Just-in-time video transcoding system on heterogeneous high performance computing architectures

Piljić, Igor

University of Zagreb
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Igor Piljić

JUST-IN-TIME VIDEO TRANSCODING SYSTEM ON HETEROGENEOUS HIGH PERFORMANCE COMPUTING ARCHITECTURES

DOCTORAL THESIS

Zagreb, 2019

University of Zagreb
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Igor Piljić

JUST-IN-TIME VIDEO TRANSCODING SYSTEM ON HETEROGENEOUS HIGH PERFORMANCE COMPUTING ARCHITECTURES

DOCTORAL THESIS

Supervisor:
Professor Mario Kovač, Ph.D.

Zagreb, 2019

Sveučilište u Zagrebu
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Igor Piljić

SUSTAV ZA PRAVOVREMENO VIDEOTRANSKODIRANJE NA RAZNORODNIM ARHITEKTURAMA ZA RAČUNARSTVO VISOKIH PERFORMANCI

DOKTORSKI RAD

Mentor:
Prof. dr. sc. Mario Kovač

Zagreb, 2019.

Doctoral thesis was made at the University of Zagreb,

Faculty of Electrical Engineering and Computing,

Department of Control and Computer Engineering

Supervisor:

Professor Mario Kovač, Ph.D.

Doctoral thesis contains: 131 pages

Doctoral thesis number: _____

ABOUT THE SUPERVISOR:

Prof. dr. sc. Mario Kovač is full professor at the Faculty of Electrical Engineering and Computing (FER), University of Zagreb, Croatia. In 1990 and 1991 he received a VLSI and Computer Architecture Scholarship at the University of South Florida, and he subsequently received the Fulbright Award in 1993. He holds several patents including US patents in multimedia systems and architecture domains. In 2008, Croatian President awarded him with the the Medal of Honor "Order of Danica Hrvatska with the image of Ruđer Bošković" for special merit in science. Professor Kovač served as Head of the Dept. of Control and Computer Engineering and Vice Dean for Business Development at FER. He was a member of the supervisory boards of: CARNet, Croatian Institute of Technology and BICRO - Business Innovation Center of Croatia. He currently holds several positions: Chief Communications Officer (CCO) at European Processor Initiative; Expert member of Governing Board as well as Research and Innovation Advisory Group Observer Member at EuroHPC Joint Undertaking; Director HPC Architectures and Applications Research Center at FER. He is senior member of the IEEE Computer Society.

O MENTORU:

Prof. dr. sc. Mario Kovač je redovni profesor na Fakultetu elektrotehnike i računarstva (FER), Sveučilište u Zagrebu. 1990. i 1991. dobio je stipendiju za VLSI i računalnu arhitekturu na Sveučilištu u Južnoj Floridi, a potom je 1993. dobio nagradu Fulbright. Autor je nekoliko patenata, uključujući američke patente u području multimedijskih arhitektura i sustava. 2008. godine odlikovan je medaljom "Orden Danice Hrvatske s likom Ruđera Boškovića" za posebne zasluge u znanosti. Profesor Kovač obnašao je dužnost predsjednika Zavoda za automatiku i računalno inženjerstvo i prodekana za poslovanje na FER-u. Bio je član nadzornih odbora: CARNet-a, Hrvatskog tehnološkog instituta i BICRO - poslovno-inovacijskog centra Hrvatske. Trenutno obnaša nekoliko pozicija: glavni direktor za komunikacije (CCO) Europskog projekta „European Processor Initiative“; Stručni član upravnog odbora, kao i savjetodavne skupine za istraživanje i inovacije EuroHPC; Direktor centra za istraživanje arhitektura i aplikacija za računarstvo visokih performanci na FER-u. Viši je član IEEE Computer Society.

SUMMARY

Latest analysis show that 82 percent of global IP traffic will be video traffic by 2022. Handling this amount of data is a very challenging task for video content providers. Another factor that highlights this problem is the continuously growing number of different devices that are able to play video content. With such diversity of devices, with different characteristics, a single copy of the encoded video cannot match requirements of all playback conditions. Just-in-Time (JiT) video transcoding has one of the key roles in resolving these issues. However, it is an extremely compute-intensive and resource-hungry process, especially when it is based on the novel High Efficiency Video Coding standard.

This thesis presents a novel algorithm for reusing coding information from the input video stream. The main concept behind the proposed algorithm is to estimate the computational complexity of re-encoding each coding block based on the information retrieved from the decoded frame and to balance the workload of the transcoder accordingly. The final goal is to achieve an optimal trade-off between video quality of the transcoded bitstream and coding efficiency while conforming to strict timing requirements imposed by Just-in-Time transcoding. To achieve a more efficient solution, a hardware accelerator for inter prediction, as one of the key compute-intensive kernels in video transcoding based on HEVC, is designed and implemented.

An integrated system composed of implemented algorithm and custom hardware accelerator is evaluated and compared to two baseline transcoders: Just-in-Time transcoder without reusing data and regular transcoder without timing restrictions. Compared with JiT transcoder, the proposed solution increases video quality by 0.945 dB and reduces bitrate on average by 35.06%. Significant speedups of up to 4 times are achieved compared with transcoder without timing requirements but with average losses of 0.592 in PSNR and 21.74% in bitrate.

Keywords: video transcoding, HEVC, reusing data, hardware accelerators, heterogeneous high-performance architectures

Sustav za pravovremeno videotranskodiranje na raznorodnim arhitekturama za računarstvo visokih performanci

Statistike pokazuju da će 82% globalnog Internet Protokol (IP) prometa do 2022. godine činiti video promet, što je povećanje sa 75% udjela u 2017. godini. Godišnji globalni IP promet će doseći granicu od 4.8 ZB (ZB= 1000 Egzabajta) do 2022. godine, što znači da će približno 3.9 ZB biti promet video sadržaja. Preračunato u minute, do 2022. svake sekunde će mrežom proći preko milijun minuta videa.

Rukovanje tolikom količinom podataka predstavlja iznimno zahtjevan zadatak za poslužitelje video sadržaja. Još jedan faktor koji naglašava složenost ovog problema je činjenica da broj različitih uređaja koji mogu prikazivati video sadržaj konstantno raste. Takvu raznovrsnost uređaja, koji mogu imati različitu procesorsku snagu, sposobnost dekodiranja i različite rezolucije, a mogu biti spojeni i na mreže s niskom ili visokom propusnosti, nemoguće je zadovoljiti samo jednom verzijom videa. Na primjer, slanje videa visoke rezolucije (1080p) koji ima 60 okvira u sekundi (60fps) mobilnom uređaju koji nema tako visoku rezoluciju zaslona i koji je spojen na mrežu niske propusnosti je, ne samo nepotreban trošak resursa, nego će vjerojatno uzrokovati i kašnjenje u prikazu videa, smanjujući time iskustvenu kvalitetu korisnika.

Trenutni poslužitelji video usluga pokušavaju riješiti ovaj problem tako da prvo kodiraju isti video u više različitih verzija te nakon toga korisniku pošalju onu verziju koja najbolje odgovara njegovim zahtjevima. Ovakav pristup zahtjeva veliku količinu memorije za pohranu svih verzija, a osim toga odabrana verzija ne mora u potpunosti odgovarati svim zahtjevima krajnjeg korisnika. Nadalje, korištenje novih rezolucija (4K, 8K,…), kao i činjenica da je video sadržaj raspodijeljen tako da 90% sadržaja gleda samo 10% korisnika i obrnuto (distribucija „dugog repa“), čine ovakav koncept teško održivim. Pravovremeno videotranskodiranje ima jednu od ključnih uloga u rješavanju ovog problema. Videotranskodiranje se odnosi na prilagodbu video sadržaja ovisno o specifičnim okolnostima i karakteristikama uređaja, a uključuje promjenu prostorne, vremenske ili amplitudne rezolucije, te video formata. Umjesto spremanja više verzija jednog videa, poslužitelji mogu spremiti samo jednu verziju s najvišom kvalitetom, a zatim na zahtjev korisnika, transkodirati video u stvarnom vremenu, ovisno o trenutnom zahtjevu, a zatim taj transkodirani video poslati korisniku. Iako pravovremeno videotranskodiranje povećava učinkovitost sustava osiguravajući najbolju iskustvenu kvalitetu, to je iznimno računalno zahtjevan postupak. Još jedan od pristupa koji se danas koristi je

Skalabilno video kodiranje (*eng. Scalable Video Coding – SVC*). SVC sadrži kodiranu najkvalitetniju verziju videa, a za slučaj da je potreban zapis manje kvalitete, potrebno je samo izbaciti određene pakete unutar SVC formata prilikom slanja. Spremanje videa u SVC formatu značajno smanjuje cijenu pohrane podataka, ali i dalje ne omogućuje najbolju iskoristivost resursa, niti najbolju kvalitetu videa. Neka istraživanja kombiniraju postupke pravovremenog videotranskodiranja sa spremanjem više verzija videa u hibridne platforme koje koriste statistiku pregleda da bi odlučili koje video zapise ili dijelove videa je potrebno pravovremeno transkodirati, a koje dijelove se isplati pohraniti u više verzija.

HEVC ili H.265 (*eng. High Efficiency Video Coding*) norma za video kodiranje postiže značajan napredak u kompresiji za razliku od prethodne AVC norme. Uz istu subjektivnu kvalitetu, HEVC ostvaruje približno 50% bolju kompresiju, ali povećava računalnu složenost i zahtjev za resursima i do 10 puta. Prilikom razvoja HEVC norme, posebna pozornost bila je usmjerena na mogućnosti paralelizacije algoritama te njihovo izvođenje na sklopovskim arhitekturama. Učinkovito iskorištavanje ovih koncepata je ključno kada govorimo o pravovremenom videotranskodiranju visokih performanci, pogotovo na raznorodnim višejezgrenim arhitekturama.

Pravovremeno videotranskodiranje zasnovano na HEVC normi je iznimno računalno zahtjevan postupak te je postizanje najboljeg odnosa između kvalitete videa i računalne složenosti tema brojnih istraživanja. Pametno iskorištavanje informacija o kodiranju početnog video zapisa ima ključnu ulogu u gotovo svim povezanim istraživanjima. Programski algoritmi mogu postići značajan napredak u smanjenju složenosti i ubrzanju postupka, ali da bi se zadovoljili strogi vremenski zahtjevi, moraju se istražiti i iskoristiti sklopovske jezgre za ubrzanje i raznorodne arhitekture na računalima visokih performanci. Optimizacija i učinkovito raspoređivanje pojedinih dijelova algoritma na različite jezgre raznorodnog sustava je nužno da bi se postigao najbolji balans između kvalitete videa, potrošnje energije i kompresije videa, a istovremeno zadovoljavajući zahtjeve pravovremenog izvođenja i željene kvalitete usluge.

Glavni cilj ove doktorske disertacije bilo je istražiti tehnike iskorištavanja informacija o kodiranju ulaznog video toka kodiranog HEVC standardom da bi se ubrzao proces ponovnog kodiranja, ali bez negativnog utjecaja na kvalitetu videa i/ili učinkovitost kodiranja. U cilju daljnjeg poboljšanja procesa videotranskodiranja, istražene su i učinkovite izvedbe pravovremenog videotranskodiranja na raznorodnim arhitekturama za računarstvo visokih performanci.

U sklopu provedenog istraživanja razvijeno je programsko-sklopovsko rješenje Bolt65. Bolt65 sastoji se od enkodera, dekodera i transkodera zasnovanog na HEVC standardu, čiji je glavni cilj ostvariti videotranskodiranje u stvarnom vremenu. Poseban fokus pri razvoju ovog rješenja postavljen je na učinkovitost s obzirom na performance ostvarenu optimizacijom za programsko-sklopovske sustave. Algoritam za iskorištavanje informacija o kodiranju ulaznog video toka, predstavljen kao jedan od doprinosa ove disertacije, ugrađen je i testiran unutar Bolt65 rješenja.

Za validaciju i verifikaciju svih provedenih eksperimenata korišten je isti set videa, s različitim vremenskim i prostornim rezolucijama da bi se pokrio što širi spektar mogućih kombinacija pri videotranskodiranju. U obzir su uzete video sekvence rezolucije veće od 1280x720 piksela s maksimalnim brojem od 120 okvira u sekundi. Raznorodni sustav na kojem su izvršena sva testiranja sastoji se od procesora opće namjene i Kintex Ultrascale FPGA pločice koji su međusobno povezani preko visoko propusne PCIe (Gen3 x8) sabirnice. Rezultati razvijenog transkodera uspoređeni su s dva transkodera: pravovremenog Bolt65 transkodera i Kvazaar transkodera. Tri glavne karakteristike transkodiranja su praćene prilikom vrednovanja svih rezultata: vrijeme izvođenja, bitovna brzina prijenosa (*eng. bitrate*) za analizu učinkovitosti kodiranja te PSNR (*eng. Peak signal-to-noise ratio*) za mjerenje kvalitete transkodiranog video zapisa.

Inteligentno iskorištavanje informacija o kodiranju ulaznog video zapisa ima ključnu ulogu u poboljšavanju procesa video transkodiranja. U ovoj disertaciji identificirane su i korištene tri vrste podataka iz ulaznog video toka:

- Veličina dekodiranih kodnih blokova (*eng. Coding Unit - CU*), odnosno broj bitova koji je bio potreban da bi se pojedini kodni blok kodirao u originalnom video zapisu
- Broj mapiranih kodnih blokova, odnosno broj kodnih blokova iz originalnog videa koji pokriva isto područje slike kao i trenutno promatrani kodni blok u transkodiranoj slici
- Vrsta predikcije mapiranih kodnih blokova, odnosno način predikcije (intra ili inter) pojedinih kodnih blokova iz originalnog videa

Jedan od glavnih aspekata predstavljenog programskog algoritma je kategorizacija kodnih blokova. Glavna zamisao ovog koncepta je razdvojiti kodne blokove u različite

kategorije ovisno o složenosti njihovog procesiranja, a zatim kodirati pojedine blokove ovisno o kategorijama kojima pripadaju. Više procesorske moći će biti uloženo u analizu i transkodiranje složenijih blokova kodiranja, s obzirom na to da je veća vjerojatnost da se upravo u tom dijelu slike nalazi više detalja. Tri su skupine kategorija u kojoj se kategorizira svaki kodni blok:

- Kategorizacija temeljena na broju bitova iz dekodiranog videa – kategorije LBC (*eng. Low Bit Complexity*), MBC (*eng. Medium Bit Complexity*) i HBC (*eng. High Bit Complexity*)
- Kategorizacija temeljena na broju mapiranih kodnih blokova – kategorije LM (*eng. Low Mapped*), MM (*eng. Medium Mapped*) i HM (*eng. High Mapped*)
- Kategorizacija temeljena na vrsti predikcije mapiranih kodnih blokova – kategorije InterM, IntraM, ComboInter, ComboIntra

Svaka od tri vrste kategorizacije kontrolirana je zasebnim koeficijentima kojima se može regulirati broj kodnih blokova u svakoj od kategorija te koji se mogu dinamički mijenjati tijekom procesa transkodiranja, što je iznimno bitno kod pravovremenog izvođenja. Primjerice, ako se prilikom transkodiranja detektira da brzina izvođenja pada te da je izvođenje u stvarnom vremenu ugroženo, koeficijenti se mogu podesiti tako da se veći broj kodnih blokova kategorizira u manje složene kategorije (npr. LBC ili LM), čime se automatski smanjuje složenost ukupnog transkodiranja te samim time ubrzava čitav proces.

Algoritam predstavljen u ovoj disertaciji iskorištava informacije o kodiranju ulaznog video toka da bi kategorizirao kodne blokove ovisno o njihovoj složenosti, nakon čega donosi odluke u fazi transkodiranja ovisno o tome kojim kategorijama pojedini kodni blok pripada. Na samom početku, nakon dekodiranja okvira, izvlače se svi relevantni podaci o kodiranju originalnog videa, nakon čega se kreće u novo kodiranje s novim parametrima. Transkodirani okvir se zatim dijeli u najveće moguće kodne blokove (64x64 piksela) te se svi blokovi kategoriziraju u svaku od tri već navedene kategorije. Idući korak je inicijalna podjela kodnih blokova na manje blokove veličine od 32x32 do 8x8 piksela. U ovom koraku odluka o podjeli temelji se na prve dvije kategorizacije (kategorizacija temeljena na broju bitova iz dekodiranog videa i kategorizacija temeljena na broju mapiranih kodnih blokova). Vjerojatnost da će kodni blok biti podijeljen na manje blokove je veća ako kodni blok pripada nekoj od složenijih kategorija (npr. HM i/ili HBC). Nakon inicijalne podjele, svaki novonastali blok ponovo prolazi kroz proces kategorizacije. Idući korak je odluka o načinu predikcije koja se donosi isključivo

ovisno o trećoj vrsti kategorizacije – kategorizacija temeljena na vrsti predikcije mapiranih kodnih blokova. Ovisno o načinima predikcije mapiranih kodnih blokova te broju mapiranih kodnih blokova koji imaju sličan način predikcije, stvara se podskup intra i inter predikcijskih kandidata koji se evaluiraju da bi se dobila najbolja moguća predikcija za transkodiranje trenutnog kodnog bloka. U slučaju da se detektira da se načini predikcije mapiranih kodnih blokova uvelike razlikuju, postoji mogućnost daljnjeg dijeljenja na manje blokove da bi se ostvarila preciznija predikcija.

Prilikom transkodiranja stanje svih parametara se konstantno prati da bi se osigurala pravovremena izvedba. Nakon svakih nekoliko okvira provjerava se brzina izvođenja, učinkovitost kodiranja te kvaliteta videa. U slučaju da pravovremeno izvođenje nije zadovoljeno, svi koeficijenti se podešavaju da bi se smanjila složenost transkodiranja u sljedećem periodu. U suprotnom slučaju, kada je pravovremeno izvođenje zadovoljeno, razmatra se podešavanje svih koeficijenata u cilju povećanja kvalitete videa i/ili učinkovitosti kodiranja u onim granicama koje ne predstavljaju rizik pravovremenom izvođenju. Stalno praćenje i nadzor svih parametara te podešavanje koeficijenata u cilju postizanja što bolje kvalitete videa bez ugrožavanja pravovremenog izvođenja čini ovaj algoritam otpornim na promjene unutar same video sekvence i unutar sustava na kojem se izvodi videotranskodiranje.

Korištenjem predstavljenog algoritma zadovoljava se pravovremenost u svim slučajevima, a postiže se bolja kvaliteta (PSNR) za sve promatrane scenarije transkodiranja u odnosu na pravovremeni Bolt65 transkoder, i to za 0.788 dB u prosjeku. Učinkovitost kodiranja uvećana je za 27.35% u prosjeku. U usporedbi s Kvaazar transkoderom koji transkodira video bez striktnih vremenskih zahtjeva, predloženi algoritam transkodira s lošijom kvalitetom za 0.749 dB te lošijom učinkovitosti kodiranja za otprilike 30% u prosjeku. Ovi gubici uzrokovani su ograničenjima u brzini izvođenja. Naime, Kvaazar ne zadovoljava pravovremeno transkodiranje ni za jednu testnu video sekvencu, a prosječno ubrzanje koje se dobije korištenjem predloženog algoritma u odnosu na Kvazaar je 2.24 puta.

Programska izvedba opisanog algoritma postiže napredak u odnosu na druge promatrane pravovremene transkodere, međutim, da bi se postiglo videotranskodiranje sa strogim vremenskim ograničenjima nužno je smanjiti set funkcionalnosti koji je moguć pri transkodiranju HEVC standardom. Neki alati, kao što su npr. interpolacijski filteri koji mogu značajno poboljšati kvalitetu videa i učinkovitost kodiranja, se ne koriste s obzirom na to da njihova složenost uvelike utječe na ukupno vrijeme transkodiranja. Uvođenjem sklopovskih

ubrzivača za pojedine jezgre može se ubrzati čitav proces, čime bi se otvorio prostor za uvođenje novih alata ili proširivanje postojećih, što bi poboljšalo kvalitetu transkodiranog videa. Naime, kod pravovremenog videotranskodiranja, krajnji cilj nije izvršiti transkodiranje što je brže moguće, već postići što bolji omjer kvalitete videa i učinkovitosti kodiranja, uz zadovoljavanje postavljenih vremenskih ograničenja.

Analizom HEVC transkodera identificirano je nekoliko jezgri koje najviše utječu na ukupno trajanje izvođenja. Kao najbolji kandidat za sklopovsko ubrzanje u razvijenom algoritmu odabrana je inter predikcija, koja ne samo da utječe na ukupno vrijeme izvođenja nego također uvelike utječe i na kvalitetu krajnjeg video zapisa te učinkovitost kodiranja. Razvijen je specijalizirani sklopovski ubrzivač za FPGA koji postiže ubrzanja od 4 do 13 puta, ovisno o veličini blokova i veličini područja pretrage, u odnosu na čistu programsku implementaciju.

Integracija razvijenog programskog rješenja i specijaliziranog sklopovskog ubrzivača izvedena je na već opisanom sustavu s jednim procesorom opće namjene i Kintex Ultrascale FPGA pločicom na kojoj je postavljen projektirani sklopovski ubrzivač za inter predikciju. Komunikacija između procesora opće namjene i sklopovskog ubrzivača odvija se preko DMA (*eng. Direct Memory Access*) sklopa i AXI sučelja, a svi podaci o video okvirima spremaju se u DDR memoriju. Maksimalna moguća brzina prijenosa koja se može dostići u ovom sustavu iznosi 8 Gb/s.

S obzirom na to da se s integriranim rješenjem dobije ubrzanje videotranskodiranja, višak vremena koji se time dobije može se iskoristiti za proširivanje područja pretrage inter predikcije ili za prilagodbu koeficijenata na način da više kodnih blokova pripada složenijim kategorijama koje se detaljnije obrađuju, čime se u konačnici dobiva na kvaliteti transkodiranog videa i učinkovitosti kodiranja. Ovakvo rješenje integrirano na raznorodnoj arhitekturi visokih performanci postiže bolju kvalitetu u odnosu na Bolt65 pravovremeni transkoder od 0.945 dB, što je rast u odnosu na 0.788 dB koji se dobio samo programskom izvedbom. Učinkovitost kodiranja također raste sa 27.35% na 35.06%. Analogno tome, gubici u odnosu na Kvazaar transkoder, koji ne realizira pravovremeno transkodiranje, su manji i što se tiče kvalitete videa i učinkovitosti kodiranja.

Ključne riječi: videotranskodiranje, HEVC, iskorištavanje informacija o kodiranju ulaznog video toka, sklopovske jezgre za ubrzanje, raznorodne arhitekture visokih performanci

TABLE OF CONTENTS

1 INTRODUCTION

Statistics show that global video IP traffic will be 82 percent of all consumer internet traffic by 2022, up from 75 percent in 2017. Annual global IP traffic will reach 4.8 Zettabytes (1 ZB = 1000 Exabytes) per year by 2022, meaning that approximately 3.9 ZB of video content will cross the network in 2022 [1]. Handling this enormous amount of data is a very challenging task for video content providers in years to come. Another factor that highlights this problem is the fact that the number of different devices that are able to play video content is constantly growing, with the number of mobile-connected devices per capita predicted to reach 1.5 by 2022 [2]. With such diversity of devices, that have different decoding capabilities, computing resources, network bandwidths, and screen resolutions, a single copy of the encoded video cannot efficiently match the requirements of all devices and different playback conditions. For example, serving video with 1080p resolution and 60fps to a low-resolution mobile device with low bandwidth connection would be not only the waste of resources, but it would probably cause playback delays and thereby lower Quality of the Experience (QoE).

Current video providers usually tackle this problem by pre-encoding input video with different configurations and storing multiple copies of the same video on the server [3]. When the user requests the video, the server provides the version that best satisfies the requirements of the end user. Such an approach has very high storage costs, and pre-encoded video streams may still not exactly match end-user requirements. Furthermore, emerging spatial resolutions (4K, 8K, etc.), as well as the long-tail distribution of video content, where 90 percent of videos are viewed by only 10 percent of users and vice versa, make this concept hardly sustainable. Just-in-Time (JiT) video transcoding has one of the key roles in resolving these issues. Video transcoding refers to the problem of adapting on-the-fly Internet video content based on user's device features or specific operational conditions. Adaptation involves changing video properties, such as spatial, temporal and amplitude resolution, bitrate and video format. Instead of storing multiple copies on the server, only one version with the highest quality can be stored and transcoded on demand in real-time.

Although Just-in-Time transcoding increases efficiency while providing the best possible QoE, it is extremely compute-intensive and data-intensive operation. Other approaches that are commonly used by video content providers are Scalable Video Coding (SVC) [4] and newer standard Scalable Extensions of the High Efficiency Video Coding Standard (SHVC)

[5]. SVC and SHVC provide high-quality video stream that contains multiple subset bit streams, representing video with lower quality, that are derived by dropping packets from the larger video to reduce bandwidth. Storing video in one of these formats reduces storage significantly but does not provide the best quality nor best resource management [6]. Some studies combine video transcoding and storing multiple copies in a hybrid system that use previously measured video statistics to determine what videos (or parts of the video) will be transcoded on demand and what videos will be stored in multiple copies ([7],[8],[9]).

High Efficiency Video Coding (HEVC/H.265) standard shows a significant advance in compression efficiency than its predecessor AVC [10]. At the same subjective quality, HEVC saves approximately 50% bitrate but increases computational complexity and resource requirements ten times [11]. Other important aspects that were considered while developing HEVC are its potential for parallel processing and support for hardware implementation. Efficient exploration of these concepts is crucial when it comes to high-performance Just-in-Time video transcoding based on HEVC, especially on heterogeneous many-core architectures.

Just-In-Time video transcoding based on HEVC is extremely computationally expensive and resource-hungry process and achieving the best possible video quality with the lowest possible computational complexity is a topic of numerous researches. Intelligent utilization of coding information extracted from the initial encoded video stream has a key role in almost every study related to this topic. Software algorithms can accomplish significant improvements, but with the strict timing requirements, hardware accelerators and heterogeneous architectures on high-performance computers have to be analysed and exploited. Efficient mapping and optimization of key compute-intensive algorithms to different types of cores to achieve the best trade-off between coding efficiency, video quality and power consumption, while fulfilling real-time constraints and QoS demands is essential.

This thesis investigates novel techniques for Just-in-Time transcoding based on HEVC standard by exploiting data retrieved from the input video stream to increase the speed of the re-encoding operation while trying to preserve video quality and bitrate of the originally encoded bitstream. The exploitation of heterogeneous architectures and performance efficient integration of system architectures composed of all associated modules in order to improve the process of the transcoding is also considered in the scope of this thesis. One of the main goals of the presented research is to contribute to video content server architectures by designing a heterogeneous system that is capable of Just-in-Time transcoding based on HEVC standard.

Previous studies on algorithms and architectures that reuse and utilize coding information from the input video stream, usually cover only a subset of the described research area and rarely cover all aspects: Just-in-Time requirement, execution on heterogeneous high performance computers and transcoding based on HEVC standard.

1.1 Thesis outline

Chapter 2 starts with the introduction to a novel standard for video compression – High Efficiency Video Coding (HEVC or H.265). All the coding tools used in HEVC and their impact on video quality and computational complexity are briefly described.

Chapter 3 gives a brief overview of video transcoding. Several transcoding architectures and techniques used to facilitate the process of video transcoding are also presented along with their advantages and disadvantages. Finally, the specifics and challenges of Just-in-Time transcoding are explained.

In Chapter 4, Bolt65 software/hardware suite consisting of the encoder, decoder and transcoder developed as a part of research activities conducted for this thesis is presented. Special focus is set on the configuration used to achieve Just-in-Time encoding and transcoding.

Chapter 5 describes the methodology and test environment used to evaluate all solutions developed in the scope of this thesis. All data sets used, as well as the system on which integrated solution is ported, are described in more details in chapter 5. Baseline transcoders used to evaluate the final integrated system are also presented.

In Chapter 6 three types of coding information that is reused in the proposed algorithm are defined: the size of decoded coding units, the number of mapped coding units and prediction modes of the mapped coding units.

Concept of categorization based on the coding information presented in Chapter 6 is described in Chapter 7, while the algorithm for Just-in-Time transcoding that processes each coding unit depending on the results of the categorization is presented in Chapter 8.

Evaluation and validation of the proposed algorithm on CPU-only architectures are given in Chapter 9, where the comparison with two baseline transcoders is conducted and presented.

Functionality, design, synthesis, and implementation of a custom hardware accelerator for inter prediction is described in Chapter 10. Analysis and impact of including hardware accelerator in a previously proposed algorithm are also presented.

Performance-efficient integration of implemented software algorithm and custom hardware-based accelerator on high performance computing architecture is given in Chapter 11. The integrated solution is compared with CPU-only implementation of the algorithm as well as with two baseline transcoders.

Finally, Chapter 12 summarizes the achievements of this thesis and proposes some directions for future work in this domain.

2 HIGH EFFICIENCY VIDEO CODING (HEVC) STANDARD

High Efficiency Video Coding (HEVC or H.265) is a video compression standard developed by the Joint Collaborative Team on Video Coding (JCT-VC), a collaboration between Video Coding Experts Group (VCEG) and ITU Telecommunication Standardization Sector (ITU-T) [13]. The HEVC standard enables major advance in compression relative to its predecessors, such as Advanced Video Coding (AVC) or MPEG-2, doubling compression rate of encoded bitstream compared to AVC without sacrificing quality. This compression efficiency of HEVC standard is not accomplished with a single novel compression technique but is a result of multiple contributions in all stages of the encoding process.

2.1 HEVC architecture

Scheme of the HEVC video encoder, which also contains all the building blocks that are incorporated in the decoder and transcoder as well, is given in Figure 2.1

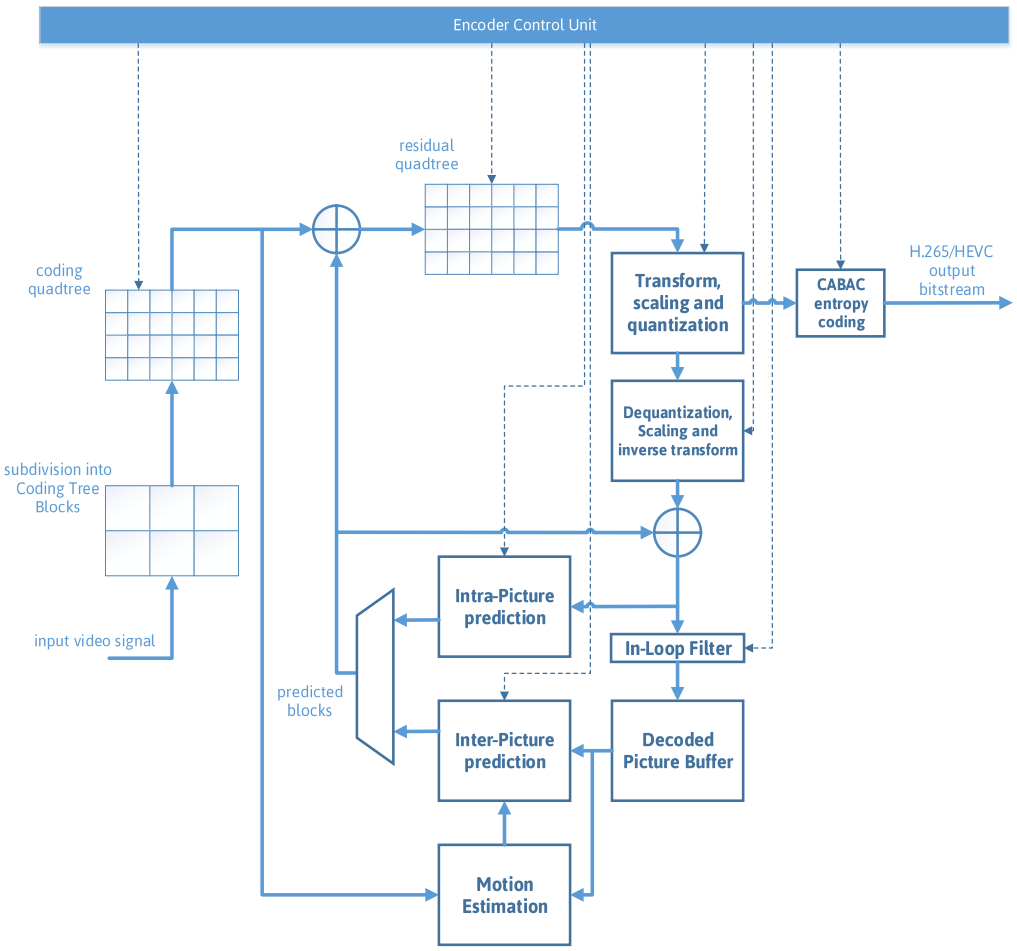

Figure 2.1: HEVC encoder scheme

HEVC follows block-based video coding, where the input frame is first partitioned to smaller blocks and then predicted using intra or inter prediction. The prediction errors or residual, formed as a difference between original and predicted block, is transformed, quantized and finally, entropy encoded into the bitstream. Decoding loop, where quantized values are de-quantized, inverse transformed and stored to the decoded picture buffer is also present in the encoder scheme to obtain a decoded frame for predictions of future frames. Other tools, such as in-loop filtering can be incorporated in the encoding, but can also be skipped, depending on the configuration of the encoder.

Final encoded bitstream must comply with the rules defined in the standard, but the standard itself does not govern the encoding process or the algorithms that are used to form the bitstream. More complex and compute demanding algorithms usually lead to better compression efficiency but at the cost of increased processing time, so finding the best trade-offs between these parameters depends on the system requirements.

All of the building blocks shown in Figure 2.1 are described in more details in the following sections.

2.2 Block partitioning

The first step after fetching the frame from the input video sequence is to divide the frame into smaller square-shaped blocks called Coding Tree Units (CTU) [14]. A CTU represents a basic processing unit in HEVC and all future operations in the encoding process are based on CTU. CTU can be split into more smaller Coding Units (CU) of variable sizes, with a minimum CU size of 8x8 and maximum of 64x64. Each coding unit consists of precisely three Coding Blocks (CB), one luma block and two corresponding chroma blocks. Dividing CTU to multiple smaller CUs follows the quadtree structure as shown in Figure 2.2.

Each leaf CU can act as a root for residual quadtree (RQT). The residual quadtree is a tree of Transform Units (TU) containing Transform Blocks (TB) that can be created to enable the adaptation of the transform functions to the varying space-frequency characteristics of the residual signal. Leaf CUs can also be split to up to four Prediction Units (PU) that can be used for more precise motion estimation.

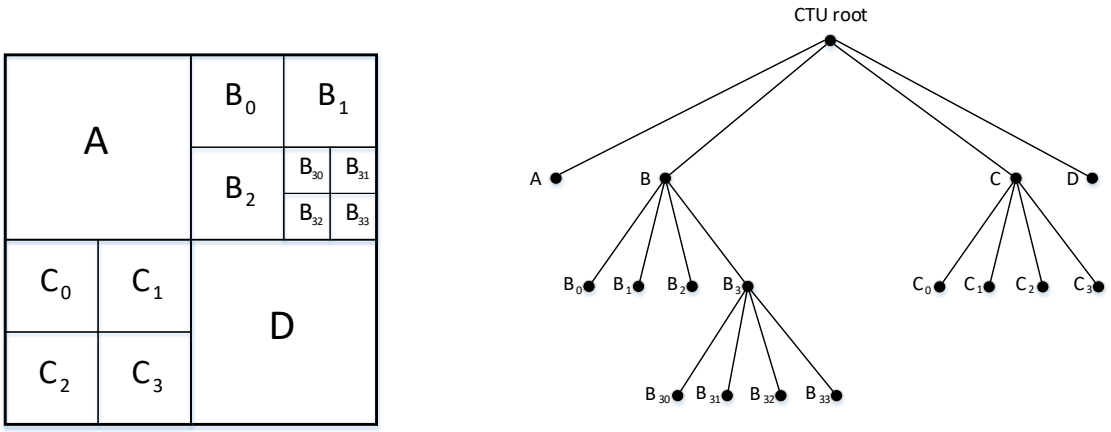

Figure 2.2: Example of partitioning CTU to smaller CUs

A flexible partitioning mechanism that enables variable blocks sizes helps to adapt the encoding process to characteristics of the specific video sequence. More detailed parts of the frame can be divided into smaller blocks to describe that area in more details, while the static parts of the frame can be divided into larger blocks. Partition decisions can also have a notable impact on the quality of the encoded bitstream, as well as on the computational complexity of the encoder. Therefore, when considering Just-in-Time encoding or transcoding, block partitioning has to be taken into account.

2.3 Prediction

Each coding unit can be predicted by exploiting either spatial (intra prediction) [15] or temporal (inter prediction) [16] redundancy in video frames. Difference between the predicted and original block forms a residual that is passed as an input in following steps of the encoder.

Depending on the defined Group of Pictures (GOP), the frame can be intra, or inter predicted. GOP represents a collection of successive pictures within a coded video stream, and it specifies the order in which intra and inter frames are arranged. In the intra frame, all coding units have to be intra predicted. Otherwise, coding units can be either inter or intra predicted, depending on the algorithm that determines prediction modes.

2.3.1 Intra prediction

Intra prediction uses neighbouring pixels from adjacent reconstructed coding blocks within the same frame to calculate the predicted block. To predict different kinds of content, HEVC supports prediction methods that can be classified into two categories: angular intra

prediction methods that accurately model structures with directional edges and Planar and DC predictions that provide an estimation of smooth image content.

Planar and DC predictions can also be used for predicting complex textures that cannot be adequately modeled with any of the angular prediction modes. In the case of DC prediction, predicted block is generated with the constant value obtained as an average of the reference pixels immediately left and to the above of the current block, while the planar mode populates predicted block by averaging horizontal and vertical linear predictions based on reference samples.

HEVC defines a set of 33 angular prediction modes that differ by a direction angle as shown in Figure 2.3. Predicted block in angular modes is generated based on the reference samples and the angle of the prediction mode.

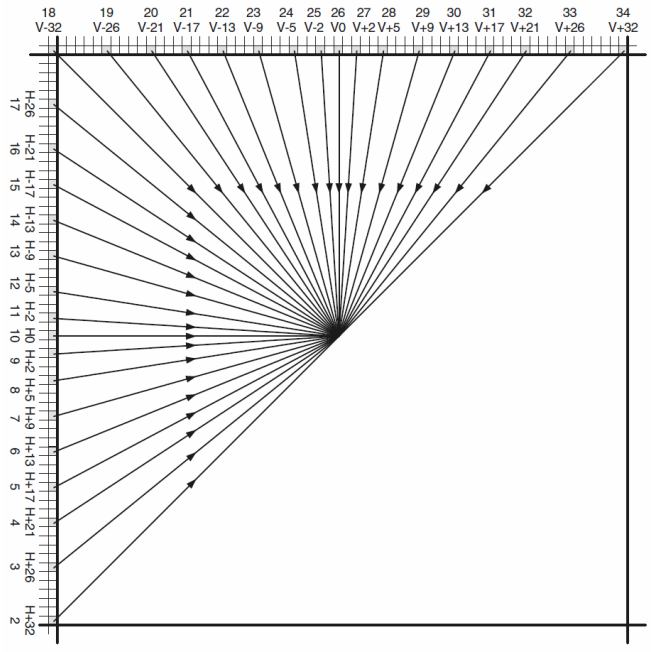

Figure 2.3: Angular intra prediction modes, source [13]

2.3.2 Inter prediction

Predicting the current frame based on previously encoded frames is also known as inter prediction, while the process for finding the block in a reference frame that is the most similar to the current block is called motion estimation. The final result of motion estimation is a motion vector that represents the movement direction of the considered block between the current and reference frame. Concept of inter prediction and the motion vector is depicted in Figure 2.4.

