TAMPEREEN TEKNILLINEN YLIOPISTO

VILLE LAATU

PAKATUN VIDEOVUON PURKAMINEN GRAFIIKKAPROSESSORIN AVULLA

Diplomityö

Tarkastaja: Timo Daniel Hämäläinen
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 1. Lokakuuta 2018

TIIVISTELMÄ

VILLE LAATU: Pakatun videovuon purkaminen grafiikkaprosessorin avulla
Tampereen teknillinen yliopisto
Diplomityö, 48 sivua
Marraskuu 2018
Tietotekniikan Diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Timo Daniel Hämäläinen,

Avainsanat: GPU, Videovuo, Grafiikkaohjelmointi, DirectX, VLC

Tämän diplomityön tarkoituksena on parantaa Liikenneviraston uuden integroidun tieliikenteen ohjausjärjestelmän suorituskykyä yhdessä sen osa-alueista. Järjestelmä on hyvin laaja ja sen kehittäminen on aloitettu jo vuonna 2013 moniasiakasprojektina. Järjestelmän tarkoituksena oli sulattaa aiemmin käytössä olevat yli 40 erilaista järjestelmää yhdeksi suureksi kokonaisuudeksi. Kyseinen järjestelmä on ollut tuotantokäytössä jo vuodesta 2015 lähtien, mutta laajuutensa myötä se on vieläkin aktiivisessa kehitysvaiheessa.

Työssä tarkasteltava kokonaisuus liittyy Suomen tieliikenteen keli- ja tiekameroihin, joista järjestelmään saadaan reaaliaikaista videokuvaa. Nykyään teknologian kehittyessä, myös näitä kameroita on alettu uusimaan, jolloin niistä saadaan entistäkin tarkempaa videokuvaa toistettavaksi. Tällä on haittapuolensa, sillä parempilaatuisen kuvan purkamisesta katsottavaan muotoon syntyy suurempi kuormitus.

Aiemmin näitä videovoita on toistettu VLC multimediasoittimen avulla, joka tarjoaa mahdollisuuden laitteistokiihdytyksen käyttämiseen, eli videovuon purkamiseen grafiikkaprosessoria hyväksikäyttäen. Ongelmana on kuitenkin se, että VLC ei mahdollista grafiikkaprosessorin valintaa itse, vaan oletuksena se valitsee aina purkamiseen sen grafiikkaprosessorin, joka on liitettynä näyttöpäätteeseen. Työn tarkoituksena onkin löytää keino, kuinka tätä kuormaa voitaisiin jatkossa jakaa usealle grafiikkaprosessorille suoritettavaksi.

Ratkaisu tähän ongelmaan löytyi DirectX ohjelmointirajapinnasta, jonka avulla Windows käyttöjärjestelmän alaisuudessa pyörivää laitteistoa voidaan käsitellä. Ohjelmointirajapinta mahdollistaa kaikkien grafiikkaprosessoreiden tunnistamisen sekä enumeroinnin, jotka ovat kytkettynä alla olevaan laitteistoon. Näiden toimintojen avulla voidaan aina valita haluttu grafiikkaprosessori, jolla videovuo tullaan purkamaan.

ABSTRACT

VILLE LAATU: Decoding a compressed video stream using the graphics processor
Tampere University of Technology
Master of Science Thesis, 48 pages
November 2018
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Timo Daniel Hämäläinen

Keywords: GPU, Video stream, Graphics programming, DirectX, VLC

This thesis explores improving the performance of a newly developed, Finnish Transport Agency's integrated road traffic control system in one of its current operational areas. The system in context, is very broad and been in development since 2013 resulting from a multi-client project. The purpose of this new system was to merge more than 40 different existing systems into one larger entity. This system has been in production since 2015, but due its broad constraints, is still currently within the active development phase.

The work involves Finnish weather and road cameras, which in turn provide real-time video streams into the system. Presently, with technological advancements, these cameras have also undergone revision, making them able to offer more accurate video streams. This has its drawbacks, because the decoding process for the higher quality image will result into a higher load onto the system.

Previously, these video streams has been played back using the VLC multimedia player. VLC provides an opportunity to utilize the hardware acceleration, meaning that the received video stream could be decoded by taking advantage of the graphics processor. However, the problem lies where VLC does not allow selection of the graphics processor itself, instead by default it always uses the one, which is connected to the monitor. So the purpose of this work is to develop a solution to share this extra load across multiple graphics processors.

The solution for the problem is the DirectX programming interface that allows the programmer to manage the hardware underlying the Windows operating system. This programming interface enables the identification and enumeration of the all graphics processors that are connected to the hardware. By using these functions, it is always possible to choose the desired graphics processor in use, which will decode the video stream.

ALKUSANAT

Tämä diplomityö on toteutettu Liikennevirastolle liittyen uuteen tieliikenteen integroituun ohjausjärjestelmään. Yhtenä projektin teknisen puolen toteuttajista toimii Bitwise Oy, jonka kautta tämän diplomityön toteuttaminen mahdollistui. Haluan kiittää molempia osapuolia tämän diplomityöaiheen tarjoamisesta. Lisäksi haluan kiittää ohjaajaani Timo Daniel Hämäläistä sekä äitiäni, jotka molemmat osaltaan auttoivat työn toteuttamisessa monin tavoin.

Tampereella, 13.11.2018

Ville Laatu

SISÄLLYS

1 JOHDANTO

Tämän diplomityön tarkoituksena on selvittää, kuinka usean pakatun videovuon purkamisesta syntyvää kuormaa voidaan jakaa usean GPU:n (Graphical Processing Unit) kesken. Videovuon esittämiseen käytetään VLC nimistä multimediasoitinta, jolla videovuo puretaan hyödyntäen GPU laitteistokiihdytystä. Työn toteuttamishetkellä VLC:llä ei kuitenkaan ole olemassa tukea sille, että videovuon purkamiseen voitaisiin valita haluttu GPU, vaan VLC valitsee GPU:n automaattisesti sen mukaan, että mikä GPU on kytkettynä näyttöpäätteeseen. VLC kuitenkin on open-source -ohjelmisto sekä lisensoitu GPL-standardin mukaan. Tämä mahdollistaa lähdekoodin muuttamisen omien tarpeiden mukaisesti.

Kuormituksen tasapainottamisella tavoitellaan sitä, että järjestelmä, johon kyseinen ominaisuus tullaan liittämään, mahdollistaisi useamman videovuon katselemisen samanaikaisesti. Tämä muutostarve järjestelmään johtuu siitä, että nykyään Suomen tieliikenteen keli- ja tiekameroista tuleva pakattu videovuo on osassa tapauksissa laadultaan HD (High Definition) -tasoista. Tälläisen korkealaatuisen videovuon purkamisesta taas syntyy suuri kuormitus. Tämä siis nykyisellään estää useamman kameran yhtäaikaisen katselemisen.

Työssä perehdytään ensiksi teoriaan aiheen ympärillä ja lopuksi kehitetään ratkaisu, joka mahdollistaa kuormituksen tasapainottamisen. Matkan varrella käydään läpi, miten VLC voidaan kääntää sen lähdekoodeista ristikäännöksenä ja tämän jälkeen tutustutaan muutoksiin, joita järjestelmään täytyy toteuttaa käyttöliittymän puolelle, sekä VLC:n lähdekoodeihin. Lopuksi vielä käydään läpi tuloksia ja näitä vertaillaan erilaisilla järjestelmä versioilla. Viimeisenä on vielä yhteenveto koko työstä.

2 LAITTEISTOKIIHDYTYS

Tietotekniikassa käsiteltävän datan määrä on kasvanut tasaista vauhtia, eikä kasvulle näy loppua. Dataa kerätään ja käytetään jatkuvasti uusiin käyttötarkoituksiin. Usein ihmiset eivät edes ole tietoisia siitä, että heidän käyttäytymisestään kerätään tietoja. Esimerkki tälläisestä tapauksesta on heidän kauppakäyttäytymisensä. Tälläistä dataa yritykset voivat hyödyntää tuotteidensa sijoittelemiseen myymälässä. Tämä takia datan käsitteleminen on näiden uusien keräysmenetelmien ja tietomäärien kasvamisen vuoksi muuttunut paljon haasteellisemmaksi. Tätä varten on kuitenkin jo kehitetty ja jatkuvasti kehitetään uusiin käyttötarkoituksiin sopivia laitteita. Näitä yhteen käyttötarkoitukseen soveltuvia laitteita on suunniteltu, koska yleiskäyttöisen prosessorin (CPU) laskentateho ei enää riitä kaikkiin käyttökohteisiin. Useissa käyttötapauksissa voi olla kriittisiä aikarajoja, joihin laitteiden on yllettävä tavalla tai toisella.

Tällaisia kehitettyjä erikoislaitteita tai tietokoneen osia ovat mm. grafiikkaprosessori (GPU), jota käytetään videon käsittelemiseen ja 3D-grafiikan esittämiseen. Sitten on salauskiihdytin, joka on suunniteltu suorittamaan laskennallisesti haastavia kryptografisia laskuja. Lisäksi on ohjelmoitava mikropiiri FPGA (Field-programmable gate array) ja sovelluskohtainen mikropiiri ASIC (Application Specific Integrated Circuit). Molemmat mikropiirit ova käytössä useissa sulautetuissa järjestelmissä aina lentotekniikasta lääketekniikkaan. Lisäksi tekoälyn kanssa työskentelemistä varten on kehitetty omia AI (Artificial Intelligence) -kiihdytimiä, jotka on suunniteltu suorittamaan hyvin dataintensiivista laskentaa. AI-kiihdyttimiä käytetään mm. neuroverkkojen, koneoppimisen, konenäön tai robotiikan parissa.

Useiden erityislaitteiden kehityksen myötä myös useat ohjelmat ja alustat tarjoavat mahdollisuuden siihen, että näitä CPU:n ulkopuolisia laitteita voidaan hyödyntää ohjelman ajamiseen kokonaan tai osittain. Yleensä näissä tapauksissa ohjelmassa tarvittava laskentateho tai datan käsittelemisen haasteellisuus on hieman normaalista poikkeavaa. Lisäksi ohjelmalle on mahdollisesti asetettu joitain suorituskykyrajoituksia, jotka voivat olla tietyissä käyttökohteissa hyvinkin kriittisiä.

Ulkoisen erityislaitteen käyttämisen hyötynä suoritustehon kasvattamisen lisäksi on se, että kun suoritus siirretään ulkoisen laitteen tehtäväksi, vähentyy CPU:n kuormitus, jolloin sitä voidaan hyödyntää muualla. Käyttöjärjestelmät käyttävät esimerkiksi CPU:ta useisiin tarkoituksiin, kuten I/O-operaatioiden lukemiseen, keskeytyksiin, muistin- sekä

prosessienhallintaan. Tällöin vähäinen kuormitus nopeuttaa käyttöjärjestelmällä suoritettavia operaatioita selvästi. Lisäksi oikeanlaisella tehtävälle soveltuvalla laitteistolla virrankulutus on yleensä huomattavasti pienempää.

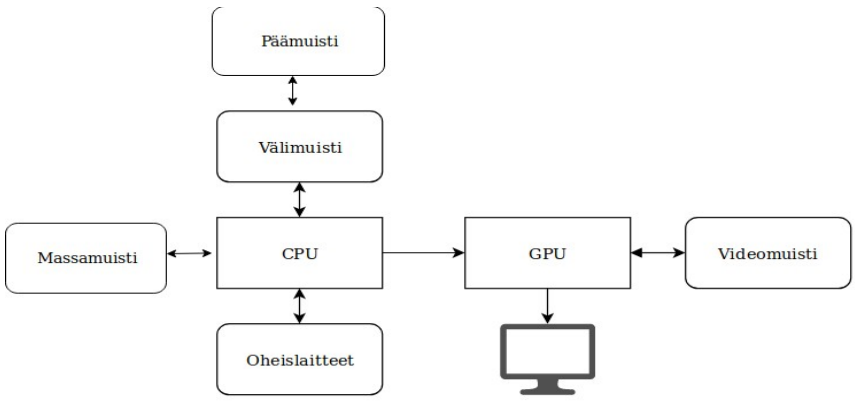
2.1 Grafiikkaprosessori (GPU)

Grafiikkaprosessorin avulla voidaan saavuttaa parempi suorituskyky esimerkiksi videonkäsittelyssä, peligrafiikan esittämisessä ja multimedian filtteröinnissä. Tämä johtuu siitä, että GPU:lla on CPU:n verrattuna moninkertainen määrä ytimiä, joilla se suorittaa laskentaa. Tällöin laskentaa voidaan jakaa useille ytimille samanaikaisesti suoritettavaksi. Vastapainona nämä ytimet ovat kuitenkin CPU:n vastaavia huomattavasti hitaampia.

Ohjelman ajaminen GPU:lla on kannattavaa tilanteissa, joissa sen suorittamista voidaan jakaa useille ytimille. Lisäksi ohjelman tekemät operaatiot eivät saisi olla liian raskaita, jotta heikompitehoiset ytimet selviävät niistä kohtuullisessa ajassa. Tyypillisiä nopeita operaatioita ovat yleisesti käytetyt aritmeettiset operaatiot: lisäys, vähennys, kertominen ja jakaminen. Hitaita konekäskyoperaatioita taas ovat hypyt ja haarautumiset.

2.1.1 Suorituksen siirtäminen GPU:lle

GPU:lla on monta laskentaydintä operaatioiden suorittamiseen. Lisäksi sillä on oma muistilohkonsa, jota kutsutaan videomuistiksi. Muistialueelle tyypillisesti säilötään ytimillä suoritettujen operaatioiden tuloksia, ylläpidetään tietoa sen hetkisestä näytöllä esitettävästä kuvasta, sekä mahdollisesti seuraavan kuvan esittämiseen tarvittavia lisätietoja. Videomuisti on yleensä erillinen fyysinen laite ja se on kytketty näytönohjaimeen emolevyn sijaan. Tällöin GPU:lla on nopea ja helppo pääsy laitteeseen [1]. Alla olevassa kuvassa 1 on nähtävissä tietokoneen arkkitehtuuri ja kuinka tieto esitetään käyttäjälle.



Kuva 1: *GPU osana tietokonetta.*

Ensin CPU vastaanottaa datan joko välimuistista tai massamuistista. Datassa voi ilmetä, että se halutaan ajaa ulkopuolisella laitteella CPU:n sijaan. Tällöin CPU siirtää suoritusvastuun GPU:lle siten, että se lähettää muistista saamansa raakadatan, ehkä hieman modifioiden sitä, jotta GPU pystyy tulkitsemaan dataa. Lähetetystä datasta ilmenee, kuinka sitä tulisi käsitellä, missä se tulisi esittää ja tarvitseeko se palauttaa takaisin CPU:lle käsittelyn jälkeen.

Vastaanotettuaan datan, GPU prosessoi sen halutunlaiseksi, jos data on sellaista, joka halutaan esittää suoraan käyttäjälle. Tällöin GPU samanaikaisesti tallentaa sen hetkistä näytettävää tietoa videomuistiin ja esittää sen käyttäjälle monitorilla. Prosessoitu data ei siis välttämättä kierrä takaisin CPU:lle, mutta esimerkiksi data-analyysissä tulokset taas palautetaan CPU:lle prosessoinnin jälkeen.

2.1.2 GPU:lla suoritettavat operaatiot

GPU:ta käytetään tavallisesti tilanteisiin, joissa suoritettava operaatio tai algoritmi on suhteellisen yksinkertainen. Alla olevassa Ohjelmassa 1 on tyypillinen esimerkki funktiosta, jonka suorittamista GPU:lla tulisi harkita isolla N:n arvolla.

```
void adder(int *a, int *b, int *c)
{
    for(int i = 0; i < N; i++)
    {
        a[i] = b[i] + c[i];
    }
}
```

Ohjelma 1: *Rinnakkaistettava koodi.*

Esimerkissä on nopeasti suoritettava artitmeettinen lisäysoperaatio, jossa muuttujien b ja c arvot sijoitetaan a:han. Kyseisen funktion voi rinnakkaistaa, koska silmukassa yhdelläkään N:n arvolla ei ole riippuvuuksia muuttujien suhteen. Tällöin sen suorittaminen voidaan jakaa usealle ytimelle. Jos for-silmukan kierrokset määrää muuttuja N, joka on mielivaltaisen suuri luku, saavutetaan sen rinnakkaistamisella mahdollisesti todella huomattava parannus suorituskykyyn.

2.1.3 CPU:lla suoritettavat operaatiot

Ohjelmissa on usein tilanteita, joita ei voida rinnakkaistaa. Tälläisiä tilanteita voi syntyä esimerkiksi silloin, kun jokin arvo on riippuvainen edeltävästä tuloksesta. Alla olevassa Ohjelmassa 2 on esimerkki edeltävästä tilanteesta, kun seuraavan for-silmukan kierroksen B tai C arvo on riippuvainen edeltävän kierroksen tuloksesta.

```
void adder(int *a, int *b, int *c)
{
    for(int i = 0; i < N; i++)
    {
        a[i] = b[i] + c[i];
    }

    if( a[i] % 2 == 0)
    {

        b[i+1] += 5;
    }
    else
    {
        c[i+1] += 10;
    }
}
```

Ohjelma 2: *CPU:lla ajettava koodi.*

Ohjelmassa 2 on lisäksi käytetty haarautumisoperaatioita, jotka ovat hyvin kalliita operaatioita suoritustehokkuuden kannalta. Tällöin on perusteltua hyödyntää CPU:ta funktion laskemiseen. Paremman laskentatehonsa takia se selviää tilanteesta GPU:ta huomattavasti nopeammin.

2.1.4 Laskennallinen esimerkki

Tyypillinen kotikäytössä oleva tietokoneen CPU sisältää nykyään 7-8 ydintä n. 3-4 GHz suoritusteholla ja GPU:n suoritusteho on n. 1,2 GHz 30-40 ytimellä varustettuna. Matemaattisesti näiden kahden laitteen eroa voidaan ilmentää seuraavalla tavalla.

