Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

Videokoodauksen rinnakkaistaminen

Kandidaatintyö

17. helmikuuta 2013

Miro Nurmela

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Miro Nurmela
Työn nimi:	Videokoodauksen rinnakkaistaminen
Päiväys:	17. helmikuuta 2013
Sivumäärä:	30
Pääaine:	Ohjelmistotekniikka
Koodi:	T3001
Vastuuopettaja:	Ma professori Tomi Janhunen
Työn ohjaaja(t):	TkT Vesa Hirvisalo (Tietotekniikan laitos)

Tämä kandinaatintyö on kirjallisuustutkimus rinnakkaislaskennan soveltamisesta videokoodaukseen. Työ esittelee videokoodauksen ja rinnakkaislaskennan perusteita sekä käytännön sovelluksia rinnakkaisista videokoodausmenetelmistä. Tavoite on löytää videokoodausta tehostavia rinnakkaisia ratkaisuja. Työssä ei keskitytä syvällisesti esimerkiksi rinnakkaisiin laitteisto- tai ohjelmistoratkaisuihin.

Tutkimuksen perusteella olemassa olevat videokoodausmenetelmät nojaavat jäykkään laskentaan, joka perustuu vahvasti videodatan sisäisiin riippuvuuksiin ja niitä hyödyntäviin optimointeihin. Videodatan laadun parantuessa tällaiset menetelmät ovat riittävän tehokkaita ainoastaan tehokkaimmilla yksiytimisillä prosessoreilla. Olemassa olevia menetelmiä ei ole suunniteltu rinnakkaisuutta silmällä pitäen, joten niiden rinnakkaistaminen on vaikeaa.

Rinnakkaislaskenta tarjoaa yksiytimisiä prosessoreita paremman suorituskyvyn, mutta tutkimuksen perusteella rinnakkaislaskentaan liittyy paljon haasteita. Rinnakkaislaskennan skaalautuvuus on haaste algoritmien suunnittelulle. Rinnakkaisuus tuo laskentaan kustannuksia, kuten laskentayksiköiden välistä viestintää, muistinhallintaa ja synkronointia, joita ei tavallisessa laskennassa kohdata. Yleispäteviä ratkaisuja näihin ongelmiin ei ole löydetty, sillä rinnakkaislaskennan kenttä on laaja, kiihdytysalustat monimuotoiset ja tekniikka jatkuvasti kehittyvää.

Haasteista huolimatta tutkimuksessa esitellään onnistuneita rinnakkaisia videokoodaustoteutuksia. Videodatan riippuvuuksien purkaminen, menetelmien optimointi ja uudet standardit mahdollistavat entistä tehokkaampaa ja parempilaatuista videokoodausta. Tutkimuksen johtopäätöksenä helppo rinnakkaistuvuus tulisi ottaa tulevissa standardeissa ja videokoodausmenetelmissä yhdeksi päätavoitteista.

Avainsanat:	videokoodaus, videokoodausstandardit, rinnakkaisohjelmointi, rinnakkainen videokoodaus, rinnakkaislaskenta
Kieli:	Suomi

Sisältö

1 Johdanto

Nykyaikainen videodata tallennetaan digitaaliseen muotoon. Videodatan käsittelyä digitaalisille tallennuslaitteille ja siirtokanaville sopivaan muotoon ja uudelleen katseltavaan muotoon kutsutaan videokoodaukseksi. Koodaaminen on tärkeää kaikenlaisen median, kuten äänen, kuvan ja videon, digitaaliselle tallentamiselle. Videodatan erikoisominaisuus on se, että se yhdistelee erilaisia medioita. Tämän seurauksena videodata vaatii suuren määrän tallennustilaa ja siirtokapasiteettia, mikä tekee videon koodaamisesta tärkeämpää. Samalla videodatan monialainen luonne lisää haasteita sen kompaktiin esittämiseen. Olemassa olevia videokoodausmenetelmiä määräävät standardi. (Richardson 2010)

Videodatan laadun kasvu ja tiedon langattoman siirtämisen yleistyminen pakottaa kehittämään uusia, tehokkaampia videokoodausmenetelmiä. Ollaan kuitenkin tultu siihen pisteeseen, että vain tehokkaimmat yksiprosessorijärjestelmät pystyvät koodaamaan korkealaatuista videodataa. Kustannustehokkaan ja riittävän laskentatehon saavuttamiseksi tarvitaan tulevaisuudessa muita ratkaisuja, kuin ainainen yksittäisen prosessorin kellotaajuuden kasvatus (Moore 1965, Vajda 2011). Rinnakkaislaskenta mahdollistaa halvemman ja paljon tehokkaamman ratkaisun laskentatehon kasvattamiseen. (Peng et al. 2012, Chi et al. 2012)

Videokoodaus on kuitenkin laskennallisesti jäykkä tehtävä videodatan runsaista sisäisistä riippuvuuksista ja olemassa olevien videokoodausmenetelmien toteutusten johdosta. Tästä johtuen videokoodausmenetelmien rinnakkaistaminen ja rinnakkaislaskennasta näkyvien hyötyjen saaminen on vaikeaa. (Pieters et al. 2012, Li et al. 2012) Tämän työn tavoite on esitellä videokoodausta, rinnakkaislaskentaa ja ratkaisuja rinnakkaisen videokoodauksen toteuttamiseksi.

Rinnakkaislaskennasta on tutkimusten mukaan hyötyä videokoodaukselle. Tärkeää on ottaa rinnakkaislaskenta huomioon uusia standardeja suunniteltaessa, jotta ne leviävät laajempaan käyttöön. Yleistyvät moniydinprosessorit tukevat tätä ajatusta. Jatkotutkimuksena kannattaisi tutkia, millaiset ratkaisut soveltuvat mahdollisimman monelle rinnakkaiselle kiihdytysalustalle ja miten mahdollisimman tehokkaita rinnakkaisia ratkaisuja olisi mahdollisimman helppo tuottaa.

Luvussa 2 käsitellään perusasioita elävästä kuvasta, videokoodauksesta ja rinnakkaislaskennasta. Kappale toimii lyhyenä johdantona työn aiheisiin, jos ne eivät ole aiemmin tuttuja.

Videodatan on luonteeltaan monijakoista, sillä se sisältää elävää kuvaa, ääntä sekä mahdollisia lisäominaisuuksia. Videokoodauksen eri osat eivät ole monimutkaisia operaatioita, mutta datan riippuvuudet ja sen määrä asettavat haasteita videokoodausmenetelmille. Luvussa 3 esitellään videokoodauksen ja videodatan peruskäsitteitä sekä mittareita

videodatan ja videokoodauksen laadulle. Perinteiset videokoodausmenetelmät käyttävät videodatan riippuvuuksia hyväkseen koodausprosessin parantamiseksi. (Richardson 2010, Du ja Swamy 2010)

Rinnakkaislaskennan tavoite on laskentatehon kasvatus monia laskentayksiköitä hyödyntämällä. Rinnakkaisuuteen liittyy kuitenkin paljon haasteita, kuten laskennan järjestäminen, sopivan rinnakkaisen toteutuksen löytäminen kulloinkin käsillä olevaan ongelmaan ja suuri määrä erilaisia rinnakkaisia kiihdytysalustoja. Luvussa 4 esitellään rinnakkaisuuden peruskäsitteitä, haasteita ja apoja mitata rinnakkaisten järjestelmien tehokkuutta. Rinnakkaisten järjestelmien tehokkuuden mittaamisessa tärkeintä on, että ne ovat peräkkäisiä toteutuksia tehokkaampia. (Grama et al. 2003, Rauber ja Gudula 2010)

Rinnakkaislaskennan soveltaminen videokoodaukseen on erityisen hankalaa videodatan riippuvuuksien ja datan määrän johdosta. Riippuvuuksien hyödyntäminen on toiminut olemassa olevilla menetelmillä, mutta videodatan laadun ja määrän kasvaessa nämä keinot alkavat olla riittämättömiä ((Chi et al. 2012, Peng et al. 2012)). Luvussa 5 esitellään rinnakkaisen videokoodauksen ongelmien lisäksi luvussa joukko ratkaisuja sen toteuttamiseen. Ratkaisuja ovat uudet standardit, riippuvuuksien ja synkronoinnin tarpeen vähentäminen ja menetelmien optimointi. Näillä keinoilla on saavutettu näkyviä hyötyjä videokoodauksen tehokkuuteen ja laatuun. (Pieters et al. 2012, Chi et al. 2012, Peng et al. 2012)

Työn johtopäätökset esitellään luvussa 6.

2 Digitaalinen liikkuva kuva ja rinnakkaislaskennan perusteet

Tässä luvussa käsitellään digitaalista videota, rinnakkaislaskennan historiaa sekä kahta liikkuvaan kuvaan ja rinnakkaislaskentaan liittyvää käsitettä, havaitsemista ja rinnakkaislaskennan oikeellisuutta. Luvun tavoitteena on toimia johdantona työn aiheisiin eikä niissä syvennytä mainittuihin teemoihin kovin tarkasti.

2.1 Digitaalinen video ja sen koodaus

Ensimmäisistä liikkuvan kuvan sovelluksista 1800-luvun lopulta alkaen elävä kuva on perustunut tallennettujen kuvanäytteiden nopeaan toistamiseen liikkuvan kuvan vaikutelman aikaansaamiseksi. Eräs ensimmäisistä ja pisimpään selvinneistä tallennusformaateista oli filmi, jolle valotettiin peräkkäisiä valokuvia ja tulosta esitettiin projektorilla. Vuosien saatossa tallennusmenetelmät ovat kehittyneet ja siirtomenetelmät kehittyneet ensin analogisiksi radiosignaalien esimerkin innoittamana ja myöhemmin digitaalisiksi

digitaalisen vallankumouksen myötä. (Mitra 2010)

Siirtymä digitaalisen videon aikakaudelle on tapahtunut viimeisen viidentoista vuoden aikana. VHS-kaseteista ja analogisista TV-lähetyksistä on siirrytty Blu-Ray -tekniikoihin, kännykkäkameroihin teräväpiirtotelevisioihin. Tekniikan kehitys ei näy ainoastaan tallennusmedioissa, vaan tiedonsiirtomenetelmät ovat niin ikään on kehittyneet ja muuttuneet monin paikoin langattomaksi.(Richardson 2010) Tallentamisen ja siirtämisen helpottumisen johdosta videodataa on maailmassa yhä enemmän (Cisco 2013, YouTube 2013).

