je završena kada svako od jezgara završi sa svojim delom obrade.



Slika 3.1. Podela obrade na više procesorskih jezgara

HuffYUV algoritam zahteva da se na ulaz u enkoder dovodi YUY2 format podataka (Y U Y V)[2], pa bi se u okruženju prilikom učitavanja video sekvenci prvo trebao pripremiti format podataka za svako od polja koje se obrađuje.

Na slici 3.2. prikazan je uopšten pogled na programsku podršku.



Slika 3.2. Uopšteni pogled na aplikaciju

Kako dekoderska aplikacija obrađuje podatke po blokovima, kompresija polja se vrši po blokovima podataka, pri čemu je broj linija po bloku identičan (izuzev u slučaju kada visina nije deljiva sa brojem procesorskih jezgara, pri čemu se za zadnje blokove u okviru polja postavi drugi broj linija po bloku). Na ovaj način se vrši particionisanje podataka. Za svaki od blokova se piše zaglavlje (slike 4.5. i 4.6.), gde se između ostalog zapisuje veličina bloka i broj linija po bloku, kako bi dekoder znao količinu podataka koju treba da dekoduje.

Kako su SSE registri 128-bitni, to znači da se u registar može smestiti šesnaest vrednosti tačaka, što znači da će se istovemeno moći realizovati obrada osam komponenti osvetljaja i osam komponenti boje.

4. Programsko rešenje

4.1 Radno okruženje

Radno okruženje je napravljeno tako da može da radi obradu na više procesorskih jezgara. Naime, može da radi sa jednim do četiri fizičkih procesorskih jezgara. U slučaju da procesor ima Hyper-threading[3], tada se jedno fizičko jezgro tretira kao dva logička. Naime, pokreću se dve niti na jednom fizičkom jezgru, a program to jedno fizičko jezgro tretira kao dva logička. Tehnologija Hyper-Threading je uvedena od strane Intela.

Broj procesorskih jezgara za obradu se unosi kao parametar programske podrške. Provera ovog parametra se vrši poređenjem broja jezgara sa unetim parametrom. U slučaju da je broj unetih jezgara za obradu manji i jednak broju jezgara procesora, obrada je moguća, dok se u suprotnom slučaju izlazi iz programa. Iz programa se izlazi i ukoliko je broj procesora za obradu manji od jedan i ukoliko je veći od najvećeg dozvoljenog broja.

Kao parametar programske podrške se unosi i ime ulazne video sekvence. U slučaju nepostojanja unetog imena sekvence, na korisnički ekran se ispisuje odgovarajuća poruka.

Ukoliko ne postoje parametri programske podrške i ukoliko parametri nisu u sledećem formatu: *ime_ulazne_sekvence broj_jezgara*, korisnik se obaveštava odgovarajućom porukom.

Kada su svi parametri ispravno učitani, vrši se postavljanje promenjlivih o broj jezgara, širini i visini slike, kao i o broju polja.

Zatim se vrši inicijalizacija tabele enkodovanja. Izračunata razlika susednih tačaka se koristi kao indeks tabele enkodovanja, koja zatim vraća 32-bitnu vrednost. Ta 32-bitna vrednost je u formatu kao na slici 4.1.

Slika 4.1. Format vraćene vrednosti tabele enkodovanja

Tabela enkodovanja je preuzeta sa interneta kao besplatna i napravljena je za određenu sekvencu, što znači da se kvalitet enkodovanja ne može garantovati za sve sekvence.

U cilju ubrzanja pristupa memoriji, vrši se poravnavanje bafera. U slučaju da se poravnavanje bafera ne uradi, to smanjuje brzinu pristupa memoriji pri pisanju ili čitanju podataka koji nisu umnožak od 16 bajta, jer procesor mora da odradi još neka dodatna računanja. Za poravnavanje se koristi sledeći makro:

#define MEM_ALIGN(var_decl, mod) __declspec(align(mod)) var_decl

Parametri makroa su definicija makroa i broj bajta na koji se vrši poravnavanje.