Vertailussa käytetään CPU:n osalta AMD:n RyZen prosessoria, joka sisältää kahdeksan 3,5 GHz Zen corea. Jokainen näistä coreista voi suorittaa 2x4 leveitä lisäysoperaatioita sekunnissa. Kaavalla 1 voidaan laskea, kuinka monta operaatiota voidaan suorittaa samanaikaisesti.

$$O_{max} = OPC \cdot Cores \tag{1}$$

Kaavassa OPC (Operations Per Cycle) tarkoittaa maksimimäärää suoritettavia operaatioita. Tällöin kaavan 1 mukaisesti yhtäaikaisesti voidaan suorittaa (2 x 4) * 8 = 48 operaatiota. Alla olevalla kaavalla 2 saadaan laskettua, kuinka monta liukuluku-operaatioita voidaan laskea sekunnissa.

$$FLOP/s = O_{max} \cdot f \tag{2}$$

Kaavassa f tarkoittaa kellotajuutta, jolla prosessori suorittaa operaatioita. Kun kaavan 1 mukaisesti saatu tulos O_{max} kerrotaan ytimen kellotaajuudella kaavan 2 mukaisesti, saadaan tulokseksi 48 * $(3{,}5 * 10^9) = 1{,}68 * 10^{11}$ operaatiota sekunnissa.

Vertailun kohteena oleva AMD RX 580 GPU sisältää 36 kpl 1,26 GHz ytimiä. Jokainen näistä ytimistä pystyy suorittamaan 4 x 16 leveitä lisäysoperaatioita. Samanaikaisesti kaavan 1 mukaan pystytään siis suorittamaan (4 x 16) * 36 = 2304 operaatiota. Kun suoritustulos kerrotaan ytimen kellotaajuudella kaavalla 2, saadaan $2304 * 1{,}26 * 10^9 = 2{,}9 * 10^{12}$ operaatiota sekunnissa.

Kun GPU:n mahdollinen maksimitulos jaetaan CPU:n vastaavalla, saadaan $2{,}9 * 10^{12} / 1{,}68*10^{11} \approx 17{,}3$ kertaa parempi suorituskyky. Tämän vuoksi Ohjelmassa 1 esitetty yksinkertainen, rinnakkaistettava koodi mahdollisuuksien salliessa, kannattaisi suorittaa GPU:lla.

Mutta tilanteessa kuten Ohjelman 2 esimerkissä, jossa ei ole mahdollisuutta käyttää rinnakkaisuutta ollenkaan hyväkseen, on CPU lähes kolme kertaa nopeampi $3{,}5 * 10^9 / 1{,}26 * 10^9$ sen suorittamiseen. Lisäksi tiedon siirtämisessä GPU:lle kuluu aina aikaa. Jos suoritus aloitetaan prosessorilla välittömästi siirtämisen sijaan, voi luku tällöin jopa tuplaantua. Eli CPU parhaassa tapauksessa pystyisi mahdollisesti suorittamaan Ohjelman 2 lähes kuusi kertaa nopeammin. [2]

2.2 Kryptografinen kiihdytin

Kryptografia tarjoaa tehokkaan ratkaisun datan suojaamiseen. Kryptografiaa käytetään suojaamaan palvelimen ja käyttäjän välistä kommunikointia, kun kyseessä on arkaluontoista tiedonvälitystä. Konkreettisia esimerkkejä tällaisesta tiedonvälityksestä ovat VPN-yhteydet, verkkokaupasta ostohetkellä syntyvä pankki- tai luottokortin maksutapahtuma tai erilaiset pankkitransaktiot.

Perinteiset kotikäytössä olevat PC:t suoriutuvat kohtalaisen hyvin tällaisesta jokapäiväisestä kryptografisesta laskennasta. Tämä johtuu siitä, että yleensä laskettavan datan määrä ei ole suuri, koska käyttäjä yleensä joutuu kommunikoimaan vain yhden osapuolen kanssa. Asia on hieman erilainen, kun kyseessä on palvelin. Palvelin voi samanaikaisesti joutua kommunikoimaan satojen eri käyttäjien kanssa. Tällöin kryptografisesta laskennasta syntyvä kuorma alkaa muodostua pullonkaulaksi järjestelmille. Tätä varten on kehitetty kryptografisia kiihdyttimiä, jotka toimivat apu-prosessorin roolissa laitteistolla. Kryptografiset kiihdyttimet on suunniteltu nimenomaan suorittamaan hyvin haastavaa kryptografista laskentaa. Kryptografisten

kiihdyttimien avulla voidaan siis laskea CPU:lla suoritettavan laskennan kuormaa, jonka suoritustehoa voidaan hyödyntää siten jossain muualla.

Laskennalliselta tehokkuudeltaan kryptografiset kiihdyttimet myös päihittävät yleiskäyttöiset CPU:t selvästi. Viitteen [3] mukaisesti kryptografisella kiihdyttimellä voidaan samanaikaisesti suorittaa jopa 18-kertainen määrä SSL-yhteyksiä ja 25-kertainen määrä SET (Secure Electronic Transaction) -transaktioita sekunnissa verrattuna CPU:n.

2.3 Application-specific integrated circuit

ASIC on mikropiiri, jolla on yksi ja tietty käyttötarkoitus ja se on suunniteltu toteuttamaan se niin hyvin, kuin suinkin mahdollista. Sen suurin etu on nopeus ja riippuen suunnittelusta, myös pieni virrankulutus. Lisäksi se voi olla kooltaan hyvin pieni, koska siihen on kytketty vain sen toimintatarkoitukseen oleellisia komponentteja. Huonona puolena on sen suunnittelun ja valmistamisen kalleus. Suunnittelun kalleus syntyy siitä, että hyvin usein kaikki yksityiskohdat suunnitellaan itse alusta alkaen. Valmistamisessa syntyvät kustannukset taas syntyvät puolijohde prosessista, jossa maskikustannus on suurin. Jos päädytään ASIC-pohjaiseen ratkaisuun, on laitteella oltava suuri valmistus-, myynti- tai käyttömäärä, jotta suunnittelukustannukset saadaan katettua.

2.4 FPGA

FPGA on ASIC:n tavoin mikropiiri, mutta eroavuus näillä kahdella on se, että FPGA:n voi aina ohjelmoida uudelleen ja siten sitä voidaan päivittää vielä sen käyttöönoton jälkeenkin. Lisäksi FPGA:n käyttöönottaminen on huomattavasti nopeampaa, eikä se vaadi yhtä paljon suunnittelutyötä kuin ASIC. FPGA:ta käytetäänkin siksi usein luomaan ASIC-pohjaiselle ratkaisulle prototyyppi, jonka pohjalta voi aloittaa koko ASIC:n suunnittelutyön. FPGA on lisäksi hyvä ratkaisu käytettäväksi tilanteissa, joissa mikropiiriä ei tarvitse valmistaa useita kappaleita ja näillä sovelluskohteilla ei ole kovin kriittisiä nopeus-, virrankulutus- tai kokovaatimuksia. Molempien mikropiirien käyttökohteet liittyvätkin vahvasti sulautettuihin järjestelmiin. [4]

2.5 AI-kiihdytin

AI-kiihdytin on kryptografisen kiihdyttimen tapaan yleensä apuprosessori tai laitteiston osa, joka on suunniteltu toteuttamaan tekoälyn, konenäön ja koneoppimisen laskentaa tehokkaasti. Alkujaan AI-kiihdyttimien sovelluskohteissa käytettiin DSP (Digital Signal

Processor) prosessoreita tai FPGA-lautoja, mutta tekoälyn kehittyessä ja yleistyessä on huomattu tarve erityisille AI-kiihdyttimille.

Usein AI-kiihdyttimet ovat moniytimisiä prosessoreita, jotka keskittyvät matalan tarkkuuden artimetiikkaan [5]. Tämä johtuu siitä, että vaikka nykyään prosessorit pystyvätkin suorittamaan nopeasti tarkempaa aritmetiikkaa, niin tiedon siirtäminen prosessorilta muistiin ei ole nopeutunut samalla vauhdilla ja siitä on tullut järjestelmien pullonkaula. Matalan tarkkuuden aritmetiikalla tästä ongelmasta on päästy eroon. Tämän vuoksi AI-kiihdyttimet ovat viimeaikoina yleistyneet.

3 VIDEOVUON TOISTAMINEN

Videovuon toistamisella tarkoitetaan sitä, että jokin osapuoli haluaa tarjota multimediadataa verkon ja jonkin tiedonsiirtoprotokollan välityksellä reaaliaikaisesti. Videovuon toistamisessa käytettävä teknologia on kehitetty ja patentoitu jo vuonna 1920, ja sen pohjalta on kehitetty myös radioteknologiaan tarvittava laitteisto [6]. Tämän jälkeen on käytetty kehittyneempää digitaalista teknologiaa televisiolähetysten siirtämiseen, sekä puhelimien ääni- ja tekstiviestidatan siirtämiseen. 1990-luvun lopulla verkon tiedonsiirtonopeuksien kasvaessa myös videokuvaa sisältävän datan lähettäminen yleistyi ja se on siitä lähtien kasvattanut suosiotaan. Yleisiä tilanteita joissa videovuon suoratoistoa käytetään, on mm. television katseleminen, videopuheluiden soittaminen, sosiaalisen median suoratoistotoimintojen hyödyntäminen tai esimerkiksi web-kameraan pohjautuvan sovelluksen käyttäminen.

Videokuvaa sisältävän datan toistamiseen liittyvän teknologian käyttäminen on ollut hyvin nousujohteista ja suosittua viime aikoina. Tästä kertoo Ciscon [7] tekemä tutkimus. Tutkimuksen mukaan jopa 80 % verkon välityksellä siirrettävästä datasta vuoteen 2021 mennessä on videon toistamiseen tai siihen rinnastettavaa dataa. Lisäksi videon toistamisessa käytettävää teknologiaa käyttävien ihmisten määrän oletetaan kasvavan vuoden 2016 1,6 biljoonasta käyttäjästä jopa 1,9 biljoonaan. Kyseisessä tutkimuksessa on lisäksi ennustettu, että kuukaudessa katsottaisiin koko maailmassa jopa kolme trijoonaa minuuttia videokuvaa verkon välityksellä, mikä vastaa viittä miljoonaa vuotta videokuvaa kuukaudessa. Tälläkin hetkellä jo 67 % verkon yli siirrettävästä datasta koostuu videokuvaa sisältävästä datasta.

3.1 Koodekit

Koodekki on tiedoston pakkauksenhallintaan tarkoitettu algoritmi tai ohjelma. Paketoinnin tarkoituksena on saada tieto pienempään muotoon, jolloin sen lähettäminen verkon yli on nopeampaa, eikä sen tallentaminen kuluttaisi paljoa levytilaa. Koodekkeja on olemassa useita erilaisia. Osa koodekeista on tarkoitettu puheelle, osa videolle, lisäksi äänelle on olemassa omat koodekkinsa.

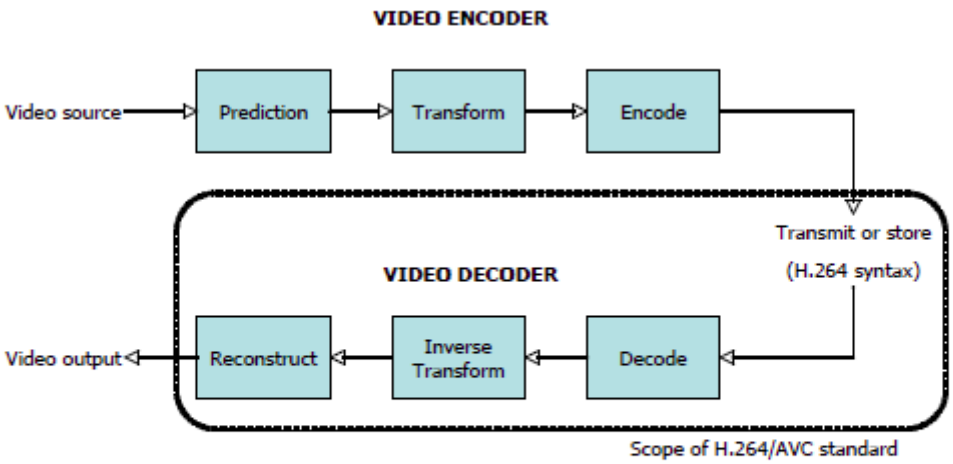
Äänikoodekin tehtävä on esimerkiksi analogisen äänen muuntaminen ja pakkaaminen digitaaliseen muotoon ja toisinpäin. Videokoodekkeja käytetään yleensä videokonferenssien, videovuon tai videoeditoinnin yhteydessä. Kun nämä kaksi

koodekkityyppiä yhdistää, saadaan multimediakoodekkeja. Multimediakoodekeilla voidaan hallinnoida ääntä sekä videota samanaikaisesti. Tunnetuimpia multimediakoodekkeja ovat H.264, VP9, AAC ja MP3.

3.1.1 H.264

H.264 on videopakkausstandardi, jonka avulla digitaalista videota pakataan alkuperäistä pienempään muotoon. Standardia käytetään esimerkiksi digitaalitelevisioiden, DVD-levyjen ja videovuon parissa. Standartointi mahdollistaa sen, että useiden laitevalmistajien tuotteet voivat toimia keskenään. [8]

Pakkaaja muuttaa videovuon tiivistettyyn muotoon ja purkaja avaa pakatun videon jälleenkatsottavaksi. Alla olevassa kuvassa 2 nähdään kuinka tallennettu video pakataan ja lopulta avataan jälleen katseltavaan muotoon.



Kuva 2: *H.264 Videoprosessointi. [9]*

Ensiksi video kuvataan jonkin laitteen, kuten puhelimen kameralla. Tämän jälkeen sen formaatti muutetaan joko ohjelmisto- tai laitteistopakkaajan avulla H.264 muotoon. Kun videovuota halutaan katsoa, tulee se ensin purkaa vastaavanlaisesti laitteisto- tai ohjelmistopurkajan avulla katsottavaan formaattiin takaisin. Purkaminen tapahtuu ohjelmassa ajonaikaisesti, eikä purettua videovuota tallenneta missään vaiheessa fyysisesti laitteistolle. Tätä varten ohjelma varaa yleensä itselleen muistia ja luo puskurin, johon se voi tallentaa videovuosta tulevaa purettua dataa tietyltä ajalta. Aina videokuvan esittämisen jälkeen kyseinen data poistetaan puskurista ja täytetään uudella.

3.1.2 VP9

VP9 on Googlen kehittämä, vapaasti käytettävissä oleva koodekki videon pakkamiseen. Aluksi sitä käytettiin pääasiassa vain Youtubessa pyörivien videoiden pakkaamiseen ja

esittämiseen, mutta myöhemmin sen suosio on kasvanut ja nykyään myös useat web-selaimet tukevat sitä.

Tällä hetkellä laitteistotasolla lähes kaikki laitevalmistajat tukevat VP9 purkamista, mutta sen pakkaamista tuetaan vielä harvakseltaan, mikä ilmenee webmproject-sivuston statistiikasta. [10]

3.1.3 AAC

AAC (Advanced Audio Coding) on äänen pakkaamiseen tarkoitettu tunnettu standardi. Sitä käytetään oletuksena useissa tunnetuissa multimedian toistamiseen tarkoitetuissa palveluissa kuten pelikonsoleissa, puhelimissa, Youtubessa sekä käyttöjärjestelmissä mm. Androidissa. Lisäksi AAC-tuki löytyy lähes poikkeuksetta jokaisesta multimediasoittimessa.

3.1.4 MP3

MP3 on toinen tunnetuista äänenpakkausmenetelmistä digitaaliselle äänelle. MP3 on menettänyt viimeaikoina hieman suosiostaan uusien äänikoodekkien, kuten AAC:n myötä. Se oli kuitenkin aikoinaan ensimmäinen tarpeeksi hyvällä tasolla operoiva koodekki, jolla pystyi pakkaamaan 128 Kbps 44.1 kHz stereotyyppistä ääntä normaalin käyttäjän tarpeita varten [11]. Tästä syystä MP3 on jäänyt elämään ja siksi lähes jokainen multimediaa soittava ohjelma tukee sitä.

3.2 FFmpeg

FFmpeg on ilmainen vuonna 2000 alkanut ohjelmistoprojekti, joka kokoaa yhteen monia kirjastoja ja ohjelmia, joiden avulla voidaan käsitellä videota, ääntä sekä multimediaa.

Se toimii kaikilla alustoilla ja pyrkii olemaan käyttöjärjestelmä- ja prosessoririippumaton. FFmpeg:n tarkoituksena on tukea kaikkia mahdollisia multimediaformaatteja välittämättä siitä, onko sen formaatin suunnitellut yritys, standardikomitea tai jokin korporaatio. [12]

FFmpeg tarjoaa ohjelmoijalle seitsemän erilaista vapaasti käytettävää kirjastoa, joita voi hyödyntää omissa ohjelmissaan ilmaiseksi tietyin ehdoin. Lisäksi FFmpeg:llä on oma komentorivityökalu FFmpeg ja oma HTTP (Hypertext Transfer Protocol) -palvelin, jolla voidaan lähettää reaaliaikaista multimediadataa. FFmpeg on toteuttanut myös hyvin yksinkertaisen multimediasoittimen FFplayn, joka pohjautuu projektissa oleviin kirjastoihin.

3.2.1 Libavcodec

Libavcodec on yksi FFmpeg tärkeimmistä kirjastoista. Se tarjoaa useita koodekkeja videon pakkaamiseen ja purkamiseen. Se on integroitu myös omana kokonaisuutenaan moniin tunnettuihin ohjelmiin ja kehyksiin. Tunnetuimpia ovat mutimediasoittimet MPlayer ja VLC, jotka käyttävät Libavcodeccia omissa sisäänrakennetuissa purkumoottoreissaan. Tämän avulla kyseiset soittimet pystyvät toistamaan erilaisia multimediaformaatteja.

