MISKOLCI EGYETEM

GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

INFOKOMMUNIKÁCIÓS CSATORNÁK TULAJDONSÁGAIHOZ ILLESZKEDŐ DIGITÁLIS VIDEÓ STREAM ÁTMÉRETEZÉS MÓDSZEREINEK KIDOLGOZÁSA ÉS VIZSGÁLATA

PhD ÉRTEKEZÉS

KÉSZÍTETTE:

FORMANEK BENCE

OKL. VILLAMOSMÉRNÖK

HATVANY JÓZSEF INFORMATIKAI TUDOMÁNYOK DOKTORI ISKOLA

A doktori iskola vezetője:

Prof. Dr. SZIGETI JENŐ, CSc

tanszékvezető, egyetemi tanár

Tudományos vezető:

Dr. CZAP LÁSZLÓ, PhD

tanszékvezető, egyetemi docens

MISKOLC

2013

Témavezetői ajánlás

Formanek Bence PhD eljárási kérelméhez

Az értekezés címe:

Infokommunikációs csatornák tulajdonságaihoz illeszkedő digitális videó stream átméretezés módszereinek kidolgozása és vizsgálata

A digitális kommunikáció világában meghatározó jelentőségű a forrásanyagok tömörítése az adatátviteli hálózatok terhelésének minimalizálása érdekében. A digitális műsorszórás és műsor elosztás szükségessé teszi az adatok átkódolását a hatékony nyalábolás céljából.

Formanek Bence és munkáltatója a CableWorld Kft. úttörő szerepet játszott a digitális műsorszóró rendszerek eszközeinek fejlesztésében. A témaválasztás időszerűségét az átlagos tájékozottságú tévénézők is igazolhatják.

A dolgozat feladat kitűzése szerint a videó transzkódolás célja a bemeneti digitális adatok minél szélesebb körű felhasználásával a számítási bonyolultság minimalizálása mellett a legjobb képminőség elérése. A legkorszerűbb többmagos digitális jelprocesszorok alkalmazása megkövetelte az alapvetően soros feldolgozás párhuzamosítását. A feladat szinte előzmény nélküli, a jelölt ezen a téren úttörő munkát végzett. A párhuzamosítás optimalizálásához alkotó módon kellett a processzormagok közötti kommunikációt kialakítani, ami a feladatmegosztás hatékonyságát jelentősen befolyásolja. A blokkolás és memóriamásolás mentes, üzenetküldés és osztott memória hibrid módszer az irodalomban ismeretlen, új tudományos eredmény.

Formanek Bence az új tudományos eredményeket fejlesztési feladatainak megoldása során dolgozta ki, amellyel bizonyította alkalmasságát a tudományos kutatásra. Munkáját önállóan, nagy szorgalommal végezte. Eredményeit folyóiratokban és konferencia kiadványokban publikálta.

A jelölt kutatási eredményei hozzájárultak a hardver és szoftver integrálásán alapuló új firmware modell kialakításához.

Fentiek alapján a fokozatszerzési eljárás megindítását javaslom.

Miskolc, 2013. július 2.

.....
Dr. Czap László
tudományos vezető

Köszönetnyilvánítás

Az értekezés a Hatvany József Informatikai Tudományok Doktori Iskola keretein belül, a CableWorld Kft. fejlesztési osztályán és a Miskolci Egyetem Automatizálási és Kommunikáció-technológiai Tanszékén végzett kutatómunkám eredményeit foglalja össze. A kutatás során barátaim, kollégáim támogatását élveztem. Köszönettel tartozom mindazoknak, akik lehetővé tették, illetve közvetlen vagy közvetett módon megkönnyítették a dolgozat elkészülését.

Külön szeretnék köszönetet mondani tudományos vezetőimnek **Dr. Czap László** úrnak és **Dr. Ádám Tihamér** úrnak, akik szakmai irányításukkal, segítőkész munkájukkal és támogatásukkal nélkülözhetetlen segítséget nyújtottak kutatómunkámhoz.

Szintén köszönetet mondok **Dr. Szigeti Jenő** professzor úrnak és **Dr. Tóth Tibor** professzor úrnak, a doktori iskola vezetőinek a támogatásáért.

Köszönettel tartozom **Zigó József** ügyvezető igazgató úrnak és a CableWorld Kft. munkatársainak, mert támogatták munkámat és lehetővé tették dolgozatom megírását.

Végül szeretném megköszönni családomnak és barátaimnak, hogy mindenben segítettek és biztattak munkám során.

Tartalomjegyzék

Ábrajegyzék

Táblázatok jegyzéke

Rövidítések jegyzéke

1-D	Egydimenziós
2-D	Kétdimenziós
AC	Alternating Current
AMD	Advanced Micro Devices
AVC	Advanced Video Coding
CBR	Constant BitRate
ccNUMA	Cache Coherent Non-Uniform Memory Architecture
Cell BE	Cell Broadband Engine
CIF	Common Intermediate Format, 352×288 (352×240)
CMP	Chip Multiprocessor
codec	Coder-Decoder
CR	Conditional Replacement
DC	Direct Current
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DSP	Digital Signal Processor
FPGA	Field-Programmable Gate Array
GOP	Group Of Pictures
GPGPU	General-Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
H.263	ITU-T H.263 Video coding for low bit rate communication
H.264	ITU-T H.264, ISO/IEC 14496-10, MPEG–4 Part 10, AVC
HD	High Definition
HT	Hyper-Threading
I/O	Input/Output
IBM	International Business Machines
IC	Integrated Circuit
IDCT	Inverse Discrete Cosine Transform
IP	Internet Protocol
IPC	Inter-Processor Communication
MAC	Multiply and ACcumulate
MC	Motion Compensation
ME	Motion Estimation
MP@ML	Main Profile @ Main Level, MPEG-2 vide
MPEG	Moving Picture Experts Group
MPEG-1	ISO/IEC 11172
MPEG-2	ISO/IEC 13818, ITU-T H.222/H.262

MPEG-4	ISO/IEC 14496
MPI	Message Passing Interface
MV	Motion Vector
NUMA	Non-Uniform Memory Architecture
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCR	Program Clock Reference
PPU	Cell BE, Power Processor Unit
PSNR	Peak Signal-to-Noise Ratio
PVM	Parallel Virtual Machine
QCIF	Quarter CIF, 176×144 (176×120)
RISC	Reduced Instruction Set Computing
SD	Standard-Definition
SIMD	Single Instruction, Multiple Data
SoC	System on Chip
SPU	Cell BE, Synergistic Processor Unit
SVT	Sveriges Television
TI	Texas Instruments
TM	Test Model
TS	Transport Stream, MPEG-2 Part 1, H.222
UDP	User Datagram Protocol
UMA	Uniform Memory Architecture
VBR	Variable BitRate
VLC	Variable Length Codes
VLD	Variable-Length Decoder
VLIW	Very-Long Instruction Word
VM	Verification Model

1 Bevezetés

A videó kommunikáció és a teljes videó átviteli lánc nagyon gyorsan fejlődik napjainkban, a tartalom-előállítástól az elosztó hálózaton és a műsorszóráson keresztül egészen a felhasználói készülékekig. A videó forgalom gyorsuló ütemben növekszik, ezért egyre nagyobb a hangsúly a hálózatok kapacitásának hatékony kihasználásán. Ezek a változások, és az új szolgáltatások megjelenése mozgatja a videó infrastruktúra megfelelő eszközeinek fejlődését is.

Digitális videó szolgáltatások jelentek meg olyan hálózatokon is, ahol korábban erre nem volt lehetőség. A vezetékes telefonon rendszerekben, az egyre nagyobb sávszélességű mobiltelefon hálózatokon, a szintén egyre gyorsabb Interneten és a televízió műsorszóró és műsorszétosztó hálózatokban. A korábban analóg videó átvitelre használt hálózatokon is digitális videó szolgáltatások jelentek meg, a műholdas műsorszórás digitalizálásával induló és a földfelszíni analóg televízió adás közeljövőben történő lekapcsolásával lezáruló digitális átállás következményeként.

Az egyes szolgáltatások és az különböző átviteli hálózatok más-más követelményeket állítanak, és paramétereket határoznak meg a videó átvitelhez. A videó átvitel paramétereinek konverzióját általában videó transzkódolásnak nevezik. Az egyes hálózatok között különböző gateway eszközök végzik a formátum és paraméter konverziót. Műsorszóró vagy műsorszétosztó hálózatokban alkalmazott gateway eszköz a remultiplexer, amely videó transzkódolást is végezhet.

Két aktuális és érdekes téma ötvözésével foglalkozik a dolgozat: videó transzkódolással és többmagos DSP multiprocesszorok programozásával. Videó transzkódolás során egy, adott paraméterekkel jellemezhető videó konvertálása történik egy új formátumra. A formátumot többek között olyan jellemzők definiálják, mint a bitsebesség, képsebesség, képfelbontás, kódolási szintaxis, a használt kódolási eszközök és a videó tartalom. A transzkódolást az különbözteti meg az egyszerű videó kódolástól, hogy egy transzkóder hozzáfér az eredeti videó kódolási paramétereihez és a bemeneti tömörített videóból kinyerhető statisztikákhoz. A kinyert bemeneti adatok ezután felhasználhatók a számítási feladat egyszerűsítéséhez és jobb minőségű videó előállításához. A videó transzkódolással foglalkozó kutatások fő célja a bemeneti videóból kinyert kódolási paraméterek és statisztikák minél intelligensebb felhasználása a számítási komplexitás minimalizálásához és a lehető legjobb képminőség eléréséhez.

Az elmúlt évtizedekben Moore törvénye alapján, az integrált áramkörben a tranzisztorok száma másfél-, kétévente megduplázódott. A megnövekedett tranzisztorszámot egyre bonyolultabb processzorok építésére felhasználva a rendszerek teljesítménye is növekedett. A processzorok a hagyományos egyszálú programokat egyre gyorsabban tudták végrehajtani, így a korábbi szoftverek is egyre gyorsabban működtek. A teljesítmény növekedés az utasítás szintű párhuzamosság egyre jobb kihasználásával volt elérhető, azonban ezt korlátozza a programok alapvetően soros jellege. A processzorok kialakításában az új irány, hogy a növekvő tranzisztorszámot egy IC-n belül több processzormag kialakítására használják fel, létrehozva egy chip szintű multiprocesszort. Ez a trend az általános célú processzorok mellett a jelfeldolgozó processzorokat is elérte, és megjelentek a többmagos DSP multiprocesszorok. Ahhoz, hogy a többmagos processzor architektúrák által elérhetővé vált számítási teljesítmény kihasználható legyen, a korábbi egyszálú problémákat, több párhuzamosan végrehajtható kisebb feladatra kell felbontani, és a processzormagok között gyors és hatékony kommunikációt kell kialakítani.

Egy általános soros probléma optimális párhuzamosítása nem megoldott kérdés. A többmagos DSP processzorok programozása pedig nem egyszerű feladat. Videó kódoló és dekódoló algoritmusok megvalósítása párhuzamos feldolgozással nem triviális feladat, jelentős kutatási téma. Videó transzkódolás párhuzamos megvalósítása azonban még kevésbé feldolgozott terület. Ez a dolgozat a videó transzkódolás, amely alapvetően soros probléma, többmagos DSP multiprocesszoron történő párhuzamos megoldási lehetőségeivel foglalkozik.

1.1 A dolgozat felépítése

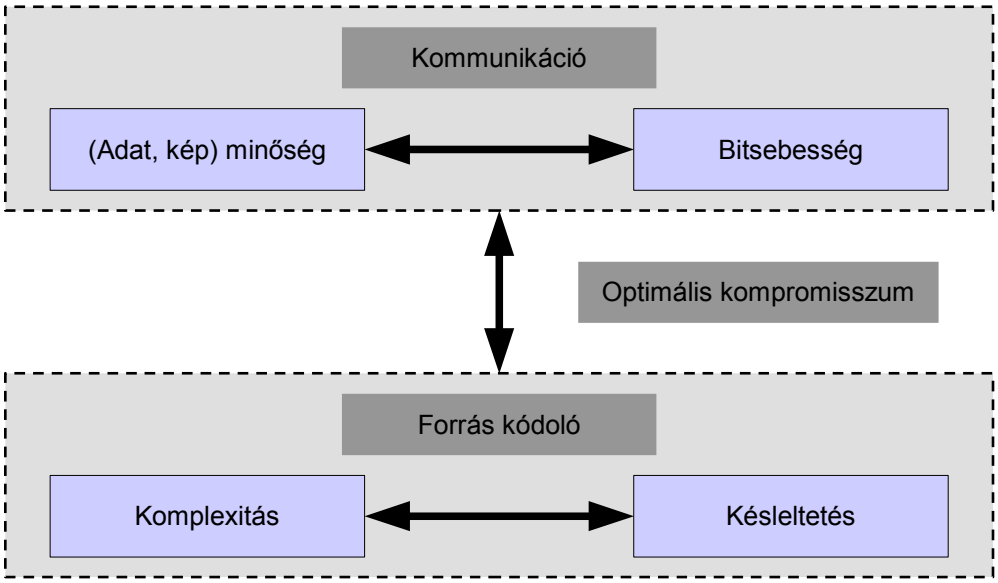
A dolgozat három részre tagolódik, az első rész azokat az ismereteket tartalmazza amelyek a disszertáció további részének megismeréséhez elengedhetetlenül szükségesek. A második fejezet összefoglalja a videó forráskódolás elméleti alapjait. A harmadik fejezet ismerteti a különböző videó transzkóder architektúrákat, azok különböző tulajdonságait. A negyedik fejezet bemutatja a párhuzamos feldolgozás lehetőségeit, a processzor és memória architektúrákat, a multiprocesszor rendszereket, kommunikációs és programozási modelleket és mindez hogyan használható videó feldolgozáshoz.

A dolgozat második része kutatási feladatokat és eredményeket tartalmazza. Az ötödik fejezetben ismertetem az általam kidolgozott adaptív, lineáris bitsebesség-kvantáló modellt, és vizsgálom a működését különböző paraméterekkel rendelkező videókkal. A hatodik fejezetben a többmagos DSP multiprocesszoron kialakított kommunikációs modellt ismertetem és vizsgálom a hatékonyságát. A hetedik fejezetben a videó transzkódolás párhuzamosításának lehetőségeit vizsgálom többmagos DSP multiprocesszoron, bemutatok három transzkóder modellt és különböző bemeneti videókkal vizsgálom tulajdonságaikat.

A dolgozat harmadik része csak a nyolcadik fejezetet tartalmazza amelyben összefoglalom az új tudományos eredményeket és bemutatom a továbbfejlesztés lehetőségeit.

2 Videó forráskódolás

A kommunikáció egyik alap problémája tekinthető úgy is, mint a forrás adatok átvitele egy adott bitsebességű csatornán a lehető legjobb minőségben, vagy úgy is mint a forrás adatok átvitele a lehető legkisebb bitsebességgel fenntartva egy adott átviteli minőséget [30]. Mindkét esetben az alapvető kompromisszum a bitsebesség és az adatátvitel minősége között van.



2.1. Ábra: Kódolási hatékonyság

Azt hogy egy forráskódoló rendszer milyen eredménnyel hozza meg ezt a kompromisszumot azt a kódolási hatékonyságával mérhetjük le. A kódolási hatékonyság az átviteli bitsebesség és az átvitt adatok megváltozása, torzítása közötti összefüggés. Egy ilyen forráskódoló rendszert gyakran codec-nek neveznek (mert egy kódoló (coder) és egy dekódoló (decoder) egységből áll). Egy videó codec ezek alapján a következő paraméterekkel jellemezhető:

- *A csatorna áteresztő képessége:* amit befolyásol a csatorna bitsebessége és az átviteli rendszer által használt protokoll és hibajavító kódolás által felhasznált bit többlet.
- *A dekódolt videó minőség romlása:* A torzítást elsősorban a videó codec tulajdonságai és az átviteli csatornában keletkezett bithibák határoznak meg.

Azonban egy gyakorlatban is megvalósított videó átviteli rendszerben további két szempontot is figyelembe kell venni.

- *Késleltetés (kezdeti késleltetés és végpontok közötti késleltetés):* a késleltetés tulajdonságait több paraméter is befolyásolja, a feldolgozási késleltetés, bufferelés késleltetése, a videó kódoló algoritmusból és a csatorna kódolásból adódó strukturális késleltetés, a csatornán való átvitel sebességéből adódó késleltetés
- *Komplexitás:* számítási komplexitás, memória kapacitás, memória hozzáférés, a videó codec komplexitása, a protokoll kezelés és a hálózat komplexitása.

Tehát egy forrás kódoló gyakorlati megvalósításának problémája a következőképpen foglalható össze. Adott a maximálisan megengedett jelkésleltetés és a maximálisan

megengedett komplexitás, érj el optimális kompromisszumot a jel (jelen problémában: a kép) minősége és az átviteli bitsebesség között a megvalósítandó alkalmazásokhoz elképzelt hálózati elrendezésekben [31].

A videó kommunikáció különböző területei jelentősen különböznek az optimális kompromisszum tekintetében. Videó konferencia vagy telefon rendszerek esetén a pont-pont közti késleltetésre erős megkötések vannak, az élő beszélgetés jellege miatt. Egy valósidejű videó kódolóra egy kábeltelevízió fejállomáson nem vonatkoznak olyan szigorú késleltetési követelmények, de minden képkockának időben el kell készülnie az átvitelhez. Offline videó tömörítésnél lényegében semmilyen késleltetési megkötés nincs, a kép minősége az elsődleges szempont.

Ez a munkapont az idővel is változik, ahogy Moore törvénye alapján a komplexitásra vonatkozó megkötések enyhülnek, és egyre nagyobb sávszélességű átviteli csatornák lesznek elérhetők. Az újabb és újabb videó szabványok ezt a megnövekedett komplexitást kihasználva érnek el hatékonyabb eredményt.

2.1 Videó ábrázolás

Digitális kép vagy egy videó képkocka tipikusan három téglalap alakú, egész értékű mintákat tartalmazó, kétdimenziós adat tömbből áll, egy-egy tömb a három komponensű színábrázolás komponenseihez. A kétdimenziós tömbök egyes térbeli pontjain található komponensek megfelelnek a képtartalom megfelelő pontjain lévő pixeleknek.

A képalkotó eszközök és a megjelenítők általában *R*, *G*, *B* komponenseket használják, azaz piros, zöld és kék színből állítják elő a színes képet. Videó kódolások leggyakrabban az *Y*, *Cb* és a *Cr* komponensekből álló színábrázolást, itt az *Y* komponens a világosságjel az egyes pixelek fényességét jellemzi, ami tulajdonképpen színes képhez tartozó fekete-fehér kép. A két színkomponens *Cb* és *Cr* a színkülönbség jelek, azt mutatják meg, hogy az egyes pixelek színe mekkora mértékben tér el a szürkétől a kék és a piros szín felé. A két színábrázolás egymásba átszámítható [4]:

$$\bar{E}_Y = 0.299\,\bar{E}_R + 0.587\,\bar{E}_G + 0.114\,\bar{E}_B\,, \tag{2.1}$$

$$\bar{E}_{PR} = \frac{0.701\,\bar{E}_R - 0.587\,\bar{E}_G - 0.114\,\bar{E}_B}{1.402}\,, \tag{2.2}$$

$$\bar{E}_{PB} = \frac{-0.299\,\bar{E}_R - 0.587\,\bar{E}_G + 0.886\,\bar{E}_B}{1.722}\,. \tag{2.3}$$

Ahol $\bar{E}_R$, $\bar{E}_G$ és $\bar{E}_B$ gamma korrigált analóg *R*, *G* és *B* színkomponens jelek, melyek értéke 0.0 és 1.0 között változhat, $\bar{E}_Y$ analóg világosságjel szintén 0.0 és 1.0 közötti értéket vehet fel. $\bar{E}_{PR}$ és $\bar{E}_{PB}$ jelek analóg színkülönbségjelek, normalizált értékük -0.5 és 0.5 között lehet. Az analóg jelekből egyenletes, általában 8 vagy 10 bites kvantálás után kapjuk a digitális *Y*, *Cb* és *Cr* jeleket.

$$Y = \mathrm{int}\left(\left(219\,\bar{E}_Y + 16\right) \times D\right) / D \tag{2.4}$$

$$Cr = \mathrm{int}\left(\left(224\,\bar{E}_{PR} + 128\right) \times D\right) / D \tag{2.5}$$

$$Cb = \mathrm{int}\left(\left(224\,\bar{E}_{PB} + 128\right) \times D\right) / D \tag{2.6}$$

A 8 és a 10 bites kvantálás miatti félreértések elkerülése végett a digitális jelek felső helyiértékű 8 bitjét tekintjük az egész résznek és a 10 bites kvantálás esetén fennmaradó 2 bitet fix pontos tört résznek. Így az *Y* jel 16 és 235, a színkülönbségjelek 16 és 240 közötti értékeket vehetnek fel, mert:

$$D = 2^{n-8} \tag{2.7}$$

tehát *D* az 1 vagy 4 értékű, ha *n* a kvantálásnak megfelelően 8 vagy 10.

A digitális kép pixeleihez tartozó minták kétfajta elrendezésben lehetnek a kétdimenziós adattömbökben: *progresszív* vagy a *váltott soros* módon. Ha úgy tekintjük a mintákat hogy azok két összefésült félképből származnak, egy felső és egy alsó félképből, akkor a felső félkép minden páros számú sort: $0, 2, \ldots, H-1$ (ahol a 0. a kép legfelső sora, és *H* az összes sorok száma), az alsó félkép minden páratlan számú sort tartalmaz: $1, 3, \ldots, H$ (a kép második sorával kezdve). Ha egy videó felvétel váltott soros formátumban készül, minden mintavételi időpontban a teljes kép helyett csak az aktuálisan következő félkép kerül rögzítésre, tehát egy teljes képhez kétszeres mintavételi időre van szükség. A teljes képre és egy félképre is gyakran hivatkoznak képként, attól függően, hogy a kódolási eljárás hogyan kezeli ezeket a képeket. Ha egy képen belül a két félkép különböző időpontban készült, a teljes képre mint váltott soros képre, egyébként pedig mint progresszív képre szoktak hivatkozni.

Az *Y*, *Cr*, *Cb* színábrázolás használatának a videótechnikában két oka van, egyrészt történelmi, másrészt gyakorlati. A történelmi, hogy a fekete-fehér televíziózásban csak az *Y* jelet használták, hasonlóan a fényképezéshez, amely szintén először fekete-fehér képek készítésével kezdődött. A gyakorlati ok pedig, amiből a történelmi is következik, hogy emberi látórendszer számára fontosabb a világosság információ mint a szín információ, tehát érdemes külön kezelni.

2.2. Ábra: Világosságjel és színkülönbség információ 4:2:0 felbontás esetén

A videó tömörítés első lépéseként ezt ki is van használva, legtöbbször olyan mintavételi struktúrát alkalmaznak, melyben a színkomponens tömbökben csak a megfelelő világosságjel tömbhöz képest negyed annyi minta van:

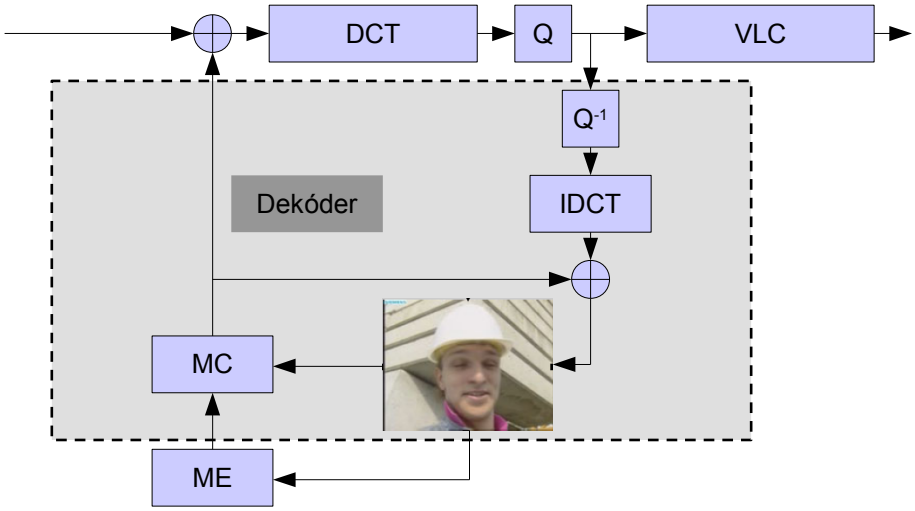
$$H_{Cr} = \frac{H_Y}{2}, \; H_{Cb} = \frac{H_Y}{2}, \tag{2.8}$$

$$V_{Cr} = \frac{V_Y}{2}, \; V_{Cb} = \frac{V_Y}{2}, \tag{2.9}$$

ezt 4:2:0 mintavételnek nevezik. Jobb minőségű videók esetén használják a 4:2:2 mintavételt, mikor csak vízszintes irányban történik alulmintavételezés, de a színkülönbségjel tömbök ugyanannyi sorból állnak mint a világosságjel, vagy a 4:4:4 mintavételt, mikor nincs alulmintavételezve a színinformáció.

2.2 Videó tömörítés

A modern videó kódoló módszerek nagy tömörítési arányt érnek el jó képminőség mellett. Ezek veszteséges tömörítő algoritmusok, tehát a pontos pixel értékek nem maradnak meg a kódolás után. A szabványos videó kódoló algoritmusok blokk alapú hibrid kódolási sémán alapszanak, amely természetes képszekvenciákban jelen lévő térbeli korreláció csökkentésére *transzformációs*, az időbeli redundancia csökkentésére *mozgáskompenzációval* segített *differenciális* kódolást jelent [2].



2.3. Ábra: Hibrid videó kódoló felépítése

A szabványok valójában csak a *bitfolyam szintaxisát* és a *dekódolási folyamatot* definiálják, a kódolási eljárást nem, tehát egy kódolónak az előírásoknak megfelelő bitfolyamot kell előállítania, a módszer azonban nincs szabályozva. Ez egyrészt jelentős rugalmasságot biztosít a kódolók készítői számára, hogy a követelményeknek megfelelően határozzák meg milyen kompromisszumokat hoznak a képminőség, a komplexitás és a fejlesztéshez szükséges idő, valamint a költségek között. Másrészt ez teszi lehetővé a kódolók és dekódolók és transzkódolók folyamatos fejlesztését, gyorsabb algoritmusok és módszerek kutatását.

Két fő kódolási módszer – bizonyos formátumoknál egyben képtípus – létezik: a *intra* kódolás és *inter* kódolás. Intra kódolás csak az adott képen lévő információkat használja a tömörítéshez, inter kódolás korábbi és későbbi képekből is használ információt. Csak intra kódolás alkalmazható értelemszerűen képkódoláshoz [7], [8], de létezik csak intra kódolt képeket tartalmazó videó is [11], [12].

2.2.1 Transzformáció

Blokk alapú kódolás esetén a képek egyenlő nagyságú, pl.: 8×8 pixel méretű négyzetekre lesznek felbontva, majd ezeken a blokkokon történik a transzformáció. Transzformáció alatt azt értjük, hogy a kép eredeti mintáinak kombinációjából új mintákat állítunk elő, általában lineáris kombinációval. A transzformáció célja egyrészt az egymáshoz közeli minták közötti redundancia csökkentése, amely lehetővé teszi a korrelálatlan minták egymástól független kódolását, másrészt a bemeneti minták legfontosabb jellemzőinek lehető legkisebb számú változóval való leírása. Tehát egy transzformációnak jó *dekorrelációs* és *energia koncentrációs* tulajdonsággal kell rendelkeznie. Ilyen tulajdonságokkal a szabványos kép és videó kódolásokban gyakran használt DCT rendelkezik. A 2-D $N \times N$ méretű mintákra a DCT definiciója (2.10), az inverz DCT vagy IDCT pedig az (2.11) [2]:

$$F(u,v) = \frac{2}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \frac{\cos((2x-1))u\pi}{2N} \frac{\cos((2y-1))v\pi}{2N}, \tag{2.10}$$

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2x+1)v\pi}{2N}, \tag{2.11}$$

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{ha } u, v = 0 \\ 1 & \text{egyébként} \end{cases}, \tag{2.12}$$

ahol x és y az eredeti minták koordinátái, u és v pedig koordináták a transzformációs térben, valamint: $u, v, x, y = 0,1,2\ldots, N-1$ és kép és videó kódolások esetén leggyakrabban $N = 8$.

A DCT transzformációnak további a gyakorlati megvalósítás szempontjából fontos tulajdonságai is vannak [33]. A *szeparálhatóság* lehetővé teszi a 2-D DCT számítását két lépésben, egymás utáni 1-D DCT számítást a kép soraira és oszlopaira külön-külön:

$$f(x,y) \rightarrow F(x,v) \rightarrow F(u,v). \tag{2.13}$$

A DCT *szimmetrikus* transzformáció, hiszen a sorokon és az oszlopokon végrehajtott transzformáció funkcionálisan megegyezik. Egy szeparálható és szimmetrikus transzformáció felírható a következő formában:

$$T = AfA \tag{2.14}$$

ahol A egy $N \times N$ méretű szimmetrikus transzformációs mátrix, f pedig $N \times N$ méretű a kép pixeleiből álló mátrix. Ez egy rendkívül hasznos tulajdonság mert lehetővé teszi a transzformációs mátrix offline számítását, majd az előre számított mátrix használatát a kép kódolásakor, ezzel nagyságrendekkel lecsökkentve a transzformáció számításigényét. A fenti tulajdonságok az inverz DCT számításra is jellemzők, tehát az inverz transzformáció felírható a következőképpen:

$$f = A^{-1}TA^{-1}, \tag{2.15}$$

mivel a DCT bázisfüggvényei ortogonálisak az A transzformációs mátrix inverze megegyezik a transzponáltjával $A^{-1} = A^T$, ami az inverz transzformációs mátrix előre számítását egyszerűsíti.

A DCT transzformáció hátránya videó kódolás szempontjából, hogy a pontos számítása lebegőpontos aritmetikát igényel. A gyakorlatban megvalósított algoritmusok általában egész aritmetikát használnak vagy az architektúra korlátai miatt vagy a számítási komplexitás alacsonyan tartása miatt. Ezek a DCT és inverz DCT transzformációk csak közelítő eredményt adnak, ami a különböző kóder és a dekóder implementációk miatt megfigyelhető képminőség romlást okoz. A közelítés megengedhető pontatlanságát szabvány határozza meg [6]. Újabb videó formátumokban elkerülendő ezt a hibát, már a szabvány nem DCT transzformációt ír elő, hanem egész aritmetikával számítható közelítő transzformációs mátrixot [3].

Gyors DCT algoritmusok általános célú és architektúra specifikus megvalósítása egy alaposan tanulmányozott terület [35], [36], [37].

2.2.2 Kvantálás

A *kvantálás* az a kódolási lépés ahol a tulajdonképpeni adatvesztés megtörténik. A lényege az egyes transzformációs *együtthatók pontatlanabb, azaz kevesebb bittel való ábrázolása*. A transzformáció energia koncentráló tulajdonsága és a pontatlanabb ábrázolás miatt a viszonylag kis amplitúdójú transzformációs – természetes képek esetén a nagyobb frekvenciájú – együtthatók nulla értékűek lesznek, amelyeket a következő lépésben a futamhossz kódolás lényegében eltávolít a kódolt videóból, mindezt a képminőség jelentős romlása nélkül.

A kvantálás mértékét két tényező befolyásolja, egy kvantálási mátrix $W[w][v][u]$ amely együtthatónként határozza meg a kvantálás mértékét és egy szorzó tényező q_s amely segítségével néhány bit átvitelével módosítható a kvantálás mértéke. Az MPEG-2 szabvány [2] szerinti inverz kvantálás:

$$F[v][u] = \left((2 \times F_Q[v][u] + k) \times W[w][v][u] \times q_s / 32\right), \text{ ha intra és } v, u \neq 0 \quad (2.16)$$

$$k = \begin{cases} 0 & \text{intra blokk} \\ sign(F_Q[v][u]) & \text{inter blokk} \end{cases}, \quad (2.17)$$

$$F[0][0] = q_I \times F_Q[0][0], \quad (2.18)$$

ahol $F[v][u]$ helyreállított transzformációs együttható mátrix, $F_Q[v][u]$ a kvantált együttható mátrix, a w segítségével választható ki a megfelelő kvantálási mátrix. Intra kódolt DC komponens külön kvantáló q_I segítségével számítható. Az egyenletek számítása egész műveletekkel történik.

2.2.3 Változó szóhosszúságú kódolás

A kétdimenziós kvantált együtthatókat tartalmazó mátrix egydimenziós vektorrá alakítása például cikk-cakk kiolvasással történik. A futamhossz kódolás a vektor nem nulla értékű elemeiből és az ezeket az elemeket megelőző nulla értékű együtthatók darabszámából számpárokat képez. A *változó szóhosszúságú kódoló* ezekhez a számpárokhoz különböző bitszámú kódszavakat rendel (2.19), az egyes számpárok a természetes videókban való statisztikus előfordulása alapján:

$$F_Q[v][u] \rightarrow F_{SQ}[n] \rightarrow F_{VQ}[m]. \quad (2.19)$$

2.2.4 Predikció

A videó kódolásokat az időbeli redundancia kihasználása különbözteti meg a képtömörítő algoritmusoktól, mivel így jelentősen javítható a kódolási hatékonyság. A videó tartalmakban előforduló nagy mennyiségű időbeni redundancia abból adódik, hogy a képtatalom nagy része általában jelentősebb változás nélkül ismétlődik az egymás utáni képkockákon.

Egy videó ezért hatékonyabban ábrázolható, ha csak a képen látható jelenet *változásait visszük át,* a teljes képtartalom ismételt kódolása helyett. Erre egy egyszerű módszer a feltételes kiegészítés, a CR kódolás [38]. Az első digitális videó kódolási szabványban – H.120, [9] – ez volt az egyetlen időbeli redundancia csökkentő módszer. A CR kódolás egy jelzés küldéséből áll, mely kijelöli a képtartalom ismételhető részeit, vagy új információk küldéséből a megváltozott területek cseréjére:

$$x_n = \begin{cases} x_n & \text{ha } \textit{Intra} \\ x_{n-1} & \text{ha } \textit{Skip} \end{cases} . \tag{2.20}$$

A CR lényeges hiányossága, hogy az ismételendő pixelek nem pontosíthatók.

Az előző képtartalom egy része gyakran jó közelítése az új kép megfelelő pixeleinek, de nem pontosan, a *különbség átvitelével* javítható a predikció hatékonysága:

$$x_n = \begin{cases} x_n & \text{ha } \textit{Intra} \\ e_n + x_{n-1} & \text{egyébként} \end{cases}, \tag{2.21}$$

ahol, mint a (2.20) egyenletben is x_n az aktuális kép pixeleinek értéke, x_{n-1} az előző kép megfelelő pixeleinek értéke, e_n a predikciós hibajel.

A legtöbb változást tipikusan, a kép síkján elmozduló objektumok okozzák. Kis mértékű elmozdulás is jelentősen megváltoztathatja a pixelek értékét, különösen az elmozduló objektumok határai közelében. Gyakran az aktuális képtartalom egy részének predikciója az előző kép néhány pixellel a kép síkjában elmozdított régiójának segítségével, jelentősen csökkentheti az átviendő pontosító pixelek értékét. A predikcióhoz használt képterület elmozdításának irányát és mértékét *mozgásvektorok* írják le. Az elmozdított képterületek használatát predikcióhoz *mozgáskompenzációnak* nevezzük, a kódoló által használt módszert, amivel a legjobb mozgásvektort próbálja megtalálni *mozgásbecslésnek*, tehát

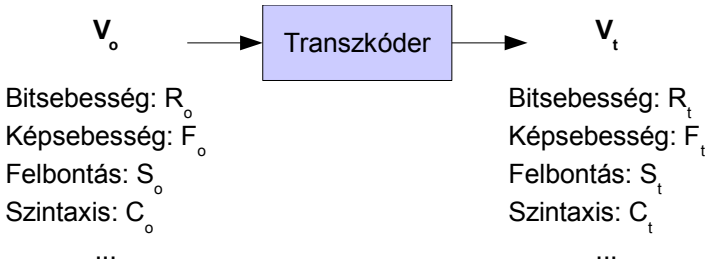
$$x_n = \begin{cases} x_n & \text{ha } \textit{Intra} \\ e_n + \mathcal{M}_n(x_{ref}) & \text{egyébként} \end{cases}, \tag{2.22}$$

ahol x_{ref} a referencia kép, nem feltétlenül az előző x_{n-1}, $\mathcal{M}_n()$ a mozgáskompenzáció operátor. A mozgáskompenzált predikció segítségével előállított javító, különbség pixeleket alkalmazó kódolását mozgáskompenzált predikciós maradvány vagy differenciális kódolásnak nevezzük. A mai formájában, ahogy a mai modern hibrid kódolásokban [2], [3], [10] megjelenik, [39]-ben publikálták először.

Megjegyezzük, hogy a mozgáskompenzált predikciós eljárások folyamatos fejlődése volt a legfontosabb okozója az egymás utáni kódoló generációk által elért kompressziós hatékonyság növekedésének. Az egyre kifinomultabb mozgáskompenzált predikciós eljárások használatának az ára azonban, a kódoló algoritmusok komplexitásának jelentős növekedése.

3 Transzkóder architektúrák

A videó transzkódolás egy adott videó konvertálása az *eredeti formátumról egy másik formátumra*. A formátumot olyan jellemzők definiálják, mint a bitsebesség, a képsebesség, a képfelbontás, a kódolási szintaxis, a használt kódolási eszközök és a videó tartalom. Ez látható a 3.1. ábrán. A bitsebesség a videó kommunikációs csatornán való átviteléhez szükséges másodpercenkénti bitek számát jelenti, a képsebesség a másodpercenként megjelenített képek száma, a képfelbontást az egyes képeken lévő pixelek száma és a képarány határozza meg. A kódolás szintaxisát valamelyik videó kódolási szabvány vagy ajánlás írja le (pl.: MPEG-1, MPEG-2, H.264), a szabvány vagy ajánlás által leírt kódolási eszközök közül az aktuális környezetben használható módszereket profilok definiálják.



3.1. Ábra: Videó transzkódolás

A videó transzkódereknek két fő típusa van: *homogén transzkóderek* és *heterogén transzkóderek*. A homogén transzkóderek azonos szintaxisú, azonos szabvány szerinti videók közötti konverziót végeznek. A konverzió lehet bitsebesség csökkentés, időbeli vagy térbeli felbontás – azaz képfelbontás vagy képsebesség – változtatás, lehet progresszív és váltott soros ábrázolás váltás, változó és állandó bitsebesség közötti konverzió. Tehát a videó tulajdonságai változhatnak, de a kódolás formátuma nem. A heterogén transzkóderek a videó szabványok közötti konverziót végzik a videó szintaxisának megváltoztatásával. Az átalakítás történhet a videó tulajdonságainak megváltoztatása nélkül is, de a heterogén transzkóderek a homogén transzkóderek feladatait is elláthatják.

A transzkódolást többek között az különbözteti meg az egyszerű videó kódolástól, hogy a *transzkóder hozzáfér az eredeti videó kódolási paramétereihez* és a bemeneti tömörített videóból viszonylag egyszerűen kinyerhető s*tatisztikákhoz*. Ezek az adatok felhasználhatók a számítási feladat egyszerűsítéséhez és jobb minőségű videó előállításához. A transzkódolás felfogható tulajdonképpen egy kétmenetes kódolási eljárásnak is: az első menet állította elő a bemeneti kódolt videó bitfolyamot, a második menet a transzkódoló által végrehajtott kódolás, amely az első menet által előállított információkat felhasználja a kódolási folyamathoz. Így egy transzkóder jobb minőségű videót is előállíthat mint egy egyszerű videó kódoló.

A videó transzkódolással foglalkozó kutatások fő célja tehát a bemeneti videóból kinyert kódolási paraméterek és statisztikák minél intelligensebb felhasználása a lehető legjobb képminőség eléréséhez és a számítási komplexitás minimalizálásához. Ezek gyakran egymásnak ellentmondó célok, így egyszerre nem teljesíthetők.

