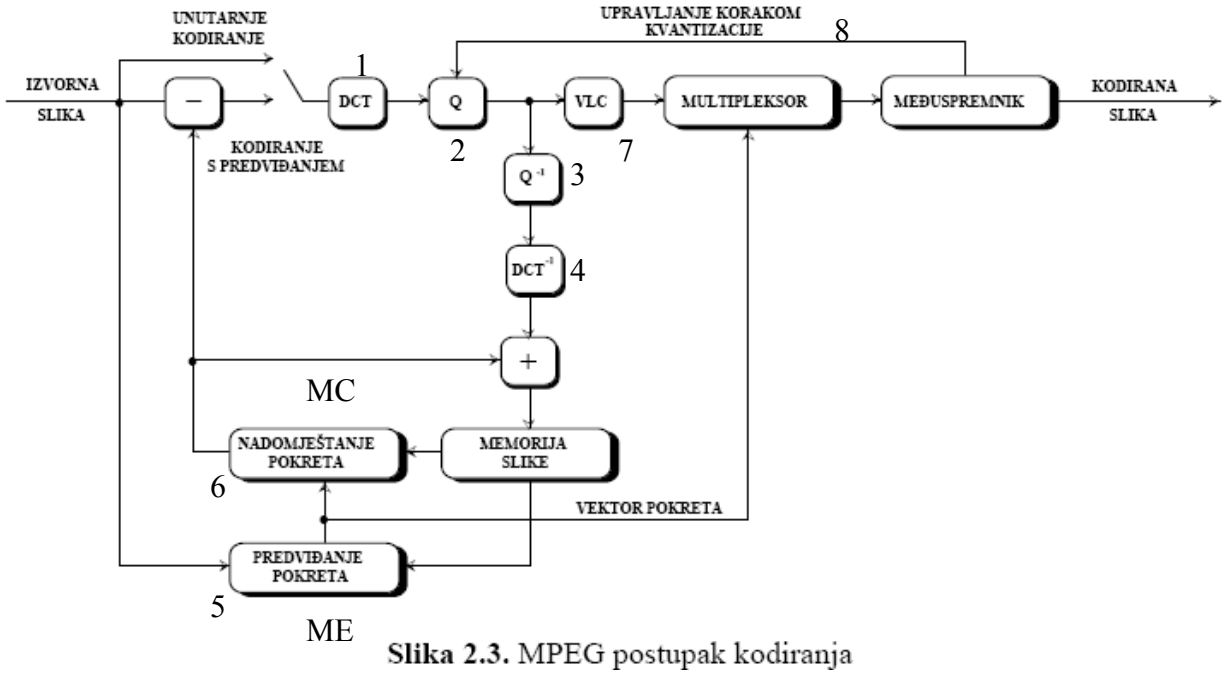
1. BLOK SHEMA MPEG KODIRANJA:



Slika 2.3. MPEG postupak kodiranja

1. DCT nad blokovima
2. Kvantizacija blokova
3. Inverzna kvantizacija
4. Inverzna DCT transformacija
5. Procjena pokreta u P i B slikama
6. Nadomještanje pokreta u P i B slikama
7. Kodiranje toka podataka varijabilnom duljinom kôda
8. Kontrola kvantizacije

⇒ **Povratna veza nakon entropijskog kôdera** (VLC ili ENC) **služi za određivanje koraka kvantizacije.**

2. MC (MOTION COMPENSATION), ME (MOTION ESTIMATION)

ME (PROCJENA POKRETA) - dio procesa nadomještanja pokreta predviđanjem u kojem se određuju vektori pokreta.

Provodi se usporedba odgovarajućih makroblokova dviju uzastopnih slika, kako bi se odredio smjer i udaljenost pomaka tih područja, jedne slike u odnosu na drugu.

Smjer i udaljenost iskazuju se pomoću dvodimenzionalnog vektora, VEKTORA POKRETA.

MC (NADOMJEŠTANJE POKRETA) – koristi vektore pokreta dobivene u procesu EC, za dobivanje trenutne slike.

$\Rightarrow$ proces nadomještanja pokreta:

EC procjenjuje vektore pokreta na temelju odnosa makroblokova trenutne i referentne slike.

MC na temelju vektora pokreta i prethodne slike stvara sliku s nadomještenim pokretom.

Pogreška nastala tim procesom računa se kao razlika trenutne slike i slike nastale procesom predviđanja.

Nad tom pogreškom vrše se DCT i nakon toga kvantizacija, te se kvantizirana pogreška kodira i prenosi.

Uz kodiranu pogrešku prenose se i vektori pokreta.