Obrada (HuffYUV enkodovanje) video sekvenci obavlja se u nitima. Pravi se onoliki broj niti, koliki je uneti broj jezgara za obradu. Kao parametar svake niti prosleđuje se struktura *ControlBlock_t*. Ovo je neophodno kako bi svako od jezgara dobilo potrebne podatke za svoj deo obrade. Polja strukture su prikazana na slici 4.2.

```
typedef struct
{
        unsigned int   uiThreadID;
        unsigned int   uiWidth;
        unsigned int   uiHeight;
        unsigned char  *pucInputBuffer_INTEL;
        unsigned char  *pucOutputBuffer_INTEL;
        BOOL  bEnd;
        HANDLE  hContinue;
        HANDLE  hDone;
}ControlBlock_t;
```

Slika 4.2. Izgled strukture koja se prosleđuje niti

Za svaku od niti, postavlja se širina sekvence, redni broj niti i visina. Visina dela sekvence se računa po sledećoj formuli: *visina_sekvence/(broj_blokova*broj_linija_po_bloku)*. U slučaju ostatka pri deljenju, poslednje jezgro dobija svoj deo visine, uvećan za taj ostatak. Nakon toga, potrebno je proslediti adresu dela ulaznog bafera iz kojeg se čitaju podaci, kao i adresu dela izlaznog bafera u koji se smešta dobijeni tok podataka.

Pre pravljenja niti, potrebno je napraviti događaje (eng. Event), na čije će signaliziranje niti početi sa svojim izvršavanjem. Ovo je neophodno kako bi se obrada mogla zaustaviti dok se ne učita polje koje se treba obraditi.

Kada polje bude učitano, poziva se funckija za pravljenje YUY2 formata podataka, a zatim se vrši signalizacija svake niti da počne sa svojim delom obrade. Nakon toga, čeka se da svaka od niti završi svoju obradu, za čim sledi pisanje toka podataka. Ovaj proces se ponavlja onoliko puta, koliko ima polja u obrađivanoj video sekvenci.

Pošto nit bude kreirana, ona poziva svoju funkciju, gde nailazi na čekanje odgovarajućeg događaja. Kada događaj bude signaliziran, započinje obrada. Po završetku obrade signalizira se događaj da je trenutno aktivna nit završila obradu.

Za merenje vremena obrade je napravljena posebna funkcija, u cilju povećanja tačnosti izmerenog vremena. Naime, računa se broj ciklusa procesora na početku obrade i na kraju obrade.

Jednostavnim oduzimanjem dobija se broj utrošenih ciklusa procesora. Kada se ovaj broj podeli procesorskim taktom, dobija se vreme utrošeno za obradu.

Obrađeni podaci (tok podataka) se zapisuje u deo izlaznog bafera, predviđenog za datu nit. Tok podataka se zapisuje u prethodno otvorenu datoteku u binarnom obliku. Po završetku obrade, vrši se zatvaranje izlazne datoteke.

4.2 HuffYUV obrada

4.2.1 Funkcija obrade polja

Funkcija obrade naziva se *HuffYUVCompressing* i definisana je u okviru *HuffYUV_Intel.cpp* programskog modula. Argumenti funkcije obrade su prikazani na slici 4.3.

```
int HuffYUVCompessing(unsigned char  *_pucInputBuffer,
                      unsigned char  *_pucOutputBuffer,
                      unsigned int   _uiInputWidth,
                      unsigned int   _uiInputHeight)
```

Slika 4.3. Funkcija za obradu polja

Argumenti funkcije obrade su: pokazivač na ulazni bafer, pokazivač na izlazni bafer, širina i visina trenutno obrađivanog polja. Povratna vrednost funkcije je broj obrađenih bajtova.

Obrada se vrši po blokovima i na ovaj način je izvršeno particionisanje podataka. Blokovski način obrade diktiran je zahtevima dekoderske aplikacije.

Na početku izvršavanja funkcije vrši se podela visine polja sa proizvodom broja blokova i broja linija po bloku. U slučaju ostatka pri deljenju, zadnji blokovi će imati manji broj linija po bloku.

Funkcija koja obavlja blokovsku obradu naziva se *HuffYUVCompressingBlocks*. Kada funkcija završi svoj deo obrade ona, preko adrese parametara koje dobija pri pozivu, postavi broj obrađenih bajtova komponente osvetljaja i broj obrađenih bajtova komponente boje. Pomoću ovih brojeva obrađenih bajtova, dobija se broj bajtova koji se kopiraju iz privremenih izlaznih bafera funkcije

blokovske obrade, u izlazni bafer programske podrške. Nakon što kopiranje bude završeno, pomera se pokazivač ulaznog bafera za širinu polja pomnoženu proizvodom broja blokova i broja linija po bloku. Dobijena cifra se mora još pomožiti sa dva zbog postojanja dve komponente (osvetljaj i boja).