Raaka videodata vaatii suuret määrät tallennustilaa eikä sen siirtäminen langattomasti ole mahdollista - tarvitaan siis teknologia, jolla videodataa pakataan ja puretaan (enkoodataan ja dekoodataan). Tätä pakkaamisen ja purkamisen prosessia kutsutaan videokoodaukseksi. Pakkaamisen ja purkamisen lisäksi videokoodaus kattaa myös signaalin reaaliaikaisesta kääntämisestä (transkoodaus). Transkoodaus tarkoittaa esimerkiksi enkoodatun datan kääntämistä toiseen koodausstandardiin (Angelides ja Agius 2011). Koodekki (CODEC, lyhenne sanaparista coder decoder pair) on videokoodausta suorittavaan ohjelmisto tai laite (Richardson 2010). Videokoodausta käsitellään luvussa 3.

Videokoodausstandardeja on useita, mainittakoon tässä H.264, MPEG-4 ja DICOM, joista kaksi ensimmäistä ovat kuluttajakäyttöön suunniteltuja standardeja ja viimeinen lääketieteelliseen kuvantamiseen suunniteltu. (Richardson 2010)

2.2 Rinnakkaislaskennan perusteita

Rinnakkaislaskennan (parallel computing) tavoite on parantaa laskennan suorituskykyä. Käsitteellisesti rinnakkaislaskentaa ei kannata sekoittaa samanaikaiseen laskentaan (concurrent computing). Akateemisessa mielessä jälkimmäinen keskittyy rinnakkaisen laskennan oikeellisuuteen, kun taas ensimmäinen keskittyy saamaan hajautetusta ja rinnakkaisesta laskennasta näkyviä hyötyjä laskentatehoon. (Grama et al. 2003, Ben-Ari 2006) Tämä työ keskittyy rinnakkaisella laskennalla saavutettuihin hyötyihin videokoodauksen saralla, mutta myöhemmin tässä luvussa esitellään lyhyesti rinnakkaislaskennan oikeellisuuden tarkastelua.

Rinnakkaislaskennan nimi on suuntaa-antava. Laskennan suorituskykyä pyritään parantamaan suorittamalla laskentaa mahdollisimman monella laskentayksiköllä samaan aikaan. Ajatus on looginen - jos yhdeltä laskentayksiköltä kestää yhden aikayksikön suorittaa jokin laskentatehtävä, niin sadan laskentayksikön suorittaminen kestää sata aikayksikköä. Sadalta laskentayksiköltä taas menee samaan tehtävään vain yksi aikayksikkö. Intuitiivisesta perusajatuksesta huolimatta rinnakkaislaskennalla on mittavat ongelmat, joiden johdosta se on pysynyt kehityksen marginaalissa vuosikymmeniä. (Grama et al. 2003) Näihin ongelmiin ja rinnakkaislaskennan muihib piirteisiin keskitytään rinnakkaislaskentaa käsittelevässä luvussa 4.

2.3 Rinnakkaislaskennan oikeellisuus

Laskentatehon lisäyksen lisäksi rinnakkaisuus lisää virhelähteitä laskentaan. Monen laskentayksikön samanaikainen muistinkäsittely, laskennan aikataulutus ja viestikanavien käyttö ovat esimerkkejä virheläheistä, joita ei perinteisessä peräkkäisessä ohjelmoinnissa esiinny. Rinnakkaisten järjestelmien oikeellisuuden tarkastelua vaikeuttaa se, että virheitä on vaikea toistaa. Virheet saattavat riippua hyvin tarkoista ajoituksista ja sopivista suoritusjärjestyksistä ohjelman ajon aikana eivätkä täten toistu kaikkien ajojen aikana. (Ben-Ari 2006)

Yrityksellä ja erehdyksellä virheiden etsiminen on kallista ja aikaa vievää, joten parempia keinoja tarvitaan. Eräs analyyttinen ratkaisu rinnakkaislaskennan oikeellisuuden tarkasteluun on mallintarkistus. Tässä menetelmässä rinnakkainen ohjelma kuvataan mallina, joka täyttää jollekin abstraktion tasolle asti ohjelmalle asetetut vaatimukset. Mallia voidaan tutkia automaattisilla työkaluilla, jotka paljastavat mahdollisia rinnakkaisuuden ongelmia, kuten muistin ylikirjoittamista, ei-toivottuja kisatilanteita tai laskennan umpikujia (deadlock). (Ben-Ari 2006) Mallintarkastus on laskennallisesti erittäin vaativa tehtävä, mutta se saattaa säästää järjestelmän myöhemmiltä, mahdollisesti hyvin kalliilta ja vaikeilta ongelmilta.

Tässä työssä ei tarkemmin syvennytä rinnakkaislaskennan oikeellisuuteen, mutta se on eräs rinnakkaislaskennan monista ongelmista ja aktiivisen tutkimustyön kohde.

3 Videokoodaus

Tässä luvussa käsitellään videokoodauksen perusteita. Luvun tavoite on antaa riittävät taustatiedot videokoodausmenetelmistä, jotta myöhemmässä vaiheessa tätä työtä voidaan esitellä rinnakkaisia videokoodausmenetelmiä sekä antaa hyvä yleiskuva videokoodauksesta. Yleisen käsittelyn lisäksi tekstissä on esitelty lyhyesti muutama mielenkiintoinen tutkimus, jotka ratkovat joitakin videokoodauksen ongelmia.

3.1 Videokoodauksen peruskäsitteet

3.1.1 Videokuvan esittäminen ja tallentaminen

Havaitsemamme maailma on täynnä erilaisia kohteita, joilla on erilaisia ominaisuuksia, kuten muoto, syvyys, tekstuuri, väri tai valotiheys. Maailma on niin ikään jatkuva, mitä esimerkiksi digitaalinen maailma ei ole. Jotta kameralla tai vastaavalla laitteella voitaisiin tallentaa esitys maailmasta, täytyy havainnot käsitellä ja tallentaa digitaaliseen maailmaan sopivaksi.

Videodataa varten maailmasta täytyy kerätä näytteitä eri ajanhetkiltä. Otanta tapahtuu kahdessa ulottuvuudessa, ajassa ja tilassa (temporal and spatial sampling). Tyypillisesti tilaotokset ovat suorakaiteen muotoisia kuvia maailmasta, kun ajallinen otanta koostuu peräkkäisistä tilaotoksista. Tilaotokset koostuvat neliönmuotoisista kuvapisteistä eli pikseleistä, jotka on järjestetty ruudukoksi. Käsittelemättömiä aika- ja tilanäytteitä kutsutaan raakadataksi. (Richardson 2010, Du ja Swamy 2010)

Jokaisen tilanäytteen kuvapisteeseen tallennetaan pisteeseen liittyvät tiedot. Yksinkertaisinta kuvaa eli mustavalkokuvaa varten riittää tallentaa vain yksi arvo kuvapisteestä (kirkkaus tai valotiheys), mutta värikuvassa täytyy tallentaa jokaisessa pisteessä esiintyvien värien määrä. Väriavaruudeksi kutsutaan menetelmää, joka on valittu kuvapisteiden kirkkauden, valotiheyden ja värin kuvaamiseksi. Värikuvan tallentamisessa suosittu tapa on RGB-väriavaruus. Tässä menetelmässä jokaisella kuvapisteellä on kolme parametria: punaisen, vihreän ja sinisen värin määrä. RGB-väriavaruudessa valotiheys ja väri ovat samanarvoisia, mutta edistyneemmät väriavaruudet hyödyntävät ihmisaivojen suurempaa herkkyyttä valotiheydelle kuin väreille. (Richardson 2010, Du ja Swamy 2010) Väriavaruudet ovat matemaattisia malleja havaitusta maailmasta. Niille on paljon erilaisia implementaatioita, joihin ei tämän työn puitteissa syvennytä tarkemmin.

Avaruudellisia näytteitä voidaan ottaa joko peräkkäin (progressive) tai lomittain (interlacing). Peräkkäisessä otannassa jokaisella ajanhetkellä otetaan otos, johon tallennetaan käytössä olevan standardin mukainen määrä kuvapisteitä. Otosta kutsutaan ruuduksi (frame). Lomittaisessa otannassa taas jokaisella ajanhetkellä tallennetaan puolet standardin määräämistä kuvapisteistä vaakasuunnassa joka toinen rivi siten, että peräkkäisillä ajanhetkillä tallennetaan lomittaiset rivit. Otosta kutsutaan kentäksi (field). Erotuksena näillä metodeilla on siinä, että lomittaisella otannalla syntyy pehmeämmin liikkuvaa kuvaa kuin peräkkäisellä otosten määrän ollessa sama. Lomittamalla jokaiseen otokseen tallennetaan myös vähemmän tietoa. (Richardson 2010, Du ja Swamy 2010)

3.1.2 Videodatan muut osat ja synkronointi

Videodata ei koostu ainoastaan varsinaisesta videokuvasta, vaan kuvan lisäksi tallennusvaiheessa tallennetaan ääntä. Ääniraita enkoodataan kuvadatan tavoin halutulla menetelmällä ja tallennetaan kuvadatan oheen. Tässä työssä ei syvennytä tarkemmin äänen koodaamiseen, mutta perusperiaatteet pakkausmenetelmissä ovat samankaltaiset. Kuvan ja äänen lisäksi videokoodauksen eri vaiheissa dataan voidaan lisätä haluttuja ominaisuuksia, kuten tekstitystä. Lisäominaisuudet tallennetaan usein varsinaisesta videodatasta erillään. (Angelides ja Agius 2011, Mujal ja Kirlin 2002)

Videodatan osien ollessa erillisiä tarvitaan keino tahdistaa data eri lähteistä hyvän katselukokemuksen takaamiseksi. Perusmenetelmiä on kolme. Ensimmäinen on aikaleima-

tahdistus (Time-Stamps Synchronization), jossa eri lähteisiin lisätään aikaleimoja. Aikaleimojen perusteella osataan eri lähteistä tuleva data näyttää käyttäjälle samanaikaisesti. Toinen keino on tahdistusmerkki (Synchronization Marker), joka käytännössä tarkoittaa eri datalähteiden välistä kommunikaatiota, käytännössä merkin lähettämistä ja vastaanottamista, tahdistuksen saavuttamiseksi. Kolmas on limitystahdistus (Multiplex Synchronization), jossa limitetään eri datalähteet yhdeksi lähteeksi hyödyntäen näin niiden luonnollista järjestystä tahdistuksen saavuttamiseksi. (Qi et al. 2011, Mujal ja Kirlin 2002)

Perusmenetelmien lisäksi on tutkittu erilaisia metodeita sulauttaa ääntä ja lisäominaisuuksia varsinaiseen videodataan pakkauksen tehostamiseksi ja synkronoinnin vähentämiseksi. Tällöin enkoodausvaiheessa kuvadataan lisätään äänidataa tai tiedot lisäominaisuuksista. Tiedon sulauttamista kutsutaan joissain tutkimuksissa tiedon piilottamiseksi. Tehdyssä tutkimuksessa videon tai kuvan laadun heikkeneminen ei ollut ihmissilmällä havaittavissa. (Qi et al. 2011, Mujal ja Kirlin 2002, Liao et al. 2011)

3.1.3 Pakkaus ja ennustaminen

Kaikenlaisen median pakkausmetodit voidaan jakaa häviöttömiin ja häviöllisiin pakkausmetodeihin (lossy, lossless). Häviöttömissä metodeissa pakatusta datasta pystytään purkuvaiheessa palauttamaan täydellinen versio alkuperäisestä datasta, häviöllisessä pakkauksessa purettu versio on approksimaatio alkuperäisestä. On olemassa häviöttömiä videokoodausmenetelmiä, mutta pakkaussuhde jää liian alhaiseksi käytännön sovelluksiin. Häviölliset pakkausmenetelmät keskittyvät poistamaan datasta subjektiivista redundanssia eli yksityiskohtia, jota havainnoija ei kykene havaitsemaan. Näin saavutetaan huomattavasti tehokkaampia videokoodausmenetelmiä. (Richardson 2010, Du ja Swamy 2010) Toisaalta esimerkiksi lääketieteelliset sovellukset tai konenäkösovellukset vaativat korkealaatuista kuvaa, jolloin häviötön pakkaaminen on ainoa vaihtoehto (Peng et al. 2012).