3.1 Dekóder – kódoló kaszkád

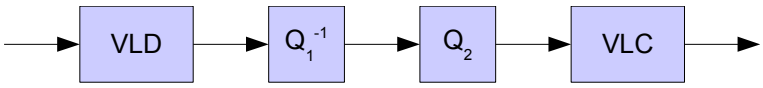
A videó transzkódolás kézenfekvő megvalósítása egy videó dekóder és egy videó kódoló közvetlen összekapcsolása. A dekóder kitömöríti a bemeneti tömörített videó bitfolyamot, a már tömörítetlen videó a kódoló bemenetére kerül, amely pedig újra tömöríti a kívánt új formátumra. Ez egy *rugalmas architektúra* mivel a dekóder és a kódoló teljesen függetlenül működhet egymástól, akár különböző bitsebességen, képsebességen, felbontással vagy akár más formátummal. Azonban a módszer számítási igény szempontjából nem egy hatékony megoldás. A közvetlen videó dekódolás-újrakódolás számítási komplexitásának csökkentése ezért az egyik jelentős területe a videó transzkódolással foglalkozó kutatási tevékenységeknek [81], [82], [83], [85]. A 3.2. ábrán a videó dekódoló-újrakódoló rendszer látható. A videó kódoló és dekóder részletes felépítését pedig a 2.3. ábra mutatja be.

3.2. Ábra: Videó dekóder-kódoló kaszkád

A transzkóder architektúrák vizsgálatánál a hangsúly a homogén típusú, bitsebesség csökkentő transzkóder megoldásokon lesz. Az ilyen transzkóder architektúráknak két fő típusa van, a *nyílthurkú transzkóderek* és a *zárthurkú transzkóderek*. A két architektúra közötti fő különbség az, hogy a nyílthurkú transzkóderekben nincs a „lélegzés” hibát kompenzáló visszacsatoló hurok, míg a zárthurkú transzkóderek, mint a nevük is utal rá, kiküszöbölik a „lélegzés” hibát egy visszacsatoló hálózat segítségével.

3.2 Nyílthurkú transzkóder

A nyílthurkú transzkódereknek szintén két fő típusa van, a *szelektív átvitelt* alkalmazó és az *újrakvantáló* típus. A 3.3 ábrán egy újrakvantáló típusú nyílthurkú transzkóder felépítése látható.



3.3. Ábra: Nyílthurkú, újrakvantáló transzkóder

A szelektív átvitelt alkalmazó módszer szerint a transzkóder, a bitsebesség csökkentéssel arányosan, lényegében eldobja a transzformációs együtthatók egy részét. A videó bitfolyamot egyébként nem módosítja. A transzkóder bemenetén a változó szóhosszúságú dekódolás helyett, csak az egyes kódszavak hosszának meghatározása történik, majd a következő lépésben az elérendő kimeneti bitsebességnek megfelelően a transzkóder eltávolít egyes együtthatókat. Az eltávolítandó együtthatók kijelölése vagy egy limit meghatározásával történik, amely frekvencia felett a komponensek eldobásra kerülnek, vagy egy vektor meghatározásával, amely közvetlenül kijelöli az eltávolítandó együtthatókat, így precízebben kontrollálva mely együtthatók maradnak a videóban [84], [85].

Az újrakvantáló típus az elérendő bitsebességtől függően változtatja a kimeneti kvantáló értékét, és ezzel újrakvantálja a transzformációs együtthatókat. A változó szóhosszúságú dekódolás után (VLD), az együtthatók inverz kvantálása történik (Q_1^{-1}), majd az elérendő

kimeneti bitsebesség függvényében meghatározott új, nagyobb kvantáló értékkel az újrakvantálása (Q_2), végül az új együtthatók változó szóhosszúságú kódolása (VLC) [85], [86]. Az újrakvantálást a (3.1) egyenlet szemlélteti, ahol d_n^1 jelöli a bemeneti, d_n^2 pedig a kimeneti transzformált képet, vagy predikciós hibaképet, $\mathcal{Q}$ és $\mathcal{Q}_1$ a kvantálás és inverzkavantálás operátorok:

$$d_n^2 = \mathcal{Q}_2\left(\mathcal{Q}_1^{-1}\left(d_n^1\right)\right). \tag{3.1}$$

A nyílthurkú transzkóder *számítási komplexitás szempontjából hatékony*, mivel közvetlenül a transzformációs együtthatókon, azaz *frekvencia tartományban*, végzi a számításokat, azonban ebből adódik a hátránya is, hogy *„lélegzés" hibát* ad a kimeneti videóhoz.

3.2.1 A „lélegzés" hiba

A „lélegzés" hiba kialakulásának a magyarázata a következő. Egy videó szekvenciában a legtöbb kép prediktíven kódolt kép, így csak egy korábbi referencia képhez képest a képtartalom megváltozását, azaz a predikciós hibaképet tartalmazza. Az eredeti képtartalom előállításához a dekóder tárolja a referencia képet, majd a prediktíven kódolt kép dekódolásakor a referencia képet kiegészíti a prediktíven kódolt képben lévő predikciós hibaképpel, így megkapja az aktuális kimeneti képet. Ahhoz hogy a dekóder kimenetén a megfelelő képet kapjuk, a dekóderben előállított és tárolt referencia képnek meg kell egyeznie a kódolóban előállított referencia képpel. A nyílthurkú transzkóder megváltoztatja a prediktíven kódolt képek adattartalmát, a predikciós hibaképeket, így a dekóderben előállított kép *nem pontosan egyezik meg* a kódlóban számítottal. Amennyiben a dekódolt kép referencia kép is egyben, akkor a dekóderben és a kódolóban előállított referencia képek nem fognak megegyezni. Ez a dekódolási hiba az egymás után előállított referencia képeken egyre halmozódik, ami a videó *képminőségét egyre jobban rontja, egészen a következő intra kódolt képig*. Az így keletkező hibát ezért „lélegzésnek" hívják, mivel a videó képminősége a halmozódó hiba miatt egyre romlik, majd a következő intra képnél helyreáll, majd a képminőség újra fokozatosan romlik a következő intra képig, és így tovább [87].

$$x_n = \mathcal{M}_n\left(x_{ref}\right) + e_n \tag{3.2}$$

A (3.2) egyenlet egy kép dekódolását írja le, ahol x_n a dekódolt kép, x_{ref} a referencia kép, e_n pedig a bitfolyamban átvitt predikciós hibakép, $\mathcal{M}_n()$ a mozgáskompenzáció operátor. Az egyenletben az egyszerűség kedvéért csak egy referenciakép szerepel, az általánosítása több referenciaképre triviális. A (3.2) egyenlet igaz és ugyanazt az eredményt adja a dekóderben dekódolt képekre és a kódolóban helyreállított később referenciaként használt képekre is, azaz a dekóderben és a kódolóban helyreállított képek megegyeznek.

$$\hat{x}_n = \mathcal{M}_n\left(\hat{x}_{ref}\right) + \hat{e}_n \tag{3.3}$$

A (3.3) egyenlet a nyílthurkú transzkódolás után írja le a kép helyreállítását a dekóderben, ahol $\hat{e}_n$ a transzkódolás során megváltoztatott predikciós hibakép – ez a változtatás a transzkódolás tulajdonképpeni célja –, $\hat{x}_n$ a helyreállított kép, $\hat{x}_{ref}$ pedig a referencia kép.

Mivel a nyílthurkú transzkóder nem állítja helyre a referencia képeket, a megváltozott predikciós hibakép, $\hat{e}_n$, továbbra is az eredeti referencia képhez képest értelmezhető. A dekóderben azonban az eredeti referenciakép, x_{ref}, nem áll rendelkezésre, csak a transzkódolás által szintén megváltoztatott referenciakép $\hat{x}_{ref}$. A helyreállított képen $\hat{x}_n$ ezért, összegződik

a transzkódolás által az aktuális képen okozott minőségromlás a referencia képen okozott hibával.

$$x_n^e = \hat{x}_n - x_n = \hat{e}_n - e_n + \mathcal{M}_n(\hat{x_{ref}}) - \mathcal{M}_n(x_{ref}) \tag{3.4}$$

A (3.4) egyenlet összefoglalja a nyílthurkú transzkódolás által az aktuális képen okozott megváltozást [82], [84]. A transzkódolt predikciós hibakép, e_n^e, megváltozását a (3.5) összefüggés írja le:

$$e_n^e = \hat{e}_n - e_n\,, \tag{3.5}$$

a referenciakép megváltozása, a_n, az aktuális kép szempontjából a (3.6) egyenlettel írható le. Amennyiben a referenciakép nem intra kódolt kép, a referenciaképre is igaz a (3.3) összefüggés, ami azt jelenti, hogy a_n egy halmozott hiba: az utolsó intra kódolt referencia képtől az aktuális referencia képig összegződött minőségromlás együttesen:

$$a_n = \mathcal{M}_n(\hat{x_{ref}}) - \mathcal{M}_n(x_{ref})\,. \tag{3.6}$$

Tehát röviden, az aktuális képen a minőségromlás, x_n^e, a (3.7) összefüggéssel írható le:

$$x_n^e = e_n^e + a_n\,. \tag{3.7}$$

A különböző képtípusok különböző módon érintettek a „lélegzés” hiba terjedésében. Az intra kódolt képeket, mivel a dekódolásukhoz referenciakép nem szükséges nem érinti a hiba, és mivel ezek a képek referenciaképek is, az intra kódolt képek megszüntetik a korábban felhalmozódott hibát. Azok a prediktíven kódolt képek amelyek a továbbiakban referenciaként nem lesznek felhasználva, érintettek a hiba által, képminőségük rosszabb, azonban a hiba tovább terjedésében nem vesznek részt. Leginkább azok a képek érintettek, amelyek egyrészt prediktíven kódoltak, tehát a dekódolásukhoz referenciakép szükséges, és a továbbiakban referencia képként lesznek felhasználva, tehát hozzájárulnak a hiba továbbterjedéséhez. MPEG-2 kódolás esetén az *I* képek az intra képek, *B* képek prediktíven kódoltak de referenciaként nem használhatók, a *P* képek prediktíven kódoltak és referencia képek is egyben.

A létrejövő „lélegzés” hiba mértéke függ a transzkóder architektúrától és a transzkódolás paramétereitől, továbbá mivel az intra képek megszüntetik a halmozódó hiba terjedését, lényegében helyreállítják a képminőséget. Az olyan alkalmazásokban tehát, ahol a két egymás követő intra kép között a prediktíven kódolt képek száma alacsony, és a „lélegzés” hiba által okozott videó minőség romlás elfogadható, a „lélegzés” hibát okozó transzkóderek használata előnyös lehet a viszonylagos egyszerűségük, kis számítási komplexitásuk és alacsonyabb memória igényük miatt.

3.3 Zárthurkú transzkóderek

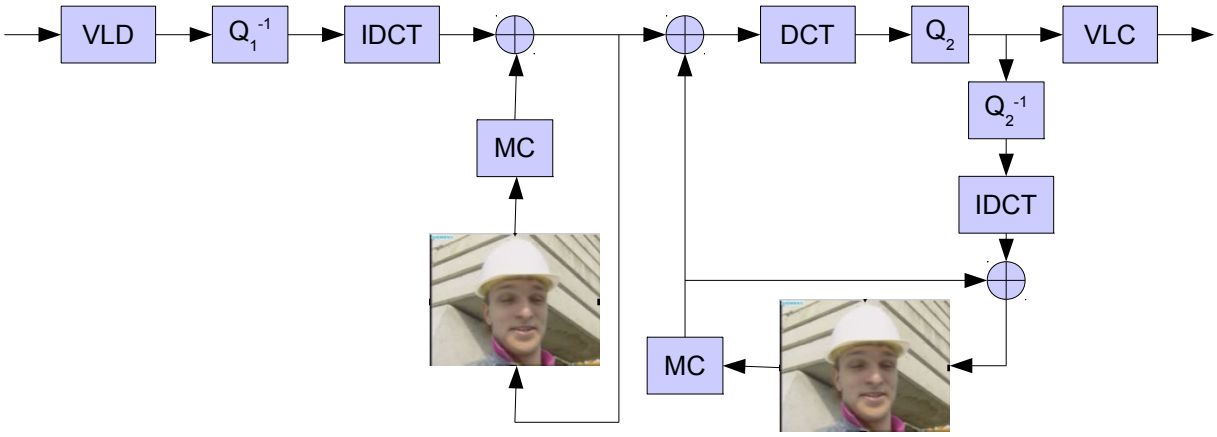
A zárthurkú transzkóderek, a nyílthurkú transzkóderekkel ellentétben, *helyreállítják a predikcióhoz használt referenciaképeket* egy visszacsatoló hurok segítségével, így a referencia képek hibája miatt nem keletkezik „lélegzés” hiba.

3.3.1 Kaszkád transzkóder térbeli tartományban

A 3.4. ábrán látható, kaszkád térbeli tartományú transzkóder, a zárthurkú transzkóderek csoportjába tartozik, mivel tartalmaz a „lélegzés” hibát kiküszöbölő visszacsatoló hurkot. Ez a transzkóder típus lényegében *egy dekóder és egy egyszerűsített kódoló* kaszkádba kapcsolása.

A transzkóder bemenetére érkező videó sok olyan információt tartalmaz amit érdemes kihasználni a kimeneti videó előállításához. Ezek az információk, pl.: a képtípus, mozgásvektorok, kvantáló értékek, bit allokáció, egyszerűsítik a transzkódoló felépítését és használatukkal jobb képminőséget lehet elérni. Lényegében a *dekódolt mozgás információk és kódolási mód információk* nagy része változtatás nélkül felhasználható a kódolóban, jelentős kimeneti képminőség romlás nélkül. A *bemeneti mozgásvektorok* felhasználásával – a mozgásbecslés, a videó kódolás legidőigényesebb művelete, amely a kódoló számítási komplexitásának 60-70%-át adja –, elkerülhető a transzkóderben [88]. A kaszkád térbeli tartományú transzkóder tulajdonképpen ezt használja ki, az egyszerűsített kódoló egységben nincs mozgásbecslés, a kimeneti videóba a bemeneti videóból származó mozgásvektorokat ülteti be [89].

A transzkóder működése a következő: a bemenetre érkező kódolt videót a dekóder egység térbeli tartományba dekódolja, először a változó szóhosszúságú dekóder (VLD) majd az inverz kvantálás (Q_1^{-1}) után, a kapott frekvencia tartománybeli információból inverz transzformációval (IDCT) a predikciós hibakép előállítása következik, majd a mozgáskompenzált összegzés után a dekódolt pixelek kerülnek át a kódoló oldalra.

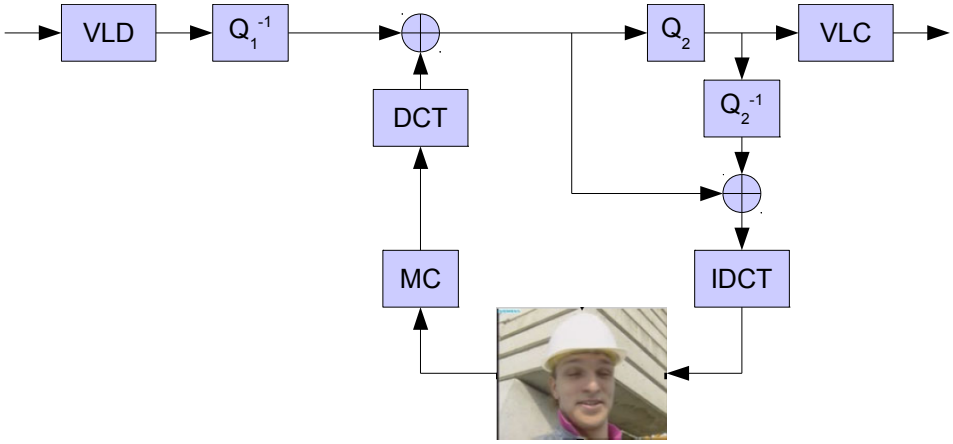


3.4. Ábra: Kaszkád térbeli tartományú transzkóder

A kódoló oldalon a mozgáskompenzált predikció – térbeli tartományban, a korábban kódolt, majd helyreállított referencia kép használatával – után, a predikciós hibakép transzformációja következik (DCT), a transzformációs együtthatók kvantálása az új kvantáló értékkel (Q_2), majd végül a változó szóhosszúságú kódolás (VLC). Amennyiben az aktuális kép referenciakép is egyben, a másik útvonalon dekódolás után a képtárolóba kerül, így a dekódolt referenciakép megegyezik a megjelenítő dekóderében helyreállított referenciaképpel, tehát nem keletkezik „lélegzés” hiba. Intra kódolt kép esetén a predikciós lépések kimaradnak, így a változó szóhosszúságú dekódolás után az inverz kvantálás, inverz transzformáció, majd transzformáció, újrakvantálás és változó szóhosszúságú kódolás következik, a másik útvonalon a dekódolás után a kép a képtárolóba kerül [82].

3.3.2 Egyszerűsített térbeli tartományú transzkóder

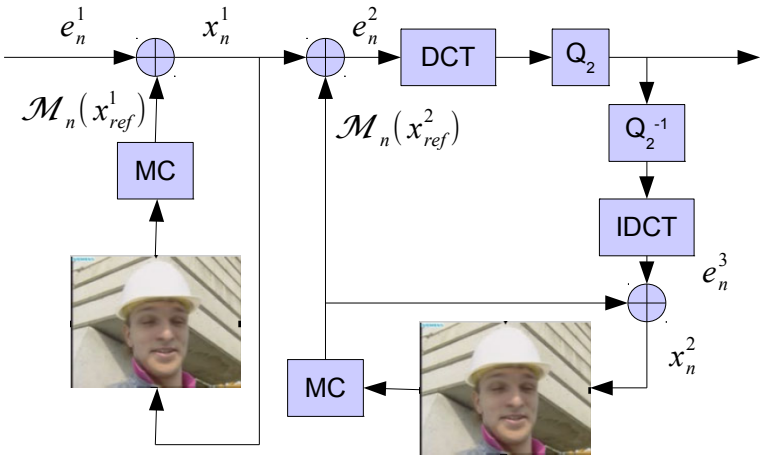
A 3.5. ábrán az egyszerűsített térbeli tartományú transzkóder látható, amely szintén a zárthurkú transzkóder architektúrák közé tartozik.



3.5. Ábra: Egyszerűsített térbeli tartományú transzkóder

A fő különbség az egyszerűsített és a kaszkád térbeli tartományú transzkóder között, hogy míg az előbbiben csak *egy referenciakép helyreállító hurok* van, egy DCT és egy IDCT lépéssel az utóbbiban kettő, egy transzformáció (DCT) és két inverz transzformáció (IDCT) művelettel. Ez az egyszerűsítés azon a feltételezésen alapszik, hogy a transzformáció, az *inverz transzformáció és a mozgáskompenzáció lineáris műveletek* [81], [83], [90], [91], [92].

Az egyszerűsítés a következőképpen magyarázható [89]. A 3.4. ábrán látható, kaszkád térbeli tartományú transzkóder, dekóder oldalának kimenetén, amely kódoló oldal bemenete is egyben, a dekódolt kép: x_n^1. A kódoló oldalon predikció után az új predikciós hibakép, e_n^2, jön létre amely transzformáció (DCT), a kvantálás (Q_2), majd a változó szóhosszúságú kódolás (VLC) után a kimenetre kerül.



3.6. Ábra: A két predikciós hurok

A másik útvonalon a kvantált együtthatók a visszacsatoló hurokba kerülnek, ahol az inverz kvantálás (Q_2^{-1}) és az inverz transzformáció (IDCT) után egy újabb dekódolt predikciós

hibakép, e_n^3, és predikció után egy dekódolt kép jön létre, x_n^2, amely a referenciakép tárolóba kerül. A kaszkád térbeli tartományú transzkóder visszacsatoló hálózata kiemelve a 3.6. ábrán látható.

Az újrakvantálás megváltoztatja az eredeti predikciós hibaképet, e_n^2, így új predikciós hibakép jön létre, e_n^3, a két predikciós hibakép különbsége egy kvantálási hibaképként is felfogható, ami felírható a következőképp:

$$e_n^3 = e_n^2 + q_n^2 . \tag{3.8}$$

A (3.8) egyenletben q_n^2 jelöli a transzkóder kódolójában keletkező kvantálási hibát vagy hibaképet. A dekódolt képeket a (3.2) összefüggés alapján felírhatjuk a következőképpen:

$$x_n^1 = \mathcal{M}_n(x_{ref}^1) + e_n^1 , \tag{3.9}$$

$$x_n^2 = \mathcal{M}_n(x_{ref}^2) + e_n^3 , \tag{3.10}$$

ahol x_{ref}^1 a transzkóder dekóderében tárolt referenciakép, x_{ref}^2 pedig a transzkóder kódoló részében helyreállított referenciakép. A kódoló oldal bemenetén számított predikciós hibakép:

$$e_n^2 = x_n^1 - \mathcal{M}_n(x_{ref}^2) \tag{3.11}$$

A (3.10) egyenletbe behelyettesítve a (3.8) egyenletet:

$$x_n^2 = \mathcal{M}_n(x_{ref}^2) + e_n^2 + q_n^2 , \tag{3.12}$$

majd (3.12) összefüggésbe a (3.11) egyenletet:

$$x_n^2 = x_n^1 + q_n^2 , \tag{3.13}$$

azt kapjuk, hogy a transzkóder kódolójában helyreállított kép, x_n^2, az a bemeneti kép, x_n^1, és az újrakvantálási hibakép, q_n^2, összege. A transzkóder kódolójában helyreállított kép, x_n^2, megegyezik azzal a képpel, ami egy megjelenítőben dekódolható.

Amennyiben feltételezzük, hogy a mozgáskompenzáció lineáris művelet [89], és felhasználva a (3.13) összefüggést, a transzkóder kódoló oldalán történő predikció két részre osztható:

$$\mathcal{M}_n(x_{ref}^2) = \mathcal{M}_n(x_{ref}^1 + q_{ref}^2) , \tag{3.14}$$

$$\mathcal{M}_n(x_{ref}^1 + q_{ref}^2) = \mathcal{M}_n(x_{ref}^1) + \mathcal{M}_n(q_{ref}^2) , \tag{3.15}$$

amelyből az első rész, $\mathcal{M}_n(x_{ref}^1)$, megegyezik a dekóder oldalon a predikcióhoz használt mozgáskompenzációval, tehát a (3.15) egyenletnek ez az összetevője a dekóder kimenetén hozzáadódik, majd rögtön a kódoló bemenetén pedig kivonódik a jelből. A fentiek szerint tehát a predikcióhoz csak a mozgáskompenzált kvantálási hibára van szükség, ami a következőkből is látszik.

A (3.11) egyenletbe behelyettesítve a (3.14), majd a (3.15) összefüggést, azt kapjuk, hogy:

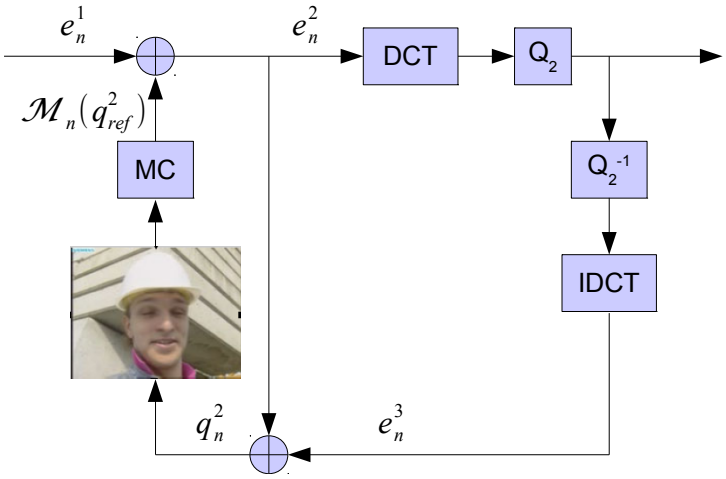
$$e_n^2 = x_n^1 - \mathcal{M}_n(x_{ref}^1 + q_{ref}^2)\,, \tag{3.16}$$

$$e_n^2 = x_n^1 - \mathcal{M}_n(x_{ref}^1) - \mathcal{M}_n(q_{ref}^2)\,. \tag{3.17}$$

A (3.9) egyenletből kifejezve a dekódolt képet, x_n^1, majd behelyettesítve a (3.17) összefüggésbe:

$$e_n^2 = e_n^1 - \mathcal{M}_n(q_{ref}^2) \tag{3.18}$$

A kimeneti videó előállításához valójában csak az előállított predikciós hibaképre, e_n^2, van szükség. A (3.18) egyenlet alapján látható, hogy ez a predikciós hibakép, e_n^2, közvetlenül előállítható a mozgáskompenzált kvantálási hiba, $\mathcal{M}_n(q_{ref}^2)$, segítségével, tehát csak a kvantálási hibaképet, q_{ref}^2, kell tárolni, így a dekóder oldalon lévő képtárolóra nincs szükség. A (3.18) egyenletből továbbá az is következik, hogy nem csak a dekóder oldali képtárolóra, hanem a dekóder oldali visszacsatoló hurokra sincs szükség, tehát a két visszacsatoló hurok egy visszacsatolásra redukálható. Az így leegyszerűsített visszacsatoló hurok a 3.7. ábrán látható.



3.7. Ábra: Az egyszerűsített visszacsatoló hurok

További egyszerűsítés érhető el a transzformáció és az inverz transzformáció linearitásának kihasználásával [89]. A (3.19) ekvivalencia alapján a visszacsatoló hurokból kivehető egy transzformáció és áthelyezhető egy inverz transzformációs lépés:

$$e_n^2 - \mathcal{D}^{-1}(\mathcal{Q}_2^{-1}(\mathcal{Q}_2(\mathcal{D}(e_n^2)))) \equiv \mathcal{D}^{-1}(d_n^2 - \mathcal{Q}_2^{-1}(\mathcal{Q}_2(d_n^2)))\,, \tag{3.19}$$

amennyiben a (3.20) ekvivalencia alapján a bemeneten a predikció előtti inverz transzformáció kikerül, a referencia kép felől pedig egy transzformációs lépés bekerül a rendszerbe:

$$e_n^1 - \mathcal{M}_n(q_{ref}^2) \equiv d_n^1 - \mathcal{D}(\mathcal{M}_n(q_{ref}^2))\,, \tag{3.20}$$

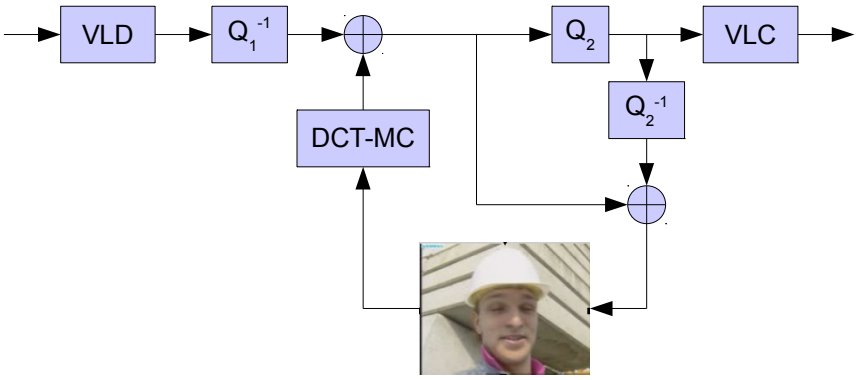
ahol $\mathcal{D}()$ a transzformációs, $\mathcal{D}^{-1}()$ az inverz transzformációs, $\mathcal{Q}()$ a kvantálás és $\mathcal{Q}^{-1}()$ az inverz kvantálás operátor, valamint alkalmazva a (3.21) egyenlőséget, melyben d_n a transzformált predikciós hibaképet jelenti:

$$d_n = \mathcal{D}(e_n) . \tag{3.21}$$

A transzformációs lépések fent bemutatott mozgatásával elérhető, hogy az egy transzformáció és két inverz transzformáció, egy transzformációra és egy inverz transzformációra redukálódjon. Így végül megkapva a 3.5. ábrán látható transzkódoló felépítést, amely komplexitása egy dekódernél csak egy extra transzformációval, egy kvantálás – inverz kvantálással és egy változó szóhosszúságú kódolással nagyobb. Egy nyílthurkú transzkóderhez képest pedig egy inverz kvantálással, egy transzformáció – inverz transzformációval és egy predikcióval nagyobb.

3.3.3 Frekvencia tartományú transzkóder

A korábbi fejezetekben leírt zárthurkú transzkóder architektúrák a makroblokkok helyreállítását frekvencia tartományban végzik, de a referenciaképek tárolása és a mozgáskompenzáció térbeli tartományban történik. A pixelek helyreállításához ezért egy inverz transzformációra és egy transzformációra van szükség. Ez elkerülhető *frekvencia tartományban működő mozgáskompenzációs algoritmusok* használatával, melyek segítségével helyreállíthatók a referencia képek térbeli tartományba való dekódolás nélkül [93], [94], [96], [101]. A 3.8. ábrán egy frekvencia tartományú transzkóder látható.



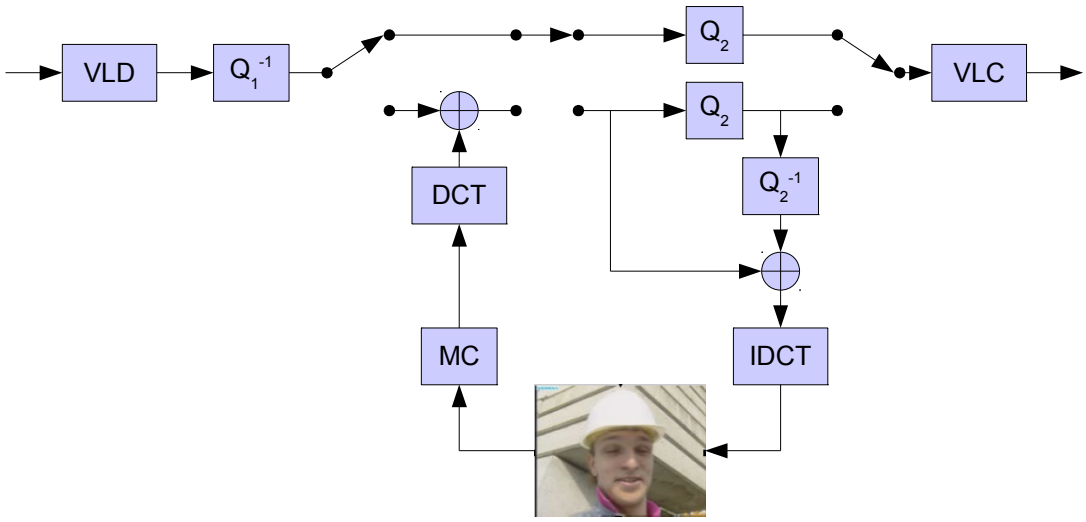
3.8. Ábra: Frekvencia tartományú transzkóder

A frekvencia tartományú mozgáskompenzáció használatával elérhető a térbeli tartományú transzkóderek képminősége, azonban *számítási komplexitást tekintve nincs jelentős javulás* a korábbi transzkóderekhez képest. A kutatások ezért a frekvencia tartománybeli mozgáskompenzáció egyszerűsítésére koncentrálnak. Az egyszerűsítés azonban a képminőség bizonyos szintű romlását okozza, és a hibás referencia képek miatt megjelenik a „lélegzés” hiba. Az egyszerűsítés történhet a lebegőpontos mátrix műveletek fixpontos közelítésével [94], [95], továbbá kihasználható, hogy a transzformációs együtthatók főleg a kisfrekvenciás területeken koncentrálódnak, így a mozgáskompenzáció közelíthető csak néhány kisfrekvenciás komponensen történő számítással [96], [97], [98], [99], [100].

3.3.4 Hibrid tartományú transzkóder

A különböző transzkóder architektúrák megfelelő *egyensúlyt próbálnak találni a számítási komplexitás mértéke és a kimeneti videó képminősége között.* A számítási komplexitás a mozgáskompenzációs, transzformációs és inverz transzformációs lépések kihagyásával csökkenthető, amennyiben lehetséges az adott videó minőségi követelmények mellett. A hibrid transzkódolás ezért a fenti *transzkóder architektúrák működését ötvözi*. Egy hibrid transzkóder felépítés látható a 3.9. ábrán.

Hibrid transzkódolási megoldás például, ha csak azokon a képeken van mozgáskompenzáció, melyek prediktíven kódoltak és referencia képek is lesznek később. Intra kódolt képek esetén egyébként sincs mozgáskompenzáció, ezért az inverz transzformáció, transzformáció lépésre sincs szükség, pontosabban, mivel az intra kódolt képek referencia képek is egyben, a helyreállításukhoz továbbra is szükség van egy inverz transzformációra, így csak egy transzformációs lépés hagyható el. Azokon a prediktíven kódolt képeken, melyek később nem lesznek referencia képek, a megjelenő „lélegzés" hiba nem terjed tovább a videó szekvenciában, ezért a mozgáskompenzáció, az inverz transzformáció és a transzformáció lépések elhagyhatók, valamint a kép helyreállítására sincs szükség, mivel a kép nem referenciakép. Az ilyen képek transzkódolása történhet teljesen a transzformációs tartományban, azaz lényegében nyílthurkú módszerrel.



3.9. Ábra: Hibrid tartományú transzkóder

A gyors mozgásokat és gyakori jelenetváltásokat tartalmazó videókban a prediktíven kódolt képek is sok intra kódolt blokkot tartalmazhatnak. A prediktíven kódolt referencia képek intra blokkjain elhagyható a mozgáskompenzáció, az inverz transzformáció és a transzformáció. Tehát az intra kódolt és a nem referencia prediktíven kódolt képek összes blokkja, a prediktíven kódolt referenciaképeknek pedig az intra blokkjai frekvencia tartományban transzkódolhatók, a térbeli tartományú mozgáskompenzációra csak a prediktíven kódolt referencia képek nem intra blokkjain van szükség [82], [102].

A fentiek alapján egy hibrid transzkóder *számítási komplexitás és képminőség tekintetében is a zárthurkú és a nyílthurkú transzkóderek között* helyezkedik el. A zárthurkú transzkódereknél kisebb számítási komplexitás, nyílthurkú transzkódereknél jobb képminőség érhető el vele.

4 Párhuzamos feldolgozás

Az elmúlt 50 évben Moore törvénye [14] pontosan megjósolta, hogy egy integrált áramkörben a tranzisztorok száma másfél-, kétévente megduplázódik. Hogy a megnövekedett tranzisztorszámot kihasználva a rendszerek teljesítményre is növekedjen, az IC tervezők *megnövelték az órajel frekvenciákat,* amelyhez hosszabb utasítás pipeline szükséges, *növelték az utasítás szintű párhuzamosságot,* amely egyidejűleg futó programszálakat és elágazás előrejelzést (branch prediction) igényel, *megnövelték a memória teljesítményét* nagyobb gyorsítótárak (cache) segítségével, és *megnövelték az IC-k energia felhasználását,* amely aktív energia gazdálkodást (power management) igényel.

Mind a négy terület elérte fejlődésének egy határát amely gátolja a további növekedést. A feldolgozási sebesség növekedése csökken az órajelfrekvenciák lassuló növekedése miatt, mert a tranzisztorok sebessége nem nő a méretük csökkenésével arányosan. Az utasítás szintű párhuzamosság kihasználását korlátozza a programok alapvetően soros jellege a belőlük hiányzó párhuzamosság. A memória teljesítmény növekedését korlátozza a processzorok és a memóriák közötti növekvő sebesség különbség. Az energiafogyasztás arányos az órajel frekvenciával, tehát egy bizonyos pont felett különleges eszközök szükségesek az IC-k hűtéséhez [15].

$$P_e = \frac{1}{t_e} = \frac{n_{IPC} \cdot f_{clk}}{n_{IC}} \tag{4.1}$$

Egy processzor számítási teljesítménye P_e lényegében egy feladat megoldására fordított idő t_e reciproka. A számításhoz szükséges idő pedig függ a processzor órajel frekvenciájától f_{clk} a feladat végrehajtásához szükséges utasítások számától n_{IC} és a feladat végrehajtása során kihasználható utasításszintű párhuzamosság mértékétől n_{IPC} [62].

Az új irány, hogy a növekvő tranzisztorszámot egy IC-n belül több processzormag kialakítására használják fel, ez a *chip szintű multiprocesszor.* Az egyes magok bonyolultsága nem nagyobb – esetleg *egyszerűbbek* is mint az egymagos változat –, ugyanakkora vagy akár *alacsonyabb órajellel működnek*, így az egyes magok energiafogyasztása a fejlettebb gyártástechnológiával csökken, miközben az *együttes számítási teljesítmény növekszik*. Még ha egy feladat valósidejű végrehajtása elérhető néhány gyorsabb processzormag használatával, több de alacsonyabb órajelen működő processzormag felhasználásával csökkenthető az energiafogyasztás. Várhatóan a processzormagok száma egy CMP-ben háromévenként fog megduplázódni [50].

A több processzormaggal elérhető számítási sebesség növekedés, *s*, azonban erősen függ a megoldandó problémától és a megoldás módjától. A gyorsulás felső korlátja Amdahl törvénye [63] néven ismert:

$$s = \frac{1}{r_s + \frac{r_p}{n}}, \quad \text{ahol:} \quad r_s + r_p = 1 \tag{4.2}$$

ahol r_s a probléma sorosan végrehajtható része, r_p pedig a párhuzamosítható rész, n a számítást végző magok száma. Az n processzormag csak a feladat párhuzamosítható részét gyorsítja, a csak szekvenciálisan végrehajtható részét nem. Ezt a felső korlátot azonban többen vitatják vagy pontosítják: [64], [65], [66].

4.1 Processzor architektúrák

Modern processzor architektúrák közötti legfontosabb különbség a párhuzamos végrehajtás módja. Egyprocesszoros esetben ez az utasítás szintű és utasításon belüli párhuzamosságot jelent. Az utasítás szintű párhuzamosságot több végrehajtó egység és pipeline elrendezés valósítja meg, az utasításon belüli egyidejű végrehajtást vektor utasítások. Videó kódolás szempontjából a főbb architektúrák a következők.

4.1.1 Szuperskalár processzor

A szuperskalár processzor egy olyan *skalár processzor* amely képes *minden órajel periódusban egynél több utasítást végrehajtani*, több végrehajtó egység segítségével. Ezt olyan módon, hogy fennmarad a skalár processzorokra jellemző szekvenciális végrehajtási modell és az ehhez tartozó pontos processzor állapot. A szekvenciális program utasításokat a processzor ütemezője rendeli az egyes egységekhez [56], [57]. Az általános célú processzorok ma kevés kivétellel szuperskalár felépítésűek.

4.1.2 Nagyon hosszú utasítás szavas processzor

VLIW architektúrájú processzorok szintén több utasítást hajtanak végre egy órajel ciklus alatt több végrehajtó egység segítségével. Azonban míg a szuperskalár rendszerű processzorokban ezek a végrehajtó egységek rejtve maradnak, VLIW processzorok programjában az egyes *utasítások explicit végrehajtó egységekhez vannak rendelve*. Minden végrehajtó egységhez egy-egy utasítás alkotja a teljes utasítás szót, ebből adódik az architektúra elnevezése [16], [58]. VLIW processzorokban, az utasítás szintű párhuzamosságot nem a processzor hardver, hanem a fordító program határozza meg. Ez egyszerűbb hardvert így kisebb fogyasztást jelent, valamint a fordítóprogram nagyobb kódrészletet tekinthet át, de a futásidőben felmerülő lehetőségeket nem tudja kihasználni. Jelfeldolgozás szempontjából fontos tulajdonsága a VLIW processzoroknak, ellentétben a szuperskalár rendszerűekkel, a *determinisztikus utasítás végrehajtási idő* [16], [17]. A fenti tulajdonságoknak köszönhetően a legtöbb modern DSP VLIW architektúrájú.