4.2.2 Funkcija obrade blokova podataka

Funkcija obrade blokova podataka naziva se *HuffYUVCompressingBlocks* i definisana je u okviru *HuffYUV_Intel.cpp* programskog modula. Argumenti funkcije obrade blokova podataka su prikazani na slici 4.4.

```
static __inline void HuffYUVCompressingBlocks(
                unsigned char *_pucInputBuffer,
                unsigned char *_pucLumaOutputBuffer,
                unsigned char *_pucChromaOutputBuffer,
                unsigned int  _uiInputWidth,
                unsigned int  _uiNumOfBlocks,
                unsigned int  _uiNumOfLinesPerBlock,
                unsigned int  *_uiYTotalSize,
                unsigned int  *_uiCTotalSize)
```

Slika 4.4. Funkcija za obradu blokova podataka polja

Argumenti funkcije obrade blokova podataka su: pokazivač na ulazni bafer, pokazivači na izlazne bafere osvetljaja i boje, širina polja, broj blokova za obradu, broj linija po bloku, adrese na koje se upisuju brojevi obrađenih bajtova osvetljaj i boje.

Za svaki blok koji se obrađuje mora se popuniti zaglavlje, u formatu prikazanom na slici 4.5. i na slici 4.6. Pri pisanju svakog bloka osvetljaja i boje odvaja se po šesnaest bajta.

Redni broj bita	Sadržaj koji se upisuje
0	Vrednost prve komponente osvetljaja u bloku
1	Ne koristi se
2	Broj obrađenih linija u okviru bloka
3	0x00
4 do 7	Veličina obrađenog bloka u bajtovima
8	0x01
9 do 15	Ne koristi se

Slika 4.5. Zaglavlje za blok obrađenog osvetljaja

Redni broj bita	Sadržaj koji se upisuje
0	Vrednost prve U komponente boje u bloku
1	Vrednost prve V komponente boje u bloku
2	Broj obrađenih linija u okviru bloka
3	0x01
4 do 7	Veličina obrađenog bloka u bajtovima
8	0x01
9 do 15	Ne koristi se

Slika 4.6. Zaglavlje za blok obrađene boje

Obrada HuffYUV enkodera funkcioniše na sledeći način: na ulaz se dovode podaci u YUY2 formatu. U zaglavlje se zapisuje prva vrednost osvetljaja, odnosno prve vrednosti boje, a zatim se na osnovu izačunate razlike piše izlazni tok podataka. Za upis u memoriju koriste se 32-bitni registi, dok se za obradu podataka koriste 128-bitni registri. Intelov SSE instrukcioni set omogućuje rad sa 128-bitnim registrima i omogućuje vektorsku obradu podataka. Razlog korišćenja 32-bitnih registara za pisanje u memoriju je nepostojanje instrukcija koje bi vršile odgovarajuću obradu (detaljno će biti objašnjeno u nastavku).

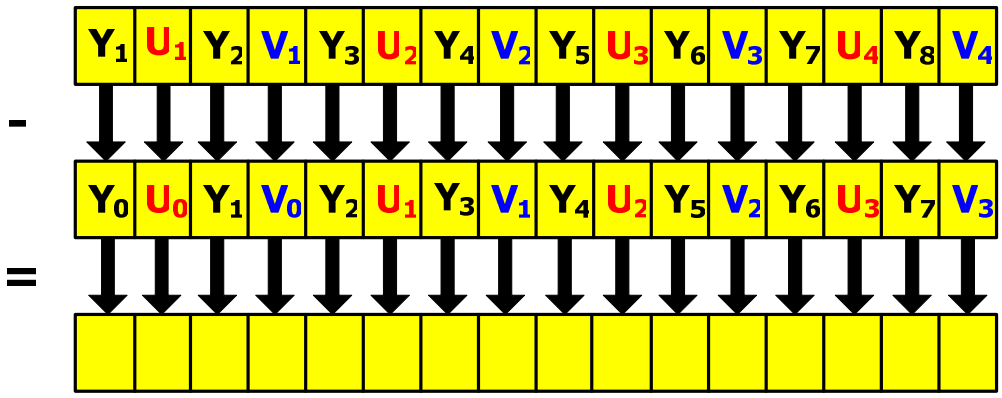
HuffYUV algoritam obrade smešten je u makro, u cilju povećanja brzine obrade.

Kako bi se izračunala razlika susednih tačaka, potrebno je prvo izvršiti dobavljanje odgovarajućih operanada. Za dobavljanje 128-bitnih vrednosti podataka iz memorije koristi se

funkcija *_mm_load_si128*. Naime, ova funkcija dobavlja 128-bitni podatak i smešta ga u ciljni registar.