Suurin osa videokoodausmenetelmistä hyödyntää videodatan vahvaa avaruudellista ja ajallista redundanssia. Käytännössä peräkkäisissä ruuduissa on suurella todennäköisyydellä lähes samat kuvapisteet (ajallinen) ja yhdessä ruudussa lähekkäiset kuvapisteet muistuttavat toisiaan suurella todennäköisyydellä (tilallinen). Monien koodekkien ensimmäinen askel videon koodaamiseen on ennustaminen - jollakin määrällä edellisiä ajallisia näytteitä voidaan melko suurella todennäköisyydellä ennustaa, mitä tässä näytteessä tulee olemaan. Samaan tapaan saman näytteen sisällä jo käsiteltyjen kuvapisteiden perusteella ennustetaan tulevien kuvapisteiden laatua. Ennustuksilla voidaan vähentää tallennettavan datan määrää, kun purkuvaiheessa näytteitä pystytään uudelleenrakentamaan edellisten näytteiden perusteella. Läheisesti ennustamiseen liittyvä käsite on liike-

kompensaatio (motion compensation). Käytännössä liikekompensaatio on ennustamisen apukeino. Liikekompensaation johdosta ei verrata ainoastaan peräkkäisissä näytteissä samoilla koordinaateilla olevia kuvapisteitä vaan myös sopivalla tavalla lähialueilta valittuja pisteitä. (Richardson 2010, Du ja Swamy 2010)

3.2 Videodatan riippuvuudet

Videodata on luonteeltaan monijakoista sen sisältäessä niin liikkuvaa kuvaa, ääntä kuin mahdollisia lisäominaisuuksia. Näiden sekä enkoodauksen aikaisen ennustamisen johdosta koodatulla videodatalla on vahvoja riippuvuussuhteita. Riippuvuuksia on kahdentyyppisiä. Ajallinen riippuvuus tarkoittaa tässä yhteydessä sitä, että onnistunut videodatan toistaminen on riippuvainen kaikista videodatan osien samanaikaisesta toistamisesta. Datariippuvuutta syntyy ruutujen ja kenttien sisällä esimerkiksi ennustettaessa uuden kuvapisteen arvoa edellisten kuvapisteiden perusteella tai uusia näytteitä ennustettaessa vanhojen perusteella. (Mujal ja Kirlin 2002)

Riippuvuuksien johdosta videokoodausprosessi on laskennallisesti jäykkä, mikä vaikeuttaa videokoodausmenetelmien rinnakkaistamista. Näitä ongelmia käsitellään tarkemmin luvussa 5.2. Videokoodauksen yksittäiset vaiheet eivät ole monimutkaisia, mutta synkronoinnin ja erilaisten laskentaa tehostavien optimointien johdosta koodekit ovat varsin suuri ohjelmistoja. Esimerkiksi Esimerkiksi x264-ohjelmisto, joka on avoimen lähdekoodin toteutus suositusta H 264 -standardista (Richardson 2010), on n. 100 000 riviä koodia (VideoLan 2013).

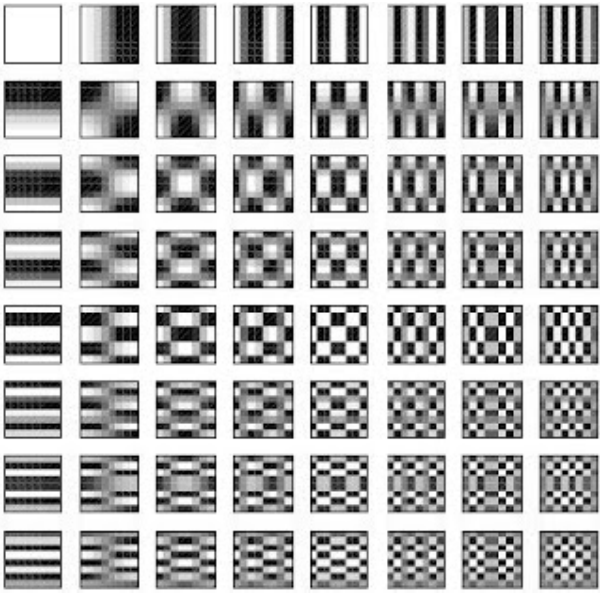
Rinnakkaislaskennan lisäksi on ehdotettu luoviakin ratkaisuja videokoodauksen tehostamiseksi. Lee et al. 2011 ehdottaa menetelmää, jossa videon äänilähteet piirretään suuremmalla tarkkuudella, koska ihmisten on luonteva keskittää huomionsa äänilähteisiin. Mainitut lähteet (Mujal ja Kirlin 2002, Qi et al. 2011) esittelevät keinoja helpottaa videodatan synkronointia ja vähentää tallennustilan tarvetta.

3.3 Diskreetti kosinimuunnos

Ennen tallentamista tai siirtämistä videodataa useimmiten käsitellään vielä jollain tavalla ennustusten lisäksi. Tavoitteena on vähentää tallentamiseen tarvittavaa muistia ja helpottaa aikanaan tapahtuvaa dekoodausta. Tekniikoita on monia, mutta esitellään tässä tunnetuin, eli diskreetti kosinimuunnos (Discrete Cosine Transformation, DCT).

DCT on Fourier-muunnoksen kaltainen muunnos, joka operoi $N \times N$ kokoisilla blokeilla avaruudellisissa näytteissä. Muunnos tuottaa niin ikään $N \times N$ kokoisen blokin, mutta kuvapisteiden arvojen sijaan uuden blokin arvot ovat kosinimuunnoksen peruslohkojen suhteellisia voimakkuuksia koodattavassa olevassa lohkoissa. Kuvassa 1 on esitetty 8×8

-lohkon DCT:n peruslohkot. (Richardson 2010)



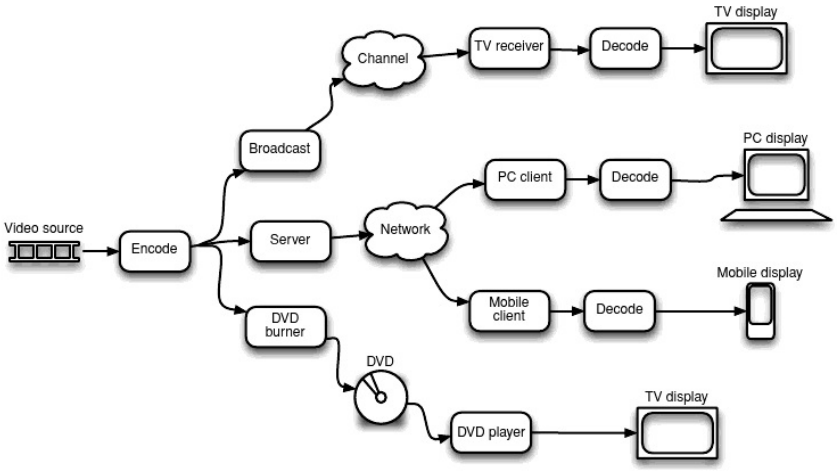
Kuva 1: 8×8 DCT:n peruslohkot

DCT:n hyöty ei ole ilmeinen, sillä muunnos tapahtuu $N \times N$ -lohkosta $N \times N$ -lohkoon, eikä tässä säästetä lainkaan tilaa. Hyöty ilmenee purkuvaiheessa - on mahdollista purkaa DCT:n avulla tallennettua tietoa riittävän tarkaksi käyttämällä vain osaa DCT:n tuottamista suhteellisista voimakkuuksista. Voidaan siis tallentaa vain osa DCT:n tuottamista arvoista ja käyttää näin vähemmän tilaa tiedon tallentamiseen. Tyypillisesti käytetään DCT:n tuottamat suurimmat arvot eli merkityksellisimmät peruslohkot, jolloin harvinaisemmat ja räikeimmät erot jäävät puuttumaan lopullisesta kuvasta. Piirtämättä jäävät sellaiset yksityiskohdat, joita ihmissilmä on muutenkin heikko havaitsemaan. DCT:tä käyttävät koodausmenetelmät ovat siis pääosin häviöllisiä. (Richardson 2010, Du ja Swamy 2010)

3.4 Videodatan matka lähteestä näyttölaitteelle

Videodata saa alkunsa lähteestä, joka on tyypillisesti kamera, joka tallentaa raakadatan. Tämän jälkeen suoritetaan koodekki suorittaa enkoodauksen, minkä jälkeen tiivistetty data voidaan tallentaa tai siirtää. Ketjun toisessa päässä on jälleen koodekki joka suorittaa tällä kertaa dekoodauksen ja esittää videon käyttäjälle. Kuva 2 havainnollistaa videodatan eri reittejä käyttäjälle. (Richardson 2010)

Koodauksen kolmas muoto, transkoodaus, tulisi tehdä, jos haluttu näyttölaite ei tuekaan sille tarjottua koodaustapaa.



Kuva 2: Videokoodaus mahdollistaa datan liikkuvuuden

3.5 Kuvadatan ja videokoodauksen laadun mittarit

Videodatan laatua voidaan mitata subjektiivisesti ja objektiivisesti. subjektiivinen mittaaminen on vaikeaa, sillä ihmisten arvioihin vaikuttavat monet seikat. Objektiivinen mittaaminen taas antaa selviä lukuja vastaukseksi, mutta tulosten merkitys ja vaikutus katsojakokemukseen on selvitettävä erikseen. Videodatan kohdalla subjektiivista mittaamista voidaan usein pitää tärkeämpänä kuin objektiivista, sillä tavoitteena usein on mahdollisimman miellyttävän katselukokemuksen tuottaminen käyttäjälle. (Richardson 2010) Toisaalta videodatan objektiivinen laatu on erityistapauksissa myös tärkeää, esimerkiksi konenäköä hyödyntävissä sovelluksissa tai koneiden muuten tulkitessa videodataa. Videodatan määrän kasvaessa jatkuvasti ja konenäkösovellusten, kuten robottien, lisääntyessä saattaa objektiivisesta laadusta tulla subjektiivista tärkeämpää (YouTube 2013, Cisco 2013).