PROCJENA POKRETA:

$\rightarrow$ izdvajanje dijelova slike (segmentacija pokreta)

$\rightarrow$ opisivanje pokreta (procjena)

$\rightarrow$ izvodi se u kôderu

KOMPENZACIJA POKRETA:

$\rightarrow$ korištenje rezultata procjene pokreta

$\rightarrow$ izvodi se u dekôderu

3. RAČUNALNO NAJZAHTJEVNIJA OPERACIJA MPEG KÔDERA:

$\rightarrow$ postupak procjene pokreta nadomještanjem (PROCJENA i KOMPENZACIJA POKRETA).

4. KONVERZIJA RGB $\rightarrow$ YUV:

JPEG enkoder – postupak:

1. Konverzija RGB $\rightarrow$ YUV
2. Poduzorkovanje (U i V se uzrokuju manjom frekvencijom tako da cijela slika bude u formatu 4:2:0)
3. Blokovska DCT
4. Kvantizacija
5. Cik-cak grupiranje
6. Entropijsko kodiranje

Konverzija se odvija prije postupka kodiranja.

Y komponenta je bitna za percepciju cjelokupne slike, dok krominantne komponente ne nose puno informacija, te je bitno luminantnu komponentu sažeti uz što manje gubitaka.

Procjenu pokreta dovoljno je obaviti samo na luminantnoj komponenti.

5. BROJ OPERACIJA KOD PUNOG ALGORITMA KONVERZIJE RBB $\rightarrow$ YUV:

DCT: Za svaki $N \times N$ blok:

- množenja: $N \cdot N \cdot (N \cdot N)$
- zbrajanja: $N \cdot N \cdot \left[(N \cdot N) - 1\right]$

Konverzija RGB → YUV (YCrCb):

R, G, B, Y, U, V su matrice vrijednosti pojedinih komponenti.

$$Y = (0{,}257 \cdot R) + (0{,}504 \cdot G) + (0{,}098 \cdot B) + 16$$
$$U = -(0{,}148 \cdot R) - (0{,}291 \cdot G) + (0{,}439 \cdot B) + 128$$
$$V = (0{,}439 \cdot R) - (0{,}368 \cdot G) - (0{,}071 \cdot B) + 128$$

⇒ 9 množenja + 9 zbrajanja, za svaki piksel (+dohvat, + spremanje)

6. SEPARABILNOST 2D DCT-A:

Za smanjenje broja operacija u izračunu DCT-a koristi se matematičko svojstvo **SEPARABILNOST**, tj. 2D DCT može se računati kao dvije uzastopne 1D DCT transformacije → prva se DCT provede po recima, druga po stupcima.

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cdot \cos\left(\frac{(2x+1)u\pi}{2N} \right)$$

⇒ teoretski, za svaki piksel jednog bloka imamo N množenja i N-1 zbrajanja
+ N izračuna funkcije kosinus (1 dijeljenje, 3 množenja, zbrajanje).

Za smanjenje broja operacija koriste se i računarske metode (kao primjerice, *lookup* tablica).

7. MEĐUBLOKOVSKA I UNUTARBLOKOVSKA KOMPRESIJA:

Metode DCT kompresije i kvantizacije koje se obavljaju nad I-slikama, čine postupak UNUTARNJEG KODIRANJA.

Metode DCT kompresije i kvantizacije obavljaju se i nad P-slikama i B-slikama, ali se za ovaj tip slika uz to koriste i postupci predviđanja, pa se ovakav način kodiranja naziva KODIRANJE S PREDVIĐANJEM.

Metode predviđanja spadaju u MEĐUBLOKOVSKU KOMPRESIJU.

8. VEKTOR POMAKA:

Najčešće, vektor pomaka računa se za luminantnu (Y) komponentu.

9. PODRUČJE PRETRAŽIVANJA:

Područje pretraživanja pri procjeni pokreta, je okolina trenutnog makrobloka, tj. područje oko makrobloka, čije se granice protežu do p (pri čemu se parametar p zadaje) piksela lijevo, desno, gore i dolje u odnosu na granice trenutnog makrobloka. Tako definirano područje pretraživanja naziva se PROZOR PRETRAGE.

⇒ Za sliku 1024x1024 najčešće područje pretrage pri procjeni pokreta nikako nije cijela slika, tj. područje 1024x1024.