4.1.3 Digitális jelfeldolgozó processzorok

A DSP tulajdonképpen nem külön processzor architektúra. Az általános célú processzoroktól abban különböznek, hogy a jelfeldolgozási számításokhoz szükséges műveleteket hatékonyabban hajtják végre, az utasításkészletük az ilyen számításokhoz lett kialakítva, a vezérlés típusú programokat, ahol gyakori a döntéshozás és sok az elágazás kevésbé. A legtöbb modern *nagy számítási teljesítményű DSP VLIW felépítésű*.

Jelfeldolgozási feladatokban a számítási idő nagy része – nagyméretű adattömbökön, például mátrixműveletek – egymásba ágyazott ciklusok végrehajtását jelenti, ahol a belső ciklusok csak számítási műveleteket tartalmaznak. A ciklusok azonban nem egyenletesek és homogének eléggé ahhoz hogy a végrehajtásuk masszívan párhuzamos processzoron hatékony legyen, VLIW architektúrán azonban a *ciklus iterációi néhány utasítás eltolással átlapolva párhuzamosan végrehajthatók* (szoftver pipeline) [18], [62].

Videó feldolgozási feladatok közül a mozgásbecslés, transzformáció, inverz transzformáció ilyen típusú művelet. Modern videó kódolásokban azonban a számításigényes feladatok a vezérlés típusú feladatokkal vegyesen vannak jelen, ezért a modern DSP-k fel vannak készítve ilyen típusú kódok végrehajtására is [16], [61].

4.1.4 Média processzorok

Videó és audió hatékony feldolgozására kifejlesztett processzorok, melyek valójában szintén nem tekinthetők külön processzor architektúrának, általában modern, nagy számítási teljesítményű *DSP architektúrákon alapulnak*. Az utasítás szintű párhuzamosságot ezekben a processzorokban is a VLIW architektúra teszi lehetővé, de kiegészül az utasításon belüli párhuzamosságot kihasználó vektor utasításokkal és a vezérlés típusú feladatok gyorsítására feltételes utasításokkal. A média processzorok továbbá a bitfolyam alapú média adatok hatékony kezelését lehetővé tevő memória alrendszerrel rendelkeznek.

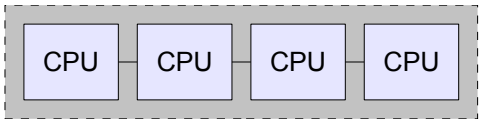
A központi processzorhoz gyakran különféle, a média feldolgozás egyes nagy számításigényű lépéseit *gyorsító perifériák* csatlakoznak, a processzor célfeladatának megfelelően. Van kis energia igényű általános célú processzorral kiegészített változat is, de az már felépítését tekintve a heterogén multiprocesszor kategóriába tartozik inkább [22], [23], [29].

4.2 Multiprocesszor rendszerek

Többmagos processzorok valójában olyan multiprocesszor rendszerek, ahol a különálló processzorok egy IC-n belül lettek kialakítva. Multiprocesszor rendszerekben a feladat szintű párhuzamosság valósítható meg, az egyes processzorokon egyszerre futhatnak az egyes feladatok. Videó kódolás szempontjából fontosabb chip szintű multiprocesszor rendszerek a következők.

4.2.1 Homogén multiprocesszor

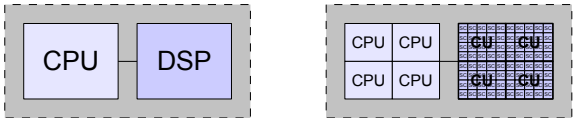
Homogén multiprocesszoros rendszerben két vagy *több egyforma processzor* működik egyenrangúan együtt általában egy operációs felügyelete alatt. Az egyforma processzorok utasításkészlete természetesen megegyezik, tehát a programok, feladatok bármelyik processzormagon futtathatók [59]. Az első általános célú többmagos processzorok homogén kialakításúak voltak.



4.1. Ábra: Homogén multiprocesszor

4.2.2 Heterogén multiprocesszor

Heterogén multiprocesszoros rendszerben *több különböző processzor* működik együtt.



4.2. Ábra: Heterogén multiprocesszor

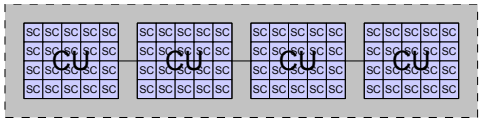
Általában egy vagy több általános célú processzor mellett szintén egy vagy több speciális processzor található. Az általános célú processzorok tipikusan vezérlő jellegű nem valósidejű feladatokat hajtanak végre, futtatják az operációs rendszert. A speciális processzormagok

amelyek lehetnek például valamilyen DSP, GPU vagy mindkettő, általában a speciális nagy számításigényű, valósidejű és masszívan párhuzamos feladatokat végzik el [23].

Heterogén multiprocesszoros rendszerek programozása bonyolultabb mint a homogén rendszereké, mert a különböző architektúrájú processzorok utasításkészlete nem egyezik meg. A különböző processzormagokhoz tehát külön futtatható kód szükséges, amihez külön fejlesztési környezet. A feladatokhoz tartozó kódot azokra a processzorokra külön-külön kell lefordítani, amelyeken futtatva lesznek [24]. Tipikus heterogén multiprocesszorok a modern többmagos általános célú GPU-val egybeépített processzorok [25].

4.2.3 Masszívan párhuzamos multiprocesszor

Egy masszívan párhuzamos multiprocesszor *nagyon sok viszonylag egyszerű processzorból* áll, melyek számító egységekbe (compute unit) vannak szervezve. Egy multiprocesszor több számító egységből áll, mindegyikben ugyanannyi processzorral. A program végrehajtás számító egységenként történik. A számító egységek minden processzora ugyanazokat a program utasításokat hajtja végre egyszerre, de más adatokon. Az egyes processzorokhoz külön regiszterek és állapot tartozik. Az ilyen felépítésű multiprocesszorokat *adatfolyam (stream) processzornak* is nevezik [60].



4.3. Ábra: Adatfolyam processzor

Speciálisan nagyon párhuzamos feladatok megoldására alkalmasak, – például képfeldolgozás –, ezért programozásuk viszonylag bonyolult, a párhuzamos végrehajtást jól leíró programnyelvre van szükség [26], [27]. A modern videókártyákon található GPU-k számítási célra is használható masszívan párhuzamos multiprocesszorok.

4.3 Memória architektúra

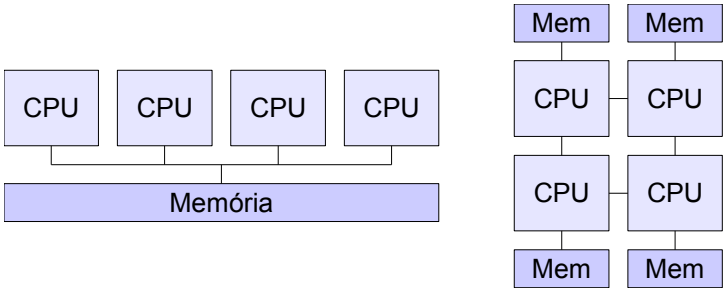
Több processzort tartalmazó rendszerekben az *adatokat valamilyen módon meg kell osztani* a feldolgozó egységek között. A rendszerben alkalmazott memória elrendezés meghatározza a processzorok közötti adatmegosztás lehetőségeit. Három alapvető memória architektúra terjedt el párhuzamos elosztott rendszerekben: a közös (shared) memória, elosztott (distributed) memória és hibrid memória.

4.3.1 Közös memória

A közös memóriát (shared memory) alkalmazó rendszerekben a processzoroknak közvetlen hozzáférése van egy bizonyos méretű memóriatömbhöz. A processzorokban általában hardver megoldás biztosítja a cache koherenciát, azt hogy minden processzor cache memóriája szinkronban maradjon. Az *egységes memória hozzáférésű memóriás* rendszerekben egy memória tömb létezik. Minden processzor ugyanakkora késleltetéssel érheti el ezt a memóriát. A modern többmagos személyi számítógépek majdnem mindig UMA rendszerűek.

A *nem egységes hozzáférésű memóriás* rendszerekben több memóriatömb létezik, ezek a memóriaegységek általában az egyes processzorokhoz tartoznak, a hozzáférést ezek a processzorok kezelik. Ez a megoldás az egyes processzorok és memóriák közötti késleltetésben és hozzáférési időben egyenlőtlen rendszer, de nagyobb skálázhatóságot tesz

lehetővé. Létezik a NUMA rendszereknek egy cache koherens változata is ahol, az UMA rendszerekhez hasonlóan, hardver megoldás biztosítja, hogy minden processzor cache memóriája szinkronban maradjon. Néhány komolyabb munkaállomás és a szerverek többsége ccNUMA felépítésű.

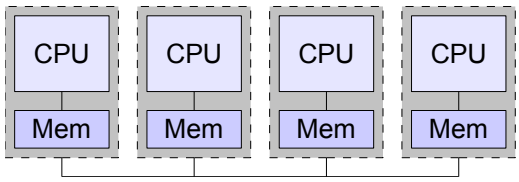


4.4. Ábra: UMA és NUMA rendszer

A közös memória rendszerek fő előnye, hogy a *programok számára a memória egységes* tömbként kezelhető, leegyszerűsítve a programozást [54].

4.3.2 Elosztott memóriatömb

Elosztott memóriás rendszerekben minden processzornak saját memória területe van, amely *nem érhető el globálisan a többi processzor számára*. A megosztandó adatok valamilyen kapcsolaton keresztül jutnak el egyik processzortól a másikra, amelyen a programoknak konkrétan át kell küldeni az adatokat.

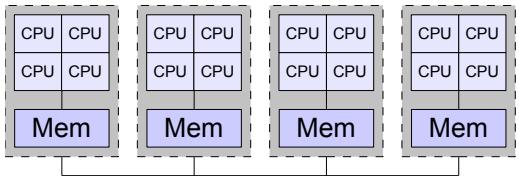


4.5. Ábra: Elosztott memóriás rendszer

Ez megnehezíti a programozást a közös memória rendszerekhez képest, de jobb skálázhatóságot biztosít. Régebbi szuperszámítógépek elosztott memóriás rendszerek voltak, de mára leváltották a hibrid memóriás gépek.

4.3.3 Hibrid memória

Hibrid memóriás rendszerek *elosztott módon összekapcsolt közös memóriás rendszerekből* állnak. Az ilyen rendszerek általában különálló UMA vagy NUMA számítógépek, gyors hálózattal összekapcsolva. Majdnem minden modern szuperszámítógép hibrid felépítésű.



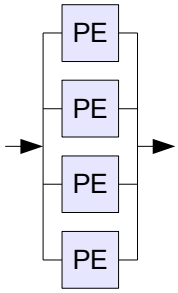
4.6. Ábra: Hibrid rendszer

4.4 Programozási modell

A párhuzamos programozás alap koncepciója a feladat (task). A feladat egy program részlet amely a többi feladattól függetlenül hajtható végre. Minden feladathoz hozzá van rendelve egy processzor állapot, regiszterek, verem (stack) memória terület, és helyi változók melyek függetlenek a többi feladattól. Operációs rendszertől függően megkülönböztethetünk feladat típusokat: feladat, folyamat (process), szál (thread). Ezek főleg a memória megosztás módjában és az elérésében különböznek, de lehet különbség az ütemezésben is.

4.4.1 Adat felosztás

Adat alapú particionálás esetén a *feldolgozó egységek ugyanazt a számítást hajtják végre* de a feldolgozandó adat különböző részein. Leggyakrabban elosztott memóriás rendszerekben alkalmazzák, ahol a programszálak az egyes processzorokon futnak, és mindegyik szál egyformán hozzáfér az adatokhoz. Korlátja a memória sávszélesség és az adatok közötti függőség.

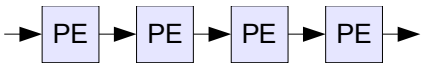


4.7. Ábra: Adat felosztás

A videó kódolásban alkalmazott időbeni és térbeli felosztás programozási modell szerint lényegében adat alapú particionálásnak tekinthető.

4.4.2 Funkcionális vagy pipeline particionálás

A feldolgozó egységek funkcionális felosztás esetén a *teljes számítási feladat egyes lépéseit* hajtják végre párhuzamosan. Ha ez egyik számítási fázis elkészült, az adatok átkerülnek a következő végrehajtó egységhez, amely megkezdi a következő számítási fázist, miközben az előző egység újabb adatokkal ismétli a hozzá tartozó számítási fázist, amely tulajdonképpen a pipeline módszer.

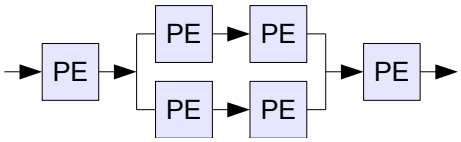


4.8. Ábra: Pipeline

Videó kódolásban a funkcionális particionálás használata a finom térbeli felbontás esetén fordul elő. Egy megoldásban az egymás utáni kódolási lépéseket egyenlő végrehajtási idejű feladatokra bontják fel, majd ezeket pipeline módszerrel hajtják végre [51]. Sokszor a kódolást végző processzor mellett speciális hardver egységek gyorsítják a szoftveres kódolást vagy dekódolást [41], [42], [43].

4.4.3 Adatfolyam

Az adatfolyam particionálás tulajdonképpen az *adat szerinti felosztás és a pipeline particionálás kombinációja*. A fentiekhez hasonlóan a végrehajtás és a vezérlés is elosztott, minden végrehajtó egység az adott számítási feladatát egy adatblokkon elvégzi, majd továbbítja a következő egységhez. A bonyolultabb számítási feladatok több feldolgozó egységhez is hozzá lehetnek rendelve, ezeket párhuzamosan hajtják végre, hogy a pipeline egyenletes kihasználtságú legyen [15].

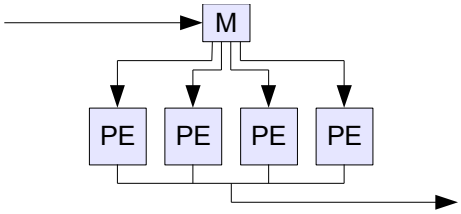


4.9. Ábra: Adatfolyam felosztás

Videó kódoló de dekódoló algoritmusok is, mivel nagy mennyiségű adat nagy számításigényű sokszor valósidejű feldolgozásáról van szó, sokszor minél több párhuzamosítási lehetőséget használnak ki, ezért gyakran a teljes rendszer adatfolyam felépítésű.

4.4.4 Master-slave

A master-slave modell *központi vezérlést és elosztott végrehajtást* jelent. A mester feladata a feladatok, és a hozzátartozó adatok hozzárendelése az egyes végrehajtó egységekhez, melyek a tényleges számítási feladatot elvégzik.



4.10. Ábra: Master-Slave rendszer

Az elosztott rendszereken megvalósított videó kódolók gyakran osztják szét a számítási feladatokat master-slave módszerrel [55].

4.5 Kommunikáció

Multiprocesszoros rendszerekben szükség van az adatok megosztására és a feldolgozó egységek között a vezérlés szinkronizálására. A processzorok közötti kommunikáció két fő eseményből áll: az *adat mozgatásból és a jelzésből*. A processzorok közötti hálózattól és a memória architektúrától függ, hogy a különböző adat mozgatási és jelzési lehetőségek közül mit érdemes használni.

4.5.1 Közös memória

Közös memóriás kommunikáció esetén az üzenethez szükséges memóriához a küldő és a fogadó processzornak is van – nem szükségszerűen egyenrangú – hozzáférése. A küldő fél

elküldi az *üzenetet a közös memóriaterületre* és erről értesíti a fogadót, aki az üzenet kimásolja a közös memóriaterületről és értesíti a küldőt, hogy a memória szabad.

4.5.2 Üzenet küldés

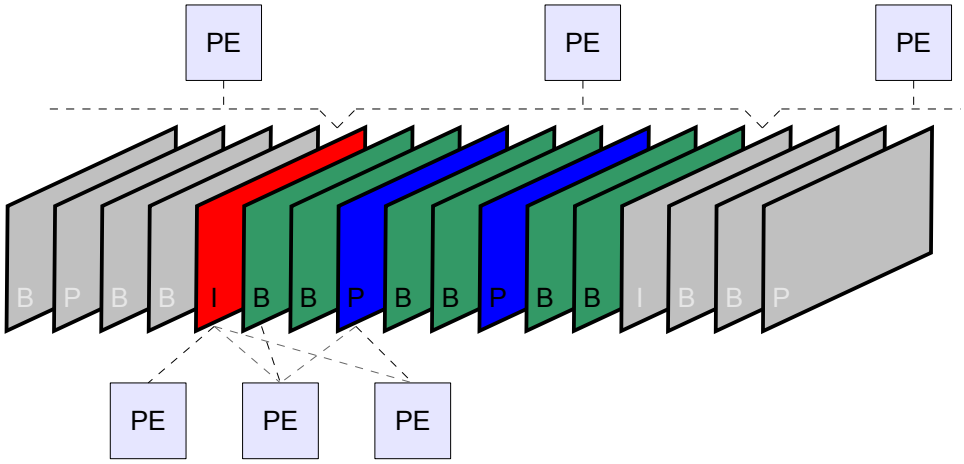
Üzenet küldést olyan rendszerekben használják leginkább, ahol a különböző processzorok között valamilyen kommunikációs hálózat tartja a kapcsolatot és *nincs közösen elérhető memória* vagy túl költséges az elérése [28]. A küldő fél az üzenetet a hálózati eszközhöz továbbítja, majd a fogadó felet a hálózat értesíti az üzenet megérkezéséről.

4.5.3 Hibrid

A hibrid megoldás *logikájában az üzenet küldésre hasonlít, de a megvalósítás közös memóriával történik*. A küldő és a fogadó is közös memóriát használ, de az üzenet memória területe nem ideiglenes, így az adatok helyben maradnak, nincs adatátvitel az üzenet továbbítása közben, csak a memória terület tulajdonga cserél gazdát. A küldő egy mutatót (pointer) küld át, a fogadó pedig az eredeti helyén használja fel az adatokat.

4.6 Párhuzamosság a videókódolásban

A videó kódolás és dekódolás párhuzamosítása alaposan feldolgozott terület, azonban a fókusz, főleg a ’90-es években, a durva felbontású módszerek nagyméretű multiprocesszoros rendszereken való megvalósításán volt. Később a hangsúly áthelyeződött a több processzort vagy DSP-t alkalmazó kisebb rendszerekre. Korábban fontos szempont volt, főleg a finom felbontású párhuzamos módszerek csoportosításában, hogy az adott megoldás hardver vagy szoftver eszközökkel valósítható-e meg, mára azonban ez a határ elmosódott. Az általános célú processzorok is végrehajtanak speciális utasításokat és a speciális hardver eszközökben is végrehajtó egységek vannak vezérlő processzor irányítása alatt.



4.11. Ábra: Időbeni felosztás, GOP és kép szinten

Általában egy kódolatlan videó feldolgozásában jelentős párhuzamosítási lehetőségek rejlenek, hiszen egymástól *független képekből* állnak amik önmagukban egymástól *független pixelekből*. A videó tömörítő algoritmusok azonban jelentős *függőségeket hoznak létre* miközben csökkentik a redundanciát. Videó kódoló és dekódoló algoritmusok megvalósítása párhuzamos feldolgozással ezért nem triviális feladat így jelentős kutatási téma [46], [50]. Videó transzkódolás párhuzamos megvalósítása azonban kevésbé feldolgozott terület [106].

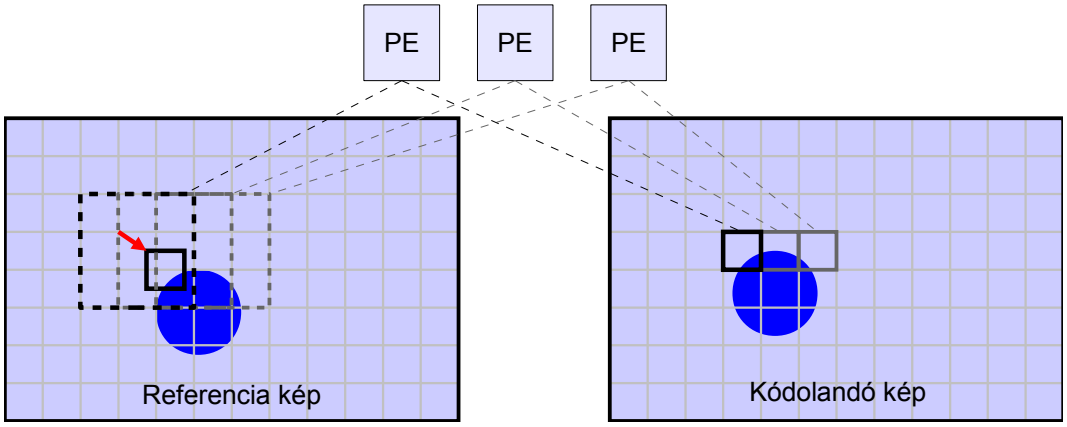
4.6.1 Időbeli párhuzamosság

Időbeli párhuzamosságnak nevezzük mikor az egyes végrehajtó egységekhez a videóból *egy vagy több kép, vagy egy teljes képcsoport* van hozzá rendelve [45], [46], [47], [52]. A problémát a képek közötti függőség okozza, mert az egyes kódolandó képek mellett a megfelelő végrehajtó egységekhez a referencia képeket is el kell juttatni. Ez egyrészt kommunikáció többletet jelent, mert egy-egy referencia képet több helyre is el kell juttatni, másrészt mivel a referencia képnek már tömörített képnek kell lennie, ütemezési problémát is jelent. A már kódolt referencia képek a kihasználható párhuzamosság mértékét is csökkentik, mert a referencia képek egymással is függőségben vannak.

Kisebb az ütemezési probléma zárt képcsoportok használata esetén, mert akkor predikciós függőség nincs a képek között, azonban a bitsebesség-vezérlésnek továbbra is információra van szüksége a korábbi képcsoportok kódolt méretéről [47].

4.6.2 Térbeli párhuzamosság

Térbeli párhuzamosság kihasználásához egy *kép különböző részei vannak más-más végrehajtó egységekhez rendelve*. A térbeli felosztás egysége lehet szelet, makroblokk vagy blokk. A processzorok egyidejűleg dolgozzák fel az egyes kép részleteket, majd kombinálják a kész videóvá. A képek közötti függőségek miatt a feldolgozó egységeknek a kódolandó képrészleteken kívül a referenciaképek megfelelő darabjaira is szüksége van a mozgásbecsléshez és mozgáskompenzációhoz, melyek a keresési ablak méretétől függően nagyobbak mint a kódolandó képrészlet, tehát az egyes processzorokhoz a referenciaképek átlapolódó részletei kerülnek, amely jelentős többlet kommunikációt és bonyolultabb ütemezést igényel. Továbbá a képek korlátozott térbeli felbontása miatt a párhuzamos feldolgozáshoz csak korlátozott számú processzor használható fel [50].



4.12. Ábra: Makroblokk alapú felosztás

Hogy mekkora térbeli egységekre van a számítás felbontva, a kommunikáció-szinkronizáció és a tényleges számítás aránya dönti el. Szorosan csatolt végrehajtó egységek, gyors kommunikáció esetén kisebb egységek (blokk, makroblokk) finomabb terhelés elosztást tesznek lehetővé. Transzkódolás esetén az egyes képrészletekre eső számítási komplexitás kisebb, rosszabb a kommunikáció-szinkronizáció és a tényleges számítás aránya, így nagyobb egység (szelet) használata a számító elemek jobb kihasználtságát eredményezi.

A szelet vagy makroblokk alapú felosztást általában *durva felbontásnak* nevezik [48], [50], [52], [53] *finom felbontású* mikor pixel vagy transzformációs együttható szintű a

párhuzamosítás. Finom felbontású párhuzamosítást használják ki a hardver mozgásbecslő megoldások [40], [41], [42], a hardver DCT algoritmusok [43], [44], a szoftveres vektor utasítások [13], vagy a GPGPU [37] megoldások.

5 Adaptív, lineáris bitsebesség – kvantáló modell

5.1 Irodalmi áttekintés

Az irodalomban főleg videó kódolókban alkalmazott bitsebességvezérlési megoldásokkal foglalkoznak, jóval kevesebb cikk témája a transzkódolás speciális bitsebességvezérlési problémái. A következőkben összefoglaltam a téma szempontjából érdekes irodalmat, három fő kategóriában: az információ elméleten alapuló bitsebességvezérlési elméletek, ρ tartománybeli megoldások és a kép komplexitásán alapuló bitsebességvezérlések.

Hang és Chen [70] egy egyszerűsített kódoló alapján ír fel egy logaritmikus bitsebesség-kvantáló modellt, amely kis kvantálási lépcsőkre, azaz nagy bitsebességű kódolásra érvényes, majd módosítja a modellt nagyobb kvantálási lépcsőkhöz és az ideálistól eltérő kódolási lépések és bemeneti jelek esetére. *Tao, Dickinson és Peterson* [71] szintén logaritmikus modellt mutat be, kiegészítve az emberi látórendszer tulajdonságait figyelembe vevő makroblokk klasszifikációval. Hat kategóriába sorolja a kódolandó makroblokkokat kvantálási zaj érzékenység szerint, és módosítja a logaritmikus modellt a klasszifikáció alapján. *Ding és Liu* [72] exponenciális bitsebesség-kvantáló függvényt ír le, amellyel a kvantáló értékét határozza meg a kódolandó képre. A kódolt kép valós bit igénye alapján szelet szinten módosítja a kvantálót a pontos cél bitszám eléréséhez. *Lin és Ortega* [73] kvantáló-torzítás összefüggés vizsgálatából kiindulva mutat be interpoláción alapuló bitsebesség-kvantáló modellt. Célja a képminőség ingadozás minimalizálása a kvantáló-torzítás összefüggés alapján. A bitsebesség-kvantáló összefüggés meghatározásához néhány kvantáló értékkel elvégzi a kódolást, a többi bitsebesség-kvantáló értéket lineáris és négyzetes interpoláció segítségével határozza meg. Az eredményeket exponenciális modellhez hasonlítja. *Chiang és Zhang* [74] az egyes képekhez tartozó bitsebesség-torzítás függvényeket négyzetes formában fejezi ki, ahol a torzítás mértékét megegyezőnek tekinti az átlagos kép kvantáló értékével. A bitsebesség-kvantáló összefüggés az inverz átlagos kép kvantáló másodfokú függvénye. Az így kialakított bitsebesség vezérlés bekerült az MPEG-4 VM5.0 modellbe.

He és Mitra [67] egy lineáris forrásmodellt mutat be, ahol a bitsebességet a kvantálás utáni nulla értékű együtthatók arányának függvényében fejezi ki, amelyet ρ tartománynak neveznek, ahol $R(\rho)$ lineáris összefüggés. A bitsebesség vezérlés kialakításához meghatározta az $R(\rho)$ paramétereit és a nulla együtthatók aránya és a kvantáló értéke közötti összefüggést, különböző videó kódolási eljárásokhoz. *Lei és Georganas* [68] kis késleltetésű változó bitsebességről állandó bitsebességre transzkódoló bitsebesség vezérlést ír le. A bitsebesség vezérlés két részből áll, a kép cél bitszámának meghatározásából és a makroblokkok kvantálóinak meghatározásából. A kvantálók meghatározásához a nem nulla értékű együtthatók aránya és a bitsebesség közötti lineáris összefüggést használja ki, a lineáris összefüggés paramétereit a bemeneti videóból nyeri ki. *Valenzise, Tagliasacchi és Tubaro* [79] statisztikus remultiplexer közös bitsebesség vezérlését mutatja be ρ tartományban. Két módszert ismertet, a minimális átlagos hibára és a torzítás változás minimalizálására optimalizálva. A bitsebességvezérléshez nem használ fel információt bemeneti videóból.

Puri és Aravind [77] egy hierarchikus komplexitás alapú bitsebességvezérlést mutat be. A komplexitás alapja a blokkok varianciája, egy kép komplexitása a blokk varianciák átlaga, a GOP komplexitás a képek komplexitásától és a mozgás intenzitásától függ. A makroblokkok komplexitását szintén hierarchikusan osztályokba sorolja, a két fő osztály a textúra és él osztály, az osztályokon belül további komplexitás szinteket határoz meg. A kvantáló értékeket a komplexitások alapján határozza meg. *Cheng és Hang* [78] egy szakaszonként lineáris

komplexitás alapú bitsebességvezérlést ír le. A komplexitás mértéke itt, az egyen komponens nélküli DCT együtthatók abszolút értékének az összege. Az egyes lineáris szakaszokat egy fa struktúra határozza meg, amely adaptívan változik a bemeneti jel függvényében. *Khan és Gu* [80] MPEG-2 transzkódolót ismertet, amely a hálózat aktuális állapotához igazítja a kimeneti bitsebességét. Komplexitás alapú, a TM5 módszeren alapuló bitsebességvezérlést alkalmaz. A képek komplexitását a bemeneti videóból, a képek kódolt mérete és a kvantáló alapján határozza meg.

5.2 Háttér

Az MPEG videó kódolásokban, intra képek esetén a képtartalom, inter kódolt képek esetén a predikciós hiba pixelek egyenlő nagyságú négyzetekre lesznek felbontva, majd ezeken a blokkokon történik a transzformáció. A transzformációs együtthatók ezután a kvantáláson esnek át.

A *kvantálás* az a kódolási lépés ahol a tulajdonképpeni *adatvesztés megtörténik*, lényege az egyes transzformációs együtthatók pontatlanabb, azaz kevesebb bittel való ábrázolása. A transzformáció energia koncentráló tulajdonsága és a kvantálás utáni pontatlanabb ábrázolás miatt a viszonylag kis amplitúdójú transzformációs együtthatók, melyek természetes képek esetén a nagyobb frekvenciájú együtthatók, nulla értékűek lesznek. A nulla értékű együtthatókat a futamhossz kódolás tulajdonképpen eltávolítja a kódolt videóból (részletesebben a 2.2.2. fejezetben).

A kép és videó kódolásokban alkalmazott perceptuális kvantálás mértékét két tényező határozza meg a kvantálási mátrix $W[w][v][u]$ amely együtthatónként határozza meg a kvantálás mértékét és egy szorzó tényező q amely segítségével néhány bit átvitelével módosítható a kvantálás mértéke.

5.2.1 Videó bitsebesség vezérlés

Állandó kvantáló érték mellett a videó bitsebessége a képtartalom függvényében jelentősen ingadozik. *A bitsebesség vezérlés feladata: a kívánt videó bitsebesség tartása* a lehető legjobb képminőség elérésével, a késleltetésre és felhasználható memóriára vonatkozó korlátok figyelembe vételével. A bitsebesség és a kvantáló közötti kapcsolatot a *bitsebesség-kvantáló függvény R(q)* határozza meg. Amennyiben ismerjük a *R(q)* függvényt az elérendő cél bitsebességhez R_t tartozó kvantáló q_t meghatározása egyszerű feladat:

$$q_t = R^{-1}(R_t) \tag{5.1}$$

Azonban a bitsebesség-kvantáló *R(q)* függvény meghatározása, a bitsebesség vezérlő kialakításának kulcskérdése, a videó kódolás összetettsége miatt nem triviális feladat.

Információ elméleti alapokon különböző *R(q)* modelleket határoztak meg, mint például a logaritmikus modell [70], [71], az exponenciális modell [72], [73] vagy a polinomiális modell [74]. Az információ elméleten alapuló bitsebesség-kvantáló modellek két feltevésre építenek. Az egyik, hogy a DCT együtthatók korrelálatlanok és Laplace vagy Gauss eloszlásúak. A másik, hogy a tömörítés utáni bitsebesség megközelítően egyenlő az entrópiával. Az első feltevés igaz az intra kódolt képekre [75], azonban nem igaz a mozgáskompenzált különbség képeken lévő DCT együtthatókra. Az entrópia meghatározása pedig, zárt formában kifejezve tetszőleges eloszlású véletlen jel esetén, általánosan nem elérhető. A transzformációs kép- és videókódások esetén, különösen nagyon kis bitsebességek mellett, az elméleti entrópia és az aktuális kódolt bitsebesség között jelentős eltérés van [67], [68]. Továbbá ezek az algoritmusok nagy számítási komplexitásúak, ha a modell paramétereit a bemeneti már előre

kódolt videó, kódolási statisztikájából kell meghatározni. Transzkóder esetén azonban ezek a statisztikák a bitsebességvezérlő rendelkezésére állnak. Kódoló esetén az aktuális kép kódolásához szükséges modell paraméterek meghatározhatók még az időben előbb kódolt képek statisztikájából, mivel az egymás utáni képek általában hasonló tulajdonságokkal rendelkeznek, ami viszont nem igaz jelenet váltáskor. Ezért az ilyen megoldások pontatlanok jelenet váltás esetén [76].

Az információ elméleten alapuló bitsebesség-kvantáló modellek mellett, a kódoláskor rendelkezésre álló adatok megfigyelésén alapuló *R(q)* modellek is léteznek. Ezek a képekhez tartozó kvantáló nagyságát valamilyen képbonyolultság, aktivitás azaz a *kép kódolási „nehézségére" jellemző mérték alapján* határozzák meg. Ennek a mértéknek a meghatározására több módszerrel is foglalkoztak az irodalomban, ilyen a DCT együtthatók valamilyen tulajdonságán alapuló módszerek – az együtthatók súlyozott varianciája, az összes együttható, vagy csak az AC együtthatók abszolút értékének összege – alapján számított aktivitás [32], [77], [78], vagy az előző képek statisztikáját használják az aktuális kép aktivitásának becslésére [1].

A fenti modellek a legjobb kifejezést próbálják meghatározni a kódolt bitsebességre a kvantáló függvényében. A forrásmodell pontosságának növelése érdekében az *R(q)* összefüggés egyre bonyolultabbá vált, növelve a kódolók számítási komplexitását. Vannak modellek azonban, melyek nem közvetlenül a bitsebesség-kvantáló függvényt próbálják meghatározni. Mivel a képtartalomhoz tartozó bitek, valójában a kvantálás utáni nem nulla értékű transzformációs együtthatókhoz tartozó, futamhossz kódokból állnak, felmerült a bitsebesség *meghatározása a kvantálás utáni együtthatókból*. Az egyik megoldás a VLC kódolt blokkok száma és a bitsebesség közötti összefüggést határozta meg [68] [69], a másik a nulla értékű transzformációs együtthatók aránya és a bitsebesség közti összefüggést vizsgálta [67]. Az utóbbi cikkben a nulla értékű együtthatók arányát ρ-val jelölték, a bitsebességet pedig ρ tartományban fejezték ki: *R(ρ)*. Az eredmények alapján megállapították, hogy tipikus transzformációs kódoláson alapuló kép- és videó tömörítő rendszerekben, a *R(ρ)* mindig lineáris függvény. A megfelelő kvantáló meghatározásához ismerni kell a *q(ρ)* összefüggést, melynek meghatározása szintén nem triviális.

$$q_t = q(\rho_t) \quad \leftarrow \quad \rho_t = R^{-1}(R_t) \tag{5.2}$$

5.3 Az adaptív, lineáris modell

A modell kialakításához a kiindulási alap az MPEG-2 kódolók esetén gyakran használt, először a TM5 [1], [32] kódolóban bemutatott módszer.

5.3.1 Kép

Az eljárás minden képhez meghatározza a kódolás által elérendő cél bitszámot és a kép kódolási „nehézségére" jellemző komplexitást. A komplexitás a kép térbeli jellemzőiből határozható meg. Az elérendő cél bitmennyiség pedig arányos a kép komplexitásával. Az így meghatározott bitszám a mérések szerint általában egy elég pontos előrejelzése a ténylegesen kapott kép méretnek.

$$b_i = f(Q_i) = \frac{c_i}{Q_i}, \tag{5.3}$$

ahol b_i a kódolandó kép tervezett mérete bitekben, Q_i a kép kvantáló értéke, c_i a kép térbeli komplexitására jellemző szám.

Azonban egy nyílt hurkú átkódolóban *nem állnak rendelkezésre a dekódolt képek* így a térbeli komplexitás meghatározása sem lehetséges a kódolás során alkalmazott módszerrel. Egy átkódolóban viszont a *bemeneti videó bitfolyam paraméterei vizsgálhatók*. Ismert az egyes képek tömörített mérete és a képhez tartozó kvantáló is. Az (5.3) egyenlet alapján feltételeztem hogy:

$$\frac{b_{ti}}{b_{oi}} = f\left(\frac{Q_{oi}}{Q_{ti}}\right) = \alpha\left(\frac{Q_{oi}}{Q_{ti}}\right) + \beta\,, \tag{5.4}$$

ahol b_{ti} a transzkódolt kép tervezett mérete, b_{oi} a bemeneti video aktuális képének mérete, Q_{oi} a bemeneti képhez tartozó kvantáló értéke, Q_{ti} az átkódolt képhez meghatározandó kvantáló, α és β a lineáris összefüggés meghatározandó paraméterei. Továbbá feltételeztem, hogy ez az érték *lineáris összefüggéssel közelíthető*.

Mivel egy videó szekvencia egymás utáni képekből áll, a bitfolyam bitsebessége az egymás utáni képek méretétől függ, tehát felírható a következő egyenletet, ahol, R_a az átlagos előre meghatározott bitsebesség, amit a kimeneten el szeretnék érni, R_o pedig a videó bitfolyam eredeti bitsebessége a bemeneten:

$$\frac{b_{ti}}{b_{oi}} \approx \frac{R_a}{R_o}\,. \tag{5.5}$$

A kimeneti bitsebesség beállításához az átkódolandó képhez tartozó új kvantáló értékét szeretném meghatározni az eredeti kvantáló ismeretében:

$$Q_{ti} = mQ_{oi}\,, \tag{5.6}$$

Q_{oi} a bemeneti képhez tartozó kvantáló értéke, Q_{ti} az átkódolt képhez meghatározandó kvantáló, m pedig az a szorzó amit szeretnék meghatározni. A (5.4) egyenletbe behelyettesítve (5.5) összefüggést és az (5.6) egyenletből kifejezve és behelyettesítve m-et a következőt kapjuk:

$$\frac{R_a}{R_o} = \alpha\left(\frac{1}{m}\right) + \beta\,. \tag{5.7}$$

Majd a fenti (5.7) egyenletből m-et kifejezve:

$$m = \frac{\alpha}{\left(\frac{R_a}{R_o}\right) - \beta}\,, \tag{5.8}$$

ahol R_a az átlagos előre meghatározott bitsebesség, amit a kimeneten el szeretnék érni, R_o pedig a videó bitfolyam eredeti bitsebessége a bemeneten. Az α és β paraméterek a (5.4) egyenletben is szereplő, meghatározandó paraméterek.

Az eddigi számításokhoz a tényleges kimeneti bitsebességet még nem vettem figyelembe csak az átlagosan elérendő bitsebességet, valamint nem vettem figyelembe azt, hogy a számításokhoz felhasznált paraméterek értéke függ a képtartalomtól. Tehát meg kell vizsgálni a tényleges kimeneti bitsebességet és ezzel korrigálni kell a számított paramétereket.