Subjektiivinen laadun mittaamiseen liittyy olennaisesti havaitseminen. Subjektiivista laatua voidaan mitata esimerkiksi kyselytutkimuksilla ja havaitsemistutkimuksen muita keinoja, kuten silmänliikeanalyysiä, voidaan käyttää videokoodausmenetelmien arvioimiseen ja kehittämiseen. (Oh ja Kim 2013)

Seuraavassa esitellään muutamia objektiivisia tapoja mitata videodatan laatua.

Tiheys, jolla ruutuja tallennetaan, määrää ruutunopeuden (frame rate). Ruutunopeus ja kuvapisteiden määrä tarjoaa helpon mittarin kuvan laadulle. Normaalitarkkuuksinen ja korkeatarkkuuksinen (Standard Definition, High Definition) videodata eroavat toisistaan juuri kuvapisteiden määrän ja ruutunopeuden perusteella. Valittu ruutujen tai kenttien esitystapa vaikuttaa kuitenkin subjektiiviseen laatuun, joten ruutunopeuden ja kuvapisteiden määrää ei voi pitää ehdottomana laadun mittarina. (Richardson 2010, Du ja Swamy 2010)

Yleisesti käytetty mittari videodatan laadun mittaamiseen on PSNR-arvo (Peak Signal to Noise Ratio). PSNR kertoo erosta alkuperäisen ja uuden kuvan välillä ja se on määritelty

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

missä MSE on keskimääräinen neliöity virhe (Mean Squared Error) alkuperäisen kuvan ja uuden kuvan välillä. (Richardson 2010)

PSNR on helppo laskea ja antaa erään objektiivisen arvion videon laadusta. Menetelmällä on puutteensa, kuten se, että alkuperäistä kuvaa ei ole välttämättä saatavilla. Toisaalta, kuten monissa objektiivissa mittareissa, pelkkä PSNR arvo ei vastaa suoraan mitään subjektiivista arvoa. Yleisesti ottaen korkea PSNR arvo tarkoittaa hyvää laatua ja matala arvo huonoa. (Richardson 2010, Du ja Swamy 2010)

Videokoodauksen laatua voidaan mitata esimerkiksi koodausnopeudella tai pakkaussuhteella, eli raakadatan ja pakatun datan koon suhteella. (Li et al. 2012, Peng et al. 2012). Koodausnopeuden yksikkönä voidaan pitää esimerkiksi $fracruutuasekunti$.

4 Rinnakkaislaskenta

Tässä luvussa käsitellään rinnakkaislaskennan perusteita ja haasteita. Tekstissä esitellään lyhyesti myös kuuluisa Amdahlin laki sekä muutama tutkimus automaattisten rinnakkaistamisen toteutukseen.

4.1 Rinnakkaisuus tietokonejärjestelmissä

Rinnakkaisuutta esiintyy tietokonejärjestelmissä monessa muodossa. Nykyaikaisissa supertietokoneissa on jopa yli miljoona ydintä (Meuer et al. 2013). Toisaalta suurten supertietokoneiden lisäksi rinnakkaisuus on tullut aivan tavallisten kuluttajatietokoneiden osaksi moniydinprosessoreiden myötä. Prosessorit ovat jo pitkään hyödyntäneet erilaisten laskentayksiköiden samanaikaista käyttöä laskennan tehostamiseen. Viimeksi mainittu menetelmä tunnetaan termillä liukuhihnaus (pipelining). Liukuhihnalaskennan tehonlisäys seuraa siitä, että laskentaa voidaan prosessorissa suorittaa suuremmalla kellotaajuudella rinnakkaisten operaatioiden johdosta. Samalla rinnakkaisuutta esiintyy kerroksittain, esimerkiksi tavallisessa moniydinprosessorissa on monta ydintä, ja jokaisella ytimellä on omat liukuhihnansa. (Grama et al. 2003, Rauber ja Gudula 2010)

4.2 Rinnakkaisten menetelmien hallintarakenteet

Rinnakkaislaskennan hallintarakenteet voidaan jakaa karkeasti kahteen luokkaan sen perusteella, miten paljon laskennan hallintaa kullekin laskentayksikölle on jaettu. Jos hallinta on keskitetty, kyseessä on SIMD-arkkitehtuuri (Single Instruction stream, Multiple Data stream). SIMD-arkkitehtuurissa yksittäinen hallintayksikkö päättää, mitä kukin laskentayksikkö laskee. MIMD-arkkitehtuurissa (Multiple Instrucrion stream, Multiple Data stream) jokaisella laskentayksiköllä on oma hallintayksikkönsä. Konkreettisena esimerkkinä arkkitehtuurien erona voi pitää esimerkiksi sitä, että MIMD-arkkitehtuurissa jokainen laskentayksikkö voi suorittaa eri ohjelmia, kun SIMD-arkkitehtuurissa hallintayksikkö päättää, mitä ohjelmaa kaikki laskentayksiköt suorittavat. Laskennan pilkkomista ja jakamista laskentayksiköille SIMD-menetelmässä kutsutaan vektoroinniksi. (Grama et al. 2003, Rauber ja Gudula 2010)

Rinnakkaiset järjestelmät voidaan jakaa myös manycore- ja multicore-järjestelmiin [1]. Yhteistä molemmille järjestelmille on useat laskentayksiköt. Multicore-järjestelmässä eri ydinten välimuistit on pakotettu yhtenäiseen tilaan. Toisin sanoen yhden ytimen laskennan tulokset päivitetään kaikkien muiden ytimien välimuisteihin. Manycore-järjestelmissä välimuistien yhtenäisyysvaatimukset eivät ole yhtä tiukkoja, joten manycore-järjestelmiä voidaan tässä mielessä pitää multicore-järjestelmiä rinnakkaisempina. (Vajda 2011)

4.3 Rinnakkaisuuden peruskäsitteitä

Hallintarkanteiden ja eri tasojen rinnakkaisuuksien lisäksi rinnakkaisuus voidaan jakaa kahteen tyyppiin sen perusteella, mihin rinnakkaisuus kohdistuu. Tehtävärinnakkaisessa laskennassa rinnakkain suoritetaan erilaisia tehtäviä - esimerkiksi saman ohjelman eri säikeitä. Datarinnakkaisessa ohjelmoinnissa taas rinnakkaisuus syntyy siitä, että operoidaan samaan aikaan datan eri osa-alueilla. (Grama et al. 2003)

4.3.1 Tiedon välittäminen laskentayksiköiden välillä

Rinnakkaisuuden kohteen lisäksi rinnakkaisilla kohteilla on erilaisia tapoja järjestää tiedon liikkuminen laskentayksiköiden välillä. Pääparadigmoja on kaksi, jaettu muisti (shared memory) ja viestinvälitys (Message Passing Interface, MPI). Jaetun muistin mallissa laskentayksiköillä on yhteinen muisti, josta jokainen saa lukea ja kirjoittaa. Jotta yhteinen muisti pysyy puhtaana eli kaksi laskentayksikköä ei lue ja/tai kirjoita samaan muistiosoitteeseen samaan aikaan, täytyy järjestelmällä olla keinot päättää milloin mitäkin muistialuetta saa käsitellä. Erilaisia ratkaisuja tähän poissulkevuuden ongelmaan (mutual exclusion) on useita, kuten monitorit tai semaforit. Samojen muistialueiden käsittelyyn

[1] Suomen kielessä sana moniydinprosessori viittaa tyypillisesti multicore-järjestelmään

liittyy myös käsite kisatilanteista (race condition). Kisatilanteessa kaksi laskentayksikköä tavoittelee samaa muistialuetta, mutta valitun muistinsuojausmenetelmän pitäisi estää tämä. Kisatilanteet eivät varsinaisesti ole rinnakkaisten ohjelmien vikoja vaan syntyvät rinnakkaisohjelmoinnin sivutuotteena. (Ben-Ari 2006)

Viestinvälitysmenetelmässä jokaisella laskentayksiköllä on oma muistiavaruutensa, ja jos esimerkiksi yksiköiden välillä halutaan synkronoida, niin ne vaihtavat viestejä. Menetelmä vaatii, että jokaisella laskentayksiköllä on tunniste, jolla sen voi erottaa muista. Tämä vaatimus ei koske jaetun muistin menetelmää. Luonnollisesti viestien välittämiseen tarvitaan myös jokin kanava niiden toimittamiseen. Viestien välittäminen on jaetun muistin käyttämistä tehottomampaa, mutta sitä käyttämällä on helpompi pitää yllä erilaisia laskentayksiköitä. Kaikkien käyttäessä samaa muistia yksiköiden tulee käsitellä muistia samalla tavalla, mutta viestejä välittäessä jokaisen yksikön täytyy vain täyttä viestinvälitysrajapinnan vaatimukset. (Grama et al. 2003, Rauber ja Gudula 2010)

4.3.2 Skaalautuvuus

Skaalautuvuus kertoo saavutetaanko rinnakkaislaskennasta laskentayksiköiden määrään suhteessa olevaa hyötyä, eli tehostuuko laskenta laskentayksiköitä lisäämällä. Laskentayksiköiden lisääminen ei loputtomasti kasvata rinnakkaislaskennan tehokkuutta. Hyvin skaalautuvan rinnakkaislaskentamallin laskenta-aika pysyy vakiona ongelmaa ja laskentayksiköiden määrää kasvattaessa. (Rauber ja Gudula 2010) Toinen hyvin skaalautuvan rinnakkaisen ratkaisun merkki on, että sen suorituskyky kasvaa lineaarisesti laskentayksiköiden määrää kasvatettaessa (Grama et al. 2003). Skaalautuvuus on välttämätön vaatimus tehokkaille rinnakkaisille järjestelmille (Grama et al. 2003. Rauber ja Gudula 2010).

4.3.3 Rakeisuus ja laskennan järjestäminen

Jotta laskentatehtäviä voidaan suorittaa rinnakkaisesti, täytyy tehtävät hajottaa (decompose) pienempiin osiin. Hajotusta tehdessä otetaan huomioon laskennan erilaiset riippuvuudet ja jaetaan laskentatehtävä pienempiin osatehtäviin. Tiedot riippuvuuksista täytyy tallentaa myöhempää käsittelyä varten. Kaikki osatehtävät eivät välttämättä ole samankokoisia ja kun jako on tehty, kukin osatehtävä on uusi, jakamaton laskennan kohde. Tyypillisesti ohjelmoijan tehtäväksi jää jakaa laskentatehtävät osatehtäviksi. (Grama et al. 2003) Tehtävienjako voidaan tehdä myös staattisesti eli ennen ohjelman ajon aloittamista tai dynaamisesti eli ajonaikaisesti (Rauber ja Gudula 2010). Automatisoitua rinnakkaistamista kuitenkin tutkitaan. Erään tutkimuksen tuloksena on laskenta-arkkitehtuuri joka optimoi peräkkäisestä koodista rinnakkaisesti ajettavaa koodia (Apopei ja Dodd 2012). Toinen tutkimus esittää työkalun, joka erottaa rinnakkaisuuden löytämisen ja optimoinnin sekä esittää ohjelmointirajapinnan (API) tämän toteuttamaan (Raman et al. 2011).