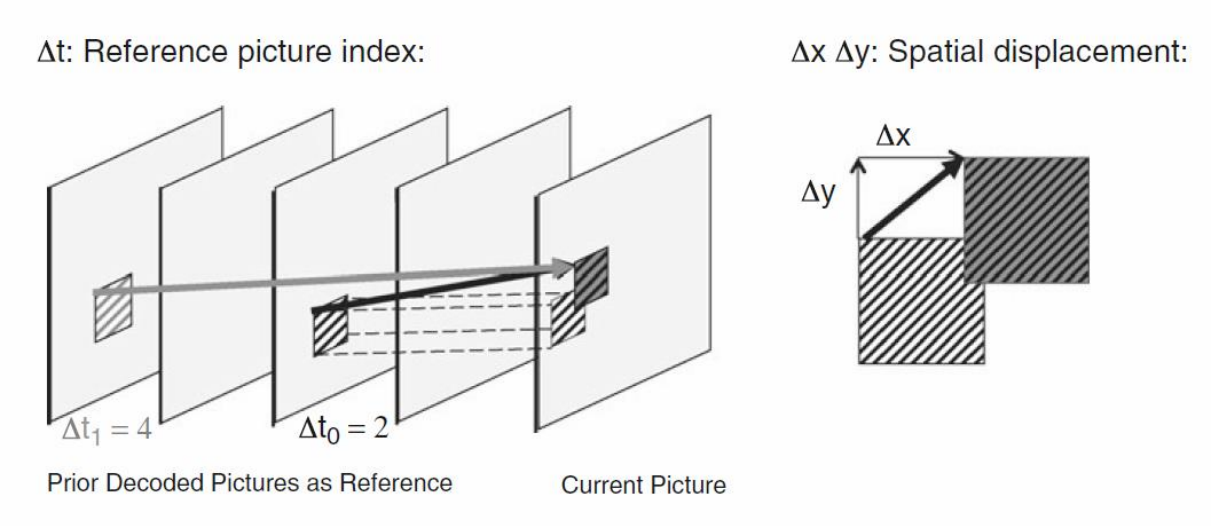

Figure 2.4: Inter prediction concept, source [13]

Finding the best motion vector can have a significant impact on the quality of encoded bitstream and coding efficiency of the encoder. More complex motion estimation algorithms, such as Full Search Motion Estimation (FSME), give the most similar block from the reference frame, forming a low-energy residual that guarantees high video quality and high compression. However, these algorithms consume most of the encoding time [17] and can not be considered in Just-in-Time domain. Therefore, many fast motion estimation algorithms that evaluate a subset of possible motion vectors within the defined search area were developed to cope with this problem, some of which are: Three Step Search (TSS) [18], Diamond Search (DS) [19], Successive Elimination Algorithm (SEA) [20]. Adaptive Seach Windows Size (ASWS) [21] and many others.

Since the real object displacement from one picture to another does not follow the grid structure in a digital representation of the picture, sub-pixel movements are used to capture continuous motions more accurately. HEVC supports quarter-pixel accuracy for luma samples and eight-pixel accuracy for corresponding chroma samples. If the motion vector has sub-pixel accuracy, samples at fractional positions have to be derived from the integer positions using the process called interpolation. The luma interpolation process in HEVC uses an 8-tap filter for half-pixel samples and 7-tap filter for quarter-pixel samples while for chroma component 4-tap filter is used. Including the need for interpolation in the motion estimation can drastically increase computational complexity. Therefore, in Just-in-Time encoding, algorithms for motion estimation usually evaluate only full-pixel motions.

2.4 *Transform and quantization*

The transform is applied to the residual signal resulting from the prediction. Each CU residual block is input to two-dimensional NxN forward transform, which is a separable operation that can be also performed as two one-dimensional transform for each row and column. The resulting transform coefficients are then quantized (i.e., divided with the quantization step - Qstep) to obtain quantized transform coefficients that are used as an input to entropy encoder. To retrieve the reconstructed frame that is stored in Decoded Picture Buffer (DBP) for future inter predictions, each block has to be de-quantized and inverse transformed as well. This process in the encoder and decoder is shown in Figure 2.5.

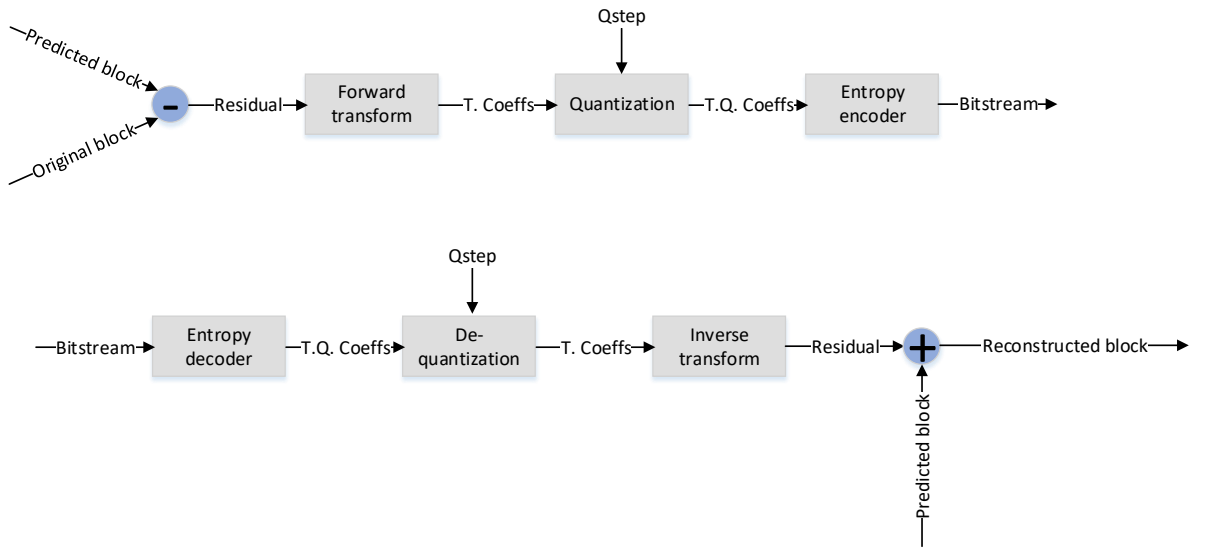

Figure 2.5: Transform and quantization process in encoder and decoder

For the transformation purposes, Discrete Cosine Transform (DCT) is used for most of the blocks in HEVC standard. In the original form, the DCT uses floating point operation which increases computational complexity and errors between the forward and inverse transforms. Therefore, HEVC specifies two-dimensional finite precision integer approximation of DCT transform, referred to as core transform. The core transform specifies the kernel matrices for each block size, designed to enable efficient implementation in both, software and hardware. To achieve more optimal de-correlation of the residual input block, HEVC also specifies alternate transform based on Discrete Sine Transform (DST), which is used exclusively for 4x4 luma blocks.

A quantization process is performed based on the quantization parameter (QP) set as an input to the encoder. Depending on the QP, that can be in the range from 0 to 51 inclusive, for 8-bit pixel samples, Qstep is calculated.

2.5 In-Lop filters

There are two types of in-loop filters defined in HEVC standard: Deblocking filter [22] and Sample adaptive offset (SAO) filter [23]. Both filters are applied in the encoding and decoding loops, deblocking filter, if enabled, first and then SAO, before storing the frame in DPB. The main goal of in-loop filters is to increase the subjective quality of reconstructed pictures by smoothening the artifacts that can appear on the block boundaries.

When two neighboring blocks are predicted from the non-adjacent blocks in the reference frame, an artifact may appear on the boundary of the two blocks. The deblocking filter attenuates this appearance by averaging pixel values near the block boundaries. Example of the deblocking filter is given in Figure 2.6, where pixels A0-A3 belong to a row in the first block, while pixels B0-B3 belong to a row in an adjacent block. The dotted line shows the pixel adaptation by using a deblocking filter.

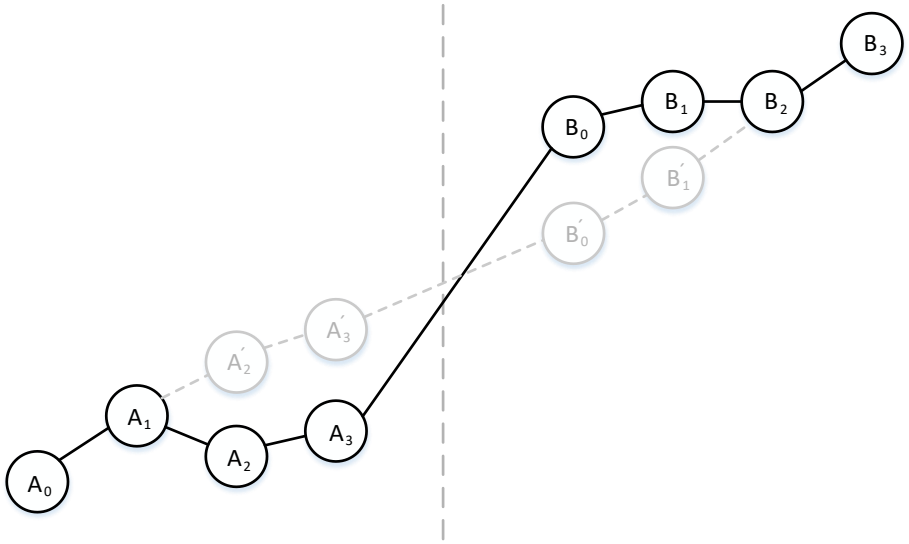

Figure 2.6: Deblocking filter example

The main goal of the SAO filter is to smooth the ringing artifacts and changes in sample intensity of some areas of the picture that can appear when transforming lager blocks.

Using in-loop filters increases the quality of the encoded video and coding efficiency, but at the same time introduces additional operation that has to be performed. Although the

deblocking filter is not as complex as some other operations, it can affect strict timing requirements set in Just-In-Time systems, which has to be taken into consideration.

2.6 HEVC syntax and entropy coding

In HEVC, high-level syntax describes the structure of the bitstream which includes the signaling of high-level information that applies to one or more slices. HEVC bitstream consists of a sequence of data units called a network abstraction layer (NAL) units. Some NAL units carry parameter sets containing control information, while other carry coded segments of an individual picture. Each picture is partitioned into one or multiple slices of which each one is independent of others. A slice consists of one or multiple slice segments where only the first one is independent while others depend on previous slice segments. Each coded slice segment consists of a slice segment header with the control information followed by slice segment data with the coded samples.

After the video input has been converted to a series of syntax elements, entropy coding is performed. Context-Based Adaptive Binary Arithmetic Coding (CABAC) is a method of entropy coding used in HEVC [24]. CABAC is a lossless compression scheme that uses the statistical properties to compress data. In HEVC bitstream, only syntax elements belonging to the slice segment data are CABAC coded, while others are coded either with zero-order Exponential (Exp)-Golomb codes or fixed-pattern bit strings. Key elements of the basic CABAC design are binarization, context modeling, and binary arithmetic coding [25].

2.7 HEVC parallelization

HEVC introduces two novel parallelization concepts, Tiles [26] and Wavefront parallel processing, along with the slices, which were also available in previous AVC standard. Tiles, performance-wise, outperform other parallelization concepts in HEVC, so for Just-in-Time encoding, the focus is set on efficient implementation and usage of tile mechanism.

Tiles are rectangular-shaped groups of CTUs that divide the frame based on set vertical and horizontal boundaries. Each tile can be encoded independently, without the need for the communication between the units that process different tiles. Dividing the frame to multiple tiles can be done in two ways: uniformly, where the boundaries are set so that each tile has approximately the same the number of CTUs, and non-uniformly with arbitrarily defined tile boundaries. Example of dividing one frame to nine (3x3) uniform tiles is depicted in Figure 2.7.

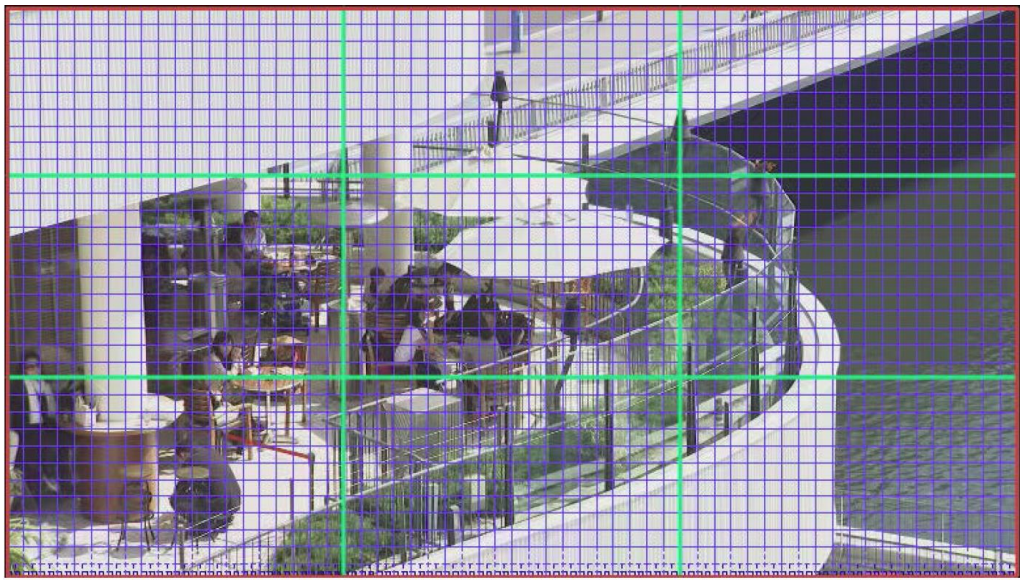

Figure 2.7: Dividing the frame to 3x3 uniform tiles

The number of tiles and tile distribution can be set for each frame individually, which enables adaptation of tile structure during the encoding process. This fact can be used on heterogeneous multicores systems to enhance load balancing between processing cores. Previous research in this field includes algorithms that calculate the time needed to process each tile in the previously encoded frame and based on that information dynamically adapt tile boundaries [27] [28]. This concept, however, is not ideal for heterogeneous architectures, where processing time highly depends on the processing core type. Therefore, a novel algorithm that overcomes this issue and approximates the computational complexity of each tile is developed as a part of the research conducted for this thesis and is presented in [29].

3 VIDEO TRANSCODING

Video transcoding is a process of converting video sequence from one format to another. A video format is defined by several characteristics, such as bitrate, frame rate, spatial resolution, and coding standard [30]. The main goal of video transcoding is to adapt the original video to specific end-user requirements in order to provide the best quality of experience. Transcoding also enables multimedia devices of diverse capabilities and formats to exchange video content on heterogeneous network platforms, which is extremely important in today's world, where the number and diversity of devices that are able to play video content continually increases. The parameters of the transcoded video can depend on multiple factors, such as network bandwidth, client's device capabilities (computing resources, display resolution, power consumption) and the limits of the human visual system (HVS). Some of the conditions can vary during the streaming process.

A transcoder is achieved as a cascade of a decoder, followed by an encoder. Input for the transcoder is an encoded video bitstream, which is then decoded or partially decoded and re-encoded to obtain transcoded bitstream. Example of homogeneous transcoding, where the video is transcoded to the same standard (HEVC) is shown in Figure 3.1. Only homogeneous, HEVC based transcodings are considered in this thesis.

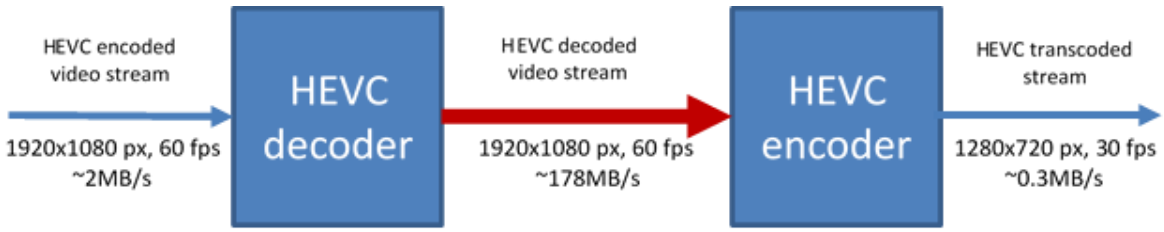

Figure 3.1: HEVC transcoder scheme example

The figure above shows an example of spatial and temporal reduction by transcoding from 1920x1280 resolution with 60 frames per second (fps) to 1280x720 resolution with 30 frames per second. By decoding compressed video stream, raw video data is retrieved and re-encoded with new parameters to form HEVC transcoded bitstream at the output. Along with the inputs and outputs to the transcoder, throughputs needed to satisfy Just-in-Time transcoding are also depicted in the figure.

A straightforward forward transcoder that fully decodes and re-encodes video sequence is extremely compute and data-intensive process. Therefore, different architectures and techniques are considered to facilitate the process of transcoding [31].

3.1 Transcoding architectures

Reducing the complexity of the straightforward realization of the transcoder is driving force behind most of the research activities related to video transcoding. The challenge is how to intelligently utilize the coding statistics and parameters that can be easily obtained from the input bitstream to achieve the best possible video quality and the lowest possible computational complexity.

Generally, there are three main transcoding architectures: open-loop transcoder [32], cascaded pixel-domain transcoder (CDPT) [33] and DCT-domain transcoder (DDT) [34]. Hybrid-domain and simplified transcoding architectures are usually derived from these three types in order to achieve trade-offs between computational complexity and picture quality.

3.1.1 Open-loop transcoder

The open-loop transcoder is simplest and computationally most efficient architecture. In the scheme, shown in Figure 3.2, after variable length decoding (i.e., entropy decoding in terms of HEVC standard) the quantized coefficients are inverse quantized and then re-quantized to satisfy the new output bit rate. Finally, the re-quantized coefficients are again variable length coded to get output video stream.

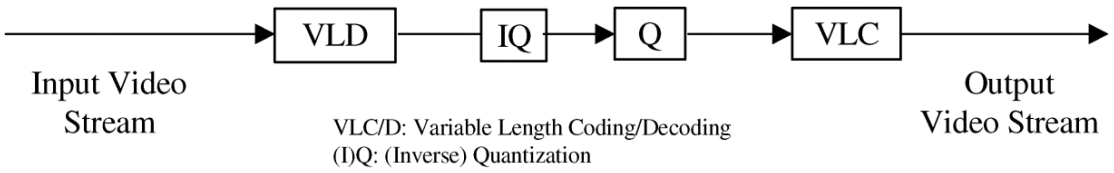

Figure 3.2: Open-loop transcoder architecture, source [30]

The open loop architectures include selective transmission, where high-frequency transform coefficients are discarded and re-quantization, where the transform coefficients are re-quantized with different quantization step. The most significant advantage of this architecture is that it is computationally efficient, but it suffers from the drift problem.

Drift can be explained as the blurring or smoothing of successively predicted frames. Video picture or a frame is predicted from its reference pictures, and only prediction errors are coded. For the decoder to work properly, reference pictures stored in decoder must be the same as those in the encoder. Otherwise, predicted frames wouldn't be the same as the original. Since open-loop transcoders change the prediction errors by re-quantizing the stream, decoder and encoder do not have same reference pictures stored in the buffer, which causes error accumulation and may cause severe degradation to the video quality. Since intra pictures are not predicted from reference pictures but are coded independently, drift will be terminated with intra pictures. For the applications where two intra pictures are relatively close in GOP structure drift could be tolerated, especially if complexity reduction is a priority. Studies in [35] and [36] show that drifting error in open-loop architectures can be reduced.

3.1.2 Cascaded pixel-domain transcoder

Unlike open-loop transcoders, cascade pixel domain transcoder (CPDT) is drift-free architecture. CPDT decodes the original signal, performs the appropriate intermediate processing and then re-encodes processed signal with new constraints. This operation is very compute-intensive, so research activities in this domain mainly focus on reducing the complexity while achieving minimal degradation of video quality.

Figure 3.3 illustrates a scheme of CPDT architecture.

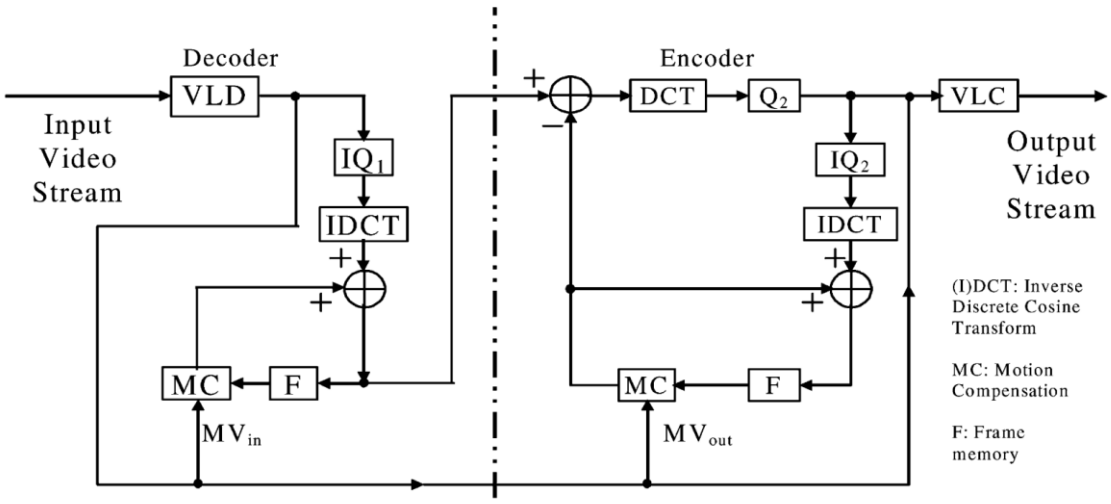

Figure 3.3: CPDT architecture, source [30]

To reduce the complexity of the full-scale transcoder, information extracted from the input video stream, such as motion vectors, can be reused and adapted for the re-encoding.

Reusing data from the input video stream in order to achieve Just-in-Time transcoding is the main focus of this thesis, so all the data that is being reused and the methods for reusing are explained in more details in following chapters.

3.1.3 DCT – domain transcoder

Besides motion estimation, one of the most compute expensive operations in video encoding is the DCT transform. In DCT-domain transform architecture, only syntax decoding and inverse quantization are performed on the decoder side. The reference frame buffer in the encoder stores DCT values after inverse quantization. These values are then used for frequency-domain motion compensation module using a motion vector reusing algorithm. Motion compensated residue errors are then encoded through re-quantization and variable length coding. Although less computation is achieved by avoiding DCT/IDCT transform operation, DDT architecture suffers from the drift problem. The simplified DCT-domain transcoder (Figure 3.4) assumes that DCT, IDCT, and MC are linear operations, reducing complexity at the expense of picture quality.

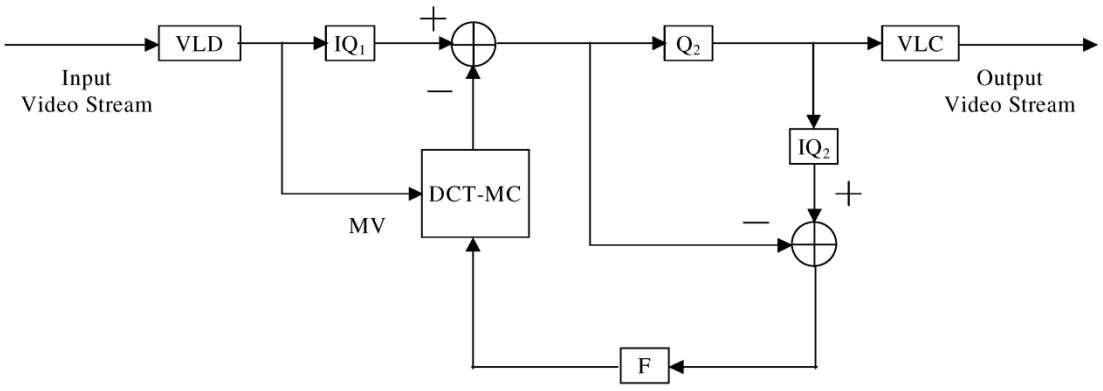

Figure 3.4: DCT - domain transcoder architecture, source [30]

3.2 Transcoding techniques

The initial need for transcoding was to reduce bitrate to meet the available network capacity. With a large number of different devices with limited displays and processing power that started to reproduce video content, transcoding for spatial and temporal resolution adaptation increased. Transcoding for error-resilience is used to gain robustness of video streaming, especially over mobile access networks.

3.2.1 Bitrate reduction

One of the main objectives of transcoding is to reduce the bitrate, but at the same time to maintain video quality as high as possible. Since the spatial and temporal reduction obviously reduce bitrate, the focus is on techniques that reduce bit rate but keep the same spatial and temporal resolution. There are two techniques that can be used for this purpose: re-quantization and selective transmission [37].

Re-quantization performs quantization with the increased quantization step at the encoder. This approach decreases the number of non-zero coefficients, thus reducing the number of bits needed to encode outgoing bitstream. Selective transmission explained in open-loop transcoder architectures can also reduce bitrate by discarding some of the higher frequency coefficients.

3.2.2 Temporal resolution reduction

Reducing the temporal resolution is achieved by dropping certain number of frames from the original video stream. It may be used to reduce the bitrate requirements imposed by a network while maintaining a higher quality of encoded frames or in cases when the end-system supports only a lower frame rate.

With frame dropping, motion vectors extracted from the decoded frame cannot be directly re-used, since they can point to a reference frame that does not exist in the transcoded video. Therefore, motion vectors for re-encoding have to be derived from the input motion vectors. There are several algorithms for reusing motion vector in temporal reduction, such as Forward Dominant Vector Selection (FDVS) [38] or Telescopic Vector Composition (TVC) [39].

Some coding standards, such as HEVC or AVC, include temporal scalability, where parts of the stream can be removed in a way that resulting substream forms another valid bitstream that represents original video content with the lower frame rate. In HEVC, this is achieved by denoting each NAL unit that contains frame information with the temporal sub-layer id. A simple example of temporal scalability with two temporal layers is depicted in Figure 3.5.

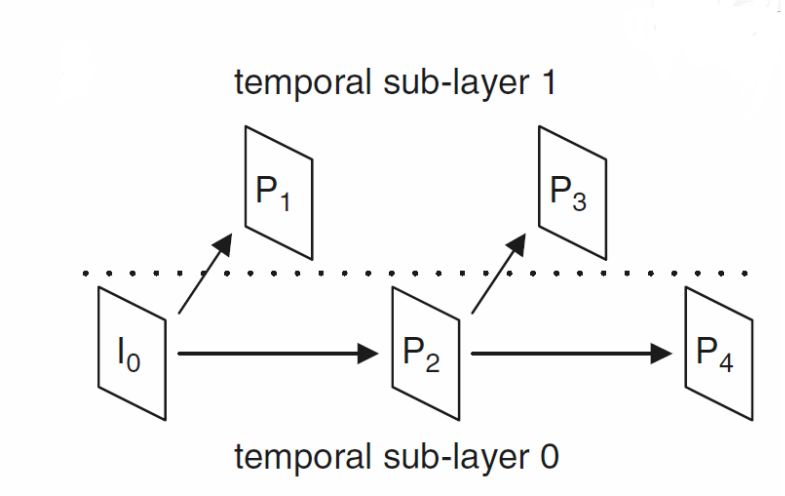

Figure 3.5: Temporal scalability in HEVC, source [13]

Frames have to be arranged in a GOP structure so that frames in any of the lower temporal sub-layers do not have any dependencies on frames in higher temporal sub-layers. This fact can also be seen in the figure above, where frames P_2 and P_4 depend only on frames within the same sub-layer. Since the temporal sub-layer 0 has no dependencies from sub-layer 1, it is possible to remove higher sub-layer without any consequences or needs for motion vector adaptation. Since the temporal resolution reduction in transcoding can be efficiently solved by using temporal scalability in the HEVC standard, the focus of this thesis is set exclusively on spatial resolution reduction.

3.2.3 Spatial resolution reduction

The original video is usually captured at a high spatial resolution and as such, stored on the server. With the emergence of mobile devices that are capable of playing video content, there is a strong need for efficient techniques for spatial resolution reduction.

When changing the resolution of the original picture, the pixels of the downsized frame have to be generated by subsampling original pixels. Several techniques are commonly used for image scaling: Filtering and subsampling, pixel averaging [40], bilinear or bicubic interpolation [41] and nearest neighbor. Another problem that arises when reducing spatial resolution is reusing motion vectors, since the original motion vectors were obtained for the higher resolution frame. This problem is visualized in Figure 3.6 for the downsizing factor of 2, where four blocks in the original picture are mapped to one block in the downsized picture.

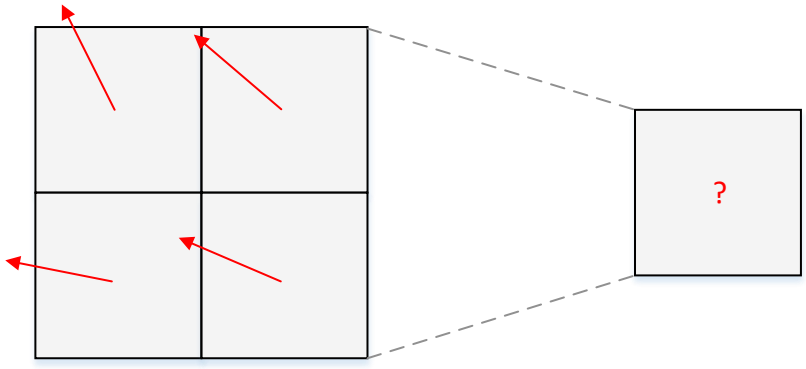

Figure 3.6: Vector remapping problem in spatial reduction transcoding

Several standard methods are used to calculate the transcoded motion vector [42]:

- Simple average – an average of all original motion vectors
- Weighted average – an average of original motion vectors based on the ratio of the original block within a transcoded block
- Area-weighted average – an average of original motion vectors based on the ratio and size of the original block within a transcoded block
- Maximum area – takes the motion vector of the original block with the largest area within a transcoded block
- Median – a median of all original motion vectors

Other techniques for motion vector remapping are usually derived as a combination or adaptation of one or more of the listed methods.

Similar as for temporal scalability, HEVC standard introduces Scalable High Efficiency Video Coding (SHVC) extension that enables simultaneous encoding in multiple layers, each with different spatial resolution [5]. With this extension, multiple versions of the same video with different spatial resolution can be stored as a single file on a server. When end-user requests the video, sub-bitstream with most suitable resolution will be sent to the user. However, SHVC contains only several predefined spatial resolutions while the transcoding offers output videos in any arbitrary resolution. Also, storing original video in this format increases storage costs by 30 % due to layering overhead, compared with storing only the video with the highest resolution [5].

3.2.4 Information insertion

Transcoding can also be used to insert additional information to the output video stream. For copyright protection, video watermarks or company logos can be added. Since logo affects only a part of the video picture, incoming motion vectors can be reused for parts of pictures unaffected by the logo, while others have to be modified. Efficient architectures for logo and watermark insertion are analyzed in [43], [44].

In practical applications, video transcoder can be placed in a network node, connected to a high-loss network to insert error-resilience features. The transcoder first extracts video features from the incoming bitstream and estimates client channel conditions based on feedback channel statistics. These features are then used to determine error-resilience policy. Other error-resilience architectures and techniques are presented in [45] and [46].

3.2.5 Standard transcoding

In many applications, video coded in one standard (e.g., AVC) has to be converted to another standard (e.g., HEVC). This type of transcoding is referred to as heterogeneous transcoding. Heterogeneous transcoding is often needed when the end device supports a standard that is different than the one stored on the server. The main challenge in cross standard transcoding is syntax translation between different formats with minimal influence on quality.

However, the focus of this theses is set on homogeneous video transcoding based on HEVC standard.

3.3 Just-in-Time video transcoding

Most of today's video content providers use hybrid architectures that combine storing multiple versions of the same video sequence with video transcoding on demand to balance the costs of content storage, power consumption and transport. Most popular and frequently accessed content is stored in multiple versions to avoid constant transcoding. However, video content stored on a server follows a long-tail distribution, meaning that the vast majority of viewers watch a very small portion of the stored content, which leaves a large amount of video content eating up storage resources. Therefore, infrequently requested content is usually stored only in the highest quality representation, while the lower quality versions are generated on the fly at the moment of request. When generating content at the moment of the request, video transcoding must be done in real time or faster, to serve the end-user without the lag. Video

transcoding restricted with this timing requirement is referred to as Just-in-Time video transcoding. Compared with upfront transcoding, JiT transcoding is done on every request and is computationally much more expensive, but can provide better overall system efficiency.

A lot of the research in the area of video transcoding is set on speeding up the process of transcoding, but very few of them aim at Just-in-Time transcoding. Authors in [47] provide a fast transcoding solution using a control stream that assumes that different predefined versions of the same video content are available. This approach increases storage costs compared to storing only one version of the video and does not aim to satisfy Just-in-Time requirements. Guided Just-in-Time transcoding architecture for cloud-based video platforms is presented in [48], where the basic idea is to execute the most complex part of the transcoding, such as motion estimation, upfront for each representation below the highest available and to store only information about obtained motion vectors. When the video with lower resolution is requested, the original bitstream is transcoded based on previously calculated motion vectors. This approach increases storage costs since it stores information for different representations of the same video and does not enable transcoding with arbitrary transcoding ratio. Research conducted in [49] proposes several optimized transrating techniques for HEVC. However, only bitrate reduction without frame downscaling was considered without Just-in-Time requirements.

The approach presented in this thesis aims at Just-in-Time video transcoding that enables arbitrary ratio downscaling without any increase in storage cost.

4 BOLT65 SOFTWARE/HARDWARE SUITE

Bolt65 is performance-optimized HEVC hardware/software suite for Just-in-Time video processing developed as a part of the research activities conducted for this thesis [50]. Bolt65 is „clean-room“ suite that consists of an encoder, decoder, and transcoder based on HEVC standard. Special focus in the development of the Bolt65 was set on the performance-efficiency achieved by low-level optimizations and hardware-software co-design adapted for the efficient exploitation of heterogeneous accelerator-based architectures. Another important focus of Bolt65 is the just-in-time processing requirement which sets constraints on processing time making Bolt65 suitable for encoding/transcoding on demand.

A novel algorithm for reusing coding information from the input video stream presented in this thesis is incorporated and tested within Bolt65 software/hardware suite.

This chapter provides a brief overview of all the tools and techniques used in HEVC encoder, decoder, and transcoder developed in the scope of Bolt65 suite.

4.1 Configuration

Configuration for Bolt65 encoder, decoder and transcoder can be defined in two ways before running the application: through configuration file which contains all necessary data or through the console, where each parameter can be set individually. Example of the configuration file used for transcoding BasketballDrive video sequence from the original resolution of 1920x1080 to 1280x720 is given below.

```
NumberOfFrames                :     500
InputFile                     :     BasketballDrive_1920x1080_50p.hevc
OutputFile                    :     BasketballDrive_1280x720_25p.hevc
PictureWidth                  :     1280
PictureHeight                 :     720
FrameRate                     :     50
QP                            :     32
BitNumber                     :     8
CtbLog2SizeY                  :     6
MinCbLog2SizeY                :     3
DeblockingFilter              :     0
SAOFilter                     :     0
DPBSize                       :     1
GOP                           :     IPPPPPPPPPP
SearchAlgorithm               :     3
SearchArea                    :     6
InterpolateAlgorithm          :     0
BlockMatching                 :     0
```

```
#Monitor variables
StatMode                        :     1
ShowStatisticsPerFrame          :     1
ShowStatisticsPerTile           :     0
CalculatePSNR                   :     1
CalculateBitsPerFrame           :     1
CalculateProcessingTime         :     1
CSV                             :     performance.csv

UsePolicy                       :     0
PolicySocketHost                :     localhost
PolicySocketPortNumber          :     5717

Threads                         :     4

TilesEnabled                    :     1
TilesInRow                      :     2
TilesInColumn                   :     2
TileLoadBalancingAlgorithm      :     0
TileLoadBalancingInterval       :     0

AVX                             :     1
DCT_HW_ACC                      :     0
INTER_HW_ACC                    :     0
```

Meaning and the usage of parameters defined in the configuration file above is explained in the following chapters.

4.2 HEVC implementation

The input to the encoder is raw video stream that is being encoded to the final HEVC bitstream. Two types of files that represent raw video stream are supported by the Bolt65 encoder: YUV and Y4M. Both of these types contain raw pixel data, with the difference that Y4M file also contains a header with additional information about the video, such as frame width, height or frame rate. Input video data with 8 bits per sample and the representation with luma brightness signal and two chroma channels that have half the luma resolution both horizontally and vertically (color space 4:2:0) is supported.

Reading large input files frame by frame is a demanding task that can become a bottleneck in a process bounded by strict timing requirements. Therefore, prefetching of the frames from the input file into a specific buffer is implemented in Bolt65 to avoid waiting for the frame data to be loaded from the source after encoding each frame. Input and output parameters are defined in configuration file with options: *InputFile*, *OutputFile*, *PictureWidth*, *PictureHeight*, *FrameRate* and *BitNumber*.

Bolt65 supports CTU sizes of NxN, where N∈{16,32,64}, resulting luma CTB size of NxN and chroma CTB sizes of (N/2xN/2) due to 4:2:0 color subsampling. CTUs can be divided into four smaller units CUs following a quadtree structure. CU supported by Bolt65 are NxN, with $N_{min} \leq N \leq N_{max}$ and N∈{8,16,32}. N_{min} and N_{max} can be defined in configuration (*CtbLog2SizeY* and *MinCbLogSizeY*), if not, default values of N_{min} =8 and N_{max} =32 are used. Each CU can act as a root for residual quadtree that is made of TUs with corresponding TBs. In Bolt65, luma TB sizes are MxM, while chroma TB sizes are (M/2)x(M/2), where M≤N and M∈{4,8,16,32}, N being the size of RQT root. Leaf CUs can also be split to up to four PUs that can be used for more precise motion estimation. HEVC standard supports 8 partitioning modes for splitting CU to PU: PxP, Px(P/2), (P/2)xP, (P/2)x(P/2), Px(P/4), Px(3P/4), (P/4)xP and (3P/4)xP, all of which are also supported in Bolt65.

All 35 (33 angular, DC and Planar) intra prediction modes are supported in Bolt65. The distortion between original and predicted block can be evaluated by several algorithms (determined with *BlockMatching* flag): Sum of absolute differences (SAD), Sum of absolute transform differences (SATD) or Mean Square Error (MSE). Different algorithms for block matching have different effects on coding efficiency and computational complexity, which is an important aspect that has to be considered in Just-in-Time transcoding. Motion estimation algorithms that are supported and can be used in inter prediction are (determined with *SearchAlgorithm* and *SearchArea* flags): Three Step Search (TSS), Diamond Search (DS) and Full Search Motion Estimation (FSME). For Just-in-Time transcoding fast TSS algorithm that evaluates only blocks moved by integer motion vectors to avoid interpolation is used. Only P frames are supported in Bolt65, while bi-predictive B frames that use two reference frames for prediction are not implemented due to increased computational complexity induced by motion estimation on two separate reference frames. After the prediction residual is transformed and quantized based on RQT partitioning.

In-loop filters can be disabled or enabled by setting the flags *DeblockingFilter* and *SAOFilter* either to 0 (disabled) or 1 (enabled).

4.3 Bolt65 on heterogeneous architectures

In the scope of the Bolt65 suite, along with the CPU implementation of HEVC codec, several kernels were offloaded to different processing nodes and integrated into one heterogeneous system.

Advanced Vector Extensions (AVX, AVX2) are extensions to the x86 instruction set architectures for processors from Intel that can support 128 and 256-bit SIMD vector instructions [51]. Two kernels were implemented in AVX2 in Bolt65: Sum of Absolute differences, and integer DCT. Both kernels were chosen for implementation on vector extensions due to their highly parallelizable nature. Integer DCT transformation for HEVC is based on matrix multiplication, where more than one element can be calculated in parallel. Similarly, in the SAD kernel, the 256-bit vector can be utilized to conduct 32 substractions of 8-bit pixel samples in parallel. Average speedup in the encoding time when SAD and DCT are ported to AVX2 compared to implementation without AVX ranges between 78% and 160% depending on the quantization parameter.

A high-throughput fully pipelined FPGA-based accelerator for HEVC DCT has also been designed and implemented. The architecture consists of two cascaded 1D DCT cores with a constant throughput of 32 pixels per cycle in all size modes. The accelerator receives the data through a 512-bit bus which enables fetching 32 16-bit samples in a single clock. The pipeline for worst-case scenario consists of 75 stages which means that for processing a single 32x32 matrix, 106 cycles are necessary. The efficient integration of the custom DCT hardware accelerator in a novel heterogeneous MANGO platform [12] is presented in [52].

Custom hardware-based accelerator for inter prediction, designed and implemented to enhance software algorithm presented in this thesis is described in chapter 10.

4.4 Monitoring and statistics

During the execution of the application, whether it is an encoder, decoder, or transcoder, several parameters can be monitored. Those parameters include processing time to observe execution speed, bitrate to analyze coding efficiency and PSNR for video quality. Depending on the user requirements, monitoring can be performed on tile, frame or video level.

In the algorithm proposed in this thesis, monitoring has one of the critical roles in achieving Just-in-Time execution, while trying to maintain video quality and coding efficiency of the input bitstream.

Bolt65 encoder and transcoder execution can be controlled during runtime by an external process that can monitor the current state of the encoding/transcoding. Changing some of the parameters, such as quantization parameter, search algorithm or GOP structure can be used to increase or decrease coding efficiency, power consumption or performance, depending

on the current requirements posed by an external factor of the system. Hence, different policies can be implemented and used while running the application, without the need for modification of existing source code.