Algoritam HuffYUV obrade funkcioniše na sledeći način: izračuna se razlika susednih tačaka, a zatim se ta razlika koristi kao indeks tabele enkodovanja. Povratna vrednost tabele enkodovanja je 32-bitni podatak, koji predstavlja dužinu komponente osvetljaja (8-bita), dužinu komponente boje (8-bita), kod komponente osvetljaja (8-bita), kod komponente boje (8-bita). Sadržaj izlaznog registra osvetljaja se pomera za dužinu osvetljaja, nakon čeka se radi ili operacija izlaznog registra i koda osvetljaja. Prilikom svakog pomeranja izlaznog registra osvetljaja, brojač pomeranja se uvećava za broj pomeraja. Kada broj pomeraja dostigne ili pređe dužinu izlaznog registra osvetljaja, sadržaj registra se upisuje u memoriju i brojač pomeraja se umanjuje za dužinu registra. Na isti način se formira sadržaj izlaznog registra boje i isto tako se sadržaj izlaznog registra boje zapisuje u memoriju.

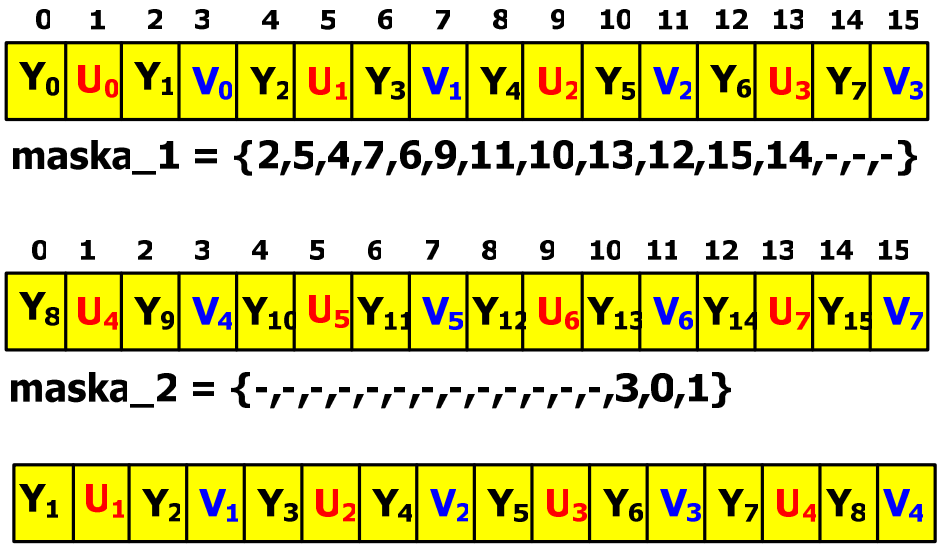
SSE instukcioni set omogućuje vektorizaciju prilikom oduzimanja vrednosti susednih tačaka. Naime, omogućeno je vektorsko oduzimanje osam komponenti osvetljaja i osam komponenti boje. Vektorsko oduzimanje ilustruje slika 4.7.



Slika 4.7. Ilustracija vektorskog oduzimanja

Kako bi se pripremio format podataka za oduzimanje, koristi se funkcija *_mm_shuffle_epi8*. Ona omogućuje da se u samo jednoj instrukciji pripremi prvi operand vektorskog oduzimanja (drugi operand se jednostavno učita iz memorije, bez ikakve pripreme). Funkcija omogućava formiranje sadržaja ciljnog 128-bitnog registra prema unapred definisanoj maski. Naime, za formiranje sadržaja registra, potrebni su odgovarajući podaci iz dva registra. Pošto funkcija *_mm_shuffle_epi8*

kao parametar ima samo jedan registar, mora se pozvati pomenuta funkcija za svaki od registara. Na odgovarajućim pozicijama u prvom registru, gde su potrebni podaci iz drugog registra, upisuju se nule. Takođe, na odgovarajućim pozicijama u drugom registru, gde su potrebni podaci iz prvog registra, upisuju se nule. Zatim se jednostavnom ili operacijom formira potreban sadržaj registra. Slika 4.8. ilustruje formiranje sadržaja registra pomoću funkcije *_mm_shuffle_epi8*.