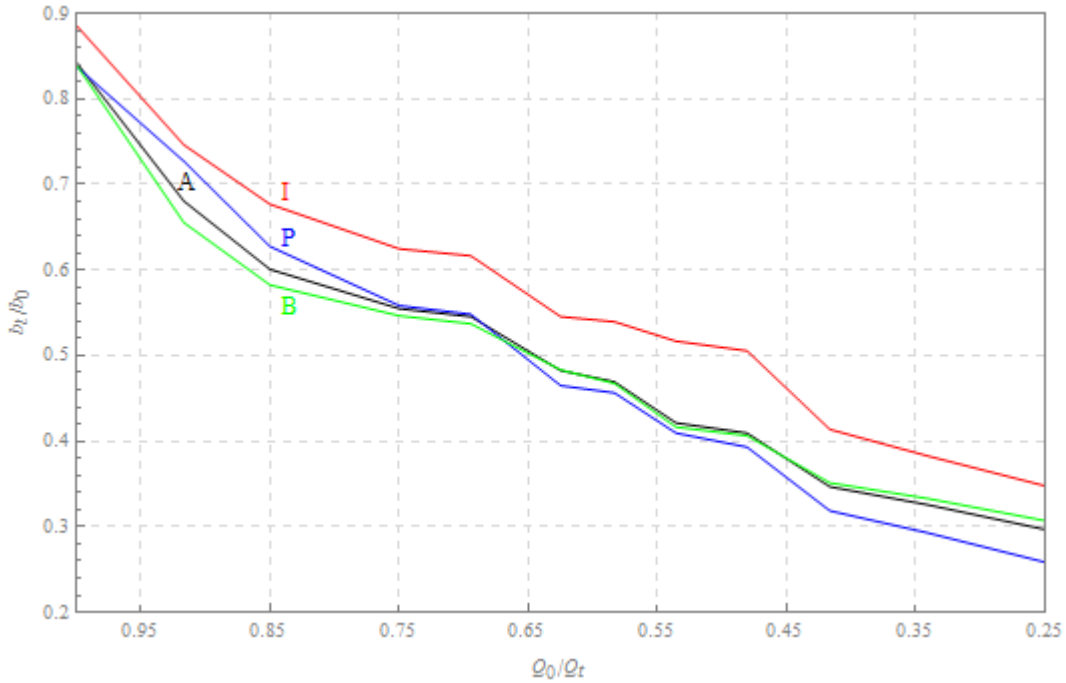
$$e = \frac{R_t - R_a}{R_a}, \tag{5.9}$$

R_t a tényleges transzkódolt bitsebesség, R_a a korábban is használt átlagosan elérendő bitsebesség, e pedig a súlyozott eltérés a kívánt bitsebességtől. Ezt az eltérést mint hiba jelet használva egy *visszacsatolást hoztam létre*, amivel befolyásolható a bitsebesség vezérlés, azaz az új kvantáló értékének meghatározása. A fentiek alapján az (5.8) egyenletet módosítva kaptam a végső összefüggést, amely minden paramétert figyelembe véve adja meg az aktuális képre használandó m szorzó értékét:

$$m = \frac{\alpha(1+e)}{\left(\frac{R_a}{R_o}\right) - \beta}, \tag{5.10}$$

R_a az átlagosan elérendő bitsebesség, R_o a videó eredeti bitsebessége a bemeneten, e a súlyozott hiba, már csak az α és β paraméterek meghatározása maradt hátra.

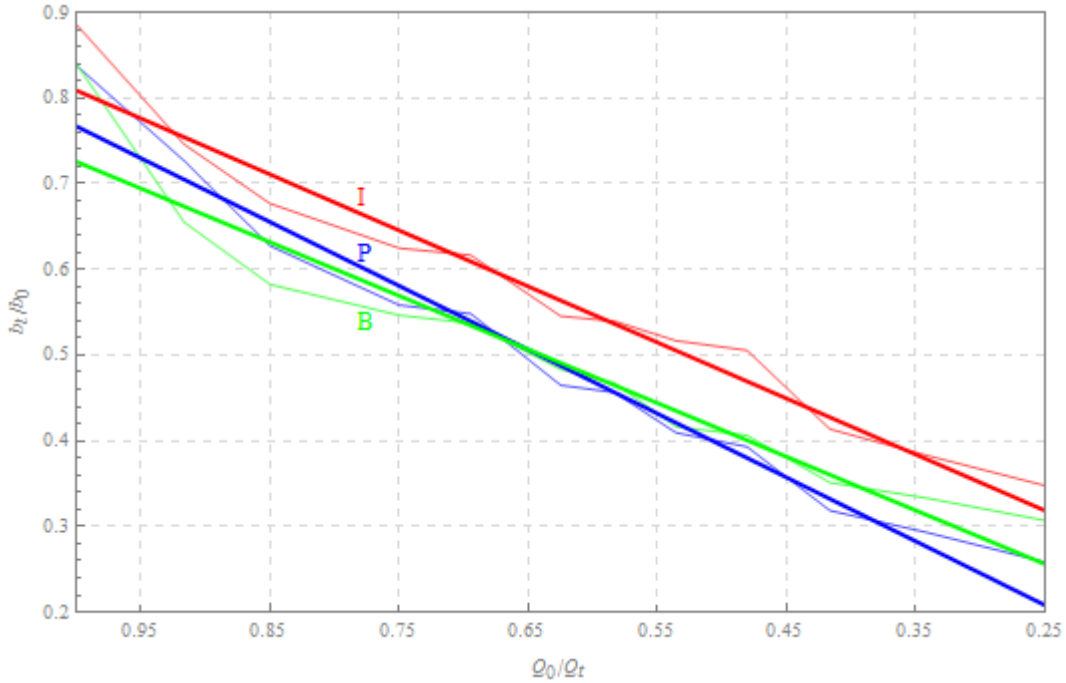
Az α és β paraméterek meghatározására méréseket végeztem, különböző beállított m értékek mellett vizsgáltam a bemeneti és a kimeneti képekhez tartozó bitszámok átlagos arányát különböző videó szekvenciákon. Az egyik, a (Mill) szekvenciához tartozó mérési eredmények a 5.1. ábrán láthatók.



5.1. Ábra: Kép méretek aránya és a kvantálók aránya közti összefüggés

A mérési eredményekből kiderül egyrészt, hogy a bitszámok aránya valóban közelíthető a lineáris (5.4) összefüggéssel, másrészt az, hogy az α és β paramétereket a különböző képtípusokhoz külön-külön kell meghatározni. A 5.2. ábrán a lineáris közelítéshez tartozó egyenesek is láthatók.

Látható továbbá, hogy az *m = 1* értékhez is, azaz változatlan kvantáló értékhez, is tartozik kép méret csökkenés. Ennek oka kerekítési hiba, amely az inverz kvantálás és az újrakvantálás számítása közben keletkezik, amely a kódolt együtthatók megváltozását okozza. Ezt a kerekítési hibát a módszer alkalmazásánál figyelembe kell venni, és a pontosabb kimeneti bitsebesség elérése érdekében az eredeti együtthatókat kell használni.



5.2. Ábra: Kép méretek és kvantálók aránya közti összefüggés lineáris közelítése

5.3.2 GOP

A különböző képtípusokban használt kódolási eljárások miatt, a különböző típusú képek kódolt mérete egy adott bitsebességen is különböző. A tényleges bitsebesség, *R,* meghatározásához adott időintervallumon, *T,* belül ismerni kell az összes kép méretét:

$$R = \frac{\sum_{n=1}^{N_I} b_n^I + \sum_{n=1}^{N_P} b_n^P + \sum_{n=1}^{N_B} b_n^B}{T\left(N_I + N_P + N_B\right)} \tag{5.11}$$

N_I, N_P és N_B az időintervallumon belül az *I*, *P* és *B* típusú képek száma, b_n az egyes képek mérete. A legegyszerűbb esetben, ha a GOP felépítése állandó, elég egy teljes képcsoportot vizsgálni, mivel egy képcsoporton belül, *I* képpel kezdődően, megtalálható az összes képtípus. Egy kódolóban, ilyen esetben általában egy GOP bitsebességét határozzák meg:

$$R_{GOP} = \frac{\sum_{n=1}^{N_{IBP}} b_n}{TN_{IBP}}, \tag{5.12}$$

ahol R_{GOP} a képcsoport bitsebessége, N_{IBP} a képcsoportban lévő képek száma, b_n a képcsoport egyes képeinek mérete, T pedig a kép idő, a kép frekvencia reciproka.

Nyílthurkú transzkódolás esetén a képcsoport felépítését nem a transzkóder határozza meg, *a GOP mérete a bemeneti videótól függ*. A videó szerkezetét a képtartalomtól és a kódolási paraméterektől függően az eredeti kódoló határozza meg. Tehát a transzkóder bitsebesség vezérlő algoritmusa nem ismeri előre az aktuálisan feldolgozott képcsoport felépítését és méretét. Az adott kimeneti bitsebességhez tartozó egyes képtípusok mérete azonban csak akkor határozható meg, ha ismert a különböző képtípusok előfordulási aránya. Tehát a bemeneten annyi időre visszamenően kell vizsgálni a videó bitfolyamot, amelyből ez az *arány meghatározható*. A változó GOP méretet is figyelembe véve ennek *több képcsoportnyi időnek kell lennie*. Ezek alapján a bemeneti R_o és a kimeneti R_t bitsebesség meghatározása (5.11) egyenlet alapján a következő:

$$R_o = \frac{\sum_{n=1}^{N_I} b_{on}^I + \sum_{n=1}^{N_P} b_{on}^P + \sum_{n=1}^{N_B} b_{on}^B}{T\left(N_I + N_P + N_B\right)} \tag{5.13}$$

$$R_t = \frac{\sum_{n=1}^{N_I} b_{tn}^I + \sum_{n=1}^{N_P} b_{tn}^P + \sum_{n=1}^{N_B} b_{tn}^B}{T\left(N_I + N_P + N_B\right)} \tag{5.14}$$

b_o és b_t a bemeneti és a kimeneti képek mérete, N_I, N_P és N_B a képek száma. A képek száma a vizsgált képcsoportok számától és felépítésétől függ:

$$N_I = N_{GOP}, \quad N_P = \sum_{n=1}^{N_{GOP}} N_{Pn}, \quad N_B = \sum_{n=1}^{N_{GOP}} N_{Bn} \tag{5.15}$$

ahol N_{GOP} a számításhoz használt képcsoportok száma, N_{Pn} és N_{Bn} az egyes képcsoportokban lévő P és B képek száma. Az I képek száma megegyezik a képcsoportok számával.

5.3.3 Makroblokk

MPEG kódolások esetén a legkisebb egység ahol a kvantáló változtatható a makroblokk. Egy kép kvantáló értéke a képben lévő *makroblokkokhoz tartozó kvantálók átlaga*.

$$Q = \frac{\sum_{n=1}^{N_{mb}} q_n}{N_{mb}} \tag{5.16}$$

5.4 Eredmények

A fenti elképzelések alapján megvalósított transzkódolót öt különböző bemeneti videó bitfolyammal vizsgáltam. Ezek két fő kategóriába tartoznak: videó kódoló és dekódoló rendszerek vizsgálatára általánosan használt *teszt videók* és valós *televízió adásból* származó vagy olyan jellegű videók.

A teszt videók közé tartozik a „Mobile”, amely bonyolult részletgazdag háttér előtt, egyszerre több irányba elmozduló szintén részletgazdag elemekből áll. A „Susie” videón egyszínű háttér előtt szinte mozdulatlan, majd egy gyors mozdulatot tevő alak látható. A

„Table Tennis” egyszerű háttér előtt mozgó játékosokat mutat, álló majd mozgó kameraállással. Videó kódolási szempontból a legbonyolultabb a „Mobile” szekvencia, legegyszerűbb a „Susie” és a kettő között helyezkedik el a „Table Tennis”. A videók 375 képkockából állnak amely a transzkódoló működése miatt kevés, ezért a vizsgálatokhoz háromszor folyamatosan egymás után kerültek lejátszásra, így lettek 1125 képkocka hosszúságúak.

Az „SVT” szekvencia valódi televízió adásból származik, főleg stúdióban felvett jelenetekből áll, a vizsgálathoz használt részlet 4000 képkocka. A „Mill” reklámokhoz és videoklipekhez hasonlóan rövid, néhány másodperces, jelenetekből és közöttük gyors váltásokból áll, a teljes szekvencia 5768 képkocka.

	Mobile			Susie			Table Tennis			Mill	SVT
R_o [Mb/s]	4	6	8	4	6	8	4	6	8	6	5.34 (VBR)
R_t [Mb/s]	1.83	1.80	1.77	1.82	1.53	1.45	1.67	1.62	1.57	2.15	2.08
m	1.32	1.75	2.12	1.33	1.60	1.78	1.47	1.79	2.10	2.35	1.95
PSNR[dB]	25.58	24.63	23.56	40.19	38.81	37.90	29.75	28.31	26.93	37.10	36.59

1. Táblázat: Bitsebesség, *m* értéke és képminőség a transzkódolt videókban

A választott kimeneti R_a és a bemeneti R_o bitsebességek a televízió műsorszórásban gyakorlatban jellemző értékek, a bemeneti bitsebesség jobb minőségű MPEG-2 SD adásnak felel meg, a kimeneti bitsebesség inkább rosszabb minőségű adásnak, ezek a bitsebességek továbbá megfelelnek a transzkóder gyakorlati felhasználásának is.

A teszt videók mindegyike három különböző bitsebességgel szerepel a vizsgálatban, 4, 6 és 8 Mb/s CBR, a bitsebességtől függetlenül azonos képtartalommal, csak minőségben tértek el egymástól. A „Mill” szekvencia 6Mb/s CBR, az „SVT” pedig változó bitsebességű, átlagosan 5.34 Mb/s.

A transzkódolás átlagos cél bitsebessége $R_a = 2\ Mb/s$-ra lett beállítva. A tényleges kimeneti bitsebességek R_t átlaga az 1. táblázatban látható. *A kimeneti bitsebességek a vártnak megfelelően nem pontosan egyeznek meg a cél bitsebességgel*, mivel a kimeneti bitsebesség VBR, de az átlagos kimeneti bitsebességek jól közelítik a cél bitsebességet. Azonban jelentősebb eltérések is megfigyelhetők a teszt videók esetén például a „Table Tennis” szekvenciánál, és mindhárom videónál a nagyobb bemeneti bitsebességek esetén. A valódi televízió műsort jobban közelítő szekvenciák esetén a kimeneti bitsebesség pontosabban közelíti a cél bitsebességet. Ennek oka lehet a videó tartalom vagy a szekvenciák hossza, a jelenség pontosabb magyarázata további vizsgálatokat igényel. A táblázatban szerepel továbbá az *m* szorzó értékének átlaga, azaz az eredeti képekhez és a transzkódolt képekhez tartozó kvantálók hányadosának átlaga.

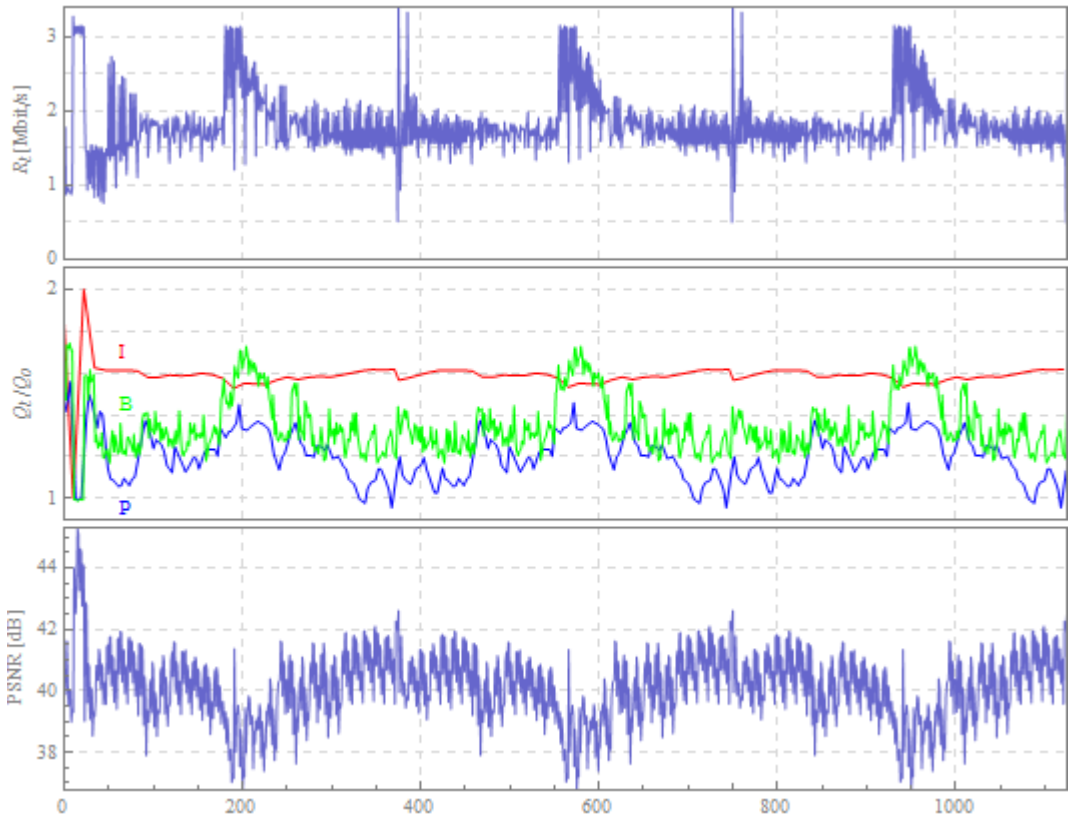
A táblázatban szerepel továbbá a transzkódolás által okozott képminőség romlás is, amely PSNR módszerrel lett meghatározva. A módszer az egyes képek eredeti és transzkódolt változatát hasonlítja össze, a táblázatban a teljes szekvencia képeihez tartozó értékek átlaga szerepel. *A minőségromlás mértéke a vártnak megfelelően alakult*. A különböző szekvenciákat összehasonlítva látható, hogy a kódolás szempontjából bonyolultabb videók esetén a bitsebesség csökkentés nagyobb mennyiségű hasznos információt távolított el a képekből, így a minőségromlás is nagyobb. A teszt videók különböző bitsebességű változatait

összehasonlítva pedig megfigyelhető, hogy a nagyobb bitsebesség csökkentés nagyobb minőségromlást okoz.

Két érdekesebb mérési eredmény, egy teszt videóhoz és egy televízió műsor, kiemelve, képkockánként ábrázolva is látható. Az 5.5. ábrán a R_o = *4 Mb/s* eredeti bitsebességű „Susie" teszt videóhoz tartozó mérési eredmények láthatók. A felső grafikon a kimeneti bitsebességet R_t ábrázolja, a középső grafikon az *m* szorzó értékét, külön az egyes képtípusokhoz, az alsón pedig az egyes képek minőségromlása figyelhető meg. Mindhárom grafikonon megfigyelhető, hogy periodikus jellegű, háromszor hasonlóan ismétlődik. Ennek oka, hogy ugyanaz a 375 képkocka ismétlődik háromszor. Megfigyelhető továbbá, hogy a grafikonok elején az értékek jelentősen ingadoznak, ami a transzkóder bitsebesség vezérlésének működéséből adódik, még nincs elegendő kép a bitsebességek kiszámításához.

A felső grafikonon jól látható, hogy a kimeneti bitsebesség változó, a cél bitsebesség R_a körül ingadozik. Ahol a képtartalom lényegében változatlan ott a kimeneti bitsebesség sem változik jelentősen, de a szekvencia közepén van néhány másodperces mozgás, itt megnövekszik a képek információ tartalma, mérete így a kimeneti bitsebesség is, majd a mozgás megszűnésével visszatér a korábbi bitsebességhez.

A középső grafikonon a bitsebesség vezérlés működése figyelhető meg, az egyes képtípusokhoz tartozó *m* szorzó értéke a kezdeti ingadozás után beáll egy érték környezetébe. Mikor a videóban az információtartalom megnő, a kimeneti bitsebesség tartása miatt az *m* értékek is megnövekszenek, majd visszatérnek egy alacsonyabb szintre.



5.3. Ábra: Bitsebesség, m értéke és képminőség, Susie 4Mb/s

Az alsó grafikonon a képminőség romlás változása hasonlóan alakul mint a fentebbi grafikonokon. A képtartalomban történő változás hatására az *m* szorzó értéke – azaz a

kimeneti és az eredeti kvantálók aránya – nagyobb lesz, így több információ veszik el az eredeti képekből, ezért a képminőség romlás nagyobb. Mikor az *m* értéke kisebb a képminőség romlás is kisebb.

Az 5.4. ábrán az „SVT” televízió műsorból származó szekvenciához tartozó grafikonok láthatók. Ezen az ábrán is a felső grafikon a kimeneti bitsebességet R_t ábrázolja, a középső grafikon az *m* szorzó értékét, külön az egyes képtípusokhoz, az alsón pedig az egyes képek minőségromlása figyelhető meg. Szintén megfigyelhető, hogy a grafikonok elején az értékek itt is ingadoznak, amíg nincs elegendő kép a bitsebességek kiszámításához. Ezek a grafikonok azonban egy hosszabb 4000 képkockából álló, nem ismétlődő videóhoz tartoznak.

A felső grafikonon megfigyelhető, hogy a változó kimeneti bitsebesség, a cél bitsebesség R_a körül ingadozik, ezen a videón azonban több jelenetváltás van, de a jelenetek hosszabbak és nem olyan hirtelen változnak.

A középső grafikonon az *m* szorzó értéke figyelhető meg, ahogy a különböző jelenetek képtartalmához alkalmazkodik a kimeneti bitsebesség megfelelő értéken tartása céljából. Látható, hogy a képtartalomtól függően jelentősen változik az *m* szorzó értéke, míg a kimeneti bitsebesség R_a körül ingadozik.

5.4. Ábra: Bitsebesség, m értéke és képminőség, SVT

Az ábra alsó grafikonján látható, hogy a képminőség romlás egyrészt a kvantálók arányának változását követi, másrészt azonban a képtartalomtól függően is ingadozik az értéke.

Mivel a GOP szerkezetet nem a transzkóder határozza meg, a kimeneti bitsebesség meghatározásához nagyobb számú képet kell együtt vizsgálni, amely több képcsoport vizsgálatát is jelenti egyben. A bitsebességvezérlés működése, főleg változó bitsebességű

bemeneti videó esetén, függ a vizsgált képek számától, amit ezért két jelentősen változó bitsebességű, valós televízió adásból származó videóval vizsgáltam. Mindkét videó egyforma hosszúságú, 15377 képkocka, ami valamivel több mint tíz perc.

	Arte			**Vox**		
N	250	1500	3000	250	1500	3000
R_o **[Mb/s]**	3.08			4.82		
R_{omin} **[Mb/s]**	1.51			2.42		
R_{omax} **[Mb/s]**	5.34			7.49		
R_t **[Mb/s]**	2.08	2.07	2.07	2.07	2.07	2.07
***PSNR*[dB]**	44.61	44.51	44.49	33.99	33.96	33.90

2. Táblázat: Bitsebességvezérlés a vizsgált képszám függvényében

A vizsgálat eredményeit a 2. táblázat foglalja össze. Mindkét videót három különböző beállítás mellett vizsgáltam, *N = 250*, *1500* és *3000* képpel, ez látható a táblázat eső sorában. A bitsebességvezérlő az utolsó *N* képet veszi figyelembe, amelyek közül csak az egész képcsoportokhoz tartozókkal végzi a kimeneti bitsebesség kiszámítását (5.14) alapján, ahol:

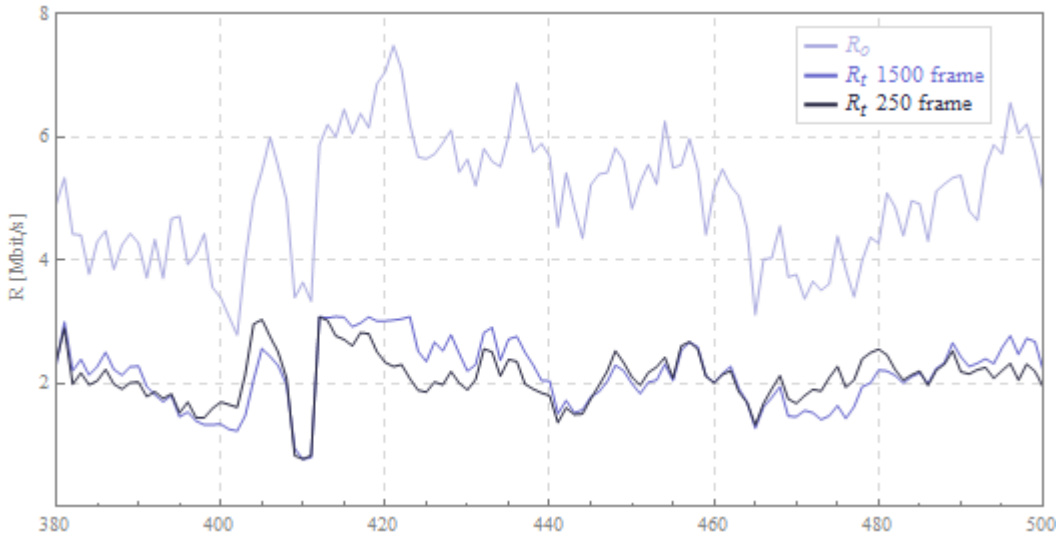
$$N \geqslant N_I + N_P + N_B . \tag{5.17}$$

A második sorban a videók átlagos eredeti bitsebessége, R_o, szerepel, a harmadik és negyedik sorban a bemeneti bitsebességek maximuma, R_{omax}, és minimuma, R_{omin}. Az ötödik sorban az átlagos kimeneti bitsebességek, R_t, láthatók a különböző képszámokhoz, ahol megfigyelhető, hogy a jelentősen változó bemeneti bitsebesség és a különböző beállítások ellenére a kimeneti átlagos bitsebességek szinte teljesen megegyeznek. A képminőség, ahogy az a táblázat hatodik sorában látható, valamilyen szinten függ a kimeneten vizsgált képek számától, de jelentős különbségek itt sem figyelhetők meg.

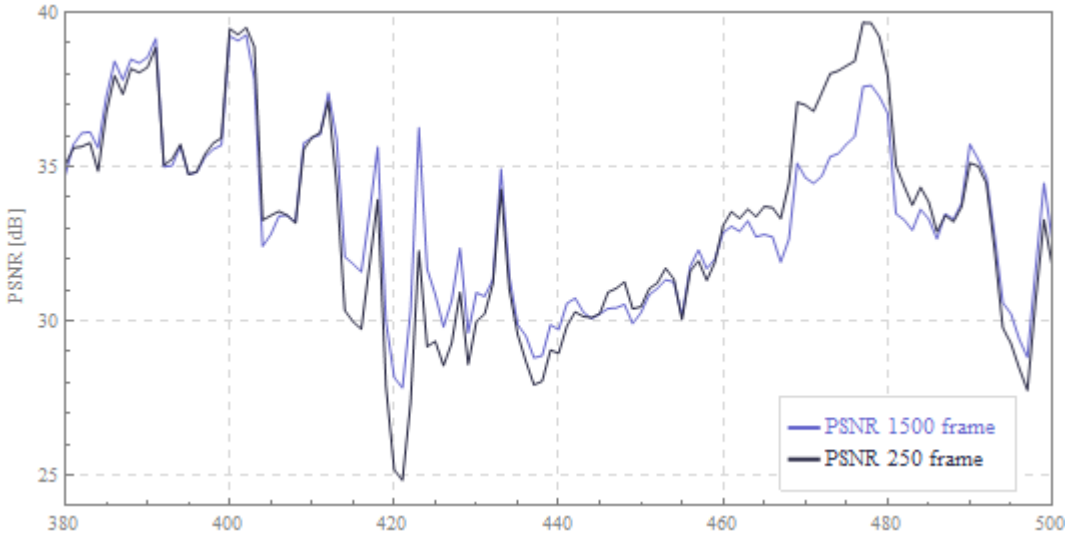
Mivel a kimeneten vizsgált képek száma, *N*, lényegében azt határozza meg, hogy a bitsebességvezérlés időben mennyivel visszamenőleg vegye figyelembe a képek méretét, érdekes vizsgálni a bitsebesség időbeli változását is. Az 5.5. ábrán a „Vox” szekvenciához tartozó grafikon látható. A grafikon a teljes videó szekvencia egy jellemző részletét, 120 képkockát, ábrázol. A függőleges tengelyen a bitsebesség, vízszintes tengelyen az idő, pontosabban a képkockák száma látható. A görbék a bemeneti bitsebességet, R_o, és az *N = 250* és az *N = 1500* értékekhez tartozó kimeneti, R_t, bitsebességet ábrázolják. Megfigyelhető, hogy a kimeneti bitsebesség a beállított R_a *= 2Mb/s* környezetében ingadozik. Továbbá megfigyelhető, hogy az ingadozás mértéke kisebb mint a bemeneti bitsebesség változásai. A kimeneti bitsebesség ingadozás az *N = 1500* képhez tartozó görbén nagyobb, azaz a kimeneti bitsebesség pontosabban követi a bemeneti bitsebességet, míg az *N = 250* beállításhoz tartozó görbe sokkal gyorsabban korrigálja a bemeneti bitsebesség változásait.

Az 5.6. ábrán a „Vox” szekvenciának ugyanahhoz a részletéhez, szintén 120 képkockához, tartozó képminőség grafikon látható. A vízszintes tengelyen az idő (képkockák száma), a függőleges tengelyen a képminőség romlás, PSNR módszerrel számítva látható. A grafikonon megfigyelhető, hogy a képminőség ingadozás, ellentétben a bitsebesség ingadozással, az *N = 1500* képhez tartozó görbén a kisebb, és a képminőség legalacsonyabb szintje is ezen a görbén

a nagyobb. Ennek magyarázata, hogy a bemeneti bitsebesség ingadozás a képminőség tartása érdekében történik. Mivel a nagyobb bitsebességhez nehezebben kódolható jelenet tartozik, és a bitsebességvezérlés ekkor leszabályoz, nem követi a bemeneti változást, akkor rosszabb képminőség lesz az eredmény. Fordítva is igaz, ha lecsökken a bemeneti bitsebesség egy könnyebben kódolható jelenetnél, és a bitsebességvezérlő megemeli a kimeneti bitsebességet, a kimeneti képminőség javulni fog ott, ahol amúgy is jobb a képminőség. Tehát egyenletesebb képminőséghez nagyobb kimeneti bitsebesség ingadozás tartozik, és fordítva.



5.5. Ábra: Bitsebesség és vizsgált képszám, Vox



5.6. Ábra: Képminőség és vizsgált képszám, Vox

5.5 Tézis

Kidolgoztam egy, a nyílthurkú transzkóder komplexitásához illeszkedően, *kis számításigényű, lineáris bitsebesség-kvantáló R(q) modellen alapuló, GOP struktúra független, visszacsatolt bitsebesség vezérlő* módszert, mely a bemeneti videó kódolási paraméterei alapján határozza meg a kimeneti videó paramétereit. Bemutattam, hogy a kimenet VBR kódolása mellett, valódi televízió adásból származó videó esetén megfelelően, 4% pontossággal, tartja a beállított átlagos bitsebességet, valamint vizsgáltam az elért képminőséget is.

5.5.1 Új tudományos eredmények

Visszacsatolt, lineáris *R(q)* modellen alapuló bitsebességvezérlés használata nyílthurkú transzkóderben, a bemeneti és kimeneti videók paramétereinek GOP struktúra független vizsgálata mellett, az irodalomban nincs publikálva.

5.5.2 Tézis igazolása

A fenti elképzelések alapján megvalósított transzkódolót teszt videókkal és valódi televízió adásból származó bemeneti videókkal vizsgáltam. Megfigyeltem a kimeneti bitsebesség és a képminőség átlagos értékét és időfüggését.

A kimeneti bitsebesség átlagos értéke a *vártnak megfelelően* nem pontosan egyezik meg a cél bitsebességgel, mivel a kimenet VBR kódolású, valódi televízió adásból származó videók esetén, 4% pontossággal közelíti a cél bitsebességet. A kimeneti képminőséget PSNR módszerrel vizsgálva, a *vártnak megfelelően,* a bemeneti bitsebességtől és a képtartalomtól függő értékeket kaptam.

5.5.3 A következtetés korlátai

A transzkódolás indulásakor nincs elegendő számú kép a bemeneti bitsebesség pontos meghatározásához és kimeneti kép a visszacsatolás pontos számításához, ezért a videó elején a kimeneti bitsebesség pontatlan lehet.

A nyílthurkú transzkóder korlátai miatt ez a bitsebesség vezérlés nem vizsgálja és kezeli a jelenetváltást és nem módosítja az egyes makroblokkok kódolási módját.

A kimeneti videó VBR kódolású, ezért a bitsebesség pillanatnyi értéke a bemeneti videó tulajdonságaitól függ. A kimeneti bitsebesség pillanatnyi és átlagos értékének vizsgálatakor, majd a következtetések levonásakor ezt figyelembe kell venni.

5.5.4 Következmények

Alacsony számításigényű bitsebességvezérlés a kis komplexitású videó transzkóder részeként lehetővé teszi valósidejű videó bitsebesség csökkentés megvalósítását többmagos homogén DSP egy processzormagján, így több videó bitsebességcsökkentését egy multiprocesszor egységgel.

5.5.5 Kapcsolódó publikációk

Bence Formanek, Tihamér Ádám, *"Rate Control in Open-Loop MPEG Video Transcoder"*, Acta Universitatis Sapientiae, Electrical and Mechanical Engineering, ISSN 2065-5916, 2009, Vol. 1., pp. 125-132.

Formanek Bence, Dalmi Dénes, Varga Attila Károly, *„Modern video kódolási eljárások és a bitsebesség csökkentés lehetőségei"*, VII. Enelko Nemzetközi Számítástechnika és Energetika-Elektrotechnika Konferencia Kiadványa, ISSN 1842-4546, 2006, p. 29-34.

Formanek Bence, *„MPEG–2 videó bitsebesség csökkentési eljárások"*, Miskolci Egyetem, Doktoranduszok Fóruma Gépészmérnöki és Informatikai Kar Szekciókiadványa, 2006, p. 58-64.

Formanek Bence, *„MPEG videó transzkódoló bitsebesség vezérlése"*, Miskolci Egyetem, Doktoranduszok Fóruma Gépészmérnöki és Informatikai Kar Szekciókiadványa, 2008, p. 12-18.

6 DSP magok közötti kommunikáció

6.1 Irodalmi áttekintés

Párhuzamos videó transzkódolókban alkalmazott kommunikációs megoldásokról nagyon kevés publikáció született, a probléma többmagos DSP-n való vizsgálatával az irodalomban nem találkoztam. *Raman et al.* [106] ismertet egy MPEG-2 formátumról H.264 formátumra konvertáló több különálló DSP-vel megvalósított elrendezést, amelyben a DSP-k között üzenetküldéses kommunikációt alakítottak ki.

Párhuzamos videó kódolóban és dekódolóban alkalmazott kommunikációval azonban sok cikk foglalkozik, ezért kiemeltem néhány, a téma szempontjából érdekes publikációt. *Xiaoshen et al.* [105] mutat be egy több DSP-ből álló heterogén rendszert és vizsgálta a kommunikációs lehetőségeket. A két TMS320DM642, egy TMS320C6455 DSP és a vezérlő PC között üzenetküldéses kommunikációt alakított ki. A két DM642 DSP között egyirányú kommunikációt Video Port segítségével, a DM642 és a C6455 között rugalmas kétirányú üzenetküldést Host Port Interface használatával, az egyik DM642 a vezérlő PC-hez PCI buszon kapcsolódott. Osztott memória alkalmazását elvetette a különböző processzorok által támogatott nem egyforma memória technológia miatt.

Több cikkben is foglalkoznak a TMS320C80 többmagos DSP-n kialakított kommunikációval, mivel ez volt az első videó feldolgozásra is alkalmas teljesítményű többmagos DSP *Cantineau és Legat* [51] funkcionális párhuzamosítást alkalmazó MPEG-2 kódolót ír le, amelyben az egyes funkciókat végrehajtó processzormagokhoz tartozó belső memóriák között, a processzor Transfer Controller egysége mozgatja az adatokat. *Tye et al.* [49] egy makroblokk alapú funkcionális párhuzamosítást használó H.263 kódolót mutat be szintén a TMS320C80 DSP-n. A kódolás során a Transfer Controller egység itt a DSP magok belső memóriái és a külső memória között mozgatja az adatcsomagokat, a DSP és a vezérlő PC PCI buszon kapcsolódik egymáshoz. *Baker et al.* [103] hasonló, azonban sokkal modernebb és gyorsabb heterogén multiprocesszoron, az IBM Cell BE [24] processzoron H.264 kódolót ismertet. Az egyes feldolgozóegységekhez a szükséges képadatok a számításokkal átlapolt DMA művelettel jutnak el, majd hasonlóan vissza a közös memóriaterületre. *Azevedo et al.* [104] egy makroblokk és kép szintű párhuzamosítást alkalmazó H.264 videó dekódert ismertet, melyet 64 darab TM3270 [29] DSP magból álló multiprocesszor szimulátoron vizsgáltak. A kommunikációhoz cache koherens osztott memóriát használt a szimulátor. *Hughes* [54] dolgozatában DSP-k közötti hálózaton MPI kommunikáció kialakítását vizsgálja.

Párhuzamos és szuperszámítógépeken megvalósított videó kódolók közül az egyik korai rendszert *Ahmad et al.* [45] írja le, egy 128 processzoros Intel Paragon XP/S számítógépen kialakított MPEG-2 kódolót, amely GOP alapú párhuzamosítást alkalmaz. A kommunikáció fő problémája ekkor a kis I/O sávszélesség volt. A fájlrendszertől a feldolgozó egységekhez eljuttatni az adatcsomagokat komoly kihívást jelentett. *Alvarez et al.* [53] makroblokk alapú párhuzamos H.264 kódolót mutat be, 64 kétmagos Intel Itanium processzort tartalmazó cache koherens NUMA rendszeren, a kommunikációhoz a megosztott memóriát használták.

Bozóki és Lagendijk [55] munkaállomások hálózatából kialakított elosztott rendszeren futó, GOP alapú párhuzamosítást alkalmazó MPEG kódolóban, kommunikációhoz a PVM rendszert használták. *Rodriguez, Gonzalez és Malumbres* [52] szintén munkaállomásokból kialakított hálózaton hierarchikusan párhuzamosított H.264 kódolót írtak le. A munkaállomások között GOP szintű párhuzamosítást alkalmaztak, a kommunikációhoz MPI

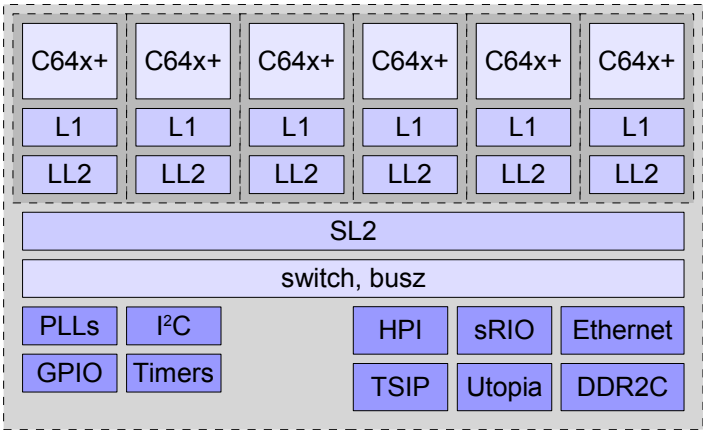
rendszert, a munkaállomásokon belül több processzor között makroblokk szintű párhuzamosítást és osztott memóriás kommunikációt használtak.

6.2 Háttér

A videó transzkódolási probléma párhuzamos felbontása mellett a processzormagok közötti *kommunikáció kialakítása jelentősen befolyásolja a párhuzamosítás hatékonyságát*. Az irodalomban található kommunikációs megoldások a felhasznált architektúrák lehetőségeit használják ki. A közös memóriás rendszerekben a megosztott memóriás kommunikáció a jellemző [53], [104]. Elosztott rendszerekben [45], [47], több különálló DSP processzorral megvalósított rendszerekben [54], [105], [106] és multiprocesszoron belül is [103] az üzenet küldéses kommunikáció valamilyen formájával találkoztam. Hibrid kommunikációs megoldást csak elosztott rendszerként kialakított közös memóriás processzorok között használtak, de ott is a párhuzamosítás két különböző hierarchia szintjén, két különböző kommunikációhoz [52]. Az általam kidolgozott párhuzamosítási megoldásokban *újdonság a kialakított processzormagok közötti kommunikációs módszer.*

6.3 A többmagos DSP

Feltevéseim igazolására olyan processzorra van szükség, amelyre jellemzőek a tipikus többmagos processzorok tulajdonságai: viszonylag *alacsony órajelen* működik *kis energia fogyasztású, több processzormag* található benne. Továbbá mivel a vizsgálandó probléma teljes egészében jelfeldolgozás, csak DSP processzormagokat tartalmazó *homogén multiprocesszorra* van szükség.