4.5 Parallelization techniques

Bolt65 implements support for tiles, a novel parallelization concept introduced in HEVC. Both, uniform and non-uniform tile distribution are available by setting the options *TilesEnabled*, *TilesInRow,* and *TilesInColumn* to appropriate values in the configuration file. Ordinarily, each tile is processed on a separate core, where the number of processing cores equals the number of tiles in a frame. However, this does not always have to be the case. When a number of available processing cores in a system is smaller than a number of tiles, smart load balancing is performed so that the waiting time between the processing cores is the smallest possible. Algorithm for dynamic load balancing in an encoding where the number of heterogeneous cores is the same as the number of tiles was developed in the scope of research activities for this thesis and is presented in [29].

5 METHODOLOGY AND TEST ENVIRONMENT

This chapter describes the test environment and methodology used for the validation of developed algorithms throughout this thesis.

5.1 Test video sequences

For all experiments conducted in this thesis, the same test set of videos, with different resolutions and frames rates is used, as shown in Table 5.1. Original videos with resolution lower than 1280x720 were not considered since the complexity of their transcoding is much smaller. Besides that, today's video content providers usually have an original video sequence stored in one of the higher resolutions. All of the test video sequences can be downloaded from the internet [53][54] and used for testing.

Table 5.1: Set of test video sequences

#	Video name	Resolution	Frames	Frame rate
1	Shields	1280x720	504	50
2	ParkRun	1280x720	504	50
3	KristenAndSara	1280x720	600	60
4	Johnny	1280x720	600	60
5	FourPeople	1280x720	600	60
6	BasketballDrive	1920x1080	500	50
7	Calendar	1920x1080	500	50
8	Cactus	1920x1080	500	50
9	BQTerrace	1920x1080	600	60
10	RushHour	1920x1080	500	25
11	Riverbed	1920x1080	250	25
12	PedestrianArea	1920x1080	375	25
13	BlueSky	1920x1080	217	25
14	Traffic	2560x1600	150	30
15	DuckTakeOff	3840x2160	500	50
16	Bosphorus	3840x2160	600	120
17	Beauty	3840x2160	600	120

Although the proposed algorithm is designed for arbitrary transcoding ratio downscaling, several fixed standard resolutions were chosen for testing. Resolutions were selected so that they can represent a broader range of possible combinations, with different width and height downsizing ratios (width ratio ρ_w and height ratio ρ_h) that do not have to keep the same aspect ratio. All possible combinations of transcoding along with the downsizing width and height ratios are shown in Table 5.2.

Table 5.2: Possible transcoding scenarios

Original resolution	Possible transcoded resolutions
3840x2160	2560x1600 (ρ_w=1.5, ρ_h=1.35) 1920x1080 (ρ_w=2, ρ_h=2) 1280x720 (ρ_w=3, ρ_h=3) 704x576 (ρ_w=5.45, ρ_h=3.75) 640x480 (ρ_w=6, ρ_h=4.5)
2560x1600	1920x1080 (ρ_w=1.33, ρ_h=1.48) 1280x720 (ρ_w=2, ρ_h=2.22) 704x576 (ρ_w=3.63, ρ_h=2.77) 640x480 (ρ_w=4, ρ_h=3.33)
1920x1080	1280x720 (ρ_w=1.5, ρ_h=1.5) 704x576 (ρ_w=2.72, ρ_h=1.875) 640x480 (ρ_w=3, ρ_h=2.25)
1280x720	704x576 (ρ_w=1.82, ρ_h=1.25) 640x480 (ρ_w=2, ρ_h=1.5)

5.2 Heterogeneous processing environment

The heterogeneous processing system used in this thesis consisted of a host side with CPU and the Kintex Ultrascale FPGA [55] on proFPGA quad Motherboard [56]. The host side and FPGA were connected via fast PCIe interconnect (gen 3, 8-line). Characteristics of the host and FPGA are given in Table 5.3 and Table 5.4 respectively.

Table 5.3: Host characteristics

Processor	Intel i5 4570 (3.2 GHz)
Memory	32 GB
L1 cache	32kB
L2 cache	256 kB
Compiler	Intel C++ 18.0
Operating system	64-bit Windows 10 Pro

Table 5.4: FPGA characteristics

System Logic Cells (K)	1451
DSP slices	5520
Block RAM (Mb)	75.9
16.6Gb/s Transceivers	64
I/O pins	832

5.3 Baseline transcoders

Validation of the proposed algorithm is done by comparing the results with two different baseline transcoders. First one is the JiT transcoder based on Bolt65 software implementation that re-encodes video sequence in real-time without reusing any of the data from the decoded frame (described in chapter 4). Another baseline transcoder is based on the open-source Kvazaar [57] encoder that encodes video sequences without strict timing requirements. Comparisons with referent HM decoder/encoder [58] were not presented in the final validation since the difference in processing times is substantial [50] and cannot be compared when it comes to JiT transcoding.

5.3.1 Bolt Just-in-Time transcoder

Bolt65 described in chapter 4 is a HEVC encoder/transcoder that was explicitly designed for Just-in-Time encoding and transcoding. Bolt65 Just-in-Time transcoder that will be referred to as Bolt65 JiT in the rest of this thesis, does not use any data reusing algorithms from the decoded frame. Instead, the decoded frame is downsized and re-encoded from scratch. Comparisons with this type of transcoder help to comprehend gains of using the proposed data reuse algorithm in terms of video quality and coding efficiency (i.e. bitrate).

In order to accomplish Just-in-Time transcoding, Bolt65 JiT uses a limited set of encoding tools and parameters. Some of the tools used in HEVC to improve video quality and increase coding efficiency, such as asymmetric prediction units, deblocking filters, and complex rate-distortion optimization are omitted. Such approach sacrifices the highest possible quality of the transcoded bitstream but is necessary in order to conform to JiT restrictions. Full set of parameters used by JiT Bolt65 transcoder is shown in Table 5.5.

As can be observed from the configuration, there are a number of limitations for the re-encoding in Bolt65 transcoder that are introduced to decrease the processing time of the transcoder. Inter prediction is implemented with a simple three-step search (TSS) algorithm to reduce the number of evaluated inter prediction candidates. In loop filters, deblocking and SAO filter that are used to reduce artifact in the frame that can appear on the block boundaries are disabled since they affect processing in both, decoder and encoder, and can thus compromise transcoder execution for JiT. Also, there are no complex Rate Distortion Optimization (RDO) algorithms that usually evaluate multiple versions of CU splits and prediction modes. Instead,

in both, intra and inter prediction, all blocks are assessed with the SAD block matching algorithm, and the one with the lowest value of SAD is chosen as a final.

Table 5.5: Bolt65 JiT transcoding configuration

Coding option	Parameter
QP	Fixed – does not change within a video sequence
Search algorithm	Three step search, with the defined search area (default search area is 64)
Decoded picture buffer	Size of the decoded picture buffer is 1
GOP structure	Only I and P frames are used
Intra prediction	35 possible modes are tested
In loop filters	Both, deblocking and SAO filter disabled
CTU size	64x64
Minimum CU size	8x8
Transform tree	Max depth = 0
Prediction Units	Only 2Nx2N PUs supported
RDO	No smart RDO algorithm
Block matching algorithm	Sum of absolute differences (SAD)

This baseline transcoder will be used to observe improvements in using a proposed algorithm that utilizes coding information from the input video stream in terms of video quality and coding efficiency.

5.3.2 Kvazaar

The second baseline will be used to observe losses in video quality and bitrate compared with the transcoder that does not have timing requirements. For this purpose, open-source Kvazaar encoder is chosen. Re-encoding with Kvazaar was performed with the default settings with preset set on “medium” [57]. However, to obtain comparable results, several tools and encoding parameters were overridden from the default preset. All overridden parameters and the specific flags that were included when running the encoder are shown in Table 5.6. All other parameters that are not shown in the table below were set to default values.

The reason behind changing the default values of Kvazaar encoder is to obtain results that are comparable between all three transcoders: Bolt65 JiT, Kvazaar and the transcoder proposed in this thesis. Including in loop filters in just one of these scenarios, would increase

video quality solely by introducing one or both of the filters, and the conclusions and observations of using the proposed algorithm would not be based on valid assumptions.

Table 5.6: Kvazaar encoder configuration

Coding option	Overridden parameter	Kvazaar flag
QP	Set as fixed	--(no-)rdoq
Decoded picture buffer	Size of the decoded picture buffer is 1	-r 1
In loop filters	Both, deblocking and SAO filter disabled	--sao="off" --no-deblock
Prediction Units	Only 2Nx2N PUs supported	--no-amp --no-smp

5.4 Evaluation

During the transcoding process, three main aspects were considered and evaluated: processing time, bitrate and PSNR.

5.4.1 Processing time

The idea of Just-in-Time transcoding is not merely to transcode the video as fast as possible but to ensure that the video will be transcoded in a given period of time while trying to provide the best trade-off between video quality and coding efficiency, depending on the requirements and constraints of the system. Therefore, the value of t_{JiT} is defined as a maximum time that needs to be satisfied to achieve Just-in-Time video transcoding. From this point on, the processing time required for JiT transcoding for a particular video sequence will be set as t_{JiT} and other times will be presented relative to t_{JiT}. This representation is also a more convenient way to show Just-in-Time constraint since it can depend on various conditions; from the video sequence itself, where the video with higher frame rate has to be processed faster, to the computing power of the system on which the transcoder is being run. With this representation, all video sequences have a unified condition: if the processing time of the transcoder for a specific video sequence is higher than 1.00 * t_{JiT} than the transcoder does not conform to JiT requirements. Otherwise, if the processing time is lower than 1.00 * t_{JiT} than the transcoder satisfies Just-in-Time. Exact processing speed expressed in frames per second (fps), for each test video sequence is given in Table 5.7.

Table 5.7: t_{JIT} for test video sequences

#	Video name	t_{JIT} [fps]
1	Shields	50.00
2	ParkRun	50.00
3	KristenAndSara	60.00
4	Johnny	60.00
5	FourPeople	60.00
6	BasketballDrive	50.00
7	Calendar	50.00
8	Cactus	50.00
9	BQTerrace	60.00
10	RushHour	25.00
11	Riverbed	25.00
12	PedestrianArea	25.00
13	BlueSky	25.00
14	Traffic	30.00
15	DuckTakeOff	50.00
16	Bosphorus	120.00
17	Beauty	120.00

5.4.2 Bitrate

One of the most important aspects of all video compression standards is bitrate. A video bitrate is a number of bits needed to encode one second of the video sequence and is calculated with the following formula:

$$Bitrate = \frac{\frac{N_{bits}}{N_{frames}} * frameRate}{1000} \; [kbps] \tag{5.1}$$

N_{bits} is an overall number of bits needed to encode video sequence, while N_{frames} is an overall number of frames within a video. By dividing these two values, the average number of bits needed to encode one frame is acquired. Multiplying average number of bits with the video frame rate (i.e., number of frames in one second) bitrate in bits per second is obtained. Final value is divided with 1000 to get bitrate in kilobits per second.

Bitrate is a measurement that determines coding efficiency. Smaller bitrate means that the same video is encoded with a lower number of bits, ergo coding efficiency for such encoding is better. Bitrate reduction can be achieved by lowering video quality (e.g., in a live video chat where throughput is more important than video quality) but could also indicate better compression algorithm or standard. As mentioned in the introduction, HEVC standard doubles the coding efficiency when compared to AVC, while keeping the same video quality. Thereby, in order to get the full evaluation of the quality of a certain standard or algorithm, both measurements have to be examined: bitrate and video quality.

5.4.3 PSNR

Peak signal-to-noise ratio (PSNR) is a term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation [59]. In the image and video processing, PSNR is used to calculate video quality by comparing the encoded video frame with the original frame. The signal is data from the original frame while the noise is the error introduced by a compression.

PSNR is calculated as follows and is expressed in terms of the logarithmic decibel scale [dB]:

$$PSNR_{dB} = 10\log_{10}\frac{(2^n - 1)^2}{MSE} \tag{5.2}$$

In the equation above, n is the number of bits used to represent one pixel of the video frame (only luma sample). Giving that each pixel is represented with 8 bits, the peak signal is 255, which is the highest value of a pixel. MSE is a mean square error between the original and an impaired reconstructed video frame and is calculated as follows:

$$MSE = \frac{1}{frameSize}\sum_{i=1}^{frameSize}(O_i - R_i)^2 \tag{5.3}$$

Where O_i is a value of a pixel in the original frame and R_i is a value of a pixel in the reconstructed frame. In the case of the transcoder, original frame is actually a frame that is decoded from the original input bitstream, since the original frame is not available when it comes to transcoding, while reference frame is transcoded frame. In all the test conducted and presented in this thesis, PSNR values are obtained by comparing decoded and transcoded frame.

6 REUSING CODING INFORMATION

Intelligent utilization of coding information extracted from the initial encoded video stream has a key role in enhancing the process of video transcoding, especially when it comes to Just-in-Time transcoding. This thesis presents a novel algorithm that reuses several data sets extracted from input bitstream to achieve Just-In-time transcoding while trying to maintain the quality of the encoded video sequence without significant impacts on coding efficiency. The proposed algorithm reuses three types of data from the input video stream: size od decoded coding units, a number of coding units mapped from the decoded frame and prediction modes of decoded coding units.

6.1 Size of decoded coding units

When encoding raw video sequence, parts of the frame with more motion will usually be encoded with a higher number of bits, while still parts of the frame, such as background, that is not being changed between multiple frames, should be encoded with much higher coding efficiency. This fact is demonstrated in Figure 6.1.

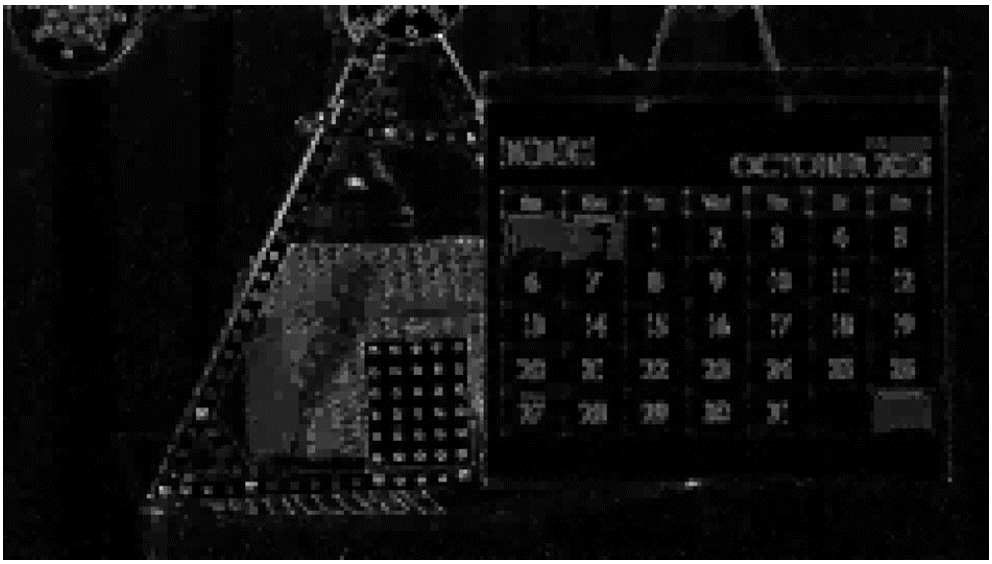

Figure 6.1: Heatmap for one frame in *Calendar* video sequence

Figure 6.1 shows a heatmap of one video frame in a *Calendar* video sequence that has been encoded using the HEVC standard. In this video, the calendar is constantly moving around the picture and represents a moving object. Darker squares show coding units that are encoded with a smaller number of bits. The lighter the square is, the more bits were needed to encode that particular coding unit. As can be expected, the background is mostly dark, while edges and

moving objects can be clearly distinguished with the presence of lighter squares. The same pattern can be seen in other video sequences depicted in Figure 6.2.

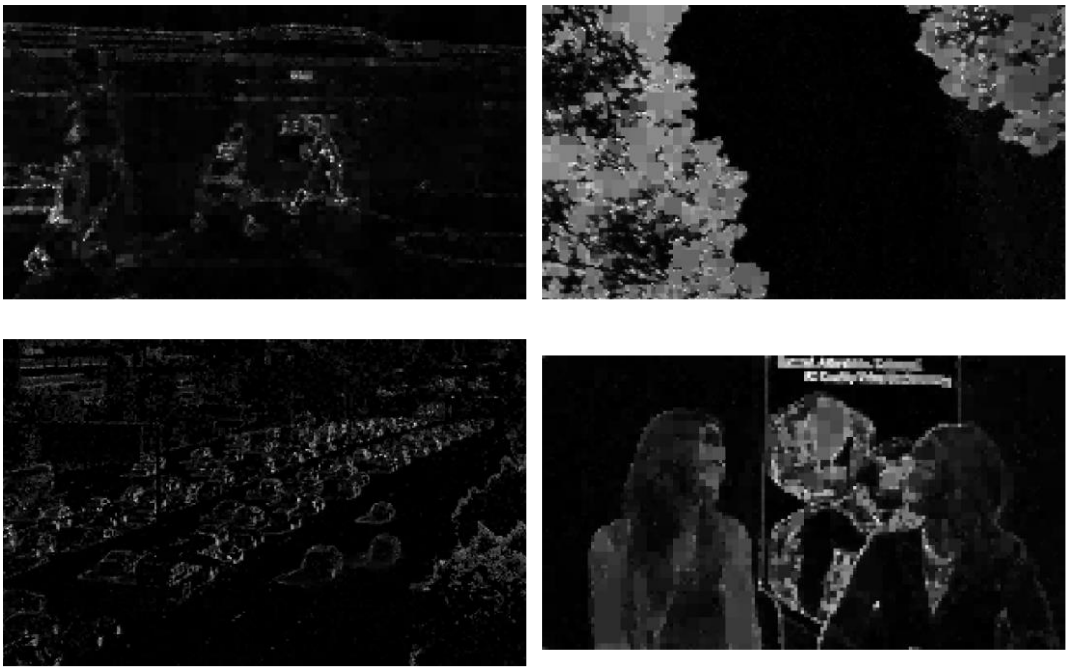

Figure 6.2: Heatmap for video sequences *BasketballDrive*, *BlueSky*, *Traffic*, and *KristenAndSara*

In some cases, different behavior can be observed. Figure 4. shows a picture where the background is encoded with a higher number of bits then the rest of the picture.

Figure 6.3: Heatmap for one frame in *Beauty* video sequence

This behavior occurs in high-resolution videos (e.g., 4K) that are encoded with low quantization parameter, where the neighboring pixels in black background, such as in this case,

can differ more than usual. In this scenario, nor intra prediction, nor inter prediction (since there is no real motion) do not form good residual, thereby resulting in a large number of transform coefficients that have to be encoded. Nonetheless, other parts of the picture follow expected behavior; moving parts (i.e., hair and eyes) are encoded with more bits then nonmoving parts (i.e., cheeks, forehead, chin), so this data can be reused in the transcoding process.

Not all videos have a pronounced background or part of the picture that remains the same throughout several frames. One example of such video sequence and its heatmap is given in Figure 6.4.

Figure 6.4: Heatmap for one frame in *Riverbed* video sequence

In this video that shows waves of the river, there are no visible objects or edges nor the big difference between neighboring coding units. Reusing information about sizes of decoded coding units when transcoding this kind of video sequences does not present the basis for relevant decisions.

6.2 Number of mapped coding units

CTU structure from the input video gives important information about RDO decisions made in the original encoding process and reusing that structure could save a lot of time in the transcoding process by avoiding full RDO in the encoding phase, which is one of the computationally most demanding processes. However, since the target video is transcoded from a higher resolution to a smaller resolution CTU structure cannot be simply taken as is and copied to the transcoded frame for several reasons:

- CTU blocks in transcoded frame cover larger areas of the picture
- Reusing same structure could form invalid CU blocks in the transcoded frame (e.g., blocks smaller than 8x8)
- Decisions made in the RDO process were based on different sets of pixels

The mapping between a higher resolution frame at the input and lower resolution frame at the output must be conducted in order to reuse CTU structure from the decoded video efficiently. Before beginning with the mapping mechanism, ratios between original and transcoded video are defined as:

$$\rho_w = \frac{inputWidth}{outputWidth} \tag{6.1}$$

$$\rho_h = \frac{inputHeight}{outputHeight} \tag{6.2}$$

ρ_w and ρ_h represent width ratio and height ratio when downsizing the picture. Although a lot of standard video formats use 16:9 aspect ratio today, and transcoding between those standards would infer same ρ_w and ρ_h , this does not have to always be the case. With a plethora of different mobile devices that are able to play video and that can have different resolutions, these ratios can be arbitrary.

Figure 6.5 shows an example of mapping one CTU from lower 1280x720 resolution, marked with a red square to CTU structure of video encoded in 1920x1080 resolution, marked with black squares.

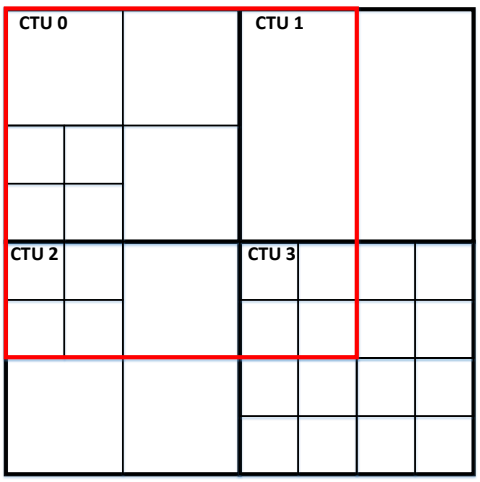

Figure 6.5: Mapping CTU structure

As it can be seen from the previous figure one CTU of downsized frame cover area of four CTUs in the original frame, some of which are covered in full (CTU 0) and some partially (CTU 1, 2, 3). Number of CTUs that are mapped from original to transcoded frame can be defined as:

$$M_{CTUs'} = (\lceil \rho_w + 1 \rceil) * (\lceil \rho_h + 1 \rceil) \tag{6.3}$$

where $CTUs'$ are CTUs in origanal frame that can be mapped, in full or partially, to the transcoded frame, and $M_{CTUs'}$ is the number of $CTUs'$. In the worst case scenario, decimal part of transcoding ratios (i.e., partially covered mapped CTUs) can cover the area below and above, or to the righ and to the left of the full mapped CTU, which is why the 1 is added to both factors. Next step in the mapping process is to determine all CUs from the original video that are incorporated in transcoded CTU. To facilitate the search of all mapped CUs, only CUs from $M_{CTUs'}$ are considered. Group of all mapped CUs can for CU in the transcoded video be defined as:

$$CU_{CUs'} = \{(\omega_i, CU'_i)\}, where\ 0 \leq i < M_{CU'}\ and\ i\ \in N \tag{6.4}$$

where $M_{CU'}$ is a number of all CUs that are mapped, i is the index of the mapped CU′ and $\omega_i$is the mapping coefficient that denotes the ratio of mapped CU within transcoded CU. Mapping coefficient ω is calculated as a ratio of the area of CU′ that is included in transcoded CU and is shown with the following equations:

$$\omega = \omega_w * \omega_h \tag{6.5}$$

$$\omega_w = \frac{width'}{CU'_{BS}} \tag{6.6}$$

$$\omega_h = \frac{height'}{CU'_{BS}} \tag{6.7}$$

Final mapping factor ω is obtained as a factor of width and height mapping factors that are calculated as a ratio between width/height (in pixels) that coincide (width′ and height′) and size of the CU' block (CU'_{BS}). To visualize the calculation of mapping factor, an example of calculating ω for CTU 1 from Figure 6.5 is given in Figure 6.6.

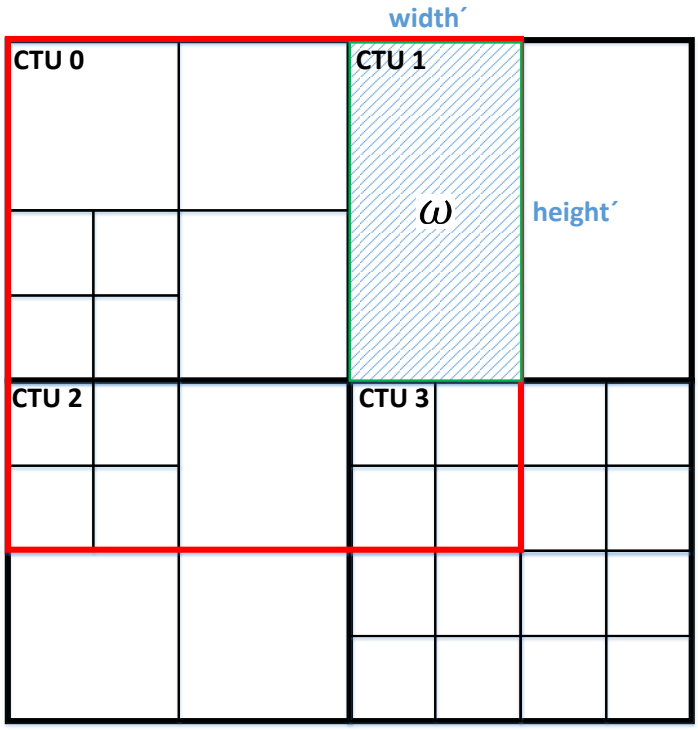

Figure 6.6: Calculation of mapping factor ω

For CTU 1 from the above figure mapping coefficients are ω_w=0.5, ω_h=1, making final ω=0.5, which, in this case, can be easily deducted from the presented picture. Finally, the entire process of mapping of one CTU, showed as a red square in Figure 6.5 and Figure 6.6 is shown with the representation (6.8) and depicted in Figure 6.7.

$$CTU_{CUs'} = \{(1.0, CU_0), (1.0, CU_1), (1.0, CU_2), (1.0, CU_3), (1.0, CU_4),$$
$$(1.0, CU_5), (1.0, CU_6), (0.5, CU_7), (1.0, CU_8),$$
$$(1.0, CU_9), (1.0, CU_{10}), (1.0, CU_{11}), (1.0, CU_{12}), \quad (6.8)$$
$$(1.0, CU_{13}), (1.0, CU_{14}), (1.0, CU_{15}), (1.0, CU_{16})\}$$

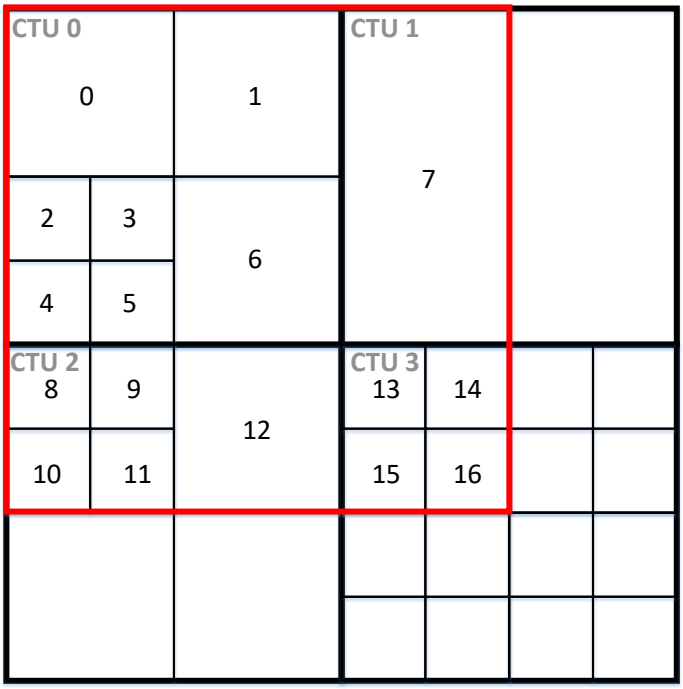

Figure 6.7: Final mapping

In the case when CTU that is being considered is split, re-mapping for all children is performed. Example of remapping CUs for the first child when the split is executed is given with Figure 6.8 and representation (6.8)(6.9).

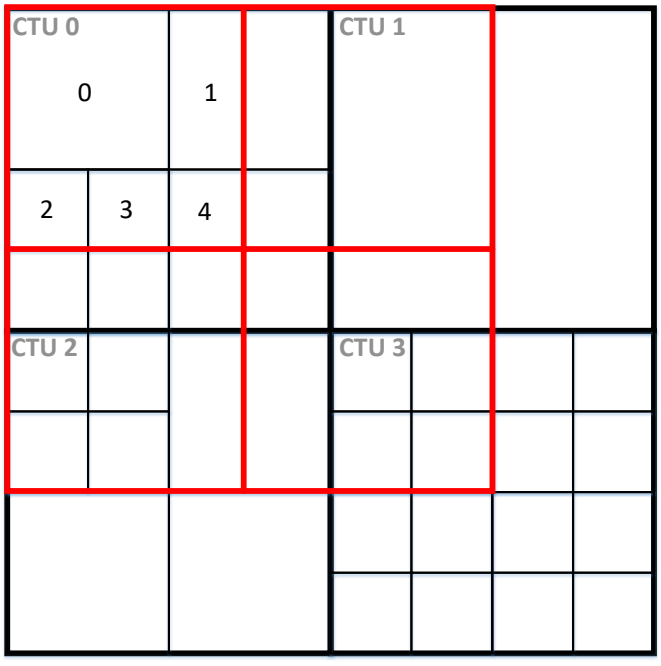

Figure 6.8: Mapping after the split

$$CU_{CUs'} = \{(1.0, CU_0), (0.5, CU_1), (1.0, CU_2), (1.0, CU_3), (0.25, CU_4)\} \quad (6.9)$$

6.3 Mode of mapped coding units

During the encoding of the video sequence, one of the most demanding tasks is to find the best possible prediction decision for each CU depending on the requirements of the system. If the main focus is to accomplish best coding efficiency, decisions will be directed to form the smallest possible bitstream, which could influence the quality of the encoded video. On the other hand, if the goal of the encoder is to create a video of the highest quality, size of the output bitstream will be larger. However, real-world video content providers rarely focus on maximizing one of these characteristics. Instead, they target to achieve the most suitable trade-off between coding efficiency and video quality. This trade-off is accomplished by implementing and incorporating complex Rate-Distortion Optimization algorithms in an encoding scheme. In RDO algorithms, multiple different modes for encoding single CU block are evaluated, and the best one is chosen based on the preferences and configuration of the encoder. In Just-in-Time encoding, less complex RDO algorithms have to be considered to ensure predictability in an environment bounded by strict timing constraints. Therefore, in Just-in-Time encoding, coding efficiency and video quality are considered only after the adequate performance is guaranteed. However, in the transcoding, modes that were chosen as the best ones in the original process of encoding can be reused and remapped when re-encoding the video. Although those decisions were made based on different conditions (i.e., on a higher resolution frame), they can be used as a starting point for CU prediction mode evaluation in the re-encoding. Reusing prediction modes can significantly reduce the number of operations needed to find the best possible mode for each CU block. In the case of Just-in-Time transcoding, in order to accelerate application execution, number of operations can be furtherly reduced by finding CU mode that does not have to be the best one, but is still "good enough", meaning that choosing that mode over the best possible one will not have a major impact on the video quality or coding efficiency. Therefore, reusing modes from decoded bitstream is one of the fundamental techniques used in Just-in-Time transcoding, not only to achieve Just-in-Time execution but to maintain quality and coding efficiency of the original video as well, which is one of the biggest shortcomings in Just-in-Time encoding.

To reuse information about intra and inter prediction modes from the decoded bitstream, modes of all mapped CU have to be considered. Figure 6.9 shows prediction modes of all mapped CUs in transcoding from 1920x1080 to 1280x720 resolution.

Figure 6.9: Distribution of prediction modes in mapped CUs

Distribution of different modes within transcoded CU can suggest its complexity for the re-encoding phase. If all mapped CUs have similar modes (e.g., all mapped CUs are inter predicted and have motion vectors that point in the same direction), a number of operations for finding the mode for transcoded CU can be reduced by evaluating only a subset of candidates (e.g., only motion vectors in the direction of mapped CUs). Otherwise, if there is a big difference between modes of mapped CUs, that could indicate a need for further splitting or a more sophisticated algorithm for searching suitable prediction candidate.

7 CATEGORIZATION

One of the main aspects of the novel software algorithm presented in this thesis is a categorization of coding units based on the coding information from the input video stream. The idea behind the categorization is to divide different blocks into certain categories and process them in different manners. Some high complex CUs, where a lot of information is contained should be analyzed in more details since they can have a higher impact on final bitstream. On the contrary, decisions about less complex CUs can be taken earlier to speed up the process, without sacrificing quality. This trade-off between coding efficiency and output video quality, while satisfying strict timing requirements can be controlled by manipulating decision process for each of the defined categories. In the next chapters categorization based on three different aspects is introduced using three different types of information, as described in chapter 6:

- Categorization based on the decoded number of bits
- Categorization based on the number of CUs mapped from the decoded frame
- Categorization based on the modes of mapped CUs retrieved from the decoded frame

7.1 Categorization based on a size of decoded coding units

While decoding input bitstream number of bits that were needed to originally encode each CU can be retrieved without any additional processing, i.e., without impact on processing time. However, when downsizing video and re-encoding it with new parameters, number of bits that will be needed to encode new video is not available in advance. Only by implementing complex RDO algorithms that implement encoding loop in which several modes are tested, encoded and evaluated in terms of coding efficiency, this could be achieved. This method introduces huge computational complexity and is not viable in Just-in-Time transcoding. Thereby, a number of bits from the decoded bitstream can be used to approximate the complexity of transcoding CUs. Equation (7.1) shows how the approximated number of bits (B′) for transcoded CU is computed.

$$B' = \sum_{i=0}^{M_{CU'}} \omega_i * B_i \tag{7.1}$$

$M_{CU'}$ and ω_i are a number of mapped CUs and mapping coefficients for each mapped CU as described in chapter 6. Notice that, if the analogue approach is followed to find B′ for

all CUs in the transcoded frame, number of bits needed to encode original video will be the same as the number of approximated bits for transcoded video. Hence:

$$\sum_{i=0}^{NCU_O} B_i = \sum_{j=0}^{NCU_T} \sum_{k=0}^{M_{CU'}} \omega_k * B_k \tag{7.2}$$

NCU_O and NCU_T are the number of coding units in the original and transcoded frame, respectively. This statement cannot be true since a number of bits needed to encode lower resolution of the video is less than a number of bits needed to encode the same video in higher resolution. Thereby, the value of B′ is defined, not as an approximated number of bits that will be needed to encode target video in a smaller resolution, but as the complexity of CU induced by the number of bits from a decoded video stream, or CU bit complexity. Bit complexity can be used as relevant information for decision making in the transcoding process because values of B′ will be considered in a relative manner.

CUs with a smaller number of bits usually present more static parts of the picture that can be predicted with less complex prediction algorithms, without considerable losses in quality or coding efficiency. Following this assumption, Category LBC (Low Bit Complexity) is defined, where a certain percentage of CUs with the lowest bit complexity are assigned. CU is categorized as LBC if the following condition is true:

$$0 \leq B' \leq \beta_L * B'_{max}, where\ 0 \leq \beta_L \leq 1\ and\ \beta_L \in Q \tag{7.3}$$

B'_{max} is the value of the highest bit complexity of a CU in a frame for the considered block size. Coefficient β_L specifies a boundary to form a subset of CUs with a smallest bit complexity. If β_L equals 0.1, all CUs with bit complexity within 10% of maximum CU bit complexity in that frame will be categorized as LBC.

Similarly, two additional categories based on bit sizes are introduced; Category HBC (High Bit Complexity), that contains CUs with highest bit complexity, and Category MBC (Medium Bit Complexity) that contains all other CUs. CU is categorized as HBC if the following condition is met:

$$\beta_H * B'_{max} \leq B' \leq B'_{max}, where\ 0 \leq \beta_H \leq 1\ and\ \beta_H \in Q \tag{7.4}$$

Coefficient β_H specifies a boundary to form a subset of CUs with the largest values of B′. All other CUs that do not fit in Category LBC or HBC, are placed in the category MBC. Hence, category MBC is restricted to the following scope:

$$\beta_L * B'_{max} < B' < \beta_H * B'_{max} \quad (7.5)$$

With this approach, where CUs are categorized relatively to B'_{max}, blocks, where the static parts of the frame are located, should be detected more accurately. In cases where there are no static areas in a frame, there will be a small number of blocks that fit in this category because of the smaller differences between CU bit complexities within the frame. To test these assumptions two videos were transcoded, one with the clearly visible static background (*BasketballDrive*) and one with the random motion throughout the frame and without the distinguishable background (*Riverbed*, also shown in Figure 6.4). The test is conducted by transcoding videos to the same resolution, just to test and observe the distribution of CUs within a frame. In this case, where there is no downsizing, B′ is equal to B since the mapping of decoded CUs is the one-to-one to transcoded CUs, meaning that the bit complexity of the transcoded CU is the same as the number of bits needed to encode the same CU in the original video (7.2). The same distribution can be expected in any of the downsized versions of the video sequence. Bit sizes for all 32x32 CUs were obtained, and their distribution is demonstrated in Figure 7.1 and Figure 7.2 for mentioned video sequences. The x-axis in the graph shows the number of bits needed to encode a CU, grouped in range of 5 bits, while Y-axis denotes the number of CUs that are encoded within the defined range. To demonstrate the difference in categorization between the sequences, coefficients β_L and β_H are fixed to 0.1 and 0.7, respectively.

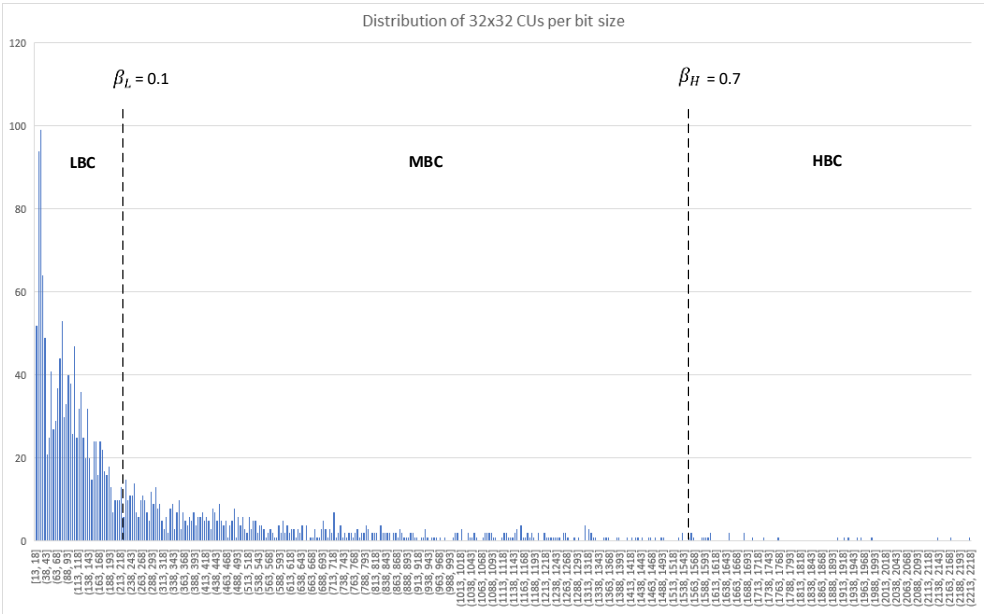

Figure 7.1: Categorization based on number of bits for *BasketballDrive* video sequence

Figure 7.1 shows a distribution for a video sequence with a pronounced background, where a lot of CUs are grouped at the start of the graph, meaning that high number of CUs belong to the static part of the image and can be categorized to Category LBC. With the B'_{max}=2218, boundaries for categorization are set on 0.1 * 2218 = 222 bits and 0.7 * 2218 = 1553 bits.