Slika 4.8. Ilustracija formiranja sadržaja registra upotrebom *_mm_shuffle_epi8* funkcije

Nakon što je konsultovana tabela enkodovanja za sve razlike i nakon svih vraćenih rezlutata tabele, daljeg prostora za vektorizaciju nema. Pošto su registri koji pišu u memoriju 32-bitni, postoji mogućnost da se četiri 32-bitna vrednosti smeste u jedan 128-bitni registar. Ključni nedostatak je nemogućnost pomeranja svake od 32-bitnih vrednosti u okviru 128-bitnog registra različit broj puta. Naime, SSE instrukcioni set omogućava pomeranje delova 128-bitnog registra samo isti broj puta.

5. Testiranje

Programska podrška je pisana u više faza, tako da je svaka faza, nakon što je implementirana, bila testirana. Sam algoritam HuffYUV enkodera je u početnoj fazi izrade bio u telu funkcije, kako bi se mogao proveriti tok izvršavanja aplikacije. Nakon što je potrvđena ispravna funkcionalnost, algoritam je preveden u makro (jer je nemoguće kontrolisano izvršavanje makroa), kako bi se postigla obrada u realnom vremenu. Razlog smeštanja algoritma u makro je, svakako, brzina izvršavanja.

Okruženje je testirano za obradu na različitom broju procesorskih jezgara. Posebno je testirana funkcija *SetThreadAffinityMask*. Ova funkcija omogućuje da određena nit izvršava svoju obradu na određenom procesorskom jezgru. Testirana je za izvršavanje niti na svakom od jezgara ponaosob, kao i za izvršavanje na više niti u paraleli.

Umesto 32-bitnih registara za pisanje toka podataka, pokušano je da registri za pisanje toka podataka budu 64-bitni, čime bi se dva puta smanjio broj pisanja u memoriju, ali je zbog većeg broja instrukcija, u odnosu na 32-bitno pisanje memorije, vreme izvršavanja bilo mnogo veće. Razlog pokušaja pisanja pomoću 64-bitnih, a ne pomoću 128-bitnih registara je nemogućnost pomeranja (nepostojanje instrukcije) celokupnog sadržaja 128-bitnog registra.

Računar na kome je aplikacija testirana je sledeće konfiguracije:

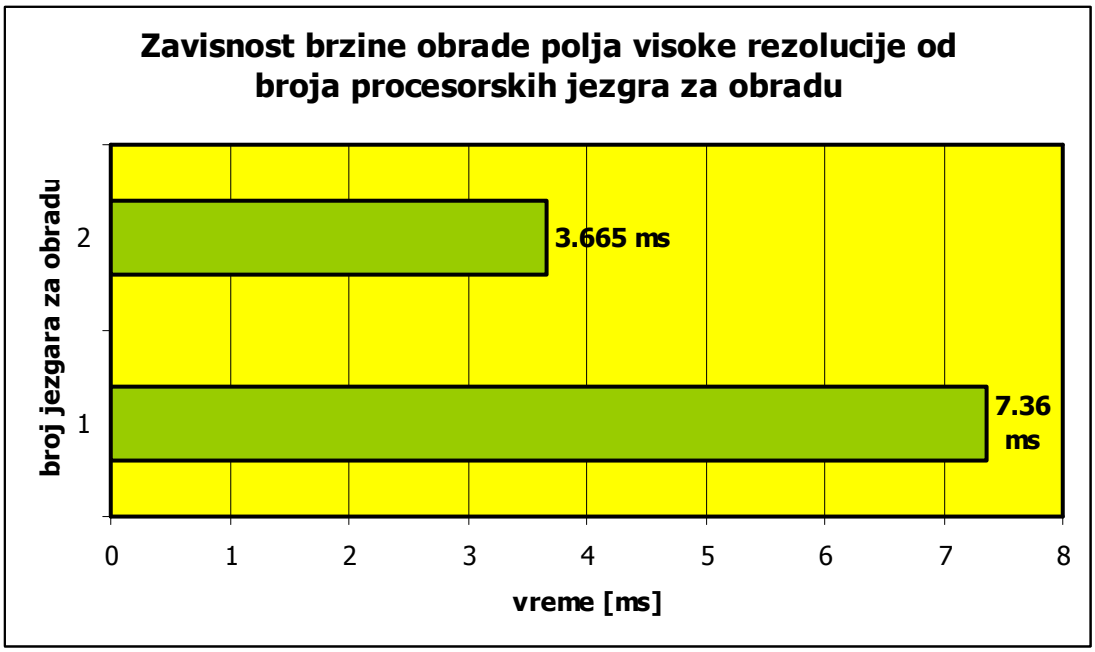
Računar na kome je aplikacija testirana	
Procesor:	Intel core 2 Duo 8200, 2.67 GHz, FSB 1333 MHz, 6 MB L2 cache
Matična ploča:	MSI P35 Neo3
RAM memorija:	2x 1 GB DDR2, 800 MHz