10. ALGORITMI PRETRAŽIVANJA:

podjela: → ALGORITMI POTPUNOG PRETRAŽIVANJA
→ NEPOTPUNI (BRZI, NAPREDNI) ALGORITMI :

- LOGARITAMSKO PRETRAŽIVANJE (LOG)
  0. step_size S
  1.Pozicioniraj se u centar (0,0)
  2.Pretraži četiri za S udaljena od centra (2 horizontalno, 2 vertikalno)
  3. novi_centar = best match
  4. Ako je novi_centar == stari_centar: S=S/2
  5. Ako je S=1: idi na 6, inače: idi na 2
  6. Pretraži 8 susjednih piksela

- PRETRAŽIVANJE U TRI KORAKA (3SS)
- ORTOGONALNO PRETRAŽIVANJE (ORT)
  - slično kao 3SS:
    0. step_size S = WS/2
    1.Izračunati SAE za dvije pozicije za S udaljene horizontalno od izvora
    2. Manji odabrati za novi izvor
    3.Ponoviti [1:2] vertikalno
    4. Manji odabrati za novi izvor
    5.S=S/2
    6.Ponavljati [1:5], dok nije S=1

- ALGORITAM GRADIJENTNOG SPUSTA TEMELJEN NA BLOKOVIMA (BBGDS) – loš za velike vektore pomaka, dobar za ujednačeno raspoređene male pokrete.

POTPUNO PRETRAŽIVANJE (FS, FULL SEARCH) - svojstva :

- pretražuje sve točke unutar područja pretrage
- računski vrlo zahtjevan
- daje optimalan rezultat

→ daje optimalan rezultat, ali u praksi je slabo primjenjiv ⇒ uglavnom se implementira hardverski (HW kôderi).

11. 3SS (THREE STEP SEARCH) – PRETRAŽIVANJE U TRI KORAKA:

Definirajmo:

parmetar_p … određuje koliko dodatnih piksela, zajedno s trenutnim blokom, čini prozor pretrage.

br_elem ... broj elemenata oko središnjem koje se svakim korakom ispituje
distance_step … pomak bloka koji se ispituje, u odnosu na onaj, za koji tražimo vektor pokreta.
MBi … makroblok trenutne slike, za koji tražimo vektor pokreta
MBj_ref … makroblok slike, u odnosu na koji procjenjujemo pomak makrobloka trenutne slike

Postupak:

I. Odredimo:

$br\_elem = \lfloor log_2(parametar\_p + 1) \rfloor$

$distance\_step_{max} = 2^{(br\_elem-1)}$ (maksimalna udaljenost bloka pretrage, ujedno, i broj koraka u kojima se za jedan makroblok, traži njemu odgovarajući)

II. Za svaki makroblok trenutne slike, ponavljamo sljedeći algoritam:

1) Za *MBi* u trenutnoj slici odredimo *MBi_ref* , tj. u referentnoj slici se pozicioniramo na mjesto makrobloka *MBi*

2) Odredimo prozor pretrage (*WS*) kao područje za *parmetar_p* udaljeno od gornjeg, lijevog, donjeg i desnog ruba *MBi_ref*

3) Postavimo $distance\_step = distance\_step_{max}$

3 - a) Za *br_elem*, na udaljenosti *distance_step,* u svim smjerovima od centralnog piksela *MBi_ref* bloka, postavimo blokove *MBj_ref*.
(Primjerice, za $br\_elem = 3$, što se dobije za $parametar\_p = \pm 8$, postaviti ćemo centre 8 *MBj_ref* istih dimenzija na pozicije za 4 piksela pomaknuta u odnosu na centar trenutnog bloka)

3 - b) Za svaki od $br\_elem \times br\_elem$ tako dobivenih blokova (uključujući i centralni *MBi_ref*) , računamo neku mjeru distorzije (primjerice, MSE), u odnosu na *MBi*.

3 - c) Blok *MBj_ref*, za koji se u koraku b) ustvrdilo da ima najmanju distorziju, postavljamo za centralni.
Korak *distance_step* prepolovljujemo (*distance_step = distance_step/2*).

3 - d) Ponavljamo korake 3-a) do 3-c) dok god je $distance\_step \geq 1$

4) Kada distance_step postane 1, dobit ćemo točnu poziciju bloka koji odgovara trenutnom, te se na temelju te pozicije računa vektor pomaka za trenutni blok.