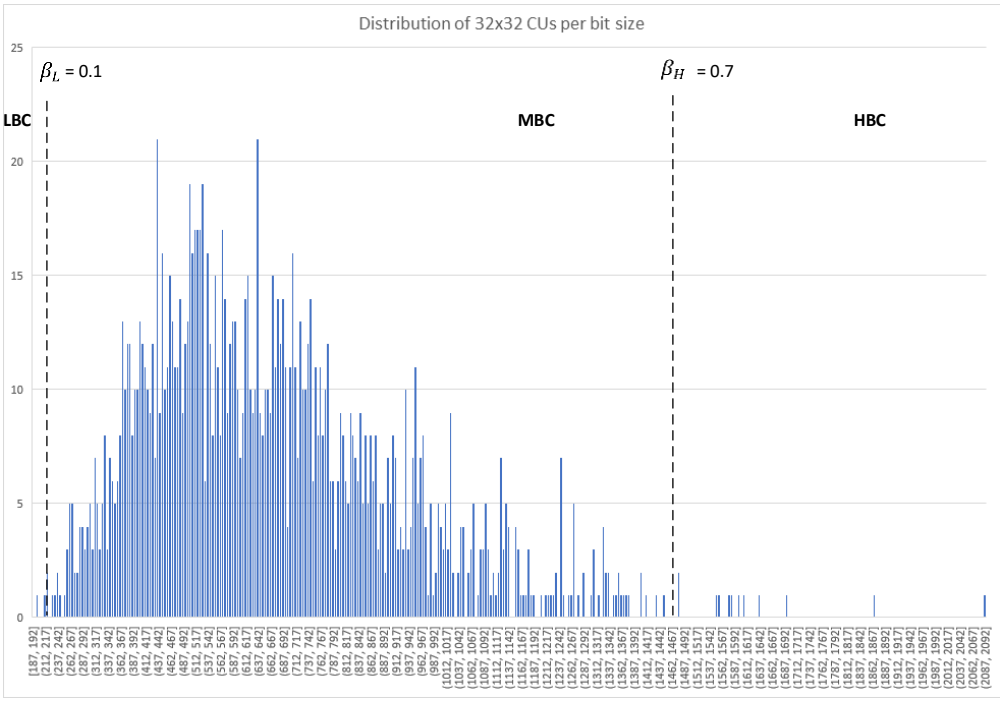

Figure 7.2: Categorization based on number of bits for *Riverbed* video sequence

In Figure 7.2 most of the CUs are grouped in the middle of the graph. With the B'_{max}=2092, boundaries for categorization are set on 0.1 * 2092= 209 bits and 0.7 * 2092 = 1464 bits. Considering that there is no clearly expressed background in this video sequence and that the motion is distributed randomly across the frame, prediction, and transformation in the encoding process give a high number of transform coefficients that are consequently encoded with a higher number of bits. This graph also depicts the major advantage of the introduced approach, where boundaries are set relatively in regard to B'_{max}. As can be seen from Figure 7.2 only one CU is categorized as LBC. Otherwise, if the lower boundary was set absolutely to 10% of all CUs in the frame, a large number of CUs would be wrongly assumed to be part of the background and the further decisions in the process of transcoding could be made on a false premise.

Using β_L and β_H as adaptive parameters enables more control over the transcoding process. Since the different categories will be processed in a different manner, higher quality

and coding efficiency can be achieved by reducing coefficients β_L and β_H and thereby increasing the number of CUs in categories MBC and HBC. Higher β_L and β_H coefficients increase number of CUs in the LBC category and decrease number of CUs in the HBC category, leading to lower computational complexity and faster execution of the transcoding process.

To conclude, CU can be categorized into three different categories, based on the value of CU bit complexity (B′).

$$CU \in \begin{cases} Category\ LBC, & 0 \leq B' \leq \beta_L * B'_{max} \\ Category\ MBC, & \beta_L * B'_{max} \leq B' \leq \beta_H * B'_{max} \\ Category\ HBC, & \beta_H * B'_{max} \leq B' \leq B'_{max} \end{cases} \quad (7.6)$$

7.2 Categorization based on the number of mapped CUs

When mapping CUs from decoded video to downsized transcoded video, a number of mapped CUs (MCU′) can be useful information that can be reused for decisions in the re-encoding phase. If a large number of CUs from input frame were mapped to one transcoded CU that means that that area of the original picture was divided into smaller CUs more frequently, which could indicate areas of the picture with more details. On the contrary, if there is a smaller number of mapped CUs, there is a higher probability of that area of the picture being uniform, containing fewer details. Figure 7.3 visualizes this fact by demonstrating CU distribution for one encoded video frame of the *BlueSky* video sequence.

Figure 7.3: CU distribution in the original video sequence (*BlueSky*)

Sky, as homogeneous part of the frame, is divided into larger blocks, meaning that CUs in the transcoded video that are located in that area will have a fewer number of mapped CUs. On the left and upper right part of the frame, where the two trees are located, CUs are split more frequently to form a more fine-grained structure. Mapping CUs that coincide with these, more detailed parts of the frame, will consequently have a higher value of MCU′.

For every CU in the transcoded video maximum and minimum number of MCU′ can be determined depending on the CU block size CU_{BS}:

$$\max(M_{CU'}) = \left\lceil \frac{CU_{BS} * (\rho_w + 1)}{CU'_{\min BS}} \right\rceil * \left\lceil \frac{CU_{BS} * (\rho_h + 1)}{CU'_{\min BS}} \right\rceil \quad (7.7)$$

$$\min(M_{CU'}) = \left\lceil \frac{CU_{BS} * \rho_w}{CU'_{\max BS}} \right\rceil * \left\lceil \frac{CU_{BS} * \rho_h}{CU'_{\max BS}} \right\rceil \quad (7.8)$$

$CU'_{\min BS}$ and $CU'_{\max BS}$ represent minumum and maximum block size for an input video stream, while ρ_w and ρ_h are width and height transcoding ratio (6.1)(6.2). In HEVC standard, maxium CU block size can be 64x64, while minumum block size is 8x8. However, these values can be restricted and adapted for each bitstream.

Based on the number of mapped CUs three categories are introduced: Category LM (Low Mapped) where the CUs with the smallest $M_{CU'}$ are located, Category HM (High Mapped) where the CUs with highest values of $M_{CU'}$ are located and Category MM (Medium Mapped) where all other CUs that do not fit in the previous two categories are associated. Similar as for the categorization that is based on bit complexity, two coefficients μ_L and μ_H that can be adapted based on the requirements of the transcoding system are introduced. Hence, CU is categorized as LM if the following condition is true:

$$\min(M_{CU'}) \leq M_{CU'} \leq \mu_L * \max(M_{CU'}), where\ 0 \leq \mu_L \leq 1\ and\ \mu_L \in Q \quad (7.9)$$

where $\min(M_{CU'})$ and $\max(M_{CU'})$ are the values for the considered CU block size. Category HB is described with the following condition:

$$\mu_H * \max(M_{CU'}) \leq M_{CU'} \leq \max(M_{CU'}), where\ 0 \leq \mu_H \leq 1\ and\ \mu_H \in Q \quad (7.10)$$

Category MM covers the remaining CUs:

$$\mu_L * \max(M_{CU'}) < M_{CU'} < \mu_H * \max(M_{CU'}) \quad (7.11)$$

Coefficients μ_L and μ_H can be adjusted at the beginning or in the runtime to enable more control during the transcoding process. Increasing or decreasing μ_L and μ_H will affect the number of CUs in each of the defined categories LM, MM, and HM. The higher the coefficient μ_L is, more CUs will be assigned to the LM category. Analogously, the lower the μ_H is, more CUs will fit in category HM. In the re-encoding phase of the transcoder, some conclusions and

decisions can be made based on the categories formed in regard to a number of mapped CUs. For example, if the CU belongs to HM category, it can be expected that considered CU would be split in the encoding of a downsized video more likely than if the same CU was part of a LM category. To verify this assumption, BlueSky video sequence from Figure 7.3 originally encoded in 1920x1080 resolution was decoded, downsized to 1280x720 resolution and then fully re-encoded without reusing any information from decoded bitstream. Distribution of CUs in the downsized video is shown in Figure 7.4.

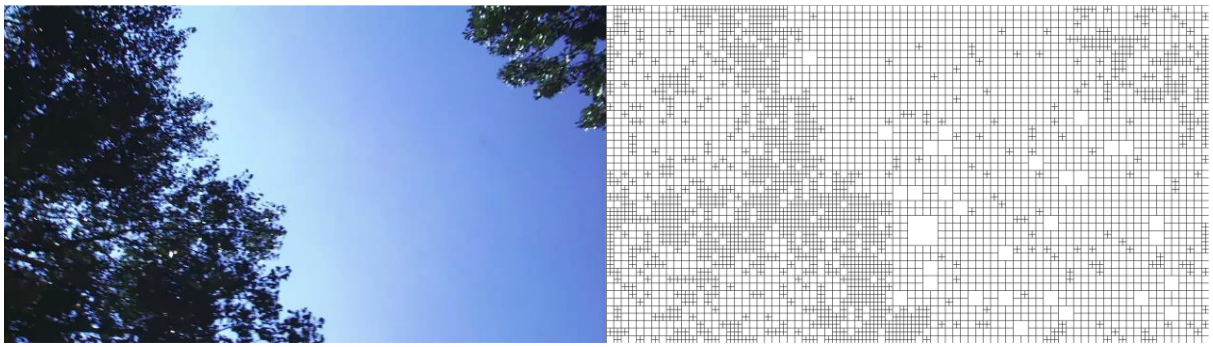

Figure 7.4: CU distribution in the downsized video sequence (*BlueSky* 1280x720)

As can be seen from the figure above, distribution of CUs and block sizes in the downsized transcoded bitstream follow a similar pattern as in original bitstream depicted in Figure 7.3.

To conclude, CU can be categorized into three different categories, based on number CUs mapped from the decoded bitstream ($M_{CU'}$).

$$CU \in \begin{cases} Category\ LM, & \min(M_{CU'}) \leq M_{CU'} \leq \mu_L * \max(M_{CU'}) \\ Category\ MM, & \mu_L * \max(M_{CU'}) < M_{CU'} < \mu_H * \max(M_{CU'}) \\ Category\ HM, & \mu_H * \max(M_{CU'}) \leq M_{CU'} \leq \max(M_{CU'}) \end{cases} \quad (7.12)$$

7.3 Categorization based on prediction modes

Transcoded CUs can be processed differently based on a variety of prediction modes of mapped CUs. Similar modes of all mapped CUs could suggest a homogeneous area that can be predicted with either inter or intra mode which can be extracted from the modes of mapped CUs. Thereby, two categories are defined, one for CUs that contain only intra mapped prediction modes and one for CUs that contain only inter predicted mapped CUs. Therefore, if all mapped CUs from the decoded bitstream are predicted in intra mode (7.13), CU is assigned to category IntraM (All intra mapped).

$$M_{CU'} = M_{CU'}(INTRA) \tag{7.13}$$

Likewise, if all mapped CUs are inter predicted (7.14), CU is assigned to category InterM (All inter mapped).

$$M_{CU'} = M_{CU'}(INTER) \tag{7.14}$$

All other CUs that do not fit in either of the two categories are categorized as ComboM (Combo mapped). Consequently, for CUs in ComboM category, the following conditions have to be fulfilled:

$$0 < M_{CU'}(INTRA) < M_{CU'} \tag{7.15}$$

$$0 < M_{CU'}(INTER) < M_{CU'} \tag{7.16}$$

$$M_{CU'}(INTRA) + M_{CU'}(INTER) = M_{CU'} \tag{7.17}$$

To obtain the frequency of occurrence for each of the defined categories, set of videos with the original spatial resolution of 1920x1080 were transcoded to two different resolutions: 1280x720 and 640x480. Input video sequences were encoded with four different QP values of 22, 27, 32, 37, as defined in Common Test Conditions [60]. For each CU block size (32x32, 16x16 and 8x8), values of $M_{CU'}$, $M_{CU'}(INTRA)$ and $M_{CU'}(INTER)$ were observed, based on which CUs were categorized to one of the above-mentioned categories. Statistics was gathered only for P frames, since in I frame all CUs have to be intra predicted. The frequency of occurrence of categories was examined and showed in following tables.

Table 7.1: Frequency of occurence of categories IntraM, InterM and ComboM for 32x32 blocks when transcoding from 1920x1080 to 1280x720 (%)

QP	22			27			32			37		
Video sequence	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**
Basketball Drive	1.86	12.70	85.44	1.20	21.24	77.56	1.17	21.14	77.69	1.14	21.69	77.17
BlueSky	2.45	30.38	67.18	4.63	38.94	56.43	3.45	41.75	54.80	1.63	41.75	56.62
BQTerrace	0.73	10.73	88.54	0.04	40.73	59.23	0.03	56.30	43.66	0.05	60.68	39.27
Cactus	0.98	27.45	71.57	0.63	52.44	46.93	0.72	58.04	41.25	0.74	60.01	39.25
Calendar	0.22	21.69	78.09	0.28	42.88	56.84	0.32	55.75	43.92	0.30	62.35	37.35
Pedestrian Area	5.00	31.81	63.18	6.03	37.37	56.60	6.69	39.48	53.83	6.99	41.32	51.68
Riverbed	46.97	0.03	53.00	42.48	0.04	57.49	32.36	0.10	67.54	18.52	0.30	81.18
RushHour	1.57	3.33	95.10	1.76	13.92	84.33	1.76	25.54	72.70	1.58	33.85	64.57

Table 7.2: Frequency of occurence of categories IntraM, InterM and ComboM for 16x16 blocks when transcoding from 1920x1080 to 1280x720 (%)

QP	22			27			32			37		
Video sequence	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**
Basketball Drive	5.93	35.12	58.96	4.21	44.40	51.39	4.27	43.91	51.82	4.27	43.86	51.87
BlueSky	7.13	53.44	39.43	9.25	58.03	32.72	6.80	59.77	33.43	3.66	61.17	35.18
BQTerrace	4.42	30.56	65.03	0.29	67.41	32.30	0.21	76.32	23.47	0.26	78.28	21.46
Cactus	2.94	53.41	43.65	1.87	71.54	26.59	2.02	73.70	24.28	2.13	74.30	23.56
Calendar	2.25	45.24	52.51	1.81	61.82	36.37	1.71	70.47	27.82	1.57	74.73	23.70
Pedestrian Area	11.92	48.54	39.54	12.44	51.72	35.84	13.00	52.63	34.38	13.32	53.46	33.22
Riverbed	67.76	0.94	31.30	63.19	1.19	35.62	54.71	1.87	43.41	40.44	3.52	56.03
RushHour	5.39	21.60	73.01	4.75	37.17	58.08	4.29	48.14	47.57	3.83	54.93	41.24

Table 7.3: Frequency of occurence of categories IntraM, InterM and ComboM for 8x8 blocks when transcoding from 1920x1080 to 1280x720 (%)

QP	22			27			32			37		
Video sequence	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**
Basketball Drive	12.66	52.44	34.91	10.46	60.53	29.01	11.29	60.22	28.49	11.76	60.19	28.05
BlueSky	12.99	65.83	21.18	14.49	68.49	17.03	11.92	70.73	17.35	8.67	73.04	18.29
BQTerrace	11.48	47.01	41.51	2.36	80.17	17.46	2.11	85.62	12.26	2.20	86.68	11.12
Cactus	6.35	68.25	25.40	4.19	81.07	14.74	4.57	82.13	13.30	4.95	82.35	12.70
Calendar	8.48	61.96	29.57	6.82	73.81	19.37	5.78	79.60	14.62	5.17	82.44	12.39
Pedestrian Area	18.32	59.39	22.29	18.67	61.72	19.61	19.25	62.28	18.47	19.62	62.72	17.66
Riverbed	77.65	4.37	17.99	73.92	5.57	20.51	67.49	7.61	24.91	55.71	11.82	32.47
RushHour	13.97	42.22	43.81	12.03	55.69	32.27	10.53	63.94	25.53	9.45	68.76	21.78

Table 7.1, Table 7.2 and Table 7.3 show that the percentage of IntraM and InterM increases as CU block sizes decrease. This behavior is expected since the maximum number of

mapped CUs (max($M_{CU'}$)) gets smaller for smaller blocks sizes, which increases the chance of all mapped CUs being predicted in the same mode. Also, the characteristics of certain videos can influence the occurrence of different prediction modes. For example, in *Riverbed* video sequence, which does not have any regular motion between the frames, there is the highest number of IntraM CUs, since, in that kind of scenario, intra prediction modes give better prediction than inter modes. Increasing quantization parameter does not follow any regular pattern in terms of categorization based on intra and inter modes.

Following three tables represent the same statistics, but for transcoding to a lower resolution of 640x480, thus with higher transcoding ratios (ρ_w=3, ρ_h=2,25).

Table 7.4: Frequency of occurence of categories IntraM, InterM and ComboM for 32x32 blocks when transcoding from 1920x1080 to 640x480 (%)

QP	22			27			32			37		
Video sequence	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**
Basketball Drive	1.86	12.70	85.44	1.20	21.24	77.56	1.17	21.14	77.69	1.14	21.69	77.17
BlueSky	2.45	30.38	67.18	4.63	38.94	56.43	3.45	41.75	54.80	1.63	41.75	56.62
BQTerrace	0.73	10.73	88.54	0.04	40.73	59.23	0.03	56.30	43.66	0.05	60.68	39.27
Cactus	0.98	27.45	71.57	0.63	52.44	46.93	0.72	58.04	41.25	0.74	60.01	39.25
Calendar	0.22	21.69	78.09	0.28	42.88	56.84	0.32	55.75	43.92	0.30	62.35	37.35
Pedestrian Area	5.00	31.81	63.18	6.03	37.37	56.60	6.69	39.48	53.83	6.99	41.32	51.68
Riverbed	46.97	0.03	53.00	42.48	0.04	57.49	32.36	0.10	67.54	18.52	0.30	81.18
RushHour	1.57	3.33	95.10	1.76	13.92	84.33	1.76	25.54	72.70	1.58	33.85	64.57

Table 7.5: Frequency of occurence of categories IntraM, InterM and ComboM for 16x16 blocks when transcoding from 1920x1080 to 640x480 (%)

QP	22			27			32			37		
Video sequence	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**
Basketball Drive	5.93	35.12	58.96	4.21	44.40	51.39	4.27	43.91	51.82	4.27	43.86	51.87
BlueSky	7.13	53.44	39.43	9.25	58.03	32.72	6.80	59.77	33.43	3.66	61.17	35.18
BQTerrace	4.42	30.56	65.03	0.29	67.41	32.30	0.21	76.32	23.47	0.26	78.28	21.46
Cactus	2.94	53.41	43.65	1.87	71.54	26.59	2.02	73.70	24.28	2.13	74.30	23.56
Calendar	2.25	45.24	52.51	1.81	61.82	36.37	1.71	70.47	27.82	1.57	74.73	23.70
Pedestrian Area	11.92	48.54	39.54	12.44	51.72	35.84	13.00	52.63	34.38	13.32	53.46	33.22
Riverbed	67.76	0.94	31.30	63.19	1.19	35.62	54.71	1.87	43.41	40.44	3.52	56.03
RushHour	5.39	21.60	73.01	4.75	37.17	58.08	4.29	48.14	47.57	3.83	54.93	41.24

Table 7.6: Frequency of occurence of categories IntraM, InterM and ComboM for 8x8 blocks when transcoding from 1920x1080 to 640x480 (%)

QP	22			27			32			37		
Video sequence	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**	**Intra M**	**Inter M**	**Comb**
Basketball Drive	12.66	52.44	34.91	10.46	60.53	29.01	11.29	60.22	28.49	11.76	60.19	28.05
BlueSky	12.99	65.83	21.18	14.49	68.49	17.03	11.92	70.73	17.35	8.67	73.04	18.29
BQTerrace	11.48	47.01	41.51	2.36	80.17	17.46	2.11	85.62	12.26	2.20	86.68	11.12
Cactus	6.35	68.25	25.40	4.19	81.07	14.74	4.57	82.13	13.30	4.95	82.35	12.70
Calendar	8.48	61.96	29.57	6.82	73.81	19.37	5.78	79.60	14.62	5.17	82.44	12.39
Pedestrian Area	18.32	59.39	22.29	18.67	61.72	19.61	19.25	62.28	18.47	19.62	62.72	17.66
Riverbed	77.65	4.37	17.99	73.92	5.57	20.51	67.49	7.61	24.91	55.71	11.82	32.47
RushHour	13.97	42.22	43.81	12.03	55.69	32.27	10.53	63.94	25.53	9.45	68.76	21.78

Table 7.4, Table 7.5 and Table 7.6 show that by increasing downsizing ratio between original and transcoded video, the possibility of occurrences of IntraM and InterM categories decreases, which is expected since the CU in the smaller, transcoded video, covers a larger area in the original video. Also, in the second case, where downsizing ratios are ρ_w=3 and ρ_h=2.25 value of max($M_{CU'}$) is higher than in the first scenario ρ_w=3 and ρ_h=2.25, meaning that the probability of all mapped CUs being predicted in the same mode is lower.

In most of the presented cases, ComboM is the most common category. However, within ComboM category there can be different cases. Even if one mapped CU is intra predicted and all others are inter predicted or vice-versa, CU will be assigned to the ComboM category. Hence, ComboM category is disjointed to two separate categories based on the ratio between a number of mapped CUs that are predicted in inter and intra mode. Therefore, two additional categories are introduced: ComboInter, where more mapped CUs are inter predicted, and ComboIntra, where more mapped CUs are intra predicted. The same test used to observe the frequency of occurrence of each category was repeated, and the average ratio between prediction modes (γ) as defined in (7.18) is observed.

$$\gamma = \frac{M_{CU'}(INTER)}{M_{CU'}(INTRA)} \tag{7.18}$$

Obtained values are presented in Table 7.7 for transcoding to 1280x720 resolution and in Table 7.8 for transcoding to 640x480 resolution.

Table 7.7:Values of γ when transcoding from 1920x1080 to 1280x720

Video sequence	QP = 22	QP = 27	QP = 32	QP = 37
BasketballDrive	1.91	1.97	1.89	1.90
BlueSky	3.39	3.21	2.78	2.81
BQTerrace	2.32	4.30	4.08	4.01
Cactus	3.13	3.16	2.48	2.26
Calendar	2.41	2.56	2.52	2.51
PedestrianArea	1.54	1.46	1.44	1.44
Riverbed	0.44	0.49	0.50	0.57
RushHour	1.78	2.15	2.32	2.45

Table 7.8:Values of γ when transcoding from 1920x1080 to 640x480

Video sequence	QP = 22	QP = 27	QP = 32	QP = 37
BasketballDrive	2.14	2.44	2.41	2.47
BlueSky	4.82	5.22	4.51	4.49
BQTerrace	2.59	6.37	7.13	7.45
Cactus	3.93	4.46	3.57	3.39
Calendar	2.99	3.99	4.21	4.20
PedestrianArea	1.92	1.95	1.98	2.03
Riverbed	0.35	0.40	0.42	0.52
RushHour	1.84	2.53	3.09	3.54

Table 7.7 and Table 7.8 show that the ratio γ generally diverge to one of the modes and that the value of γ is rarely close to 1, which would indicate the same number of inter and intra mapped CUs. Therefore, instead of simply defining categories ComboIntra and ComboInter as categories that have a higher number of one of the prediction modes, these categories are defined relatively, based on coefficient δ. CU is allocated to ComboInter category if the following condition is true:

$$M_{CU'}(INTER) > \delta * M_{CU'}(INTRA), where\ \delta > 0\ and\ \delta \in Q \tag{7.19}$$

Otherwise, CU is categorized as ComboIntra.

$$M_{CU'}(INTER) \leq \delta * M_{CU'}(INTRA), where\ \delta > 0\ and\ \delta \in Q \tag{7.20}$$

With this approach, condition for categorization in one of these categories can be set dynamically and can be adapted depending on the specific video sequence. Also, this concept enables detection of CUs that stand out from the average CU ratio, which would not be possible if the categorization was based solely on the higher number of prediction modes (i.e., if δ is fixed to 1).

All the facts presented above can be used in the decision-making process of a transcoder. Depending on the occurrence of the particular category within the decoded frame, focus can be directed to a different category in the re-encoding phase. For, example, if video sequence *Riverbed* is being transcoded, more precise intra prediction will be performed, since the occurrence of IntraM category for that particular video sequence is very high.

To conclude, CU can be categorized into four different categories, based on prediction modes of mapped CUs ($CU_{CU'}$).

$$CU \in \begin{cases} Category\ InterM, & M_{CU'} = M_{CU'}(INTER) \\ Category\ IntraM, & M_{CU'} = M_{CU'}(INTRA) \\ Category\ ComboInter, & M_{CU'}(INTER) > \delta * M_{CU'}(INTRA) \\ Category\ ComboIntra, & M_{CU'}(INTER) \leq \delta * M_{CU'}(INTRA) \end{cases} \quad (7.21)$$

8 ALGORITHM FOR REUSING CODING INFORMATION

An optimized algorithm for just in time video transcoding based on the utilization of coding statistics from the input video stream is presented in this thesis. The proposed algorithm targets to achieve an optimal trade-off between video quality of the transcoded bitstream and coding efficiency while conforming to strict timing requirements imposed by Just-in-Time transcoding. The basic concept behind this novel algorithm is to estimate the computational complexity of re-encoding each block based on the information retrieved from the decoded frame and to balance the workload of the transcoder accordingly. More computing resources will be assigned to processing more complex CUs that usually contain more detailed information within the video frame.

After introducing the concept of categorization of coding units based on the data extracted from the decoded frame, this chapter presents an algorithm for Just-in-Time video transcoding built based on this idea. Depending on the affiliation of each CU to particular categories, different algorithms are used to decide whether CU will be split or not, and which mode will be used for its prediction in the re-encoding phase.

8.1 Input data

To be able to categorize CU, several coefficients that define boundaries for categorization have to be defined:

- β_L and β_H for categorization based on the bit complexity
- μ_L and μ_H for categorization based on the number of CUs mapped from the decoded frame
- δ for categorization based on the modes of mapped CUs retrieved from the decoded frame

In the initial version of the algorithm, values of all coefficients are preset to default values for all video sequences and will not be adapted during the execution of the transcoder. Coefficients are defined in the configuration file of the application. Dynamic adaptation of coefficients are introduced later in this thesis.

Besides coefficients needed to form boundaries for the categorization, several other values are required before the beginning of the transcoding process and have to be calculated. These include:

- Transcoding width and height ratios - ρ_w and ρ_h
- Maximum and minimum number of mapped CUs for each block size - $\min(M_{CU'})$, $\max(M_{CU'})$
- Maximum bit complexity for each block size – B'_{max}
- Set of mapped CUs for each CU – $CU_{CU'}$

While transcoding ratios and values of $\min(M_{CU'})$ and $\max(M_{CU'})$ can be calculated immediately after decoding sequence parameter set (SPS) of the original bitstream, set of mapped CUs and maximum bit complexity have to be determined in the encoder part of the transcoder. Whenever a CU is formed in the encoder, either by creating new CTU or by splitting existing CU, CUs from the decoded frame are mapped and the set $CU_{CU'}$ is assembled, while the corresponding value of B'_{max} is updated accordingly.

8.2 Initial split

At the beginning of the encoding phase, a frame is divided into Coding Tree Units, CUs with largest block sizes. Each of these CTUs can be further split multiple times to form a tree of CUs (as shown in Figure 2.2). The maximum size of CTU is defined in HEVC standard (64x64) but can be limited for each bitstream to 32x32 or 16x16. So, the first step in the algorithm is to decide whether to split CTU or not. The analysis was made to determine the frequency of occurrence of 64x64 blocks that are not split. Only videos with resolutions of 1920x1080 and higher were taken into consideration since the probability of 64x64 CUs in lower resolutions is very small. Table 8.1 shows the percentages of occurrence of 64x64 blocks that are not split in the original bitstream.

The analysis shows that the presence of 64x64 CU blocks in an encoded bitstream is very low for all video sequences and all QP values. Also, if the CTU is not split to smaller CUs, it has to be split to multiple transcoding units (TU) that are predicted individually with the same intra mode. Thereby, coding efficiency when using 64x64 CUs is achieved only by not signaling intra mode for each TU independently, which is almost negligible when the number of such CTUs is low, as depicted in Table 8.1.

Table 8.1: Frequency of occurence of non-split 64x64 blocks

Video sequence	Resolution	QP = 22	QP = 27	QP = 32	QP = 37
BasketballDrive	1920x1080	0.03	0.04	0.04	0.05
BlueSky	1920x1080	0.06	0.20	0.35	0.40
BQTerrace	1920x1080	0.00	0.01	0.01	0.02
Cactus	1920x1080	0.01	0.02	0.02	0.02
Calendar	1920x1080	0.01	0.02	0.04	0.04
PedestrianArea	1920x1080	0.06	0.12	0.21	0.32
Riverbed	1920x1080	1.46	1.24	0.71	0.34
RushHour	1920x1080	0.02	0.02	0.03	0.03
Traffic	2560x1600	0.00	0.00	0.00	0.00
Beauty	3840x2160	3.11	0.34	0.04	0.00
Bosphorus	3840x2160	0.00	0.00	0.00	0.03
DuckTakeOff	3840x2160	0.22	0.02	0.00	0.00

Taking into consideration results presented above, all CTUs in the re-encoding phase are unconditionally split if their size is 64x64. Bypassing evaluation of modes for the largest coding blocks in HEVC standard helps to reduce the overall number of operations for finding final prediction mode with no significant losses in coding efficiency. Thus, the first operation in this phase of the algorithm is to split all CTUs. While splitting CTU following operations are performed:

- All mapped CUs ($CU_{CU'}$) are re-mapped from CTU to appropriate child CU as depicted in Figure 6.8
- Bit complexity (B′) is calculated for each newly created CU, and the value of B'_{max} is updated accordingly
- Child CUs are categorized based on bit complexity, number of mapped CUs and modes of mapped CUs as described in chapter 7

After initial CTU split, every 32x32 CU block is evaluated based on the associated categories, and the initial decision about the further splitting of the CU is made. If the 32x32 CU block belongs to both categories, LBC (Low bit complexity) and LM (Low mapped), that CU block is not split and is marked as “split concluded” by setting a flag *SC* to 1. When a *SC* flag in CU is set to 1, it means that decision about the split is final and that no further considerations for splitting will be conducted later in the algorithm. All other combinations of

categories based on bit complexity and a number of mapped CUs lead to splitting 32x32 CU and repeating the same process for four 16x16 CUs. Notice that, for initial split decisions, only the first two categorizations are taken into account. Categorization based on the modes of mapped CUs will be used later in the algorithm. For 16x16 CU blocks, the same procedure is repeated if they belong to the LBC and LM category. However, for 16x16 block sizes, only CUs that belong to most complex categories, HBC and HM, are split. All other 16x16 CUs that are associated with different combinations of categories are not split, and their *SC* flag is set to 0, meaning that this CU can be still examined by a different process and that the decision about the split is not final. For 8x8 CU blocks that are formed after splitting 16x16 CUs, only re-mapping and categorization is performed, since they cannot be furtherly split. SC flag for 8x8 blocks is set to 1 by default. Figure 8.1 shows a flowchart of an initial split decision for CU.

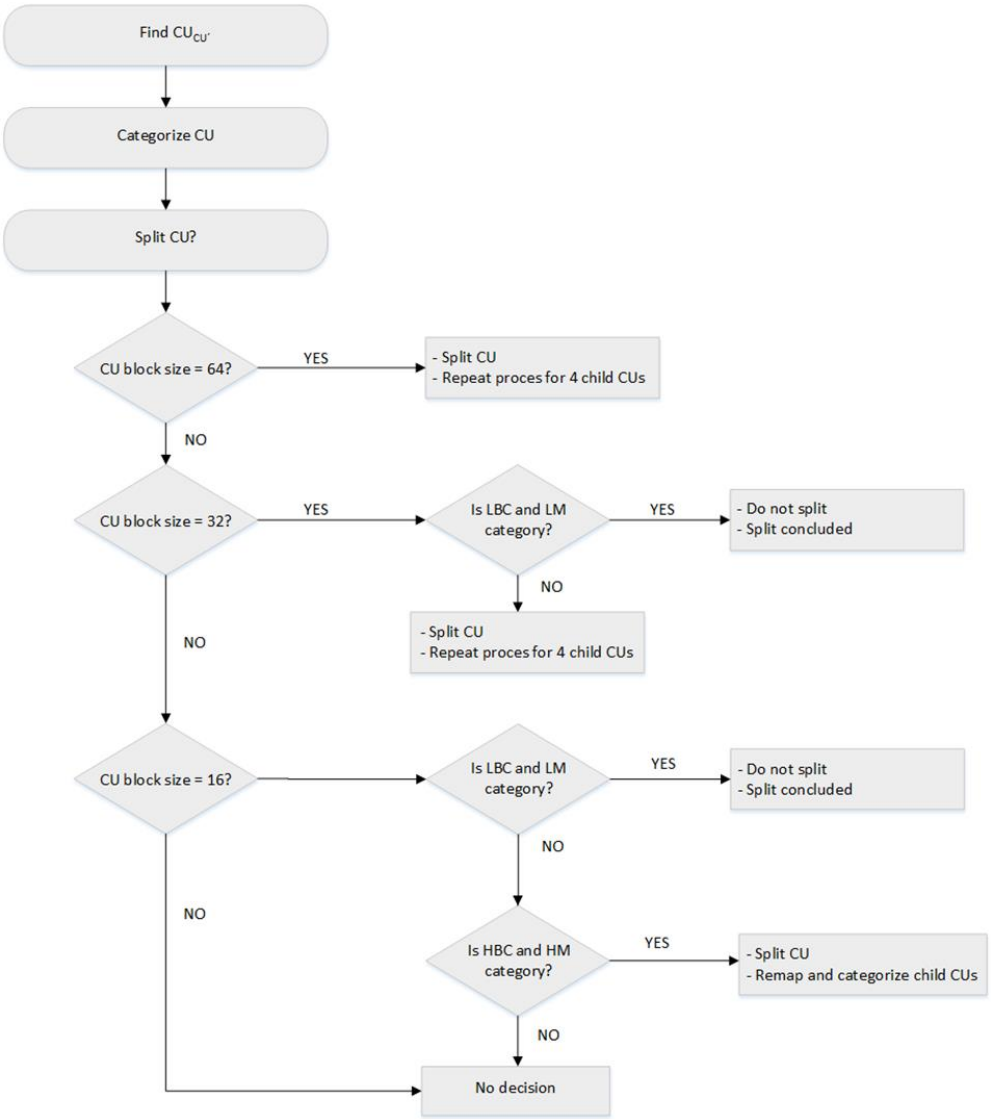

Figure 8.1: Initial split flowchart

Initial split only makes final split decisions for two types of CUs:

- Most complex CUs - 16x16 CU blocks with the highest bit complexity and the highest number of mapped CUs
- Least complex CUs - 32x32 blocks with the lowest bit complexity and the lowest number of mapped elements

Number of CUs for which split decision is final (i.e., SC flag is equal to 1) after an initial split depends on the coefficients β_L ,β_H , μ_L and μ_H. Figure 8.2 shows the CU distribution within one frame of three video sequences (*BlueSky, KristenAndSara and BasketballDrive*) after performing the initial split, for three different sets of coefficients:

A. $\beta_L = 0.4$, $\beta_H = 0.9$, $\mu_L = 0.5$, $\mu_H = 0.9$
B. $\beta_L = 0.3$, $\beta_H = 0.6$, $\mu_L = 0.3$, $\mu_H = 0.6$
C. $\beta_L = 0.1$, $\beta_H = 0.5$, $\mu_L = 0.2$, $\mu_H = 0.5$

Figure 8.2: CU distribution after the initial split for three different sets of coefficients

The first set of predefined coefficients (A) is most suitable for fast transcoding since a large number of CUs are categorized as an LBC and LM. Setting high values of coefficients β_H and μ_H decreases the number of CUs in most complex categories HBC and HM, which is why the number of 8x8 CU blocks is relatively small. Third set of coefficients (C) does exactly the

opposite, which has decreased number of 32x32 CU blocks as a consequence. This configuration increases overall video quality but can compromise Just-in-Time execution. By changing these coefficients to different values, the balance between video quality and execution time can be achieved. One such examplesis given with the second set of coefficients (B) that shows the coefficients set between the two extremes (A and C). To quantify the number of CUs for which decision has been made during the initial split, Table 8.2 that shows a number of blocks with sizes of 8x8, 16x16 and 32x32 for each of predefined sets of coefficients is presented. As expected, the number of 32x32 blocks is the highest for the first configuration (A) and the lowest for the third and vice-versa for 8x8 CU blocks

Table 8.2: Distribution of CU blocks after the initial split (per block size)

Configuration		A)	B)	c)
Video sequence	**Blocks**			
BlueSky	*32x32*	507	263	128
	16x16	1571	2490	2858
	8x8	4	232	920
KristenAndSara	*32x32*	539	346	181
	16x16	1435	2184	2807
	8x8	36	128	276
BasketballDrive	*32x32*	452	161	14
	16x16	1778	2886	3380
	8x8	16	280	656

8.3 Prediction decisions

Next step in the algorithm is to determine a prediction mode for each CU that is formed after the initial split. This decision is based on the modes of mapped CUs since it can be expected that transcoded block will have similar prediction mode as one or more of its mapped CUs. Next few chapters describe the process of determining prediction modes for each of the categories: IntraM, InterM, ComboInter, and ComboIntra.

The main idea behind the concept of categorization based on modes of CUs mapped from the decoded frame is to find the best prediction candidates with the number of operations as low as possible in order to conform to strict timing requirements imposed by Just-in-Time video transcoding. The first step in this approach is to divide blocks into different categories based on characteristics of the mapped coding units. For each of the defined categories, a different set of rules and algorithms are used to find a prediction that is close to the best possible prediction which would be selected if a full re-encoding, without time constraints, is used.

8.3.1 Prediction for IntraM category

There are 35 possible modes for intra prediction: DC, Planar and 33 angular modes. In the encoding process, if the CU is intra predicted, all the modes can be evaluated, and the one with the best prediction (i.e., the smallest residual) is selected. Different algorithms can be used to decrease the number of intra modes that are being evaluated ([61],[62],[63]), thereby reducing the complexity of intra prediction.

In the transcoding, information about modes of CUs from the input frame is available and can be utilized to minimize the number of evaluated modes. If the CU belongs to IntraM category, there is a high probability that this CU would also be intra predicted in the encoding phase. However, during the downsizing of the decoded frame, some information is lost, and the transcoded pixels, although similar, are not the same as in the decoded frame. Therefore, transcoded CU in some cases can be inter predicted, regardless of all mapped CUs being intra predicted. So, when determining prediction mode in transcoded CU, this has to be taken into consideration. Figure 8.3 shows the prediction decision process for CU that is categorized as IntraM.

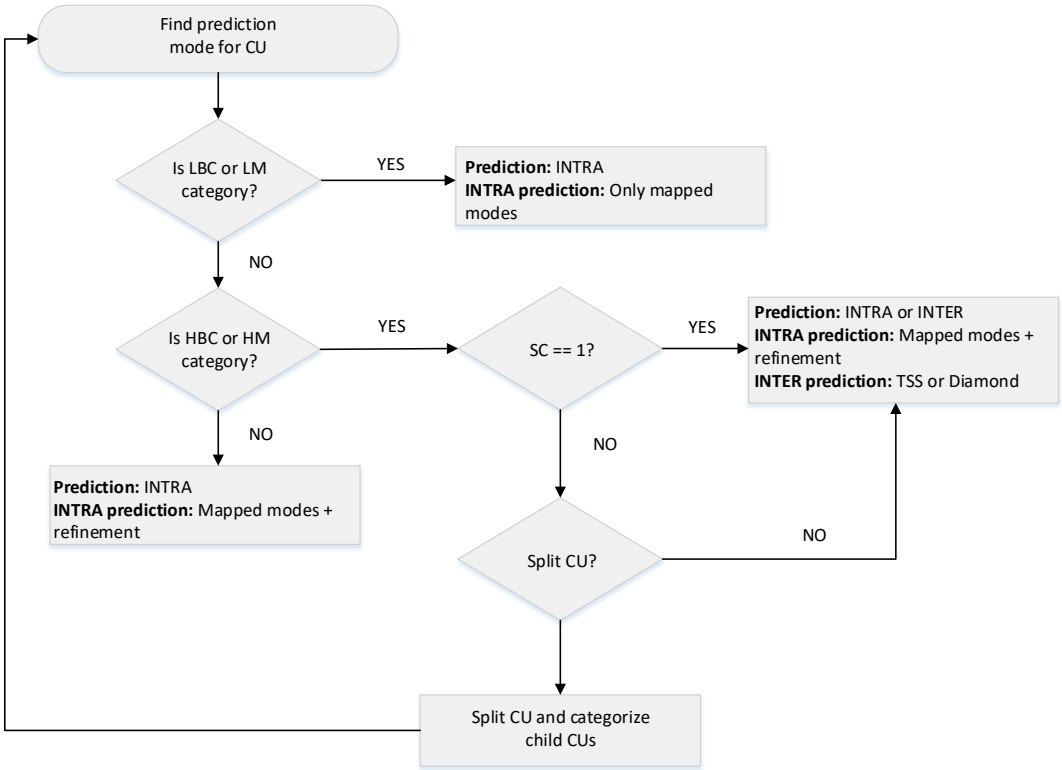

Figure 8.3: Prediction decision for IntraM category

If the IntraM CU is also categorized as LBC or LM, only intra prediction modes of mapped CUs are evaluated, and the best among them is taken as a final prediction mode. In this scenario, the maximum number of tested modes is $M_{CU'}$. However, since the intra modes of mapped CUs can be the same, the average number of tested intra modes is often lower than $M_{CU'}$.