Cilj zadatka bio je da se polje visoke rezolucije (rezolucija 1920x540 tačaka) kompesuje u realnom vremenu. Slika 5.1 ilustruje dobijene rezultate pri testiranju HuffYUV enkodera za polje visoke rezolucije.

broj procesorskih jezgara	naziv sekvence	dimenzije	broj polja	brzina obrade	faktor kompresije
1	hdboat	1920x540	30	7.360 ms	2.906
	hdparkrun	1920x540	10	9.700 ms	1.349
2	hdboat	1920x540	30	3.665 ms	2.906
	hdparkrun	1920x540	10	4.800 ms	1.349

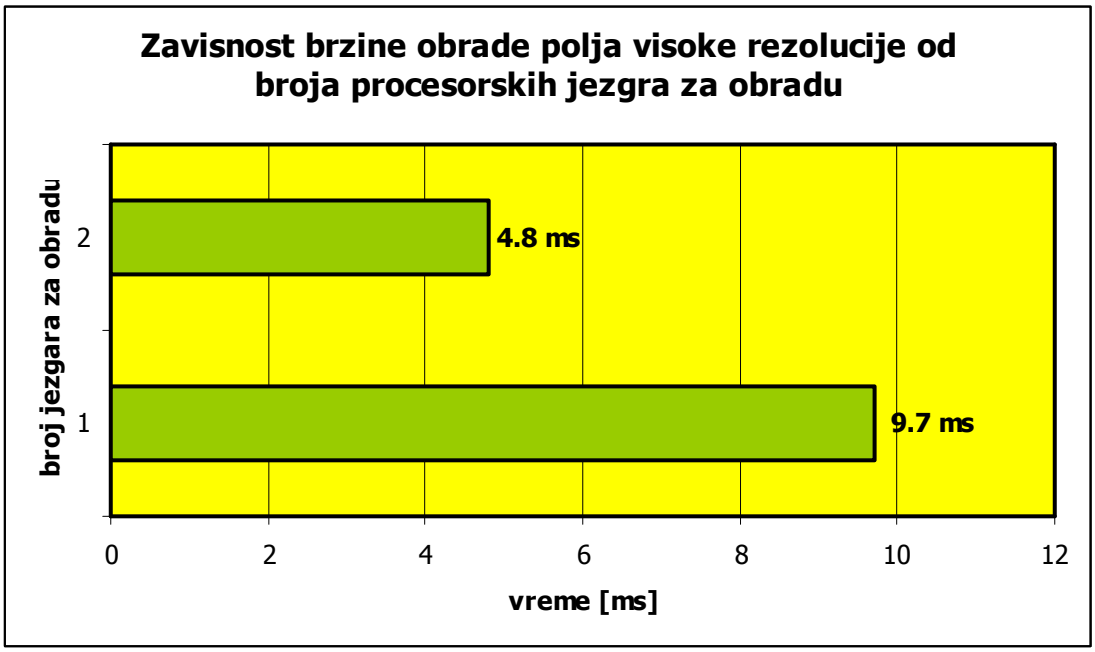
Slika 5.1. Rezultati pri testiranju aplikacije za polje visoke rezolucije

Slika 5.2. ilustruje zavisnost brzine obrade od broja jezgara za obradu, za *hdboat* DVS datoteku.



Slika 5.2. Rezultati pri testiranju aplikacije za polje visoke rezolucije

Slika 5.3. ilustruje zavisnost brzine obrade od broja jezgara za obradu, za *hdparkrun* DVS datoteku.