3.2.2 Libavutil

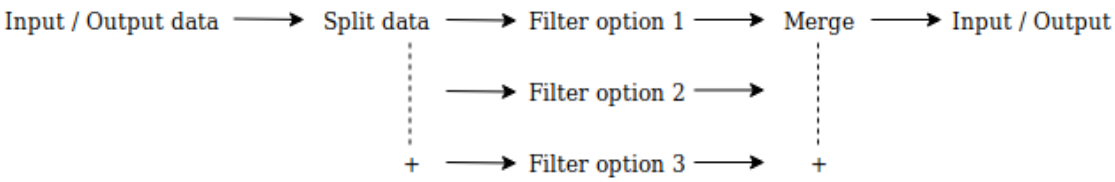
Libavutil on apukirjasto muille FFmpeg:n kirjastoille. Se pitää sisällään muiden kirjastojen käyttämiä merkkijonojen käsittelyyn tarvittavia toimintoja, satunnaisdatageneraattoreita, tärkeitä tietorakenteita, matemaattisten operaatioiden tekemiseen tarvittavia funktioita sekä kryptografiaan ja multimediaan muita yleisesti tarvittavia toimintoja.

3.2.3 Libavdevice

Libavdevice tarjoaa yleiskäyttöisen kehyksen, jonka avulla voidaan käsitellä laitteita, joita multimedian pyörittämiseen tarvitaan. Sen avulla voidaan listata kaikki mahdolliset sisään- ja ulostulolaitteet. Lisäksi se tarjoaa mahdollisuuden tarkistaa, minkälaisia multimediaformaatteja nämä laitteet tukevat. Tämän avulla käyttäjä voi valita itselleen sopivimman laitteen sisääntulevan ja uloslähtevän datan vastaanottamiseen ja lähettämiseen.

3.2.4 Libavfilter

Libavfilterin avulla multimediadataa voidaan filtteröidä ja siten saada muokattua toistettavaa videovuota halutunlaiseksi. Se toimii allaolevan kuvan 3 osoittamalla tavalla.



Kuva 3: *Multimediadatan filtteröinti.*

Ensiksi vastaanotettu tai lähetettävä data jaetaan useaan videovuohon. Jokaisessa videovuossa tehdään vain jokin tietty filtteröintitoimenpide. Lopulta kun kaikki videovuot on saatu käsiteltyä, ne yhdistetään tietyssä järjestyksessä takaisin yhteen,

jolloin data on käsitelty halutulla tavalla ja valmiina katsottavaksi tai eteenpäin lähetettäväksi.

3.2.5 Libavformat

Libavformat-kirjasto tarjoaa yleiskäyttöisen kehyksen äänen, videon ja videovuon limittämiselle. Se sisältää useita muxereita ja demuxereita erilaisille multimediaformaateille. Lisäksi se tukee useita sisään- ja ulostuloprotokollia, joilla käsitellään multimediadataa.

3.2.6 Libswscale

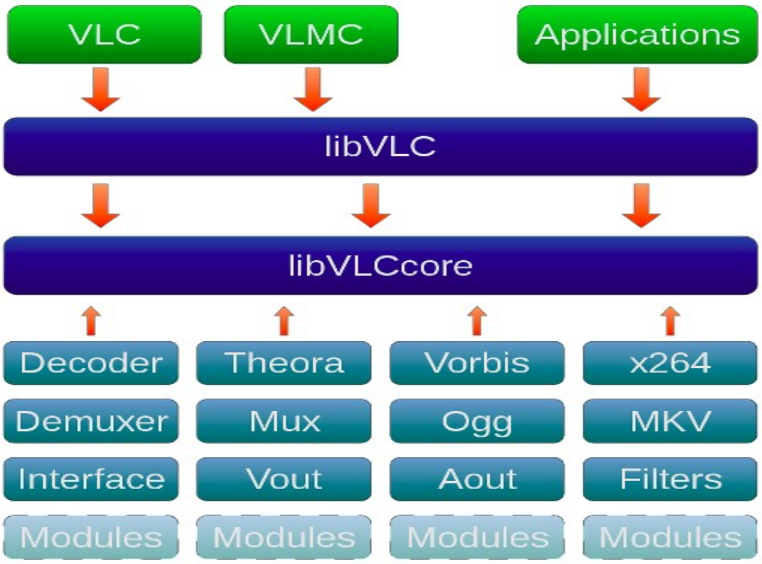
Libswscale tarjoaa optimoituja ratkaisuja kuvan skaalautuvuuden, väritilan ja pikseliformatoinnin tarpeita varten. Erityisesti, kun videon kokoa muutetaan ajonaikaisesti, astuu libswscale mukaan. Se tarjoaa tätä varten erilaisia vaihtoehtoja jälleenskaalautuvuudelle ja algoritmeja, joiden avulla kuvan kokoa voidaan muuttaa niin, että käyttäjä ei huomaa eroa kuvan laadussa.

3.2.7 Libswresample

Libswresample tarjoaa ominaisuuksia äänen käsittelyyn, kuten tajuuden tai äänikanavan muuttamiseen stereosta monotyyppiseksi. Lisäksi se tarjoaa toiminnallisuuden äänen muunnosoperaatioille mm. muuttamalla 16 bittistä ääntä 8 bittiseksi.

3.3 VLC

VLC on voittoa tavoittelematon säätiö, joka kehittää käyttöjärjestelmäriippumatonta, erilaisten multimediaformaattien toistamiseen tarkoitettua soitto-ohjelmaa. VLC sisältää useita kolmannen osapuolen tarjoamia kirjastoja, jotka on integroitu yhteen. Tunnetuin ja todennäköisesti tärkein sen käyttämä kolmannen osapuolen komponentti on Ffmpeg. Sen avulla VLC voi käsitellä multimediadataa laitteisto- ja alustariippumattomasti usein eri tavoin. FFmpeg projektista se käyttää erityisesti Libavcodeccia ja Libavformattia hyödykseen. Libavcodeccia se käyttää useiden eri multimediatiedostojen pakkaamiseen ja purkamiseen. Libavformattia se käyttää tunnistamaan multimediatiedostoista, kuinka ne on mahdollisesti pakattu tai kuinka raakadata halutaan pakata. Tällöin Libavformat muxerien ja demuxerien avulla osaa ohjata dataprosessoinin eteenpäin sitä parhaiten toteuttavalle osalle. Alla olevassa kuvassa 4 on kerrosnäkymä VLC:stä.



Kuva 4: *VLC kerrosnäkymä. [13]*

Kuvassa VLC tarkoittaa itseään, eli multimediasoitinta. VLMC (VideoLAN Movie Creator*)* on hieman tuntemattomampi VLC:n projekti, se on videon editoimiseen tarkoitettu työkalu, joka pohjautuu LibVLC kirjastoon. Applications tarkoittaa ohjelmia tai applikaatioita, jotka hyödyntävät tavalla tai toisella LibVLC-kirjaston palveluita.

Näin useat eri ohjelmat rakentuvat LibVLC-kirjaston ympärille tarjoten rajapinnan LibVLCcore:n palveluille. LibVLCcore hallitsee kaikkia matalan tason asioita, kuten säikeitä, moduuleja, kelloja, soittolistoja ja monia muita asioita.
Seuraava kerros sisältää dekooderit, eli purkajat. Purkaja valitsee vastaanotetulle datalle oikean demuxerin, jonka tehtävänä on lukea datan otsikkotietoja ja näiden tietojen avulla päätellä, kenelle kyseinen data tulisi lähettää eteenpäin prosessoitavaksi.

Theora on vapaassa käytössä oleva videokoodekki, joka pohjautuu On2- ja VP3-koodekkeihin. Se on osa Ogg-projektia, joka on avoin ja standartoitu tiedostomuoto multimediadatan säilytykseen. Ääni tähän tiedostomuotoon saadaan Vorbis:n avulla, joka on avoin ja vapaasti saatavilla oleva äänenpakkausmenetelmä audiolle. Näiden kaikkien kolmen kehittämistä ohjaa, valvoo ja ylläpitää Xiph.org-säätiö.

x264 on VideoLan:n kehittämä ohjelmistokirjasto, jolla videovuon voi pakata helposti H.264/MPEG-4 AVC-formaattiin. Lisäksi LibVLCcore tarjoaa tuen useille erilaisille multimediaformaateille, kuten kuvassa esiintyvä MKV (Matroska). MKV sallii useiden erilaisten ääni- ja tekstitystiedostojen lisäämisen yhteen ja samaan tiedostoon.

Vout (Video out) ja Aout (Audio out) komponentit tarjoavat mahdollisuuden käyttäjälle valita äänen- ja videonlähtömoduulin. VLC käyttää moduuleita tekemään suurimman osan työstä. Se saattaa putkittaa datan käsittelyä moduulilta toiselle, jolloin ne ladataan käytettäväksi aina ajonaikaisesti tarvittaessa. Tällä pyritään siihen, ettei soitin itsessään söisi liian suurta osaa tietokoneen välimuistista ja mahdollistaa myös soittimen käyttämisen laitteissa, joissa muistin suhteen tulee olla tarkkana. Jokainen moduuli tarjoaa erilaisia ominaisuuksia ja ne on toteutettu suorittamaan vain yhtä tiettyä asiaa. Tämä helpottaa VLC:n siirrettävyyttä laitteelta toiselle, koska voi valita vain itselle tarvittavat modulit käytettäväksi.

3.4 Videovuon pakkaaminen

Videovuon pakkaaminen tarkoittaa sitä, että videovuossa lähtevä data tiivistetään pienempään muotoon, jolloin sen lähettämisestä tai tallentamisesta syntyvät kustannukset ovat paljon raakadataa pienempiä. Tällä saavutetaan nopeampi tiedonsiirto verkon välityksellä, mikä on elintärkeää varsinkin TV-lähetysten osalta.

Videovuota voi pakata joko siihen tarkoitetulla ohjelmalla, jolloin videovuon pakkaamiseen käytetään CPU:ta. Vaihtoehtoisesti ja useissa videovuon pakkaamiseen liittyvissä tilanteissa kannattaa kuitenkin käyttää sille omistettua laitteistoa. Videovuon pakkaamisessa pyritään mahdollisimman häviöttömään lopputulokseen. Häviöttömyys tarkoittaa sitä, että muunnettu raakadata saadaan tietyllä purkualgoritmillä takaisin alkuperää vastaavaan muotoon. [14]

Pakkaamisessa on tyypillistä se, että laitteistolla suoritettava tiedon tiivistäminen tuottaa hieman huonompilaatuista kuvaa, kuin ohjelmistolla tehtävä tiedon tiivistäminen. Laitteistolla tehtävä tiivistäminen on kuitenkin nopeampi tapa paketoida videovuota eteenpäin jaettavaksi. Tästä syystä usein reaaliaikaisesti tuotettavassa videovuossa tulisi harkita ulkoisen laitteen käyttämistä jos se on mahdollista. Lisäksi CPU:lla ei välttämättä aina ole tarpeeksi resursseja videovuon pakkaamiseen ohjelmistokoodekkien avulla, koska pakkaaminen vaatii suurta laskentatehoa sitä suorittavalta prosessorilta.

3.4.1 Ohjelmistopakkaaja

Ohjelmistopakkaajalla tarkoitetaan ohjelmaa, jolla dataa tiivistetään. Yleensä tälläisen ohjelman saa helposti asennettua PC:lle tai kannettavalle tietokoneelle. Usein ohjelmistopakkaajalla tiivistetyn datan laatu on erinomaista ja datan muokkaaminen vaivatonta.

Ohjelmistopakkaajien päivittäminen on useissa tapauksissa hyvin helppoa. Tällöin niille voidaan päivittää kaikki tarvittavat ominaisuudet uusien pakkausstandardien ja koodekkien tarpeisiin. Ainoana huonona puolena ohjelmistopakkaajalle on niiden pakkaamisnopeus. Hidas pakkaamisnopeus johtuu siitä, että tietokoneella pyörii samanaikaisesti paljon muutakin, johon tarvitaan prosessorilta suoritinaikaa. Tällöin ohjelmistopakkaajalle ei voida tarjota täyttä prosessointiaikaa prosessorilta, mikä johtaa hitaampaan lopputulokseen. Lisäksi CPU kuormittuu huomattavasti, ja muiden asioiden tekeminen siinä sivussa ei välttämättä onnistu enää niin sutjakasti.

3.4.2 Laitteistopakkaajat

Laitteistopakkaajat ovat tätä varten omistautuneita prosessointiyksiköitä, joilla käytetään ennalta määriteltyjä algoritmejä videovuon pakkaamiseen. Usein laitteistopakkaajat ovat vain ammattikäytössä ja saattavat olla hyvinkin kalliita hankintoja.

Laitteistopakkaajan suunnittelu on työlästä ja vie paljon aikaa. Tämä johtaa siihen, että niissä ei välttämättä aina ole saatavilla kaikkein uusimpia koodekkeja. Lisäksi ne eivät ole kovin joustavia. Kun laitteistopakkaaja on kerran suunniteltu ja valmistettu, ei siihen tämän jälkeen voi enää tehdä suuria muutoksia. Sillä on vain tietyt ennalta määritellyt säätöominaisuudet, mikä usein estääkin uuden koodekin käyttöönottamisen. Uuden markkinoille tulleen pakollisen koodekin myötä saattaa siis koko laitteen joutua uusimaan.

3.5 Videovuon purkaminen

Purkaminen on pakkaamisen vastakohta, eli siinä vastaanotettu pakattu data puretaan. Data muutetaan purkamisalgoritmillä takaisin alkuperäiseen muotoon. Tällöin pakattu videovuo on jälleen katsottavissa. Purkamista voi tehdä yhtälailla joko siihen tarkoitetulla ohjelmalla tai laitteistolla.

Useat multimediasoittimet tarjoavat nykyään mahdollisuuden laitteistokiihtyvyyden hyödyntämisen videovuon katselemiseen. Laitteistolla pyöritettävä video kuluttaa luonnollisesti vähemmän CPU:n kapasiteetta ja virtaa, joten usein se on paras vaihtoehto videovuon purkamiseen. Ongelmana laitteistokiihtyvyyttä käytettäessä ovat uudet koodekit, joita kaikki vanhemmat laitteet eivät välttämättä enää tue. Näissä tapauksissa joutuu videovuota pyörittämään ohjelmiston avulla.

Jotta laitteistoa voitaisiin hyödyntää videovuon katseluun, tarvitsee se videovuota pyörittävältä ohjelmalta tuen laitteistokiihtyvyyden käyttämiseen.

3.6 DirectX ohjelmointirajapinta

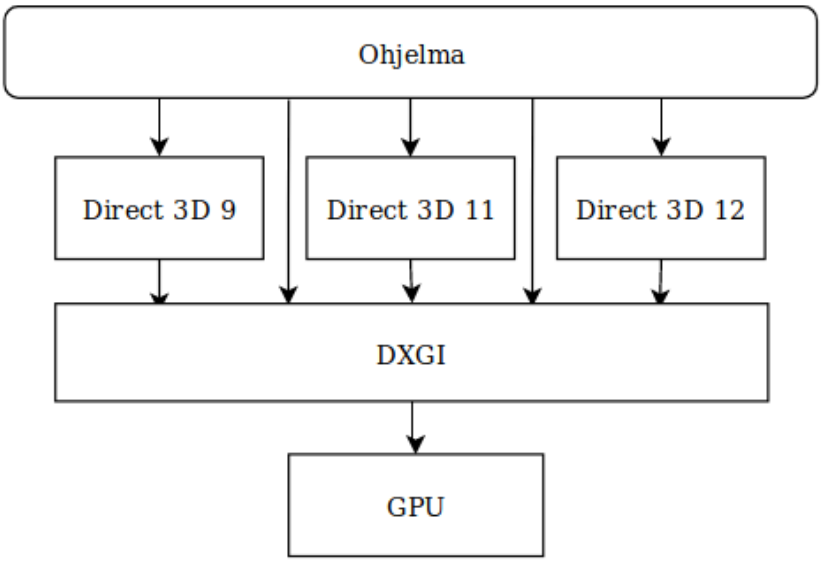
DirectX on Microsoftin tarjoama ohjelmointirajapinta. Se on suunniteltu tarjoamaan matalan tason rajapinta ohjelmoijille, jotta he voivat saada kaiken mahdollisen hyödyn irti laitteistosta, joka pyörii Windows-käyttöjärjestelmällä.

VLC:n DirectX liittyy ainakin Libavcodecin kautta. Libavcodec tarjoaa käyttäjälle mahdollisuuden valita, kuinka pakattua videota puretaan käyttäjälle, jolloin sen purkamista voidaan hallita ohjelmallisesti myös itse. Kun Libavcodecille antaa tiedon videovuon toistamisen aloituksen yhteydessä, että vastaanotettu videovuo halutaan purkaa hyödyntäen tiettyä DirectX:n rajapintaa, voidaan koodia puukottamalla tehdä sinne itse haluttuja muutoksia ja uusia ominaisuuksia.

DirectX koostuu useista erilaisista komponenteista, joiden avulla voidaan hallita 2D- ja 3D-grafiikkaa sekä ääntä. Lisäksi se antaa mahdollisuuden fonttien muuttamiseen, oheislaitteilta saatavan tiedon keräämiseen ja se sisältää useita erilaisia diagnostiikan työkaluja. Kaikki nämä ovat irrallisia, toisistaan riippumattomia komponentteja, joten ohjelmoija voi valita ja käyttää vain itse tarvitsemiaan palasia DirectX-perheestä.

DirectX:n idea on antaa ohjelmoijalle kaikki tarpeellinen oman ohjelmansa toteuttamiseen ja tarjota standardin matalan tason rajapinnan laitteiden hallintaan. Tällöin ohjelmoija pystyy kehittämään paljon yleiskäyttöisempiä ohjelmia, joita voidaan käyttää laitteistomallista riippumatta, eikä siten tarvitse ottaa huomioon, millaisella laitteistolla kyseistä ohjelmaa tullaan ajamaan. DirectX hoitaa laitteiston tunnistamisen sekä sen kanssa kommunikoinnin ohjelmoijan puolesta.

Laitevalmistajat myös huomioivat tämän standardin laitteita kehittäessään, jotta heidän laitteensa pystytään tunnistamaan näiden standardien mukaisesti. Tämän avulla ohjelmaa, joka on julkaistu ennen uudempaa laitetta, pystytään myös ajamaan uudella tehokkaammalla laitteella [15]. Alla olevassa kuvassa 5 on nähtävillä, kuinka ohjelma voi kutsua DirectX:n komponentteja.