12. BRZI ALGORITMI:

→ Prednosti: - manje potrebnih operacija

→ **Nedostatci**: - veće razlike podataka koje treba kodirati (izlazna količina podataka je veća)

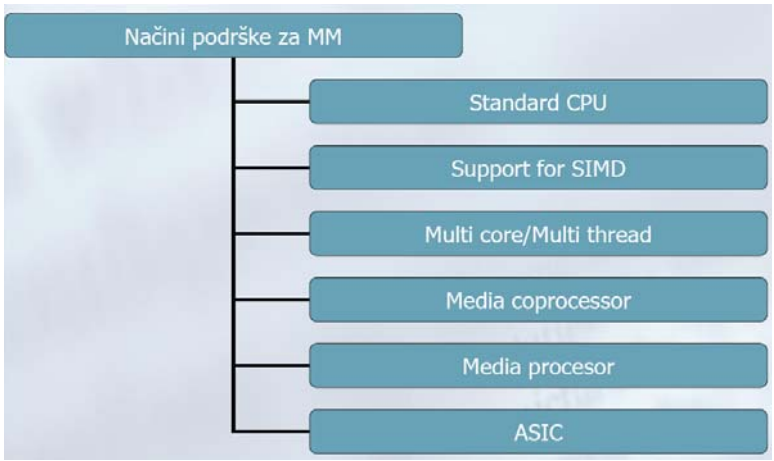
- (dodatno) učinkovitost pojedinog algoritma ovisi o vrsti sadržanog pokreta → nisu svi jednako dobri za sve tipove pokreta (zato se radije koriste adaptivni algoritmi)

13.PROGRAMSKA PODRŠKA ZA MM:

Programi u višim programskim jezicima, sporije se izvode od kôda prevedenog za ciljni procesor.

Te općenito, viši programski jezici, generiraju kod koji je sporiji u odnosu na kôd pisan u kombinaciji s asemblerom.

14. NAČINI PODRŠKE ZA MM:



USPOREDBA:

1) Standardni CPU:
   - izvodi se jedna ALU operacija po periodu. (Širina operacije u bitima, određena je širinom ALU jedinice.)
   - optimizacije se postižu programski (optimizacija petlji i „pažljiv“ način dohvata podataka).
   - eventualna sklopovska poboljšanja moguća su posebnom izvedbom (sklopovska podrška za množenje, ili za množenje sa zbrajanjem – Multiply and Accumulate).
2) Podrška za SIMD (SINGLE INSTRUCTION MULTIPLE DATA):
   - arhitektura toka podataka koja omogućuje obradu više podataka jednom naredbom.

Širina AL jedinice obično je veća od podataka koji se njome obrađuju (npr. 64-bitna ALU vrlo često radi s 8-bitnim podatcima).

⇒ velik dio procesorskog takta je neiskorišten, te bi bilo puno efikasnije reorganizirati ALU na način da istovremeno može obrađivati više podataka manje preciznosti. (npr. istovremeno 64 boolean tipa podataka, 2 32-bitna float podatka, 4 16-bitnih int, 8 8-bitnih int)

Uloga SIMD-a kod MM algoritama je upravo ta, da na gore opisani **„dijeljeni“ način, koristi** raspoloživu **AL jedinicu**, čime se **postiže znatno ubrzanje MM algoritama**.

15. (DVIJE BITNE)VERZIJE SIMD PROŠIRENJA KOD PROCESORA:

MMX (inicijali nemaju značenje): (Pentium s MMX i Pentium II)

- **uvedeno**:
  - → novi tip podatka: Packed 64 bit (zadovoljavajuće ubrzanje, neveliki zahvati na arhitekturi)
  - → zasebni MMX registri (64-bitni)
  - → novih 57 instrukcija
- sve naredbe rade nad 16-bitovnim podacima, samo neke nad 8 i 32 bitovnim
- **nedostatak**: Preklapanje sa FP (floating point) instrukcijama
- naredbe iznimno korisne za MM algoritme:
  - → wrap-aroud – modulo dijela koji se preljeva
  - → saturate – punjenje maksimalnom vrijednošću dijela koji se preljeva (zasićenje)

SSE (Streaming SIMD Extentions): (Pentium III)

- popravljeni najvažniji nedostaci MMX-a
- **uvedeno:**
  - → 8 potpuno novih 128-bit SIMD floating-point registara kojima se može direktno pristupati (preslikavanja kao kod MMX-a)
  - → (naslijeđeno od MMX) 50 instrukcija za rad s *packed floating-point* podatcima

→ (naslijeđeno od MMX) 8 naredbi za nadziranje toka „cache-iranja“ MMX and 32-bitovnih tipova podataka

→ 12 potpuno novih naredbi koje proširuju MMX skup naredbi.

16. AMDAHALOV ZAKON:

Amdahlov zakon ponajprije se odnosi na performanse paralelnog sustava, ali ima velik značaj i u cjelokupnom problemu ubrzanja, tj. optimizacije vremena izvođenja.