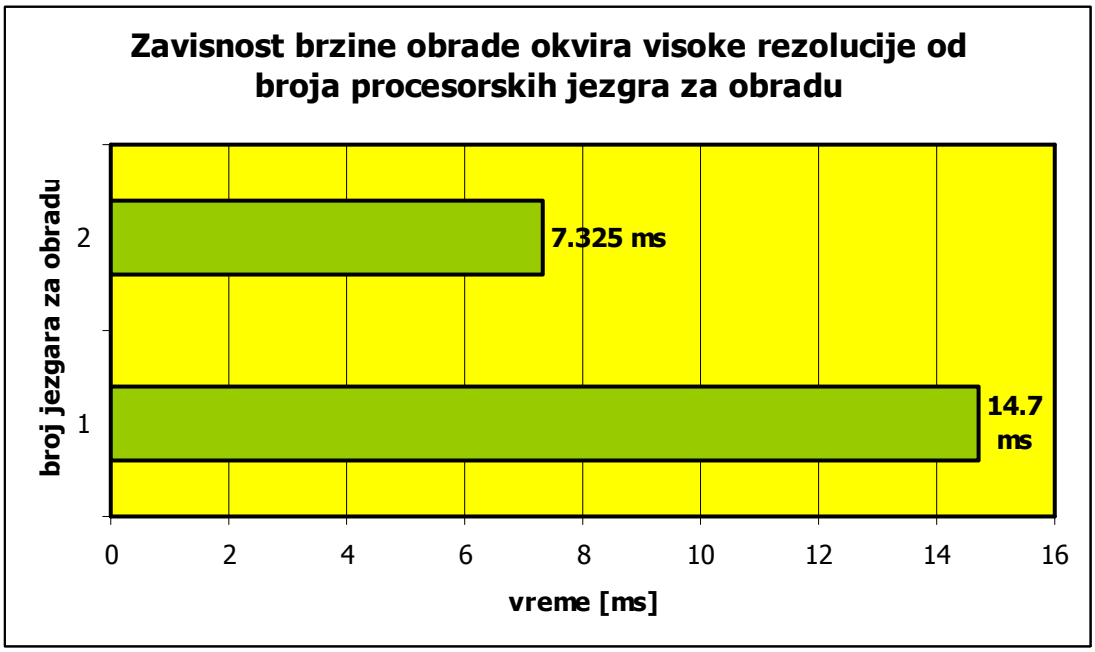
Slika 5.3. Rezultati pri testiranju aplikacije za polje visoke rezolucije

Aplikacija je pored polja visoke rezolucije, testirana i za okvir (eng. frame) visoke rezolucije (1920x1080 tačaka), iako je on dva puta veći (po količini podataka koji se trebaju kompresovati) od polja visoke rezolucije. Slika 5.4 ilustruje dobijene rezultate pri testiranju HuffYUV enkodera za okvir visoke rezolucije.

broj procesorskih jezgara	naziv sekvence	dimenzije	broj okvira	brzina obrade	faktor kompresije
1	hdboat	1920x1080	30	14.700 ms	2.899
	hdparkrun	1920x1080	10	19.430 ms	1.360
2	hdboat	1920x1080	30	7.325 ms	2.899
	hdparkrun	1920x1080	10	9.660 ms	1.360

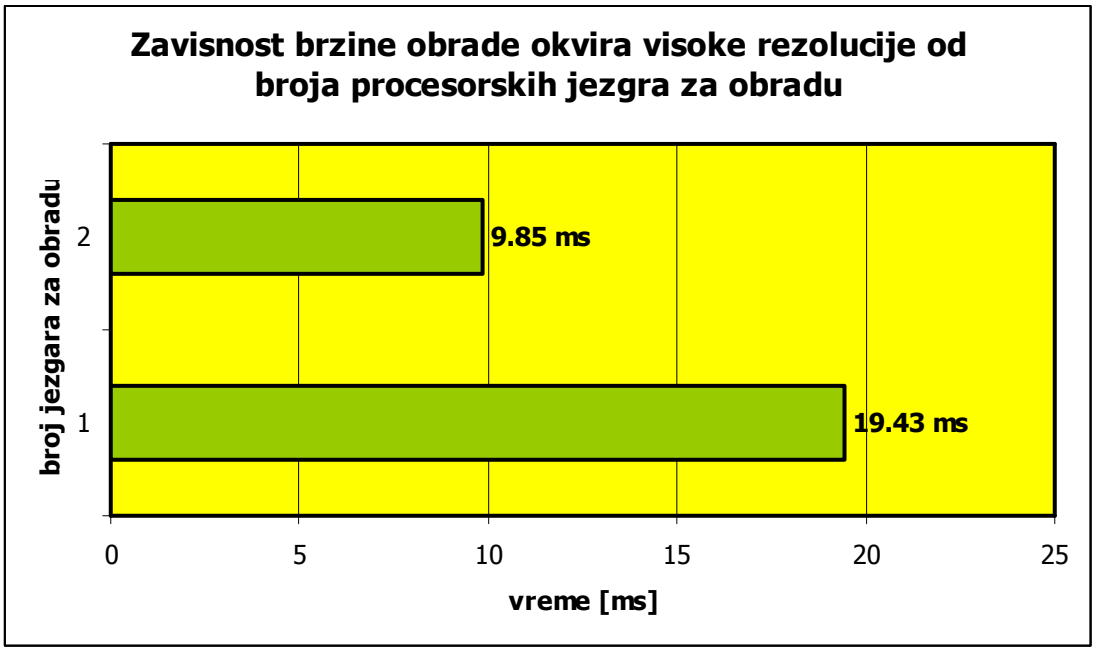
Slika 5.4. Rezultati pri testiranju aplikacije za okvir visoke rezolucije