Kuva 5: *DirectX API 3D-grafiikalle.*

Ohjelma voi siis kutsua tarvittaessa suoraan DXGI-rajapintaa, jonka avulla saadaan yhteys GPU:n. Vaihtoehtoisesti, jos ohjelmalla ei ole erityisiä tarpeita, voi ohjelmoija käyttää Direct 3D X-versiota grafiikan esittämiseen, joka pellin alla kutsuu ja käyttää itse DXGI-rajapintaa grafiikkaohjaimen hyödyntämiseen.

3.6.1 DXGI

DXGI (DirectX Graphics) on komponentti, joka on lähimpänä rautaa. Sen rajapintaa käyttävät hyväkseen muut Direct 3D -komponentit ja sen pääasiallisena tehtävänä on esittää kuvaa ruudulla sekä suorittaa kuvalle joitain yksinkertaisia operaatioita kuten gamman sääteleminen. Lisäksi DXGI selvittää, millaista resoluutiota näyttöpääte ja näytönohjain tukevat. Yksi sen tärkeimmistä tehtävistä on myös tunnistaa laitteistolla olevat GPU:t ja enumeroida ne, jotta Direct 3D -rajapintojen avulla näitä laitteita voidaan käyttää halutulla tavalla.

Ohjelmat voivat kommunikoida suoraan DXGI-rajapinnan kanssa, mutta tarvittaessa kommunikointi tapahtuu Direct 3D -rajapintojen kautta oletusasetuksilla. Yleensä suoraa kommunikointia tehdään erityisesti silloin, kun käytössä on useita laitteita ja niitä halutaan käyttää omiin tarkoituksiinsa. [16]

3.6.2 D3D11VA

D3D11VA nimi tulee sanoista Direct3D 11 Video Acceleration, jossa Direct3D:llä tarkoitetaan grafiikan ohjelmointirajapintaa ja sitä käytetään erityisesti renderöimään 3D-grafiikkaa tilanteissa, joissa suorituskyky on tärkeässä osassa. Direct3D käyttää hyväkseen laitteistoa, jonka avulla se pystyy suorituskykyisesti renderöimään kyseistä

grafiikkaa. Numero 11 taas tulee sen versiosta ja kyseinen versio on esitetty julkisesti ensimmäisen kerran vuonna 2008 Gamefesteillä 22. heinäkuuta [17]. Loppuosalla ”VA”, taas tarjoitetaan sitä, että sillä kiihdytetään videota.

Se on siis ohjelmointirajapinta, jonka avulla voidaan nopeuttaa videoprosessointia, eli joko pakkaamattoman videokuvan käsittelyä, jotta se saataisiin pakattua haluttuun muotoon tai purkaa pakattu videokuva, jotta se saataisiin takaisin toistettavaan muotoon käyttämällä GPU:ta datan pakkaamiseen tai purkamiseen CPU:n sijaan. Osa D3D11VA:n purkamiseen liittyvistä operaatioista on toteutettuna suoraan GPU:n ajureissa, joita se hyödyntää aina mahdollisuuksien salliessa. Tämä tarkoittaa sitä, että jos D3D11VA-purkajaa halutaan käyttää, tarvitaan siihen näytönohjain, jolla on tuki rajapinnan käyttämistä varten.

Kaikki purkamiseen liittyvät operaatiot joita GPU:lla suoritetaan, tarvitsevat sen isäntäohjelmalta tai laitteelta ensin purettavan datan, ja tiedon siitä, mitä operaatioita tälle datalle tulisi suorittaa. Tälläistä toimintaa kutsutaan laitteistokiihdytykseksi, kun data siirretään käsiteltäväksi erilliselle, sille tarkoitetulle laitteelle. Kun data on käsitelty GPU:lla, se palautetaan takaisin käyttäjälle esitettäväksi.

Purkamisessa tarvitaan myös muitakin operaatioita kuin vain GPU:n ajureilla tehtäviä. Tällöin tarvittavat operaatiot täytyy löytyä niitä käyttävissä käyttäjätason ohjelmissa. Yleensä loput purkamisessa tarvittavat operaatiot, joita ei voida suorittaa näytönohjaimen ajureista saatavilla komennoilla, ovat saatavilla erillisinä koodekkikirjastoina. [18]

4 TYÖKALUKOKOELMA

Työkalukokoelma on lähes poikkeuksetta aina ensimmäinen asia, jota tulee miettiä, kun uutta ohjelmointiprojektia ollaan käynnistämässä. Työkalukokoelmalla tarkoitetaan niitä ohjelmia, joilla ohjelmointityöstä syntyvä lähdekoodi muutetaan ajettavaksi tiedostoksi, jota kohdealustan prosessori ymmärtää ja pystyy suorittamaan. Huomioitavia asioita kohdealustan prosessorissa ovat mm. arkkitehtuurin tyyppi, eli onko se MIPS, ARM, vaiko kenties x86_64. Onko kyse Big- vai Little-endianista, eli missä järjestyksessä tavuja lasketaan. Little-endian ympäristössä laskujärjestys alkaa aina vähiten merkitsevästä tavusta, kun taas Big-endianissa se on päinvastoin. Voi myös olla, että kohdeprosessori ei ymmärrä liukulukuoperaatioita, jolloin työkalukokoelman tulee kutsua ja liittää käännettävään ohjelmaan liukulukujen laskentaan tarkoitettu kirjasto raudalla tehtävien operaatioiden sijaan.

Yleensä työkalukokoelma sisältää kääntäjän, jolla ohjelman lähdekoodi käännetään joko assembly- tai konekielelle. Käännöksen lopuksi lähdekoodi muutetaan objektitiedostoksi. Lisäksi työkalukokoelma sisältää linkitysohjelman, joka yhdistää käännetyt objektitiedostot yhdeksi ajettavaksi ohjelmatiedostoksi kuten .exe. Työkalukokoelmaan kuuluvaksi voidaan laskea vielä ajonaikaiset kirjastot, joita ohjelma tarvitsee suoritustaan varten. Työkalukokoelman voi rakentaa joko natiivina, eli ohjelma käännetään samalle alustalle kuin missä itse kehitystyö tehdään tai se rakennetaan tukemaan ristikäännöksiä. Ristikäännöksissä kohdealusta on eri kuin se, millä kehitystyö tehdään. Tässä diplomityössä keskitytään enemmän ristikääntäjiin, koska se on työn toteuttamisen kannalta suositeltavampi tapa VLC:n puolesta.

Huomioitavaa on, että vaikka kohdealustalla ja kehitysympäristöllä olisi sama prosessoriarkkitehtuuri esim. x86_64 ja ne muutenkin vastaisivat toisiaan, ei se siltikään takaa sitä, että ohjelma toimisi molemilla alustoilla samalla tavalla. Aina kun kehitystyötä tehdään toiselle alustalle, tulisi ohjelmat kääntää erityisesti tätä alustaa varten. Tietyissä tapauksissa testausmielessä on hyväksyttävää ja jopa suotavaa välillä testata, miltä ohjelma näyttää kehitystympäristössä. Tämä siksi, että koko ohjelmiston siirtäminen ja testaaminen toisella laitteella voi olla hyvin aikaavievää ja raskasta. Jos siis on mahdollista testata joitain ohjelman uusia toimintoja nopeasti kehitysympäristöllä, kannattaa se varmistaa sillä ensin. Tällä voidaan nopeuttaa ohjelman kehitystä, sekä optimoida ajankäyttöä.

4.1 Työkalukokoelman rakentaminen

Työkalukokoelman rakentaminen voi olla hyvinkin yksinkertaista. Joskus saattaa riittää, että ladataan vain jokin pakattu esim. zip- tai tar-tiedosto. Ladattu tiedosto puretaan ja sieltä löytyy valmis asennuspaketti, joka suorittaa kaiken automaattisesti käyttäjän puolesta. Lisäksi useilla Unix-pohjaisilla käyttöjärjestelmillä on mahdollista ladata pakettienhallinnan avulla valmiiksi rakennettu kääntäjä erilaisiin käyttötarpeisiin.

Näissä kahdessa vaihtoehdossa on kuitenkin huonona puolena se, että niiden räätälöitävyys on vähäistä. Joissakin ohjelmistoprojekteissa saatetaan tarvita esimerkiksi uusin versio GCC-kääntäjästä siksi, että se tarjoaa jonkin uuden tärkeän ominaisuuden tai bugikorjauksen, joka on olennaista tämän projektin toteuttamisen kannalta, mutta se ei ole saatavilla valmiiksi käännetyissä tai ladatuissa paketeissa.

Tällaisessa tapauksessa voi joutua rakentamaan itse oma työkalukokoelman ja se ei aina ole välttämättä kovin kivuton toimenpide. On mahdollista, että joissakin projekteissa työkalukokoelman rakentaminen voi olla yksi sen vaikeimmista ja suurimmista ongelmista. Toimivan työkalukokoelman suunnitteluun ja rakentamiseen tulee kuitenkin panostaa, sillä huonosti toteutun työkalukokoelman vaikutukset voivat johtaa koko projektin epäonnistumiseen.

Useimpia tapauksia varten on kuitenkin olemassa jo valmiiksi rakennettu työkalukokoelma. Valmiiden työkalukokoelmien käyttöönottaminen on nopeaa ja vaivatonta, joten kannattaa aina ensisijaisesti lähteä etsimään tälläistä ratkaisua. Usein myös valmiiksi rakennettu työkalukokoelma siihen perehtyneen asiantuntijan puolesta on paljon pidemmälle asti mietitty kuin itse toteutettu. [19]

Kun työkalukokoelma on saatu rakennettua ja todettu toimivaksi, on tärkeää pitää se samanlaisena läpi koko projektin, jottei synny ongelmia eri versioiden välillä. Esimerkiksi, jos käyttöjärjestelmästä asennetaan uusi versio, on hyvinkin mahdollista, että tässä voi olla esiasennettuna jokin uudempi kääntäjä tai sen pakettihallinnan ladattavat paketit ovat päivittyneet uudemmiksi versioiksi vanhaan verrattuna. Tämä saattaa johtaa taas siihen, ettei käännös enää toimi kohdealustalla tai koko ohjelma ei käänny uudella kokoonpanolla.

4.2 Ajonaikaiset kirjastot

Kun jotakin ohjelmaa ajetaan käyttöjärjestelmällä, tarvitsee se aina jonkin kommunikointivälineen, jolla se voi keskustella käyttöjärjestelmän kanssa. Ohjelma voi esimerkiksi tarvita lisämuistia, tiedoston avausta, I/O-komennon lukemista sekä monia muita asioita, joiden toteuttamisesta huolehtii käyttöjärjestelmä. Tätä varten

käyttöjärjestelmällä on olemassa oma ohjelmointirajapinta, jonka avulla käyttäjän ohjelmisto voi pyytää käyttöjärjestelmää suorittamaan käyttöjärjestelmäoperaatioita.

Unix-pohjaisissa järjestelmissä käytetään POSIX (Portable Operating System Interface) -standardia ja Windowsilla käytetään Windows API:a (Application Programming Interface). Nykyään on olemassa myös ohjelmia, joilla pystytään ajamaan toisen käyttöjärjestelmän ohjelmistoja ilman lähdekoodin muuttamista. Wine on hyvä esimerkki tälläisestä ohjelmasta. Sillä pystyy ajamaan .exe päätteisiä tiedostoja Unix-pohjaisella käyttöjärjestelmällä, mikä ei normaalisti olisi muuten mahdollista.

Nämä ajonaikaiset kirjastot sijaitsevat pääsääntöisesti käyttöjärjestelmän suojatulla alueella, johon käyttäjätasolta ei voi tehdä kuin pyyntöjä, jotka joko toteutetaan tai hylätään. Etuna tässä kuitenkin on se, että sillä voidaan säästää levytilaa huomattava määrä, kun kirjastoja ei tarvitse linkittää staattisesti jokaiseen ohjelmaan, vaan niitä voidaan kutsua ajonaikaisesti.

Haittapuolena ovat erilaiset käyttöjärjestelmät, jotka toteuttavat tämän kommunikoinnin hieman eri tavoin. Tällöin samaa lähdekoodia saattaa joutua muuttamaan hieman, jotta se toimisi myös toisella käyttöjärjestelmällä siten, että sekin pystyisi ymmärtämään näitä pyyntöjä. Tätä varten on kuitenkin olemassa ohjelmia ja työkaluja, jotka käännösprossesin aikana tekevät tämän automaattisesti käyttäjän puolesta.

4.3 Linkitysohjelma

Linkitysohjelman tehtävänä on yhdistää kaikki kääntäjän luomat objektitiedostot yhdeksi ajettavaksi ohjemaksi, kirjastoksi tai toisiksi objektitiedostoiksi. Hyvänä esimerkkinä voitaisiin pitää tilannetta, jossa pääohjelma kutsuu printf()-metodia. Tälle metodille löytyy määrittely <stdio.h> nimisestä kirjastosta, joten kyseinen kirjasto pitää pystyä linkittämään ohjelman suoritukseen mukaan jollakin tavalla [20].

Tapoja eri lähdekoodi- ja objektitiedostojen linkittämiseen on kaksi. Linkitysohjelma linkittää tiedostot yhteen käyttäen joko dynaamista linkitystä, jossa ajonaikaisia kirjastoja linkitetään ohjelmaan vain silloin, kun sitä ajetaan. Vaihtoehtoisesti, jos koko ohjelmiston arkkitehtuuri on hyvin sekava ja se on riippuvainen monista erilaisista kirjastoista, joita ei välttämättä jokaiselta käyttäjältä löydy. Tällöin on suotavaa käyttää ainakin osittain linkittämismetodia, eli staattista linkitystä. Staattista linkitystä tulisi tehdä ainakin niiden ohjelmistokomponenttien osalta, joiden puuttuminen loppukäyttäjiltä on todennäköistä.

4.3.1 Dynaaminen linkitys

Yleensä dynaamista linkitystä tehdään oletuksena, koska siten käännettävän ohjelman koko pystytään pitämään maltillisena. Tämä tarkoittaa sitä, että ohjelmassa on olemassa määrittelemättömiä symboleita, mutta näille ladataan tarvittavat määrityksen omaavat kirjastot ja tiedostot ohjelman käynnistyessä. Linkitetyt kirjastot ovat yleensä sellaisia, joita käytetään usein monissa eri ohjelmissa ja ne ovat monissa tapauksissa siten saatavilla suoraan käyttöjärjestelmältä.

Dynaamisen linkityksen etuna pienen ohjelmankoon lisäksi on se, että ohjelmaa ei tarvitse kääntää uudelleen, jos jostakin ohjelman käyttämästä kirjastosta on löytynyt bugi. Tässä tapauksessa riittää vain kirjaston korvaaminen uudella toimivalla versiolla.

4.3.2 Staattinen linkitys

Staattisella linkityksellä tarkoitetaan päinvastaista kuin dynaamisella, eli siinä linkitetään kaikki tarvittavat tiedostot staattisesti ajettavaan ohjelmaan, joka kasvattaa sen kokoa huomattavasti. Hyötynä staattisessa linkityksessä on kuitenkin se, että tällöin ohjelman siirrettävyys kasvaa. Tällöin ohjelma ei ole riippuvainen käyttöjärjestelmän tarjoamista kirjastoista, eli eri kirjastoversiot käyttöjärjestelmissä eivät vaikuta sen toimintaan, eikä konflikteja siten synny. Ohjelmaa on lisäksi mahdollista pyörittää useilla eri laitteilla ongelmitta.

4.4 Ristikääntäjä

Ristikääntäjä on työkalu, jolla lähdekoodi muutetaan objektikoodiksi. Objektikoodia voidaan ajaa toisella alustalla, kuin missä kääntäminen ensisijaisesti tapahtuu. Java on hyvä esimerkki tästä. Kun Java-ohjelmaa käännettään, siitä tehdään JVM:lle (Java Virtual Machine) yhteensopiva käännös. Kone tai laite, jolla ohjelmaa ajetaan on siis aina eri, kuin millä se on käännetty. Yksinkertaistettuna kääntäjä luo konekoodia, jota voidaan ajaa vain kohdealustalla, eikä sen kääntäneellä koneella.

Tarvittavat ohjelmat ja paketit, joita ristikääntäjän rakentamiseen tarvitaan, ovat Binutils ja MinGW (Minimalist GNU for Windows). Binutils on kokoelma binäärityökaluja ja sisältää mm. linkitysohjelman ja Assembler-kääntäjän. MinGW puolestaan mahdollistaa ohjelmakoodin kääntämisen Windows-ympäristölle. Lisäksi tämän työn puitteissa työkalukokoelmaan tarvitaan myös GCC-kääntäjä, koska VLC:n lähdekoodi koostuu C/C++ tiedostoista.

4.4.1 Binutils

Binutils nimensä mukaisesti ”Binary Utilities” tarjoaa kokoelman tärkeitä työkaluja, joilla voidaan hallita ja luoda binäärisiä ohjelmatiedostoja. Näiden työkalujen avulla käännetään aluksi myös GCC sekä koko ohjelman käännösvaiheessa toteutetaan tiedostojen linkittäminen yhteen.

4.4.2 MinGW

MinGW on avoimen lähdekoodin projekti, joka tarjoaa työympäristön ohjelmistokehitystä varten Windows-ympäristöön. Se on ajonaikainen ympäristö, jolla saadaan GCC tukemaan Windowsin natiivi binäärejä. Natiivi binääreillä tarkoitetaan sitä, että käyttöjärjestelmä ymmärtää ohjelmaa ja pystyy siten kommunikoimaan raudan kanssa ja ajamaan ohjelmakoodia. MinGW:tä käytetään erityisesti silloin, kun halutaan tehdä ristikäännöksiä, eli kääntää ohjelmakoodia Linuxilta Windowsille. Sillä on lisäksi mahdollista tehdä natiivi käännöksiä, mutta vielä tällä hetkellä Visual C++:n kääntäjä on tehokkaampi ja nopeampi vaihtoehto natiivikääntämiseen.