Amdahlov zakon koristi se previđanje maksimalno očekivanog poboljšanja cjelokupnog algoritma, ukoliko se poboljša jedan njegov dio. Poboljšanje se izražava poboljšanjem vremena izvođenja cjelokupnog programa, tj. ubrzanjem.

Izvorno, kako je Amdahl poboljšanje razmatrao u smislu ubrzanja paraleliziranjem, vrijeme izvođenja paraleliziranog programa , pri čemu smo njegov P-ti dio ubrzali N puta izvođenjem na N procesora, a proces paralelizacije cjelokupnom sustavu unosi kašnjenje od O_N, možemo izraziti kao: $T_{paralel} = \left\{(1-P) + \frac{P}{N}\right\} \cdot T_{total} + O_N$

Maksimalna skalabilnost (ubrzanje) algoritma, koje se sada postiže, jest:

$$U_{\max} = \frac{T_{total}}{T_{parallel}} = \frac{1}{(1-P) + \frac{P}{N}}$$

Pri tome, valja primijetiti da, u stvarnosti O_N nikada nije zanemariv, te da je maksimalno ubrzanje uvijek manje od ovdje predočenog teoretskog.

Općenito, pretpostavimo da je vrijeme izvođenja nekog programa t.

Pretpostavimo da je p ($0 < p < 1$) omjer dijela programa koji možemo ubrzati (N puta), naspram cijelog programa, pri čemu nije bitno na koji način to ubrzanje postižemo.

Novo vrijeme izvođenja cjelokupnog programa sada je: $t' = t(1-p) + p\frac{t}{N}$,

te je cjelokupno ubrzanje: $U = \frac{t}{t'} = \frac{t}{t \cdot \left((1-p) + \frac{p}{N}\right)} = \frac{1}{(1-p) + \frac{p}{N}}$.

Pretpostavimo sada da p možemo ubrzati proizvoljno puno i uzmimo da je N jako veliko ubrzanje ($N \to \infty$). Ukupno je ubrzanje u tom slučaju maksimalno i iznosi: $U_{max} = \frac{1}{(1-p)}$.

Odavde možemo uočiti osnovni smisao Amdahlovog zakona :

⇒ ubrzanje koje je moguće ostvariti u nekom programu, najviše ovisi o **UDJELU** dijela koji se ubrzava, u odnosu na cjelokupni program.

NAPOMENA: Pristup paraleliziranju koristi se kod sustava koji imaju:
- SMT (Simultaneous Multithreading)
- više procesora
- više jezgri unutar jednog procesora
- arhitekturu procesora predviđenu paralelnom izvođenju

17. MAKSIMALNA SKALABILNOST ALGORITMA (1):

Pretpostavke:
- 80% algoritma može se paralelizirati
- na raspolaganju su 4 procesora

Traži se: - maksimalna skalabilnost algoritma

⇒

P = 0.8
N = 4

$$U_{\max} = \frac{1}{(1-P)+\frac{P}{N}} = \frac{1}{0.2+\frac{0.8}{4}} = 2.5$$

18. MAKSIMALNA SKALABILNOST ALGORITMA (2):

Pretpostavke:
- 60% algoritma može se paralelizirati
- na raspolaganju su 6 procesora

Traži se: - maksimalna skalabilnost algoritma

⇒

P = 0.6
N = 6

$$U_{\max} = \frac{1}{(1-P)+\frac{P}{N}} = \frac{1}{0.4+\frac{0.6}{6}} = 2$$

19. MEDIJSKI KOPROCESOR:

→ zaseban procesor koji obrađuje samo zahtjevnije dijelove MM algoritma i potpuno je funkcijom prilagonjen samo obradi takvih dijelova.

Procesori opće namjene, imaju veće mogućnosti, ali nisu pogodni za primjenu u multimediji, jer imaju veliku potrošnju i/ili cijenu.

Svojstva:
- → u osnovi DSP (Digital Signal Processor) s mnoštvom periferija
- → jeftiniji od procesora opće namjene
- → manja potrošnja
- → !programibilan! (mogućnost poboljšanja i dodavanja aplikacija)
- → prilagođen određenoj porodici algoritama
- → uz njega, za ostale poslove, dostatan je jednostavan 32-bitovni CPU (npr. ARM)

→ performanse manje nego kod GPP-a (General Purpose Processor) jer su prilagođene pojedinim aplikacijama.