Slika 5.5. ilustruje zavisnost brzine obrade od broja jezgara za obradu, za *hdboat* DVS datoteku.



Slika 5.5. Rezultati pri testiranju aplikacije za okvir visoke rezolucije

Slika 5.6. ilustruje zavisnost brzine obrade od broja jezgara za obradu, za *hdparkrun* DVS datoteku.



Slika 5.6. Rezultati pri testiranju aplikacije za okvir visoke rezolucije

6. Zaključak

Tekstom zadatka zahteva se da obrada polja visoke rezolucije bude ispod 16 ms. Cilj zadatka je ostvaren. Polje visoke rezolucije (1920x540 tačaka) je kompresovano u realnom vremenu. Vreme obrade varira, u zavisnosti od konkretnog sadržaja u okviru polja, ali je daleko ispod referentne vrednosti (što je ilustrovano slikom 5.1.).

Slika 5.2. prikazuje da je za sekvencu *hdboat*, vreme obrade na dva procesorska jezgra 2.0081 puta brže u odnosu na obradu na jednom procesorskom jezgru. Slika 5.3. prikazuje da je za sekvencu *hdparkrun*, vreme obrade na dva procesorska jezgra 2.0208 puta brže u odnosu na obradu na jednom procesorskom jezgru. Na osnovu ova dva rezultata, zaključuje se da vreme obrade linearno opada sa porastom broja procesorskih jezgara.

Nakon što je cilj zadatka ostvaren, pošlo se korak napred. Pokušana je kompresija okvira visoke rezolucije (1920x1080 tačaka) u realnom vremenu na jednom procesorskom jezgru. Slika 5.4. prikazuje rezultate dobijene kompresijom okvira visoke rezolucije. Na osnovu slike, zaključuje se da je obrada okvira visoke rezolucije u realnom vremenu moguća, ali samo za određene sekvence. Takođe, sa slike 5.4. zaključuje se da je obrada okvira visoke rezolucije u realnom vremenu, za svaku video sekvencu, moguća sa brojem jezgara većim od jedan.

Slika 5.5. prikazuje da je za sekvencu *hdboat*, vreme obrade na dva procesorska jezgra 2.0068 puta brže u odnosu na obradu na jednom procesorskom jezgru. Slika 5.6. prikazuje da je za sekvencu *hdparkrun*, vreme obrade na dva procesorska jezgra 1.9725 puta brže u odnosu na

obradu na jednom procesorskom jezgru. Na osnovu ova dva rezlutata, zaključuje se da vreme obrade linearno opada sa porastom broja procesorskih jezgara.

Na faktor kompresije i brzinu obrade, direktno utiče konkretni sadržaj sekvence. Slika 5.1. prikazuje faktore kompresije za dve različite sekvence. Faktor kompresije je veći, ukoliko su varijacije između susednih tačaka u okviru polja manje, odnosno faktor kompresije je manji, ukoliko su varijacije između susednih tačaka u okviru polja veće. Isto važi i za brzinu obrade.

Faktor kompresije i brzina obrade diktirani su kako konkretnim sadržajem polja, tako i izborom tabele enkodovanja. Eventualnim izborom druge tabele enkodovanja brzina obrade i faktor kompresije bi varirali. Zato bi najbolje bilo imati promenljivu tabelu enkodovanja, koja bi se pravila za svaku od obrađivanih sekvenci ponaosob.

7. Literatura

[1] *Intel® C++ Intrinsics Reference*, Document Number: 312482-002US. http://linux.iingen.unam.mx/pub/Documentacion/Programacion/C/Intel_C_Compiler_v10/Intrinsics%20reference.pdf

[2] YUY2 format podataka, http://en.wikipedia.org/wiki/YUV

[3] *Intel® Hyper-Threading Technology*, Technical User's Guide, http://www.cslab.ntua.gr/courses/advcomparch/material/readings/Intel%20Hyper-Threading%20Technology.pdf