4.4.3 GCC

GCC (GNU Compiler Collection) on yksi suosituimmista, ellei jopa suosituin C/C++ kääntäjä. Se sisältää kokoelman työkaluja, jolla voidaan kääntää korkean tason ohjelmakoodia binääritasolle objektitiedostoiksi. GCC tukee useita eri prosessoriarkkitehtuureja, minkä takia sen käyttäminen on suosittua ja helppoa. GCC tukee myös muitakin ohjelmointikieliä, kuten Javaa, Objective-C:tä, Fortran:a, Adaa ja Pascalia. [21]

4.4.4 Ristikääntäjän rakentaminen

Yksinkertaistettuna MinGW-pohjaisen ristikääntäjän rakentaminen koostuu viidestä vaiheesta. Jokaisessa vaiheessa tulee ladata tarvittavat paketit ja ne tulee kääntää käännettävän tietokoneen omalla natiivikääntäjällä.

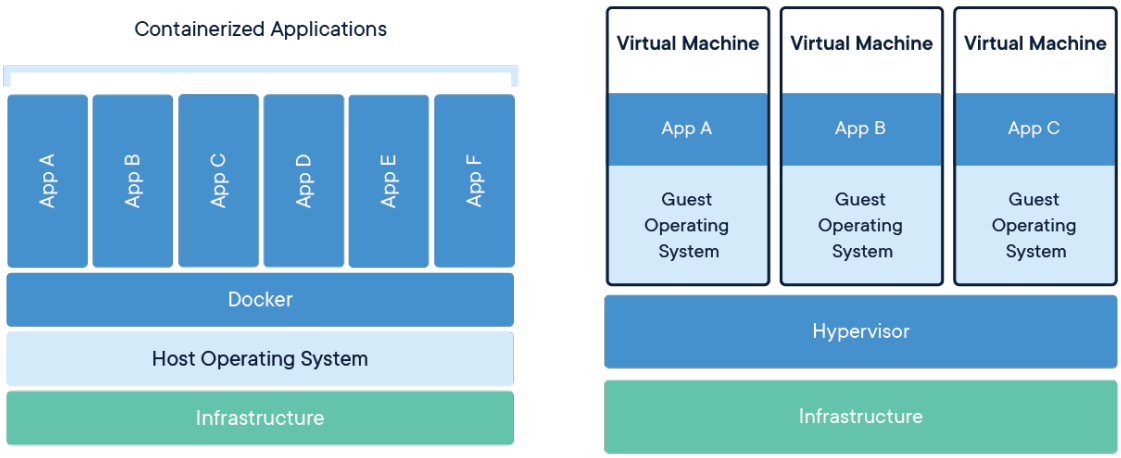
Ensimmäisestä vaiheesta alkaen tulee huomioida, minkä tyyppistä kohdearkkitehtuuria varten ohjelmat tullaan kääntämään. Ensimmäinen vaihe ristikääntäjää rakentaessa on Binutils:n kääntäminen. Onnistuneen Binutils asennuksen jälkeen puretaan MinGW:n header-tiedostoja sisältävä kokoelma. Nämä ovat Windowsille ominaisia header-tiedostoja. Tämän jälkeen asennetaan GCC:n ydinominaisuudet, jotta seuraavan vaiheen, eli MinGW-crt (C-Runtime) tiedostot voidaan kääntää halutulle arkkitehtuurille. Lopuksi asennetaan vielä GCC:n muut puuttuvat komponentit. Asennuksen jälkeen ristikääntäjän tulisi toimia, jos asennukset on tehty oikein.

Tarkemmin tietoa MinGW ristikääntäjän asennuksesta ja kääntämisestä suoraan sen lähdekoodista saa SourceForgen sivuilta [22].

4.5 Docker

VLC tarjoaa vaihtoehtoisen tavan koota koko ohjelman docker-työkalun avulla. Docker on ohjelma, joka suorittaa käyttöjärjestelmätason virtualisointia. Se on käyttöjärjestelmä- ja laitteistoriippumaton, joten se mahdollistaa vaivattoman tavan ohjelmistojen kehittämiseen, kokoamiseen ja ajamiseen. Hieman vastaavana esimerkkinä voisi pitää JVM:ä, jonka avulla ajetaan Java-pohjaisia ohjelmia. [23]

Alla olevasta kuvasta 6 nähdään, millaisia eroja dockerilla ja tavanomaisilla virtualisointityökaluilla on.



Kuva 6: *Docker vs Virtualisointityökalu. [24]*

Docker ajaa konttejaan samassa käyttöjärjestelmässä, kuin mihin se on asennettu. Virtualisointityökalut taas pyörittävät erillisiä virtuaalikoneita, joissa ohjelmia ajetaan vieraskäyttöjärjestelmässä.

Dockerilla on kuvan mukaisesti mahdollista ajaa useita ohjelmia samanaikaisesti lukuisissa konteissa. Tässä säästetään aikaa ja vaivaa, koska uutta käyttöjärjestelmää ei tarvitse erikseen asentaa ja konfiguroida. Usein docker-kontin koko on myös huomattavasti pienempi, kuin kokonaisen virtuaalikoneen käyttäminen. Tätä kokoeroa selittää tietenkin se, että jokaiseen virtuaalikoneeseen joutuu asentamaan koko käyttöjärjestelmän.

4.6 VLC-kääntäminen dockerilla

Kuten jo tiedämme, VLC on riippuvainen hyvin monesta eri palikasta, joten sen kääntäminen tavallisella ristikääntäjälla voi olla hieman haastavaa. Helpottaakseen näiden riippuvuuksien paketointia yhteen, on VLC luonut eri arkkitehtuureille sopivia docker-imageja. Näihin docker-imageihin VLC on esiasentanut toimivan työkalukokoelman, jolla käännöksen tulisi onnistua suhteellisen helposti.

Ainoa asia, josta käyttäjän tulee huolehtia, on VLC:n lähdekoodikansion kopiointi konttiin, jossa se voidaan kääntää tätä varten tehdyllä käännös-komentosarjalla. Käännös-skriptin tehtävänä on linkittää ja kääntää kaikki VLC:ä tarvittavat kirjastot sekä komponentit yhteen. Tässä tulee kuitenkin huomioida se, että vaikka tiedostot kopioidaan konttiin, niin käännöksen tulokset ovat kuitenkin saatavilla samassa paikassa, josta lähdekoodit on kopioitu. Huomioitavaa on myös se, että kun kontti suljetaan, niin konttiin kopioidut tiedostot eivät jää talteen, vaan uudelleenkäynnistyksen yhteydessä ne täytyy kopioida uudelleen.

4.7 Toteutettu työkalukokoelma

Tämän työn tapauksessa ristikääntäjä tuli ensiksi kääntää itse, koska VLC:n kolmannen osapuolen kirjastoilla on riippuvuuksia GCC-8 version kanssa ja käyttöjärjestelmän esikonfiguroidut ladattavat ristikääntäjäpaketit sisältävät vain joko GCC-6 tai 7 version. Ristikääntäjäksi valikoitui MinGW, koska sillä pystyy tekemään ristikäännöksiä Unix-pohjaiselta koneelta Windows-ympäristöön. VLC lisäksi suosittelee ristikäännöstä natiivin sijaan.

VLC:ssä on myös vaihtoehtoinen ratkaisu koko sovelluksen kääntämiseen docker-imagen muodossa. Docker-imagessa oli valmiiksi asennettu toimiva työkalukokoelma, joka toimii vastaavalla tavalla kuin pakettienhallinasta ladattu kääntäjä. Lisäksi VLC tarjoaa valmiin build.sh -skriptin, jonka avulla koko käännösprosessi tapahtuu automaattisesti. Konttia käyttäessä tarvitsee kääntämisen jälkeen vain päättää, millaiseen formaattiin haluaa VLC:n pakata. Vaihtoehtoja pakkaamiselle ovat NSIS (Nullsoft Scriptable Install System) -asennuspaketin tekeminen. NSIS on ohjelma, jolla voidaan tehdä Windows pohjaisia asennuspaketteja. Toisena vaihtoehtona pakkaamiselle on perinteinen kansio, johon vain puretaan kaikki käännöksessä syntyvät tiedostot. Lisäksi näistä tiedostoista voi luoda zip- tai 7zip-pakatun tiedoston, jonka purkamalla saa saman kansion, josta ohjelmaa voi ajaa.

Molemmilla tavoilla VLC:n kääntäneenä suosittelen docker-imagen käyttämistä, vaikka docker olisikin uusi tuttavuus. Perinteisessä kääntämisessä ainakin työn suoritushetkellä törmää hyvinkin moniin ongelmiin, joiden korjaamisessa menee huomattavasti

enemmän aikaa, kuin dockerin opetteluun. Docker-imagen avulla kääntäminen onnistuu helposti, eikä omaa säätöä juuri tarvita. Lopuksi kannattaa NSIS:n avulla tehdä asennuspaketti, jolla ristikäännös on helppo asentaa halutulle työasemalle.

5 TOTEUTUS

Tässä työssä tehdyt muutokset tehdään järjestelmään, jolla hallinnoidaan Suomen tieverkostoa. Tämän tieliikenteen ohjausjärjestelmän tarkoituksena on nopeuttaa tiedonkulkua tienkäyttäjille ja helpottaa liikennepäivystäjien sekä muiden aiheeseen liittyvien sidosryhmien työtä. Järjestelmää on kehitetty pitkään ja se on ollut tuotantokäytössä muutaman vuoden. Järjestelmä on kuitenkin vielä aktiivisessa kehitysvaiheessa ja siihen pyritään lisäämään jatkuvasti uusia ominaisuuksia. Näiden uusien ominaisuuksien sekä järjestelmän kasvamisen myötä viimeaikoina on törmätty ongelmiin suorituskyvyn kanssa. Esimerkiksi vasta viimeaikoina Suomen tieliikenteeseen on lisätty Full HD-tason kameroita, jotka on kytketty järjestelmään. Kameroiden videovuon katselu reaaliaikaisesti on PC:lle ja järjestelmälle hyvin kuormittavaa, eikä järjestelmässä oltu varauduttu tälläiseen aiemmin.

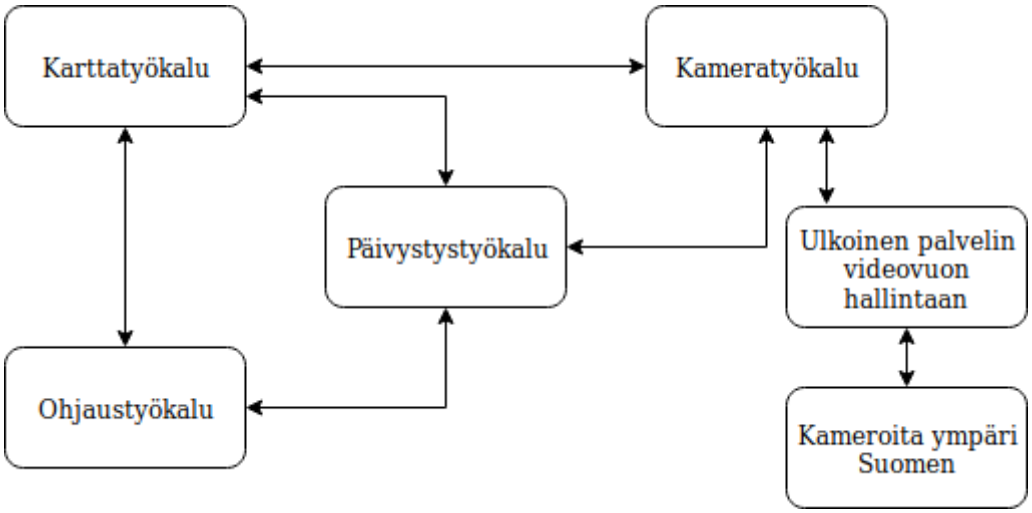
Kyseisten videovuo:n katseleminen järjestelmässä on kuitenkin ollut jo pitkään mahdollista. Toteutettuna on siis jo kameralta videovuon saaminen järjestelmään, kommunikointi palvelinpuolen kanssa sekä VLC-instanssin luominen, jossa videovuon purkaminen lopulta tapahtuu. Lisäksi aiemmin kameroilta tulevaa videovuota on jo pyöritetty GPU:lla hyödyntäen Direct3D:n tarjoamaa ohjelmointirajapintaa, mutta ilman työssä toteutettavaa kuormituksen tasausta. Tämä tarkoittaa sitä, että kaikki videovuon purkamiseen tarvittava prosessointi suoritetaan vain yhdellä GPU:lla. Tähän pyritään nyt tekemään muutos, jotta kuormitusta pystyttäisiin jakamaan usealle GPU:lle, jolloin kaikki työ ei olisi vain yhden GPU:n vastuulla.

Työn toteuttamishetkellä Vlc.DotNet:ssä tai LibVLC:ssä ei ollut mahdollista valita GPU:ta, jolla videovuo purettaisiin. GPU:n valinta tapahtui tällöin automaattisesti, jolloin oletusarvona käytettiin aina sitä GPU:ta, johon monitori oli kytkettynä.

Tätä varten täytyi siis ensin löytää keino, kuinka VLC voidaan kääntää suoraan sen lähdekoodeista. Tällä tavalla sen lähdekoodia on mahdollista puukottaa ja siten lisätä tarvittavat ominaisuudet, jotta kuormituksen tasaaminen mahdollistuu. Lisäksi käyttöliittymän puolelle täytyy toteuttaa oma riippumaton komponentti. Komponentin tehtävänä on pitää kirjaa, kuinka monta videovuota on kullakin GPU:lla pyörimässä aina uuden VLC-instanssin luonnin aikana. Tämän perusteella uuden instanssin luonnin yhteydessä voitaisiin LibVLC:lle antaa aina parametrinä haluttu GPU seuraavan videovuon purkamiseen.

5.1 Järjestelmän arkkitehtuuri

Järjestelmä koostuu neljästä suuresta komponentista, joista muutoksia tehdään vain yhteen työkaluista. Toteutettavilla muutoksilla ei myöskään ole riippuvuuksia muiden komponenttien kanssa. Alla olevassa kuvassa 7 on todella korkean tason yleisnäkymä järjestelmän arkkitehtuurista.



Kuva 7: *Korkean tason arkkitehtuuri.*

Kuvan mukaisesti suuria osakokonaisuuksia järjestelmästä löytyy neljä, joiden alta löytyy toki monia laajojakin kokonaisuuksia, mutta karkeasti lajiteltuna järjestelmä voidaan jakaa näihin neljään komponenttiin. Komponentit kommunikoivat keskenään hyvinkin intensiivisesti, lähettäen toisilleen heidän tarvitsemiaan tietoja. Jokainen komponetti kuuntelee niille ennalta asetettuja topikkeja ja jonoja, joiden avulla kommunikointi tapahtuu. Yksinkertainen esimerkki voisi olla, että karttatyökalu pyytää päivystystyökalulta jonkin tiedotteen tietoja tietyn topicin tai jonon välityksellä. Päivystystyökalu vastaanottaa viestin ja hakee tarvittavan tiedon omasta tietokannastaan ja lähettää sitten tiedon eteenpäin jälleen jonkin tietyn topikin välityksellä, jota kameratyökalu kuuntelee. Vaihtoehtoisesti työkalut voivat tiedottaa muita työkaluja, että se on vastaanottanut tärkeätä informaatiota, joka muita saattaisi kiinnostaa. Tälläisessä tilanteessa muut työkalut eivät siis pyydä tietoja, vaan he vain vastaanottavat jotain muuttunutta tietoa, joka vaikuttaa myös heidän toimintoihinsa. Vastaanotettuaan tälläisen tiedon, työkalut suorittavat sille jonkinlaisen ennalta määritellyn toimenpiteen, kuten uuden muuttuneen arvon päivittämisen tietokantaan tai esittää saadun tiedon käyttäjälle käyttöliittymän kautta.

Järjestelmän tärkein työkalu on päivystystyökalu, jolla hallinnoidaan järjestelmän kokonaisuutta ja jonka kautta useissa tapauksissa kaikki data välitetään työkalulta toiselle. Tietyissä tapauksissa ei kaikkea dataa ole kuitenkaan järkevää välittää eteenpäin päivystystyökalun kautta, koska sen ei tarvitse olla tietoinen kaikesta liikkuvasta datasta. Tällä parannetaan koodin ymmärrettävyyttä ja saavutetaan parempi ja nopeampi tiedonvälitys työkalulta toiselle. Päivystystyökalun täytyy kuitenkin olla tietoinen useasta asiasta ja se on syynä siihen, miksi data välitetään usein sen kautta.

Karttatyökalu on olennainen osa järjestelmää. Sieltä on nähtävillä koko Suomen kartta, johon on piirtyneenä haluttujen suodattimien mukaan joko osittain tai kaikki sen hetkiset aktiiviset hälytykset, tietyöt, kamerat ja monia muita asioita, jotka ovat relevantteja suomen tieliikenteen kannalta. Kartalta voi myös avata kameroita, hälytyksiä ja muita tiedotteita toisiin työkaluhin. Kartalla lisäksi luodaan osa järjestelmään tulevista tiedotteista käsin päivystäjien toimesta.