20. MEDIJSKI PROCESOR (I FPGA/ASIC):

MEDIJSKI PROCESOR:

→ procesor se projektira PRVENSTVENO za obradu MM podataka, ostale stvari obrađuje, ali tek sporedno

projektira se s ciljem visokih performansi za određeni skup algoritama, međutim, predstavlja i dalje procesor, što znači da ga je moguće programirati (prednost: promjena algoritama, dodavanje funkcionalnosti, …)

Svojstva:

→ izuzetno velike performanse (veće od GPP-a i koprocesora)
→ složen postupak programiranja
→ paralelno izvođenje naredbi
→ !programibilan!
→ podrška za opće zadatke podržana je od općeg procesora

Primjer: Trimedia TM3270 (NPX)

→ VILW (VERY LARGE INSTRUCTION WORD) arhitektura
→ neke naredbe omogućuju SIMD
→ VILW ima prednost nad superskalar arhitekturom, jer paralelizam nije određen od strane procesora, već od strane programera i kompajlera tijekom dizajna (⇒ procesor je jeftiniji i brži, manje rasipanje snage)

FPGA/ASIC:

→ posebni sklopovi za sklopovsko izvođenje algoritama:
FPGA (Field Programmable Gate Array)
ASIC (Application Specific Integrated Circuit) – NEmogućnost poboljšanja i nadogradnje, jedan dobavljač – potencijalna opasnost

- **RISC tip naredbe** – kratke, jednostavne, malobrojne
⇒ MEDIJSKI PROCESOR **NE KORISTI** RISC tip naredbe.

21. SLOŽENOST AL JEDINICE KOD VLIW MP:

→ zbog složenosti naredbi (VLIW arhitektura paralelizira naredbe, ALU ima izrazito veliku složenost.
Provjera nezavisnosti i raspodjela naredbi u paralelne tokove obavlja se tijekom kompajliranja, pa zahtjevi na hardver koji određuje raspored izvođenja, nisu toliko veliki kao kod ostalih arhitektura temeljenih na paraleliziranju (→ manja komplaksnost hardvera, ali veća komplaksnost kompajlera). VLIW arhitekture pojednostavljuju hardver, te su stoga jednostavnije i jeftinije od RISC arhitektura.

Međutim, VILW arhitektura koristi često više AL jedinica, te je složenost svake od njih (pogotovo u MM sustavima) veća, budući da su AL operacije koje izvode prilagođene posebnim primjenama.

22. ZAŠTITA DIGITALNOG SADRŽAJA:

FINGERPRINT:

→jedna od metoda koja se koristi kako bi se omogućilo otkrivanje izvora podataka (koji su neovlašteno kopirani)

- u **svaki** digitalni sadržaj umeće se jedinstveni kod koji identificira originalnog vlasnika te kopije.

Problemi pristupa:

- DSP algoritmi (i druge vrste „napada“) jednostavno ga otklanjaju
- Simetrični i asimetrični otisak (napad na privatnost)
  - Simetrični → poznat i prodavaču i kupcu – ne može se ustvrditi tko je izvor kopije ⇒ zaštita privatnosti kupca
  - Asimetrični → otisak nepoznat prodavaču, kompleksna izvedba, teška za implementaciju

WATERMARKING:

- umetanje informacije (fingerptint ili ostali tipovi podataka) u sadržaj
- mogu biti vidljivi ili nevidljivi

KRIPTO SUSTAVI:

- kriptiranje sadržaja simetričnim ili asimetričnim algoritmima

→ kod njih se javlja problem s verifikacijom ključeva

→ *on-line* su relativno uspješni, sok su *off-line* gotovo beskorisni

- ukoliko su zahtjevi za procesiranjem veliki, kriptiraju se samo neki ključni podatci.

23. RAZLOG KONVERZIJE RGB → YUV:

Glavni uzrok takvog tipa konverzije laži u ljudskom vizualnom sustavu (HVS, Human Visual System).

U postupku kodiranja koristi se neka (najčešće DCT za video signal i sliku) transformacija dvodimenzionalnih sustava, nakon čega se provodi kvantizacija. Cilj kvantizacije DCT koeficijenata je postizanje kompresije → kvantizacijom se odbacuju komponente koje nisu bitne za ljudski vizualni sustav.

S druge strane, HVS je jako osjetljiv na pogreške u DC koeficijentima i niskofrekvancijskim AC koeficijentima → osjetljivost na niskofrekvencijske komponente.

YUV sustav omogućuje razlaganje signala na luminantnu komponentu (Y) i dva signala razlike (Er-Ey i Eb-Ey). Signali razlike su visokofrekventni što dopušta da se više sažmu. S druge strane, ludski vizualni sustav osjetljiviji je na luminantnu komponentu, te je nju poželjno sažeti što manje.

Dakle, konverzijom signala iz RGB sustava u YUV signal postaje podatniji za „malverzaciju“ svojstvima koja su bitna ljudskom vizualnom sustavu i donose mu najviše informacije, a takve izmjene RGB sustav jednostavno ne dopušta.