Osatehtävien kokoa ja määrää kutsutaan rakeisuudeksi. Jos osatehtäviä on vähän ja ne ovat suuria, puhutaan karkeasta rakeisuudesta (coarse-grained decomposition), kun taas suurta määrä pieniä osatehtäviä kutsutaan hienojakoiseksi (fine-grained decomposition). Luvussa 4.5 esitellään rinnakkaisten järjestelmien tehokkuuden mittaamista, ja rakeisuudella on tärkeä vaikutus rinnakkaisuusasteeseen. (Grama et al. 2003, Rauber ja Gudula 2010)

Kun jako osatehtäviin on tehty, täytyy osatehtävä jakaa laskentayksiköille. Purkuvaiheessa tallennetut riippuvuustiedot ovat tärkeitä laskentaa järjestettäessä. Tässä työssä ei tarkemmin syvennytä erilaisiin hajotustekniikoihin tai laskennan järjestämiseen, mutta esitellään niiden kantavat ajatukset. Laskennan järjestämien voidaan nähdä kuvauksena, jonka lähtöjoukkona on looginen laskentaongelma ja kohdejoukkona fyysiset laskentayksiköt. Hyvät kuvausalgoritmit pyrkivät hyödyntämään suuren määrän laskentayksiköitä mahdollisimman tehokkaasti - muista riippumattomat osatehtävät eri laskentayksiköille, mahdollisimman moni laskentayksikkö käytössä, paljon viestivät osatehtävät samoille laskentayksiköille. Toisaalta pitäisi pitää laskentayksiköitä vapaana sellaisille tehtäville, joista muut tehtävät ovat riippuvaisia. (Grama et al. 2003, Rauber ja Gudula 2010)

Kuvauksen päämäärät ovat ristiriitaisia, sillä ei ole esimerkiksi joidenkin laskentayksiköiden vapaana pitämien on selvästi ristiriitaista mahdollisimman suuren käyttöasteen saavuttamiseksi. Sopivan tasapainon ja hyvän kuvauksen löytäminen ei ole helppo tehtävä, ja vaikka rinnakkaisuusaste antaa teoreettisen parhaan arvon rinnakkaisuudelle, käytännössä kuvaus määrää, kuinka rinnakkaista laskenta oikeasti on. Onnistunut kuvaus takaa laskennan hyvän skaalautuvuuden. (Grama et al. 2003, Rauber ja Gudula 2010)

4.3.4 Amdahlin laki

Useasti on mainittu, että rinnakkaislaskennan tavoite on laskentatehon lisäys. Rinnakkaislaskennan laskentatehon lisäystä verrattuna peräkkäiseen laskentaan kuvaa Amdahlin laki, joka kuuluu

$$\frac{1}{r_s + \frac{r_p}{n}}, \tag{2}$$

missä $r_s + r_p = 1$ ja r_s kuva ei-rinnakkaistuvaa osaa ohjelmasta, r_p rinnakkaistuvaa osaa ja n laskentayksiköitä (Amdahl 1967). Kaavasta voidaan päätellä, että laskentayksiköiden määrää suurentaessa laskentateho ei kasva äärettömästi.

4.4 Rinnakkaisuuden haasteita

Rinnakkaistaminen tehostaa laskentaa, mutta rinnakkaistaminen tuo uudenlaisia haasteita tietokonejärjestelmien suunniteluun. Mainitut hallintarakenteet tuovat ylimääräisiä kustannuksia laskennan oheen, kuten viestintään liittyviä vaatimuksia. Tietokoneiden tehokkuutta ei määrittele ainoastaan prosessorin tai prosessorien nopeus, vaan esimerkiksi muistin vasteaika ja kaistanleveys asettaa rajoja sille, kuinka paljon laskentaa voidaan suorittaa. Muistin hitautta voidaan kompensoida monilla keinoilla, kuten välimuisteilla (cache), monisäikeisyydellä (multithreading) tai ennakkohauilla (prefetching). Muistin hitaus ei ole yksinomaan rinnakkaislaskennan ongelma, mutta ongelma pahenee hallintakustannuksien ja monien laskentayksiköiden ongelmien kertautuessa. (Grama et al. 2003)

Puhtaasti rinnakkaisuuteen liittyviä ongelmia ovat esimerkiksi tyhjäkäynti (idling) ja turha laskeminen (excess computation). Tyhjäkäynnissä jotkin laskentayksiköt eivät suorita mitään laskentaa. Tämä saattaa johtua esimerkiksi siitä, että hallintajärjestelmä ei ole antanut laskentayksikölle laskettavaa, koska tarpeeksi tehtäviä ei ole. Videokoodauksen yhteydessä mainitut riippuvuudet aiheuttavat myös tyhjäkäyntiä. Tällöin resursseja hukataan. Turhaa laskentaa on esimerkiksi se, että useampi laskentayksikkö laskee samaa asiaa tahoillaan. Tällainen tilanne saattaa syntyä esimerkiksi silloin, kun rinnakkaislaskentaan valittu algoritmi ei ole tehtävään optimaalinen. Tilanne ei ole harvinainen, sillä monet tehokkaat peräkkäiset algoritmit käyttävät aiemmin laskettua tietoa hyväkseen. Laskennan ollessa hajautettuna eri laskentayksiköille aiemmin lasketun tiedon hyödyntäminen on mahdotonta. (Grama et al. 2003)

Laskennan oikeellisuutta ja siihen liittyviä ongelmia sivuttiin luvussa 2.3.

4.5 Rinnakkaisten järjestelmien tehokkuuden mittaaminen

Aiemmin mainittiin Amdahlin laki, joka antaa karkean arvion ohjelman rinnakkaistamisen tuomasta laskentatehon nopeuden kasvusta. Tämä ei kuitenkaan ole ainoa tapa mitata rinnakkaisia järjestelmiä. Muita tapoja ovat esimerkiksi rinnakkaisuuden lisäkustannukset, ohjelman suoritusaika, laskennan tehokkuus, ja laskennan hinta. Kaikkia tuloksia voidaan verrata peräkkäisen ohjelman vastaaviin suorituksiin ja todeta, onko rinnakkaistaminen järkevää. (Grama et al. 2003)

Seuraavassa esitellään joitakin rinnakkaisten järjestelmien tehokkuuden mittareita.

Peräkkäisen ohjelman tapauksessa ohjelman suoritusaika on aika ohjelman suorituksen alkamisesta sen päättymiseen. Rinnakkaisen ohjelman suoritusaika on aika rinnakkaisen ohjelman suorituksen alkamisesta viimeisen laskentayksikön laskennan loppumiseen. Näiden erotuksesta samassa laskentatehtävässä saadaan yksi empiirinen tulos rinnakkaisen järjestelmän tehokkuudesta. (Grama et al. 2003)

Pelkkä empiirinen tulos ei yksin riitä osoittamaan rinnakkaista tai peräkkäistä toteutusta paremmaksi. Amdahlin laki antaa erään arvion laskennan nopeuden kasvusta, mutta rinnakkaiset algoritmit eivät aina käytä samaa menetelmää ongelman ratkaisemiseksi kuin peräkkäiset algoritmit. Teoreettiseen ja empiiriseen vertailuun valitaan yleensä tehokkaimmat algoritmit molemmista luokista. (Grama et al. 2003)

Korkein rinnakkaisuusaste kuvaa, kuinka montaa osatehtävää laskea samanaikaisesti ohjelman ajon aikana. Korkein rinnakkaisuusaste on yleensä osatehtävien määrää pienempi osatehtävien riippuvuussuhteiden johdosta. Korkeinta rinnakkaisuusastetta yleisemmin käytetty rinnakkaisuuden tunnusluku on keskimääräinen rinnakkaisuusaste, joka kertoo, kuinka monta osatehtävää oli laskennassa keskimäärin ohjelman ajon aikana. (Grama et al. 2003, Rauber ja Gudula 2010)

Ihannetilanteessa rinnakkaisen ohjelman tehokkuus olisi 100%, eli jokainen laskentayksikkö olisi koko ajan käytössä mahdollisimman suurella kapasiteetilla. Tehokkuus voidaan lausua yksinkertaisena yhtälönä

$$\frac{S}{p}, \tag{3}$$

missä S on laskennan halutulla tavalla laskettu nopeuden kasvu ja p laskentayksiköiden määrä. (Grama et al. 2003)

5 Videokoodaus ja rinnakkaislaskenta

Tässä luvussa esitellään muutamia uusia ratkaisuja videokoodauksen rinnakkaistamiseen. Luvun tavoite on näyttää, että vaikka rinnakkaislaskenta tuo mukanaan suuren joukon ongelmia, niin sopivalla ongelman rajauksella voidaan rinnakkaislaskennalla saavuttaa hyötyjä videokoodauksen saralla.

5.1 Videokoodauksen rinnakkaistamisen tarpeellisuus

Videokoodaus on aina ollut laskennallisesti vaativa ongelma. Viime vuosien kehitys korkeatarkkuuksista videokuvaa kohti on johtanut siihen, että yksiytimisten tietokoneiden laskentateho alkaa olla riittämätön. Aiemmin ratkaisuna ovat olleet erilaiset multimediaan keskittyneet laskentayksiköt, mutta niiden joustamattomuus ja hinta eivät sovellu nopeasti kehittyvien menetelmien ja jatkuvasti kasvavien laatuvaatimusten tyydyttämiseen. Rinnakkaisohjelmoinnin paradigma siirtää rinnakkaisuuden tietokoneen laitteistoratkaisuista ohjelmistojen puolelle. Laitteistopohjaiset ratkaisut ovat edelleen ohjelmistol-

lisia ratkaisuja tehokkaampia, mutta ero pienenee yleisluontoisten moniydinprosessorien yleistyessä ja rinnakkaisohjelmointimenetelmien kehittyessä. (Choi ja Jang 2012)

Nykypäivänä tehokkaimmat yksiprosessorit pystyvät koodaamaan korkeatarkkuuksista (1080p) videodataa. Kaikilla ja kaikkialla ei ole kuitenkaan mahdollisuutta hyödyntää parhaita saatavilla olevia prosessoreita, joten halvempia ratkaisuja on löydettävä. Samalla videodatan tarkkuus jatkaa kasvamistaan, joten rinnakkaisuus tulee tulevaisuudessa olemaan välttämätöntä niin multimedian esittämiselle kuin tieteellisille videokoodaussovelluksille. On myös turha olla hyödyntämättä lisääntyvää laskentayksiköiden määrää kaikissa tietokoneissa. (Chi et al. 2012, Peng et al. 2012)

5.2 Videokoodauksen rinnakkaistamisen ongelmat

Suurin videodatalle erityinen ongelma ovat luvussa 3.2 esitellyt riippuvuudet. Joitakin ehdotuksia riippuvuuksien karsimiseksi ehdotettiin, mutta ne eivät sovellu sellaisenaan rinnakkaisiin ratkaisuihin. Tässä aliluvussa esitellään tarkemmin riippuvuuksista johtuvia rinnakkaistamisongelmia sekä muita rinnakkaisen videokoodauksen ongelmia.