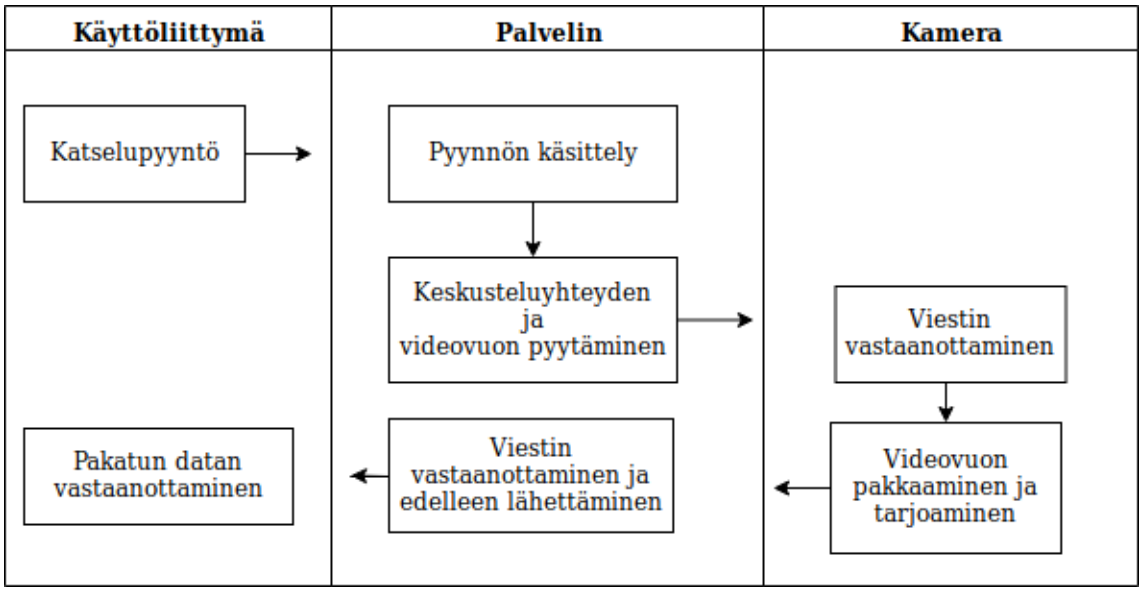
Ohjaustyökalulla voidaan hallinnoida tunneleita ja tiettyjä tieosuuksia, jotka ovat digitalisoituja. Tavalliselle tienkäyttäjälle varmaan konkreettisin esimerkiksi tälläisestä osittain digitalisoituneesta tieosuudesta ovat tiet, joissa on digitaalisia nopeusrajoituksia. Näiden arvoja voidaan muuttaa etänä vain nappia painamalla tämän työkalun avulla. Ohjaustyökalu lähettää lisäksi reaaliaikaista tietoa kyseisiltä tieosuuksilta ja saattaa antaa erilaisia ehdotuksia päivystäjille, että nyt tulisi laskea nopeusrajoitusta jollakin tieosuudella huonojen sääolosuhteiden takia.

Viimeinen suuri komponentti on kameratyökalu. Tähän työkaluun on tarkoitus tehdä muutoksia. Kameratyökalun avulla pystytään seuraamaan Suomen teillä olevista keli- ja tiekameroista videovuota reaaliaikaisesti. Lisäksi pystytään hakemaan tallenteita tietyltä aikajaksolta, sekä ohjaamaan näitä kameroita kääntämällä ja zoomaamalla.

5.2 Videovuon vastaanottaminen järjestelmässä

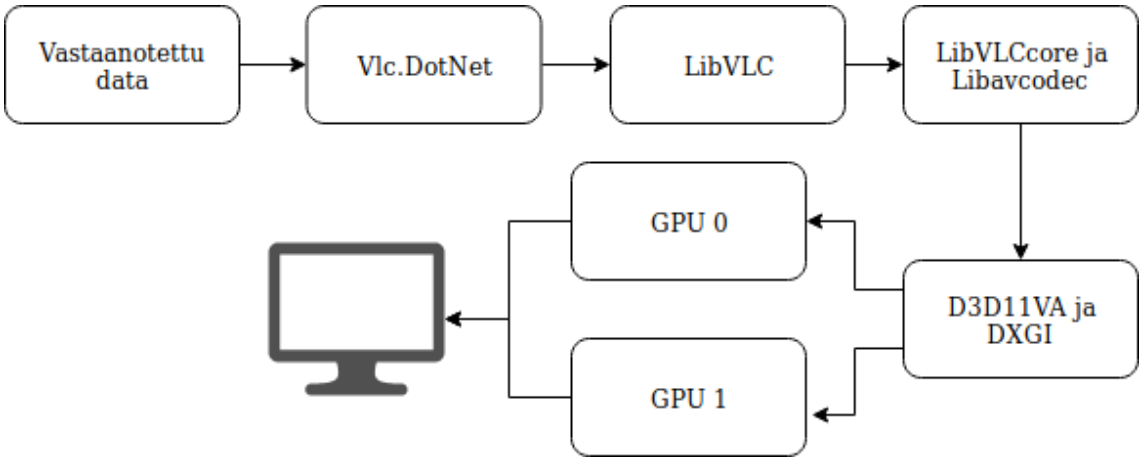
Järjestelmään tuleva videovuo saadaan Suomen tieliikenteessä olevilta keli- ja tiekameroilta. Videovuota toistetaan C#:lla toteutetussa käyttöliittymässä, johon on upotettu Vlc.DotNet niminen NuGet-paketti. NuGet-paketti on Microsoftin kehitysympäristöön tarkoitettu pakettienhallintaohjelma, jonka avulla jostakin ohjelmistopaketista, kuten Vlc.DotNet:stä, voidaan luoda instansseja. Instanssin luomisen jälkeen sen tarjoamia toimintoja voidaan hyödyntää ilman, että sillä on mitään sen suurempia riippuvuuksia sitä käyttävässä ohjelmistossa.

Vlc.DotNet ohjelmistopaketti mahdollistaa videoiden toistamisen hyödyntäen VLC:n LibVLC-kirjastoa. Se on siis eräänlainen wrapperi, jonka avulla LibVLC saadaan integroida osaksi C#-ohjelmaa.
Alla olevan kuvan 8 on tarkoitus kuvata korkealla tasolla, kuinka järjestelmä lähettää pyynnön halukkuudesta seurata jonkin tietyn kameran videovuota.



Kuva 8: *Videovuon pyytäminen käyttöliittymästä.*

Ensin käyttöliittymästä tehdään katselupyyntö palvelimelle jonkin tietyn kameran videovuosta. Tämän jälkeen palvelin käsittelee pyynnön ja avaa keskusteluyhteyden kyseisen kameran ja itsensä välille. Luodessaan keskusteluyhteyttä, palvelin samanaikaisesti varmistaa kameralta, että sillä on kaikki kunnossa. Saatuaan varmistuksen, palvelin pyytää kameraa lähettämään videovuon itselleen. Tällöin data pakataan H.264-formaattiin kameran raudalla ja pakkauksen jälkeen kamera lähettää datan palvelimelle. Vastaanotettuaan datan, palvelin lähettää sen vielä eteenpäin käyttöliittymälle. Periaattessa palvelin toimii järjestelmässä vain tiedon välittäjänä, sekä huolehtii siitä, että molemmissa päissä kaikki on kunnossa, jotta videovuon lähettäminen ja vastaanottaminen on mahdollista. Alla olevassa kuvassa 9 kuvataan karkeasti, kuinka vastaanotettu pakattu data puretaan ja esitetään käyttäjälle.



Kuva 9: *Videovuon purkaminen järjestelmässä.*

Käyttöliittymä vastaanottaa H.264-formaatissa olevan datan. Jotta videovuota voitaisiin katsoa käyttöliittymässä, tulee sen purkaa data ennen kun se voidaan esittää käyttäjälle. Järjestelmän vastaanotettua datan, se siirtää vastuun purkamisesta välittömästi Vlc.DotNet:lle, jonka tehtävänä on luoda VLC-instanssi videovuolle LibVLC-kirjaston avulla.

Tässä vaiheessa suoritus on siirtynyt LibVLC:lle, joka pyytää apua pakatun videovuon purkamiseen LibVLCcorelta, jolla on rajapinta purkamista toteuttavaan itsenäiseen Libavcodec-moduuliin.

VLC-instanssin luomisen yhteydessä järjestelmässä annetaan LibVLC:lle parametreinä ohjeita, jotka se välittää eteenpäin Libavcodec:lle. Näiden parametrien avulla Libavcodec saa tiedon siitä, että kuinka data halutaan purkaa. Videovuota voidaan purkaa useilla eri ohjelmointirajapinnoilla, jotka käyttävät purkamiseen laitteistoa. Vaihtoehtoisesti purkaminen voidaan myös toteuttaa kokonaan CPU:lla, jolloin purkamiseen käytetään ohjelmistopurkajaa.

Tässä työssä ja järjestelmässä purkaminen toteutetaan GPU:lla käyttäen D3D11VA-ohjelmointirajapintaa. Aiemmin data on purettu käyttäen Dxva2-ohjelmointirajapintaa, joka on yksi D3D11VA:n edeltäjistä. Sen avulla ei vain ole mahdollista hyödyntää useata GPU:ta usean videovuon purkamiseen samanaikaisesti, minkä takia halutaan hyödyntää juuri D3D11VA-rajapintaa.

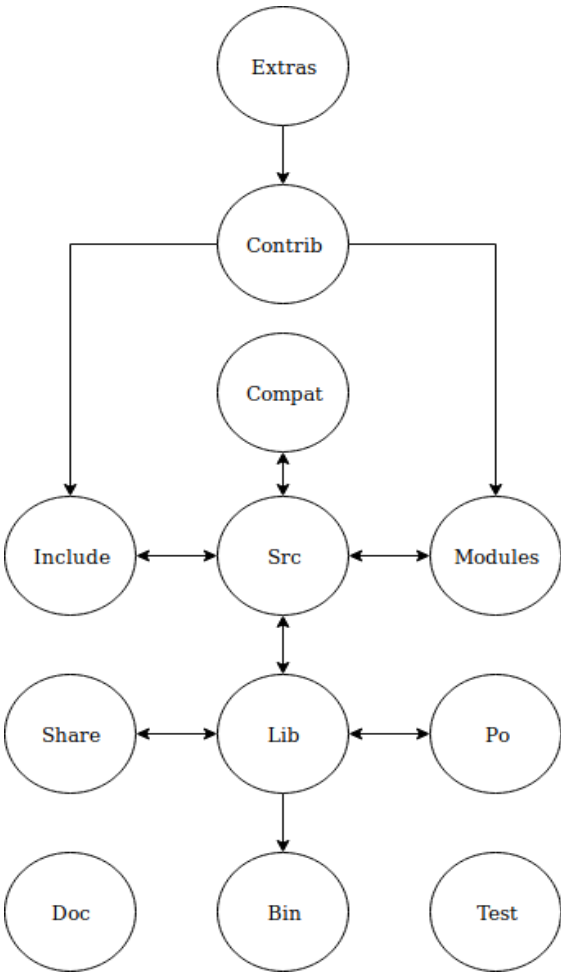
Vanhempien Direct3D-rajapintojen ongelmana on se, että näissä GPU:n tunnistaminen on toteutettu monitoriin kytketyn GPU:n avulla. Eli kaikki muut GPU:t, jotka eivät ole kytkettyinä mihinkään monitoriin, eivät ole käytettävissä. Kun taas D3D11VA:ssa ja sitä uudemmissa versioissa on mahdollista käyttää uudempaa DXGI-rajapintaa, jolla

jokainen tietokoneeseen kytketty GPU on mahdollista enumeroida. Enumeroinnin avulla voidaan DXGI-rajapinnalla valita haluttu GPU suorittamaan sille määrätty tehtävä.

D3D11VA:ta uudempaa versiota ei valittu sen vuoksi, että sellaista ei ollut vielä saatavilla VLC:n lähdekoodissa, jolloin sen olisi joutunut integroimaan kokonaan osaksi VLC:tä alusta alkaen.

5.3 VLC lähdekoodi

VLC:n lähdekoodi jakautuu karkeasti kahteentoista eri palikkaan, joista useat palikat saattavat keskustella keskenään, mutta se on pyritty pitämään mahdollisimman modulaarisena. Alla olevassa kuvassa 10 on näkymä, miten palikat ovat liitoksissa toisiinsa. Kuvassa on ajatuksena se, että ylhäältä lähdettäessä on tarvittavat palikat VLC:n kääntämiseen ja alhaalla Bin:n kohdalla on tarvittavat tiedostot ajettavan .exe-tiedoston luomiseen.



Kuva 10: *VLC lähdekoodi.*

Kaikki VLC:n lähdekoodissa alkaa extras kansiosta, joka tarjoaa VLC:n kääntämiseen ja lopulliseen paketoimiseen erilaisia skriptejä. Porrasta alempana on contrib niminen palikka, jonka tehtävänä on kääntämisen yhteydessä ladata kaikki kolmannen osapuolen kirjastot. Näiden ladattujen kirjastojen pohjalta luodaan include ja modules kansiot. Include sisältää suurimman osan header-tiedostoista, joita VLC käyttää ja modules osa sisältää kaikki VLC:n modulit. Modules on selvästi suurin VLC:n kansoista ja täältä kansiosta löytyy myös libavcodec, johon tarvittavat muutokset olisi tarkoitus toteuttaa.

Src:llä tarkoitetaan LibVLCcorea ja sillä on yhteys moneen paikkaan, kuten moduleihin ja otsikkotiedostoihin. Lisäksi src on liitoksissa compat nimiseen kansioon, jonka tehtävänä on huolehtia ohjelman yhteensopivuudesta alla olevalle laitteelle. Aiemmin on jo mainittu, että LibVLCcore kommunikoi LibVLC:n kanssa, joten kuvassa lib:llä tarkoitetaan tätä kirjastoa. Kirjastolla on yhteys po kansioon, joka tarjoaa kielivalinnan soittimelle. Share kansio tarjoaa yleisiä resurssitiedostoja mm. ohjelman visualisointiin. Tämän jälkeen jäljelle jää vain bin kansio, joka sisältää tarvittavat lähdekoodit ajettavan .exe -tiedoston luomiseen. Test kansio on hieman irrallinen ja sillä on liitoksia aika moneen paikkaan. Nimensä mukaisesti siellä on VLC:n yksikkötestejä. Doc kansiolla tarkoitetaan dokumentaatiota, joten sieltä löytyy kattavasti tietoa järjestelmästä.

5.4 GPU laitteen luominen

DXGI-rajapinnan kautta on mahdollista valita adapteri kahdella eri tavalla Direct 3D11 ja sen jälkeisillä versioilla. Adapterilla tarkoitetaan GPU:ta jolla ohjelmaa tullaan pyörittämään. Adapterin valitsemiseen tarvitaan IDXGIFactory-rajapinta, jota käytetään GPU:n enumerointiin sekä swapchainin luomiseen. Tässä tapauksessa ollaan kiinnostuneita vain GPU:n enumeroinnista.

Ensimmäisessä tavassa luodaan ensin virtuaalinen laite, jolta voidaan kysyä tämän periyttäjää, jonka rajapinta luo automaattisesti. Alla olevassa Ohjelmassa 3 on esitelty, kuinka factory voidaan hakea GetParent toiminnon avulla.

```
IDXGIDevice * pDXGIDevice = nullptr;
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice),
(void **)&pDXGIDevice);

IDXGIAdapter * pDXGIAdapter = nullptr;
hr = pDXGIDevice->GetAdapter( &pDXGIAdapter );

IDXGIFactory * pIDXGIFactory = nullptr;
pDXGIAdapter->GetParent(__uuidof(IDXGIFactory), (void
**)&pIDXGIFactory);
```

Ohjelma 3: *Factoryn saaminen. [25]*

Ensiksi luodaan IDXGIDevice-olio, jonka rajapinta on suunniteltu käytettäväksi silloin, kun tarvitaan pääsy muihin DXGI-objekteihin. Se on hyödyllinen rajapinta erityisesti silloin, kun ohjelma tarvitsee tiettyjä DXGI-objektien toimintoja, joita ei voi määritellä suoraan Direct3D-rajapinnan kautta. IDXGIDevice-olion avulla voidaan luoda IDXGIAdapter-rajapinta, jonka kautta puolestaan voidaan hakea tietoa näytönohjaimista, sekä käyttää tätä oliota Direct3D-laitteen luomiseen. Ohjelma 1:n esimerkissä adapteriksi valitaan oletuksena se näytönohjain, johon monitori on kytkettynä kiinni, mutta tätä kautta on mahdollista päästä käsiksi IDXGIFactoryyn, joka mahdollistaa kaikkien oikein kytkettyjen näytönohjaimien enumeroinnin. Kun pIDXGIFactory-olio on luotu onnistuneesti, voidaan ohjelman 4 mukaisesti hakea ja enumeroida kaikki näytönohjaimet.

```
UINT i = 0;
IDXGIAdapter* pAdapter = nullptr;
std::vector<IDXGIAdapter*> adapterList;
while(pIDXGIFactory->EnumAdapters(i, &pAdapter) !=
DXGI_ERROR_NOT_FOUND)
{
  DXGI_ADAPTER_DESC aDesc;
  adapter->GetDesc(&aDesc);
  std::wstring adapterInfo = "Adapter: ";
  adapterInfo += aDesc.Description;
  std::cout << adapterInfo.toString() << std::endl;
  adapterList.push_back(adapter);
  ++i;
}
```

Ohjelma 4: *Adapterien listaaminen.*

Ohjelmassa 4 listataan kaikki laitteistolta löytyvät adapterit while-silmukassa ja niitä etsitään niin kauan, kunnes DXGI palauttaa viestin ”DXGI_ERROR_NOT_FOUND”. Silmukan sisällä tulostetaan debuggaus-mielessä adapterin tiedot konsoliin DXGI_ADAPTER_DESC-tietueen avulla. Lopuksi adapter-osoitin tallennetaan std::vector-tietorakenteeseen, mutta tässä tapauksessa mikä tahansa muukin tietorakenne kävisi yhtälailla. Alla Ohjelmassa 5 on esillä yllä mainittu adapterin tietue.

```
typedef struct DXGI_ADAPTER_DESC
{
  WCHAR  Description[128];
  UINT   VendorId;
  UINT   DeviceId;
  UINT   SubSysId;
  UINT   Revision;
  SIZE_T DedicatedVideoMemory;
  SIZE_T DedicatedSystemMemory;
  SIZE_T SharedSystemMemory;
  LUID   AdapterLuid;
};
```

Ohjelma 5: *DXGI_ADAPTER_DESC tietue*

Tämä tietue sisältää kaiken tarvittavan tiedon adaptereista, jota voi käyttää hyödykseen. Descriptionista saa tulostettua ohjaimen nimen. VendorID kuvastaa laitteen myyjän yksilöllistä id:tä ja DeviceID heidän valmistamansa laitteen id:tä. SubSysId on periaatteessa toinen osa DeviceID:lle, eli, jos laitteessa on kaksi samaa GPU:ta, näillä kaikilla kolmella olisi täysin identtiset arvot näissä kentissä. Revisionilla tarkoitetaan sitä, että mikä versio tästä tuotteesta on kyseessä. Joissain tapauksissa valmistajat saattavat haluta tehdä pieniä päivityksiä saman ohjaimen seuraavaan valmistuserään, jolloin he päivittävät samalla myös Revision-numeron. DedicatedVideoMemory ilmaisee tavuina laitteelle omistettua videomuistia, jota ei ole jaettu CPU:lle. DedicatedSystemMemory kertoo tavuina ohjaimen järjestelmämuistin koon, eikä sekään ole jaettu CPU:lle. SharedSystemMemory on vastakohta näille kahdelle, se kertoo maksimiarvon, kuinka paljon ohjain voi kuluttaa muistia käytön aikana. AdapterLuid tarjoaa yksilöllisen arvon tälle laitteelle.