⇒ U i V komponente mogu se prilično sažeti, bez znatnijeg gubitka na kvaliteti rekonstruiranog signala

⇒ U i V komponente moguće je poduzorkovati, te takve komponente koristiti u procesu kodiranja (automatski se smanjuje veličina podataka koje je potrebno obraditi, te je posljedica ubrzanje cjelokupnog procesa).

24. IZRAČUN UDJELA ALGORITMA ČIJE UBRZANJE DAJE ŽELJENO UBRZANJE CJELOKUPNOG ALGORITMA:

Pretpostavke:

- algoritam želimo ubrzati 2 puta
- određeni dio koda može se ubrzati 4 puta

Traži se: - udio koda u cjelokupnom algoritmu da bi dobili željeno cjelokupno ubrzanje

⇒

U = 2

N = 4

$$U = \frac{1}{(1-P)+\frac{P}{N}} \Rightarrow U = \frac{N}{N-NP+P} \Rightarrow N+P(1-N) = \frac{N}{U} \Rightarrow P(1-N) = \frac{N}{U} - N$$

$$\Rightarrow P(1-N) = \frac{N-UN}{U} \Rightarrow P = \frac{N(1-U)}{U(1-N)}$$

$$P = \frac{N(1-U)}{U(1-N)} = \frac{4(1-2)}{2(1-4)} = \frac{-4}{-6} = \frac{2}{3}$$

25. IZRAČUN UBRZANJA DIJELA ALGORITMA ZA ŽELJENO UKUPNO UBRZANJE:

Pretpostavke:

- algoritam želimo ubrzati 3 puta
- omjer djela koda koji želimo ubrzati

Traži se: - ubrzanje dijela koda da bi se dobilo željeno ubrzanje

⇒

U = 3

N = 3/4

$$U = \frac{1}{(1-P)+\frac{P}{N}} \Rightarrow U\left((1-P)+\frac{P}{N}\right) = 1 \Rightarrow \frac{UP}{N} = 1-U+UP \Rightarrow N = \frac{UP}{1+U(P-1)}$$

$$N = \frac{UP}{1+U(P-1)} = \frac{3 \cdot 3/4}{1+3\left(3/4-1\right)} = 9$$

26. ODABIR DIJELA KODA ZA OPTIMIZACIJU:

Najčešće se odabire dio koda čiji udjel je veći, ali za provjeru ipak uvrstiti u formulu.

27. PROFILING:

→ Postupak koji omogućuje pronalaženje dijelova kôda koji uzimaju najviše vremena, te pomaže u definiranju strategije ubrzavanja identificiranih dijelova.

Moguće strategije ubrzavanja:

- asambler
- razne biblioteke (boost, IPP)
- paralelizam
- novi pristupi → stream programming

→ ključno je dobro poznavanje arhitekture (organizacija ALU, cache, memorije)

U MATLABU:

- ocjena vremena izvođenja:
  - → odsječak koji se ispituje okruži se naredbama `tic` i `toc`
  - dobije se ispis trajanja u sekundama
- detaljniji uvid u postupak izvođenja:
  - u skripti koja definira funkciju, dodamo ispod deklaracije funkcije `profile on`; te `profile off`; na sam kraj funkcije.
  - nakon poziva funkcije u radnom prostoru postavimo naredbu `profile viewer`; čime dobivamo pregled profilinga za tu funkciju.

28. JEDNOSTAVNE OPTIMIZACIJE DCT-A NA BLOKU 8X8:

→ optimizacija SEPARACIJOM:

2D DCT računa se pomoću dviju uzastopnih 1D DCT-a, čime se smanjuje broj operacija.

→ optimizacija LOOK UP TABLICAMA:

- Iz formule za izračun DCT-a vidimo da su vrijednosti $\alpha(u)$ i $\alpha(v)$ jednake za svaki blok.
- Također, primijetimo da su vrijednosti kosinusa jednake za sve blokove.

⇒ možemo jednom izračunati te vrijednosti za 8x8 blok, pohraniti ih, te koristiti te vrijednosti za proračune nad svakim blokom (LOOK UP TABLICA).

→ optimizacija KORIŠTENJEM BRZIH TRANSFORMACIJA:

- Za izračun 2D DCT-a može se koristiti skalirana brza DFT uz svojstvo odvojivosti.

29. RAZLIČITI KVANTIZACIJSKI KOEFICJENTI :

Krominantne komponente sadrže većinom niskofrekvencijske komponente nakon DCT transformacije, te se kod njih podatci mogu više sažeti, budući da je veliki dio visokofrekvencijskih komponenti u ovim komponentama zanemariv. Visoke se komponente još dodatno mogu „prigušiti“ korištenjem posebno formiranih kvantizacijskih koeficijenata.