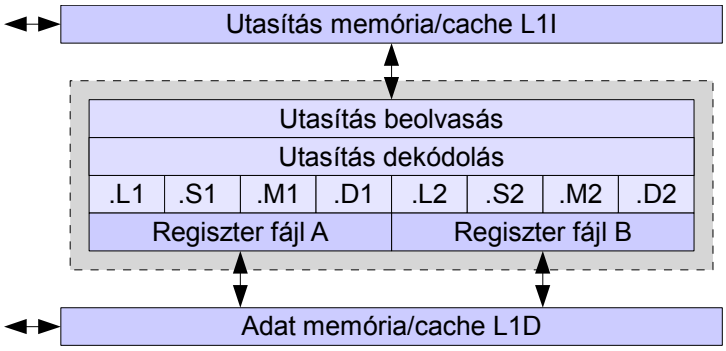
6.1. Ábra: TMS320C6472 felépítése [19]

A választott processzor a Texas Instruments TMS320C6472 típusú multiprocesszor [19], továbbiakban C6472, amelyet hat darab C64+ típusú DSP mag épít fel. A fentieken túl a választott processzor további előnye, hogy az egymagos transzkóder modell vizsgálatához használt processzor ugyanilyen C64+ típusú DSP magon alapul, megkönnyítve ezzel az eredmények összehasonlítását. Az órajel frekvenciák azonban különböznek a korábban használt TMS320DM6437 média processzor [20], továbbiakban DM6437, 600MHz-es órajelen, a most vizsgált processzor 700MHz-en működik.

6.3.1 Fizikai felépítése

A kiválasztott multiprocesszor *SoC* felépítésű, tehát a hat processzormagon túl, tartalmaz két szintű belső memória/cache rendszert és különféle perifériákat, köztük külső memória vezérlőt és Gigabit Ethernet hálózati vezérlőt, felépítése a 6.1. ábrán látható.

A 6.2. ábrán egy C64+ típusú DSP mag látható, amely fixpontos, 32 bites, *VLIW architektúrájú, nyolc végrehajtó egységet* tartalmaz. A végrehajtó egységek két szimmetrikus adatútvonalra vannak osztva, négy-négy végrehajtó egységgel és 32-32 általános célú regiszterrel. A nyolc végrehajtó egység egymástól függetlenül, *párhuzamosan hajtja végre az utasításokat*.



6.2. Ábra: C64+ DSP mag [16]

A végrehajtó egységek különböző utasítás típusokat tudnak értelmezni, de a leggyakrabban előforduló utasításokat több egység is végre tudja hajtani. A két logikai egység (.L) logikai és aritmetikai műveleteket, az eltolási egységek (.S) bitműveleteket és aritmetikai műveleteket hajtanak végre, a szorzó egységek (.M) szorzást, szorzás-akkumuláció és SIMD műveleteket, a két adatkezelő egység (.D) címszámítást és memória műveleteket végez. Egy C64+ processzormag egyszerre egy 256 bit hosszúságú utasítás csomagot olvas be, amely mind a nyolc végrehajtó egység számára egy-egy 32 bites utasítást tartalmaz, ez a nagyon hosszú utasítás szó. Ha az utasítások végrehajtásához minden feltétel adott, akkor a processzor ezt a nyolc utasítást egyszerre tudja végrehajtani.

A modern DSP-k részben örökölték a hagyományosan DSP-kre jellemző osztott adat és utasítás memóriát, de a modern DSP-k memóriarendszere rugalmasan konfigurálható. A C6472 multiprocesszor *memória hierarchiája* három szintű. Kis méretű elsőszintű (L1D, L1I) adat és program memóriát tartalmaz amely elsőszintű cache memóriaként is használható. A másodszintű memória két részből áll, az egyik lokális minden processzormaghoz (LL2) a másik egy megosztott minden mag számára elérhető memória (SL2), a lokális másodszintű memória szintén konfigurálható másodszintű cache ként is (6.1. ábra). A processzorhoz külső memória is kapcsolható, amely minden processzormag számára egységesen elérhető harmadszintű memóriaként látható. Az elrendezés ellenére minden processzormag számára minden memóriaterület hozzáférhető az egységes címtartományban, a különbség a hozzáférési időben és a címtartományban való elhelyezkedésben van.

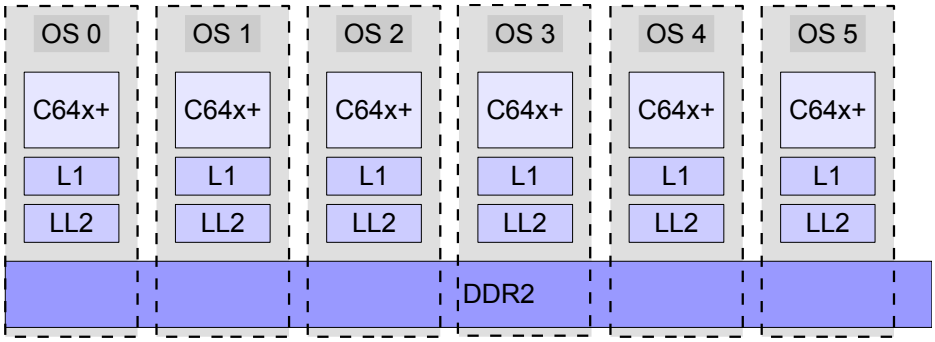
Az általános célú többmagos processzorokkal ellentétben, de a DSP-kre jellemzően, amennyiben a C6472 processzorban az L1 vagy az LL2 vagy mindkét memória cacheként van konfigurálva, a cache koherencia csak részben van biztosítva a hardver által. A cache a processzormagon belüli memória műveletekre koherens. A külső perifériák, vagy a többi processzormag általi külső memória módosítás nem jelenik meg automatikusan a cacheben,

továbbá a processzormag által módosított memória sem látható a többi mag és a perifériák által, ha csak a cache módosul. A cache megfelelő állapotát szoftveres úton kell biztosítani ilyen esetekben, a cache frissítésére és kiírására processzor utasítások léteznek.

Fizikai felépítéstét tekintve tehát, a memória konfigurációjától függően, a DSP osztott memóriás rendszer. Ha a lokális másodszintű memória (LL2), memória területként van konfigurálva, nem cache koherens NUMA architektúrának, ha az LL2 cache memóriaként van használva a külső memória mellett, nem cache koherens UMA rendszernek fogható fel.

6.3.2 Logikai felépítése

Logikai szempontból a multiprocesszor egységes memória címtartománya fel van osztva az egyes processzormagok között. Ezeken a szeparált memóriaterületeken az egyes processzormagokon, *egymástól függetlenül, valósidejű operációs rendszerek* futnak, a jelfeldolgozó alkalmazások pedig az operációs rendszereken dolgoznak, az elrendezés a 6.3. ábrán látható. Ez a felépítés jelentősen különbözik az általános célú multiprocesszorokon megszokott elrendezéstől, ahol az egységes címtartományt egyetlen operációs rendszer kezeli, a processzormagok pedig egyenrangúan futtatják az operációs rendszert vagy az alkalmazásokat.



6.3. Ábra: Független operációs rendszerek az egységes memórián

Logikai szempontból tehát, mivel az operációs rendszerek az egyes DSP magokon szeparált memóriaterületen függetlenül futnak, a multiprocesszor *elosztott rendszernek* tekinthető. Ha az operációs rendszerek memória területe az LL2 memória, akkor valóban elosztott rendszerről beszélhetünk, mert lokálisan minden rendszer azonos címtartományon működik. Ha az LL2 cacheként van konfigurálva, akkor az egyes magokon futó operációs rendszerek a külső memória címtartományán osztozva alkotnak elosztott rendszert.

Az általunk használt valósidejű operációs rendszer, a DSP fizikai felépítését is kihasználva, több szinten is támogatja a processzormagok közötti kommunikációt. Támogatja a különböző megosztott memóriaterületek kialakítását, kihasználva a belső megosztott memóriát (SL2) vagy a külső memóriát. A DSP magok közötti jelzésre támogatja a processzormagok közötti megszakítás használatát.

6.4 A kommunikáció

A processzormagok közötti kommunikáció célja, adatok *valósidejű* átvitele *kis számítási komplexitás* és *kis késleltetés* mellett, minél jobban kihasználva a processzor lehetőségeit. A 7.3. ábra szerinti, két processzormagos transzkóder kialakításban, nagy mennyiségű adatot, a teljes videó komponenst, kell minél hatékonyabban átvinni az egyik processzormagtól

(DSP 0) a másik processzormagra (DSP 1), majd a feldolgozott videót visszajuttatni az első processzormagra (DSP 0).

A DSP multiprocesszor fizikai és logika felépítése közötti, 6.3. fejezetben részletezett, különbség/ellentét miatt, az operációs rendszer által nyújtott támogatás ellenére, sem triviális a processzormagok közötti kommunikáció hatékony kialakítása. A *logikai felépítés az üzenetküldéses kommunikáció* valamilyen formájának megvalósítása felé mutat, míg a DSP *fizikai felépítése a megosztott memóriás* kommunikáció megvalósítását tenné lehetővé.

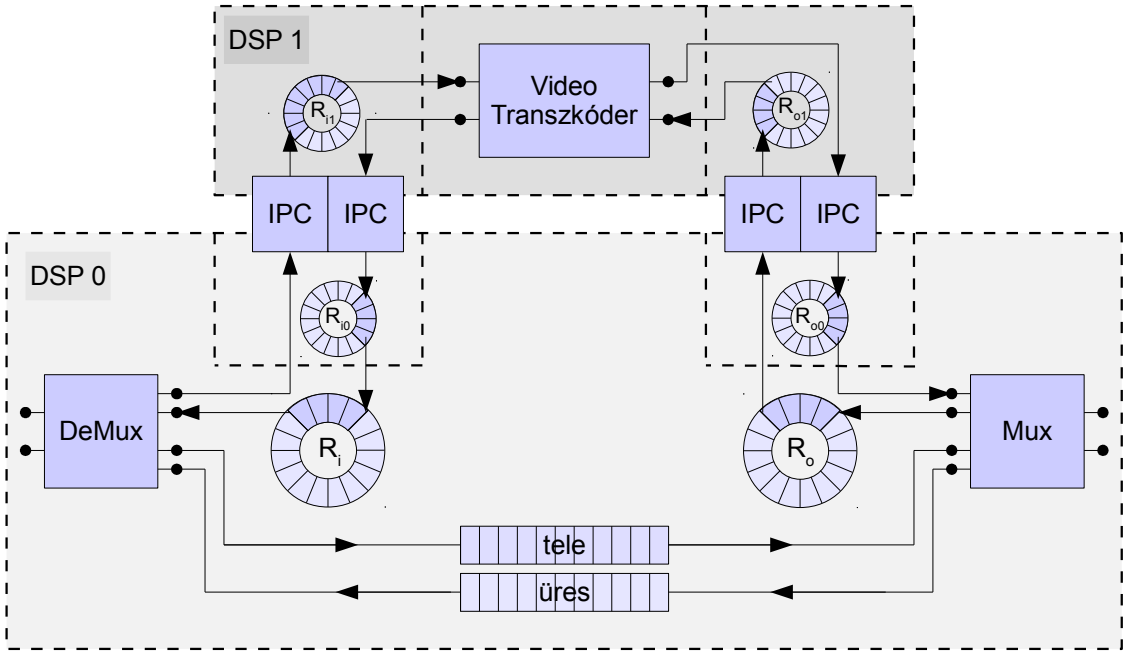
A C6472 DSP-ben a magok közötti *megosztott memóriás* kommunikációra a 6.1. ábrán látható belső megosztott memória (SL2), és a 6.3. ábrán látható külső memória (DDR2) használható. A belső megosztott memória kisebb de gyorsabb elérésű, a külső memória nagyobb hozzáférési idejű de nagyobb méretű is.

Üzenetküldéses kommunikációra egyrészt a DMA vezérlő használható, amely átmásolja az üzenetet az egyik processzormag memóriaterületéről a másik DSP mag memóriaterületére, majd jelzést küld a processzormagoknak a feladat teljesítéséről, másrészt az egyes processzormagok közvetlenül *megszakítást* küldhetnek bármelyik másik DSP maghoz és a megszakítással együtt néhány bájtnyi adat átvitelére is lehetőség van, ami szintén használható jelzések átvitelére.

Transzkóderen belül a processzormagok között a videó adatok átvitele nagy mennyiségű adat valósidejű mozgatását jelenti, amit a fenti feltételek teljesülése mellett kell megvalósítani, azaz a kis számítási komplexitással és kis késleltetéssel. Nagy mennyiségű adat másolása egyik memória területről a másikra jelentős időt igényel, amit vagy a processzor végez el, de addig számítási műveleteket nem tud végrehajtani, vagy a számításokkal átlapolva DMA, de a másolási műveletek befejezését ekkor is ki kell várni, tehát valósidejű alkalmazásoknál a memória *másolások számát minimalizálni* kell. Nagy memória bufferek gyakori lefoglalása és felszabadítása szintén jelentős számítási igényű feladat, továbbá a memóriafoglalásokhoz szükséges idő a korábbi memóriafoglalásoktól és a szabad memória mennyiségétől függően véletlenszerű és felülről nem korlátos, ezért valósidejű rendszerekben a *dinamikusan foglalt memória használata kerülendő*. A memória bufferekhez való hozzáférést a processzormagok között szinkronizálni kell, egyszerre csak egy DSP mag férhet az adatokhoz, a másiknak várnia kell. Valósidejű rendszereknél azonban a szinkronizációból adódó *blokkolást el kell kerülni*.

A fenti célok és problémák, valamint a C6472 DSP multiprocesszor hardver adottságai alapján a videó adatok gyors és kis számításigényű átviteléhez a *megosztott memóriás kommunikáció* használata adódik, és mivel nagy adatmennyiségről van szó, a belső megosztott memória (SL2) mérete nem elegendő, így a *külső memória (DDR2) használatára* van szükség. A valósidejű megvalósítás, ami a kis számítási igény mellett a korlátos maximális késleltetést is megköveteli, nem teszi lehetővé dinamikusan lefoglalt memória használatát.

A kialakított megoldás *blokkolás és memória másolás mentes, üzenetküldés és osztott memória hibrid,* tripla gyűrűs buffer segítségével megvalósítva. A dinamikus memóriafoglalás elkerülése érdekében, a processzorok közötti adatátvitelhez szükséges bufferek, egy előre lefoglalt memóriaterületen lettek elhelyezve, kialakítva ezzel az elsődleges gyűrűs buffert (R_i, R_o). A gyűrűs kialakításra azért van lehetőség mert a multimédia adatforgalom, így a videó is, bitfolyam jellegű, az egymás után érkező adatokat az érkezés sorrendjében kell feldolgozni. Továbbá az átviteli bufferek által elfoglalt memóriaterület nem lesz felszabadítva mikor már nincs rá szükség, hanem újra lesz használva, így nyilván kell tartani mely bufferek üresek és melyek vannak használatban.



6.4. Ábra: IPC tripla gyűrűs bufferrel

A DSP magok közötti kommunikáció kialakítása a 6.4. ábrán látható. A kommunikációnak két oldala van, a videó feldolgozó processzormag (DSP 1) szemszögéből, egy bementi és egy kimeneti oldal. Mindkét oldal hasonló tripla gyűrűs buffer felépítésű, de nem teljesen megegyező. A szaggatott vonallal körbevett részek az egyes processzorokon párhuzamosan végrehajtott feladatokat, valamint a processzorokon belül külön szálon, feladatban végrehajtható funkciókat jelölik.

A bemeneti oldalon az elsődleges gyűrűs bufferből (R_i) üres bufferek, pontosabban bufferekre hivatkozó mutatók, lépnek be a demultiplexer (DeMux) kimenetének bemeneti oldalán, amely feltölti ezeket a buffereket a beérkező multiplexből kiemelt videó adatokkal. A demultiplexer (DeMux) kimenetének kimeneti oldaláról a teli bufferekre hivatkozó mutatók, processzormagok közötti jelzés küldés (IPC) segítségével jutnak át a másik processzormagra (DSP 1). A jelzés küldése aszinkron művelet így nem blokkolja a küldő folyamatot. A videó transzkódolást végző DSP magon (DSP 1) a jelzés fogadása a számítási műveletekkel átlapolva, külön programszálon történik, ahol a teli bufferre hivatkozó mutató bekerül a fogadó gyűrűs bufferbe (R_{i1}). Mikor a videó transzkódernek (Videó Transzkóder) újabb adatra van szüksége, blokkolás nélkül kerül a gyűrűs bufferből a transzkóder egység (Videó Transzkóder) bemenetére a következő teli buffer. A bemeneti feldolgozás végeztével, a videó transzkóder (Videó Transzkóder) bemenetének kimeneti oldalán megjelenik a kiürített bufferre hivatkozó mutató, amely aszinkron processzormagok közötti jelzés (IPC) segítségével kerül vissza a multiplexálási feladatokat végző DSP magra (DSP 0). Ezen a processzormagon (DSP 0), szintén külön folyamatban a számításokkal átlapolva, bekerül az itteni fogadó gyűrűs bufferbe (R_{i0}) a mutató, amelyből a következő adandó alkalommal a demultiplexer (DeMux) blokkolása nélkül visszakerül az elsődleges gyűrűs bufferbe (R_i). Az elsődleges gyűrűs bufferbe (R_i) való visszakerülés tulajdonképpen csak újra felhasználhatónak jelöli a mutató által hivatkozott memóriaterületet.

A videó transzkódoló DSP mag (DSP 1) kimeneti oldalán a kommunikáció menete nagyon hasonló a bemeneti oldali kommunikációhoz. A különbség az, hogy itt az elsődleges gyűrűs

bufferből (R_o) az üres bufferekre hivatkozó mutatók először processzormagok közötti jelzés (IPC) küldés segítségével, jutnak a fogadó gyűrűs bufferbe (R_{o1}), majd onnan a transzkóder (Videó Transzkóder) kimenetének bemeneti oldalára, aszinkron módon, blokkolás nélkül. A transzkódolt videó, illetve a videó adatokra hivatkozó mutató, processzormagok közötti jelzés (IPC) segítségével jut vissza a multiplexelést végző processzormagra (DSP 0), a fogadó gyűrűs bufferbe (R_{o0}), majd onnan a multiplexer (Mux) bemenetének bemeneti oldalára. A kiürített bufferek a multiplexer (Mux) bemenetének kimeneti oldaláról kerülnek vissza az elsődleges gyűrűs bufferbe (R_o).

A fentiekből látható, hogy memória másolás nem történik a kommunikáció folyamán, csak az adatokra hivatkozó mutatók átvitele történik meg a processzormagok között. A bemeneti és a kimeneti oldali kommunikációhoz tartozó tényleges memória bufferek, a multiplexelést végző DSP magon (DSP 0) vannak lefoglalva, fix méretű gyűrűs bufferben, így dinamikus memória foglalás nem történik. A számítást végző folyamatok pedig várakozás nélkül jutnak a feldolgozandó adatokhoz, blokkolás csak akkor történik, ha nincs elég feldolgozandó adat.

A processzormagok közötti kommunikációhoz szükséges időt a kommunikáció két oldalán, t_{IPC1} és t_{IPC0}, következőképpen írhatjuk fel:

$$t_{IPC1} = t_{IPC1i} + t_{IPC1o}\,, \tag{6.1}$$

$$t_{IPC0} = t_{IPC0o} + t_{IPC0i}\,, \tag{6.2}$$

ahol t_{IPC1i} és a t_{IPC0i} a processzorok közötti kommunikáció bemeneti oldalának feldolgozási idejét jelenti, a t_{IPC1o} és a t_{IPC0o} pedig a processzorok közötti kommunikáció küldési oldalának feldolgozási idejét mutatja. Az átlagos DSP-k közötti kommunikációs időt, az egyes képekhez felhasznált idők átlagaként írhatjuk fel:

$$\bar{t}_{IPC1} = \frac{t_{IPC11}+t_{IPC12}+\ldots+t_{IPC1n}}{n}\,, \tag{6.3}$$

$$\bar{t}_{IPC0} = \frac{t_{IPC01}+t_{IPC02}+\ldots+t_{IPC0n}}{n}\,. \tag{6.4}$$

A kommunikációhoz felhasznált átlagos processzorkapacitás vagy processzor kihasználtság, $\bar{u}_{IPC1}$ és $\bar{u}_{IPC0}$, a számításhoz felhasznált idő és a vizsgált teljes időintervallum hányadosa, ami ebben az esetben a kommunikációs idők összegének és a vizsgált időintervallumra eső képidők, t_F, hányadosa:

$$\bar{u}_{IPC1} = \frac{t_{IPC11}+t_{IPC12}+\ldots+t_{IPC1n}}{n \cdot t_F}\,, \tag{6.5}$$

$$\bar{u}_{IPC0} = \frac{t_{IPC01}+t_{IPC02}+\ldots+t_{IPC0n}}{n \cdot t_F}\,. \tag{6.6}$$

A DSP-k közötti kommunikációhoz felhasznált processzorkapacitást kifejezhetjük az átlagos kommunikációs idővel is:

$$\bar{u}_{IPC1} = \frac{\bar{t}_{IPC1}}{t_F}\,, \tag{6.7}$$

$$u_{\overline{IPC0}} = \frac{t_{\overline{IPC0}}}{t_F}. \tag{6.8}$$

6.5 Eredmények

A processzormagok közötti kommunikációt a két processzormagos transzkóder modell segítségével egy Texas Instruments TMS320C6472 típusú, hat C64+ DSP magot tartalmazó multiprocesszoron vizsgáltam. A vizsgálat célja annak a feltevésnek az igazolása, hogy a *blokkolás és memória másolás mentes, üzenetküldés és osztott memória hibrid* kommunikáció hatékony és kis erőforrás igényű. A vizsgálathoz a multiprocesszor hat DSP magja közül csak kettő volt kihasználva.

A mérésekhez ugyanazok a teszt videók lettek felhasználva mint az 5.4. és a 7.4.2. fejezetben, a „Mobile & Calendar", amely bonyolult részletgazdag háttér előtt, egyszerre több irányba elmozduló szintén részletgazdag elemekből áll. A „Susie" videón egyszínű háttér előtt szinte mozdulatlan, majd egy gyors mozdulatot tevő alak látható. A „Table Tennis" egyszerű háttér előtt mozgó játékosokat mutat, álló majd mozgó kameraállással. A teszt videók 45 másodperc hosszúságúak. A „Mill" reklámokhoz és videoklipekhez hasonlóan rövid, néhány másodperces, jelenetekből és közöttük gyors váltásokból áll, a szekvencia 231 másodperc hosszúságú. A teszt videók mindegyike három különböző bitsebességgel szerepel a vizsgálatban, $R_o = 4,\ 6$ és $8\,\text{Mb/s}$ CBR, a bitsebességtől függetlenül azonos képtartalommal, csak minőségben tértek el egymástól. A „Mill" szekvencia $R_o = 6\,\text{Mb/s}$ CBR bitsebességű. A transzkódolás átlagos cél bitsebessége $R_a = 2\,\text{Mb/s}$ -ra lett beállítva.

	Mobile & Calendar			Susie			Table Tennis			Mill
R_o [Mb/s]	4	6	8	4	6	8	4	6	8	6
$t_{\overline{IPC1}}$ [µs]	1031	1433	1805	1044	1424	1781	1046	1449	1820	1462
$u_{\overline{IPC1}}$ [%]	2.58	3.58	4.51	2.61	3.56	4.45	2.61	3.62	4.55	3.66

3. Táblázat: DSP 1, IPC processzorterhelés és feldolgozási idők

A transzkódolást végző DSP magon (DSP 1) a különböző videókhoz tartozó mérési eredmények összefoglalva a 3. táblázatban láthatók. A táblázat első sorában a videók bemeneti bitsebessége, R_o, látható, a második sorban az egyes képek feldolgozásához szükséges processzorok közötti kommunikáció által felhasznált idő átlaga, $t_{\overline{IPC1}}$, szerepel mikroszekundumban, a harmadik sorban az átlagos idő alapján a (6.7) összefüggés szerint számított, a kommunikációhoz tartozó átlagos processzor terhelés, $u_{\overline{IPC1}}$, látható.

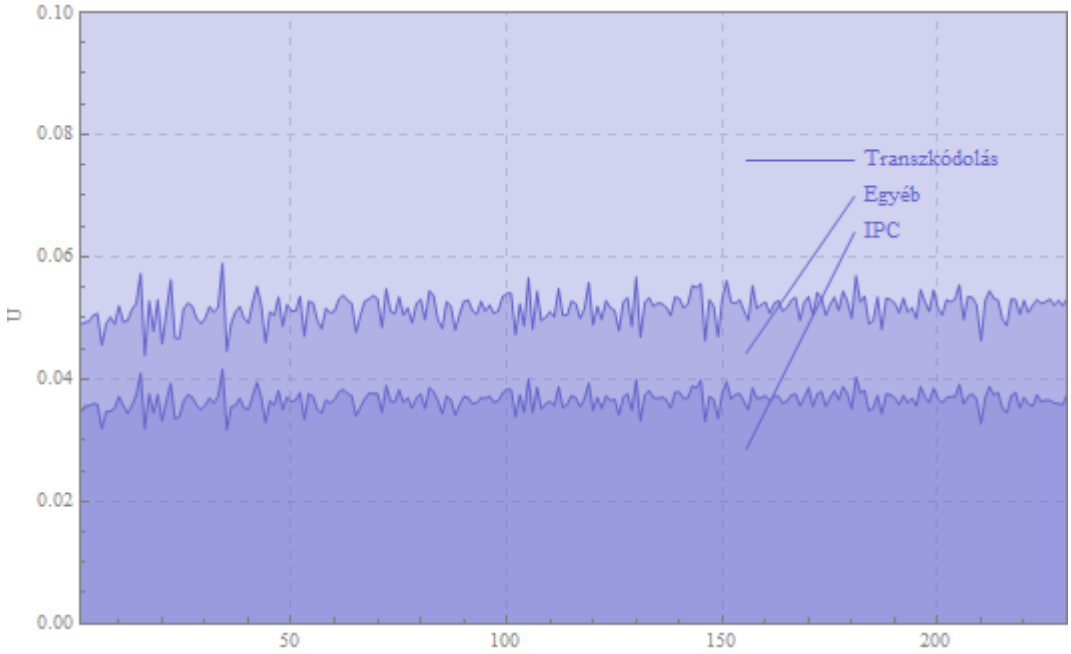
A táblázatból kiolvasható, hogy a *vártnak megfelelően* a *processzorok közötti kommunikáció végrehajtása nem terheli jelentősen* a DSP magot, és látható, hogy a terhelési adatok függnek az egyes szekvenciák bitsebességétől.

A 4. táblázatban a kommunikációt és a multiplex feldolgozást végző DSP (DSP 0) magon a különböző videókhoz tartozó mérési adatok láthatók. A táblázat első sorában ismét a videók bemeneti bitsebessége, R_o, látható, a második sorban a processzormagok közötti kommunikációra fordított átlagos idő, $t_{\overline{IPC0}}$, a harmadik sorban a 3. táblázathoz hasonlóan, az előző átlagos idő alapján és a (6.7) összefüggés szerint számított, DSP magok közötti kommunikáció, $u_{\overline{IPC0}}$, terhelése szerepel.

	Mobile & Calendar			Susie			Table Tennis			Mill
R_o [Mb/s]	4	6	8	4	6	8	4	6	8	6
$\bar{t}_{IPC0}$ [µs]	1272	1859	2460	1204	1773	2358	1223	1788	2396	1835
$\bar{u}_{IPC0}$ [%]	3.18	4.65	6.15	3.01	4.43	5.90	3.06	4.47	5.99	4.59

4. Táblázat: DSP 0, IPC processzorterhelés és feldolgozási idők

Ebben a táblázatban, hasonlóan az előző, 3. táblázathoz, is látható, hogy a processzorok közötti kommunikáció nem terheli jelentősen a DSP magokat, de ez a terhelés függ a videók bitsebességétől.



6.5. Ábra: DSP 1, IPC processzorterhelés, Mill

A 6.5. ábrán a transzkódolást végző DSP (DSP 1) terhelése látható az idő függvényében, a „Mill” szekvencia transzkódolása folyamán. A vízszintes tengelyen az eltelt idő, *t*, másodpercben, a függőleges tengelyen a processzorterhelés, *u*, látható. A terhelés $u = 0\ldots10\%$ tartományban van ábrázolva, a processzormagok közötti kommunikáció terhelésének jobb láthatósága érdekében. A processzorterhelés teljes tartományban a 7.5. ábrán látható. A világos árnyalattal színezett görbe alatti terület, tisztán a képek transzkódolásához szükséges processzorterhelést mutatja, sötétebb árnyalattal a processzorterhelés egyéb szükséges számításokhoz tartozó része látható. A legsötétebb árnyalatú görbe alatti terület a *processzormagok közötti kommunikáció által létrehozott terhelés*, u_{IPC1}. Ez a terhelés *kicsi,* $u_{IPC1} < 4\%$ *alatt marad*, és megfigyelhető, hogy *nem függ a képtartalomtól*. Ennek oka, hogy a képtartalomtól függetlenül a bemeneti videó állandó bitsebességű, tehát a processzorok között állandó bitsebességű adatot kell továbbítani.

A 6.6. ábrán szintén a „Mill” szekvenciához tartozó terhelés látható, de most a multiplex feldolgozást és kommunikációt végző DSP (DSP 0) terhelése az idő függvényében. A

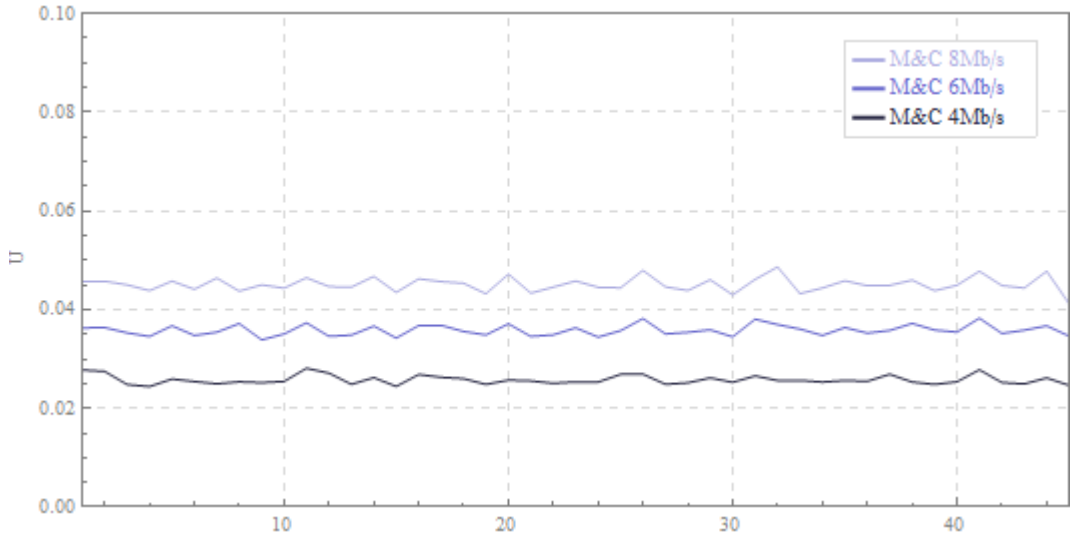
vízszintes tengelyen az eltelt idő, t, másodpercben, a függőleges tengelyen a processzorterhelés, u, látható, a terhelés $u = 0\ldots10\,\%$ tartományban van ábrázolva. A világos árnyalattal színezett görbe alatti terület a multiplex feldolgozáshoz, a videónak a multiplexből való kiemeléséhez és a transzkódolás utáni újra multiplexeléshez, szükséges processzorterhelés. A sötétebb árnyalatú görbe alatti terület az UDP/IP/Ethernet hálózaton érkező és küldött adatcsomagok kezeléséhez tartozó terhelés. A legsötétebb árnyalat a processzorok közötti kommunikációhoz tartozó terhelés, t_{IPC0}. Mivel a videó állandó bitsebességű, a multiplex feldolgozás és a kommunikáció is közel állandó terhelést hoz létre.

6.6. Ábra: DSP 0, IPC processzorterhelés, Mill

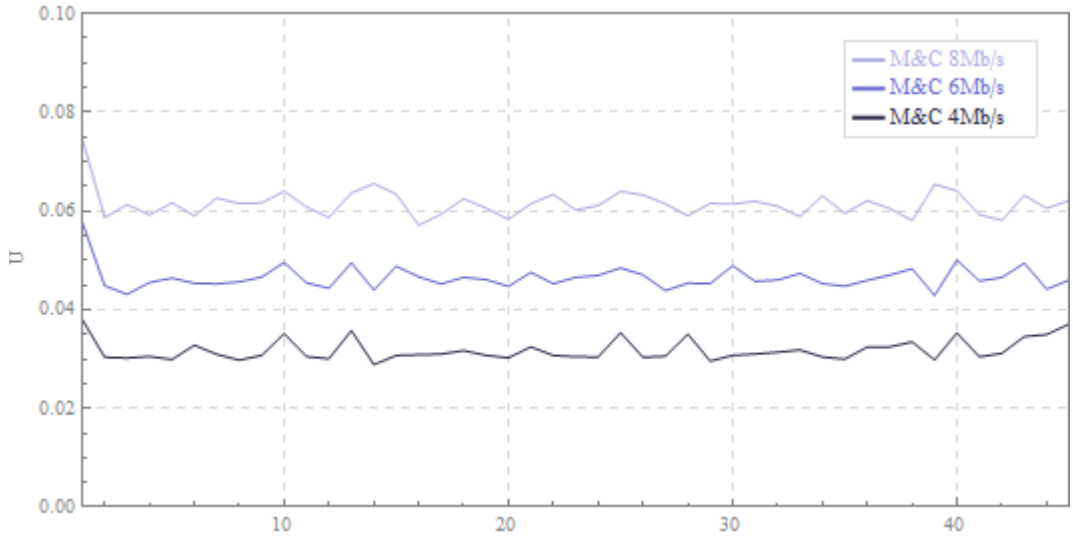
A 6.7. ábrán a transzkódolást végző DSP (DSP 1) processzorok közötti kommunikációhoz tartozó terhelése, t_{IPC1}, látható az idő függvényében, a három különböző bitsebességű „Mobile & Calendar" szekvenciához. Hasonlóan a korábbi grafikonokhoz a vízszintes tengelyen az eltelt idő, t, másodpercben, a függőleges tengelyen a processzorterhelés, u, látható a $u = 0\ldots10\,\%$ tartományban. A legnagyobb terhelést a világos árnyalatú grafikon mutatja, amely az $R_o = 8\,\text{Mb/s}$ bemeneti bitsebességű szekvenciához tartozik, világosabb árnyalattal az $R_o = 6\,\text{Mb/s}$ bemeneti bitsebességű szekvenciához tartozó alacsonyabb terhelés látható, a legsötétebb árnyalatú görbe, amely a legkisebb processzorterhelést mutatja, az $R_o = 4\,\text{Mb/s}$ bemeneti bitsebességű „Mobile & Calendar" szekvencia transzkódolásához tartozik. Megfigyelhető, hogy a *DSP-k közötti kommunikáció által létrehozott processzorterhelés jelentősen függ a videó bemeneti bitsebességétől.*

A 6.8. ábrán látható görbék a multiplex feldolgozást és kommunikációt végző DSP (DSP 0) processzorok közötti kommunikációhoz tartozó terhelését, t_{IPC0}, *mutatják az idő függvényében, három különböző bitsebességű „Mobile & Calendar" szekvenciához. A vízszintes tengelyen az eltelt idő, t, másodpercben, a függőleges tengelyen a processzorterhelés, u, látható* a $u = 0\ldots10\,\%$ tartományban. A görbék megfelelnek a 6.7. ábrán lévő görbéknek, a világos árnyalatú tartozik a legnagyobb bitsebességű szekvenciához, a sötétebb árnyalat a középső bitsebességhez, a legsötétebb árnyalat pedig a legkisebb

bitsebességű „Mobile & Calendar” szekvenciához tartozik. Megfigyelhető ezeken a görbéken is a *processzorok közötti kommunikáció bitsebesség függése*, adott képtartalom mellett.



6.7. Ábra: DSP 1, IPC processzorterhelés, Mobile & Calendar



6.8. Ábra: DSP 0, IPC processzorterhelés, Mobile & Calendar

6.6 Tézis

Többmagos DSP multiprocesszoron kialakított párhuzamos videó transzkóderhez kidolgoztam egy processzormagok közötti, az architektúrához igazodó, *logikai szempontból üzenetküldéses* és *fizikai szempontból osztott memóriás hibrid* kommunikációs megoldást, amely *blokkolás és memória másolás mentes*, hatékony kommunikációt tesz lehetővé.

6.6.1 Új tudományos eredmények

A többmagos homogén DSP fizikai és logikai felépítéséből és a valósidejű operációs rendszerből adódó speciális lehetőségeket hatékonyan kihasználó *üzenetküldéses és osztott memória hibrid* processzorokmagok közötti kommunikáció kialakítás*,* amely *blokkolás és memória másolás mentes* újdonság. Az irodalomban megosztott memóriás kommunikáció és az üzenet küldéses kommunikáció különböző formáival találkoztam. Hibrid kommunikációs megoldást azonban csak elosztott rendszerként kialakított közös memóriás processzorok között használtak, de ott is a párhuzamosítás két különböző hierarchia szintjén.

6.6.2 Tézis igazolása

A processzormagok közötti kommunikációs megoldás vizsgálatához a két processzormagon működő teljes transzkóder modellt használtam. A vizsgálat MPEG-2 MP@ML teszt videókkal és valódi televízió adásból származó bemeneti videókkal történt. A *vártnak megfelelően* a kommunikáció *nem terheli jelentősen* a DSP magokat. A tapasztalt átlagos processzor terhelés a bemeneti videó bitsebességétől függően, 2.58% és 6.15% között volt.

6.6.3 A következtetés korlátai

A kidolgozott processzormagok közötti kommunikáció vizsgálatát TI TMS320C6472 típusú többmagos DSP multiprocesszoron vizsgáltam. A kidolgozott megoldás az adott architektúra kialakítását és lehetőségeit használja ki. A vizsgálttól eltérő multiprocesszor esetén figyelembe kell venni a rendszerek felépítése közötti különbségeket.

6.6.4 Következmények

Gyors és hatékony processzormagok közötti kommunikáció segítségével az egyes DSP magok számítási kapacitása jobban kihasználható a videó feldolgozásra, ami nagyobb bitsebességű és nagyobb felbontású videók transzkódolását teszi lehetővé.

6.6.5 Kapcsolódó publikációk

Bence Formanek, Tihamér Ádám, *"DSP implementation of an MPEG-2 video bitrate transcoder"*, Proc. of the 11th International Carpathian Control Conference, ISBN 978-963-06-9289-2, 2010, pp. 195-198.

Formanek Bence, Czap László, *"Videó bitsebesség csökkentés többmagos DSP-n"*, XXVII. microCAD Nemzetközi Tudományos Konferencia Kiadványa, 2013.

7 Párhuzamosítási lehetőségek többmagos DSP-n

7.1 Irodalmi áttekintés

Videó transzkódolás párhuzamosítási lehetőségeiről nagyon kevés publikáció született, a párhuzamosítási probléma többmagos DSP-n való vizsgálatával az irodalomban nem találkoztam. *Raman et al.* [106] ismertet egy MPEG-2 formátumról H.264 formátumra konvertáló több különálló DSP-vel megvalósított elrendezést, ahol a párhuzamosítás során egy vezérlő processzor az egyes képeket a feldolgozó processzorok számával megegyező darabra bontja fel, majd a feldolgozó processzorok a képrészleteken a teljes transzkódolást elvégzik.