Otherwise, if the CU fits in either HBC or HM category, the more detailed decision process is performed to obtain better residual. First, if the split has not been considered in the initial split phase (i.e., if the flag SC is set to 0), then the decision about the split is reevaluated. Algorithm for this decision is given with the following pseudocode:

```
if SC = 1
      Do not split

avgFirst = Find the average mode of first two mapped CUs
avgLast = Find the average mode of last two mapped CUs

difference = abs(avgFirst – avgLast)

if difference > (Number of INTRA modes / 2)
      Split CU
      Categorize child CUs
else
      Do not split CU

SC = 1
```

The split is performed only if there is a considerable difference in modes between the first two mapped and last two mapped CUs. This occurrence should indicate pixel diversity within the same block, which is why the decision to split is made. If the CU is split, four children are formed and categorized, and the same process is repeated for each newly created CU. If the split is not performed, then the prediction is made by evaluating the subset of both, intra and inter candidates. For intra prediction, modes from mapped CUs with the refinement of ± 1 are evaluated. Refinement of ±1 evaluates three modes if possible; that exact mode, and the modes below and above. For example, let's assume that there are five mapped CUs with the following intra modes:

$$Mapped\ modes\ \{0, 9,12,11,9\} \tag{8.1}$$

Giving that the second mapped CU is predicted with intra mode 9, for the transcoding, three modes will be evaluated: 8, 9 and 10. With this approach, a wider range of modes is checked. Since the neighboring intra modes are very similar (Figure 2.3) and the resized frame

can have slight differences within the same area of the picture in terms of texture, checking neighboring modes can give more precise residual. Following the same concept for all mapped modes, a set of all evaluated intra modes will be:

$$Evaluated\ modes\ \{0,1,8,9,10,11,12,13\} \tag{8.2}$$

As already mentioned, the best prediction for re-encoding IntraM CU does not necessarily have to be intra prediction, which is why for high complex IntraM CUs inter prediction is also tested. However, in IntraM category, there is no inter mapped coding units, so there is no possibility of reusing any information about motion vectors from decoded frames. Therefore, a simple search algorithm, such as Three Step Search or Diamond Search, is used to find best inter predicted candidate. After both predictions, intra and inter are tested, the best residual is selected, and the CU is predicted with appropriate mode. Of course, if the frame that is being re-encoded is intra frame, inter prediction is not possible and will be skipped.

Finally, if the IntraM CU is not in any of these categories, hence belongs to MM and MBC, only intra prediction that checks modes from mapped CUs with the refinement of ± 1 will be conducted.

Notice that for IntraM category, inter prediction is only considered for most complex CUs, i.e., those who fit in either HBC or HM category. Evaluation of inter prediction for less complex IntraM coding units is omitted, and the number of evaluated candidates is restricted to reduce computational complexity. Including inter prediction in such blocks, could, in some cases, result in finding better residual blocks, but the difference between best inter predicted and best intra predicted block is usually not substantial in terms of final bit rate and quality.

8.3.2 Prediction for InterM category

For InterM category, all mapped modes are inter predicted, so it is highly unlikely that the best prediction mode for InterM CU in the re-encoding will be intra predicted. Thereby, for InterM category, only inter prediction is considered, but the algorithm and search area for finding best motion vector depends on the other categories. Figure 8.4 shows the prediction decision process for InterM category.

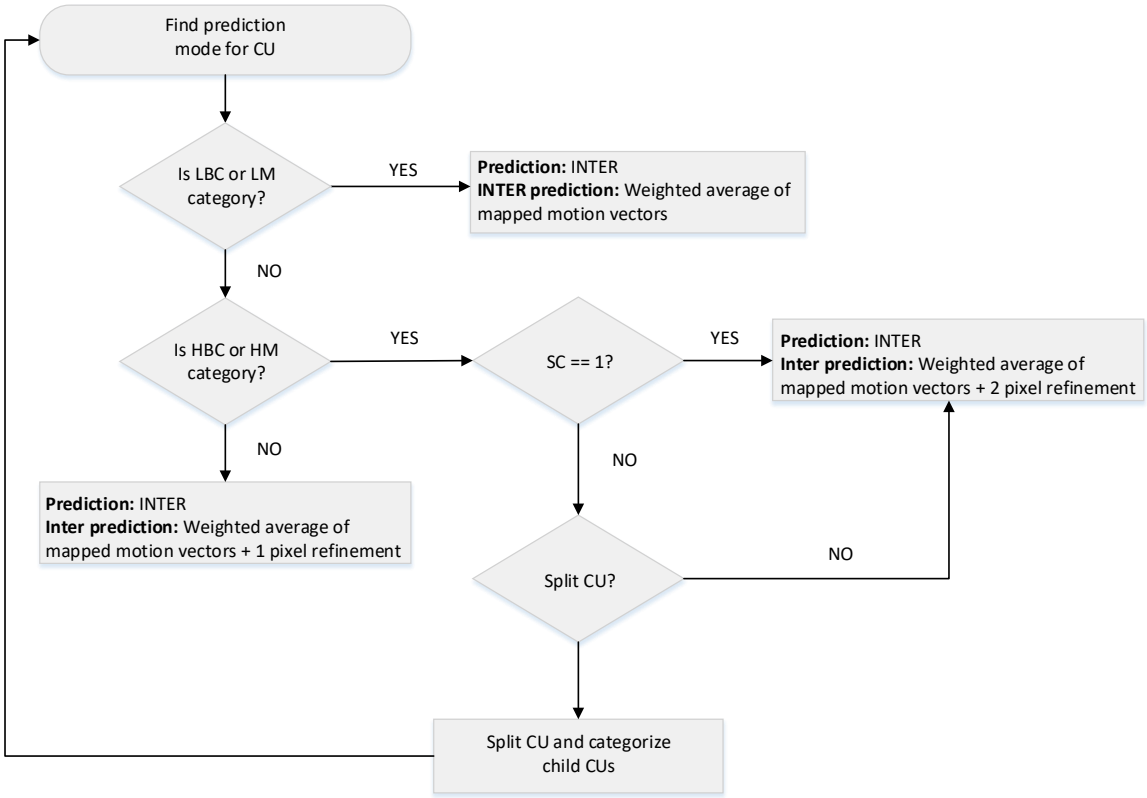

Figure 8.4: Prediction decision for InterM category

If the CU also belongs to either of the least complex categories LBC or LM than the weighted average of mapped motion vectors is chosen as a final motion vector, and no additional operations or search algorithms are performed. The transcoding motion vector is derived as follows:

$$mv_x = \left[\frac{\sum_0^{M_{CU^{''}}} mv'_x * \omega * \frac{1}{\rho_w}}{M_{CU'}} \right] \tag{8.3}$$

$$mv_y = \left[\frac{\sum_0^{M_{CU^{''}}} mv'_y * \omega * \frac{1}{\rho_h}}{M_{CU'}} \right] \tag{8.4}$$

Each motion vector from the mapped CUs is scaled to fit the transcoded frame. If the motion in the decoded frame was by X pixels to the right, that same motion vector in the frame that was downsized from 1920x1080 to 1280x720 has to be scaled to $X * \frac{1}{\rho_w} = 0.66 * X$ to fit the new picture dimensions.

Apart from the downsizing factor, the motion vector is also multiplied with its weight ω (6.5). Obtained motion vector is rounded to the nearest integer to form a valid motion vector. The weighted average approach to calculate transcoding motion vector is chosen because it gives the best trade-off between precision and computational complexity [42].

If the CU fits in one of the complex categories (HBC or HM) than the split decision is reevaluated, but only if the SC flag is set to 0. Pseudocode for deciding whether to split the current CU or not is given below:

```
if SC = 1
      Do not split

avgFirst = Find the average phase of the first two mapped motion vectors
avgLast = Find the average phase of the last two mapped motion vectors

difference = abs(avgFirst – avgLast)

if difference > π/2 AND difference < 3 * π /2
      Split CU
      Categorize child CUs
else
      Do not split CU

SC = 1
```

If the motion vectors of the first two and last two CUs point to different directions, then the split is performed, and the prediction process is repeated for four children CU blocks. Such example is given in Figure 8.5, where the motion vectors of mapped CUs, located at the bottom right area of the transcoded block, point to a different direction than the rest. The cause of this behavior in this particular block is the presence of a basketball that is being moved by a player, while the rest of the block is located in the picture background.

Figure 8.5: Difference in mapped motion vectors

If the mapped motion vectors have similar direction, the split is not performed, and the transcoding motion vector is being calculated for that CU. As a base for a calculation of motion vector for higher complex InterM CUs, weighted motion vector (8.3) is used. Unlike before, the weighted average MV is not simply taken as the best prediction. Instead, the refinement by ±2 pixel is carried out. The refinement process tests area around the block to which the weighted average motion vector is pointing and finds the best among them. In order to avoid interpolation, which is one of the computationally most demanding kernels in entire HEVC algorithm, only integer motion vectors are evaluated.

Otherwise, if the InterM CU belongs to both, MM and MBC category, the same algorithm with the refinement is used, but with the reduced search area around the weighted motion vector. Refinement area in such cases is ±1.

Described inter prediction algorithm tries to minimize the number of searches for each block, since that process includes fetching the block from the reference frame and comparing that block to the one in the original frame and can thus be very memory and data intensive. Reusing motion vectors from the decoded frame helps to focus the search area to more relevant parts of the frame and to reduce the number of evaluated prediction candidates.

8.3.3 Prediction for ComboIntra category

For ComboIntra category, both prediction modes are tested in all cases. However, since the number of intra modes in mapped CUs is higher than average (as set by a γ coefficient), more focus is set on intra modes, although, for the most complex CUs within this category larger number of intra and inter candidates are tested. Prediction decision for ComboIntra category follows the same pattern as for IntraM and InterM and is depicted in Figure 8.6.

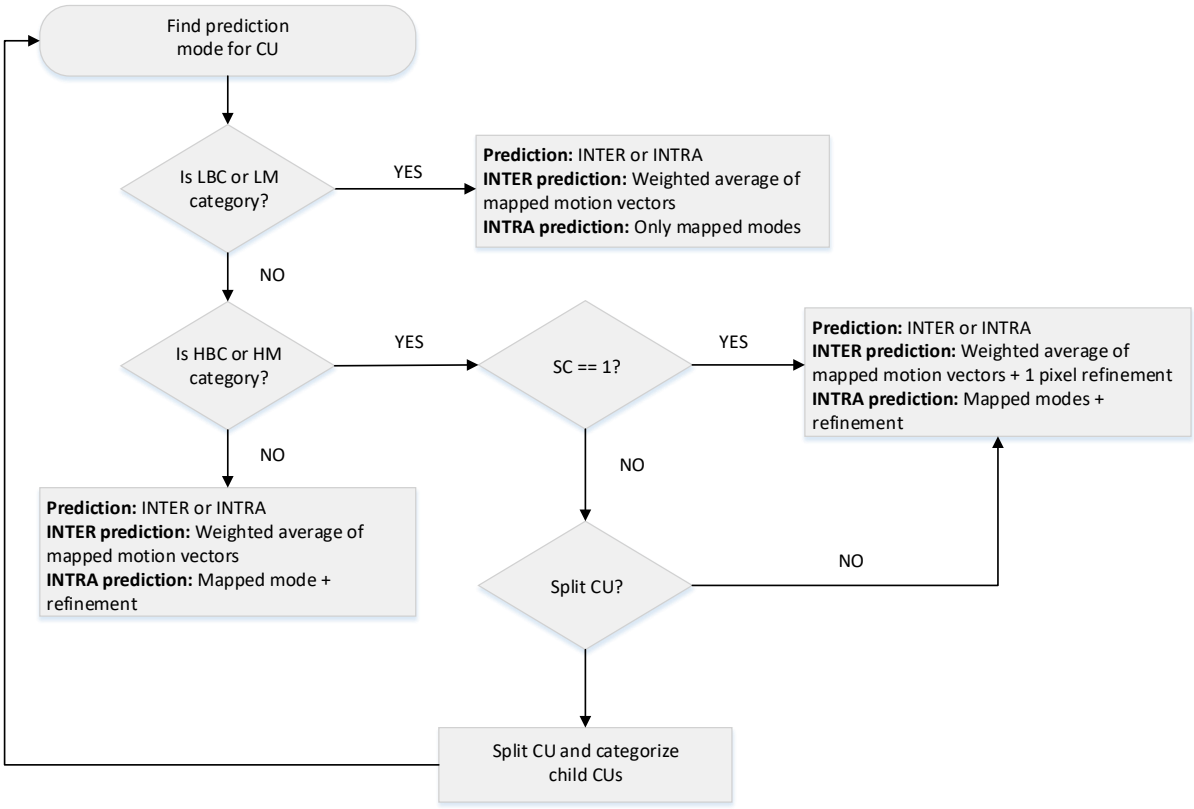

Figure 8.6: Prediction decision for ComboIntra category

For the least complex CUs within ComboIntra category, simple intra and inter prediction is conducted. Intra prediction tests only mapped intra modes in the same manner as for least complex IntraM CUs, while inter predicted checks the residual only for one motion vector that is calculated as a weighted average of mapped inter modes, as in InterM category. The prediction mode that gives the best residual is chosen as a final.

Split for High complex CUs in ComboIntra category is evaluated as follows:

```
if SC = 1
      Do not split

Initialize SplitFlag = 0
Check prediction modes of all mapped CUs

if the first three mapped CUs are INTER predicted
      SplitFlag = 1

if the last three mapped CUs are INTER predicted
      SplitFlag = 1

avgFirst = Find the average mode of first two INTRA mapped CUs
avgLast = Find the average mode of last two INTRA mapped CUs

difference = abs(avgFirst – avgLast)
```

```
if difference > (Number of INTRA modes / 2)
SplitFlag = 1

if SplitFlag = 1
      Split CU
      Categorize child CUs
else
      Do not split CU

SC = 1
```

If the first three or the last three mapped CUs are inter predicted, in a category that is mostly intra predicted, then the split is immediately performed. Otherwise, the same condition as for IntraM category is checked to evaluate the difference between the intra modes in the top left corner of the block (first two intra mapped CUs) and the bottom right corner of the block (last two intra mapped CUs). If the decision not to split CU has been made, intra prediction candidates of intra mapped CUs with the refinement of ±1 and inter prediction candidates of weighted average with ±1 refinement are tested.

Medium complexity CUs within ComboIntra category evaluate the same set of intra candidates as a higher complex ones, but limit the inter prediction only to the weighted average motion vector, without the refinement.

8.3.4 Prediction for ComboInter category

In ComboInter category, most of the mapped CUs are inter predicted, or at least a number of inter predicted mapped modes is higher than average for a given video sequence and the transcoding configuration. Therefore, the focus is set on inter prediction, while, for intra prediction, only mapped modes are tested, regardless of other categories. Figure 8.7 shows the flowchart for the prediction process of ComboInter category. As can be seen from the figure, intra mode evaluation is the same for all ComboInter CUs, while the inter prediction algorithm is very similar, but with different size of refinement area depending on the complexity. Most complex CUs have refinement area of ±2, medium complexity is evaluated with ±1 refinement area, while the least complex CU test only one motion vector in the same manner as for InterM and ComboIntra categories.

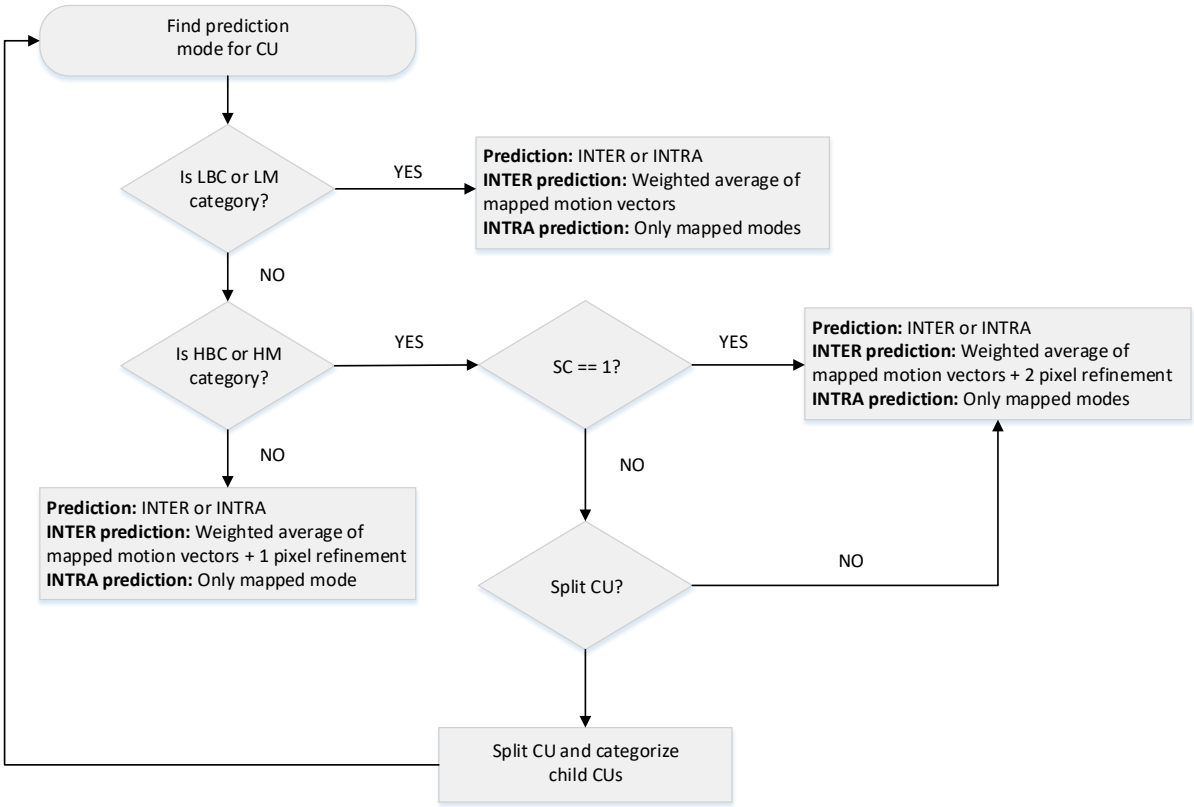

Figure 8.7: Prediction decision for ComboInter category

Split reevaluation for ComboInter CUs follows a similar concept as reevaluations of previously described mode-based categories:

```
if SC = 1
      Do not split

Initialize SplitFlag = 0
Check prediction modes of all mapped CUs

if the first three mapped CUs are INTRA predicted
      SplitFlag = 1

if the last three mapped CUs are INTRA predicted
      SplitFlag = 1

avgFirst = Find the average phase of the first two mapped motion vectors
avgLast = Find the average phase of the last two mapped motion vectors

difference = abs(avgFirst – avgLast)

if difference > π/2 AND difference < 3 * π /2
      SplitFlag = 1

if SplitFlag = 1
      Split CU
      Categorize child CUs
else
```

```
      Do not split CU

SC = 1
```

If the first or the last three mapped modes are intra predicted, in a category that has most inter predicted mapped modes, decision to split current CU is made. Also, if the difference in phase between motion vectors in the top left area of the block and the bottom right is larger than $\pi/2$ and smaller than $3*\pi/2$ (or between 90° and 270°) the split is performed. In all other cases, CU is not split further.

8.4 Determining coefficients

Before evaluating the proposed algorithm, the method for determining several adaptive coefficients introduced throughout this thesis is explained in this chapter. Adaptive coefficients include: β_L, β_H, μ_L and μ_H and δ.

8.4.1 Coefficient δ

As described in section 7.3, coefficient δ is used to divide ComboM category in two different categories based on the ratio (γ) of intra and inter CUs mapped from the decoded bitstream: ComboInter and ComboIntra. Changing this coefficient affects the number of CUs that fit in one of these categories. Increasing δ will result in a higher number of CUs being categorized as ComboIntra and vice versa. If the δ is set to 1, then the categorization is straight-forward; if the CU has more intra mapped CUs it is assigned to ComboIntra, otherwise, it belongs to ComboInter category. However, setting coefficient δ to 1 is not always the optimal solution. Depending on the video sequence and quantization parameter, frequency of occurrence of intra and inter modes in original bitstream can vary significantly. Some video sequences can have a higher number of intra predicted blocks than average (e.g., *Riverbed*) or a higher number of inter predicted block (e.g., *BlueSky*) as presented in Table 7.7 and Table 7.8. Thereby, it is important to identify those CUs who diverge from the average ratio within the same video sequence. For example, if the ratio γ is equal to 1, it does not have the same significance in the sequence where an average ratio is 0.5 and in the sequence with an average ratio is 4. In the first case, observed CU diverge from the average distribution by having more inter mapped CUs than most of other CUs within that video sequence. On the contrary, in the second video sequence, observed CU has much more intra predicted CUs than average, so that should be taken into consideration when determining prediction mode. This fact could infer that

setting the coefficient δ to the average ratio for each video sequence and configuration would be a good solution to distinguish CUs below and above average. However, the higher ratio still means that most of the mapped CUs are inter predicted and that the probability of inter prediction in the transcoding is higher, so setting the coefficient δ at the average ratio is still not the optimal solution. If this was the case, CU from the *BlueSky* video sequence (with γ=4) that has three times more inter predicted than intra predicted mapped CU blocks (γ=3) would be categorized as ComboIntra, which would set more focus in evaluating intra modes, which is not an ideal scenario. Therefore, the coefficient δ is calculated during the transcoding process and is adapted in runtime depending on the video sequence and the current configuration of the transcoder. Pseudocode of determining δ is given below.

```
Set δ_max = 2.0
Set δ_min = 0.5

If frameNum = 1
      Set δ_curr = 1

If frameNum % FR = 0
      Get average ratio γ

      If γ > δ_curr and δ_curr < δ_max
            δ_curr = δ_curr + (γ − δ_curr)/2

      Else if γ < δ_curr and δ_curr > δ_min
            δ_curr = δ_curr - (δ_curr − γ)/2

      If δ_curr > δ_max
            δ_curr = δ_max

      Else if δ_curr < δ_min
            δ_curr = δ_min

Calculate and update average ratio γ
```

At the beginning of the transcoding minimum and maximum possible values (δ_{max} and δ_{min}) of coefficient δ are set to 0.5 and 2.0 respectively. These boundaries are fixed so that every CU, regardless of the configuration and video sequence, that has more than the double of the amount of mapped CUs with certain prediction mode are assigned to the appropriate category. Setting these limits ensures that no CUs is wrongly categorized, which could happen if the δ is simply set to average γ as described in a paragraph above.

For the first several frames, distribution of intra and inter modes is not known, so the initial δ_{curr} is set to 1. After that, the coefficient δ_{curr} is adapted periodically on every frame

number that corresponds to frame rate (FR), meaning that for the video sequence with a frame rate of 30, δ_{curr} will be updated on every 30th frame. Adaptation is based on the statistics that is constantly calculated and updated during the transcoding phase. Ratio γ between intra and inter frames is updated after every frame and is examined during the adaptation process. If the ratio γ is larger than than δ_{curr} than the current value of the δ_{curr} is increased by half of the difference between the two values. Similarly, δ_{curr} is decreased by the same value if the ratio γ is smaller than δ_{curr}. If, at any point of the algorithm, the value δ_{curr} exceedes minimum and maximum extremes δ_{max} or δ_{min}, δ_{curr} is clipped to fit within the boundaries.

With the adaptive calculation of coefficient δ, CUs in the transcoded frame are categorized depending on the characteristics of the video sequence that is being transcoded and the current configuration of the encoder. This approach enables more precise categorization that can lead to better prediction evaluation.

8.4.2 Coefficients β_L,β_H,μ_L and μ_H

Coefficients $\beta_L$and β_H that are used for categorization based on bit complexity, and coefficients μ_L and $\mu_H$that are used for categorization based on a number of mapped CUs are also adaptive coefficients that have to be determined before running any evaluations of the proposed algorithm. As mentioned in sections 7.1 and 7.2 that describe the concept of categorization, changing coefficients β_L,β_H,μ_L and μ_H ultimately affect the complexity of the transcoding by distributing CUs to different categories. If the more CUs are located in more complex categories, such as HBC or HM, the computational complexity of the transcoder is increased, but the quality of the output bitstream should be higher. To observe the impact of changing values of the coefficients on the transcoding process, three fixed set of values were defined in Table 8.3.

Table 8.3: Sets of fixed coefficients

Set	β_L	β_H	μ_L	μ_H	Complexity
1	0.4	0.9	0.5	0.9	Low
2	0.3	0.6	0.3	0.6	Medium
3	0.1	0.5	0.2	0.5	High

The first set of coefficients sets a high boundary for most complex categories. Only the small number of CUs with the highest complexity will be categorized as HBC or HM, which is why the complexity of the first set is set as low. Set 3 has the opposite influence on categorization. With set 3 very few CUs will be assigned to low complex categories, so this set of coefficients is marked with high complexity. The second set from Table 8.3 puts the

boundaries between set 1 and set 3 and is referred to as a set with medium complexity. Notice that these are the same sets of coefficients that were used to demonstrate the initial split phase of the proposed algorithm in section 8.2.

Video transcoder was run with the proposed algorithm with all three sets of coefficients to obtain results that will be used to measure the impact of the defined coefficients on the processing time, quality and bitrate. Results are obtained for transcoding from original resolution to 1280x720 resolution and on two quantization parameters: 22 and 32. Due to simplicity, only the results for these two transcoding scenarios are shown, since the obtained conclusions can be mapped to all cases. Notice that these scenarios also include different width and height transcoding ratios as defined in Table 5.2. Tables below (Table 8.4 and Table 8.5) show the results for each of the quantization parameters. All the values are shown in relative when compared to the with Bolt JiT transcoder:

- Processing time – processing time (t) compared to Just-in-Time requirement t_{JiT} (section 5.4.1)
- PSNR - the difference in PSNR between the observed transcoder and Bolt JiT in dB
- Bitrate - the difference in bitrate compared to Bolt JiT in kilobits (negative values indicate better coding efficiency)

Table 8.4: Comparison of transcoding with fixed sets of coefficients (QP=22)

Video sequence	Set 1			Set 2			Set 3		
	Processing time [t / t_{JiT}]	**PSNR [ΔdB]**	**Bitrate [Δkbps]**	**Processing time [t / t_{JiT}]**	**PSNR [ΔdB]**	**Bitrate [Δkbps]**	**Processing time [t / t_{JiT}]**	**PSNR [ΔdB]**	**Bitrate [Δkbps]**
Shields	64.68%	1.131	-28507	65.73%	1.271	-28914	80.36%	1.303	-23774
ParkRun	69.78%	0.794	-28403	75.55%	0.827	-28233	83.86%	0.897	-25306
KristenAndSara	83.95%	0.628	-2428	89.39%	0.941	-2778	91.89%	1.37	-3113
Johnny	81.92%	0.519	-2163	87.17%	0.783	-2477	92.30%	1.08	-2841
FourPeople	83.40%	0.406	-1533	88.27%	0.73	-1940	90.99%	0.991	-2547
BasketballDrive	88.85%	0.967	-6650	98.75%	1.085	-5350	109.33%	1.051	-4098
Calendar	88.26%	0.964	-1102	92.29%	1.563	-1381	98.96%	1.582	-1472
Cactus	87.07%	0.412	-6022	89.13%	0.513	-6059	102.27%	0.494	-4847
BQTerrace	85.28%	0.548	-14744	87.92%	0.619	-13793	100.28%	0.615	-9988
RushHour	99.81%	0.818	-511	117.07%	1.114	-347	121.72%	1.134	-46
Riverbed	80.47%	1.69	-4025	87.19%	1.68	-4021	93.88%	1.647	-3773
PedestrianArea	89.57%	1.235	-1822	92.02%	1.47	-1773	93.20%	1.466	-1581
BlueSky	79.93%	0.575	-7453	87.81%	0.512	-4770	98.42%	0.508	-3705
Traffic	84.48%	0.367	-5253	87.40%	0.471	-5627	92.23%	0.459	-5171
DuckTakeOff	99.03%	0.164	-4212	144.25%	0.105	-702	154.77%	0.103	672
Bosphorus	99.60%	0.468	-7221	124.46%	0.687	-7796	138.62%	0.716	-5387
Beauty	98.01%	0.932	-1741	117.83%	0.993	-702	124.14%	1.017	193
Average	**86.12%**	**0.742**	**-7281**	**96.01%**	**0.904**	**-6862**	**103.96%**	**0.967**	**-5693**

Table 8.5: Comparison of transcoding with fixed sets of coefficients (QP =32)

Video sequence	Set 1			Set 2			Set 3		
	Processing time [t / t_{JiT}]	PSNR [Δ dB]	Bitrate [Δkbps]	Processing time [t / t_{JiT}]	PSNR [Δ dB]	Bitrate [Δkbps]	Processing time [t / t_{JiT}]	PSNR [Δ dB]	Bitrate [Δkbps]
Shields	33.36%	1.428	-6480	33.17%	1.603	-6825	31.34%	3.251	-8232
ParkRun	33.17%	1.564	-11463	33.95%	1.771	-11595	34.69%	2.125	-12166
KristenAndSara	59.61%	0.616	-456	60.73%	0.833	-498	65.64%	1.762	-698
Johnny	57.95%	0.543	-457	58.56%	0.746	-479	66.82%	1.545	-600
FourPeople	57.42%	0.475	-321	57.75%	0.766	-425	64.46%	1.229	-656
BasketballDrive	55.72%	1.288	-1186	59.06%	1.491	-548	63.97%	1.389	514
Calendar	59.44%	0.808	-309	61.84%	1.077	-116	72.16%	1.201	767
Cactus	46.81%	0.464	-1287	49.65%	0.65	-1012	55.01%	0.624	-369
BQTerrace	41.65%	0.641	-4516	44.25%	0.693	-3499	50.32%	0.639	-245
RushHour	69.20%	0.746	-135	72.79%	0.921	6	79.87%	0.965	267
Riverbed	52.08%	2.298	-1105	55.88%	2.345	-1004	62.04%	2.341	-841
PedestrianArea	58.19%	1.507	-423	61.19%	1.741	-356	67.41%	1.724	-156
BlueSky	39.54%	1.175	-3712	45.30%	0.768	-1637	54.79%	0.2	1895
Traffic	48.74%	0.25	-935	51.98%	0.495	-1010	54.42%	0.485	-1008
DuckTakeOff	67.85%	0.091	-676	74.69%	0.043	-162	84.82%	0.016	848
Bosphorus	76.17%	0.141	-573	76.76%	0.277	-314	86.01%	0.487	359
Beauty	70.62%	1.013	-488	76.23%	1.186	-247	87.83%	1.272	326
Average	**54.56%**	**0.885**	**-2030**	**57.28%**	**1.024**	**-1748**	**63.62%**	**1.250**	**-1176**

Presented results show the expected behavior in terms of processing time and video quality. In both cases (i.e., both quantization parameters) average processing time, as well as video quality, is increasing as the more complex set of coefficients is used. However, although average values follow expected behavior, this is not always the case for specific video sequences. Sometimes minimal gain in video quality can affect significant losses in bitrate (e.g., *Beauty* for QP =22), which imposes the question: is it worth to significantly sacrifice bitrate to get little better video quality. In some cases, using the more complex set does not give better quality nor better coding efficiency (*BlueSky* for QP =22), while in the other instances (e.g., *Calendar* for QP=22 or *ParkRun* for QP =32) using more complex set significantly improves both, video quality and coding efficiency. Trade-offs between video quality and coding efficiency can depend on the requirements of the system, that can favor one of the two. However, with Just-in-Time requirements, one condition must be met: processing time has to be below 100% of t_{JiT}. Therefore, using a high complex set of coefficients for transcoding some of the video sequences with the quantization parameter 22 is not possible because it would cause violation of timing constraints.

Giving that characteristics of transcoded bitstream highly depend on a particular video sequence, coefficients β_L,β_H,μ_L and μ_H are not fixed at the beginning of the transcoding process, but are adapted during the transcoding process as follows:

```
Set β_Lmin = 0.1, β_Lmax=0.5
Set β_Hmin = 0.5, β_Hmax=0.9
Set μ_Lmin = 0.2, μ_Lmax =0.5
Set μ_Hmin = 0.5, μ_Hmax =0.9
Set AdaptHigher = 1

If frameNum = 1
      Set β_Lcurr = 0.5
      Set β_Hcurr = 0.9
      Set μ_Lcurr = 0.5
      Set μ_Hcurr = 0.9


If frameNum % FR = 0
      time = Get processing time for last period
      diffFPS = (FR /time) - FR

      If diffFPS < 0
            If (-diffFPS) < 0.3 * FR
                  β_Hcurr=β_Hcurr+ (diffFPS/FR)
                  μ_Hcurr=μ_Hcurr+ (diffFPS/FR)
                  β_Lcurr=β_Hcurr+ (diffFPS/FR)
                  μ_Lcurr=μ_Lcurr+ (diffFPS/FR)
            Else
                  β_Lcurr = 0.5
                  β_Hcurr = 0.9
                  μ_Lcurr = 0.5
                  μ_Hcurr = 0.9

      Else if diffFPS < 0
            Calculate quality gains/losses from the last adaptations
            Calculate bitrate gains/losses from the last adaptations

            Revert = 0

            If quality worse and bitrate worse
                  Revert = 1
            If quality gains < 0.1dB and bitrate loss > 10%
                  Revert = 1

            If Revert = 1
                  Revert to best previous set of coefficients
            If Revert = 0
                  If AdaptHigher = 1
                        β_Hcurr= β_Hcurr- 0.1
                        μ_Hcurr=μ_Hcurr - 0.1
                        AdaptHigher = 0
                  Else
                        β_Lcurr=β_Hcurr - 0.1
                        μ_Lcurr=μ_Lcurr - 0.1
                        AdaptHigher = 1
```

```
        Clip β_Lcurr in range between β_Lmin and β_Lmax
        Clip β_Hcurr in range between β_Hmin and β_Hmax
        Clip μ_Lcurr in range between μ_Lmin and μ_Lmax
        Clip μ_Hcurr in range between μ_Hmin and μ_Hmax
        If β_Lcurr = β_Hcurr
              β_Lcurr = β_Lcurr - 0.1
        If μ_Lcurr= μ_Hcurr
              μ_Lcurr=μ_Lcurr - 0.1


Monitor video quality and bitrate
```

At the beginning of the process, when no information about the video sequence is available, coefficients are set for low complexity transcoding to ensure Just-in-Time execution at the start of the transcoding. On every frame number that is a multiplier of a frame rate of a video, coefficients are reevaluated and adapted. In the period between two adaptations, exactly N number of frames were processed, with N being frame rate, which means that the time between two evaluations must not exceed 1 second. Otherwise, Just-in-Time constraints are not met. Therefore, the difference between calculated time and maximum allowed time of 1 second is used to determine if JiT execution is satisfied and to decide how the coefficients will be adapted. In the above pseudocode, value *diffFPS* contains the difference between achieved and required fps (i.e., frames processed per second). If that value is smaller than 0 than the coefficients have to be adapted to increase the speed of the transcoder. To achieve this, β_L,β_H,μ_L and μ_H are increased so that the smaller number of CUs in future categorizations fit in high complex categories. Degree of increasing the coefficients is proportional to *diffFPS*. For example, if the last 30 frames in a video sequence that has a frame rate of 30 were processed in 1.1 s, *diffFPS* will be (30/1.1)-30 = -2,72 fps, meaning that the transcoder was too slow to achieve JiT, but it was relatively close. Thereby, coefficients are adapted just slightly by the value of (2.72/30) = 0.09. If the difference was higher than one third of the required fps than the coefficients are set to initial values. This scenario should not happen often in this algorithm, since the evaluation is performed periodically, and the execution time of the transcoder is constantly monitored. When the execution for JiT is already satisfied, then the possibility of improving video quality is considered. First step is to compare quality and bitrate with previously tested sets of coefficients. If one of the previous sets gave better results than currently tested in terms of video quality and/or coding efficiency while fulfilling JiT constraints, then it is chosen as a new set again. Otherwise, the new set is formed and tested by

adapting the current values of β_L,β_H,μ_L and μ_H. In this case, where the JiT is satisfied and where only the trade-off between bitrate and quality is considered, coefficients are decreased gradually, by alternately decreasing β_H and μ_H and than β_L and μ_L and overseeing impact on all of the characteristics. This way, as the video sequence progresses, coefficients are slowly advancing towards the most optimal solution for the given video sequence. Any sudden decrease of any of the considered coefficients in this phase could jeopardize JiT execution, which is why there are no adjustment larger than 0.1. Not only that this approach adjusts the algorithm for a specific video sequence but is also resistant to changes within the same sequence (e.g., change of scene). At the end of the adaptation process, each coefficient is validated to ensure that invalid set is not used for the categorization of future frames.

8.5 Final algorithm

After determining methods for computation of all the adaptive coefficients that are used to form the boundaries for the suggested categorization mechanism, all the aspects of the proposed algorithm are considered and set based on the analysis of the transcoding process. The high-level scheme of the final algorithm is given in Figure 8.8.

At the beginning of the transcoding, input bitstream is decoded frame by frame, and all the data from the decoded frame is gathered so it can be reused for improving the re-encoding phase. All the coefficients used in the proposed algorithm are set to their initial values. After decoding, the encoded frame is formed and split to the largest coding units – CTUs, that are immediately categorized based on the data previously obtained from the decoded frame. Each CTU is categorized based on three different information sets from the decoded input bitstream: number of bits, number of mapped CUs and prediction modes of mapped CUs. Depending on the categorization results, the decision to split CTUs to multiple 32x32, 16x16 and 8x8 blocks is made in the initial split phase of the algorithm, after which newly created blocks are categorized again. Prediction modes for each of the CUs are chosen based on the affiliation in one of the categories: InterM, IntraM, ComboInter, and ComboIntra. On every N^{th} frame, with N being frame rate, all the coefficients are updated depending on the current state of the transcoding process in terms of video quality, bitrate, and processing time.

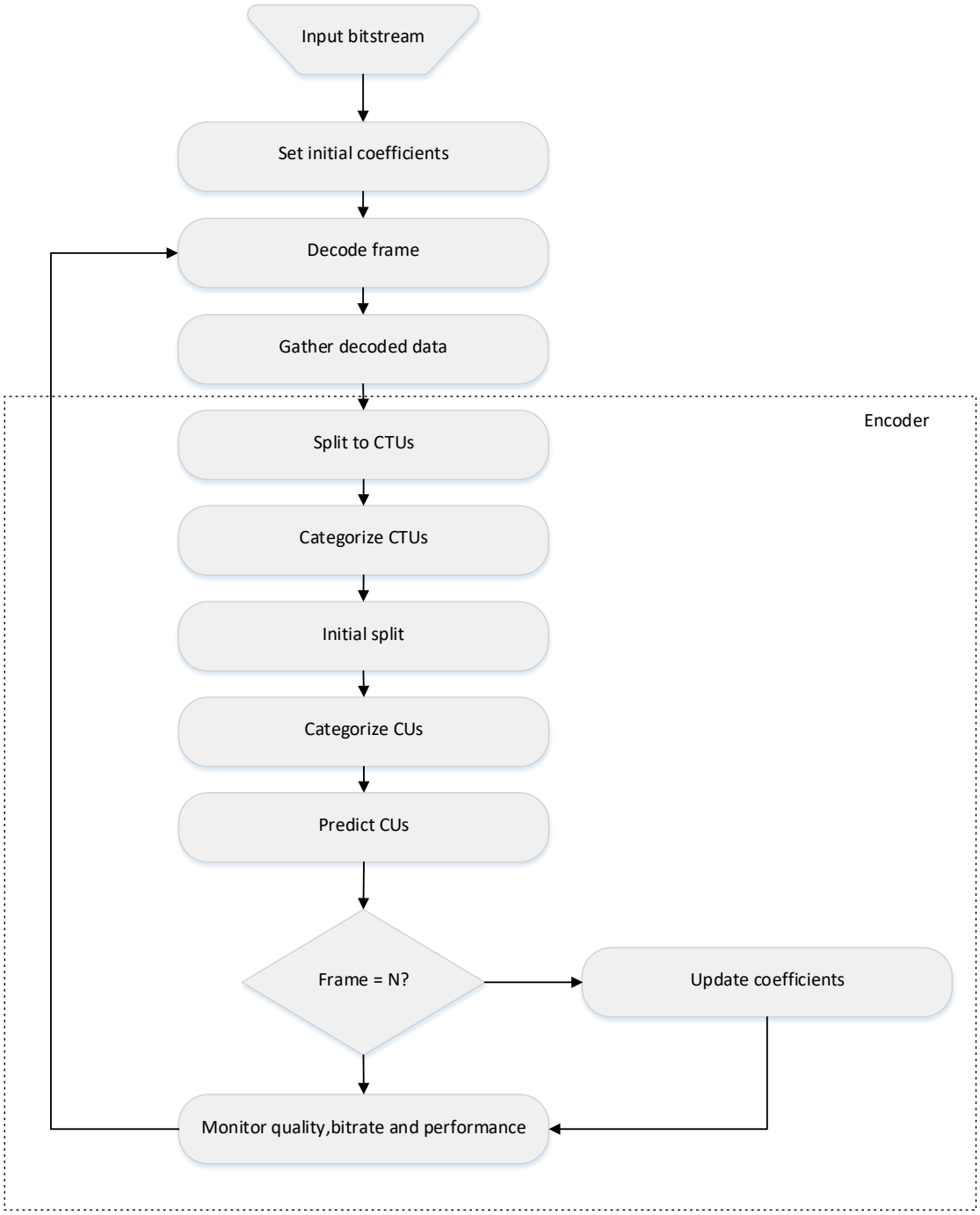

Figure 8.8: Final high-level diagram of the proposed algorithm

9 EXPERIMENTAL RESULTS ON CPU-ONLY ARCHITECTURE

This chapter presents experimental results of the proposed algorithm based on the categorization of the data from the decoded frame implemented on the CPU-only architecture.