Rinnakkaisissa laskentayksiköt suorittavat laskentaa itsenäisesti, jolloin toisten laskentayksiköiden tulostan käsiksi pääseminen on vaikeaa ja aikaa vievää. Perinteisissä videokoodausmenetelmissä hyödynnetty ennustaminen johtaa siihen, että sellaisenaan rinnakkaistetuissa perinteisissä menetelmissä on paljon ylimääräisiä kustannuksia laskentayksiköiden välisestä kommunikaatiosta ja niiden välisen synkronoinnin odottamisesta. Toisaalta perinteisissä videokoodausmenetelmissä ruudut on jaettu lohkoihin, jotka ovat toistaan riippumattomat. Lohkojen määrä on kuitenkin liian pieni tehokkaiden, vahvasti rinnakkaisten ratkaisujen toteuttamiseen. (Pieters et al. 2012)

Viestinvälitysparadigmalla toimivien rinnakkaislaskentayksiköiden ongelmaksi muodostuu viestikaistan tai kaistojen leveys. Raaka videodata, erityisesti korkeatarkkuuksinen videodata, vaatii paljon kapasiteettia viestikanavalta. Jos datan määrä ylittää siirtokapasiteetin, niin laskentayksiköiden käyttöaste laskee ja rinnakkaislaskenta hidastuu merkittävästi. Luonteva ratkaisu pullonkaulan purkamiseen olisi kanavan kapasiteetin kasvatus, mutta tämä on useimmiten mahdotonta kustannusten, kanavan saavuttamattomuuden tai tekniikan kehittymättömyyden johdosta. (Li et al. 2012)

5.3 Rinnakkaisia videokoodaustoteutuksia

Rinnakkainen videokoodaus kärsii selvästi siitä, että videokoodausmenetelmiä ei ole alun alkaen suunniteltu rinnakkaislaskentaa silmällä pitäen. Rinnakkaisuutta on alettu vaatia, kun laskennan vaatimukset ovat kasvaneet, mutta standardit ja menetelmät eivät ole kehittyneet yhtä nopeasti. Tässä aliluvussa käsitelläänkin uusia standardeja, tapo-

ja muokata olemassa olevista standardeista helpommin rinnakkaistuvia ja menetelmien optimointia.

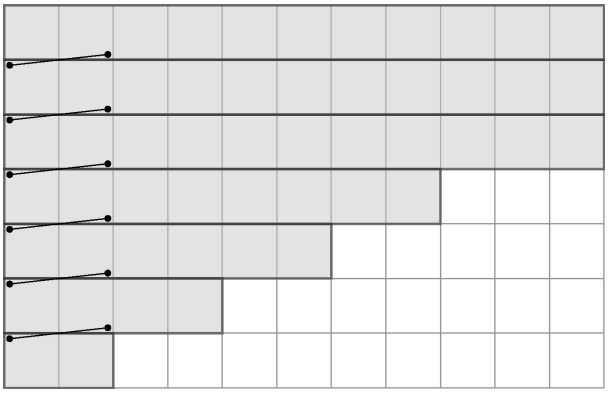
5.3.1 Uudet standardit ja menetelmät

Rinnakkaisten ratkaisujen yleistyessä on perinteisen videokoodausmenetelmien ongelmat alettu ottaa huomioon uusia videokoodausstandardeja suunniteltaessa. Uusien standardien ja menetelmien tavoitteena on korkean tason riippuvuuksien tai matalan tason riippuvuuksien vähentäminen . Korkean tason riippuvuus viittaa enkoodatun bittijonon riippuvuuksiin ja matalan tason riippuvuus yksittäisten työkalujen ja algoritmien riippuvuuksien vähentämistä. (Choi ja Jang 2012) Käsitellään seuraavaksi, miten kehitteillä olevaan HEVC-standardiin (High Efficiency Video Coding) on sisällytetty rinnakkaisuutta ja sille ehdotettua parannusta sekä RVC-CAL-menetelmä, joka tavoittelee helposti monille alustoille käännettäviä rinnakkaisia ratkaisuja. HEVC pyrkii korkean tason riippuvuuksien vähentämiseen ja RVC-CAL matalan tason riippuvuuksien vähentämiseen.

5.3.1.1 HEVC-standardi

HEVC-standardissa on ehdotettu joitakin keinoja rinnakkaisten ratkaisujen pohjaksi. Ehdotettuja keinoja ovat laattapohjainen (tile based) ja aaltorintamapohjainen rinnakkaisuus (Wavefront Parallel Processing, WPP). Laattapohjaisessa lähestymistavassa videodatan ruudut jaetaan muuttuvan kokoisiin laattoihin, jotka ovat toisistaan riippumattomia. Normaalista lohkojaosta tämä poikkeaa laattojen muuttuvalla koolla, siinä missä lohkot ovat aina saman kokoisia. Muuttuvan kokoiset lohkot mahdollistavat samankaltaisten muotojen tai muiden kokonaisuuksien saamisen samaan laattaan, jolloin laatan sisäinen korrelaatio on korkeampi. Ruudunlaajuiset riippuvuussuhteet kuitenkin rikotaan ja pakkaustehokkuus laskee verrattuna peräkkäiseen koodaukseen. Aaltorintamarinnakkaisuudessa riippuvuussuhteet rikotaan ruutujen rivien välillä. Uuden rivin prosessointi alkaa aina edellisen rivin toisen komponentin prosessoinnin jälkeen. Edellisen rivin toista komponenttia käytetään seuraavan rivin ennustamiseen koodaushäviön pienentämiseksi. Kuvassa 3 on kuvattu aaltorintamakoodauksen eteneminen. (Chi et al. 2012)

Aaltorintamarinnakkaisuuden riippuvuudet tarkoittavat sitä, että laskentayksiköitä voi olla käytössä samaan aikaan ruutujen rivien verran. Riippuvuuksista seuraa myös se, että laskentayksiköt eivät voi aloittaa koodausta tai dekoodausta samaan aikaan, mikä aiheuttaa rinnakkaislaskennan tehottomuutta erityisesti suurella määrällä laskentayksiköitä. Chi et al. 2012 ehdottaa tähän ratkaisuksi päällekkäistä aaltorintamaa (Overlapped Wavefront, OWF), jonka kantava idea on, että laskentayksiköt voivat siirtyä laskemaan seuraavaa kuvaa saatuaan oman rivinsä valmiiksi. Tämä asettaa rajoitukset sille, kuinka paljon liikekompensaatiota voidaan käyttää. Kun komponentti tulee koodattavaksi,



Kuva 3: Aaltorintamarinnakkaisen laskennan eteneminen

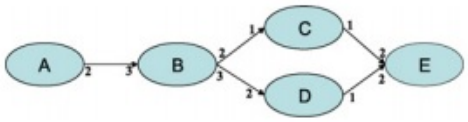
sen ennustamiseen tarvittavat komponentit täytyy olla jo koodattu. Aaltorintaman tapauksessa tämä tarkoittaa ylempiä rivejä. Käytännössä täytyy rajoittaa ruutujenvälistä pystysuuntaista liikekompensaatiota. (Chi et al. 2012)

Chi et al. 2012 esittää, että päällekkäinen aaltorintama tuottaa parhaat tulokset, laattamenetelmä toiseksi parhaat ja tavallinen aaltorintama kolmanneksi parhaat heidän testiympäristössään. Tulos antaa ymmärtää, että ehdotettu päällekkäinen aaltorintama -menetelmä kannattaa ottaa huomioon HEVC-standardia kehittäessä. Huomattavaa on myös, että menetelmät toimivat huomattavasti paremmin rinnakkaisina kuin peräkkäisinä, sillä yhdessä säikeessä ajetut testit tuottivat selvästi huonoimmat tulokset.

5.3.1.2 RVC-CAL-menetelmä

Rinnakkaisen videokoodauksen ongelmia on käsitelty aiemmissa alaluvuissa. Tässä alaluvussa esitellään eräs keino hyödyntää rinnakkaisuutta ja sen soveltamista videokoodaukseen. Keino on RVC-CAL, erityisesti videokoodaukseen kehitetty CAL-ohjelmointikielen toteutus (Cal Actor Language). CALin vahvuus on se, että sillä on kääntäjiä monille alustoille, mukaan lukien moniydinprosessorit. RVC-CALiin on saatavilla myös avoimen lähdekoodin toteutus (Raulet et al. 2013).

CAL on korkean tason toimijakeskeinen tietovuo-ohjelmointikieli (high level actor oriented dataflow programming language). Tietovuo-ohjelmoinnissa ohjelmat mallinnetaan suunnattuina verkkoina, joiden solmuja kutsutaan toimijoiksi. Toimijat kuvaavat mielivaltaisen monimutkaisia laskutoimituksia ja toimijoiden väliset kaaret tiedon liikkumista toimijoiden välillä. Liikkuva tieto abstrahoidaan merkeiksi (token). Toimijat toistuvasti ottavat sisääntulevista kaarista merkkejä, suorittavat laskutoimituksensa ja tuottaa merkkejä ulospäin meneviin kaariin. Kuvan 4 tietovuokaaviossa on viisi toimijaa, A, B, C, D ja E. Kunkin kaaren lähtöpuolella kerrotaan, kuinka monta merkkiä kyseiseen kaareen tuotetaan ja päättymispuolella kuinka monta merkkiä kyseinen toimija ottaa vastaan. (Gu et al. 2009)



Kuva 4: Yksinkertainen esimerkki tietovuokaaviosta

Tietovuomalli perusmuodossaan ei määrää mitään aikarajoitteita toimijoiden laskennalle. CALissa jokainen toimija on itsenäinen komponentti eivätkä muut toimijat voi vaikuttaa sen tilaan (muuten kuin merkkejä lähettämällä). CALin toimijat on määritelty niiden toimintojen (action) perusteella. Jokainen toiminto määrittää miten toimijan sisäinen tila muuttuu. Toimintoja suoritetaan (fire) riippuen merkkien sisältämästä datasta tai toimijan sisäisistä tiedoista. Toiminnot suoritetaan peräkkäin, mutta missä toimijoiden suoritusjärjestystä ei ole määrätty. Ominaisuuksiensa johdosta CAL sopii hyvin kuvaamaan rinnakkaisia järjestelmiä ja rinnakkaisia algoritmeja. (Gu et al. 2009)