Videó kódolás és dekódolás párhuzamosításával azonban sok cikk foglalkozik, ezért kiemeltem néhány tipikus vagy érdekes publikációt. Az első valós időben működő szoftver kódolók párhuzamos szuperszámítógépeken futottak, *Ahmad et al.* [45] ír le egy 128 processzoros Intel Paragon XP/S számítógépen kialakított MPEG-2 videó kódolót, amely GOP alapú párhuzamosítást alkalmaz. Szintén GOP alapú párhuzamosítást használ *Bozóki és Lagendijk* [55] a munkaállomások hálózatából kialakított elosztott rendszeren futó MPEG kódoló kialakításhoz, de az egyes feldolgozó egységek egyszerre több GOP-ot dolgoznak fel. *Iwata és Olukotun* [48] MPEG-2 videó dekóder és kódoló, makroblokk és szelet szintű párhuzamosításának hatékonyságát vizsgálta, különböző multiprocesszor felépítések mellet, multiprocesszor szimulátor segítségével. Az utasítás szintű párhuzamosság kihasználhatóságát egyszerű skalár processzorral és két különböző bonyolultságú szuperskalár processzoron vizsgálták. A folyamat szintű párhuzamosítást egy négymagos bonyolultabb és egy nyolcmagos egyszerűbb processzormagokból felépített multiprocesszorral modellezték. Az utasításon belüli párhuzamosításhoz SIMD műveleteket vezettek be. Videó kódolás és dekódoláshoz a több, egyszerűbb processzormagból felépített és SIMD utasításokat is végrehajtó multiprocesszort találták a legjobbnak.

Az első videó feldolgozásra is alkalmas teljesítményű többmagos DSP a Texas Instruments TMS320C80 [21] (50MHz) heterogén multiprocesszor volt, amely négy DSP és egy RISC processzort tartalmazott. *Tye et al.* [49] mutat be egy makroblokk alapú funkcionális párhuzamosítást használó H.263 kódolót a TMS320C80 DSP-n amely QCIF (176×144) felbontású videóval 10fps képfrekvenciát ért el. Teljes kódoló rendszert vizsgáltak, de a hang kódolást és a multiplex kialakítást a vezérlő PC végezte, kialakítva ezzel egy heterogén többprocesszoros rendszert. *Cantineau és Legat* [51] szintén a TMS320C80 DSP-n alakított ki funkcionális párhuzamosítást alkalmazó MPEG-2 kódolót, amely CIF (288×352) felbontású videót 25fps képfrekvenciával kódolt. *Baker et al.* [103] hasonló, azonban sokkal korszerűbb és gyorsabb (3.2GHz) heterogén multiprocesszoron, a nyolc DSP szerű SPU és egy RISC PPU processzorból álló, IBM Cell BE [24] processzoron H.264 kódolót ismertet. A kódoló szoftver az x264 [108], makroblokk alapú párhuzamosítás kihasználásához, módosított változata. HD (1080p) felbontású videót 2.5Mb/s és 16Mb/s bitsebesség között 35fps és 15fps képfrekvencia tartományban kódolt. *Azevedo et al.* [104] ismertet egy H.264 videó dekóderben alkalmazott speciális makroblokk és kép szintű párhuzamosítást, 2D-Wave és 3D-Wave, amely figyelembe veszi a H.264 videókban a képen belüli függéseket. A módszereket egy szimulált, 64 darab TM3270 [29] (300MHz) DSP magból álló multiprocesszoron vizsgálták. Az eredményeik alapján, a csak makroblokk alapú párhuzamosítás maximum nyolc DSP magot tud kihasználni, míg a kép és makroblokk alapú mind a 64 DSP magot hatékonyan kihasználja.

7.2 Háttér

Videó transzkóder fejlesztésekor sokkal több lehetőség van, *rugalmasabban méretezhető a számítási komplexitás és a képminőség tekintetében*, mint egy kódoló vagy egy dekóder, ahogy ez a 3. fejezetben is látható. A fejlesztés kezdetekor ezért pontosan definiálni kell a méretezési célokat, ami alapján kiválasztható a megfelelő transzkóder architektúra, maghatározhatók a használandó hardver és szoftver megoldások.

A fejlesztés célja egy teljes, valósidejű, legalább MPEG-2 MP@ML videót kezelni képes bitsebesség csökkentő rendszer kidolgozása volt. Egy teljes rendszer, tehát nem csak a bitsebesség csökkentő eljárás, hanem teljes műsor multiplexet fogadni képes és teljes multiplexet előállító transzkóder. A rendszer a bemenete és kimenete MPEG-2 Transport Stream, UDP/IP (multicast), Gigabit Ethernet hálózaton. További feltétel az egyszerű, gyors felépítés, lehetőleg minél több csatorna feldolgozása, a képminőség csak e feltételek teljesülése mellett szempont.

Teljes transzkódoló rendszer kialakítására azért van szükség, mert a felmerülő problémák csak így vizsgálhatók valós környezetben, a szükséges szoftver és hardver feltételek így határozhatók meg. Az irodalomban teljes rendszer vizsgálatát nem találtam, a videó transzkódoló algoritmusokat általában különállóan vizsgálják, sokszor olyan feltételezések mellett, amely a valós videóátviteli rendszerekben nem várhatók el, például fix és ismert GOP struktúra, állandó bitsebesség, stb. [68], [73], [79], [80].

7.2.1 Transzkóder architektúra választás

A transzkódoló architektúrák közül a *homogén nyílthurkú* bitsebesség csökkentő megoldás került kiválasztásra. Ez az architektúra felel meg legjobban a tervezési céloknak, ez a transzkóder architektúrák közül az egyik legkisebb számítási komplexitású, ami illeszkedik a kis energiafogyasztású, sokmagos multiprocesszoron való implementációhoz és a célnak megfelelő képminőséget állít elő.

Hátránya, hogy jelfeldogozó processzoron, DSP-n, való implementációhoz a nyílthurkú transzkóder nem a legideálisabb architektúra, mivel a videó transzkódolás folyamatából a nagyobb számításigényű lépések kimaradnak (3.2. fejezet), ami marad az gyakori döntéshozásból és sok elágazásból álló, hagyományosan nem DSP-kre optimalizált feladat. Modern DSP-k azonban a vezérlés jellegű feladatokat is hatékonyan végre tudják hajtani.

7.2.2 Hardver kiválasztás

A processzor kiválasztásakor négy fő lehetőség merült fel: *programozható logika, használata, általános célú processzor, DSP vagy média processzor.* Az FPGA-k kis fogyasztásúak programozható funkciójúak, de nehézkesebb és nem annyira rugalmas a programozásuk mint a processzoroké, általában olyan környezetben alkalmazzák ahol a számítási komplexitás nem nagy, de nagy mennyiségű adatot kell feldolgozni nagy sebességgel. Az általános célú processzorok vagy nagy energia igényűek, nagy a számítási teljesítményük, vagy kis fogyasztásúak de akkor a számítási teljesítmény is kicsi. A média processzorok speciálisan a videó feldolgozásra lettek kifejlesztve, azonban a nyílthurkú transzkódoláshoz a speciális perifériákat nem lehet kihasználni, így csak a média processzor központi processzora használható, ami lényegében egy DSP. Mivel a videó bitsebesség csökkentés tulajdonképpen jelfeldolgozási probléma, erre a feladatra kifejlesztett processzorral, DSP-vel, kisebb energia fogyasztás mellett nagyobb számítási teljesítmény érhető el [61].

Videó transzkódolás megvalósításához azonban, mint a későbbi eredményekből is látható, általában több DSP számítási kapacitására van szükség, ami korábban csak több különálló

DSP-vel volt megvalósítható. A többmagos SoC kialakítású DSP-kel azonban a teljes transzkóder rendszer egy processzorral kialakítható, mivel ezek a multiprocesszor mellett különböző bemeneti és kimeneti perifériákat is tartalmaznak [19].

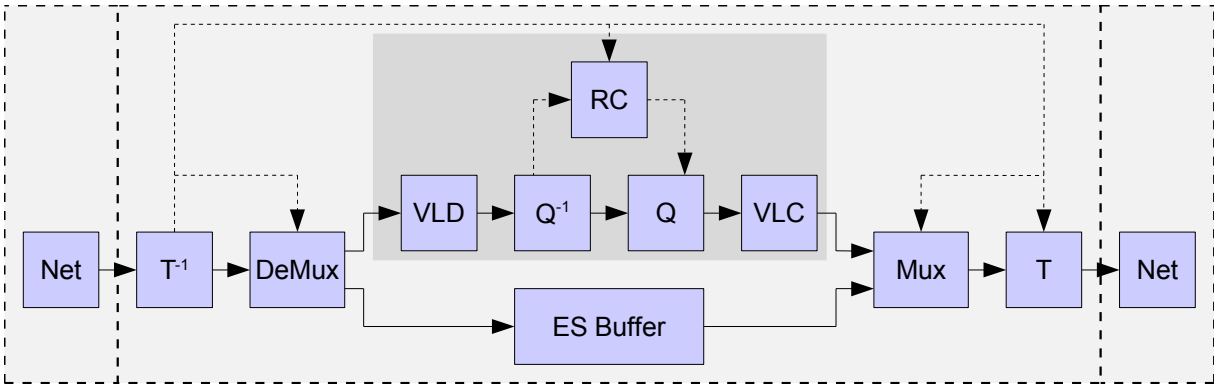
7.3 Párhuzamos megoldások

A többmagos architektúra kihasználásához az egyszálú problémát, több párhuzamosan végrehajtható kisebb feladatra kell felbontani, ami nem triviális feladat. Az irodalomban található párhuzamosítási megoldások főleg a videó kódolás és dekódolás problémáival foglalkoznak [46], [48], [50], [51], [52], [53] és nagyon kevés cikk foglalkozik speciálisan transzkódolás párhuzamosítási megoldásokkal [106]. Ezek a kutatások sem teljes rendszert vizsgálnak, hanem csak az aktuális algoritmusok párhuzamosítási lehetőségeit. A transzkóder párhuzamosítási probléma többmagos DSP-n való vizsgálatával az irodalomban nem találkoztam. Videó kódolás vagy dekódolás párhuzamosítását vizsgálták többmagos DSP-n [49], [51] és hasonló architektúrákon, mint az IBM Cell BE processzor [103], vagy sokmagos DSP szimulátor [104]. A többmagos DSP-hez legközelebb álló architektúra amin transzkódolást vizsgálták több különálló DSP-ből felépített rendszer volt [106].

A különböző párhuzamosítási lehetőségek vizsgálatára három transzkóder modell került megvalósításra. Az egy processzoron működő teljes transzkóder, a két processzormagon működő teljes transzkóder és a több processzormag lehetőségeinek vizsgálatára egy több magot is kihasználó transzkóder modell. Az általam kidolgozott párhuzamosítási megoldásokban az *újdonság egyrészt a teljes transzkódoló rendszer vizsgálata*, másrészt a *többmagos DSP multiprocesszor* használata a feladat particionálásához.

7.3.1 Egy processzoros modell

A 7.1. ábrán látható egy teljes transzkóder rendszer egyprocesszoros változata. Egyprocesszoros implementáció ellenére a hálózat kezelés külön programszálon, feladatban lett megvalósítva. Az egyes párhuzamosan végrehajtható egységeket, a 7.1. ábrán szaggatott vonallal körbevett rész jelöli, a sűrű szaggatott vonal a rendszer órajelének útvonalát mutatja. A párhuzamos végrehajtás célja ebben az esetben, nem a párhuzamos számítás, hanem a hálózati műveletek átlapolása a számítási műveletekkel, mivel a hálózatkezelésben jelentős idő a csomagok közti várakozással telik.



7.1. Ábra: Teljes transzkóder rendszer, egy processzorra

A rendszer bemenete és kimenete IP hálózat. A bemeneti egység (Net) az UDP csomagokban érkező Transport Stream-et fogadja, majd a beérkező TS multiplexből a

kiválasztott program időbélyegei, PCR, alapján az órajel helyreállító egység (T^{-1}) előállítja a rendszer órajelét. A TS demultiplexer egység (DeMux) a Transport Stream-et műsorokra bontja, majd a kiválasztott műsor videó adatait a bitsebesség átszámító egységhez felé irányítja, a többi adatcsomagot a késleltető tárolóba (ES Buffer) helyezi. A TS multiplexer (Mux) az átkódolt videót és a késleltető tárolóban lévő adatcsomagokat az időbélyegek szerint összefűzi, majd az előállított Transport Stream-et a kimeneti időzítő (T) a rendszer órajelnek megfelelő időpillanatban a kimeneti hálózati (Net) egységhez küldi, amely a kész UDP csomagokat a hálózaton továbbítja.

A videó bitsebesség ászámító egység az ábrán sötétebb szürke színnel kiemelve látható. Ez egy nyílthurkú transzkóder, melynek bemenete az eredeti demultiplexált videó elementary stream, a kimenete a transzkódolt, kisebb bitsebességű szintén videó elementary stream. A bitsebesség vezérlő egység (RC) szabályozza a kimeneti bitsebességet.

A transzkóder által feldolgozott egyes képek számítási ideje jelentősen ingadozik, függően a bemeneti és a kimeneti bitsebességtől, a képtartalomtól és a kódolt kép típusától. A jelen vizsgálathoz a valósidejű feldolgozás feltétele úgy lett meghatározva, hogy az átlagos képfeldolgozási idő $\bar{t}_{Ft}$ kisebb (vagy egyenlő) legyen a videóban a kép ismétlődési időnél t_F, vagy a feldolgozás átlagos képfrekvenciája $\bar{f}_{Ft}$ nagyobb (vagy egyenlő) legyen a videó képsebességénél f_F, valamint a maximális képfeldolgozási idő t_{Ftmax} ne haladja meg a transzkódolás kezdeti késleltetését t_{dt}:

$$\bar{t}_{Ft} \leqslant t_F, \quad \text{vagy} \quad \bar{f}_{Ft} \geqslant f_F, \tag{7.1}$$

$$t_{Ftmax} \leqslant t_{dt}. \tag{7.2}$$

Az egyes képek számításához szükséges idő t_{FC}, a 7.1. ábrán látható felépítés szerint, a változó szóhosszúságú dekódoláshoz szükséges időből t_{VLD}, az inverz kvantálás idejéből t_{Q1}, a bitsebesség vezérlés számítási idejéből t_{RC}, a kvantálás t_Q és a változó szóhosszúságú kódolás t_{VLC} idejéből áll össze. Az átlagos képfeldolgozási idő $\bar{t}_{Ft}$, a mérésben felhasznált egyes képek feldolgozási idejének átlaga:

$$t_{FC} = t_{VLD} + t_{Q1} + t_{RC} + t_Q + t_{VLC}, \tag{7.3}$$

$$\bar{t}_{Ft} = \frac{t_{FC1} + t_{FC2} + \ldots + t_{FCn}}{n}. \tag{7.4}$$

A teljes transzkódolási idő, szintén a 7.1. ábra szerint, kiegészül továbbá a kommunikációhoz szükséges idővel t_{nin}, t_{nout}, a demultiplexálás és a multiplexálás idejével t_{DMUX}, t_{MUX}, valamint a szinkronizáció és órajel helyreállítás és előállítás idejével t_{TI}, t_T. Az átlagos képfeldolgozási idő ebben az esetben is az egyes képek feldolgozási idejének átlaga, de most ez a, $\bar{t}_{Ftc}$, teljes feldolgozási idő:

$$t_{TC} = t_{nin} + t_{TI} + t_{DMUX} + t_{FC} + t_{MUX} + t_T + t_{nout}, \tag{7.5}$$

$$\bar{t}_{Ftc} = \frac{t_{TC1} + t_{TC2} + \ldots + t_{TCn}}{n}. \tag{7.6}$$

A transzkódoláshoz felhasznált átlagos processzor kapacitás vagy processzor kihasználtság, $\bar{u}$, a számításhoz felhasznált idő és a vizsgált teljes időintervallum hányadosa,

ami ebben az esetben a képfeldolgozási idők összegének és a vizsgált időintervallumra eső képidők hányadosa:

$$\bar{u_F} = \frac{t_{FC1}+t_{FC2}+\ldots+t_{FCn}}{n \cdot t_F}, \tag{7.7}$$

$$\bar{u_F} = \frac{\bar{t_{Ft}}}{t_F}, \tag{7.8}$$

ha a (7.7) egyenletben a vizsgált képek számával egyszerűsítjük a törtet, (7.8) egyenletet kapjuk amiben az átlagos processzor kihasználtságot, $\bar{u_F}$, az átlagos képszámítási idő és a képidő hányadosa határozza meg.

$$\bar{u_{Ftc}} = \frac{\bar{t_{Ftc}}}{t_F} \tag{7.9}$$

A (7.8) egyenlet csak a kép átkódolás idejét veszi figyelembe, de hasonlóan a teljes transzkódolásra is felírható a processzor kihasználtság (7.9). A valósidejű feldolgozás fenti feltételei alapján (7.1), tehát teljesülnie kell, hogy:

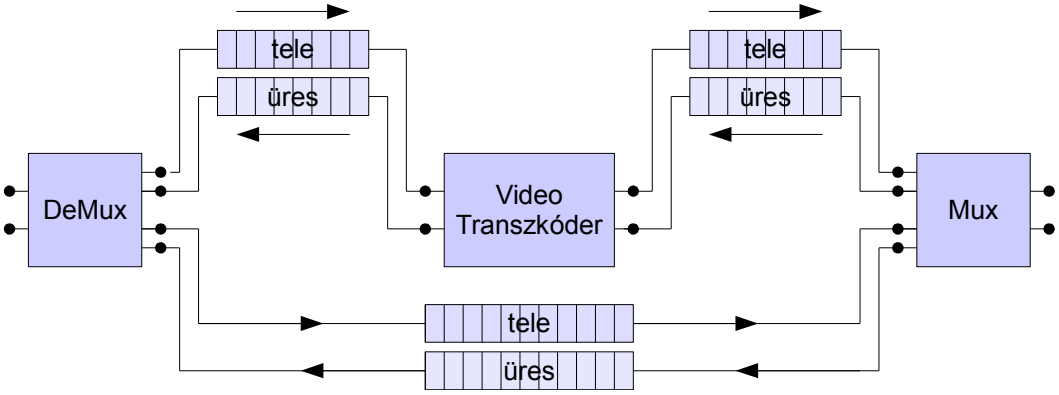
$$\bar{u_{Ftc}} \leqslant 100\% . \tag{7.10}$$

7.3.2 Két processzormagos modell

A két processzormagos transzkóder modell *funkcionális párhuzamosításnak* tekinthető, ahol a videó transzkódolás funkció az egyik processzormagon, a kommunikáció és a multiplex feldolgozás a másik processzormagon működik. Többmagos, n_p, processzort tekintve ez a funkcionális felosztás, egy olyan elrendezés vizsgálatát teszi lehetővé, amely egyszerre több műsor, több multiplex feldolgozására alkalmas. Egy kivételével a processzormagok, $n_p - 1$, a videó transzkódolást végzik, egy processzormag pedig az összes multiplexhez tartozó többi feladatot ellátja. Terhelés elosztás szempontjából ez a funkcionális felosztás akkor hatékony, ha a videó transzkóderek kihasználják egy-egy processzormag számítási teljesítményét, és az összes műsorhoz tartozó kommunikációt és multiplex feldolgozást egy processzormag el tudja látni.

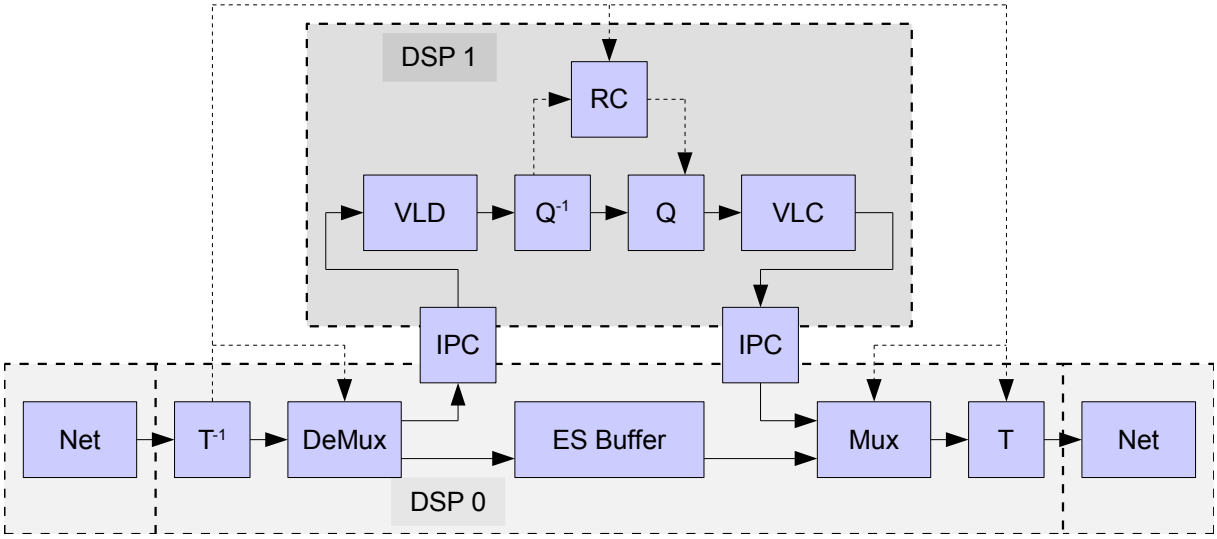
A videó transzkóderek számítási komplexitás szempontjából rugalmasabban méretezhetők, mint a kódolók vagy dekódolók, ahogy ez a 3. fejezetben is látható. A feladat tehát, egyrészt egy olyan videó transzkóder kialakítása, amely teljesen kihasználja a multiprocesszor egy processzormagjának számítási teljesítményét, kielégíti a valósidejű működés feltételeit és a képminőséggel szemben támasztott követelményeket, másrészt a kommunikáció és multiplexálási feladat számítási igényét megfelelően alacsonyan tartja.

A particionálási probléma ebben az esetben tehát nem a videó átszámító algoritmus párhuzamosítása, hanem *rendszer szintű párhuzamosítási feladat*. Multimédia rendszerekben, amilyen rendszer egy teljes videó transzkóder is, az egyes funkcionális, multimédia elemek egységes felülettel, interfésszel csatlakoznak egymáshoz. A videó transzkóder egy része kiemelve, multimédia elem szinten, a 7.2. ábrán látható. Az egyes multimédia elemek, interfészeik között, adat bufferek segítségével továbbítják a feldolgozandó adatokat, amiből adódik, hogy a funkcionális particionálás *multimédia elem szinten* könnyebben elvégezhető.



7.2. Ábra: Transzkóder, multimédia elem szinten

A fentiek alapján tehát a feltevésem a következő: valósidejű transzkóder rendszer megvalósítható úgy, hogy a *videó feldolgozó egység egy (DSP 1)* magon működik, a rendszer *többi eleme pedig egy másik (DSP 0)* magon. Továbbá feltesszük, hogy a tervezési célok alapján kiválasztott nyílthurkú transzkóder *a videó átszámítását egy processzormagon el tudja végezni és közel teljesen kihasználja a processzormag számítási teljesítményét*, amennyiben a processzorok közötti kommunikációhoz szükséges számítási igény alacsonyan tartható, és a kommunikáció és a multiplex feldolgozáshoz szükséges számítási komplexitás egy processzormag teljesítményének csak egy részét használja ki. A két processzormagos transzkóder modell logikai felépítése a 7.3. ábrán látható.



7.3. Ábra: Teljes transzkóder rendszer, két processzormagra

A két processzormagos transzkóder felépítése nagyon hasonló az egy processzoros kialakításhoz, lényegében ugyanolyan funkciójú logikai elemekből épül fel. A különbség, hogy a szaggatott vonallal körbevett részek itt az egyes processzorokon párhuzamosan végrehajtott feladatokat, nem csak a processzoron belül külön szálon, feladatban végrehajtható funkciókat jelenti. A processzorok közötti kommunikációt (IPC) külön elemek jelölik.

A két processzormagos rendszer bemenete és kimenete is IP hálózat. A hálózatkezelés a processzoron belül külön programszálon fut, így a hálózati műveletek átlapolhatók a számítási műveletekkel. Az első processzoron (DSP 0) működő bemeneti egység (Net) az UDP csomagokban érkező TS-t fogadja, majd a beérkező TS multiplexből a kiválasztott program időbélyegei, PCR, alapján az órajel helyreállító egység (T^{-1}) előállítja a rendszer órajelét. A TS demultiplexer egység (DeMux) a Transport Stream-et műsorokra bontja, majd a kiválasztott műsor videó adatait a másik processzoron (DSP 1) futó bitsebesség átszámító egység felé irányítja (IPC), a többi adatcsomagot a késleltető tárolóba (ES Buffer) helyezi. A TS multiplexer (Mux) az átkódolt videót és a késleltető tárolóban lévő adatcsomagokat az időbélyegek szerint összefűzi, majd az előállított Transport Stream-et a kimeneti időzítő (T) a rendszer órajelnek megfelelő időpillanatban a kimeneti hálózati (Net) egységhez küldi, amely a kész UDP csomagokat a hálózaton továbbítja.

A videó bitsebesség átszámító egység a másik processzoron (DSP 1) fut, az ábrán sötétebb szürke színnel kiemelve látható. Ez a nyílt hurkú transzkóder, melynek bemenete az eredeti demultiplexált videó elementary stream, a kimenete a transzkódolt, kisebb bitsebességű szintén videó elementary stream. Processzorok közötti kommunikáció (IPC) segítségével jut el a demultiplexált videó az egyik processzoron (DSP 0) lévő demultiplexer (DeMux) kimenetéről a másik processzormagon (DSP 1) lévő transzkóder bemenetére. Feldolgozás után, a második processzorok közötti kommunikációs egység (IPC) juttatja vissza a kisebb bitsebességű videót az első processzormagon (DSP 0) lévő multiplexerhez (Mux). A bitsebesség vezérlő egység (RC) szabályozza a videó kimeneti bitsebességét.

Ebben a felépítésben a valósidejű működéshez a (7.1) és a (7.2) feltételeknek mindkét processzormagon, a következő összefüggések alapján kell teljesülnie:

$$t_{DSP1} = t_{IPC1i} + t_{VLD} + t_{Q1} + t_{RC} + t_{Q} + t_{VLC} + t_{IPC1o}\,, \tag{7.11}$$

$$t_{DSP0} = t_{nin} + t_{T1} + t_{DMUX} + t_{IPC0o} + t_{IPC0i} + t_{MUX} + t_{T} + t_{nout}\,, \tag{7.12}$$

$$\bar{t}_{Ft1} = \frac{t_{DSP11} + t_{DSP12} + \ldots + t_{DSP1n}}{n}\,, \tag{7.13}$$

$$\bar{t}_{Ft0} = \frac{t_{DSP01} + t_{DSP02} + \ldots + t_{DSP0n}}{n}\,, \tag{7.14}$$

ahol az egyes képek számításához szükséges idő az egyik processzoron (DSP 0) t_{DSP0}, a másik processzoron (DSP 1) t_{DSP1}, az átlagos képfeldolgozási idők az egyes processzorokon: $\bar{t}_{Ft0}$, $\bar{t}_{Ft1}$. Az új változók a t_{IPC1i} és a t_{IPC0i} a processzorok közötti kommunikáció bemeneti oldalának feldolgozási idejét jelenti, a t_{IPC1o} és a t_{IPC0o} pedig a processzorok közötti kommunikáció küldési oldalának feldolgozási idejét mutatja.

A transzkódoláshoz felhasznált átlagos processzor kapacitás vagy processzor kihasználtság, a számításhoz felhasznált idő és a vizsgált teljes időintervallum hányadosa, amit az egyes processzormagokra a (7.8) egyenlet alapján úgy lehet felírni, hogy:

$$\bar{u}_{F1} = \frac{\bar{t}_{Ft1}}{t_F}\,, \tag{7.15}$$

$$\bar{u}_{F0} = \frac{\bar{t}_{Ft0}}{t_F}\,, \tag{7.16}$$

ahol $\bar{u}_{F0}$ a kommunikációs és multiplex feldolgozó processzormag (DSP 0), $\bar{u}_{F1}$ a videó transzkódolást végző processzormagon (DSP 1) az átlagos processzor kihasználtság. Ahhoz hogy a feltevéseim igazolva legyenek, teljesülnie kell a második processzormagon (DSP 1):

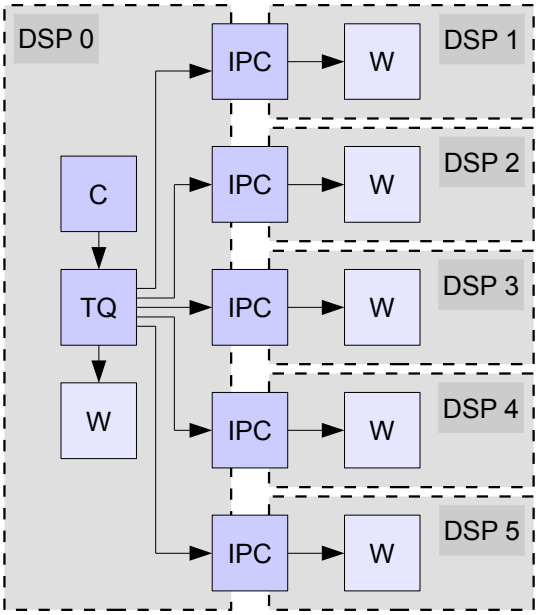
$$\bar{u}_{F1} \leqslant 100\%, \quad \text{és} \quad \bar{u}_{F1} \approx 100\%, \tag{7.17}$$

valamint, ha a vizsgálatokhoz használt multiprocesszor alapján az $n_p = 6$ processzormagos esetet tekintjük, $n_p - 1$ transzkódoló DSP maggal és egy kommunikációs és multiplex feldolgozó processzormaggal, akkor az első processzormagon teljesülnie kell:

$$\bar{u}_{F0} \leqslant \frac{1}{n_p - 1} = \frac{100}{5}\% = 20\%. \tag{7.18}$$

7.3.3 Több processzormagos modell

Párhuzamos rendszerekben a feladatok hatékony felosztása, függ a processzorok számítási teljesítményétől, a kommunikáció sebességétől, a feladatok komplexitásától. A többmagos DSP processzormagjai között az osztott memóriás elrendezés gyors kommunikációt tesz lehetővé, a videó transzkódolás pedig nagy számítás igényű feladat, ezért a videó feldolgozás térbeli párhuzamosítása hatékonyan használható. Az eddigi modellektől eltérően több processzormagos transzkóder modell nem egy teljes transzkóder rendszer, csak a videó átszámító algoritmus párhuzamosíthatóságának vizsgálatára készült, logikai felépítése a 7.4. ábrán látható.



7.4. Ábra: Transzkóder modell, több processzorra

A feltevésem a következő: *többmagos DSP használata esetén a nyílthurkú transzkóderben* hatékonyan kihasználható a *térbeli párhuzamosítás* mivel a videó adatok feldolgozásakor nincsenek függőségek a képadatokon belül, és a szorosan csatolt processzormagok között gyors a kommunikáció. Videó kódolás vagy dekódolás esetén a képek között, vagy képen belül, a predikció miatt adat függőségek vannak [53], azonban nyílthurkú transzkóderben

mivel a predikciós lépés kimarad, ezek a függőségek nem jelennek meg. A transzkódolás során a változó szóhosszúságú dekódolás és kódolás lépés azonban alapvetően lineáris feladat, nem párhuzamosítható, de ebben a modellben ez nincs figyelembe véve. Ennek oka egyrészt, hogy a vizsgálat célja a párhuzamosíthatóság hatékonyságának elemzése a multiprocesszoron, másrészt képszelet szinten a változó szóhosszúságú dekódolás és kódolás is párhuzamosítható.

A több processzormagos transzkóder modell, 7.4. ábra, működése a következő: a vezérlő DSP magon a vezérlő egység (C) felbontja a feladatot párhuzamosan feldolgozható részfeladatokra, a részfeladatokat elhelyezi a *központi feladat sorban* (TQ). Az egyes DSP magokon lévő végrehajtó egységek (W), processzormagok közötti kommunikáció segítségével (IPC), a feladat sorból kiemelnek egy részfeladatot, amit végrehajtanak. A vezérlő processzormagon is fut végrehajtó egység (W), de mivel a feladat sor számára lokális, nem szükséges a processzormagok közötti kommunikációt igénybe vennie egy részfeladat leemeléséhez. A végrehajtó egységek (W) amint befejezték az aktuális részfeladat végrehajtását, a *dinamikus ütemezés* szabályai szerint, a központi feladatsorhoz (TQ) fordulnak és leemelnek egy újabb részfeladatot.

A *központi feladatsor* és *dinamikus ütemezés* alkalmazása viszonylag egyszerű megvalósítást tesz lehetővé, a statikus ütemezéshez képest azonban jobb terhelés eloszlást biztosít, hátránya, hogy az egyes DSP magok hozzáférését a feladat sorhoz *szinkronizálni kell*. Sok processzoros rendszerekben, vagy olyan elosztott rendszerekben ahol a szinkronizációhoz tartozó kommunikáció késleltetése nagyobb, a központi feladatsor nem a legjobb választás [53], [104]. A vizsgált $n_p = 6$ kommunikáció szempontjából közeli processzormag esetén, a szinkronizációból adódó veszteség várhatóan nem jelentős.

A megvalósítást nehezíti, hogy az egyes DSP magokon futó végrehajtó egységek egymástól *független valósidejű operációs rendszeren* futnak, bár *közös memória címtartományt* használnak, ez részletesebben a 6.3.2. fejezetben látható. A közös címtartomány miatt a feldolgozandó adatokhoz minden DSP egyformán hozzáférhet, de a *cache koherenciát* a DSP magok között szoftverből kell biztosítani. A külön operációs rendszerek miatt a feladatsorban nem lehet *program szálakat vagy függvényhívásokat* elhelyezni, mivel azok a *független operációs rendszereken nem értelmezhetők*. A részfeladatokat végrehajtó programkódoknak minden DSP magon külön-külön jelen kell lennie. A feladat sor tehát egy adatstruktúrát tartalmaz, amely a részfeladat kódját és a paramétereit tartalmazza. A szinkronizációt szintén bonyolítja a több operációs rendszer használata, az egyszerű szinkronizációs primitívek csak egy operációs rendszeren belül érvényesek. A processzormagok közötti kommunikációnak tehát, a részfeladatok leírását tartalmazó struktúra átvitelén túl, a két operációs rendszeren lévő független *szinkronizációs egységeket is össze kell hangolnia*. Ez a megvalósítás *jelentősen különbözik az általános célú multiprocesszorokon futó, közös operációs rendszeren lehetséges megoldástól*, ahol az egységes memória címtartomány mellett, egységes programszál és szinkronizáció kezelés elérhető.

A több processzormagos végrehajtás által elérhető gyorsulás a (4.2) összefüggés alapján a következőképpen írható fel:

$$s = \frac{t_{cs} + t_w}{t_{cp} + t_{syn} + \frac{t_w}{n_p}}, \tag{7.19}$$

ahol a tört számlálójában a soros végrehajtáshoz tartozó idő szerepel, t_{cs} a soros vezérlés funkcióhoz tartozó idő, t_w pedig a teljes számításhoz szükséges idő. A tört nevezőjében a párhuzamos végrehajtáshoz szükséges idő látható: t_{cp} a párhuzamos vezérléshez tartozó idő, t_{syn} a szinkronizációból adódó várakozás időtartama. A vezérlés és a szinkronizáció soros feladatok nem párhuzamosíthatók. A nevezőben n_p a részfeladatokat számító processzormagok száma.

Ideális esetben a számításhoz szükséges idő jelentősen nagyobb, mint ami a vezérléshez és a szinkronizáláshoz szükséges:

$$t_{cs}, t_{cp}, t_{syn} \ll t_w , \tag{7.20}$$

$$s_{max} = \frac{t_w}{\frac{t_w}{n_p}} = n_p . \tag{7.21}$$

A (7.20) feltétel teljesülése esetén az $n_p = 6$ processzormagos DSP multiprocesszort tekintve, az elérhető elméleti maximális gyorsulás a (7.21) összefüggés szerint $s_{max} = 6$.

7.4 Eredmények

A feltevéseim igazolására az egyes transzkóder modellek implementációit különböző processzorokon, többféle bemeneti videojellel vizsgáltam.

7.4.1 Egy processzoros modell

Az egy processzoros transzkóder modellt Texas Instruments TMS320DM6437 *egymagos média processzoron* vizsgáltam, amely 600MHz órajelű C64+ típusú DSP magon alapul. Összehasonlítás céljából további méréseket végeztem a transzkóder egy-egy implementációjával *három általános célú szuperskalár processzoron* is: Intel Core2Duo T7200, AMD Athlon64 3800+ és Intel Pentium 4 HT 2.8GHz típusú processzorokkal, a kétmagos processzorokon csak egy processzormagot kihasználva. Mind a négy processzoron lényegében ugyanaz a transzkóder, megegyező beállításokkal működött. A mérésekhez egy 200 képből álló, televízió adásból rögzített, általános tartalmú, MPEG-2 MP@ML SD videó szekvencia lett felhasználva. Az 5. táblázat összegzi a mérési eredményeket.

	Core2Duo 2GHz	Athlon64 2GHz	P4 HT 2.8GHz	DM6437 600MHz
$\bar{t}_{Ft}$ [µs]	4007	8269	5072	29954
$\bar{f}_{Ft}$ [fps]	249.56	120.93	197.16	33.38
t_{Ftn} [µs]	4007	8269	7100.8	8986.2
f_{Ftn} [fps]	249.56	120.93	140.83	111.28

5. Táblázat: Képsebesség különböző processzorokon [61]

Az 5. táblázat első sorában az átlagos képfeldolgozási idő látható mikroszekundumban mérve, a második sorban ennek a reciproka, a másodpercenként feldolgozott képek száma.

Mivel a vizsgált processzorok közül kettő is 2GHz órajelfrekvencián működött, a harmadik és negyedik sorban 2GHz-es órajelfrekvenciára normalizálva láthatók az adatok, amelyekből messzemenő következtetést ugyan nem lehet levonni, mert a feldolgozási sebesség nem csak a processzor órajelétől függ, de érdekes tájékoztató jellegű összehasonlítást lehet tenni: ugyanaz a C++ program régebbi szuperskalár általános célú processzorokon összehasonlítható sebességgel fut a vizsgált DSP-hez képest, de a modern szuperskalár processzorok azonban jelentősen gyorsabbak.

Előzetes sejtés szerint a 600MHz órajelen működő C64+ DSP számítási teljesítménye MPEG-2 MP@ML SD felbontású videó bitsebesség csökkentéséhez, nyílthurkú transzkóder esetén elegendő lehet. A valósidejű működés (7.1) összefüggés szerinti feltételének ellenőrzéséhez a DM6437 médiaprocesszorhoz tartozó átlagos képfeldolgozási idő, $\bar{t}_{Ft}$, és átlagos képfrekvencia, $\bar{f}_{Ft}$, a 5. táblázatból kiolvasható:

$$\bar{t}_{Ft} = 29954\,\mu s \quad \text{és} \quad \bar{f}_{Ft} = 33.38\,\text{fps} . \tag{7.22}$$

A képfrekvencia értéke európai műsorszóró rendszerekben tipikusan 25 kép másodpercenként, észak-amerikai rendszerekben 30 kép másodpercenként:

$$f_{Fu} = 30\,\text{fps} \;\rightarrow\; t_{Fu} = \frac{1}{f_{Fu}} = 33333.33\,\mu s , \tag{7.23}$$

$$f_{Fe} = 25\,\text{fps} \;\rightarrow\; t_{Fe} = \frac{1}{f_{Fe}} = 40000\,\mu s . \tag{7.24}$$

A médiaprocesszor átlagos terhelése vagy kihasználtsága az észak-amerikai és európai kép ismétlődési időkkel a (7.8) egyenlet alapján a következőképp számítható:

$$\bar{u}_{Fu} = \frac{\bar{t}_{Ft}}{t_{Fu}} = \frac{29954\,\mu s}{33333.33\,\mu s} = 89.86\,\% , \tag{7.25}$$

$$\bar{u}_{Fe} = \frac{\bar{t}_{Ft}}{t_{Fe}} = \frac{29954\,\mu s}{40000\,\mu s} = 74.89\,\% . \tag{7.26}$$

A fenti mérési eredményekből és a számítási eredményekből látható, hogy a transzkóder implementáció a vizsgált DSP-n *valósidejű működésre képes*. A mérés azonban csak tisztán a videó kisebb bitsebességre történő átszámítását vizsgálta, az átszámításhoz szükséges időt mérte. A (7.25) és (7.26) eredményekből az is kiolvasható, hogy a videó bitsebesség csökkentéséhez szükséges számítások valós időben történő végrehajtása jelentős mértékben kihasználja a processzor számítási teljesítményét. Ha a médiaprocesszornak a teljes transzkódoláshoz szükséges többi feladatot is el kell végeznie, azaz a kommunikációt és a multiplexelési feladatokat is, akkor a videó transzkódolásra nem jut elegendő processzoridő a valósidejű végrehajtáshoz. Tehát a vizsgált egymagos DSP-n a teljes transzkódolás *valósidejű végrehajtása nem lehetséges*.