9.1 Methodology

Algorithm for utilization of coding information from the input video stream is evaluated by running three different transcoders and comparing the obtained results. Scheme of the evaluation process is given in Figure 9.1.

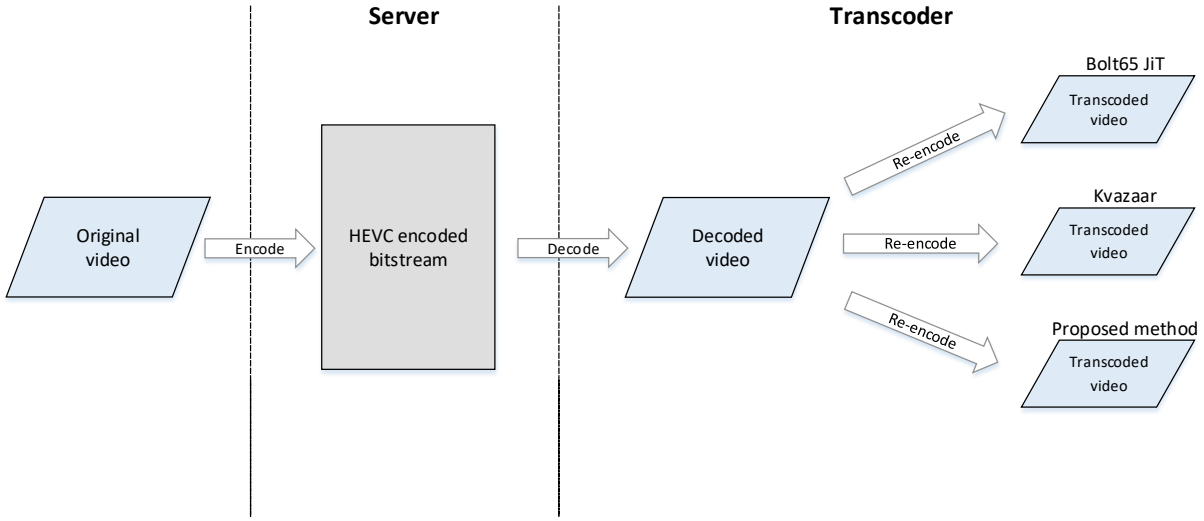

Figure 9.1: Evaluation scheme

The evaluation presented in the scheme above simulates the scenario of a real-world video content provider. Initially, the original video is encoded to the desired format and stored on the server. The original video is any video that is uploaded from the user and can be represented in several different formats and resolutions. After the initial encoding and storing the bitstream on a server, the original video is usually discarded, so all the quality loss imposed by the initial encoding is irreversible. Thereby, the usual practice of most video content providers is to store this video in highest quality so that every future transcoding does not introduce even larger quality degradations. Afterward, if the user requests a certain video, that video sequence is fetched from the server and transcoded on-the-fly based on the end user requirements.

In the evaluation of the proposed algorithm, original videos are 17 raw video sequences as listed in Table 5.1. All videos are encoded with Kvazaar and stored on the server and will be

used as an input for all observed transcoders. Generated input bitstreams are transcoded with three different transcoders to every possible resolution (Table 5.2), and the results are compared. Giving that the decoding part of the transcoder is always the same, i.e., it does not depend on the transcoding algorithm that is being used, only re-encoding phase for all transcoders was observed and evaluated in terms of video quality and bitrate. During the transcoding process, three main aspects were considered and assessed: processing time, bitrate and PSNR.

9.2 Comparison with Bolt65 JiT

Comparison with Bolt JiT transcoder (section 5.3.1) is conducted by comparing video quality and bitrate between transcoded bitstreams. In the analysis of different categorization coefficient sets (Table 8.4) it is shown that for every video sequence in the most complex transcoding mode, where the video is transcoded with the smallest width and height ratios and with quantization parameter 22, JiT execution is always satisfied for the low complex Set 1. However, video quality and bitrate can be increased for some video sequences by using more complex sets of coefficients without compromising JiT, which is why coefficients are dynamically adapted during the transcoding. Therefore, when comparing the proposed algorithm with Bolt JiT transcoder only the improvement in video quality (PSNR) and bitrate (kbps) is observed.

The difference in the bitrate is presented in two ways, as a percentage of improvement accomplished by using the proposed algorithm (negative values represent worse PSNR and better bitrate) and as a relative difference in bitrate. The reason behind representing data in this fashion is to get a better perspective of the obtained results. The small relative reduction in bitrate (e.g., 500 kbps) can in some cases signify major improvement, such as for small transcoding resolutions where overall bitrate is small. On the contrary, improvement by a small percentage can be relevant in some cases, when a large video file is being sent over the network. For the PSNR only relative improvement is shown since the percentages are always relatively small and do not represent meaningful information. Notice that larger PSNR indicates higher quality, while lower bitrate indicates better coding efficiency. Results here are shown only for the quantization parameter of 22 due to simplicity and better visibility of the presented data. Tables below show rwsults grouped by a resolution of the transcoded video, so in every table

different ratios ρ_w and ρ_h are present. Only valid transcoding options are shown, meaning that situations were video is upscaled to higher resolution are not considered.

Table 9.1: Proposed algorithm vs. Bolt65 Jit - transcoding to 2560x1600 with QP =22

Video sequence	Original resolution	ρ_w	ρ_h	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
DuckTakeOff	3840x2160	1.5	1.35	0.346	-14102	-5.98%
Bosphorus	3840x2160	1.5	1.35	0.265	-215	-28.09%
Beauty	3840x2160	1.5	1.35	1.336	-57767	-51.33%
Average				**0.649**	**-24028.0**	**-28.47%**

Table 9.2: Proposed algorithm vs. Bolt65 Jit - transcoding to 1920x1080 with QP =22

Video sequence	Original resolution	ρ_w	ρ_h	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
Traffic	2560x1600	1.33	1.48	0.445	-2600	-9.01%
DuckTakeOff	3840x2160	2	2	0.412	-5100	-4.04%
Bosphorus	3840x2160	2	2	0.640	-7464	-23.01%
Beauty	3840x2160	2	2	1.613	-9431	-19.00%
Average				**0.778**	**-6148.8**	**-13.77%**

Table 9.3: Proposed algorithm vs. Bolt65 Jit - transcoding to 1280x720 with QP =22

Video sequence	Original resolution	ρ_w	ρ_h	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
BasketballDrive	1920x1080	1.5	1.5	1.080	-5420	-16.71%
Calendar	1920x1080	1.5	1.5	1.601	-1354	-16.54%
Cactus	1920x1080	1.5	1.5	0.513	-5222	-25.60%
BQTerrace	1920x1080	1.5	1.5	0.620	-10243	-29.61%
RushHour	1920x1080	1.5	1.5	1.131	-94	-1.84%
Riverbed	1920x1080	1.5	1.5	1.688	-3910	-22.07%
PedestrianArea	1920x1080	1.5	1.5	1.472	-1640	-31.53%
BlueSky	1920x1080	1.5	1.5	0.571	-4680	-17.27%
Traffic	2560x1600	2	2.22	0.470	-5127	-37.21%
DuckTakeOff	3840x2160	3	3	0.166	-300	-0.83%
Bosphorus	3840x2160	3	3	0.685	-5941	-35.84%
Beauty	3840x2160	3	3	1.038	-380	-3.08%
Average				**0.920**	**-3692.6**	**-19.84%**

Table 9.4: Proposed algorithm vs. Bolt65 Jit - transcoding to 704x576 with QP =22

Video sequence	Original resolution	ρ_w	ρ_h	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
Shields	1280x720	1.82	1.25	0.686	-5818	-36.72%
ParkRun	1280x720	1.82	1.25	0.443	-7390	-20.73%
KristenAndSara	1280x720	1.82	1.25	0.669	-894	-22.46%
Johnny	1280x720	1.82	1.25	0.479	-577	-17.43%
FourPeople	1280x720	1.82	1.25	0.597	-876	-23.78%
BasketballDrive	1920x1080	2.72	1.875	1.032	-2020	-15.54%
Calendar	1920x1080	2.72	1.875	1.441	-939	-15.56%
Cactus	1920x1080	2.72	1.875	0.540	-2518	-18.64%
BQTerrace	1920x1080	2.72	1.875	0.691	-5419	-21.35%
RushHour	1920x1080	2.72	1.875	0.984	-298	-7.03%
Riverbed	1920x1080	2.72	1.875	1.103	-1996	-20.85%
PedestrianArea	1920x1080	2.72	1.875	1.330	-1014	-30.46%
BlueSky	1920x1080	2.72	1.875	0.428	-2088	-13.83%
Traffic	2560x1600	3.63	2.77	0.470	-3444	-44.58%
DuckTakeOff	3840x2160	5.45	3.75	0.055	-852	-3.61%
Bosphorus	3840x2160	5.45	3.75	0.385	-4442	-26.70%

Beauty	3840x2160	5.45	3.75	0.936	-601	-1.84%
Average				**0.722**	**-2422.7**	**-20.07%**

Table 9.5: Table 9.4: Proposed algorithm vs. Bolt65 Jit - transcoding to 640x480 with QP =22

Video sequence	**Original resolution**	ρ_w	ρ_h	**PSNR [Δ dB]**	**Bitrate [Δ kbps]**	**Bitrate [%]**
Shields	1280x720	2	1.5	0.815	-11964	-79.88%
ParkRun	1280x720	2	1.5	0.542	-9743	-48.56%
KristenAndSara	1280x720	2	1.5	0.828	-1343	-42.26%
Johnny	1280x720	2	1.5	0.568	-787	-33.25%
FourPeople	1280x720	2	1.5	0.750	-1351	-43.90%
BasketballDrive	1920x1080	3	2.25	1.003	-2762	-28.13%
Calendar	1920x1080	3	2.25	1.431	-1528	-33.13%
Cactus	1920x1080	3	2.25	0.564	-2816	-31.28%
BQTerrace	1920x1080	3	2.25	0.698	-5141	-37.05%
RushHour	1920x1080	3	2.25	0.956	-464	-17.83%
Riverbed	1920x1080	3	2.25	0.895	-1462	-18.32%
PedestrianArea	1920x1080	3	2.25	1.312	-1142	-44.54%
BlueSky	1920x1080	3	2.25	0.511	-3409	-39.31%
Traffic	2560x1600	4	3.33	0.480	-3253	-53.27%
DuckTakeOff	3840x2160	6	4.5	0.060	-1067	-4.73%
Bosphorus	3840x2160	6	4.5	0.487	-4099	-39.40%
Beauty	3840x2160	6	4.5	0.941	-489	-6.52%
Average				**0.755**	**-3107.1**	**-35.37%**

Tables displayed above show that for every video sequence and every pair of original and transcoded resolutions proposed algorithm gives transcoded bitstream with the better video quality, where, in the same time, the coding efficiency, represented by bitrate, of the transcoder that implements this algorithm is also higher for every transcoding scenario observed in the scope of this analysis.

For smaller transcoding ratios, as in most of the examples from Table 9.1, Table 9.2 and Table 9.3, advantages of using the proposed algorithm is higher, which can be seen from the average values of PSNR. This behavior can be expected since the data mapped from the decoded frame provides more meaningful information for the decisions in re-encoding phase. For example, when transcoding from 3840x2160 to 640x480, where ρ_w equals to 6 and ρ_h equals 4.5 small area in the transcoded frame is mapped to rather large area of the original frame. One 32x32 block is such case is mapped to a 192x144 size area, which makes decisions in the re-encoding phase less precise. Nevertheless, results for high ratio transcoding are still significantly better than Bolt65 JiT. Figure 9.2 shows the trend of PSNR improvement when increasing transcoding ratio for several video sequences. Transcoding ratio is calculated as a product of width and height ratios.

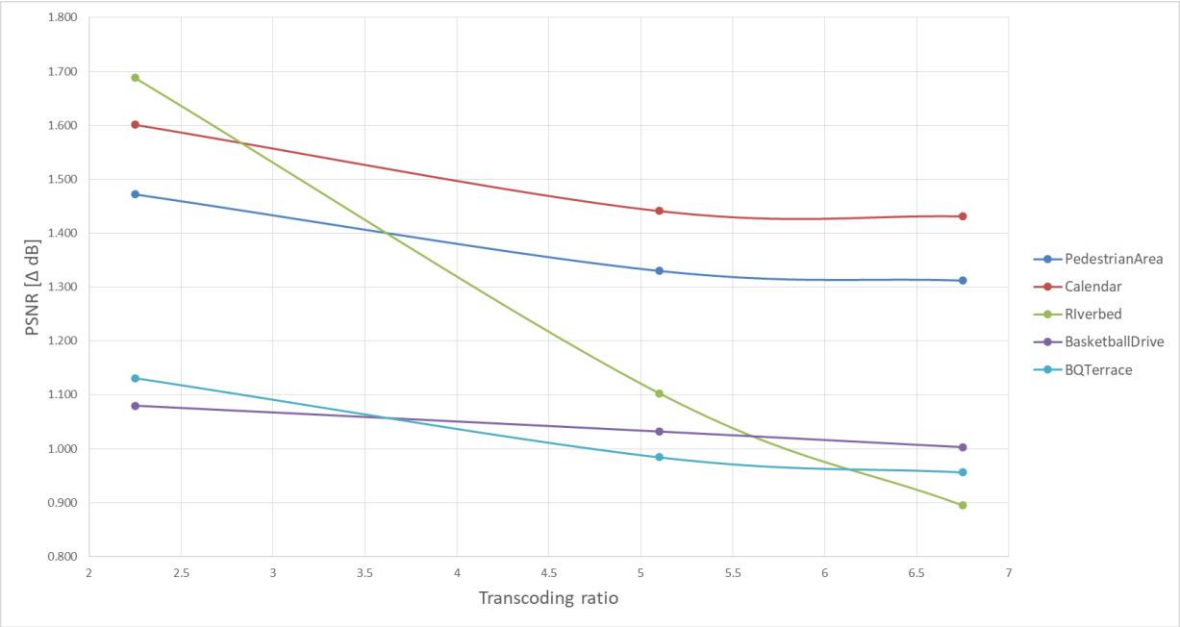

Figure 9.2: Improvements in PSNR depending on transcoding ratio

As it can be observed from the presented graph, improvements in PSNR slowly decrease as the transcoding ratio grows. On the contrary, bitrate reduction grows with the transcoding ratio as shown in Figure 9.3. Therefore, if the frame is downscaled by a larger factor, gains in video quality are smaller, but the bitrate is reduced significantly, which demonstrates the trade-off between coding efficiency and video quality achieved with the proposed algorithm.

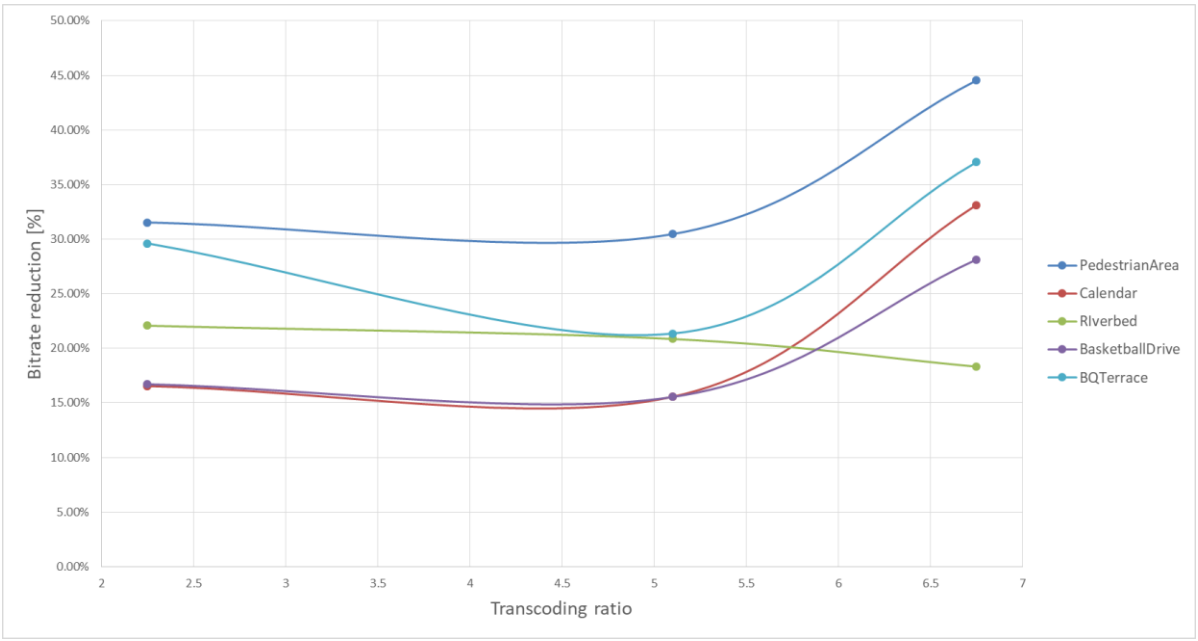

Figure 9.3: Bitrate reduction depending on transcoding ratio

Overall gain per video sequence is calculated as an average gain of all possible transcoding scenarios for the particular sequence and is given in the following table.

Video sequence	Original resolution	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
Shields	1280x720	0.750	-8891.0	-58.30%
ParkRun	1280x720	0.493	-8566.5	-34.65%
KristenAndSara	1280x720	0.749	-1118.5	-32.36%
Johnny	1280x720	0.524	-682.0	-25.34%
FourPeople	1280x720	0.674	-1113.5	-33.84%
BasketballDrive	1920x1080	1.038	-3400.7	-20.13%
Calendar	1920x1080	1.491	-1273.7	-21.74%
Cactus	1920x1080	0.539	-3518.7	-25.17%
BQTerrace	1920x1080	0.670	-6934.3	-29.34%
RushHour	1920x1080	1.024	-285.3	-8.90%
Riverbed	1920x1080	1.229	-2456.0	-20.41%
PedestrianArea	1920x1080	1.371	-1265.3	-35.51%
BlueSky	1920x1080	0.503	-3392.3	-23.47%
Traffic	2560x1600	0.473	-3941.3	-45.02%
DuckTakeOff	3840x2160	0.208	-1829.8	-3.84%
Bosphorus	3840x2160	0.492	-4432.2	-30.61%
Beauty	3840x2160	1.173	-13733.6	-16.35%
Average		**0.788**	**-3931.5**	**-27.35%**

Figure 9.4: Average gains per video sequence compared with Bolt65 JiT

The proposed algorithm achieves 0.788 dB better PSNR on average for all tested transcoding scenarios. The most considerable improvement is for *Calendar* video sequence where there is 1.491 dB difference compared with Bolt65 JiT, while the smallest improvement can be seen for *DuckTakeOff* sequence where PSNR is improved by 0.208 dB. Regarding the bitrate, the biggest reduction of 58.30% is achieved for *Shields* and the smallest of 3.84 % for *DuckTakeOff* video sequence. Average bitrate savings are 27.35%.

9.3 Comparison with Kvazaar

In order to analyze the degradation of transcoded bitstream when compared with encoders that do not have Just-in-Time requirements, Bolt65 JiT transcoder with the implemented proposed algorithm is compared with open-source Kvazaar encoder. Since there is no Kvazaar decoder or transcoder available, input bitstream was decoded with Bolt65 software suite, and the decoded video was set as an input to Kvazaar encoder. Besides the difference in PSNR and bitrate that were shown in the previous comparison, execution speedup was also analyzed. Speedup represents the difference in the processing times between two transcoders, where the value of 2.0x means that transcoder with the proposed algorithm is two times faster than Kvazaar. Tables below depict results for each transcoding resolution.

Table 9.6: Proposed algorithm vs Kvazaar- transcoding to 2560x1600 with QP=22

Video sequence	Original resolution	ρ_w	ρ_h	Speedup	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
DuckTakeOff	3840x2160	1.5	1.35	3.11x	-0.940	-16384	-7.06%
Bosphorus	3840x2160	1.5	1.35	3.51x	-1.280	25680	33.22%
Beauty	3840x2160	1.5	1.35	4.13x	-0.416	-15141	-21.46%
Average				**3.58x**	**-0.879**	**-1948.3**	**1.57%**

Table 9.7: Proposed algorithm vs Kvazaar- transcoding to 1920x1080 with QP=22

Video sequence	Original resolution	ρ_w	ρ_h	Speedup	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
Traffic	2560x1600	1.33	1.48	2.15x	-0.618	11170	37.20%
DuckTakeOff	3840x2160	2	2	2.07x	-0.770	-7076	-5.88%
Bosphorus	3840x2160	2	2	2.32x	-0.644	7681	22.86%
Beauty	3840x2160	2	2	3.27x	-0.013	-95	-0.18%
Average				**2.45x**	**-0.511**	**2920.0**	**13.50%**

Table 9.8: Proposed algorithm vs Kvazaar- transcoding to 1280x720 with QP=22

Video sequence	Original resolution	ρ_w	ρ_h	Speedup	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
BasketballDrive	1920x1080	1.5	1.5	2.46x	1.009	5043	21.95%
Calendar	1920x1080	1.5	1.5	2.45x	-1.310	4387	41.55%
Cactus	1920x1080	1.5	1.5	2.20x	-0.533	4321	27.11%
BQTerrace	1920x1080	1.5	1.5	2.04x	-0.682	6642	27.47%
RushHour	1920x1080	1.5	1.5	2.65x	-1.050	759	15.12%
Riverbed	1920x1080	1.5	1.5	3.43x	-1.020	-579	-3.35%
PedestrianArea	1920x1080	1.5	1.5	2.69x	-0.638	1293	22.70%
BlueSky	1920x1080	1.5	1.5	2.11x	-1.422	4368	21.10%
Traffic	2560x1600	2	2.22	2.00x	-0.534	6320	45.75%
DuckTakeOff	3840x2160	3	3	1.82x	-0.919	-2319	-4.29%
Bosphorus	3840x2160	3	3	2.03x	-0.441	6700	34.06%
Beauty	3840x2160	3	3	2.79x	-0.114	3293	20.36%
Average				**2.39x**	**-0.638**	**3352.3**	**22.46%**

Table 9.9: Proposed algorithm vs Kvazaar- transcoding to 704x576 with QP=22

Video sequence	Original resolution	ρ_w	ρ_h	Speedup	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
Shields	1280x720	1.82	1.25	1.74x	-0.506	13536	47.77%
ParkRun	1280x720	1.82	1.25	1.91x	-0.639	11770	30.64%
KristenAndSara	1280x720	1.82	1.25	2.39x	-1.320	1653	42.81%
Johnny	1280x720	1.82	1.25	2.36x	-0.645	1385	44.59%
FourPeople	1280x720	1.82	1.25	2.41x	-1.067	1809	44.64%
BasketballDrive	1920x1080	2.72	1.875	2.16x	-1.017	4790	33.33%
Calendar	1920x1080	2.72	1.875	2.19x	-1.638	3632	54.55%
Cactus	1920x1080	2.72	1.875	2.04x	-0.795	4427	32.08%
BQTerrace	1920x1080	2.72	1.875	1.80x	-0.198	8702	38.37%
RushHour	1920x1080	2.72	1.875	2.42x	-0.710	854	28.17%
Riverbed	1920x1080	2.72	1.875	3.05x	-0.879	-295	-3.08%
PedestrianArea	1920x1080	2.72	1.875	2.46x	-0.682	1176	31.86%
BlueSky	1920x1080	2.72	1.875	1.75x	-1.350	6790	42.15%
Traffic	2560x1600	3.63	2.77	1.76x	-0.542	3469	45.34%
DuckTakeOff	3840x2160	5.45	3.75	1.56x	-0.828	-286	-1.10%
Bosphorus	3840x2160	5.45	3.75	1.66x	-0.574	5861	43.50%
Beauty	3840x2160	5.45	3.75	2.50x	-0.234	2268	26.75%
Average				**2.13x**	**-0.801**	**4208.3**	**34.26%**

Table 9.10: Proposed algorithm vs Kvazaar- transcoding to 640x480 with QP=22

Video sequence	Original resolution	ρ_w	ρ_h	Speedup	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
Shields	1280x720	2	1.5	1.91x	-0.404	3986	29.66%
ParkRun	1280x720	2	1.5	2.01x	-0.364	2771	12.62%
KristenAndSara	1280x720	2	1.5	2.29x	-1.210	1126	38.70%
Johnny	1280x720	2	1.5	2.30x	-0.596	2776	44.97%
FourPeople	1280x720	2	1.5	2.33x	-1.083	1268	40.00%
BasketballDrive	1920x1080	3	2.25	2.04x	-0.969	2899	31.08%
Calendar	1920x1080	3	2.25	2.03x	-1.420	2842	53.63%
Cactus	1920x1080	3	2.25	1.96x	-0.740	2685	31.97%
BQTerrace	1920x1080	3	2.25	1.70x	-0.295	5128	36.17%
RushHour	1920x1080	3	2.25	2.22x	-0.678	628	27.93%
Riverbed	1920x1080	3	2.25	2.76x	-0.833	-286	-3.60%
PedestrianArea	1920x1080	3	2.25	2.30x	-0.628	867	29.08%
BlueSky	1920x1080	3	2.25	1.74x	-1.210	3850	46.20%
Traffic	2560x1600	4	3.33	1.69x	-0.484	5846	43.49%
DuckTakeOff	3840x2160	6	4.5	1.48x	-0.722	-428	-2.15%
Bosphorus	3840x2160	6	4.5	1.57x	-0.386	3958	42.76%
Beauty	3840x2160	6	4.5	1.99x	-0.264	1881	27.15%
Average				**2.02**	**-0.723**	**2458.6**	**31.16%**

As can be concluded from the presented results, video quality is higher for all video sequences and all transcoding resolution, while bitrate is better in most of the cases when using Kvaazar encoder. This behavior can be expected since the focus of the Kvazaar is aimed at achieving higher video quality and coding efficiency, which is not a case in transcoders whose primary goal is to satisfy timing requirements. Consequently, transcoder based on the proposed algorithms achieves significant speedups, from 3.58x on average when transcoding to 2560x1600 resolution, to approximately 2x for downsizing to smaller resolutions (704x576 and 640x480). Average losses and speedups per video sequence, as well as overall statistics, are shown in Table 9.11.

Table 9.11: Average losses per video sequence compared to Kvazaar

Video sequence	Original resolution	Speedup	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
Shields	1280x720	1.83x	-0.455	8761.0	38.72%
ParkRun	1280x720	1.96x	-0.502	7270.5	21.63%
KristenAndSara	1280x720	2.34x	-1.265	1389.5	40.76%
Johnny	1280x720	2.33x	-0.621	2080.5	44.78%
FourPeople	1280x720	2.37x	-1.075	1538.5	42.32%
BasketballDrive	1920x1080	2.22x	-0.326	4244.0	28.79%
Calendar	1920x1080	2.22x	-1.456	3620.3	49.91%
Cactus	1920x1080	2.07x	-0.689	3811.0	30.39%
BQTerrace	1920x1080	1.85x	-0.392	6824.0	34.00%
RushHour	1920x1080	2.43x	-0.813	747.0	23.74%
Riverbed	1920x1080	3.08x	-0.911	-386.7	-3.34%
PedestrianArea	1920x1080	2.48x	-0.649	1112.0	27.88%

BlueSky	1920x1080	1.87x	-1.327	5002.7	36.48%
Traffic	2560x1600	1.90x	-0.545	6701.3	42.95%
DuckTakeOff	3840x2160	2.01x	-0.836	-5298.6	-4.10%
Bosphorus	3840x2160	2.22x	-0.665	9976.0	35.28%
Beauty	3840x2160	2.94x	-0.208	-1558.8	10.52%
Average		**2.24x**	**-0.749**	**3284.4**	**29.45%**

Highest loss in video quality of 1.456 dB between the transcoded bitstreams can be observed for video sequence *Calendar*, while the smallest difference of 0.208 dB is when transcoding *Beauty* video sequence. The overall average loss for all transcoding scenarios is 0.749 dB. Regarding the bitrate, some video sequences achieve even better bitrate when using the proposed algorithm, such as *Riverbed*, *DuckTakeOff*, and *Beauty*, while the highest bitrate loss of almost 50% can be seen when transcoding *Calendar*. Average bitrate loss is approximately 30%. These losses, however, come with the increased transcoding speed, which is imperative if the goal is to achieve JiT encoding. Average speedup of the proposed algorithm for all scenarios is 2.24x, which mostly depend on the video resolution to which original bitstream is being transcoded.

9.4 Comparison with State-of-the-art algorithms

Most of the research activities in the area of video transcoding are focused on speeding up the process of transcoding, but very few of them cover same aspects that are considered in this thesis, i.e., Just-in-Time video transcoding and homogeneous transcoding based on HEVC standard.

Research presented [47] and [48] have similar approaches to transcoding, where a single video is pre-transcoded to several versions and only specific data, such as information about motion vectors, is stored on the server. When a low fidelity version of the video stream is requested from the system, the original video is transcoded by using pre-stored motion vectors. These approaches decrease the computational complexity of transcoder since they avoid complex motion estimation but increase storage costs because additional information for all different versions of the video has to be stored on a server. While authors in [47] verify their approach by comparing bitrate to scalable coding, without taking into consideration Just-in-Time execution, authors in [48] are focused on achieving Just-in-Time transcoding. However, both solutions increase storage costs and do not enable transcoding to arbitrary ratios. Instead, only the versions for which pre-transcoding was performed are available. Since most of the transcoding process is performed upfront, the results are not comparable with the algorithm proposed in this thesis.

In [49] authors present different transcoding techniques that can reduce the transcoder complexity in both CU and PU optimization levels. The fastest proposed approach is able to reduce the complexity of the transcoder by 83% while keeping the bitrate loss below 3%. However, time savings are presented in relative to other transcoders and Just-in-Time execution is not addressed. Also, presented techniques do not include spatial resolution reduction, which is one of the main aspects of this thesis.

10 HARDWARE ACCELERATOR FOR INTER PREDICTION

Achieving JiT transcoding requires a lot of sacrifices that have to be made in a re-encoding process to conform to imposed timing requirements, which ultimately affects video quality and coding efficiency of the transcoding process. Exploiting coding information from the decoded frame helps to improve the processing time of the transcoding but can still not ensure JiT transcoding. The software-based algorithm proposed in this thesis reuses information from the decoded bitstream to re-encode original bitstream while monitoring and adapting the computational complexity of the transcoding process to guarantee JiT execution. However, during the design of the algorithm, some functionalities had to be reduced to ensure the predictability of the execution. Using hardware accelerator for some of the functions and kernels in the transcoding process could help to achieve the same results in terms of timing, but by using a broader set of functionalities that can further improve quality of output bitstream.

Therefore, analysis is made to identify suitable kernels in the application that could be accelerated in hardware to enhance final transcoded bitstream. Results of the analysis are shown in chapter 10.1. Overview of the functionality of hardware accelerator is given in 10.2. Architecture and implementation of a hardware accelerator for the selected kernel are presented in chapters 10.3 and 10.4 while the performance validation of stand-alone hardware accelerator is shown in chapter 10.5.

10.1 Kernel analysis

Complexity analysis of HEVC encoder shows that one of the most exhaustive kernels in the encoding process is inter prediction that consists of motion estimation and motion compensation. Depending on the configuration of the encoder motion estimation can consume up to 85% of overall encoding time [11]. Although in Just-in-Time transcoding share of motion estimation is not nearly as high, because faster motion estimation algorithms are employed to reduce the processing time, this fact emphasizes the importance of inter prediction on quality of output bitstream. More precise inter prediction process is able to find a better predicted block, forming residual with smaller prediction errors.

The algorithm proposed in this thesis retrieves motion vectors found in a motion estimation process in the original encoding and reuses them to facilitate inter prediction in the re-encoding phase of the transcoder. Motion estimation in the defined algorithm depends on the complexity of currently observed CU which is determined based on its categorization. For more

complex CUs more precise refinement is conducted, while for less complex CUs, a motion vector is calculated as the weighted average of appropriate motion vectors from the original bitstream. Refinement steps are minimal, increasing the search area around the considered CU by 1 or 2 pixels for higher complexity, to ensure fast execution. However, by adding hardware accelerator modules specifically designed for inter prediction, the refinement area could be widened for all CU categories, without sacrificing the performance of the transcoder. To test the behavior of the transcoder and to verify benefits of increasing the refinement search area and meaningfulness of creating special hardware accelerator for this specific purpose, inter prediction of the proposed algorithm is adapted as listed in Table 10.1.

Table 10.1: Inter prediction mode adaptation

Prediction category	Condition	SW inter prediction	HW inter prediction
InterM	LBC or LM	Weighted MV	Weighted MV + 1 pixel refinement
InterM	MBC and MM	Weighted MV + 1 pixel refinement	Weighted MV + 2 pixel refinement
InterM	HBC or HM	Weighted MV + 2 pixel refinement	Weighted MV + 4 pixel refinement
ComboIntra	LBC or LM	Weighted MV	Weighted MV + 1 pixel refinement
ComboIntra	MBC and MM	Weighted MV	Weighted MV + 1 pixel refinement
ComboIntra	HBC or HM	Weighted MV + 1 pixel refinement	Weighted MV + 2 pixel refinement
ComboInter	LBC or LM	Weighted MV	Weighted MV + 1 pixel refinement
ComboInter	MBC and MM	Weighted MV + 1 pixel refinement	Weighted MV + 2 pixel refinement
ComboInter	HBC or HM	Weighted MV + 2 pixel refinement	Weighted MV + 4 pixel refinement

Refinement area larger than four pixels was not considered since it has been shown in previous research [3] that the overall gain in video quality becomes almost negligible when increasing the search window by more than four pixels.

Transcoder was run with both version of inter prediction: software-based version as described in previous chapters in this thesis and proposed hardware-based version with adapted inter prediction. To ensure comparable results, coefficients β_L,β_H,μ_L and μ_H are set to fixed values defined with set 1 (Table 8.3). Otherwise, the difference in video quality and/or bitrate would be caused by a different distribution of CUs into categories. Average results per video sequence are shown in Table 10.2, where the differences in PSNR and bitrate are compared with the software version of the algorithm, while the processing time is compared to t_{JiT} to show if the JiT execution is satisfied.

Table 10.2: Comparison between SW and proposed HW algorithm with adapted inter prediction

Video sequence	Original resolution	Processing time [t / t_{JiT}]	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]
Shields	1280x720	70.40%	0.248	-7357.0	-26.24%
ParkRun	1280x720	81.56%	0.128	-7767.0	-20.39%
KristenAndSara	1280x720	98.12%	0.022	-577.5	-15.10%
Johnny	1280x720	96.53%	0.074	-466.0	-15.14%
FourPeople	1280x720	99.30%	0.102	-270.5	-7.29%
BasketballDrive	1920x1080	109.11%	0.134	-533.3	-4.49%
Calendar	1920x1080	104.03%	0.160	-178.3	-2.78%
Cactus	1920x1080	101.97%	0.118	-1711.0	-10.90%
BQTerrace	1920x1080	98.60%	0.138	-2715.3	-9.49%
RushHour	1920x1080	125.16%	0.122	-48.3	-2.12%
Riverbed	1920x1080	98.12%	0.127	-384.0	-3.35%
PedestrianArea	1920x1080	103.53%	0.110	-373.3	-9.55%
BlueSky	1920x1080	103.60%	0.096	-3852.3	-19.75%
Traffic	2560x1600	101.25%	0.091	-1548.3	-13.83%
DuckTakeOff	3840x2160	164.30%	0.082	-1693.2	-3.28%
Bosphorus	3840x2160	140.12%	0.252	-7041.8	-24.61%
Beauty	3840x2160	129.52%	0.168	-54.6	-0.35%
Average		**107.37%**	**0.128**	**-2144.9**	**-11.10%**

Results show that by increasing the refinement search area by 1 pixel for low and medium complex CUs and by 2 pixels for complex CUs PSNR increases by 0.128 dB, while bitrate reduces for 11% on average. However, Just-in-Time requirement is not satisfied for most of the video sequences, giving that average processing time is 107.37% of t_{JiT} , which is an increase from 96.01% for the original algorithm with the same set of coefficients (Table 8.4, set 2). With presented results, a conclusion can be reached, that increasing the refinement area in inter prediction can lead to notable improvements of the final bitstream, but in order to keep the algorithm within the same timing constraints this operation should be accelerated. Therefore, a custom hardware accelerator for inter prediction operation is designed, implemented and presented in the following subchapters.

10.2 Functionality

The core functionality of a custom hardware accelerator that is designed, implemented and integrated with JiT transcoder in the scope of this thesis is to find best inter predicted block based on motion vector obtained as a weighted average of mapped motion vectors and the defined refinement search area. Figure 10.1 depicts the inter prediction scheme that is employed as a custom hardware accelerator.

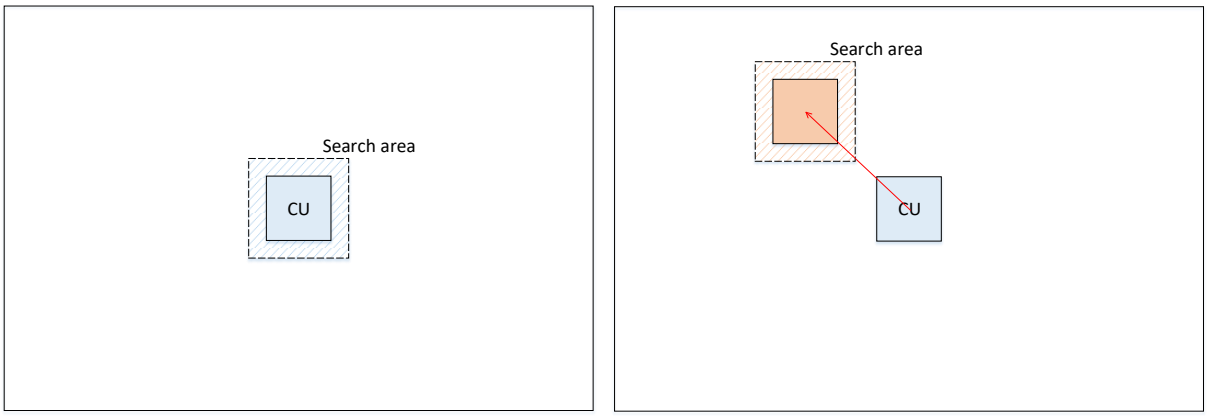

Figure 10.1: Inter prediction search scheme implemented in hardware accelerator

The left part of the figure shows a search area in regular motion estimation for the current CU block in the frame. Usually, the area in a reference frame that corresponds to the position around that exact block in the current frame is searched for the best candidate. However, in the transcoding, information about motion vectors from the original bitstream can be used to steer the search to an area where there is a higher probability for finding better inter predicted candidate. Therefore, in the proposed algorithm search area is located around the block to which weighted motion vector is pointing, as visualized in the right part of the figure above.

Another important functionality that is implemented in hardware accelerator is the possibility of processing CUs with variable block sizes and different sizes of search areas. Thus, one instance of hardware accelerator can be used for all blocks and for all transcoding configurations. Although hardware accelerator in this particular case is used to implement function depicted in the right frame in Figure 10.1, it can also be used for regular inter prediction full integer search by setting the input parameter values to appropriate values (i.e., setting the starting index of original and reference CU to the same location).

10.3 Architecture

Hardware accelerator is designed to find the best inter predicted block in a defined search area and to return the motion vector as a result. Black-box model of hardware accelerator with defined inputs and outputs is given in Figure 10.2.