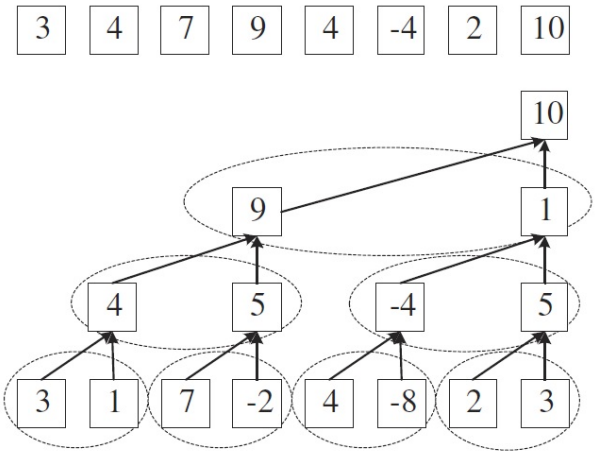
RVC (Reconficurable Video Coding) on MPEG-ryhmän (Moving Picture Experts Group) kehittämä standardi rinnakkaisten videokoodausmenetelmien suunnitteluun. Standardin tavoitteena on olla yhtenäinen, korkean tason suunnittelutyökalu. Se määrittelee laskentayksiköt toiminnallisiksi yksiköiksi (functional unit) ja toiminnallisten yksiköiden väliset yhteydet laskentayksiköiden välisiksi datapoluiksi. Nämä vuorostaan kuvautuvat CALin toimijoiksi ja kuvaajien kaariksi. CALin käyttö väliesityksenä mahdollistaa kääntämisen monenlaisille alustoille ja jopa suoran syntetisoinnin laitteistoksi. (Gu et al. 2009)

5.3.2 Videodatan riippuvuuksien ja synkronoinnin tarpeen vähentäminen

Videodatan riippuvuudet aiheuttavat sen, että rinnakkaistaminen on vaikeaa tai että suurin osa rinnakkaislaskennasta kulutetaan synkronointiin eli käytännössä tyhjäkäyntiin. Tässä aliluvussa esitellään kaksi lähestymistapaa, joilla riippuvuuksien määrää ja niistä seuraavia ylimääräisiä kustannuksia on pyritty vähentämään. Edellisessä aliluvussa esitellyt aaltorintamamenetelmät ja laattamenetelmä sopisivat tähänkin lukuun, mutta seuraavassa esitettyjä menetelmiä ei ole ehdotettu minkään standardin osaksi.

Aaltorintamamenetelmää muistuttava rinnakkaisuutta vähentävä keino on riveittäin suoritettava koodaus (Line-By-Line Coding, LBLC). Aaltorintaman tavoin rivikoodaus käsittelee videodatan ruutujen rivejä. Laskentayksiköille jaetaan kuitenkin rivien sisään rivien osia. Näin rivi koodataan rinnakkaisesti ja koodattua riviä voidaan käyttää seuraavan rivin ennustamiseen. Rikottu riippuvuus on siis rivien vierekkäisten osien välinen. Synkronoinnin tarvetta on vähennetty staattisella aikataulutuksella. Saman rivin osat jaetaan eri laskentayksiköille ja eri rivien vastaavan osat samoille laskentayksiköille. (Peng et al. 2012)

Esitellyt menetelmät videokoodauksen rinnakkaistamiseen ovat keskittyneet videodatan rinnakkaisuuden rikkomiseen. On ehdotettu myös ennustamisen tehokkaampaa rinnakkaistamista ja näin synkronoinnin huomattavaa vähenemistä. Pieters et al. 2012 esittää ennustamiseen tehokkaasti rinnakkaistuvan rekursiiviseen kaksinkertaistamiseen (recursive doubling) perustuvaa etuliitesummaa (prefix sum) (Blelloch 1989). Menetelmä saa syötteekseen perusennustuksen sekä jäännösarvot, joiden avulla näytteiden arvot on tallennettu, kuten normaalissakin ennustamisessa. Etuliitesumma lasketaan rinnakkaisilla laskentayksiköillä, eikä edellisten näytteiden koodausta tarvitse odottaa. Etuliitesumma etenee puumaisesti, minkä johdosta siinä on korkeintaan logaritminen määrä synkronointipisteitä näytteiden määrään nähden. Kuva 5 esittelee etuliitesumman etenemisen. Puun tasot ovat mahdollisia synkronointipisteitä. (Pieters et al. 2012)



Kuva 5: Etuliitesumma johtaa näytteen arvot jäännösarvoista

Molemmat tässä kappaleessa esitellyt videokoodauksen rinnakkaistamismetodit ovat osoittautuneet testeissä tehokkaiksi. Rivikoodaus nopeuttaa videokoodausta lähes lineaarisesti laskentayksiköiden määrään nähden. Parhaiten se toimii häviöttömillä ja lähes häviöttömillä koodausmenetelmillä, mikä on sopivaa, sillä sellaiset menetelmät tarvitsevat tehokkuutta ollakseen käytännöllisiä. (Peng et al. 2012) Paremmin rinnakkaistuvat ennustaminen nopeutti H.264-standardilla enkoodatun videon dekoodaamista 2.2-7.9 kertaisesti aaltorintamamenetelmään verrattuna sekä 2.2 kertaista nopeutusta normaaliin ennustamiseen 2160p -tarkkuuksisella videolla. (Pieters et al. 2012)

5.3.3 Optimointi

Uusien menetelmien kehittämisen lisäksi voidaan rinnakkaisuutta tuoda olemassa oleviin menetelmiin niitä muuttamatta. Tällöin on tärkeää löytää kulloinkin ratkaistavissa olevaan ongelmaan sopiva rinnakkainen lähestymistapa. Aiemmin esiteltyjä teorioita rinnakkaisuuden mittaamisesta voidaan käyttää erilaisten ratkaisujen analysointiin, mutta

todelliset tulokset saadaan vasta käytännön testeillä. (Li et al. 2012)

Li et al. 2012 on tutkinut H.264-standardin mukaisen videokoodauksen optimointia rinnakkaiseen järjestelmään. Tutkimus ei esitä uusia algoritmeja videokoodauksen ongelmien ratkaisemiseen vaan tutkii erilaisia rinnakkaislaskentaan vaikuttavia parametreja, kuten laskentayksiköiden määrää ja konekielisen tason optimoinnin hyödyntämistä. Suurin haitta rinnakkaislaskennan tehokkuudelle tutkimuksessa oli laskentayksiköiden välisen viestinnän hitaus. (Li et al. 2012)

Tutkimuksen tulokset osoittavat, että eräänlainen raaka rinnakkaistaminen tehostaa videokoodausta hieman. Koska H.264 ei ole suunniteltu rinnakkaiseksi, sen rinnakkainen tehonlisäys ei kasva lineaarisesti laskentayksiköiden määrää kasvattaessa. Samoin yksittäisen laskentayksikön tehokkuus laskee huomattavasi laskentayksiköiden määrää lisätessä erityisesti hitaammilla tietoliikenneyhteyksillä. Paras tulos saavutettiin esijakamalla koodattava videodata laskentayksiköille, jolloin sen siirtämien ei enää aiheuttanut pullonkaulaa. Reaaliaikaiseen koodaukseen tämä ei kuitenkaan sovi. (Li et al. 2012)

5.3.4 Rinnakkaislaskennan tuomat hyödyt suorituskykyyn

Videodataa esittelevässä luvussa esiteltiin lyhyesti erilaisia mittareita videodatan laadulle ja videokoodauksen laadulle. Taulukkoon 1 erilaisten rinnakkaisten videokoodausmenetelmien tuomia hyötyjä videokoodaukseen.

Taulukko 1: Empiirisiä tuloksia videokoodauksen rinnakkaistamisesta

Menetelmä	Mitattu suure	Saavutettu hyöty	Lähde
Optimointi	Koodausnopeus	Viestikapasiteetin riittäessä lineaarinen laskentatehon kasvu laskentayksiköihin nähden	Li et al. 2012
Rinnakkainen ennustaminen	Dekoodausnopeus	18.3-21.5 kertainen nopeus verrattuna aaltorintamarinnakkaisuuteen	Pieters et al. 2012
Rinnakkaisuuksien rikkominen	Pakkaussuhde	Parhaimmillaan 14% parempi pakkaussuhde verrattuna peräkkäiseen implementaatioon	Peng et al. 2012

Rinnakkaisuuksien rikkominen	Skaalautuvuus	Lähes lineaarinen skaalautuvuus laskentayksiköiden määrään nähden	Peng et al. 2012
Rinnakkaisuuksien rikkominen	PSNR	Korkeampi PSNR kuin peräkkäisellä menetelmällä erityisesti korkealaatuisessa videossa	Peng et al. 2012

5.4 Erilaiset kiihdytysalustat

Yksi rinnakkaislaskennan haasteista on siinä, että rinnakkaiset järjestelmät ovat hyvin erilaisia ja yhteisiä tekniikoita ja toteutuksia niille on vaikea löytää. Tässä aliluvussa käsitellään muutamia erilaisia kiihdytysalustoja ja niiden sopivuutta rinnakkaiseen videokoodaukseen.

Kaikenlaiset tietokoneet aina älypuhelimista tehokkaisiin palvelimiin alkavat olla nykypäivänä moniydinprosessoreilla varustettuja (Choi ja Jang 2012). Tämän johdosta tätä kiihdytysalustaa voidaankin pitää tärkeänä erityisesti multimediasovelluksille, joissa videokoodaus on yleinen ja tärkeä operaatio. Esitellyistä tutkimuksista Chi et al. 2012 käytti kiihdytysalustanaan 12-ytimistä prosessoria, kun taas Li et al. 2012 käytti klusteria moniytimisiä prosessoreita. Esitetyt tulokset osoittavat, että videokoodaus voidaan onnistuneesti rinnakkaistaa moniydinprosessoreille, mikä ennustaa hyvää tulevaisuuden rinnakkaisille videokoodaustoteutuksille kuluttajatason tietokoneissa.

Klusteri tarkoittaa useampaa yhteen liitettyä tietokonetta, jotka on valjastettu suorittamaan samaa tehtävää. Klusterit ovat tyypillisiä tieteelliselle ja todella vaativalle laskennalle, joka vaatii yksittäisiä koneita suuremman laskentatehon. Esitellyistä tutkimuksista Li et al. 2012 esitteli kaksi erilaista klusteria, toisen Windows-käyttöjärjestelmän PC:illä ja toisen Linux-käyttöjärjestelmällä toimivilla koneilla. Klusterit tuovat rinnakkaislaskentaan omat haasteensa, kuten klusterin laskentayksiköiden heterogeenisyyden, laskentayksiköiden välisen viestinnän ja esimerkiksi klusterin hinnan. Multimedian toistamiseen klusterit ovat liian tehokkaita, mutta esimerkiksi konenäköön tai muihin tieteellisiin sovelluksiin klustereista ja rinnakkaisesta videokoodauksesta voi olla hyötyä.