A fenti vizsgálatokból is látható, hogy egy teljes videó transzkóder rendszer megvalósításához egy DM6437 média processzor, azaz *egy 600MHz-en működő C64+ típusú DSP processzormag, számítási teljesítménye nem elegendő*, még az egyik legegyszerűbb, nyílthurkú megoldás esetén sem. A megoldás vagy nagyobb teljesítményű processzor

választása, vagy ami a jelenlegi processzor fejlődési trendhez jobban igazodik, többmagos processzor használata.

7.4.2 Két processzormagos modell

A két processzormagos transzkóder modellt Texas Instruments TMS320C6472 típusú, hat C64+ DSP magot tartalmazó multiprocesszoron vizsgáltam. A vizsgálat célja annak a feltevésnek az igazolása, hogy a transzkódolás hatékonyan felbontható funkcionális szempontból úgy, hogy a videó transzkódolás funkció az egyik processzormagon, a kommunikáció és a multiplex feldolgozás a másik processzormagon működik. Azaz a *rendszer szintű párhuzamosítási* feladat *multimédia elem szinten hatékonyan* megoldható. A vizsgálathoz a multiprocesszor $n_p = 6$ DSP magja közül csak kettő volt kihasználva. A vizsgálat célja volt továbbá bemutatni, hogy ezzel a felosztással a multiprocesszor mind a hat magja kihasználható lehet.

A mérésekhez ugyanazok a teszt videók lettek felhasználva mint az 5.4. és a 6.5. fejezetben, a „Mobile & Calendar”, amely bonyolult részletgazdag háttér előtt, egyszerre több irányba elmozduló szintén részletgazdag elemekből áll. A „Susie” videón egyszínű háttér előtt szinte mozdulatlan, majd egy gyors mozdulatot tevő alak látható. A „Table Tennis” egyszerű háttér előtt mozgó játékosokat mutat, álló majd mozgó kameraállással. A teszt videók 45 másodperc hosszúságúak. A „Mill” reklámokhoz és videoklipekhez hasonlóan rövid, néhány másodperces, jelenetekből és közöttük gyors váltásokból áll, a szekvencia 231 másodperc hosszúságú. A teszt videók mindegyike három különböző bitsebességgel szerepel a vizsgálatban, $R_o = 4$, 6 és 8 Mb/s CBR, a bitsebességtől függetlenül azonos képtartalommal, csak minőségben tértek el egymástól. A „Mill” szekvencia $R_o = 6$ Mb/s CBR bitsebességű. A transzkódolás átlagos cél bitsebessége $R_a = 2$ Mb/s -ra lett beállítva.

	Mobile & Calendar			Susie			Table Tennis			Mill
R_o [Mb/s]	4	6	8	4	6	8	4	6	8	6
$\bar{t}_{Ft}$ [µs]	23941	28726	33241	25336	30343	35132	23649	28706	33350	28862
$\bar{t}_{Ftc}$ [µs]	25393	30821	35919	26833	32429	37805	25127	30834	36080	30926
$\bar{u}_{Ft}$ [%]	59.85	71.81	83.10	63.34	75.86	87.83	59.12	71.76	83.38	72.16
$\bar{u}_{Ftc}$ [%]	63.48	77.05	89.80	67.08	81.07	94.51	62.81	77.08	90.20	77.31

6. Táblázat: DSP 1, Processzorterhelés és feldolgozási idők

A transzkódolást végző DSP magon (DSP 1) a különböző videókhoz tartozó mérési eredmények összefoglalva a 6. táblázatban láthatók. A táblázat első sorában a videók bemeneti bitsebessége, R_o, látható. A második sorban a képek transzkódolásához szükséges idők átlaga, $\bar{t}_{Ft}$, a harmadik sorban a teljes feldolgozás átlagos ideje, $\bar{t}_{Ftc}$, van mikroszekundumban. A következő sorokban az előző átlagos idők alapján, a (7.24) és a (7.26) összefüggés szerint, számított processzor terhelések láthatók. Az ötödik sorban a képek transzkódolásához tartozó terhelés, $\bar{u}_{Ft}$, a hatodikban a teljes terhelés, $\bar{u}_{Ftc}$, látható.

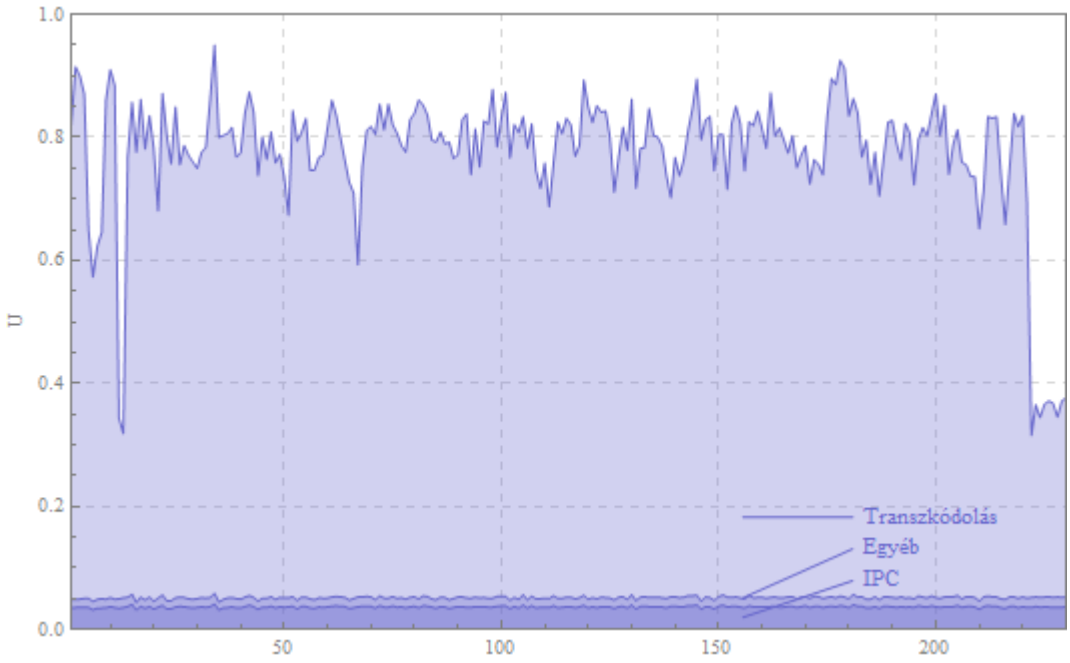
Megfigyelhető, hogy a DSP *mag terhelése a vártnak megfelelően nagy és a teljes terhelés legnagyobb részét a transzkódolási művelet* végrehajtása teszi ki. Látható továbbá, hogy a végrehajtási idők és a terhelési adatok függnek a szekvenciától, azaz a képtartalomtól és az egyes szekvenciák bitsebességétől.

	Mobile & Calendar			Susie			Table Tennis			Mill
R_o [Mb/s]	4	6	8	4	6	8	4	6	8	6
$\bar{t}_{F0}$ [µs]	2556	3580	4628	2418	3369	4403	2444	3449	4508	3726
$\bar{u}_{F0}$ [%]	6.39	8.95	11.57	6.04	8.42	11.01	6.11	8.62	11.27	9.31

7. Táblázat: DSP 0, Processzorterhelés és feldolgozási idők

A 7. táblázatban a kommunikációt és a multiplex feldolgozást végző DSP (DSP 0) magon a különböző videókhoz tartozó mérési adatok láthatók. A táblázat első sorában ismét a videók bemeneti bitsebessége, R_o, látható, a második sorban egy videó multiplex teljes feldolgozásának átlagos ideje, $\bar{t}_{F0}$, a harmadik sorban a 6. táblázathoz hasonlóan, az előző átlagos idő alapján és a (7.24) és a (7.26) összefüggés szerint számított teljes processzorterhelés, $\bar{u}_{F0}$, látható.

Látható, hogy a multiplex feldolgozás és a kommunikáció nem terheli meg jelentősen a DSP magot. Ha az $n_p = 6$ processzormagos esetet tekintjük, $n_p - 1 = 5$ transzkódoló DSP maggal és egy kommunikációs és multiplex feldolgozó processzormaggal, minden vizsgált esetben teljesül a (7.18) feltétel, azaz $u_{F0} \leqslant 20\,\%$.

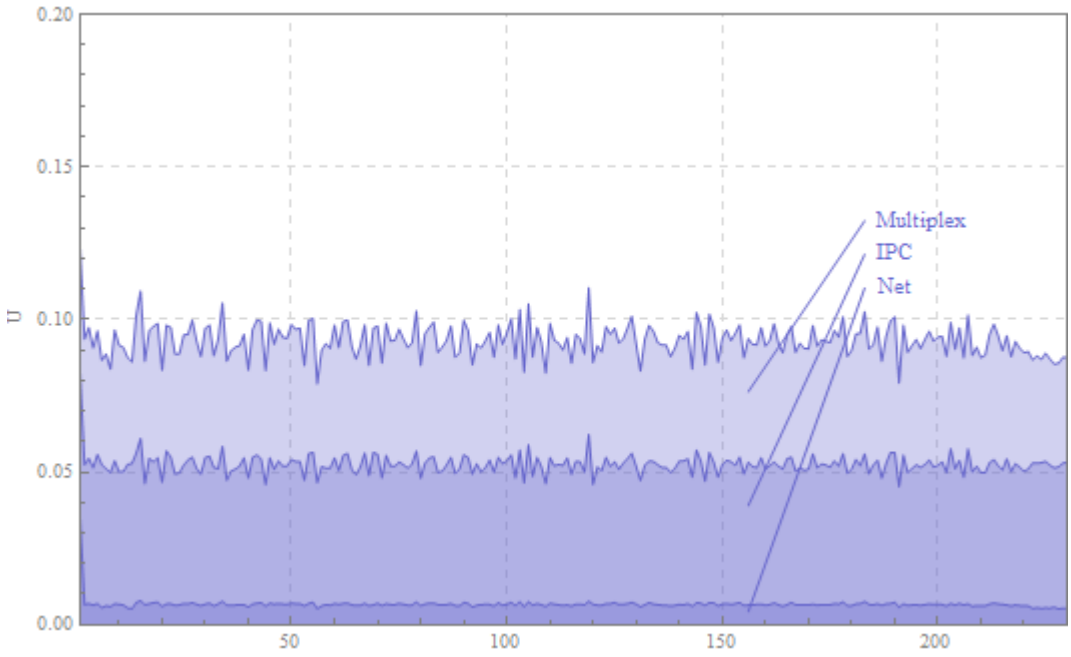


7.5. Ábra: DSP 1 processzorterhelés Mill szekvenciával

A 7.5. ábrán a transzkódolást végző DSP (DSP 1) terhelése látható az idő függvényében, a „Mill” szekvencia transzkódolása folyamán. A vízszintes tengelyen az eltelt idő, *t*, másodpercben, a függőleges tengelyen a processzorterhelés, *u*, látható. A görbe a teljes terhelést, u_{Ftc}, mutatja az idő függvényében, és megfigyelhető, hogy nem csak az átlagos terhelés, de a *pillanatnyi processzor terhelés is jelentősen függ az aktuális képtartalomtól*. A teljes terhelés az idő nagy részében $u_{Ftc} \approx 80\,\%$ körül mozog, de van, hogy a 90%-ot is meghaladja, $u_{Ftc} > 90\,\%$. A szekvencia végén azonban a terhelés 40% alá esik, $u_{Ftc} < 40\,\%$,

az egyszerű képtartalom miatt. A világos árnyalattal színezett görbe alatti terület, tisztán a képek transzkódolásához szükséges processzorterhelést, u_{Ft}, mutatja, ez a terhelés legnagyobb része. Kicsit sötétebb árnyalattal a processzorterhelés egyéb szükséges számításokhoz tartozó része látható. A legsötétebb árnyalatú görbe alatti terület a processzormagok közötti kommunikáció által létrehozott terhelés. A processzormagok közötti kommunikációhoz és az egyéb szükséges számításokhoz tartozó terhelés kicsi, és ellentétben a transzkódolással nem függ a képtartalomtól. Ennek oka, hogy ezek a számítások a bitsebességtől függenek, a bemeneti videó pedig a képtartalomtól függetlenül állandó bitsebességű.

A 7.6. ábrán szintén a „Mill” szekvenciához tartozó terhelés látható, de most a multiplex feldolgozást és kommunikációt végző DSP (DSP 0) terhelése az idő függvényében. A vízszintes tengelyen az eltelt idő, t, másodpercben, a függőleges tengelyen a processzorterhelés, u, látható. A terhelés $u = 0\ldots20\,\%$ tartományban van ábrázolva, a (7.18) feltétel szemléltetése miatt.

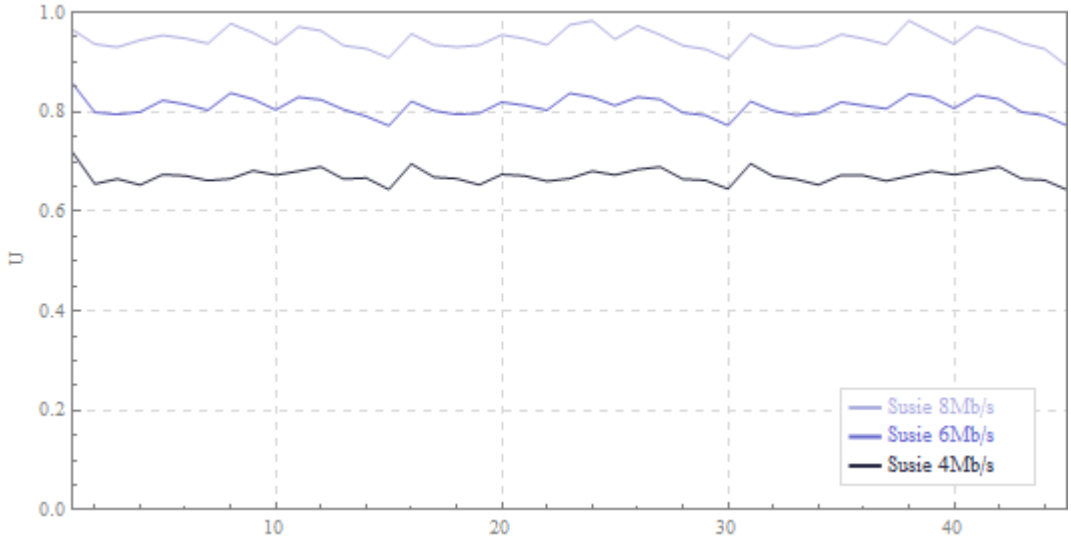


7.6. Ábra: DSP 0 processzorterhelés Mill szekvenciával

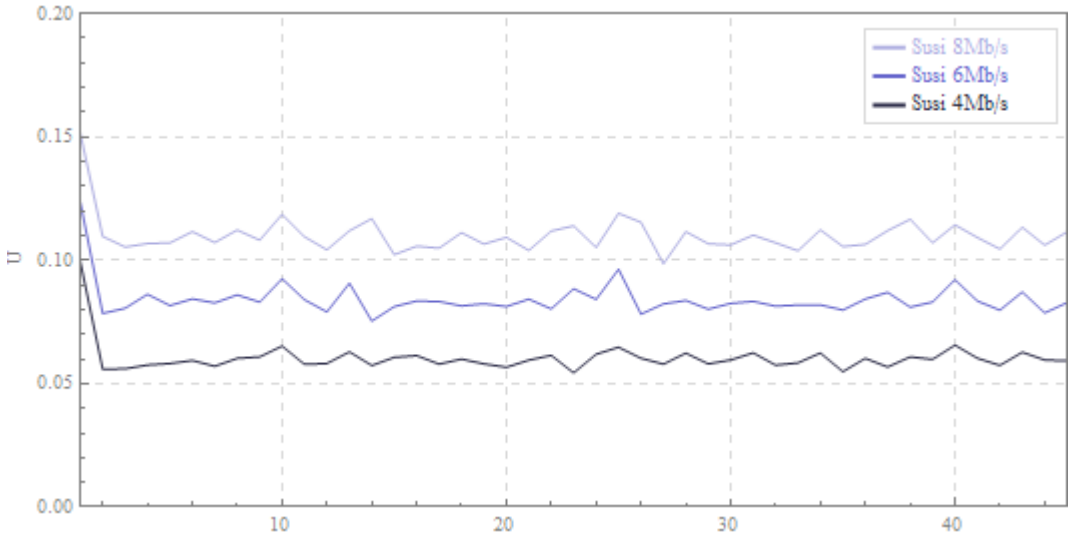
A görbe a teljes terhelést, u_{F0}, mutatja az idő függvényében, mivel a videó állandó bitsebességű, a multiplex feldolgozás és a kommunikáció is közel állandó terhelést hoz létre. A teljes terhelés az egész szekvencia folyamán bőven $u_{F0} \leqslant 20\,\%$ alatt marad, tehát nem csak átlagosan teljesül a (7.18) feltétel. A világos árnyalattal színezett görbe alatti terület a multiplex feldolgozáshoz, a videónak a multiplexből való kiemeléséhez és a transzkódolás utáni újra multiplexeléshez, szükséges processzorterhelés. A sötétebb árnyalat a processzorok közötti kommunikációhoz tartozó terhelés. A legsötétebb árnyalatú görbe alatti terület az UDP/IP/Ethernet hálózaton érkező és küldött adatcsomagok kezeléséhez tartozó terhelés.

A 7.7. ábrán a transzkódolást végző DSP (DSP 1) processzor teljes terhelése, t_{Ftc}, látható az idő függvényében, a három különböző bitsebességű „Susie” szekvenciához. Hasonlóan a korábbi grafikonokhoz a vízszintes tengelyen az eltelt idő, t, másodpercben, a függőleges tengelyen a processzorterhelés, u, látható. A legnagyobb terhelést a világos árnyalatú grafikon mutatja, amely az $R_o = 8\,\mathrm{Mb/s}$ bemeneti bitsebességű szekvenciához tartozik. Világosabb

árnyalattal az $R_o = 6\,\text{Mb/s}$ bemeneti bitsebességű szekvenciához tartozó alacsonyabb terhelés látható. A legsötétebb árnyalatú görbe, amely a legkisebb processzorterhelést mutatja, az $R_o = 4\,\text{Mb/s}$ bemeneti bitsebességű „Susie” szekvencia transzkódolásához tartozik. Megfigyelhető, hogy a *transzkódolás által létrehozott processzorterhelés jelentősen függ a videó bemeneti bitsebességétől, azonos képtartalom mellett*. A görbék ingadozása pedig a DSP terhelés képtartalom függését mutatja.



7.7. Ábra: DSP 1 processzorterhelés Susie szekvenciákkal



7.8. Ábra: DSP 0 processzorterhelés Susie szekvenciákkal

A 7.8. ábrán látható görbék a multiplex feldolgozást és kommunikációt végző DSP (DSP 0) terhelését mutatják az idő függvényében, a három különböző bitsebességű „Susie” szekvenciához. A vízszintes tengelyen az eltelt idő, *t*, másodpercben, a függőleges tengelyen a processzorterhelés, *u*, látható. A terhelés a 7.6. ábrához hasonlóan $u = 0\ldots20\,\%$ tartományban van ábrázolva. A görbék megfelelnek a 7.7. ábrán lévő görbéknek, a világos

árnyalatú tartozik a legnagyobb bitsebességű szekvenciához, a sötétebb árnyalat a középső bitsebességhez, a legsötétebb árnyalat pedig a legkisebb bitsebességű „Susie” szekvenciához tartozik. Megfigyelhető ezeken a görbéken is a *multiplex feldolgozás és kommunikáció bitsebesség függése*, adott képtartalom mellett, valamint a transzkódoláshoz képest kisebb összefüggés a képtartalom és a DSP terhelése között.

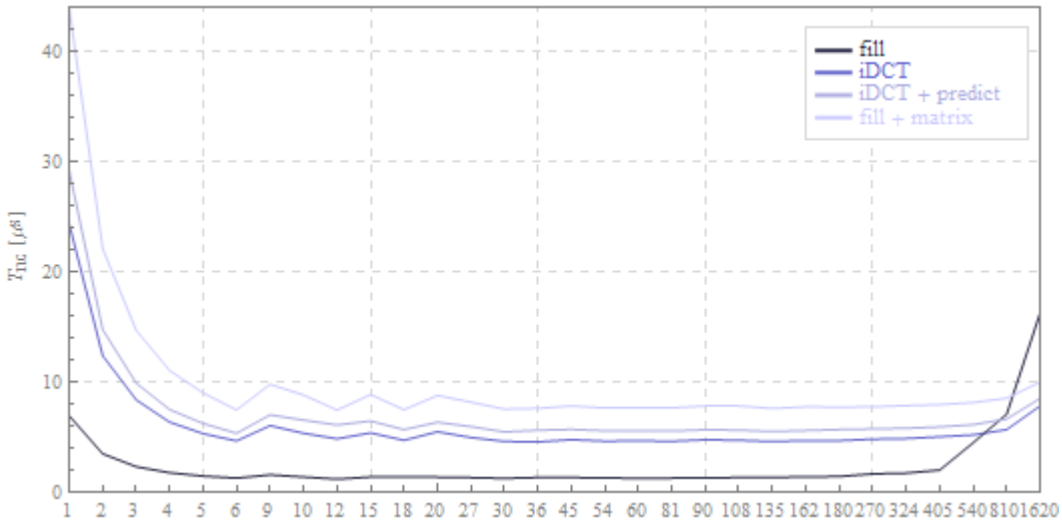
7.4.3 Több processzormagos modell

A több processzormagos transzkóder modellt Texas Instruments TMS320C6472 típusú, hat C64+ DSP magot tartalmazó multiprocesszoron vizsgáltam. A vizsgálat célja annak a feltevésnek az ellenőrzése, hogy többmagos processzoron megvalósított nyílthurkú transzkóderben hatékonyan kihasználható a térbeli párhuzamosítás, mivel egyrészt a videó adatok feldolgozásakor nincsenek függőségek a képadatokon belül, másrészt a többmagos DSP használata esetén, a szorosan csatolt processzormagok miatt, a DSP magok között gyors a kommunikáció.

A vizsgálat SD (720×576 pixel) felbontású mesterségesen előállított teszt képek felhasználásával történt. A térbeli felbontás alapja a makroblokk, amely 16×16 pixel világosságjelet és két 8×8 pixel színkülönbségjelet tartalmaz 4:2:0 színábrázolás esetén. Egy SD kép $n_{mb} = 1620$ makroblokkból áll. A térbeli felbontás során a képek, n_j, párhuzamosan feldolgozandó darabra lettek felosztva, minden feldolgozandó egység egyenlő és egész számú makroblokkot tartalmaz:

$$n_j = 1 \ldots n_{mb}, \quad \text{ha} \quad n_j^i \mid n_{mb} \,. \tag{7.27}$$

Valódi videó transzkódolása esetén az egyes képek átkódolásához szükséges számítási idő jelentősen függ a képek tartalmától, a párhuzamosítás hatékonysága pedig az egyes párhuzamos feladatokhoz tartozó számítási idő és kommunikáció arányától.



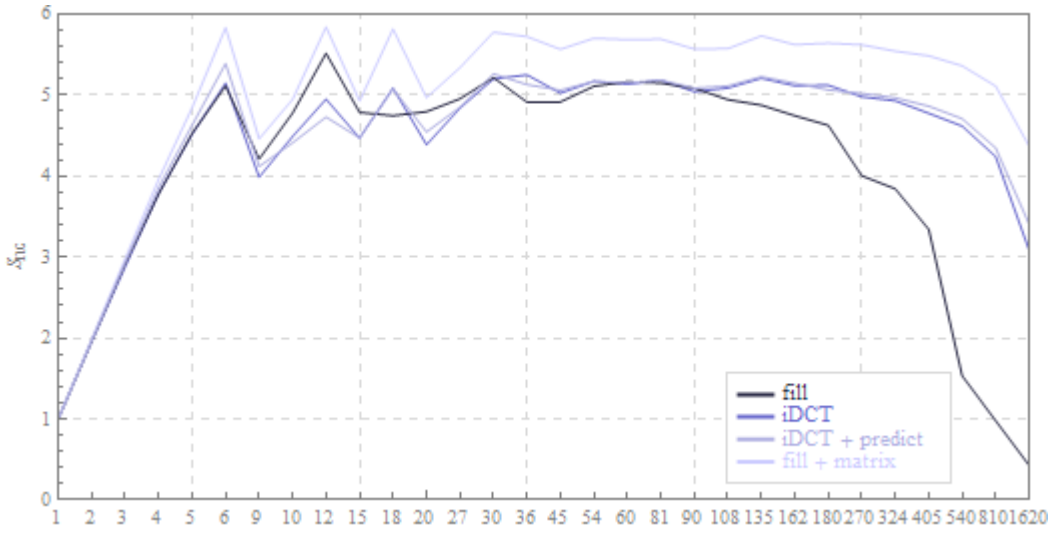
7.9. Ábra: Több processzormagos végrehajtási idő

Ebben a vizsgálatban a képtartalom állandó ezért több különböző számítási komplexitású feladat végrehajtása modellezi a változó képtartalmat. A négy különböző számítási komplexitású teszt feladat vizsgálata külön-külön történt. A legkisebb komplexitású teszt feladat a „fill”, egy egyszerű mintával tölti fel a képeket, az „iDCT” inverz DCT

transzformációt hajt végre az egyes makroblokkokon, az „iDCT + predict” inverz DCT mellet egy referencia kép használatával predikciót hajt végre, így a több képhez egyszerre történő párhuzamos hozzáférést szimulálja, a legnagyobb számítási komplexitású a „fill + matrix”, a képek feltöltése mellett egy bonyolultabb mátrixműveletet hajt végre a kép makroblokkjain.

A 7.9.ábrán látható grafikon a teszt feladatok végrehajtási idejét ábrázolja a térbeli pérhuzamosítás mértékének függvényében. A grafikon függőleges tengelyén a teljes kép feldolgozásához szükséges idő, t_{nc}, a vízszintes tengelyen a térbeli felosztás mértéke látható, n_j, azaz a feldolgozási egységek száma képenként. A grafikonon jól látható az egyes teszt feladatok közötti futási idő különbség.

A grafikon négy szakaszra osztható: ameddig a párhuzamosítás mértéke nem éri el a végrehajtó egységek számát, a számítási idő monoton csökken a párhuzamosítás mértékének növelésével, ennek oka, hogy nem minden processzorra jut feladat, így kevesebb végrehajtó egység végzi el a teljes munkát. A párhuzamosítás mértékének növelésével ezután egy ingadozás figyelhető meg a végrehajtási időben, melynek magyarázata a processzormagok nem egyenletes kihasználtsága. A számítási idő minimumok azokon a pontokon vannak, ahol a párhuzamosítás mértéke a DSP magok számának egész számú többszöröse, tehát egyenletes a végrehajtó egységek kihasználtsága. Az egyre finomabb felbontású párhuzamosítás miatt, a feladatok nagyobb számával, ez az ingadozás kisimul és egy egyenletes szakasz következik. A párhuzamosítás mértékének további növelésével a szükséges kommunikáció mennyisége és ezzel az ehhez szükséges idő is növekszik. Mikor a párhuzamosítás mértéke eléri azt a szintet, hogy az egyes részfeladatok végrehajtási ideje összemérhető a kommunikációhoz és a szinkronizációhoz szükséges idővel, a teljes kép feldolgozásához szükséges idő ismét növekedni kezd. Látható, hogy ez a növekedés először a legkisebb számítási komplexitású, majd sorban a többi feladatot is érinti.



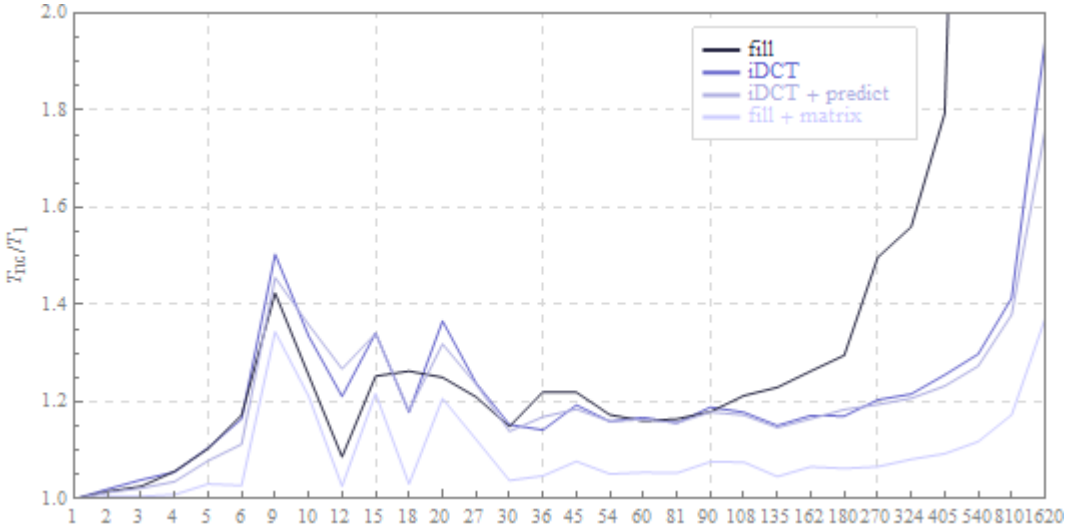
7.10. Ábra: Gyorsulás mértéke több processzormagon

A több processzormaggal végzett számítás által elért gyorsulás, s_{nc}, a (4.2) és (7.19) összefüggés alapján, a mérési adatokból a következőképpen számítható:

$$s_{nc} = \frac{t_1}{t_{nc}} . \tag{7.28}$$

A 7.10. ábrán látható grafikonon a több processzormaggal végzett számítás által elért gyorsulás látható a térbeli párhuzamosítás mértékének függvényében. A grafikon függőleges tengelyén az elért gyorsulás, a vízszintes tengelyen a térbeli felosztás mértéke látható. Megfigyelhető, hogy általában a nagyobb komplexitású számítási feladatokkal nagyobb gyorsulás érhető el.

A 7.10. ábrán látható grafikon is négy szakaszra osztható: ameddig a párhuzamosítás mértéke kisebb vagy egyenlő a processzormagok számával, az elért gyorsulás közel lineárisan növekszik, minél nagyobb a feladat számítási komplexitása, annál jobban közelíti a lineáris növekedést. A következő szakaszban, a számítási idő grafikonon is megfigyelhető ingadozás látható, de a gyorsulás nagyobb mértékű ingadozást mutat mint a számítási idő. A gyorsulás maximumok azokon a pontokon vannak, ahol a párhuzamosítás mértéke a DSP magok számának egész számú többszöröse, tehát egyenletes a végrehajtó egységek kihasználtsága. A harmadik az egyenletes szakasz, ahol az egyre finomabb felbontású párhuzamosítás miatt, a feladatok nagyobb számával, ez az ingadozás kisimul. A negyedik szakaszban, mikor a párhuzamosítás mértéke eléri azt a szintet, hogy az egyes részfeladatok végrehajtási ideje összemérhető a kommunikációhoz és a szinkronizációhoz szükséges idővel, a gyorsulás mértéke csökkenni kezd, szintén először a legkisebb számítási komplexitású, majd sorban a többi feladat is.



7.11. Ábra: Veszteség többmagos processzoron

A 7.11. ábrán az egy processzormaggal végzett feladat megoldáshoz képest, a számítási idő többlet látható. Ez a veszteség, o_{nc}, a teljes kép feldolgozása alatt az egyes processzormagok által számítással töltött idők összege, és az egy processzormaggal végzett számítás idejének aránya, ami a következő összefüggéssel számítható:

$$o_{nc} = \frac{t_{nc} n_{jc}}{t_1}, \tag{7.29}$$

ahol, t_{nc}, a teljes kép feldolgozásához szükséges idő több processzormagos számítás esetén, t_1, a teljes kép feldolgozásához szükséges idő egy processzormagos számítás esetén, n_{jc}, pedig a következő összefüggéssel határozható meg:

$$n_{jc} = \begin{cases} n_j & \text{ha } n_j < n_c \\ n_c & \text{egyébként} \end{cases}, \tag{7.30}$$

tehát a számítást végző processzormagok száma, n_{jc}, a párhuzamosítás mértékével, n_j, egyezik meg, ameddig az kisebb a DSP magok számánál, majd a számítást végző feldolgozó egységek száma megegyezik processzormagok számával, n_c.

A 7.11. ábrán látható grafikon függőleges tengelyén a számítási idő többlet, a vízszintes tengelyen a térbeli felosztás mértéke látható. Ezen az ábrán is megfigyelhető az előző grafikonok négy szakasza. Az első szakasz monoton növekszik, amíg a párhuzamosítás mértéke eléri a processzormagok számát. A második szakaszban a veszteség jelentősen ingadozik, a vártnak megfelelően az ingadozás minimumai a DSP magok számának egész számú többszörösénél vannak. A harmadik rész egy rövid egyenletes szakasz, majd a negyedik részben a veszteség meredeken növekszik, szintén a számítási komplexitással fordított arányban.

A mérési adatokból az a következtetés vonható le, hogy a párhuzamosítás hatékonysága szempontjából a grafikonok első szakaszának eleje, azaz *kevés processzormag használata előnyös, ez van kihasználva a két processzormagos transzkóder esetén*, ám ekkor a gyorsulás is kisebb mértékű. A második szakaszban, azaz viszonylag *kevés párhuzamos feladat esetén*, akkor hatékony a párhuzamosítás, ha a feladatok száma a *feldolgozó processzorok számának egész számú többszöröse* lehet. A harmadik, *egyenletes szakasz jól kihasználható, a szelet szintű párhuzamosítás* erre a szakaszra esik. A negyedik szakaszban azonban már akkorák a párhuzamosítás veszteségei, hogy ebben a formában *nem érdemes kihasználni, erre a szakaszra esne a makroblokk szintű párhuzamosítás*.

7.5 Tézis

A videó transzkódolás *rugalmasabban méretezhető a számítási komplexitás és a képminőség tekintetében*, mint egy videó kódoló vagy dekódoló eljárás. Több processzormagos DSP multiprocesszoron, különböző bonyolultságú transzkódoló megoldások párhuzamosítási lehetőségeinek vizsgálatához, kidolgoztam három transzkóder modellt:

- Bemutattam, hogy a kiindulási feltételeknek megfelelő videó transzkódolása egy processzormagos DSP-n közel valósidőben elvégezhető. Kidolgoztam egy, egy processzormagos teljes videó transzkóder modellt, és *különböző processzor architektúrákon* vizsgáltam a *valósidejű feldolgozás feltételeit*.
- Bemutattam, hogy *többmagos DSP multiprocesszoron* a particionálási probléma nem csak a videó átszámító algoritmus párhuzamosításaként fogható fel, hanem tekinthető *rendszer szintű párhuzamosítási feladatként* is, és a *particionálás multimédia elem szinten* hatékonyan kialakítható, ehhez kidolgoztam egy két processzormagon működő teljes videó transzkóder modellt.
- *Többmagos DSP multiprocesszoron* vizsgáltam a videó transzkódoló algoritmus *térbeli párhuzamosításának hatékonyságát a párhuzamosítás mértékének függvényében,* amihez kidolgoztam egy több processzormagos videó transzkóder modellt.

7.5.1 Új tudományos eredmények

Az irodalomban található párhuzamosítási megoldások a videó kódolás és dekódolás problémáival foglalkoznak, és nagyon kevés cikk foglalkozik speciálisan transzkódolás párhuzamosítási megoldásokkal.

- *Homogén nyílthurkú* videó transzkódolás *valósidejű* vizsgálata *egy processzormagos DSP-n* az irodalomban nincs publikálva.
- Az irodalomban található párhuzamosítási megoldások nem teljes rendszert vizsgálnak, hanem csak az aktuális algoritmusok párhuzamosítási lehetőségeit. Az itt bemutatott párhuzamosítási megoldásban az *újdonság a teljes transzkódoló rendszer vizsgálata* a feladat particionálásához, valamint az ebből adódó *multimédia elem szintű particionálás*.
- Újdonság továbbá *többmagos homogén DSP multiprocesszor* használata videó transzkódoló particionáláshoz. Videó kódolás vagy dekódolás párhuzamosítását vizsgálták hasonló architektúrákon, mint az IBM Cell BE processzor, vagy sokmagos DSP szimulátor. A többmagos DSP-hez legközelebb álló architektúra amin transzkódolást vizsgálták több különálló DSP-ből felépített rendszer volt.

7.5.2 Tézis igazolása

A különböző párhuzamosítási lehetőségek vizsgálatára három transzkóder modellt fejlesztettem ki és valósítottam meg. Az egy processzoron működő teljes transzkóder, a két processzormagon működő teljes transzkóder és a több processzormag lehetőségeinek vizsgálatára egy több magot is kihasználó transzkóder modellt. A transzkóderek vizsgálata MPEG-2 MP@ML teszt videókkal és valódi televízió adásból származó bemeneti videókkal történt, hat DSP magot tartalmazó multiprocesszoron.

Az egy processzormagon és a két processzormagon működő teljes videó transzkóder modellek esetén, a *vártnak megfelelően,* a transzkódolás átlagos processzorterhelése nagy (62.81% és 94.51% között) és a bemeneti videó bitsebességétől és a képtartalomtól függ, a

kommunikáció processzorterhelése pedig alacsony (6.04% és 11.57% között) és a bitsebességtől függ.

A több processzormagos videó transzkóder modell esetén, a térbeli párhuzamosítás által elérhető gyorsulás akkor a legnagyobb, ha a párhuzamosítás mértéke nagyobb a DSP magok számánál és a terhelés egyenletesen oszlik el a processzormagok között. A párhuzamosítás mértékének további növelésével arányosan növekszik a kommunikációhoz szükséges idő, amely csökkenti a számításra felhasználható processzorteljesítményt.

7.5.3 A következtetés korlátai

Az vizsgálatokat egy konkrét többmagos DSP multiprocesszorral végeztem, kihasználva az architektúra adottságait. Az eredmények függenek a processzormagok számítási teljesítményétől és a DSP magok közötti kommunikációs lehetőségektől. A vizsgálttól eltérő felépítésű multiprocesszor esetében, a fenti eredmények felhasználásakor, figyelembe kell venni az architektúrák közti különbségeket.

7.5.4 Következmények

A két DSP magot kihasználó transzkódoló rendszerrel és a többmagos homogén DSP SoC kialakítású integrált áramkörök megjelenésével lehetővé vált egy *multiprocesszoron belül több transzkóder megvalósítása*. Ez leegyszerűsítheti statisztikus remultiplexerek fejlesztését.

A két DSP magot kihasználó transzkóder modell akkor hatékony, ha a videó feldolgozás közel *teljesen kihasználja egy DSP mag számítási teljesítményét,* gyorsabb processzormagok esetén *lehetőség nyílik a transzkódoló algoritmus továbbfejlesztésére*, továbbra is teljesen kihasználva a gyorsabb processzormag számítási teljesítményét.

A több processzormagot kihasználó transzkódolási módszerekkel, a megnövekedett számítási teljesítménnyel arányosan, lehetőség nyílik *nagyobb felbontású videó (HD), és nagyobb számítási komplexitású,* jobb képminőséget elérő (zárthurkú) transzkódoló algoritmusok kialakítása.