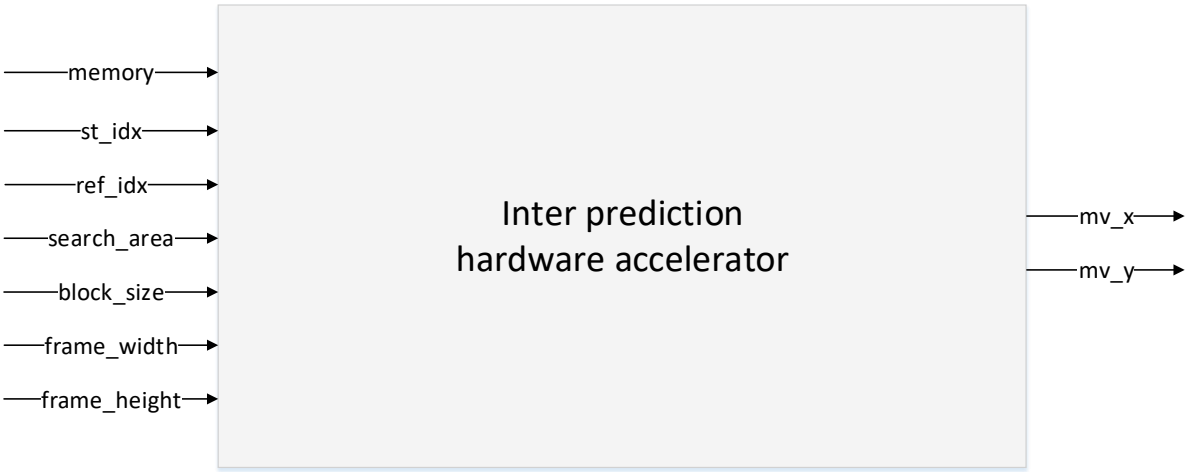

Figure 10.2: Model of a custom hardware accelerator for inter prediction

Inputs to the custom hardware accelerator are:

- memory - pointer to an address in memory where original and reference frame are located
- st_idx - starting index of the most-top-left pixel of current CU within the current frame
- ref_idx - starting index of the most-top-left pixel of CU predicted with weighted motion vector within the reference frame
- search_area - the size of a refinement search area (around the block with starting index ref_idx)
- block_size - size of CU block (e.g., for 32x32 CU block, block_size =32)
- frame_width, frame_height – dimensions of original and reference frame

Outputs from the accelerator are:

- mv_x, mv_y – motion vector (X, Y) for the best predicted block found in the defined search area

High-level block scheme of a custom hardware accelerator is given in Figure 10.3.

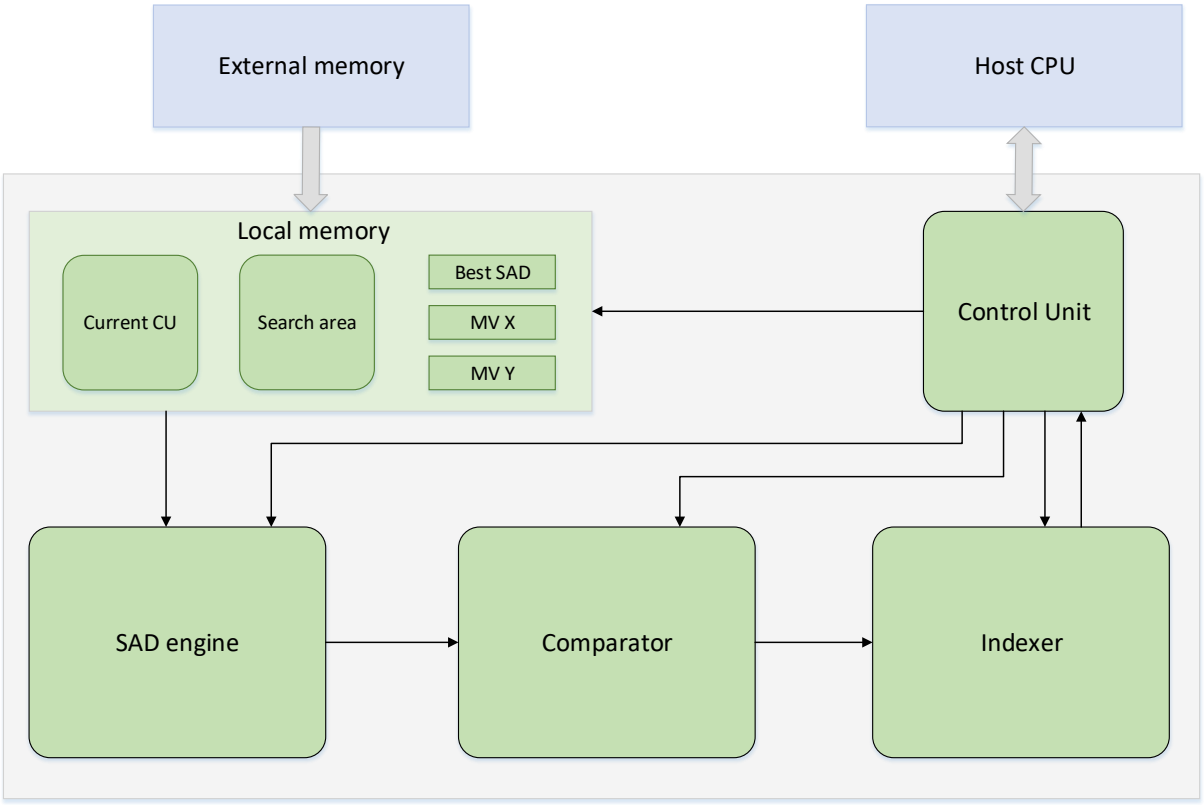

Figure 10.3: High-level hardware accelerator block scheme

Inter prediction hardware accelerator consists of several modules: Control unit, SAD engine, Comparator, and Indexer. The inter prediction operations starts by fetching data of the original CU block and search area from the external memory. Number of pixels fetched from memory for original CU block are equal to $block\_size * block\_size$, while number of pixels that need to be fetched for search area with defined search_area are $(2 * search\_area + bloc\_size) * (2 * search\_area + block\_size)$. Data fetched from external memory is stored in local memory. All computations within the hardware accelerator from this point on are based on a data stored in local memory, meaning that only one access to external memory is needed per one inter prediction operation.

After fetching the data needed for computation, the control unit initiates the process of finding the best predicted block from the defined search area. SAD engine calculates the sum of absolute differences between original CU and the predicted CU at the location *ref_idx*. Result of the SAD engine is forwarded to Comparator module that compares given value with the best SAD. If the calculated SAD value is better than best SAD stored as a register in local memory, then the value of best SAD is updated with current SAD and motion vector for that predicted block is calculated and stored in registers *MV X* and *MV Y*. After comparison, the Indexer

module refreshes index of next predicted block within search_area (*ref_idx*) making sure that the index does not point beyond frame boundaries. Control unit repeats the process until all possible blocks from search area are evaluated. Finally, at the end of the evaluation process, the best motion vector from registers *MV X* and *MV Y* are sent to the host processor.

10.4 Implementation and synthesis

Custom hardware accelerator was implemented and synthesized using Vivado High-Level Synthesis (HLS) tool. Vivado HLS allows functions written in C, C++, System-C and OpenCL kernels to be synthesized into RTL implementation and directly targeted into Xilinx programmable devices [64]. Each function from the high-level source code is translated into an RTL block in hardware. The top-level function interPredictionAcc that describes inter prediction accelerator is defined as:

```
void interPredictionAcc(volatile unsigned int *memory, unsigned int
st_idx, unsigned int ref_idx, unsigned int search_area, unsigned int
block_size, unsigned int frame_width, unsigned int frame_height, int
*mv_x, int *mv_y)
```

Besides the main accelerator function, other modules from Figure 10.3 are defined as functions, along with two functions that help accelerate computation by avoiding multiplication and division to facilitate hardware translation. The list of all functions used to design hardware accelerator with a brief description of functionality is listed below.

- *fetchBlockFromMem()* – fetching block from memory, depending on the offset address. It is used to fetch both, original CU block from the original frame and search area from the reference frame
- *sad()* – calculates the sum of absolute differences between original and reference block. Instead of fetching the predicted block from the search area in local memory, differences are calculated based on an index of the predicted block in the search area.
- *comparator()* – compares calculated SAD value with the best SAD value found so far in the process. If the value is better, registers Best SAD, MV X and MV Y are updated accordingly
- *indexer()* – refreshes indexes of next predicted block within the reference frame and search area. It checks if the value is valid and notifies control unit if all the blocks have been evaluated.
- *divideInt()* – a function that divides two integer numbers. Result of this function is quotient and reminder. Since there are no demands for floating point division in inter prediction algorithm, this function is implemented in order to avoid

regular division which could influence translation to hardware. The implemented algorithm for the division was based on

- *multiplyInt()* – a function that multiplies two integer numbers. Implemented for same reason as *divideInt()*.

Finding the best predicted block with the described functions of hardware accelerator follows the procedure given with the pseudocode below.

```
originalCU = fetchBlockFromMem(original_cu_address)
searchArea = fetchBlockFromMem(search_area_address)
isEnd = false;
predictedIndex = 0

while(!isEnd)
{
      currentSAD = sad(originalCU,predictedIndex)
      comparator(currentSAD)
      isEnd = indexer(predictedIndex)
}
```

Functions in code represent the design hierarchy that is translated into RTL blocks in hardware design, while the arguments of top-level function determine the hardware RTL interface ports. Giving that the top-level function is *interPredictionAcc(),* its arguments are synthesized as an input or output ports in exported design, as shown in Figure 10.4.

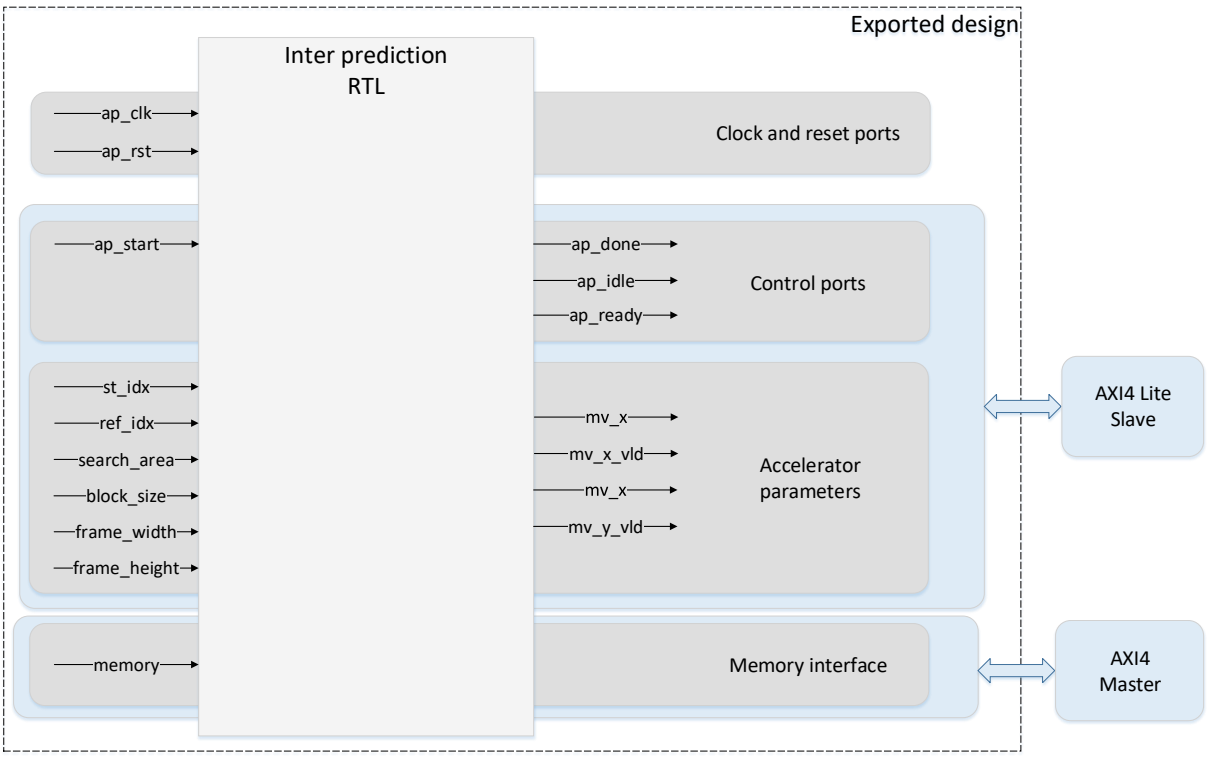

Figure 10.4: Exported hardware accelerator scheme

Along with accelerator parameters, synthesis adds additional input and output ports necessary for the successful operation of the accelerator. In addition to standard input signals, clock and reset, HLS also adds block level control signals: start, done, idle and ready. The *ap_start* signal controls the block execution and must be asserted to logic 1 for the design to begin operation. The *ap_ready* is an output signal that indicates when the accelerator is ready to receive new inputs, while *ap_done* indicates when the accelerator has completed all the operations in the current transaction, i.e., when the inter prediction is finished, and the best predicted block is found. When the accelerator is not doing any work, the *ap_idle* signal is set to logic 1. For each output parameter, in this case, motion vector X and Y, an additional signal that indicates if the valid data is set to output is added to design (*mv_x_vld* and *mv_y_vld*).

All the accelerator parameters and the control ports are grouped into a single common AXI4 lite slave interface that is used to pass parameters from CPU host to accelerator and vice versa. AXI-lite is light-weight, low-throughput memory mapped interface that has a small logic footprint and it is suitable for passing control and status signals to and from the accelerator [65]. A larger amount of data, however, cannot be efficiently transferred via AXI lite interface, since it allows only one data transfer per transaction. Thereby, memory port for accessing DDR memory to fetch data from original and reference frame is connected to the AXI4 Full interface that allows a burst of up to 256 data transfer cycles with just a single address phase.

The designed hardware accelerator was exported as an IP core using the Vivado HLS tool, and the final block is shown in Figure 10.5.

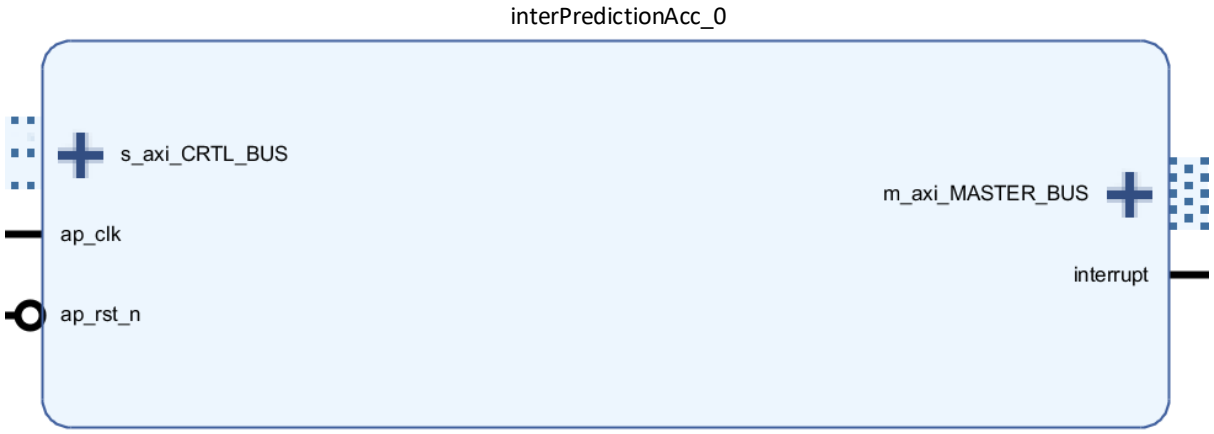

Figure 10.5: IP core of a custom hardware accelerator for inter prediction

Table 10.3 shows a report of how many resources are necessary for the implemented design (a number of BRAMs, DSPs, FFs, and LUTs). Percentages of overall hardware area

resources used are based on the target device for synthesis: Xilinx Kintex Ultrascale FPGA (*xcku115-flvb2104-2-e*).

Table 10.3: Hardware utilization of inter prediction custom hardware accelerator

	BRAM_18K	DSP48E	FF	LUT
Total used	6	24	3501	6065
Available	4320	5520	1326720	663360
Utilization (%)	**0.0013%**	**0.0043%**	**0.0026%**	**0.0091%**

10.5 Performance validation

Functional validation of custom hardware accelerator for inter prediction was conducted by comparing output results with the software version of the same algorithm. The most important aspect of the designed accelerator is performance improvement compared with software implementation. As demonstrated in section 10.1, expanding the search area for all inter prediction cases in the proposed transcoding algorithm causes violation of JiT requirements. Therefore, custom hardware accelerator has been introduced to cope with the increased computational complexity induced with the expansion of the search area. Two use cases were tested to validate and compare the performance of the accelerator:

- Inter prediction of CU blocks with different block sizes ($block\_size \in \{8,16,32\}$) and fixed search area ($search\_area = 5$)
- Inter prediction of CU blocks with a fixed block size ($block\_size = 32$) and variable search area ($search\_area \in \{1,2,3,4,5,6,7\}$)

Hardware accelerator for inter prediction was validated and verified on operating frequency of 250MHz. To benchmark the performance, 100 inter prediction tasks were offloaded to hardware accelerator and compared the same amount of tasks run in software implementation. Average time needed to finish the operation for one block (in milliseconds) is retrieved and depicted in the figures below. Figure 10.7 shows that the difference in execution time between software and hardware implementation significantly increases for larger blocks. When processing smaller block sizes, the impact of memory accesses to overall time is much higher than for larger blocks, which is not ideal for hardware accelerators that mainly focus on the enhancement of computational aspects of the algorithm. Therefore, hardware accelerator performs much better in situations where the ratio between computation and memory accesses leans toward computation.

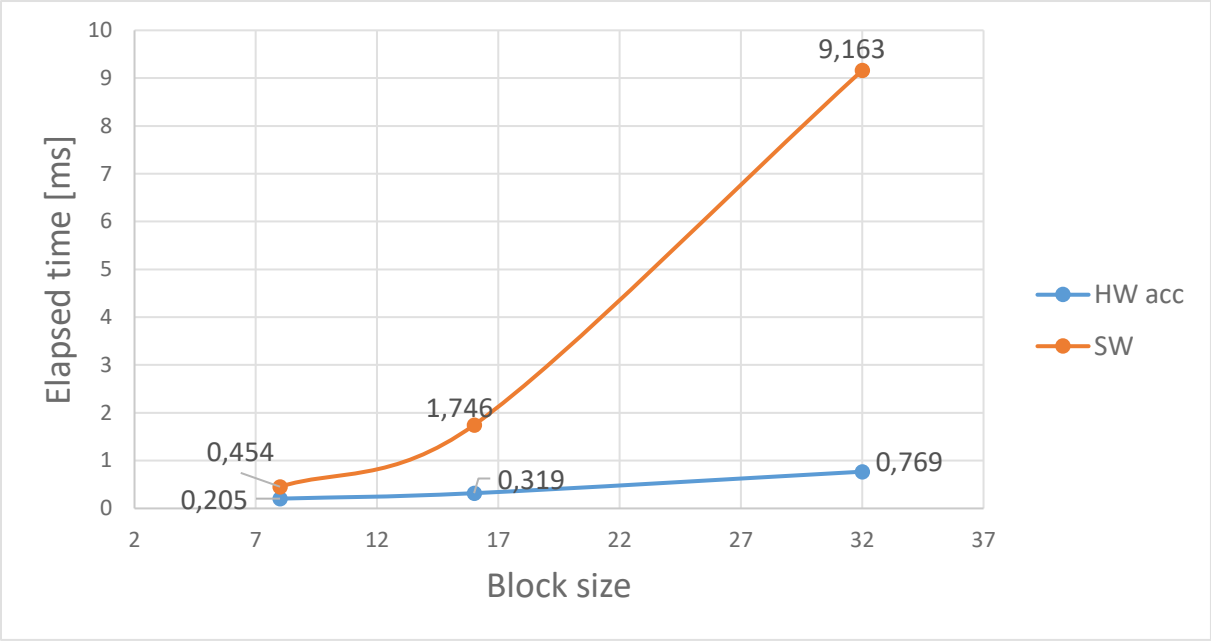

Figure 10.6: Comparison between hardware accelerator and software implementation of inter prediction (per CU block size for search area size 5)

Similar behavior can be observed when increasing the search area in the inter prediction (Figure 10.7).

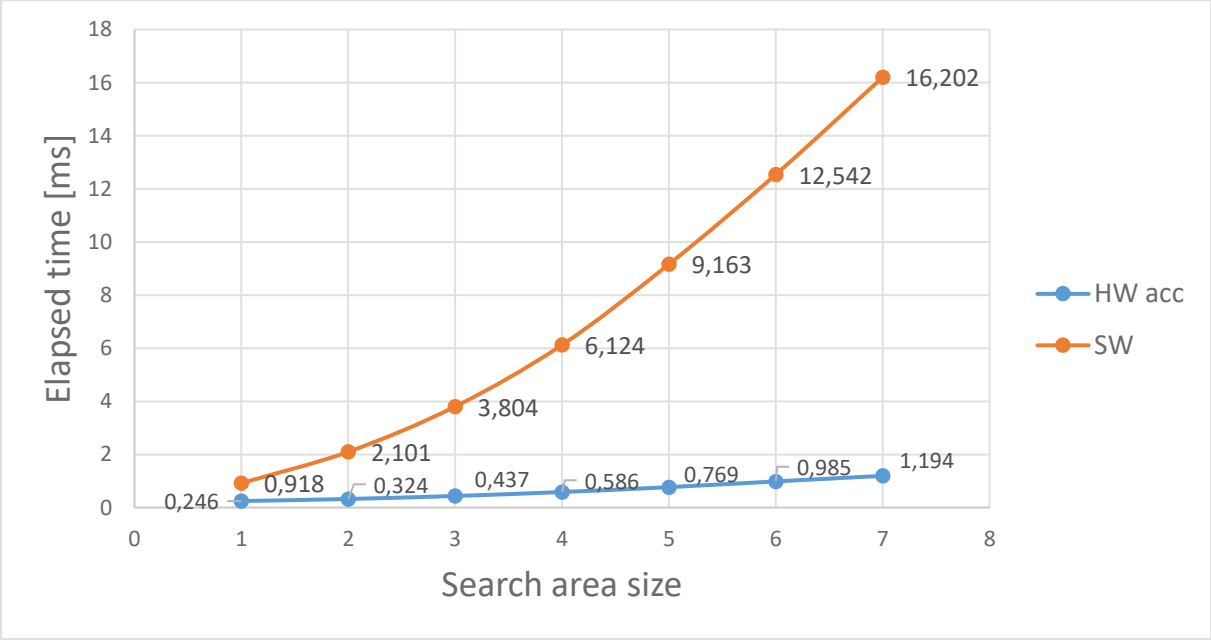

Figure 10.7: Comparison between hardware accelerator and software implementation of inter prediction (per search area size for block size 32)

Although the elapsed time of hardware accelerator gradually increases with larger block sizes, the software implementation increases at a much higher rate. Speedup of using hardware accelerator increases from 3.73x for search area of 1 pixel to 13.5x for search area of 7 pixels.

Now, the changes in inter prediction proposed in the Table 10.1 to increase the video quality of the transcoded bitstream can be observed form the hardware accelerator standpoint, since the software implementation of these adaptations was not viable due to increased transcoding time that broke the limits set by JiT transcoding. By comparing times needed for software execution of original inter prediction in the proposed algorithm that conforms to JiT requirements and times needed for hardware execution of the adapted inter prediction algorithm, some conclusions about the feasibility of the JiT transcoding with the new proposed inter prediction scheme can be made. For example, for most complex CUs within InterM category original prediction was made with the refinement of 2 pixels, but with the proposed custom hardware accelerator refinement of 4 pixels is approximately four times faster than a refinement of 2 pixels on same CU block in software. This fact should ensure JiT transcoding, taking into account that software 2-pixel refinement is proven to satisfy JiT requirements. However, in order to confirm these assumptions, integration of hardware accelerator to a Bolt65 transcoder with the implemented proposed JiT algorithm for data reusing has to be conducted and verified.

11 INTEGRATION AND FINAL RESULTS

This chapter describes the integration scheme of a custom hardware accelerator for inter prediction with video transcoding algorithm based on the utilization of decoded information from the original stream presented in this thesis. A platform for connecting hardware accelerator with the host side using high-performance PCI Express interconnect is presented in chapter 11.1. Different approaches to using the accelerator from the host side and its performance validation within the transcoding application is described in chapter 11.2. Final results of the integrated solution are shown in chapter 11.3.

11.1 Integration platform

Before starting the accelerator, the input data (i.e., data from the original CU and search area) has to be transferred from the host side. In heterogeneous systems, the time needed to transfer data from the host side to heterogeneous node and vice versa can often quash all the performance benefits gained by faster execution of hardware accelerator, becoming one of the main bottlenecks in the entire system. Therefore, for the connection between inter prediction hardware accelerator and the host that is running video transcoding, high-performance PCI Express Gen3x8 interconnect, that is able to achieve a speed of up to 8 Gbytes/sec was used to minimize the influence of data transfers on overall transcoding time. The block design of the platform that contains custom hardware accelerator is given in Figure 11.1.

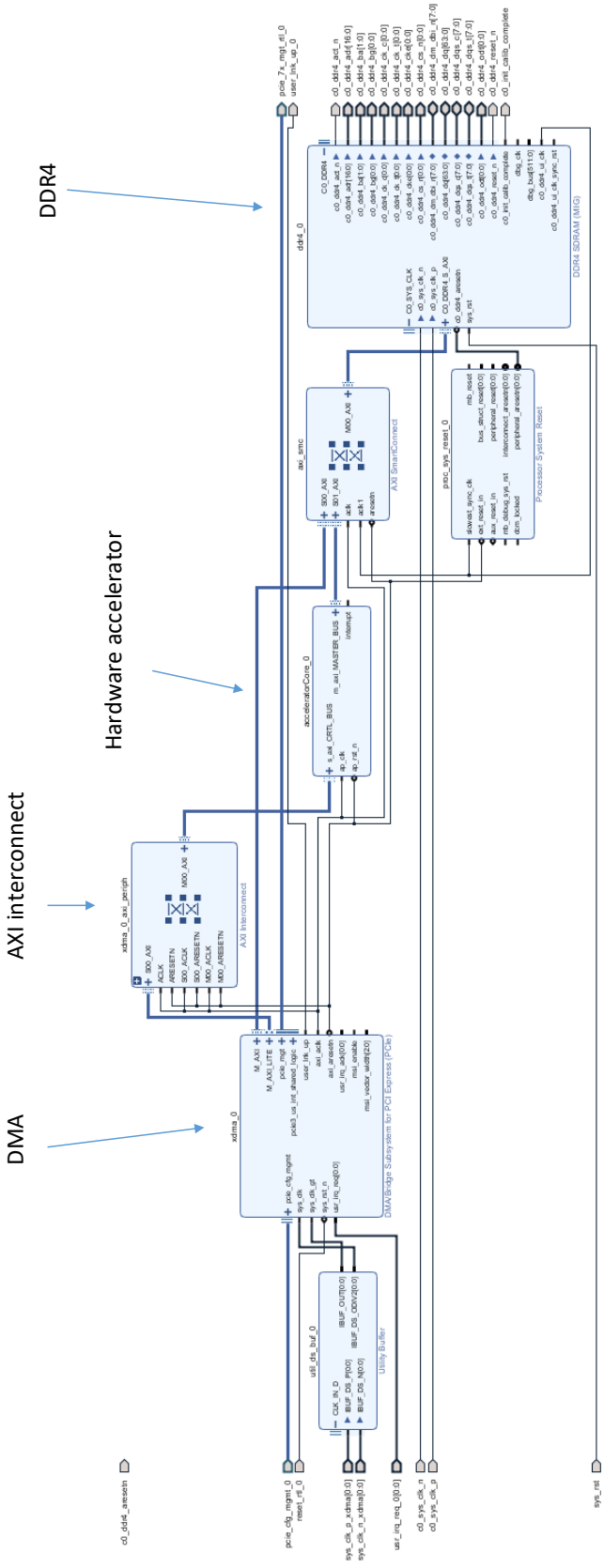

Figure 11.1: Block design of the integrated platform

The DMA system for PCIe masters read and write requests on the PCI Express and enables performing direct memory transfers from host to the FPGA platform and from the FPGA platform to the host [66]. AXI interconnect core is used to connect more AXI memory mapped master devices to one or more memory-mapped slave devices. In the design above, the procedure for running the accelerator follows the data path controlled by AXI interconnect as follows:

- Input data for inter prediction accelerator, pixels of current CU and search area from the reference frame, is written to DDR memory. Data is passed over the DMA and AXI interconnect, directly to DDR memory.
- The arguments for the accelerator (block size, size of the search area, etc.) are passed from the host side via AXI lite interface. Besides input parameters, addresses where the output motion vector will be located, as well as the address where the input data is located in DDR memory are also passed to the accelerator in the same transfer
- Control signal *ap_start* is set to logic 1, also over AXI lite interface, to initiate the start of the execution
- Hardware accelerator fetches the data from DDR, previously transferred from the host side, and starts the execution.
- After completion of the kernel, the calculated motion vector is written on defined addresses, and the control signal *ap_done* is set to 1
- After the host reads that *ap_done* is set to 1, the output from the accelerator is transferred to the host side, over the AXI-lite interface

Hardware utilization of the designed architecture is obtained with Vivado tool and is depicted in Figure 11.2.

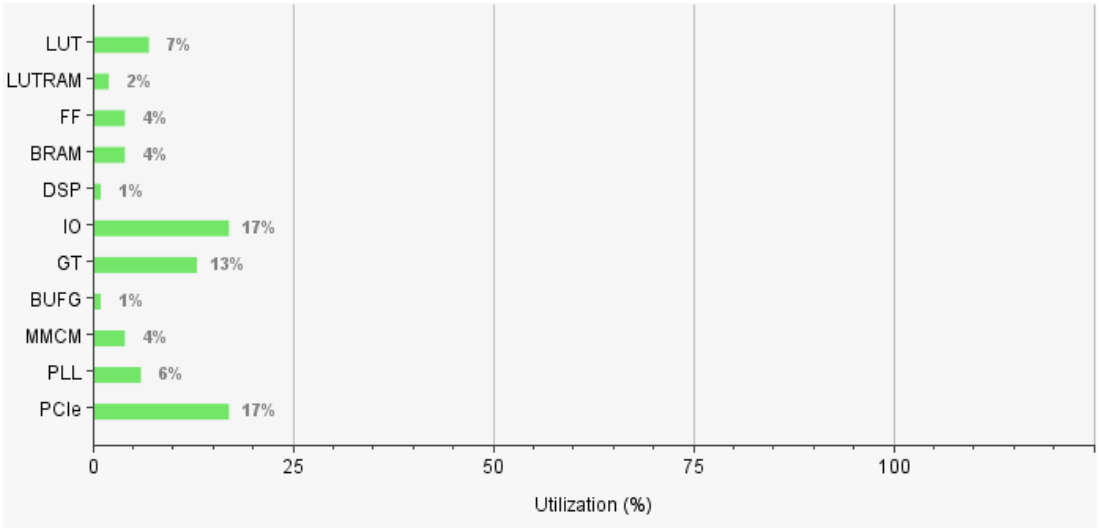

Figure 11.2: Hardware utilization of integrated design

11.2 Performance validation

There are several factors that influence the performance of an integrated solution, including data transfer rate, interrupt processing and the smart utilization of hardware accelerator. An important factor that affects data throughput is interrupt processing. Once the accelerator is finished with finding the best inter prediction candidate, the accelerator sends an interrupt to the host side and waits for the host side to process the status. However, this wait time is not predictable, which is why another approach is considered in the design of the integrated system. Instead of waiting for the interrupt from the hardware accelerator, the host uses poll mode, which gives the best data transfer rates [67]. In poll mode, the host needs to monitor the completion status of hardware accelerator (*ap_done* signal) to check if the operation is executed. However, while waiting for the completion, additional processing can be done on the host side in parallel, which will be presented later in this chapter.

Another important aspect of the overall performance is the utilization of hardware accelerator. Two working modes were tested and compared in the scope of this integration:

- Standalone mode – Hardware accelerator receives the data for processing individual tasks. All the parameters, along with original CU and search area data are sent every time accelerator is started.
- Iterative mode – Hardware accelerator receives only the input parameters for processing individual task. Original CU and search area data are fetched from DDR memory, where the two whole frames (original and reference) have been previously transferred from the host side.

Notice that for the standalone mode there will be multiple smaller data transfers for processing one frame, while in iterative mode, only one, significantly larger, data transfer is needed per frame. To compare the two working modes a test that encodes one Full HD frame in both ways is conducted. The frame was split only to 32x32 blocks, with the refinement search area of 5 pixels (i.e., size od search area is 42x42). In Full HD frame (1920x1080) there are 2040 32x32 CU blocks, which means that the number and size of data transfers per mode will be:

$$Standalone\ mode: 2040 * \big((32 * 32) + (42 * 42)\big) = 2040\ transfers\ x\ 2788\ bytes$$

$$Iterative\ mode: 1 * (2 * 1920 * 1080) = 1\ transfer\ x\ 4147200\ bytes$$

The ratio between time spent on memory transfers and processing in two modes is given with Figure 11.3.

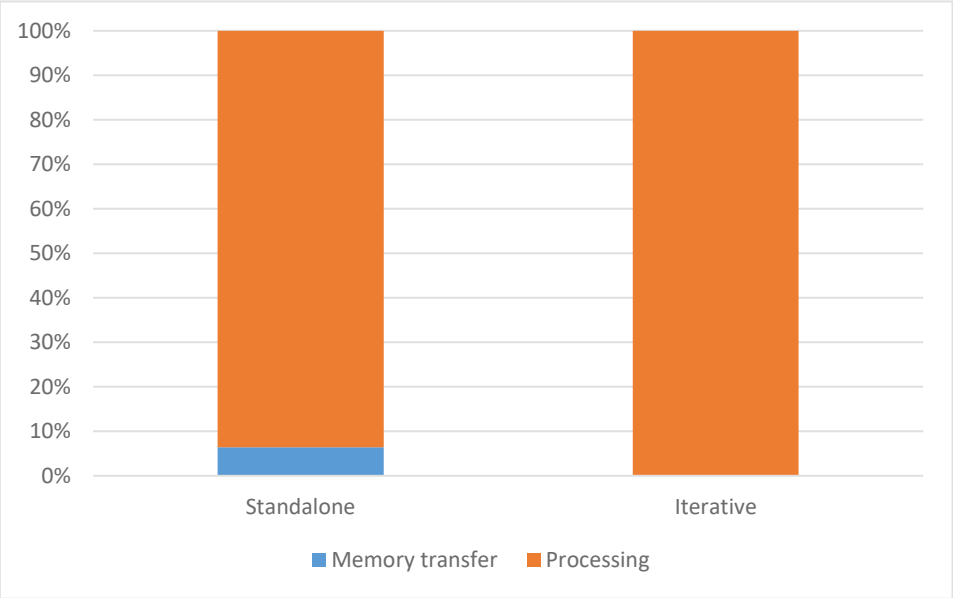

Figure 11.3: Ratio between memory transfer and processing for two working modes

In a standalone mode, where there are multiple transfers between the host side and the hardware platform, data transfer takes about 6.4% of the time needed to perform inter prediction for the entire frame. In an iterative working mode, this time significantly decreases and is negligible compared to the time necessary to process the inter prediction operation, taking 0.005% of the overall time. This behavior can be explained by the nature of data transfers via PCIe interconnect, where a much higher transfer speed is achieved by transferring larger amounts of data, as shown in Figure 11.4, where to use cases are observed: transfer from host side to FPGA platform and vice versa.

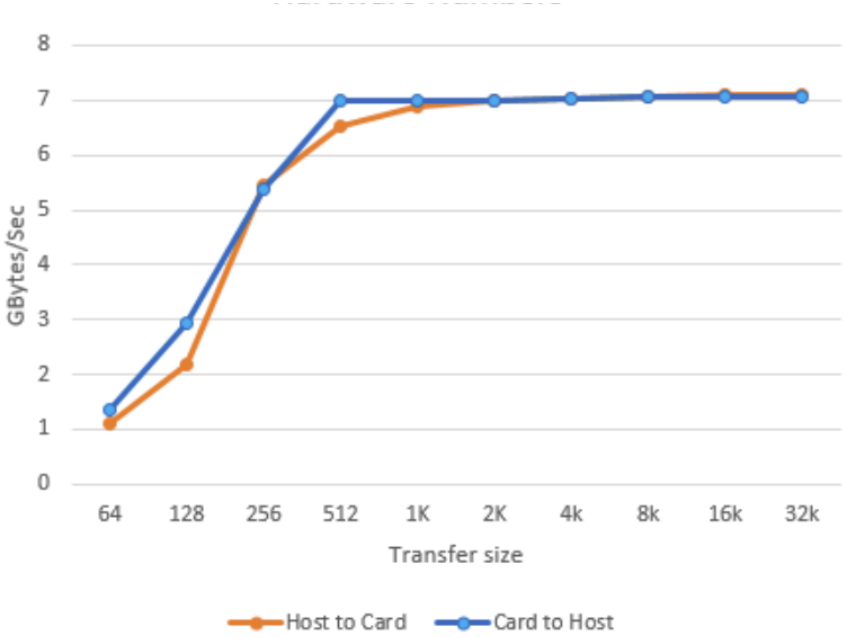

Figure 11.4: PCI Express transfer speeds in correlation with transfer size, source [67]

With the accomplished speed of the data transfers between the host side and the FPGA platform, overhead of passing the data to and from the accelerator has been reduced to a minimum and does not present a bottleneck in the integrated system.

After the host initiates the start of a hardware accelerator, the transcoding process can be further optimized by running other tasks on the host side in parallel with the inter prediction that is being executed on the hardware platform. Giving that the proposed data reusing algorithm evaluates inter and intra prediction candidates for most of the CUs, transcoding process can be parallelized so that inter prediction is conducted on customized hardware accelerator, while the intra prediction is performed on the host side. Taking into account the adopted changes, the final scheme of the proposed data reusing algorithm on the heterogeneous platform is presented in Figure 11.5. Modules that imply using a hardware platform, transferring decoded and reference frame to DDR memory and inter prediction, are denoted with the green colors.

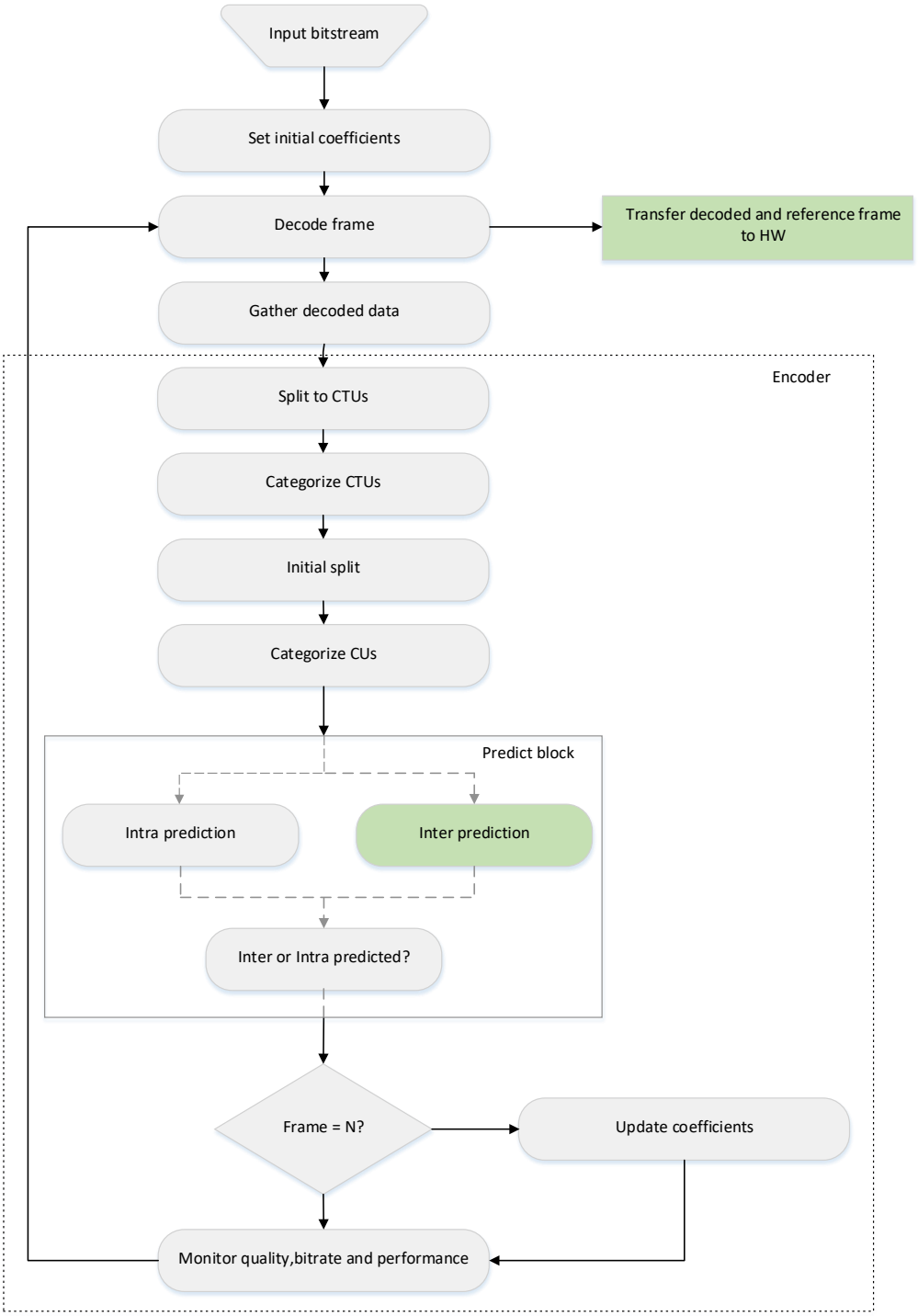

Figure 11.5: Final scheme of the algorithm on the integrated heterogeneous platform

11.3 Final results

Transcoder with implemented novel data reuse algorithm and the custom hardware accelerator for inter prediction has been deployed to the integrated heterogeneous platform and compared with the results of the CPU-only implementation of the same algorithm and with Just-

in-Time configuration of Bolt65 transcoder that does not reuse any data from the original bitstream (Bilt65 JiT). Results presented in Figure 11.5 show average gains in PSNR and bitrate for all transcoding scenarios, compared to baseline Bolt65 JiT straight-forward transcoder.