Nyt DXGI-rajapinnan avulla on saatu enumeroitua adapterit ja haettua näistä haluttuja tietoja DXGI_ADAPTER_DESC-structin avulla. Lisäksi kaikki löydetyt adapterit on saatu talteen tietosäiliöön, josta ne on helppo hakea. Sen jälkeen tulee luoda Direct3D-objekti, jolla itse ohjelmaa lopulta ajetaan. Ohjelmassa 6 esitellään, kuinka luodaan Direct3D-objekti.

```
HRESULT D3D11CreateDevice(
  IDXGIAdapter            *pAdapter,
  D3D_DRIVER_TYPE         DriverType,
  HMODULE                 Software,
  UINT                    Flags,
  CONST D3D_FEATURE_LEVEL *pFeatureLevels,
  UINT                    FeatureLevels,
  UINT                    SDKVersion,
  ID3D11Device            **ppDevice,
  D3D_FEATURE_LEVEL       *pFeatureLevel,
  ID3D11DeviceContext     **ppImmediateContext);
```

Ohjelma 6: *Direct 3D-laitteen luominen.*

HRESULT on Windowsin datatyyppi, joka palauttaa arvon laitteen luomisen onnistumisesta tai epäonnistumisesta. Tässä voi syöttää adapterin kohdalle arvoksi halutun adapterin säiliöstä, jonka aikaisemmin loimme. Tärkeä asia, joka tulee huomioida, kun itse haluaa valita adapterin käytettäväksi on D3D_DRIVER_TYPE kohta. Microsoftin dokumentaatiossa on pienellä tekstillä ilmaistu, että tälläisessä tapauksessa, jossa ei käytetä oletusadapteria, tulee sen arvoksi vaihtaa oletuksesta D3D_DRIVER_TYPE_HARDWARE.

Software kohtaan voi antaa parametriksi null. Sen avulla voidaan valita jokin tietty ohjelmistopurkaja käytettäväksi. Flagsillä voidaan antaa erilaisia lippuja, joiden avulla

voi määrittää ohjelmaa ajettavaksi vain yhdellä säikeellä, sekä antaa erilaisia debuggaus-parametrejä. Kohtaan D3D_FEATURE_LEVELS voidaan antaa osoitin, jossa on määritelty, että minkä Direct3D-version ominaisuuksia voi maksimissaan käyttää. Tähän kannattaa syöttää null, koska oletuksena funktio osaa valita aina parhaimman arvon käytettäväksi. Seuraavalle UINT FeatureLevels muuttujalle voidaan antaa arvo, että kuinka monen Direct3D-version ominaisuuksia ollaan käyttämässä. SDKVersion (Software Development Kit), tarkoittaa että mitä ohjelmistokehityspakkausta käytetään. Lopulta ID3D11Device palauttaa osoittimen tähän luotavaan objektiin ja D3D_FEATURE_LEVEL palauttaa valitun Direct3D tason. ID3D11DeviceContext palauttaa osoittimen, jonka rajapinta kuvastaa laitteen kontekstia, joka luo kaikki laitteen renderöinti-komennot. [25]

5.5 VLC muutokset

Ensiksi tuli löytää keino, kuinka VLC:lle pystyy ulkopuolelta lähettämään omia parametrejä. Oman parametrin määrittäminen on kohtalaisen helppoa. Täytyy vain lisätä Ohjelman 7 mukainen parametri avcodecille. Kaikki avcodecin tukemat parametrit löytyvät avcodec.c tiedostosta. Määritykset, kuten parametrien kuvaamisessa käytetyt tekstit taas kirjoitetaan avcodecin header-tiedostoon.

```
add_integer ( "avcodec-hwaccel_device", 0, HWACCEL_TEXT,
         HWACCEL_LONGTEXT, false)
   change_safe ()
   change_integer_list( nloopf_list )
```

Ohjelma 7: *Parametrin lisääminen avcodeciin.*

Add_Integer-funktio ottaa vastaan ensin nimen, tämän jälkeen oletuksen kaksi määriteltyä tekstiä kyseiselle parametrille, sekä tiedon onko tämä erityinen parametri, joka tässä tapauksessa on asetettu epätodeksi. Tämän jälkeen change_safe():lla tarkoitetaan sitä, että kyseistä parametriä voi turvallisesti muuttaa toiseksi oletusarvosta. Lopuksi parametrille vielä annetaan lista, mitä arvoja kyseiselle parametrille voidaan antaa.

Kun parametri saatiin lähetettyä VLC:lle halutulla tavalla käyttöliittymästä, tuli VLC:n lähdekoodia muuttaa siten, että vastaanotetulla parametrillä pystyttäisiin valitsemaan haluttu GPU videovuon purkamiseen. Tässä vaiheessa oli löydettävä lähdekoodista kohta, jossa D3D11VA:ta käytetään videon purkamiseen. Se löytyy d3d11_fmt.c-tiedostosta, johon suurin osa lähdekoodin muutoksista tehtiin. Sille oli lisättävä tuki uusimpien dxgi_1.5 ja 1.6-versioiden käyttöön. Lisäksi tiedoston oli pystyttävä vastaanottamaan lähetetty parametri. Tämä tapahtuu ohjelman 8 mukaisesti.

```
// Fetch hwaccel_device, default = 0
 int hwaccel_device;
 hwaccel_device = var_CreateGetInteger( obj,
 "avcodec-hwaccel_device" );
```

Ohjelma 8: *Parametrin vastaanottaminen.*

Vastaanotetun parametrin perusteella voidaan valita laite, jolla purkaminen tulee tapahtumaan. Ensiksi vain luodaan IDXGIFactory, jonka avulla enumeroidaan kaikki PC:n kytkety GPU:t. Enumeroinnin jälkeen kyseisellä hwaccel_device-muuttujan avulla valitaan haluttu GPU käytetettäväksi, josta luodaan D3DDevice Ohjelman 6 funktion avulla. Lopuksi vielä ohjelman 9 mukaisesti kirjoitetaan debug-lokiin tieto siitä, mikä laite on valittu ja että sen valinta on ollut onnistunut.

```
if(SUCCEEDED(hr))
{
    msg_Dbg(obj, "Created the D3D11 device type%d level %x.",
    D3D_DRIVER_TYPE_UNKNOWN,out>feature_level);
    D3D11_GetDriverVersion( obj, out );
    break;
}
```

Ohjelma 9: *Debug-lokiin kirjoittaminen.*

Jos jostain syystä laitteen valinnassa on ollut ongelmaa, niin kaikki aiemmat laitteen valitsemessa käytetyt resurssit vapautetaan ja videovuota aletaan purkaa ohjelmistopurkajan avulla.

Loppupeleissä muutoksia ei VLC:n tarvinnut paljoa tehdä, .patch-tiedoston mukaan n. 200 riviä. Suurin osa tämän vaiheen toteuttamiseen kuluvasta ajasta meni ohjelman debuggaamiseen, missä kohtaa lähdekoodia toteutaan GPU:n valinta ja missä vaiheessa ohjelman suoritus siirtyy avcodecille. Debuggaus VLC:ssä onnistuu lisäämällä koodin väliin debuglokitus viestejä, esimerkiksi Ohjelman 9 tavalla.

5.6 Käyttöliittymä toteutus

Käyttöliittymässä suoritettava komponetti pitää kirjaa jatkuvasti avoinna olevista kameroista ja jakaa sen perusteella kuormaa GPU:n kesken. Kameratyökalussa on mahdollisuus ylläpitää useita eri välilehtiä, joissa jokaisessa voi samanaikaisesti olla 12 kameraa avoinna, mutta vain yksi välilehti voi olla auki kerrallaan. Kameroita pystyy myös sulkemaan joko ryppäissä yhdellä painalluksella tai yksi kerrallaan. Järjestelmä

mahdollistaa myös saman kameran avaamisen useaan kertaan, joten komponentin pitää pystyä tunnistamaan jollain tavalla jokainen avattu kamera erikseen.

Näiden rajoitusten perusteella päädyttiin ratkaisuun, että jokaisen kamera-avauksen yhteydessä jokaiselle kameralle luodaan komponenttia varten oma uniikki ID:nsa, joka lisätään kameran tietoihin ajonaikaisesti. Lisäksi komponentin pitäisi pystyä tunnistamaan, mihin välilehteen kamera avataan, jotta se pystyy välilehtikohtaisesti jakamaan kuormaa. Tästä syystä päädyttiin tietojen säilönnässä ohjelman 10 tyyliseen tietueeseen, jota pidetään C#:n List<GpuCameraLoadBalance> tietorakenteessa yllä.

```
public struct GpuCameraLoadBalance
        {
            public int TabIndex;
            public string  GpuIndex;
            public Guid CameraId;

            public GpuCameraLoadBalance(int cameraTabId,
                                        Guid kcameraId,
                                        string gpuDevId)
            {
                TabIndex = cameraTabId;
                CameraId = cameraId;
                GpuIndex = gpuDevId;
            }
        }
```

Ohjelma 10: *Kamera-tietue.*

Tietorakenteen avulla pystytään helposti hakemaan tietoja, mitä jollakin tietyllä välilehdellä on. Riittää, kun suodatetaan listalta kaikki kamerat, joilla on tämän välilehden numero tallennettuna. Tämän jälkeen täytyy enää tehdä vertailu, kumpaa GpuIndexiä välilehdeltä löytyy enemmän. Vertailun perusteella annetaan VLC:lle sen GPU:n numero, joita oli vähemmän ajossa tällä kyseisellä välilehdellä. Tämän jälkeen kamera tallennetaan listaan näillä tiedoilla ja sille luodaan samalla oma uniikki ID, jotta lista pysyy tasapainotettuna. Kameroiden poistaminen listalta on myös hyvin yksinkertaista, ei tarvitse kuin etsiä kyseisen, poistettavan kameran arvo CameraId:n avulla ja siten voidaan poistaa kyseinen olio listalta.

6 TULOKSET

Tuloksia on kerätty Tampereen liikennekeskuksessa. Testaukset tehtiin liikennekeskuksessa olevalla testityöasemalla. Testityöasema vastaa raudaltaan hyvin pitkälti tuotantokäytössä olevia työasemia. Ainoana erona näiden työasemien välillä on se, että testityöaseman raudan osalta sen komponentit ovat hieman vanhempia versioita tuotantokäytössä oleviin työasemiin verratuina.

Olennaisimmat tiedot testityöasemasta ovat:

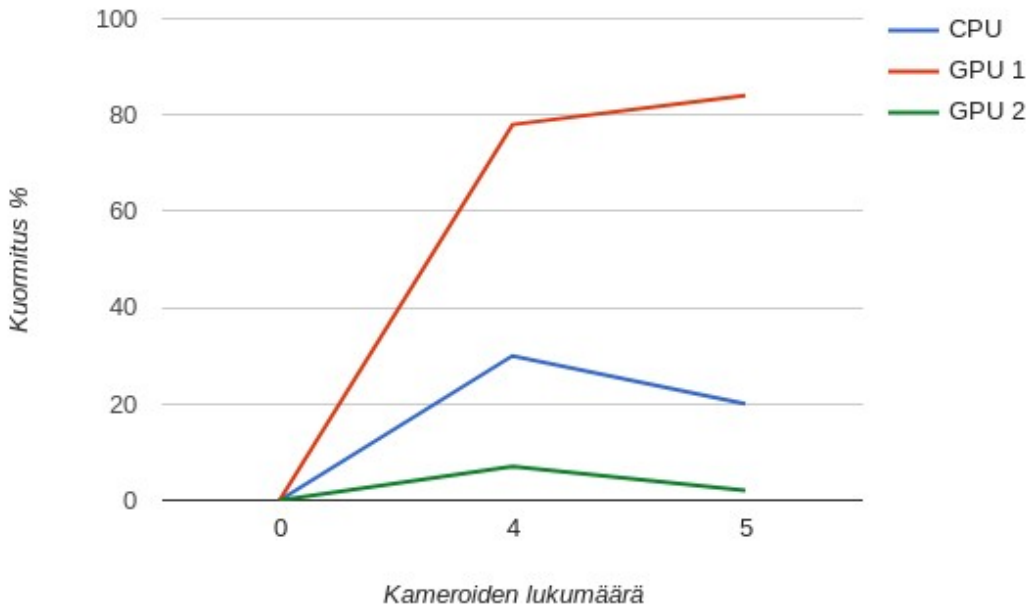
- 2x GPU: AMD FirePro™ W7000
- CPU: Intel® Xeon® E5-1620 v2 neliytiminen HT, 3,7 GHz:n Turbo
- RAM: 8GB 1866MHz DDR3 ECC RDIMM
- 6x Monitori: Dell UltraSharp U3014
- Käyttöjärjestelmä: Windows 7

Testauksessa kerättiin tietoja CPU:n ja GPU:n käytöstä erilaisilla avoinna olevilla kameramäärillä. Testauksessa käytettiin järjestelmässä tällä hetkellä kaikkein korkealaatuisinta kuvaa tarjoavia kameroita. Nämä kamerat lähettävät Full HD-tason kuvaa järjestelmälle purettavaksi. Testidataa kerättiin HWMonitorin avulla, koska testityöasemalla olevalla Windows 7-käyttöjärjestelmällä ei ole vielä mahdollista seurata GPU:n kuormitusta. HWMonitor on laitteiston valvontaohjelma, jonka avulla PC:n laitteiston erilaisia arvoja voidaan monitoroida.

Testauksia tehtiin aiemmalle, vain yhtä GPU:ta hyödyntävälle versiolle. Uudelle tässä työssä rakennettua moni-GPU:ta hyödyntävälle versiolle, sekä pelkästään CPU:ta videovuon purkaamiseen käytettävälle versiolle. Testauksessa huomioitiin se, että järjestelmän muut komponentit ovat ajossa samanaikaisesti, jolloin pystytään simuloimaan mahdollisimman aitoa työskentelytilannetta. Testausta tehdessä koitettiin pyrkiä mahdollisimman autenttiseen järjestelmäkäyttäjän työskentelykäyttäymiseen. Eli testauksessa käytettiin järjestelmän muita osia, kuten niitä käytetään joka päiväisessä työskentelyssäkin. Muista järjestelmän komponenteista on huomioitavaa se, että mikään näistä ei käytä laitteistokiihdytystä hyväkseen, vaan jokainen muu komponentti käyttää toimintaansa CPU:ta, jolloin kaikki testituloksissa ilmenevä GPU:n kuormitus tulee kameratyökalulta.

6.1 Yhden GPU:n versio

Alkuperäisessä, laitteistokiihdytykseen yhtä GPU:ta hyödyntävässä versiossa kaikki videovuon purkamisesta syntyvä kuorma suoritettiin yhdellä GPU:lla. Alla olevassa kuvassa 11 on nähtävillä käyrät molemmille GPU:lle ja CPU:lle. Käyriltä on nähtävissä, minkälainen kuormitus näille syntyi eri kameramäärillä.



Kuva 11: *Yksi kiihdytin käytössä (GPU 1).*

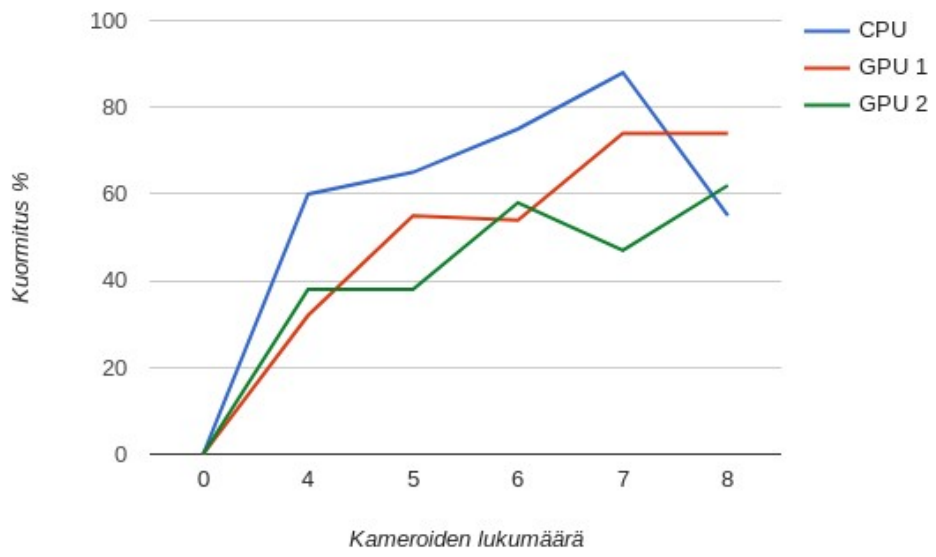
Kuvassa sininen kuvaa CPU:ta, punainen ensimmäistä GPU:ta ja vihreä toisen GPU:n kuormitusta. Graafista ilmenee selvästi, että ensimmäiseltä GPU:lta loppuu kyky prosessoida vastaanottamaansa dataa jo neljännen HD-kameran jälkeen. Tuloksia ei listattu suuremmilla kameramäärillä, koska kaikki videovuot alkoivat jo viidennen samanaikaisesti avoinna olevan kameran kohdalla pätkimään niin pahasti, ettei niiden katsominen ollut enää mahdollista. Kuvataajuus oli noin 30 fps (Frame Per Second) luokkaa ensimmäisellä neljällä kameralla, jonka jälkeen se tippui alta 10 fps.