Luminantni signal ima veliki značaj za percepciju video signala i bitno ga je što manje sažeti. Dodatno „gušenje“ viših frekvencija u određenoj se mjeri provodi, ali ne u razmjerima kao za U i V komponentu, budući da je poželjno što je više moguće frekvencijskih komponenti očuvati.

30. ZASIĆENJE PERFORMANSI JEDNOPROCESORSKIH SUSTAVA:

Problemi koji su doveli do zastoja povećanja performansi jednoprocesorskih sustava:

→ prevelika disipacija snage
→ smanjena efikasnost
→ povećanje kompleksnosti
→ kašnjenje u interkonekcijama
→ usporen rast performansi
→ instrukcijski paralelizam ne utječe više na poboljšanje performansi

⇒ u jednoprocesorskim sustavima, tehnologija ostvarenja odavno je dosegnula nanometarske dimenzije, te je Mooreov zakon došao do određene granice, koja se tiče fizikalnih zakona. Iako je daljnje smanjenje čipovnih elemenata moguće, zbog kompliciranosti postupka i visoke cijene tehnologije potrebne za takvo ostvarenje, napuštena je ideja daljnjeg „mikro-mikriziranja“, i prešlo se na razvoje višejezgrenih i višeprocesorskih sustava.

31. VIŠEPROCESORSKI SUSTAVI VS. VIŠEJEZGRENI SUSTAVI:

Osnovna razlika je u tome što se kod VIŠEJEZGRENIH PROCESORA više jezgri integrira na istom čipu čime je konekcija procesa koje sa na njima izvode ubrzana, te nema efekta „uskog grla“, budući da je svakoj jezgri pridružena zasebna priručna memorija. Kod VIŠE PROCESORSKIH SUSTAVA procesori su fizički odvojeni, te dijele priručnu memoriju.

32. SPE U CELL BE ARHITEKTURI:

SPE (Synergetic Processing Elements)

→ obavljaju procese računanja (ima ih obično 8) dok se upravljanje vrši od strane PPE-a (Power Processing Elements), višedretvene jezgre.

→ osnovana uloga je obavljenje (sekvencijalno ili doista paralelno) paralelnih operacija čime se postiže efekt višedretvenosti → obrađuje dijelove istog procesa

Uloga:

→ obrada podataka, vektorsko procesiranje, paraleliziranje
→ priprema TLP-a na razini kompajlera (Thread Level Paralelism)

PARALELIZAM U MM APLIKACIJAMA !!!
- Podatkovni paralelizam DLP(datalevel)
- Paralelizam među zadatcima TLP (task, thread level)
- Protočni paralelizamPLP(pipelinelevel)
- Instrukcijski paralelizam ILP (instruction level)

33. Računarska Zahtjevnost Koraka JPEG Kompresije:

1) DCT
2) Konverzija RGB → YUV
3) Entropijsko kodiranje

34. Mjere Poremećaja U Postupku Procjene Pokreta:

Neke moguće mjere poremećaja dvaju blokova:

- Srednja kvadratna pogreška (MSE) - računski zahtjevna

$$MSE(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left[Y(x+i,y+j,t_1)-Y(x+d_x+i,y+d_y+j,t_0)\right]^2$$

- Srednji apsolutni poremećaj (MAD) – široko prihvaćena

$$MAD(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left|Y(x+i,y+j,t_1)-Y(x+d_x+i,y+d_y+j,t_0)\right|$$

- Broj podudarajućih slikovnih elemenata (MPC)

$$MPC(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}T\left[Y(x+i,y+j,t_1),Y(x+d_x+i,y+d_y+j,t_0)\right]$$

$$T(\alpha,\beta)=\begin{cases}1, & \text{za } |\alpha-\beta|\le T_0\\ 0, & \text{inace}\end{cases}$$

MPC (Matching Pel Count):
Pikseli bloka kandidata uspoređuju se s odgovarajućim pikselima referentnog bloka.
Oni pikseli čija je razlika manja od postavljenog praga se broje.
Blok za koje je broj sličnih piksela najveći određuje vektor pokreta.

35. SIMD u Računalnom Sustavu:

⇒ Osnovna ideja jest učinkovitije iskorištavanje procesorskog vremena paralelizacijom na razini instrukcije (instrukcijski paralelizam kakav se koristi u superskalarnim sustavima, tj sustavima koji paralelizam ostvaruju na jednom procesoru)