Table 11.1: Comparison between CPU-only implementation and implementation on a heterogeneous integrated platform

Video name	CPU-only implementation			Heterogeneous implementation			Speedup [t_{SW}/t_{HW}]
	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]	PSNR [Δ dB]	Bitrate [Δ kbps]	Bitrate [%]	
Shields	0.750	-8891.0	-58.30%	1.100	-12881.0	-66.12%	1.03x
ParkRun	0.493	-8566.5	-34.65%	0.695	-11935.0	-41.51%	1.09x
KristenAndSara	0.749	-1118.5	-32.36%	0.853	-1328.5	-39.67%	1.08x
Johnny	0.524	-682.0	-25.34%	0.669	-1084.5	-39.25%	1.07x
FourPeople	0.674	-1113.5	-33.84%	0.864	-1280.0	-39.14%	1.02x
BasketballDrive	1.038	-3400.7	-20.13%	1.222	-3991.0	-26.33%	1.06x
Calendar	1.491	-1273.7	-21.74%	1.615	-1315.3	-27.47%	1.06x
Cactus	0.539	-3518.7	-25.17%	0.706	-4204.7	-34.08%	1.01x
BQTerrace	0.670	-6934.3	-29.34%	0.859	-8914.3	-43.60%	1.03x
RushHour	1.024	-285.3	-8.90%	1.174	-274.7	-8.31%	1.02x
Riverbed	1.229	-2456.0	-20.41%	1.430	-2389.3	-20.34%	1.07x
PedestrianArea	1.371	-1265.3	-35.51%	1.466	-1349.3	-39.75%	1.10x
BlueSky	0.503	-3392.3	-23.47%	0.622	-5874.0	-44.93%	1.11x
Traffic	0.473	-3941.3	-45.02%	0.588	-5012.3	-56.16%	1.09x
DuckTakeOff	0.208	-1829.8	-3.84%	0.286	-3321.4	-6.02%	1.08x
Bosphorus	0.492	-4432.2	-30.61%	0.616	-7458.0	-47.29%	1.06x
Beauty	1.173	-13733.6	-16.35%	1.295	-13655.2	-16.03%	1.04x
Average	**0.788**	**-3931.5**	**-27.35%**	**0.945**	**-5074.6**	**-35.06%**	**1.06x**

Results show that the proposed algorithm ran on heterogeneous platform outperforms CPU-only implementation of the same algorithm in both, average PSNR and bitrate. Compared with Bolt65 JiT transcoder, CPU-only solution has an average increase in PSNR of 0.788 dB, while for implementation on the heterogeneous platform this gain rises to 0.945 dB. Regarding the bitrate, reduction of 27.35% obtained with SW implementation is improved to 35.06%. Transcoding on a heterogeneous platform also increases performance time by an average of 6%. However, since the proposed algorithm adapts the coefficients that define categorization boundaries depending on the processing time, the number and configuration of inter prediction tasks are not the same, so the observed speedup is not based solely by increasing the speed of inter prediction operation. Notice that for the same reason, PSNR gains are higher compared with CPU-only implementation than in previously observed scenario (section 10.1).

After presenting and explaining all the concepts of the proposed algorithm for JiT transcoding based on the utilization of coding information from the input bitstream and the heterogeneous architecture with a custom hardware accelerator for inter prediction, a final

comparison between two baseline transcoders and the two versions of the novel algorithm presented in this thesis can be made.

Figure 11.6 shows the average PSNR gains for transcoding video sequence from the original resolution to all possible resolutions defined in the test methodology in chapter 5. Lowest quantization parameter defined in Common Test Conditions [60] of 22 was observed since it represents the most complex transcoding scenario with the highest demands on computational resources.

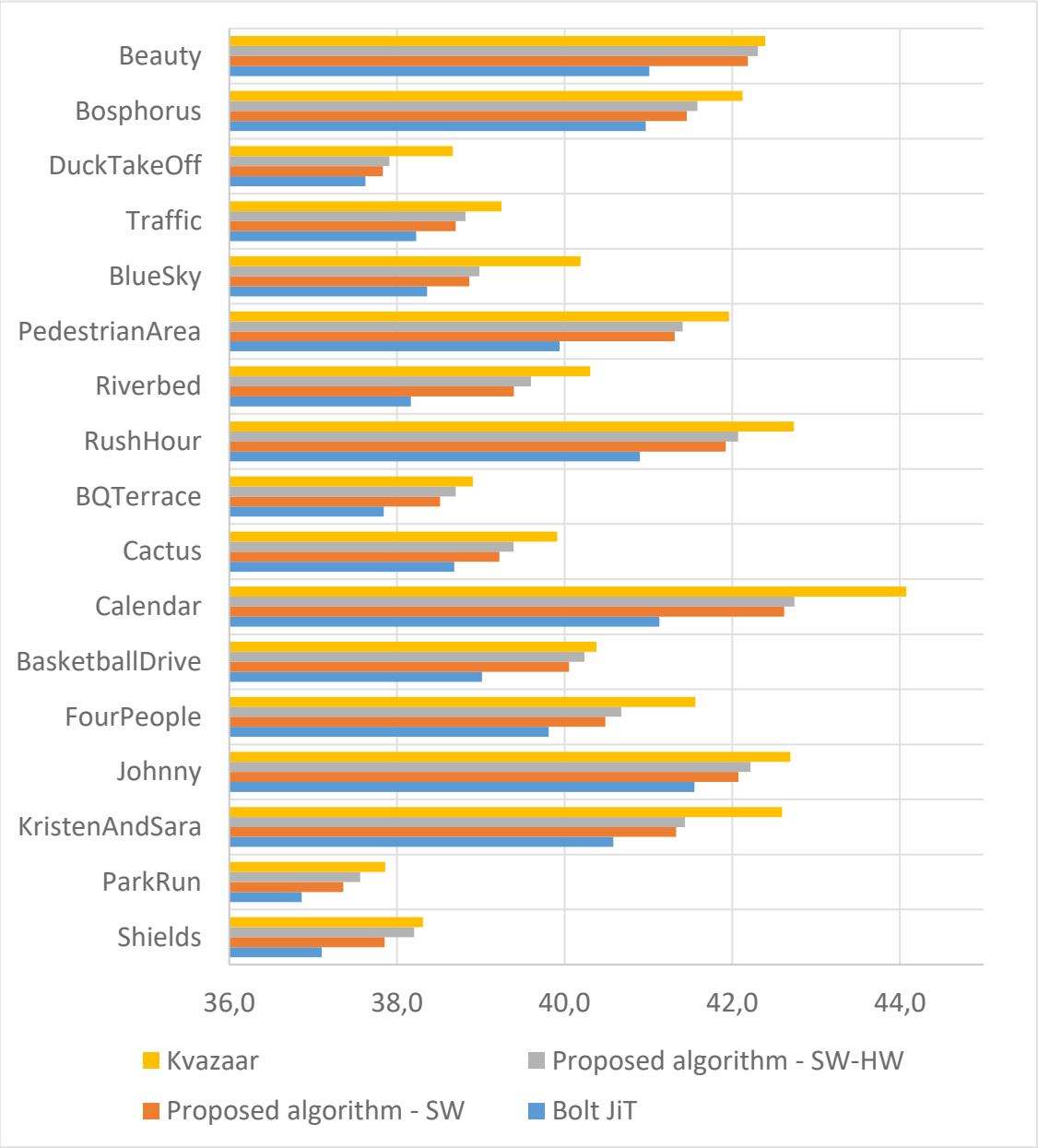

Figure 11.6: Final PSNR comparison

Proposed algorithm deployed to a system that contains only CPU achieves a significant increase in video quality compared to Just-in-Time transcoder that re-encodes video sequence from scratch, without reusing information extracted from the input bitstream. By introducing custom hardware accelerator for inter prediction on the FPGA platform and adapting the algorithm to heterogeneous architecture that consists of FPGA and CPU, even larger gains in video quality can be observed. However, average PSNR is still lower than for transcoder that uses open-source Kvaazaar encoder that does not comply with JiT transcoding.

A similar analysis is made for bitrate reduction, where the percentage of the reduction compared with Bolt JiT transcoder is presented in Figure 11.7.

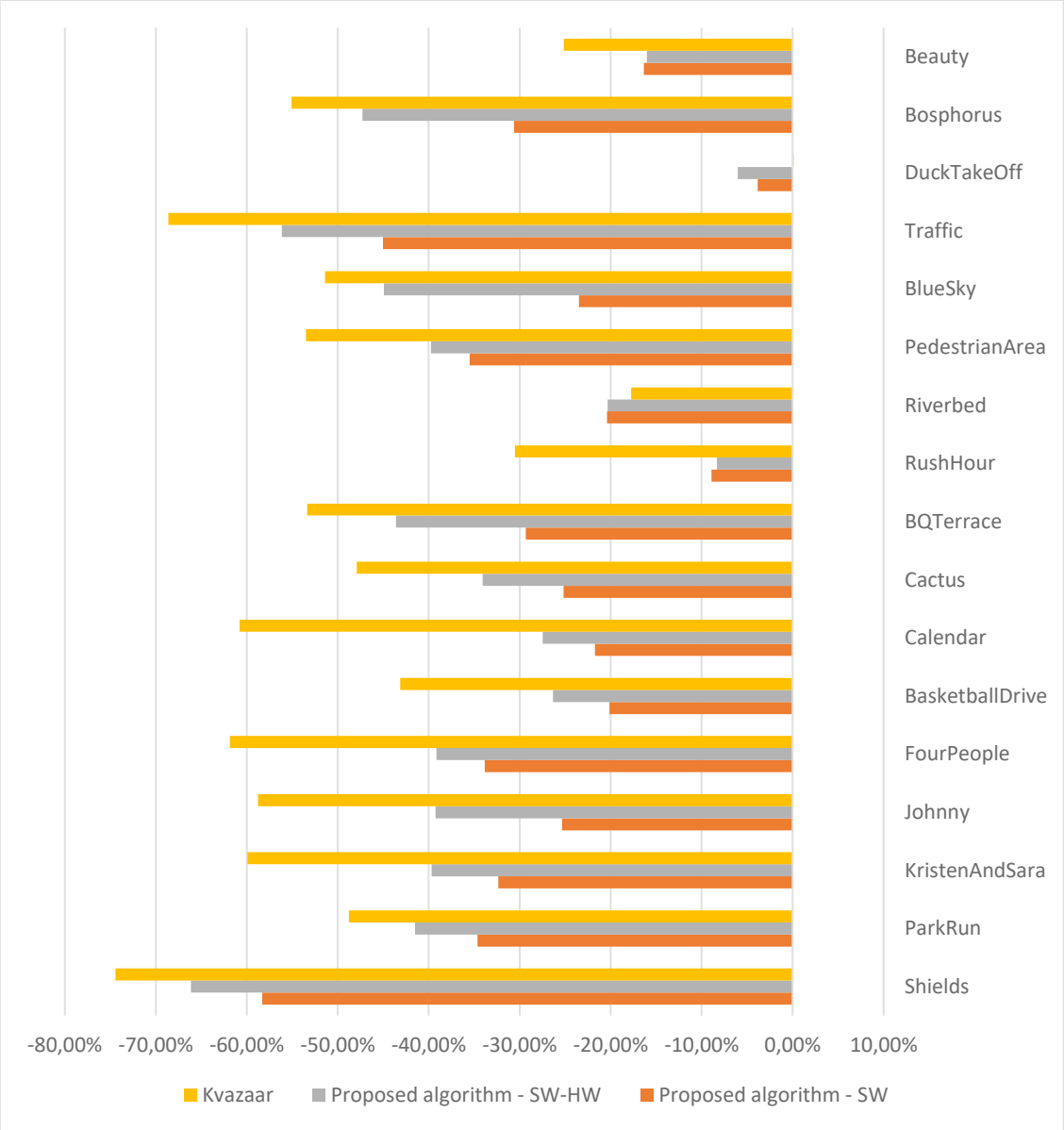

Figure 11.7: Final bitrate comparison

Constant improvement in bitrate reduction compared with straight-forward Bolt65 transcoder can be seen in the figure above. Implementation of the transcoder on the heterogeneous platform gives better bitrate in most of the video sequences compared with CPU-only implementation. For some video sequences, achieved bitrate reduction is even better than for Kvazaar transcoder.

Better video quality and bitrate in Kvazaar transcoder come at the cost of increased processing time, which causes violation of Just-in-Time transcoding. Figure 11.8 depicts differences in processing times between the observed transcoders. The dotted red line in the graph illustrates the limit for achieving JiT transcoding (100% of t_{JiT}).

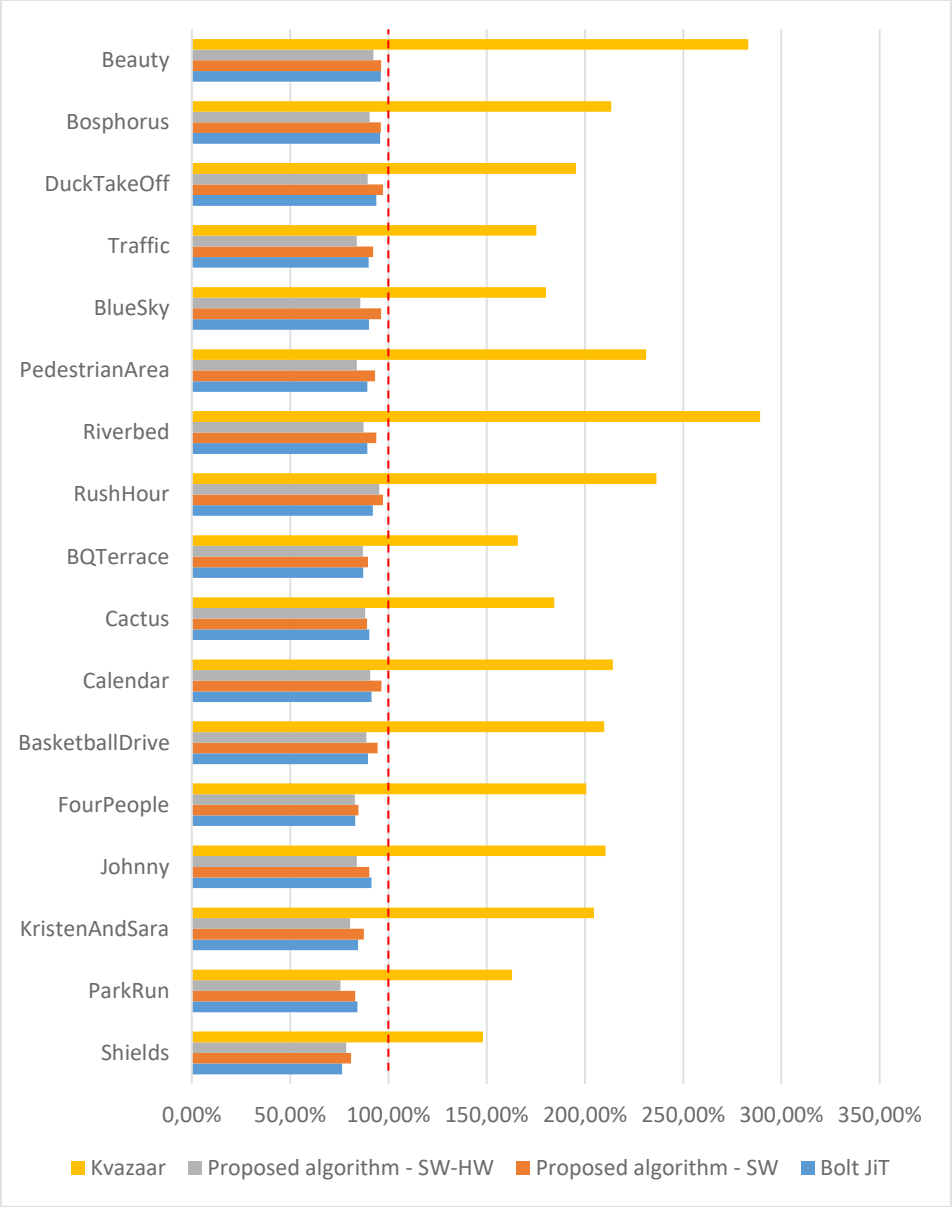

Figure 11.8: Final processing time comparison

The regulation process of the proposed algorithm (i.e., monitoring and adaptation of the categorization coefficients) ensures Just-in-Time transcoding for all scenarios and all video sequences, which is manifested in the graph above. Kvazaar transcoder, contrarily, breaks the JiT limits by a minimum of 50% (for *Shield* video sequence), while the processing time for some sequences is almost 3 times higher than needed for JiT transcoding.

12 CONCLUSION

Algorithms for efficient utilization of coding information extracted from the input video stream in the process of video transcoding were investigated in the scope of research conducted for this thesis. Since the software solutions alone cannot most efficiently provide Just-in-Time video transcoding, hardware-based accelerators of key compute-intensive kernels were also examined in order to achieve the best trade-off between coding efficiency and video quality while fulfilling Just-in-Time constraints. Performance-efficient integration of algorithms and hardware-based accelerator kernels into one high-performance system was the final contribution of this thesis.

The algorithm presented in this thesis tries to estimate the computational complexity needed for re-encoding each coding block. The estimation is based on the concept of categorization, where each coding unit is categorized in regard to three different types of information extracted from the decoded frame: the size of decoded coding units, number of coding units mapped from the decoded frame and prediction modes of mapped coding units. Depending on the output of the categorization process, different algorithms are used to encode particular coding unit. More computing resources will be assigned to processing more complex CUs that usually contain more detailed information within the video frame and that have a higher impact on the quality of final transcoded bitstream. Boundaries for categorization are adapted during the transcoding process to ensure predictability and guarantee Just-in-Time execution. The developed algorithm, ported on CPU-only architecture achieves higher PSNR and reduced bitrate for all transcoding scenarios compared to Just-in-Time transcoder that does not reuse data from the input bitstream.

Furthermore, a custom hardware accelerator for one of the most compute-intensive kernels in the process of video transcoding, inter prediction, was designed and implemented. By utilizing hardware accelerator, inter prediction used in the software version of the algorithm was enhanced by expanding the search area in the motion estimation process, without compromising Just-in-Time execution. With the expanded search area, a larger set of inter prediction candidates could be evaluated, increasing possibility of fining the best inter predicted block.

Integration of custom hardware accelerator for inter prediction with the proposed algorithm was conducted on a system that consists of CPU on the host side and the FPGA

hardware platform. Communication between the host side and the hardware accelerator was implemented using high-throughput PCI Express interconnect to minimize the influence of time needed for memory transfers on the overall processing time and to avoid possible bottlenecks. The integrated solution increases video quality by 0.945 dB and reduces bitrate by 35.06% on average compared with JiT transcoder. Compared to transcoder without timing requirements, average losses of 0.592 in PSNR and 21.74% in bitrate were achieved, but with significant speedups of up to 4 times.

Finally, contributions of the research conducted for this thesis are:

- Perfomance-optimized algorithms and hardware-based accelerator kernels on heterogeneous high performance computing architectures for just-in-time video transcoding based on utilisation of coding information from input video stream
- Performance-efficient integration of system architectures composed of implemented algorithms and hardware-based accelerator kernels on heterogeneous high performance computing architectures for just-in-time video transcoding

Future work in this research area will include several aspects. A larger set of encoding tools introduced in HEVC will be analysed to try to achieve even higher gains in video quality compared with straight-forward transcoders. Inclusion of tools, such as symmetric and asymmetric prediction units, deeper transform trees, in-loop filters or interpolation, can help to enhance transcoded bitstream. However, this compromises Just-in-Time execution, so detailed analysis must be performed before adding any of the proposed tools in the existing solution.

More compute-intensive kernels will be investigated, and their influence on the overall process of video transcoding will be analysed. Custom hardware accelerators for other suitable kernels will be designed and integrated with the integrated solution. Possible kernel candidates for hardware acceleration include interpolation, transformation, and quantization. Different types of processing units, besides custom accelerator-based cores on FPGA platform, will also be investigated, such as GPU core, RISC-V, and GPU-like core.

With the increased number of kernels and processing cores, the advanced resource manager will have to be developed to control and monitor the execution of the entire process. Some other aspects of the system, besides video quality, coding efficiency and performance, such as power consumption will also have to be taken into consideration.

REFERENCES

[1] Cisco Visual Networking Index: Forecast and Methodology, 2017–2022, available at https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html, (3th May 2019.)

[2] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022, available at https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html, (3th May 2019.)

[3] Vetro A., Christopoulos C., Sun H., "Video transcoding architectures and techniques: an overview," in IEEE Signal Processing Magazine, vol. 20, no. 2, pp. 18-29, March 2003.

[4] Schwarz H., Marpe D., Wiegand T., "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard", in IEEE Transactions on Circuits and Systems for Video Technology, vol. 17, no. 9, pp. 1103-1120, September 2007.

[5] Boyce J. M., Ye Y., Chen J., Ramasubramonian A. K., "Overview of SHVC: Scalable Extensions of the High Efficiency Video Coding Standard," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, no. 1, pp. 20-34, January 2016.

[6] Dragić L., Hofman D., Kovač M., Žagar M., Knezović J., "Power consumption and bandwidth savings with video transcoding to mobile device-specific spatial resolution," 2014 9th International Symposium on Communication Systems, Networks & Digital Sign (CSNDSP), pp. 348-352, Manchester 2014.

[7] Gao G., Wen Y., Zhang W., Hu H., "Cost-efficient and QoS-aware content management in media cloud: Implementation and evaluation," 2015 IEEE International Conference on Communications (ICC), pp. 6880-6886, London 2015.

[8] Kim J.W., Kwon G.-R-, Kim N.-H., Morales A., Ko S.-J., "Efficient video transcoding technique for QoS-based home gateway service," in IEEE Transactions on Consumer Electronics, vol. 52, no. 1, pp. 129-137, February 2006.

[9] Alsrehin, N., Clyde S., "QoS-Aware Video Transcoding Service Selection Process," in Journal of Media & Mass Communication, vol. 1, no. 2, pp. 61-68, December 2015.

[10] Sullivan G. J., Ohm J. R., Han W. J., Wiegand T., "Overview of the High Efficiency Video Coding (HEVC) Standard," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[11] Bossen F., Bross B., Suhring K., Flynn D., "HEVC Complexity and Implementation Analysis," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1685-1696, December 2012.

[12] Flich, J., Agosta G., Ampletzer P., Atienza D., Brandolese C., Cilardo A., Fornaciari W., Hoornenborg Y., Kovač M., Piljić I., Duspara A., Dragić L., Knezović J., Sruk V., Hofman D., Maitre B., Massari G., Mlinarić H., Papastefanakis E., Roudet F., Tornero R., Zoni D., "MANGO: Exploring Manycore Architectures for Next-GeneratiOn HPC Systems", 2017 Euromicro Conference on Digital System Design (DSD), pp. 478-485., Vienna, 2017.

[13] Sze V., Budagavi M., Sullivan G.J., "High Efficiency Video Coding (HEVC)," Springer, 2014.

[14] Kim I., Min J., Lee T., Han W., Park J., "Block Partitioning Structure in the HEVC Standard," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1697-1706, December 2012.

[15] Lainema J., Bossen F., Han W., Min J., Ugur K., "Intra Coding of the HEVC Standard," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1792-1801, December 2012.

[16] Ugur K., Alshin A., Alshina E., Bossen F., Han W., Park J., Lainema J., "Motion Compensated Prediction and Interpolation Filter Design in H.265/HEVC," in IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 946-956, December 2013.

[17] Ismail Y., El-Medany W., Al-Junaid H., Abdelgawad A., "High performance architecture for real-time HDTV broadcasting," J. Real-Time Image Process., vol. 11, no. 4, pp. 633–644, 2016.

[18] Amirpour H., Mousavinia A., Shamsi N., "Predictive Three Step Search (PTSS) algorithm for motion estimation," 2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP), pp. 48-52, Zanjan 2013.

[19] Zhu S., Ma K.-K., "A new diamond search algorithm for fast block-matching motion estimation," in IEEE Transactions on Image Processing, vol. 9, no. 2, pp. 287-290, February 2000.

[20] Choi C., Jeong J., "Successive Elimination Algorithm for Constrained One-bit Transform Based Motion Estimation Using the Bonferroni Inequality," in IEEE Signal Processing Letters, vol. 21, no. 10, pp. 1260-1264, October 2014.

[21] Goel S., Ismail Y., Bayoumi M. A., "Adaptive search window size algorithm for fast motion estimation in H.264/AVC standard," in 48th Midwest Symposium on Circuits and Systems, vol 2., pp. 1557-1560, Covington, KY 2005.

[22] Norkin A., Andersson K., Fuldseth A., Bjøntegaard G., "HEVC deblocking filtering and decisions," in Proc. SPIE. 8499, Applications of Digital Image Processing XXXV, no. 849912, October 2012.

[23] Fu C., Chen C., Huang Y., Lei S., "Sample adaptive offset for HEVC," in 2011 IEEE 13th International Workshop on Multimedia Signal Processing, pp. 1-5., Hangzhou 2011.

[24] Sze V., Budagavi M., "High Throughput CABAC Entropy Coding in HEVC," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1778-1791, December 2012

[25] Zhu X., Yu L., "Binarization and context model selection of CABAC based on the distribution of syntax element," in 2012 Picture Coding Symposium, pp. 77-80., Krakow 2012.

[26] Misra K., Segall A., Horowitz M., Xu S., Fuldseth A., Zhou M., "An overview of tiles in HEVC," in IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 969-977, December 2013.

[27] Koziri M.G., Papadopoulos P., Tziritas N., Dadaliaris A.N., Loukopoulos T., Khan S.U., Xu C.-Z., "Adaptive Tile Parallelization for Fast Video Encoding in HEVC," in IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Chengdu, December 2016.

[28] Storch I., Palomino D., Zatt B., Agostini L., "Speedup-aware history-based tiling algorithm for the HEVC standard, " in 2016 IEEE International Conference on Image Processing (ICIP), pp. 824-828, Phoenix AZ, 2016.

[29] Dragić L., Piljić I., Kovač M., “Dynamic load balancing algorithm based on HEVC tiles for Just-in-Time video encoding for heterogeneous architectures,” in Automatika Journal for Control, Measurement, Electronics, Computing and Communications, vol. 60, no. 2, pp. 239-244, 2019.

[30] Xin J., Lin C-W., Sun M-T., “Digital Video Transcoding,” in Proceedings of the IEEE, vol. 93, no. 1, pp. 84-97, January 2005

[31] Ahmad I., Wei X., Sun Y., Zhang Y., “Video transcoding: An Overview of Various Techniques and Research Issues,” in IEEE Transactions on Multimedia, vol. 7, no. 5, pp. 793–804, October 2005.

[32] Eleftheriadis A., Anastassiou D, "Constrained and general dynamic rate shaping of compressed digital video," in International Conference on Image Processing, vol 3., pp. 396-399, Washington 1995.

[33] Sun H., Kwok W., Zdepski J.W., "Architectures for MPEG compressed bitstream scaling," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, no. 2, pp. 191-199, April 1996.

[34] Assuncao P. A. A., Ghanbari M., "A frequency-domain video transcoder for dynamic bit-rate reduction of MPEG-2 bit streams," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 8, no. 8, pp. 953-967, December 1998.

[35] Yuan L., Wu F., Chen Q., Li S., Gao W., "The fast close-loop video transcoder with limited drifting error", in 2004 IEEE International Symposium on Circuits and Systems, pp.769 -772 , Vancouver 2004.

[36] Yin P., Vetro A., Liu B., Sun H., "Drift compensation for reduced spatial resolution transcoding," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 12, no. 11, pp. 1009-1020, November 2002.

[37] Ahmad I., Wei X., Sun Y., Zhang Y.-Q., "Video transcoding: an overview of various techniques and research issues," in IEEE Transactions on Multimedia, vol. 7, no. 5, pp. 793-804, October 2005

[38] Youn J., Sun M.-T., "Adaptive motion vector refinement for high performance transcoding," in Proceedings 1998 International Conference on Image Processing, vol 3, pp 596-600, Chicago 1998.

[39] Shanableh T., Ghanbari M., "Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats," in IEEE Transactions on Multimedia, vol. 2, no. 2, pp. 101-110, June 2000.

[40] Mohan R., Smith J. R., Li C.-S., "Adapting multimedia Internet content for universal access," in IEEE Transactions on Multimedia, vol. 1, no. 1, pp. 104-114, March 1999.

[41] Carey W. K., Chuang D. B., Hemami S. S., "Regularity-preserving image interpolation," in IEEE Transactions on Image Processing, vol. 8, no. 9, pp. 1293-1297, September 1999.

[42] Tan Y.-P., Sun H., Liang Y.Q., "On the methods and applications of arbitrarily downsizing video transcoding," in IEEE International Conference on Multimedia and Expo, vol. 1, pp. 609-612, Lausanne 2002.

[43] Xu D., Nasiopoulos P., "Logo insertion transcoding for H.264/AVC compressed video", in IEEE International Conference on Image Processing (ICIP), pp. 3693-3696, Cairo 2009.

[44] Zhang J., Ho A. T. S., Qiu G., Marziliano P., "Robust Video Watermarking of H.264/AVC," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 54, no. 2, pp. 205-209, February 2007.

[45] Reyes de los G., Reibman A. R., Chang S.-F., Chuang J. C.-I., “Error-resilient transcoding for video over wireless channels,” in IEEE J. Sel. Areas Commun., vol. 18, no. 6, pp. 1063–1074, June 2000.

[46] Dogan S., Cellatoglu A., Uyguroglu M., Sadka A. H., Kondoz A. M., "Error-resilient video transcoding for robust internetwork communications using GPRS," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 12, no. 6, pp. 453-464, June 2002.

[47] Van Wallendael G., De Cock J., Van de Walle R., "Fast transcoding for video delivery by means of a control stream," in 19th IEEE International Conference on Image Processing, pp. 733-736, Orlando 2012.

[48] Rusert T., Andersson K., Yu R., Nordgren H., "Guided just-in-time transcoding for cloud-based video platforms," in IEEE International Conference on Image Processing (ICIP), pp. 1489-1493, Phoenix 2016.

[49] Pham Van L., De Praeter J., Van Wallendael G., Van Leuven S., De Cock J., Van de Walle R., "Efficient Bit Rate Transcoding for High Efficiency Video Coding," in IEEE Transactions on Multimedia, vol. 18, no. 3, pp. 364-378, March 2016.

[50] Piljić I., Dragić L., Duspara A., Čobrnić M., Mlinarić H., Kovač M.," Bolt65 – performance-optimized HEVC HW/SW suite for Just-in-Time video processing," in International Convention on Information and Communication Technology, Electronics and Microelectronics, pp. 1121-1126, Opatija 2019.

[51] Intel Corporation, Intel Advanced Vector Instructions 2, available at https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-overview-intrinsics-for-intel-advanced-vector-extensions-2-intel-avx2-instructions (5th May 2019.)

[52] Piljić I., Dragić L., Kovač M.," Performance-efficient integration and programming approach of DCT accelerator for HEVC in MANGO platform," in Automatika Journal for Control, Measurement, Electronics, Computing and Communications, vol. 60, no. 2, pp. 245-252, 2019.

[53] Aalborg University: Video Trace Library, YUV video sequences, available at http://trace.eas.asu.edu/yuv/ (6th May 2019.)

[54] Xiph Video Test Media, available at https://media.xiph.org/video/derf/ (6th May 2019.)

[55] Xilinx, "UltraScale Architecture and Product Data Sheet: Overview," available at https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf, February 2019.

[56] ProFPGA, "FPGA Based Prototyping Solution," available at https://www.profpga.com/products/motherboards-overview/profpga-quad (6th May 2019.)

[57] Kvazaar HEVC encoder, available at https://github.com/ultravideo/kvazaar (6th May 2019.)

[58] Joint Collaborative Team on Video Coding Reference Software ver. HM 16.20, available at http://hevc.hhi.fraunhofer.de/ (6th May 2019.)

[59] Wikipedia contributors, “Peak signal-to-noise ratio,” available at https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio (6th May 2019.)

[60] Bossen F., “Common test conditions and software reference configurations,” Tech. Rep. JCTVC-H1100, 2012

[61] Yao F., Zhang X., Gao Z., Yang B., "Fast mode and depth decision algorithm for HEVC intra coding based on characteristics of coding bits," int IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), pp. 1-4., Nara 2016,

[62] Shen L., Zhang Z., An P., "Fast CU size decision and mode decision algorithm for HEVC intra coding," in IEEE Transactions on Consumer Electronics, vol. 59, no. 1, pp. 207-213, February 2013.

[63] Zhao L., Zhang L., Ma S., Zhao D., "Fast mode decision algorithm for intra prediction in HEVC," in Visual Communications and Image Processing (VCIP), pp. 1-4, Tainan 2011.

[64] Xilinx, “Vivado Design Suite User Guide: High-Level Synthesis,” available at https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug902-vivado-high-level-synthesis.pdf, February 2018.

[65] Xilinx, “AXI Reference Guide,” available at https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf, March 2011.

[66] Xilinx, “7 Series FPGAs Integrated Block for PCI Express 3.0”, available at https://www.xilinx.com/support/documentation/ip_documentation/pcie_7x/v3_0/pg054-7series-pcie.pdf, November 2014.

[67] Xilinx, “DMA subsystem for PCI Express – Performance numbers,” available at https://www.xilinx.com/support/answers/68049.html, April 2016.

LIST OF FIGURES

LIST OF TABLES

BIOGRAPHY

Igor Piljić was born in 1989 in Sarajevo, Bosnia and Herzegovina. He graduated in 2013. at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. After graduation, he started working as a software developer in “Hrvatski autoklub” (HAK). From 2014. he works in the Faculty of Electrical Engineering and Computing in Zagreb as a research associate in the Department of Control and Computer Engineering, where he started his postgraduate studies under the mentorship of prof.dr.sc. Mario Kovač. During postgraduate studies, he was included in several national and international research projects. His area of expertise includes high-performance computing and algorithm optimization for heterogeneous many-core architectures with special focus set on exploring multimedia architectures and just-in-time video transcoding. He is a member of HPC architecture and application research center and HiPEAC organization. He is author and co-author of several scientific papers published in international conferences and journals.

PUBLISHED PAPERS

1. Piljić I., Dragić L., Kovač M.,” Performance-efficient integration and programming approach of DCT accelerator for HEVC in MANGO platform”, in Automatika Journal for Control, Measurement, Electronics, Computing and Communications, vol 60, no 2., pp. 245-252, 2019.
2. Piljić I., Dragić L., Duspara A., Čobrnić M., Mlinarić H., Kovač M.,” Bolt65 – performance-optimized HEVC HW/SW suite for Just-in-Time video processing”, in International Convention on Information and Communication Technology, Electronics and Microelectronics, pp. 1121-1126, Opatija 2019.
3. Piljić I., Dragić L., Franček P., Kovač M., Mlinarić H., Gvozdanović D., “Bringing generations together: Social network adapted for elders”, in 57th International Symposium ELMAR (ELMAR), pp. 255-258, Zadar 2015.
4. Dragić L., Piljić I., Kovač M., “Dynamic load balancing algorithm based on HEVC tiles for Just-in-Time video encoding for heterogeneous architectures”, in Automatika Journal for Control, Measurement, Electronics, Computing and Communications, , vol 60, no 2., pp. 239-244, 2019.
5. Iranfar A., Terraneo F., Simon W.A., Dragić L., Piljić I., Zapater Sancho M., Fornaciari W., Kovač M., Atienza Alonso D., "Thermal characterization of next-generation workloads on heterogeneous MPSoCs", in International Conference on Embedded

Computer Systems: Architectures, Modeling, and Simulation (SAMOS), pp. 286-291., Pythagorion 2017.

6. Flich, J., Agosta G., Ampletzer P., Atienza D., Brandolese C., Cilardo A., Fornaciari W., Hoornenborg Y., Kovač M., Piljić I., Duspara A., Dragić L., Knezović J., Sruk V., Hofman D., Maitre B., Massari G., Mlinarić H., Papastefanakis E., Roudet F., Tornero R., Zoni D., "MANGO: Exploring Manycore Architectures for Next-GeneratiOn HPC Systems", 2017 Euromicro Conference on Digital System Design (DSD), pp. 478-485., Vienna, 2017.
7. Flich J., Giovanni A., Ampletzer P., Atienza D., Brandolese C., Cappe E., Cilardo A., Dragić L., Dray A., Duspara A., Fornaciari W., Fussela E., Gagliardi M., Guillaume G., Hofman D., Hoornenborg Y., Iranfar A., Kovač M., Libutti S., Maitre B., Martinez J.M., Massari G., Meinds K., Mlinarić H., Papastefanakis E., Picornell Sanjuan T., Piljić I., Pupykina A., Reghenzani F., Staub I., Tornero R., Zanella M., Zapater M., Zoni D., "Exploring Manycore Architectures for Next-Generation HPC Systems through the MANGO Approach", in Microprocessors and Microsystems, vol 61, pp. 154-170, Septemner 2018.
8. Dragić L., Piljić I., Kovač M., Mlinarić H., Franček P., Žagar M., Sruk V., Gvozdanović., "Home Health Smart TV - Bringing e-Health closer to elders", in Proceedings of the Fourth International Conference on Telecommunications and Remote Sensing Including a Special Session on eHealth Services and Technologies (EHST), pp. 116-121., Rhodos 2015.
9. Dragić L., Piljić I., Franček P., Kovač M., Mlinarić H., Gvozdanović D.," Home Health Smart TV - platform for accessing multimedia e-health content", in 57th International Symposium ELMAR (ELMAR), pp. 251-254., Zadar 2015.
10. Franček P., Piljić I., Dragić L., Mlinarić H., Kovač M., Gvozdanović D., "Overcoming e-health interoperability obstacles: Integrating PHR and EHR using HL7 CCD", in 57th International Symposium ELMAR (ELMAR), pp. 73-76., Zadar 2015.
11. Gvozdanović D., Kovač M., Mlinarić H., Dragić L., Piljić I., Franček P., Žagar M., Sruk V., "Interoperability Within E-Health Arena", in Proceedings of the Fourth International Conference on Telecommunications and Remote Sensing Including a Special Session on eHealth Services and Technologies (EHST), pp. 81-86., Rhodos 2015.

ŽIVOTOPIS

Igor Piljić rođen je 1989. godine u Sarajevu u Bosni i Hercegovini. Diplomirao je 2013. godine na Fakultetu elektrotehnike i računarstva u Zagrebu. Nakon diplome zaposlio se kao projektant programmer u Hrvatskom autoklubu (HAK). Od 2014. godine zaposlen je na matičnom fakultetu kao zavodski suradnik na Zavodu za automatiku i računalno inženjerstvo, a 2015. godine upisuje poslijediplomski studij pod mentorstvom prof. dr. sc. Maria Kovača. Tijekom doktorata sudjelovao je na provedbi nekoliko nacionalnih i međunarodnih istraživačkih projekata. Područja znanstvenog interesa su mu računarstvo visokih performanci s posebnim fokusom na algoritme za video procesiranje optimirane za izvođenje na raznorodnim arhitekturama. Član je centra za istraživanje arhitektura i aplikacija za računarstvo visokih performanci ("HPC Architecture and Application Research Centar") i organizacije HiPEAC. Autor je ili koautor više znanstvenih radova objavljenim u međunarodnim konferencijama i časopisima.