Grafiikkaprosessorit ovat nykypäivänä käytännössä kaikista kuluttajatietokoneista löytyviä prosessoreita, jotka ovat erikoistuneet grafiikan piirtämiseen näyttölaitteille. Rinnakkaiseksi kiihdytysalustaksi grafiikkaprosessorit sopivat niiden rinnakkaisen luonteen puolesta. Nykyaikaisissa grafiikkaprosessoreissa on satoja laskentayksiköitä, joiden kello-

taajuus ei ole suuri, mutta riittävä nopeaan rinnakkaiseen laskentaan. Esitellyistä tutkimuksista Pieters et al. 2012 käytti kiihdytysalustanaan grafiikkaprosessoria.

Grafiikkaprosessorit sopivat hyvin videokoodauksen rinnakkaislaskenta-alustoiksi, mikä on luontevaa, piirretäänhän dekoodattu videokuva näyttölaitteelle. Joissain uusissa näytönohjaimissa on sisäänrakennettuna tuki ja valmiit ohjelmistoratkaisut joidenkin videokoodausstandardien dekoodaamista varten, joten omia ratkaisuja tätä varten ei välttämättä edes tarvitse tehdä (NVidia 2013).

6 Yhteenveto

Rinnakkaiset videokoodausratkaisut ovat tulevaisuudessa välttämättömiä videolaadun parantuessa ja rinnakkaisten laskentayksiköiden yleistyessä. Tässä työssä esiteltiin videokoodauksen ja rinnakkaislaskennan perusteita ongelmineen ja mahdollisuuksineen sekä videokoodauksen ja rinnakkaislaskennan yhdistämistä. Mahdollisuuksistaan huolimatta rinnakkaislaskennalla on haasteensa, joihin ei ole vielä yhtenäisiä ratkaisuja. Olemassa olevia videokoodausmenetelmiä ei ole kehitetty rinnakkaislaskentaa silmällä pitäen, joten niiden rinnakkaistaminen on erityisen vaikeaa.

Ongelmista huolimatta rinnakkaislaskennasta on tutkimusten mukaan hyötyä videokoodaukselle. Rinnakkaislaskenta nopeuttaa videokoodausta ja paikoitellen parantaa jopa pakkaussuhdetta. Jotta rinnakkaiset ratkaisut videokoodauksen saralla yleistyisivät, on ne otettava huomioon uusia standardeja määriteltäessä. Määrittelyjä vaikeuttaa rinnakkaisten kiihdytysalustojen monenkirjavuus. Askelia oikeaan suuntaan on otettu, sillä uusissa videokoodausstandardeissa rinnakkaisuus on otettu huomioon jo suunnitteluvaiheessa ja joissain uusissa grafiikkaprosessoreissa on sisäänrakennettuna videodekooderi.

Tässä työssä keskityttiin videokoodauksen ja rinnakkaislaskennan perusteisiin sekä esiteltiin joitakin ratkaisuja rinnakkaiseen videokoodaukseen keskittyen näkyvään laskentatehon kasvuun. Videokoodauksen, rinnakkaislaskennan ja rinnakkaisen videokoodauksen saralla tehdään aktiivista tutkimusta. Esimerkiksi varsinainen rinnakkaislaskennan tutkimus on vahvasti kääntäjien ja käyttöjärjestelmien tutkimusta, joihin ei tässä työssä syvennytty lainkaan. Rinnakkaislaskennan laitteistojen ja ohjelmistojen tutkimus on käytännön sovelluksien kannalta tärkeää, mutta yksityiskohdat jätettiin tämän työn ulkopuolelle. Työ toimii johdantona rinnakkaiseen videokoodaukseen ja ohjaa syvällisemmän tutkimuksen pariin.

Jatkotutkimuksena olisi tärkeää selvittää, millainen rinnakkainen lähestymistapa videokoodaukselle on helpointa toteuttaa niin, että se palvelee mahdollisimman monia kiihdytysalustoja. RVC-CAL-menetelmän esittämä työkalu, joka mahdollistaa rinnakkaisen koodekin kääntämisen monenlaisille alustoille on mielenkiintoinen ja tärkeä sovellus.

Lähteet

Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings*, sivut 483–485, 1967.

Marios C. Angelides ja Harry Agius. *Handbook of MPEG Applications : Standards in Practice*. Wiley, Chichester, West Sussex, United Kingdom, 2011. ISBN 978-0-470-75507-0 (nid.), 978-0-470-97458-2 (elektr.). 551 s.

B. Apopei ja T.J. Dodd. Automatic parallelisation for lti mimo state space systems using fpgas. an optimisation for cost &; performance. *Journal of Parallel and Distributed Computing*, 72(8):990 – 1007, 2012. ISSN 0743-7315. doi: 10.1016/j.jpdc.2012.04.009. URL `http://www.sciencedirect.com/science/article/pii/S0743731512001050`.

M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Pearson, Harlow, Essex, United Kingdom, toinen painos, 2006. ISBN 978-0-321-31283-9. 361 s.

G.E. Blelloch. Scans as primitive parallel operations. *Computers, IEEE Transactions on*, 38(11):1526–1538, 1989. ISSN 0018-9340. doi: 10.1109/12.42122.

Chi Ching Chi, Mauricio Alvarez-Mesa, Ben Juurlink, Gordon Clare, Felix Henry ja Thomas Schriel. Parallel scalability and efficiency of hevc parallelization approaches. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1827–1838, joulukuu 2012.

Kiho Choi ja Euee S. Jang. Leveraging parallel computing in modern video coding standards. *IEEE Computer Society*, 12(0):7–11, heinäkuu 2012.

Cisco. Ciscon ennustus mobiilidatan kehityksestä. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html`, tammikuu 2013.

Ke-Lin Du ja M. N. S. Swamy. *Wireless Communication Systems - From RF Subsystems to 4G Enabling Technologies*. Cambridge University Press, New York, USA, 2010. ISBN 9780511716898 (elektr.). 551 s.

Ananth Grama, Anshul Gupta, George Karypis ja Vipin Kumar. *Introduction to Parallel Computing*. Pearson, Harlow, Essex, United Kingdom, toinen painos, 2003. ISBN 0-201-64865-2. 636 s.

Ruirui Gu, J.W. Janneck, S.S. Bhattacharyya, M. Raulet, M. Wipliez ja W. Plishker. Exploring the concurrency of an mpeg rvc decoder based on dataflow program analysis. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(11):1646–1657, marraskuu 2009. ISSN 1051-8215. doi: 10.1109/TCSVT.2009.2031517.

Jong-Seok Lee, Francesca De Simone ja Touradj Ebrahimi. Efficient video coding based on audio-visual focus of attention. *Journal of Visual Communication and Image Representation*, 22(8):704 – 711, 2011. ISSN 1047-3203. doi: 10.1016/j.jvcir.2010.11.002. URL http://www.sciencedirect.com/science/article/pii/S104732031000146X. Emerging Techniques for High Performance Video Coding.

Dongmei Li, Shuai Peng ja Zhaohui Li. Design and optimization of high efficiency parallel video coding system. *Fifth International Joint Conference on Computational Sciences and Optimization*, sivut 592–596, 2012.

Ke Liao, Shiguo Lian, Zhichuan Guo ja Jinlin Wang. Effcient information hiding in h.264/avc video coding. *Journal of Visual Communication and Image Representation*, 22(8):704–711, kesäkuu 2011.

Hans Meuer, Erich Strohmaier, Jack Dongarra ja Horst Simon. Listaus maailman supertietokoneista. http://www.top500.org/, maaliskuu 2013.

A. Mitra. *Digital Video: Moving Images and Computers*. Digital world. Facts On File, Incorporated, 2010. ISBN 9781438134611. URL http://books.google.fi/books?id=6M86Q9_1oMAC.

Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38 (8), huhtikuu 1965.

Miquel Mujal ja Lynn R. Kirlin. Compression enhancement of video motion of mouth region using joint audio and video coding. *Fifth IEEE Southwest Symposium on Image Analysis and Interpretation*, 2002.

NVidia. Nvidian kehittäjien verkkosivut. https://developer.nvidia.com/nvidia-video-codec-sdk, huhtikuu 2013.

Hyungsuk Oh ja Wonha Kim. Video processing for human perceptual visual quality-oriented video coding. *Image Processing, IEEE Transactions on*, 22(4):1526–1535, 2013. ISSN 1057-7149. doi: 10.1109/TIP.2012.2233485.

Xiulian Peng, Jizheng Xu, You Zhouu ja Feng Wu. Highly parallel line-based image coding for many cores. *IEEE Transactions on Image Processing*, 21(1):196–206, tammikuu 2012.

Bart Pieters, Charles-Frederik Hollemeersch, Jan De Cock, Peter Lambert ja Rik Van de Walle. Data-parallel intra decoding for block-based image and video coding on massively parallel architectures. *Signal Processing: Image Communication*, 27(3):220 – 237, 2012. ISSN 0923-5965. doi: 10.1016/j.image.2012.01.001. URL http://www.sciencedirect.com/science/article/pii/S0923596512000021.

Xiaoin Qi, Mianshu Chen ja Hexin Chen. *A CAVLC Embedded Method for Audio-Video Synchronization Coding Based on H.264*. IEEE, 2011. ISBN 9781612847719.

Arun Raman, Kim Hanjun, Taewood Oh, Jae W. Lee, David I. August, Mary Hall ja David Padua. Parallelism orchestration using dope: the degree of parallelism executive. *32nd ACM SIGPLAN conference*, sivut 26–37, 2011.

Thomas Rauber ja Rünger Gudula. *Parallel Programming: For Multicore and Cluster Systems*. Springer, Berlin, Germany, 2010. ISBN 978-3-642-04817-3. 455.

Mickael Raulet, Matthieu Wipilez, Jean-Francois Nezan ja Marco Mattavelli. Orcc-ohjelmiston kotisivu. `http://orcc.sourceforge.net/`, maaliskuu 2013.

Iain Richardson. *H.264 Advanced Video Compression Standard*. Wiley, Chichester, West Sussex, United Kingdom, toinen painos, 2010. ISBN 978-0-470-51692-8 (nid.), 978-0-470-98928-9 (elektr.). 348 s.

A. Vajda. *Programming Many-core Chips*. Springer US, 2011. ISBN 9781441997395. URL `http://books.google.fi/books?id=pSxa\_anfiG0C`.

VideoLan. x264-ohjelmiston kotisivut. `http://www.videolan.org/developers/x264.html`, maaliskuu 2013.

YouTube. Youtube-statistiikkaa. `http://www.youtube.com/t/press_statistics`, tammikuu 2013.