7.5.5 Kapcsolódó publikációk

Bence Formanek, Tihamér Ádám, *"DSP implementation of an MPEG-2 video bitrate transcoder"*, Proc. of the 11th International Carpathian Control Conference, ISBN 978-963-06-9289-2, 2010, pp. 195-198.

Formanek Bence, Czap László, *"Videó bitsebesség csökkentés többmagos DSP-n"*, XXVII. microCAD Nemzetközi Tudományos Konferencia Kiadványa, 2013.

8 Összefoglalás, a továbbfejlesztés lehetőségei

8.1 Adaptív, lineáris bitsebesség – kvantáló modell

A bitsebesség vezérlés feladata a kívánt videó bitsebesség tartása a lehető legjobb képminőség elérésével. A műsorszórásban is használt MPEG videó kódolásokban a bitsebesség a kvantálás mértékének változtatásával szabályozható. A kvantálás az a kódolási lépés ahol a tulajdonképpeni adatvesztés megtörténik, lényege az egyes transzformációs együtthatók pontatlanabb, azaz kevesebb bittel való ábrázolása, ami azonban a képminőség romlásával jár. A bitsebesség és a kvantáló közötti kapcsolatot a bitsebesség-kvantáló függvény *R(q)* határozza meg. Amennyiben ismerjük a *R(q)* függvényt az elérendő cél bitsebességhez tartozó kvantáló meghatározható. A bitsebesség-kvantáló függvény meghatározása, amely a bitsebesség vezérlő kialakításának kulcskérdése, a videó kódolás összetettsége miatt azonban nem triviális feladat.

Az adaptív lineáris bitsebességvezérlő modell kialakításához kiindulási alapként a kép komplexitáson alapuló bitsebességvezérlési módszert választottam. A nyílthurkú transzkóderben azonban nem állnak rendelkezésre a dekódolt képek, így a térbeli komplexitás meghatározása sem lehetséges a kódolás során alkalmazott módszerrel. Egy transzkóderben viszont a bemeneti videó bitfolyam paraméterei vizsgálhatók. Feltételeztem majd bizonyítottam, hogy a bemeneti és a kimeneti bitsebességek aránya lineárisan közelíthető a bemeneti és a kimeneti kvantálók értékének arányával, amelyből számítható a kimeneti kvantáló értéke. A lineáris összefüggés paramétereit különböző videó szekvenciák vizsgálatával határoztam meg.

A lineáris közelítés azonban nem veszi figyelembe a bitsebesség-kvantáló függvény képtartalomtól való függését. A helyes bitsebesség fenntartása érdekében ezért vizsgálom a tényleges kimeneti bitsebességet. A meghatározott értéktől való eltérést, mint hiba jelet használva, egy visszacsatolást hoztam létre, amely így adaptívan befolyásolja az új kvantáló értékének meghatározását.

Az MPEG videó kódolásoknál a különböző képtípusok mérete különböző, ezért a bitsebesség meghatározásának legkisebb egysége a képcsoport. Egy nyílthurkú transzkóder nem befolyásolja a képcsoportok méretét, ezért képcsoport független bitsebesség vezérlést alakítottam ki. A bemeneti és a kimeneti bitsebesség meghatározásához is a képcsoportnál jelentősen nagyobb számú képet használtam, és vizsgáltam a képek számának a bitsebességvezérlésre gyakorolt hatását.

8.1.1 Új tudományos eredmények

Tézis: Kidolgoztam egy, a nyílthurkú transzkóder komplexitásához illeszkedően, *kis számításigényű, lineáris bitsebesség-kvantáló R(q) modellen alapuló, GOP struktúra független, visszacsatolt bitsebesség vezérlő* módszert, mely a bemeneti videó kódolási paraméterei alapján határozza meg a kimeneti videó paramétereit. Bemutattam, hogy a kimenet VBR kódolása mellett megfelelően tartja a beállított átlagos bitsebességet, valamint vizsgáltam az elért képminőséget is.

A visszacsatolt, lineáris *R(q)* modellen alapuló bitsebességvezérlés használata nyílthurkú transzkóderben, a bemeneti és kimeneti videók paramétereinek GOP struktúra független vizsgálata mellett, új tudományos eredmény, az irodalomban nincs publikálva.

A fenti elképzelések alapján megvalósított transzkódolót teszt videókkal és valódi televízió adásból származó bemeneti videókkal vizsgáltam. A kimeneti bitsebesség és a képminőség

átlagos értékét és időfüggését megfigyelve, a legtöbb esetben a *vártnak megfelelő eredményeket* kaptam.

Az eredmény következménye, hogy az alacsony számításigényű bitsebességvezérlés egy kis komplexitású videó transzkóder részeként lehetővé teszi valósidejű videó bitsebesség csökkentés megvalósítását többmagos homogén DSP egy processzormagján, így több videó bitsebességcsökkentését egy multiprocesszor SoC egységgel.

8.2 DSP magok közötti kommunikáció

Hatékony processzormagok közötti kommunikáció kialakításához a többmagos DSP lehetőségeit minél jobban kihasználó megoldásra van szükség. A DSP multiprocesszorok felépítésüket tekintve, fizikai szempontból nem cache koherens osztott memóriás rendszernek foghatók fel, logikai szempontból azonban elosztott rendszernek tekinthetők. A logikai felépítés az üzenetküldéses kommunikáció valamilyen formájának megvalósítása felé mutat, míg a DSP fizikai felépítése a megosztott memóriás kommunikáció megvalósítását tenné lehetővé. A valósidejű működés további feltételeket támaszt a kommunikációval szemben: a nagy mennyiségű adat átvitele miatt a memória másolások számát minimalizálni kell, a dinamikus memóriafoglalásokhoz szükséges idő véletlenszerű és felülről nem korlátos, ezért valósidejű rendszerekben nem használható és a szinkronizációból adódó blokkolást el kell kerülni.

A fenti feltételeknek megfelelően, kialakítottam egy blokkolás és memória másolás mentes, üzenetküldés és osztott memória hibrid módszert, tripla gyűrűs buffer segítségével megvalósítva. Az adatátvitelhez szükséges memóriát egy előre lefoglalt megosztott címterületen helyeztem el, kialakítva ezzel az elsődleges gyűrűs buffert. A gyűrűs kialakításra azért van lehetőség mert a multimédia adatforgalom bitfolyam jellegű, az adatokat az érkezés sorrendjében kell feldolgozni. Az egyes processzormagokon további gyűrűs buffereket alakítottam ki, ezek az üzenetküldéses kommunikáció segítségével küldött, az elsődleges gyűrűs bufferben lévő adatokra hivatkozó mutatókat tartalmazzák. A mutatók átvitele a DSP magok közötti megszakítással történik, a tényleges adatok a mutató által a közös memóriában érhetők el. A mutatókat tartalmazó gyűrűs bufferek segítségével elkerülhető a szinkronizáció okozta blokkolás lehetősége.

Meghatároztam a kommunikációhoz szükséges idő és az okozott processzorterhelés számításának módszerét, vizsgáltam a kommunikáció erőforrásigényét.

A transzkóder modelleket és a processzormagok közötti kommunikációt Texas Instruments TMS320C6472 típusú, hat C64+ DSP magot tartalmazó multiprocesszoron vizsgáltam. A vizsgálathoz több teszt videót és valódi televízió adásból származó műsorrészletet is felhasználtam. A fenti bitsebességvezérlést az egy DSP magon, valamint a két processzormagon kialakított transzkóder modellek részeként is vizsgáltam. A modelleket C++ programozási nyelven valósítottam meg, ami lehetővé tette több processzor típus felhasználását is az egy processzormagos transzkóder modell elemzésére.

8.2.1 Új tudományos eredmények

Tézis: *Többmagos DSP multiprocesszoron* kialakított párhuzamos videó transzkóderhez kidolgoztam egy processzormagok közötti, az architektúrához igazodó, *logikai szempontból üzenetküldéses* és *fizikai szempontból osztott memóriás hibrid* kommunikációs megoldást, amely *blokkolás és memória másolás mentes*, hatékony kommunikációt tesz lehetővé.

A többmagos homogén DSP fizikai és logikai felépítéséhez és a valósidejű operációs rendszerhez igazodó *üzenetküldéses és osztott memória hibrid* processzorokmagok közötti kommunikáció kialakítás*,* amely *blokkolás és memória másolás mentes* újdonság. Az

irodalomban megosztott memóriás kommunikáció és az üzenet küldéses kommunikáció különböző formáival találkoztam. Hibrid kommunikációs megoldást azonban csak elosztott rendszerként kialakított közös memóriás processzorok között használtak, de ott is a párhuzamosítás két különböző hierarchia szintjén.

A processzormagok közötti kommunikációs megoldás igazolásához a két processzormagon működő teljes transzkóder modellt használtam. A vizsgálat MPEG-2 MP@ML teszt videókkal és valódi televízió adásból származó bemeneti videókkal történt. Az eredmények a *vártnak megfelelően* alakultak.

Az új eredmények alapján kialakított gyors és hatékony processzormagok közötti kommunikáció segítségével az egyes DSP magok számítási kapacitása jobban kihasználható a videó feldolgozásra, ami nagyobb bitsebességű és nagyobb felbontású videók transzkódolását teszi lehetővé.

8.3 Párhuzamosítási lehetőségek többmagos DSP-n

A nyílthurkú transzkóder modellek megvalósítására rugalmassága, energiafogyasztása és számítási teljesítménye miatt a többmagos DSP-t választottam. A különböző párhuzamosítási lehetőségek vizsgálatára három modellt használtam: az egy processzoron működő teljes transzkódert, a két processzormagon működő teljes transzkódert és a több processzormag lehetőségeinek vizsgálatára egy több magot is kihasználó transzkóder modellt.

A teljes transzkóder modell moduláris, multimédia elemekből álló rendszer, fő egységei: maga a videó átszámító elem, az IP hálózat kezelő bemeneti és kimeneti modulok, a demultiplexer és a multiplexer elemek. Egy processzormagos kialakításban minden elem ugyanazon a processzormagon működik.

A két processzormagos párhuzamos transzkóder modell funkcionális párhuzamosításnak tekinthető, ahol a videó transzkódolás funkció az egyik processzormagon (DSP 1), a kommunikáció és a multiplex feldolgozás a másik processzormagon (DSP 0) működik. Több, n_p, processzormagot tekintve ez a funkcionális felosztás, egy olyan elrendezés vizsgálatát teszi lehetővé, amely egyszerre több műsor, több multiplex feldolgozására alkalmas. Egy kivételével a processzormagok, n_p-1, a videó transzkódolást végzik, egy dedikált processzormag pedig az összes feldolgozandó multiplexhez tartozó többi feladatot ellátja. Terhelés elosztás szempontjából ez a funkcionális felosztás akkor hatékony, ha a videó transzkóderek maximálisan kihasználják egy-egy processzormag számítási teljesítményét, és az összes műsorhoz, n_p-1, tartozó kommunikációt és multiplex feldolgozást egy processzormag el tudja látni. A particionálási probléma ebben az esetben tehát nem a videó átszámító algoritmus párhuzamosítása, hanem rendszer szintű párhuzamosítási feladat.

Az egy processzormagos esetre, valamint a két processzormagos modellben az egyes processzormagokra meghatároztam a valósidejű működés feltételeit, és a processzor terhelés számításának módszerét.

A több processzormagos transzkóder modell nem egy teljes transzkóder rendszer, csak a párhuzamos videó átszámító algoritmus. Nyílthurkú transzkódolás esetén, a videó kódolással ellentétben, nincs adatfüggőség a képeken belül és a képek között a predikciós lépés hiánya miatt. A modell célja a párhuzamos nyílthurkú transzkódolás hatékonyságának elemzése multiprocesszoron. A modellhez központi feladatsort és dinamikus ütemezést alkalmaztam, amely viszonylag egyszerű megvalósítást tesz lehetővé, ugyanakkor a statikus ütemezéshez képest jobb terheléseloszlást biztosít. Hátránya, hogy az egyes DSP magok hozzáférését a feladat sorhoz szinkronizálni kell.

Meghatároztam a soros végrehajtáshoz képest elérhető gyorsulás maximális mértékét, és az ehhez szükséges feltételeket. Vizsgáltam a gyorsulás mértékét a térbeli felbontás mértékének függvényében.

8.3.1 Új tudományos eredmények

Tézis: A videó transzkódolás *rugalmasabban méretezhető számítási komplexitás és képminőség tekintetében*, mint a videó kódoló vagy dekódoló eljárások. Több processzormagos DSP multiprocesszoron, különböző bonyolultságú transzkódoló megoldások párhuzamosítási lehetőségeinek vizsgálatához, kidolgoztam három transzkóder modellt:

- Bemutattam, hogy a kiindulási feltételeknek megfelelő videó transzkódolása egy processzormagos DSP-n közel valósidőben elvégezhető. Kidolgoztam egy, egy processzormagos teljes videó transzkóder modellt, és *különböző processzor architektúrákon* vizsgáltam a *valósidejű feldolgozás feltételeit*.
- Bemutattam, hogy *többmagos DSP multiprocesszoron* a particionálási probléma nem csak a videó átszámító algoritmus párhuzamosításaként fogható fel, hanem tekinthető *rendszer szintű párhuzamosítási feladatként* is, és a *particionálás multimédia elem szinten* hatékonyan kialakítható, ehhez kidolgoztam egy két processzormagon működő teljes videó transzkóder modellt.
- *Többmagos DSP multiprocesszoron* vizsgáltam a videó transzkódoló algoritmus *térbeli párhuzamosításának hatékonyságát a párhuzamosítás mértékének függvényében,* amihez kidolgoztam egy több processzormagos videó transzkóder modellt.

Az irodalomban található párhuzamosítási megoldások a videó kódolás és dekódolás problémáival foglalkoznak, és nagyon kevés cikk foglalkozik speciálisan a transzkódolás párhuzamosítási megoldásaival. *Homogén nyílthurkú* videó transzkódolás *valósidejű* vizsgálata *egy processzormagos DSP-n* az irodalomban nincs publikálva. A cikkekben található párhuzamosítási megoldások továbbá nem teljes rendszert vizsgálnak, hanem csak az aktuális algoritmusok párhuzamosítási lehetőségeit. Az itt bemutatott párhuzamosítási megoldásban az *újdonság a teljes transzkódoló rendszer vizsgálata* a feladat particionálásához, valamint az ebből adódó *multimédia elem szintű particionálás.* Újdonság továbbá *többmagos homogén DSP multiprocesszor* használata videó transzkódoló particionáláshoz. Videó kódolás vagy dekódolás párhuzamosítását vizsgálták hasonló architektúrákon, mint az IBM Cell BE processzor, vagy sokmagos DSP szimulátor. A többmagos DSP-hez legközelebb álló architektúra amin transzkódolást vizsgálták több különálló DSP-ből felépített rendszer volt.

A különböző párhuzamosítási lehetőségek vizsgálatára három transzkóder modellt fejlesztettem ki és valósítottam meg. Az egy processzoron működő teljes transzkóder, a két processzormagon működő teljes transzkóder és a több processzormag lehetőségeinek vizsgálatára egy több magot is kihasználó transzkóder modellt. A transzkóderek vizsgálata MPEG-2 MP@ML teszt videókkal és valódi televízió adásból származó bemeneti videókkal történt. Az eredmények a legtöbb esetben a *vártnak megfelelően* alakultak.

Az eredmények következménye, hogy a két DSP magot kihasználó transzkódoló rendszerrel és a többmagos homogén DSP SoC kialakítású integrált áramkörök megjelenésével lehetővé vált egy *multiprocesszoron belül több transzkóder megvalósítása*. A több processzormagot kihasználó transzkódolási módszerekkel lehetőség nyílik *nagyobb felbontású videó, és nagyobb számítási komplexitású* transzkódoló algoritmusok kialakítása.

8.4 Továbbfejlesztés lehetőségei

A kutatási céloknak megfelelően, és a vizsgálati eredmények alapján, kialakítható olyan transzkóder, amely két DSP processzormagon, MPEG-2 MP@ML formátumú videó valósidejű feldolgozására képes. A vizsgálatok elvégzéséhez el is készítettem egy változatot TMS320C6472 típusú, hat C64+ DSP magot tartalmazó multiprocesszoron. Kisebb továbbfejlesztés után, ugyanezen a multiprocesszoron, az eredmények alapján, megvalósítható egy, öt műsort feldolgozó transzkóder is. Ez a transzkóder önállóan, vagy remultiplexerekbe integráltan alkalmas műsorszóró hálózatok közötti videó transzkódolásra.

A dolgozatom egyik célja a továbbfejlesztési lehetőségek keresése. A továbbfejlesztés egyik iránya lehet a nagy felbontású, HD, műsorok és/vagy újabb, hatékonyabb videó kódolások támogatása, mint a műsorszórásban is használt H.264 formátum. Ezek feldolgozása azonban nagyobb számítási teljesítményt igényel.

További fejlesztési irány bonyolultabb átkódoló algoritmus használata lehet. A kis komplexitású nyílthurkú transzkóder tipikus „lélegzés” hibát okoz a kimeneti videón. Ez a hiba csökkenthető vagy elkerülhető bonyolultabb átkódoló algoritmus használatával, amely azonban szintén nagyobb számítási kapacitást igényel.

Ebben a dolgozatban egy homogén videó transzkódoló módszert vizsgáltam, heterogén módszerhez amely formátumok közötti transzkódolást végez, nagyobb számítási kapacitásra van szükség.

Nagyobb számítási teljesítmény eléréséhez több DSP mag kihasználása lehet a megoldás, ezt vizsgáltam a több processzormagos transzkóder modell segítségével.

Egy speciális igény digitális műsorszóró rendszerekben a statisztikus remultiplexelés. Az analóg rendszerektől örökölt frekvencia kiosztás miatt, az egyes televízió csatornák sávszélessége jóval nagyobb, mint amit egy digitális műsor elfoglal. A hatékony frekvenciakihasználás érdekében ezért az egyes csatornákon olyan multiplexeket visznek át, amelyek több műsort is tartalmaznak, így kihasználják ezt a sávszélességet. A fix csatorna sávszélesség miatt a multiplexek bitsebessége is állandó. Az egyes műsorok bitsebességére azonban nincs ilyen megkötés, azok lehetnek változó bitsebességűek, de az együttesen nem haladhatják meg a multiplex bitsebességét. A statisztikus remultiplexer az egyes műsorok időben változó képtartalmától függően osztja szét úgy a teljes bitsebességet, hogy az egyes műsorok képminősége lehetőleg egy szinten maradjon. A statisztikus remultiplexereknek tehát olyan videó transzkódereket kell tartalmazniuk, amelyek szorosan kapcsolódnak egymáshoz, gyors az információ áramlás köztük. Egy többmagos DSP-n kialakított, több transzkódert tartalmazó rendszer alkalmas a feladatra. A továbbfejlesztés egyik iránya lehet tehát, egy statisztikus transzkódolást lehetővé tevő bitsebességvezérlő kifejlesztése.

Irodalom jegyzék

[1] ISO/IEC-JTC1/SC29/WG11, *Test model 5*, MPEG93/457, Apr. 1993.

[2] Telecommunication Sector of ITU (ITU-T) and ISO/IEC JTC 1., *"Information technology – Generic coding of moving pictures and associated audio information: Video",* Recommendation ITU-T H.262, ISO/IEC 13818-2, (MPEG-2 Video), feb. 2000.

[3] Telecommunication Sector of ITU (ITU-T) and ISO/IEC JTC 1., *"Coding of audio-visual objects – Part 10: Advanced Video Coding"*, Recommendation ITU-T H.264, ISO/IEC 14496-10, (MPEG-4 AVC), Mar. 2005.

[4] Radiocommunication Sector of ITU (ITU-R), „*Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios",* Recommendation ITU-R BT.601-7, march 2011.

[5] Radiocommunication Sector of ITU (ITU-R), „*Parameter values for the HDTV standards for production and international programme exchange"*, Recommendation ITU-R BT.709-5, Apr. 2002.

[6] IEEE Circuits and Systems Society, *"IEEE Standard Specifications for the Implementations of 8X8 Inverse Discrete Cosine Transform"*, IEEE Std. 1180-1990, dec 1990.

[7] Telecommunication Sector of ITU (ITU-T) and ISO/IEC JTC 1, *"Digital compression and coding of continuous-tone still images"*, Recommendation ITU-T T.81, ISO/IEC 10918-1 (JPEG), Sept. 1992.

[8] Telecommunication Sector of ITU (ITU-T) and ISO/IEC, *"Information technology – JPEG 2000 image coding system: Core coding system"*, Recommendation ITU-T T.800, ISO/IEC 15444-1 (JPEG 2000), Aug. 2002.

[9] Telecommunication Sector of ITU (ITU-T), *"Codec for videoconferencing using primary digital group transmission"*, Recommendation ITU-T H.120., Mar. 1993.

[10] Telecommunication Sector of ITU (ITU-T), *"Video Codec for Audiovisual Services at p x 64 kbit/s"* Recommendation ITU-T H.261, Mar. 1993.

[11] Berc, et. al., *"RTP Payload Format for JPEG"*, Standards Track RFC 2435, Oct 1998

[12] IEC, *"Recording – Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems)"*, IEC Standard 61834, Aug. 1998.

[13] Intel *"Intel 64 and IA-32 Architectures Software Developer's Manual"* Vol. 2. Instruction Set Reference, May 2011.

[14] Intel, *"Moore's law"*, http://www.intel.com/about/companyinfo/museum/exhibits/moore.htm

[15] D. Bell, G. Wood, *"Multicore Programming Guide",* Texas Instruments, SPRAB27A, Aug. 2009.

[16] *"TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide"*, Texas Instruments, SPRU732, July 2010.

[17] *"TMS320C6000 Optimizing Compiler 7.2 User's Guide"*, Texas Instruments, SPRU187, Jan. 2011.

[18] *"TMS320C6000 Programmer's Guide"*, Texas Instruments, SPRU198, Apr. 2010.

[19] *"TMS320C6472 Fixed-Point Digital Signal Processor"*, Texas Instruments, SPRS612, Oct. 2010

[20] *"TMS320DM6437 Digital Media Processor"*, Texas Instruments, SPRS345, Jun. 2008.

[21] *"TMS320C8x System-Level Synopsis"*, Texas Instruments, SPRU113, Sept. 1995.

[22] *"DaVinci Technology Overview"*, Texas Instruments, SPRB189, Sept. 2008.

[23] D. Allred, G. Martinez, *Maximizing the Power of ARM with DSP"*, Texas Instruments, SPRY152, Oct. 2010.

[24] IBM, *"Cell Broadband Engine Programming Handbook"*, Apr. 2009.

[25] AMD, "*APU 101: All about AMD Fusion Accelerated Processing Units*", 2011.

[26] AMD, *"AMD Accelerated Parallel Processing, OpenCL, Programming Guide"*, Jun. 2011.

[27] NVIDIA, *"NVIDIA CUDA C Programming Guide"*, ver. 4.0, May 2011.

[28] Message Passing Interface Forum, *"MPI: A Message-Passing Interface Standard"*, ver. 2.2, Sept. 2009

[29] Jan-Willem van de Waerdt, *"The TM3270 Media-processor"*, Proefschrift Technische Universiteit Delft, 2006.

[30] C. E. Shannon, *"Coding theorems for a discrete source with a fidelity criterion",* IRE National Conv. Record, Part 4, pp. 142-163, 1959.

[31] G. J. Sullivan, T. Wiegand, *"Video Compression – From Concepts to the H.264/AVC Standard."*, in Proceedings of the IEEE, Vol. 93, No. 1., pp. 18-31. ISSN: 0018-9219, jan 2005

[32] P. H. Westerink; R. Rajagopalan; C. A. Gonzales, *„Two-pass MPEG-2 variable-bitrate encoding"*, IBM Journal of Research and Development, Vol. 43, No. 4, pp: 471-488, July 1999.

[33] Syed Ali Khayam, *"The Discrete Cosine Transform (DCT): Theory and Application",* ECE 802 – 602: Information Theory and Coding, March 2003.

[34] G. Strang, *"The Discrete Cosine Transform"* SIAM Review, Volume 41, No 1, pp. 135-147, 1999.

[35] Z. Wang, *"Fast Algorithms for the Discrete W-Transform and for the Discrete Fourier Transform"* IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, No. 4, pp. 803-816, Aug. 1984

[36] C. Loeffler, A. Ligtenberg, G.S. Moschytz, *"Practical fast 1-D DCT algorithms with 11 multiplications"*, proc. of Acoustics, Speech, and Signal Processing, 1989. ICASSP-89, Vol. 2, pp. 988-991, May 1989.

[37] A. Obukhov, A. Kharlamov, *"Discrete cosine transform for 8x8 blocks with CUDA"*, White paper, nVidia, Oct. 2008

[38] F. W. Mounts, *"A video encoding system with conditional picture element replenishment"*, Bell System Technical Journal, Vol. 48, No. 7, pp. 2545-2554, Sep. 1969.

[39] J. R. Jain, A. K. Jain, *"Displacement measurement and its application in interframe image coding",* IEEE Trans. On Communication., Vol. COM-29, No. 12, pp. 1799-1808, Dec. 1981.

[40] C.H. Hsieh, T.P. Lin, *"VLSI architecture for block-matching motion estimation algorithm"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 2, No. 2, pp. 169–175, June. 1992.

[41] L. De Vos, M. Schobinger, *"VLSI architecture for a flexible block matching processor"*, IEEE Transaction on Circuits and Systems for Video Technology, Vol. 5, No. 5, pp. 417–428, 1995.

[42] A. Pirson, *"A Programmable Motion Estimation Processor for Full Search Block Matching"*, IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 5, pp. 3283–3286, 1995.

[43] M.T. Sun, T.C. Chen, A.M. Gottlieb, *"VLSI implementation of a 16×16 discrete cosine transform"*, IEEE Transactions on Circuits and Systems Vol. 36, No.4, 610–617, Apr. 1989.

[44] I.-K. Kim, J.-J. Cha, H.-J. Cho, *"A design of 2-D DCT/IDCT for real-time video applications"*, IEEE International Conference on VLSI and CAD, pp. 557–559, 1999.

[45] I. Ahmad, S.M. Akramullah, M.L. Liou, M. Kafeel, *"A scalable off-line MPEG-2 video encoding scheme using a multiprocessor system"*, Parallel Computing, Jul. 2000.

[46] I. Ahmad, Y. He, M. L. Liou, *"Video compression with parallel processing"*, Parallel Computing 28 (2002), pp: 1039–1078, Dec. 2001.

[47] A. Rodriguez, A. González, M.P. Malumbres, *"Performance evaluation of parallel MPEG-4 video coding algorithms on clusters of workstations",* International Conference on Parallel Computing in Electrical Engineering, 2004. PARELEC 2004. Sept. 2004.

[48] E. Iwata, K. Olukotun, *"Exploiting Coarse-Grain Parallelism in the MPEG-2 Algorithm"*, 1998

[49] B. Tye, K. Goh, W. Lin, G. Powell, T. Ohya, S. Adachi, *"DSP implementation of very low bit rate videoconferencing system"*, in: Proc. of IEEE International Conference on Information, Communications and Signal Processing, vol. 3, pp. 1275–1278, Sept. 1997.

[50] C. Meenderinck, A. Azevedo, B. Juurlink, M. A. Mesa, A. Ramirez, *"Parallel Scalability of Video Decoders"*, Springerlink.com, Aug. 2008.

[51] O. Cantineau, J.D. Legat, *"Efficient parallelization of an MPEG-2 codec on a TMS320C80 video processor"*, IEEE International Conference on Image Processing, vol. 3, pp. 977–980, Oct. 1998.

[52] A. Rodriguez, A. Gonzalez, M. P. Malumbres, *"Hierarchical Parallelization of an H.264/AVC Video Encoder"*, Proc. of the International Symposium on Parallel Computing in Electrical Engineering, pp. 363–368, Sept. 2006.

[53] M. Alvarez, A. Ramirez, M. Valero, A. Azevedo, C.H. Meenderinck, B.H.H. Juurlink, *"Performance Evaluation of Macroblock-level Parallelization of H.264 Decoding on a cc-NUMA Multiprocessor Architecture"*, Proceedings of the 4CCC: 4th Colombian Computing Conference, Apr. 2009.

[54] Bryan Hughes, *"An Oblivious Routing Scheme for Parallel Computing in General Embedded Networks"*, Ph. D. Dissertation In Electrical Engineering, Texas Tech University, May 2010.

[55] S. Bozóki, R.L. Lagendijk, *"Scene adaptive rate control in a distributed parallel MPEG video encoder"*, IEEE International Conference on Image Processing, vol. 2, pp. 780–783, Oct. 1997.

[56] J.E.Smith, G.S. Sohi, *"The Microarchitecture of Superscalar Processors"*, Proceedings of the IEEE, vol. 83, no. 12, pp. 1609, Aug. 1995.

[57] W. M. Johnson, *"Super-Scalar Processor Design"*, Technical Report No. CSL-TR-89-383, Jun. 1989.

[58] C. McNairy, D. Soltis, *"Itanium 2 processor microarchitecture"*, IEEE Micro, Vol. 23, Num. 2, pp. 44-55, Apr. 2003.

[59] P. Stravers, J. Hoogerbrugge, *"Homogeneous multiprocessing and the future of silicon design paradigms"*, IEEE International Symposium on VLSI Technology, Systems, and Applications, Proceedings of Technical Papers, pp. 184–187, Apr. 2001.

[60] S. Venkatasubramanian, *"The Graphics Card as a Stream Computer"*, AT&T Labs Research, 2003.

[61] B. Formanek, T. Ádám, *"DSP implementation of an MPEG-2 video bitrate transcoder"*, Proc. of the 11th International Carpathian Control Conference, pp. 195-198, May 2010.

[62] B.D. Sutter, P. Raghavan, A. Lambrechts, *"Coarse-Grained Reconfigurable Array Architectures"*, S.S. Bhattacharyya et al. (eds.), Handbook of Signal Processing Systems, Springer Science+Business Media, part 2, pp. 449-484, 2010.

[63] G.M. Amdahl, *"Validity of single-processor approach to achieving large-scale computing capability"*, Proc. of AFIPS Conference, pp. 483-485, 1967.

[64] J.L. Gustafson, *"Reevaluating Amdahl's Law"*, CACM, Vol. 31, Num 5, pp. 532-533, 1988.

[65] Y. Shi, *"Reevaluating Amdahl's Law and Gustafson's Law"*, Temple University, http://www.cis.temple.edu/~shi/docs/amdahl/amdahl.html, Oct. 1996.

[66] X.-H. Sun, Y. Chen, *"Reevaluating Amdahl's law in the multicore era"*, Journal of Parallel and Distributed Computing, Vol. 70, No. 2, pp. 183-188, Feb. 2010.

[67] Z. He, S. K. Mitra, "*A Linear Source Model and a Unified Rate Control Algorithm for DCT Video Coding"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, No. 11, pp: 970-982, Nov. 2002.

[68] Z. Lei, N. D. Georganas, *"Rate Adaptation Transcoding for Precoded Video Streams"*, Proc. of the 10th ACM International Conference on Multimedia, pp: 127-136, Dec. 2002.

[69] Z. Lei, N. D. Georganas, *"Accurate bit allocation and rate control for DCT domain video transcoding"*, Proc. of IEEE Canadian Conference on Electrical and Computer Engineering, Vol. 2, pp: 968-973, May. 2002.

[70] H.-M. Hang, J.-J. Chen, *"Source model for transform video coder and its application. I. Fundamental theory"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 7, No. 2, Apr. 1997.

[71] B. Tao, B.W. Dickinson, H. A. Peterson, *"Adaptive model-driven bit allocation for MPEG video coding"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 1, Feb. 2000.

[72] W. Ding, B. Liu, *"Rate control of MPEG video coding and recording by rate-quantization modeling"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 1, pp: 12-20, Feb. 1996.

[73] L.-J. Lin., A. Ortega, *"Bit-Rate Control Using Piecewise Approximated Rate–Distortion Characteristics"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 8, No. 4, pp: 456-449, Aug. 1998.

[74] T. Chiang, Y.-Q. Zhang, *"A New Rate Control Scheme Using Quadratic Rate Distortion Model"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 7, No. 1, pp: 246-250, Feb. 1997.

[75] E. Y. Lam, J. W. Goodman, *"A Mathematical Analysis of the DCT Coefficient Distributions for Images"*, IEEE Transactions on Image Processing, Vol. 9, No. 10, pp: 1661-1666, Oct. 2000.

[76] J. Ribas-Corbera, S. Lei, *"Rate control in DCT video coding for low-delay communications"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 9, No. 1, pp: 172-185, Feb. 1999.

[77] A.Puri, R. Aravind, *"Motion-compensated Video Coding with Adaptive Perceptual Quantization"*, IEEE Transaction on Circuits and Systems for Video Technology, Vol. 1, No. 4, pp: 351-361, Dec. 1991.

[78] J.-B. Cheng, H.-M. Hang, *"Adaptive piecewise linear bits estimation model for MPEG based video coding"*, Proc. of International Conference on Image Processing, Vol. 2, pp: 551-554, Oct. 1995.

[79] G. Valenzise, M. Tagliasacchi, S. Tubaro, *"A Smoothed, Minimum Distortion-Variance Rate Control Algorithm for Multiplexed Transcoded Video Sequences"*, Proc. of International Workshop on Mobile Video, pp: 55-60, Sep. 2007.

[80] Javed I. Khan, Q. Gu, "*Network Aware Symbiotic Video Transcoding for in Stream Rate Adaptation on Interactive Transport Control"*, IEEE International Symposium on Network Computing and Applications, pp: 201-213, Oct. 2001.

[81] J. Xin, C.-W. Lin, M.-T. Sun, *"Digital Video Transcoding"*, Proceedings of the IEEE, Vol. 93, No. 1, pp: 84 – 97, Jan. 2005.

[82] I. Ahmad, Xiaohui Wei, Yu Sun, Ya-Qin Zhang, *"Video transcoding: an overview of various techniques and research issues"*, IEEE Transactions on Multimedia, Vol. 7, No. 5, pp: 793-804, Oct. 2005.

[83] A. Vetro, C. Christopoulos, Huifang Sun, *"Video transcoding architectures and techniques: an overview"*, IEEE Signal Processing Magazine, Vol. 20, No. 2, pp.: 18-29, Mar. 2003.

[84] A. Eleftheriadis, D. Anastassiou, *"Constrained and general dynamic rate shaping of compressed digital video",* Proc. of IEEE International Conference on Image Processing, Vol. 3, pp. 396–399, 1995.

[85] H. Sun, W. Kwok, J. W. Zdepski, *"Architectures for MPEG compressed bitstream scaling"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 2, pp.: 191–199, Apr. 1996.

[86] Y. Nakajima, H. Hori, T. Kanoh, *"Rate conversion of MPEG coded video by re-quantization process"*, Proc. IEEE International Conference on Image Processing, Vol. 3, pp.: 408–411, Oct. 1995.

[87] J. Youn, M.-T. Sun, *"Video transcoding with H.263 bit streams"* Journal of Visual Communication and Image Representation, Vol. 11, pp. 385–404, Dec. 2000.

[88] T. Shanableh, M. Ghanbari, *"Heterogeneous Video Transcoding to Lower Spatio-Temporal Resolutions and Different Encoding Formats"*, IEEE Transactions on Multimedia, Vol. 2, No. 2, pp. 101–110, Jun. 2000.

[89] G. Keesman, R. Hellinghuizen, F. Hoeksema, G. Heideman, *"Transcoding of MPEG bitstreams"*, Signal Processing: Image Communication., Vol. 8, No. 6, pp. 481–500, Sept. 1996.

[90] D. G. Morrison, M. E. Nilson, M. Ghanbari, *"Reduction of the bit-rate of compressed video while in its coded form"*, in Proc. of 6th International Workshop on Packet Video, pp.: D17.1–D17.4, Sept. 1994.

[91] P. Assunçno, M. Ghanbari, *"Post-processing of MPEG-2 coded video for transmission at lower bit-rates"*, in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 4, Atlanta, GA, pp. 1998-2001, May 1996.

[92] P. Zhang, X. Jia, W. Ga, Q. Huangc, *"Drift Compensated Coding Optimization for Fast Bit-rate Reduction Transcoding"*, Proc. of SPIE – Visual Communications and Image Processing, Vol. 6508, pp. 65081K, Jan. 2007.

[93] S.F. Chang, D.G. Messerschmidt, *"Manipulation and compositing of MC-DCT compressed video"*, IEEE Journal on Select. Areas Communication, Vol. 13, No. 1, pp. 1-11, Jan. 1995.

[94] P. Assunção, M. Ghanbari, *"A frequency-domain video transcoder for dynamic bit-rate reduction of MPEG-2 bitstreams"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 8, No. 8, pp. 953-967, Dec. 1998.

[95] N. Merhav, *"Multiplication-free approximate algorithms for compressed-domain linear operations on images"*, IEEE Transactions on Image Processing, Vol. 8, pp. 247-254, Feb. 1999.

[96] C.W. Lin, Y.R. Lee, *"Fast algorithms for DCT-domain video transcoding"*, in Proc. of IEEE International Conference on Image Processing, Vol.1, pp. 421-424, Oct. 2001.

[97] S.-Z. Liu, A. C. Bovik, *"Local bandwidth constrained fast inverse motion compensation for DCT-domain video transcoding"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, No. 5, pp. 309–319, May 2002.

[98] T. Shanableh, M. Ghanbari, *"Transcoding architectures for DCT-domain heterogeneous video transcoding"*, in Proc. IEEE International Conference on Image Processing 2001, Vol. 1, pp. 433–436., Oct. 2001.

[99] J. Song, B.-L. Yeo, *"A fast algorithm for DCT-domain inverse motion compensation based on shared information in a macroblock"*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 5, pp. 767–775, Aug. 2000.

[100] S. Acharya, B. Smith, *"Compressed domain transcoding of MPEG"*, in Proc. IEEE International Conference on Multimedia Computing and Systems, pp. 295–304, Jun. 1998.

[101] U.-V. Koc, K. J. R. Liu, *"Discrete-cosine/sine-transform based motion estimation",* in Proc. IEEE International. Conference on Image Processing, Vol. 3, Nov. 1994, pp. 771–775, Nov. 1994.

[102] R. Xie, J. Liu, X. Wang, *"Efficient MPEG-2 to MPEG-4 compressed video transcoding"*, in Proc. of SPIE – Visual Communications and Image Processing, Vol. 4671, pp. 192–201, Jan. 2002.

[103] M. A. Baker, P. Dalale, K. S. Chatha, S. B. K. Vrudhula, *"A Scalable Parallel H.264 Decoder on the IBM Cell Broadband Engine Architecture"*, Proc. of the International Conference on Hardware-Software Codesign and System Synthesis (CODES-ISSS), Oct. 2009.

[104] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hoogerbrugge, M. Alvarez, A. Rammirez, *"Parallel H.264 Decoding on an Embedded Multicore Processor"*, Proc. of the 4th International Conference on High Performance and Embedded Architectures and Compilers, pp. 404-418, Jan. 2009.

[105] Xu Xiaoshen, Jiang Hongxu, Jin Liang, Lei Dandan, Li Bo, *"A Multi-DSP System for High-Performance Video Applications"*, Proc. of the 11th IEEE Singapore International Conference on Communication Systems, pp. 778 – 782, Nov. 2008.

[106] A. Raman, S. Sethuraman, A. Kumar, M. Bhaskaranand, O. Sharma, M. Agrawal, *"Scalable Multi-Processor Software Architecture for a MPEG-2 to H.264 Transcoder"*, GSPx 2006 -The International Signal Processing Conference, Nov. 2006.

[107] A. Raman, K. Singh, M. A. Mohan, N. Shigihalli, S. Sethuraman, B. S. Supreeth, *"MPEG-2 to H.264 Transcoder on TI TMS320DM642"*, Proc. of the IEEE International Conference on Multimedia and Expo, Vol. 1, pp. 322., Jun. 2004.

[108] x264, http://www.videolan.org/developers/x264.html