Huomioitava havainto on CPU:n kuormitus, joka lähtee laskuun viidennen kameran kohdalla. Todennäköisenä syynä tähän on se, että GPU pyöri jo neljännen kameran kohdalla maksimiteholla, jolloin CPU ei pystynyt yksinkertaisesti lähettämään sille enempää dataa prosessoitavaksi. Lisäksi viidennen kameran tulosten kirjaamisen yhteydessä prosessoitavan datan tyyppi oli mahdollisesti poikkeavaa verrattuna neljän kameran dataan. Eroavaaisuus piilee siinä, että kyseisellä hetkellä testikameroiden

tieosuuksilla ei näkynyt autoja, jotka tuovat aina pienen lisänsä prosessoitavaan dataan, mikä kasvattaa CPU:n käyttöastetta. Tätä myös tutkittiin testausvaiheessa ja ilmeni, että mitä enemmän trafiikkia kameralla on, sitä suuremmaksi CPU:n kuormitus kasvaa. Lisäksi huomioitavaa on, että myös toisella GPU:lla oli pientä kuormitusta havaittavissa, ilmeisesti käyttöjärjestelmä oli tietoinen toisen GPU:n olemassaolosta ja osasi informoida tämän laitteistopurkajalle, joka käytti toista GPU:ta ainakin osittain hyödykseen.

6.2 Moni-GPU versio

Moni-GPU:ta hyödyntävässä versiossa kaikki videovuo:n purkamisesta syntyvä kuormitus jaetaan kahden GPU:n kesken. Alla olevassa kuvassa 12 on nähtävillä testauksesta saatuja tuloksia.



Kuva 12: Purkaminen hyödyntäen kahta GPU:ta.

Kuvan käyrien värit vastaavat kuvan 11 värimaailmaa, eli sininen esittää CPU:ta, punainen ensimmäistä GPU:ta ja vihreä toista.

Saadut tulokset olivat hieman yllättäviä. Yllätyksenä tuli se, kuinka suureksi CPU:n kuormitus muuttui verrattuna siihen, kun kuorma puretaan vain yhdellä GPU:lla. Syynä tähän on ilmeisesti se, että nyt CPU joutui kommunikoimaan kahden eri GPU:n kanssa, mikä ilmeisesti vaikuttaa huomattavasti CPU:n kuormitukseen.

Kuormaa kuitenkin saatiin jaettua melko tasaisesti ja vielä seitsemännen kameran kohdalla pystyttiin kohtalaisen hyvin seuraamaan jokaista videovuota ja näissä oli kuvataajuus noin 27 fps luokkaa. Nähtävissä oli kyllä jo seitsemännen kohdalla pientä jäätymistä, joka ilmeni kuvataajudeen pienentymisenä, mutta se ei ollut vielä häiritsevää tai haittavaa. Kahdeksannen kohdalla CPU selvästi ylikuormittui, ja tälle on olemassa oma terminsä, throtlaus, joka tarkoittaa sitä, että CPU ryhtyy kasvattamaan tarkoituksenmukaisesti aikaa, joka sillä kuluu yhden konekäskyn suorittamiseen. Tässä vaiheessa myös kuvataajus tippui huomattavasti, noin 15 fps. Throtlaamisella CPU pyrkii siihen, että ylikuormittunutta tai kuumennettua konetta voidaan jäähdyttää ja saada se jälleen suorituskykyiseksi.

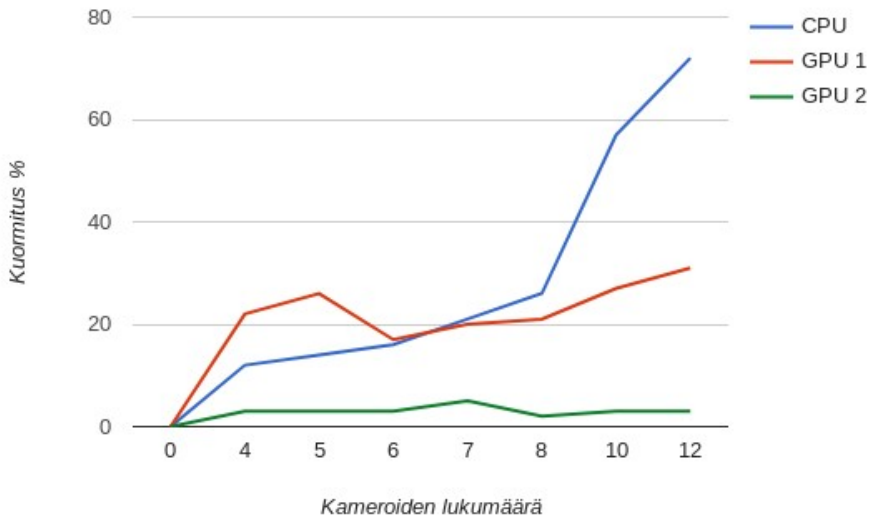
Järjestelmää kuitenkin pystyi käyttämään vielä normaalilla tavalla jokaisen vaihtoehdon kohdalla. Seitsemänen kohdalla ilmeni pieniä jäätymishetkiä järjestelmän toisten CPU:ta suuresti käyttävien komponenttien osalta, kun niitä yritti käyttää samanaikaisesti. Tosin avoinna olevissa kameroissa sattui juuri kyseisellä hetkellä olemaan suurta trafiikkia, mikä osaltaan saattoi vaikuttaa asiaan.

Tulokset moni-GPU:ta hyödyntävässä versiossa olivat siis hieman poikkeavat verrattuna yhtä GPU:ta hyödyntävään versioon, koska tässä tapauksessa CPU oli se, jolla ei riittänyt enää tehoa näin monen videovuon pyörittämiseen.

6.3 CPU versio

Testidataa CPU:ta hyödyntävästä versiosta otettiin vain tätä diplomityötä varten. Oletus oli, että CPU:lla pystyisi purkamaan maksimissaan kahta tai kolmea korkealaatuista videovuota kerrallaan. Purkamisessa käytettiin lisäksi erilaista purkajaa nimeltä VA-API (Video Acceleration API), jonka avulla purkaminen suoritetaan ohjelmallisesti CPU:ta käyttäen, eikä sitä siirretä laitteelle prosessoitavaksi.

Oletus lähti siitä, että kukaan projektissa ei ollut koskaan ajatellutkaan, että videovuo:ta voisi purkaa CPU:n avulla. Tälläinen GPU:lla toteutettava videovuon purkamisperintö on jäänyt järjestelmään jo niiltä ajoilta, kun järjestelmää aloitettiin rakentamaan vuosia sitten. Tosin siihen aikaan järjestelmään ei ollut liitettynä yhtäkään HD-tason kameraa, jolloin on varmaankin ollut ihan perusteltua purkaa nämä heikompilaatuiset videovuot hyödyntäen GPU:ta. Alla olevassa kuvassa 13 on nähtävillä mittaustulokset, joita CPU:ta hyödyntävällä versiolla saatiin.



Kuva 13: *Purkaminen CPU:lla.*

Tulokset olivat kerrassaan pöyristyttäviä. CPU:lla pystyi helposti purkamaan kaikkia järjestelmän maksimimääräksi määriteltyä 12 videovuota. Lisäksi jokainen videovuo pyöri noin 30 fps:llä. CPU:n kuormitus oli myös todella pientä vielä kahdeksannen kameran kohdalla, jonka jälkeen sen kuormitus kasvoi huomattavasti nopeammin. Mutta vielä 12:nnen kameran kohdalla kuormitus ei ollut niin suurta, kuin moni-GPU:ta käyttävällä versiolla se oli jo kahdeksannen kohdalla. Yllätyksenä tuli myös se, että CPU selvästi osasi jollain tavalla hyödyntää GPU:ta, koska molemmilla GPU:lla oli jopa huomattavaa kuormittavuutta.

6.4 Tulosten analysointi

Saadut tulokset poikkesivat aika paljon oletetusta. Ainoastaan yhtä GPU:ta purkamiseen käytettävässä versiossa oli selvää, että yhdellä GPU:lla ei riittänyt rahkeet enää useamman videovuon pyörittämiseen. Tämän jälkeen, kun siirryttiin testaamaan moni-GPU:ta. Ensimmäisenä yllätyksenä tuli se, että tässä versiossa purkamisen yhteydessä CPU:lla ei riittänyt enää tehoa kommunikoida molempien GPU:n kanssa. Moni-GPU:lla pystyttiin kuitenkin kasvattamaan samanaikaisesti aukinaisten videovoiden määrää lähes puolella.

Todella suurena yllätyksenä tuloksia kerätessä ilmeni kuitenkin CPU:ta hyödyntävä versio, jossa koko purkamistyö toteutetaan CPU:lla. Sillä saatiin ylivoimaisesti paras tulos, eli se mahdollisti järjestelmälle määritellyn maksimimäärän avoinna olevia HD-

tason videovoita. Lisäksi CPU:n kuormitus oli pienempää verrattuna moni-GPU:ta hyödyntävään versioon. Hieman yllättävänä tuloksena saatiin lisäksi se, että CPU osasi selvästi hyödyntää ainakin osittain GPU:ta videovuon purkamisessa, sillä ilman yhtäkään avointa videovuota, ei GPU:lla ollut juuri yhtään kuormitusta. Kun kameroita avattiin, jo kolmannen kameran kohdalla ensimmäisellä GPU:lla kuormitus oli yli 20 %. Lisäksi myös toisellakin GPU:lla oli pieni noin 5 % kuormitus. Tähän on varmaankin syynä se, että ohjelmistopurkaja osasi jollain tasolla hyödyntää alla olevaa laitteistoa hyödykseen. Se kuitenkin hyödynsi sitä selvästi vain niin vähän, ettei tiedon siirtämiseen kulu niin paljoa CPU:n tehoa, kuin koko prosessoitavan datan siirtämiseen kuluu.

7 YHTEENVETO

Tämän työn tarkoituksena oli löytää ratkaisu siihen, miten useasta Full HD-tasoisen videovuon purkamisesta syntyvää kuormaa voitaisiin jakaa usean GPU:n kesken. Lähtötilanne työn aloittamiselle oli se, että sillä hetkellä kaikki videovuot purettiin vain yhden GPU:n turvin, jolloin toinen tai useampi GPU jäi vaille työtä. Lisäksi VLC:n ohjelmoijiin oli oltu yhteydessä ennen työn aloittamista. He olivat kertoneet, että heillä ei ole intressejä toteuttaa kyseistä ominaisuutta ohjelmaansa. He myös totesivat, että kyseisen ominaisuuden toteuttaminen olisi hyvin haasteellista.

Työn toteuttaminen oli tullut ajankohtaiseksi siksi, että nykyään Suomen tie- ja kelikameroiden laatu on parantunut siitä, mitä ne ovat olleet järjestelmän kehityksen alkutaipaleella. Nykyään yhden korkealaatuisen videovuon purkamiseen kuluu jo lähes 20 % GPU:n kapasiteetista ja se on johtanut siihen, ettei niitä ole voinut pitää avoinna samanaikaisesti neljää enempää.

Ensiksi työssä käytiin läpi teoriaa ja sen toteuttamisen kannalta tärkeitä työkaluja ja kirjastoja. Teoriaosuudessa saatiin laskemalla selvä ero siihen, että GPU:lla voidaan suorittaa huomattavasti nopeammin yksinkertaista laskentaa, kuin CPU:lla. Videovuon purkaminen on pääosin yksinkertaista laskentaa, minkä vuoksi laitteistokiihdytyksen käyttäminen sen purkamiseen on perusteltua.

Tärkeitä työkaluja, joihin työssä tutustuttiin olivat VLC ja FFmpeg, joiden avulla pystytään pakattu videovuo esittämään käyttäjälle. Lisäksi näiden kahden työkalun tutustumisen yhteydessä käytiin läpi, mitä ovat koodekit ja miten ne vaikuttavat videon pakkaamiseen ja purkamiseen. Huomioitava fakta myös aiheeseen liittyen oli se, että jo nykyisellään kaikki verkon yli siirrettävä data koostuu lähes 70 % videoista. Mikä osaltaan selittää, miksi koodekit ovat tärkeitä standardisoida, sekä löytää uusia, vieläkin tehokkaampia ratkaisuja videoiden pakkaamiseen ja purkamiseen. Käyttämällä jotain videopakkauksenhallinta-standardia, saadaan videon kokoa pienennettyä, jolloin sen siirtäminen verkon yli nopeutuu.

Teoriaosuuteen tutustumisen jälkeen siirryttiin tutkimaan VLC:n lähdekoodia, sekä pureuduttiin DirectX:n saloihin grafiikkaohjelmoinnin osalta. Työssä esitellään, kuinka IDXGIFactory voidaan luoda ja kuinka sen avulla voidaan enumeroida kaikki GPU:t.

Enumeroinnin avulla pystytään valitsemaan haluttu GPU videovuon purkamiseen, jolloin kuormaa pystytään jakamaan näiden kesken.

Tämän jälkeen siirryttiin toteutukseen ja käytiin läpi, mihin väliin VLC:n lähdekoodia tulee iskeytyä, jotta löytää mahdollisen puukotuspaikan oman ominaisuuden toteuttamiselle. Loppupeleissä työssä tarvittavan ominaisuuden toteuttamiseen ei tarvinnut lisätä paljoakaan koodia, mutta kyseisen ohjelman debuggaaminen oli huomattavan haastavaa. VLC:n debuggaaminen tapahtui käyttöliittymän avulla lähettämällä sille parametrinä tiedon, että se haluaa myös lokitustietoja itse määriteltyyn tiedostopolkuun. Näiden lokitustietojen avulla pystyttiin paikantamaan kohta, jossa GPU:n valinta tapahtuu, jolloin oman ominaisuuden lisääminen mahdollistui.

Lopuksi, kun kaikki tarvittava kuormituksen tasaamiseen oli toteutettu, käytiin tätä uutta ominaisuutta testaamassa paikan päällä Tampereeen liikennekeskuksessa, jossa on olemassa testityöasema, joka vastaa raudaltaan tuotantokäytössä olevia koneita. Ainoana erona näillä oli se, että testityöaseman rauta oli hieman vanhempaa, saman mallista kuitenkin.

Tuloksia kerätessä oli selvää, että työssä toteutettu moni GPU:ta hyödyntävä VLC ja järjestelmäversio paransi selvästi järjestelmän käytettävyyttä. Toteutettu versio mahdollisti lähes kaksinkertaisesti HD-tasoisten videovoiden katselemisen samanaikaisesti.

Yllättävä ero yhden ja kahden GPU:n versioissa oli se, että yhden GPU:n tapauksessa ei CPU:n kanssa juuri ollut mitään ongelmaa, mikä osaltaan voi johtua siitä, että sen ei tarvinnut lähettää GPU:lle prosessoitavaksi kuin neljän HD-kameran edestä tietoja. Tässä tapauksessa GPU:lla ei enää riittänyt suorituskyky purkaa näitä kameroita. Mutta kahden GPU:n tapauksessa CPU:n kuormitus alkoi kasvaa huomattavasti. Todennäköisenä syynä tähän on se, että nyt CPU joutui kommunikoimaan kahden eri GPU:n kanssa, joka vaatii siltä enemmän, kuin vain yhden kanssa kommunikointi.

Tulosten keräämisen loppuvaiheessa haluttiin vain kerätä dataa tätä diplomityötä varten siten, että kaikki purkaminen tapahtuu kokonaan CPU:lla. Tässä kohtaa tapahtui jotain hämmästyttävää, nimittäin CPU:lla pystyi ajamaan vielä neljä kameran enemmän, kuin moni-GPU -versiolla pystyi. Lisäksi tässä vaiheessakaan CPU:n kuormittavuus ei ollut yhtä suurta, kuin se oli moni-GPU:n kanssa kahdeksannen kameran kohdalla.

Erona näillä on se, että CPU:lla käytetään eri purkajaa kuin laitteistolla tehtävässä. Tällä on selvästi vaikutusta siihen, että ohjelmistopurkajat on voitu optimoida paremmaksi. Lisäksi CPU osasi selvästi käyttää ainakin osittain hyväkseen GPU:ta, sillä toisen

GPU:n kuormitus oli 20 % luokkaa tai sitä enemmän. Normaalissa tilanteessa se oli taas vain muutaman prosentin.

Työssä löydettiin siis hieman erilainen ratkaisu, kuin oli tarkoitus toteuttaa. Mutta tämän löydöksen avulla päästiin eroon suorituskykyongelmista tämän järjestelmäkomponentin osalta.

On mahdollista, että joskus tulevaisuudessa saatetaan harkita siirtymistä takaisin laitteistokiihdytystä hyödyntävään versioon, kun vastaaotetun videovuon kuvan laatu paranee entisestään. Nykyisellään järjestelmään kiinnitetyt kamerat lähettävät kuitenkin maksimissaan vain Full HD-kuvaa purettavaksi ja sen siirtämiseen näyttää vielä kuluvan enemmän aikaa ja resursseja kuin sen purkamiseen. Nykyteknologia kuitenkin mahdollistaa jo 4K-videokuvan lähettämisen, jonka saapuminen myös järjestelmään on mahdollista tulevaisuudessa. Mutta ainakin vielä järjestelmän suorituskyvyn kannalta on järkevämpää käyttää videovuon purkamiseen CPU:ta laitteiston sijaan.

LÄHTEET

[1] http://www.directxtutorial.com/Lesson.aspx?lessonid=11-4-1. Understanding Graphics Concepts, DirectXTutorial. Viitattu 7.9.2018

[2] TIE-51257 2018-1 Parallel Embedded Computing. Exercise 1. Viitattu 10.9.2018

[3] http://user.ceng.metu.edu.tr/~e1190545/what.htm. What are Cryptography and Cryptographic accelerators. Viitattu 24.10.2018

[4] Pang, Aiken, and Peter Membrey. Beginning FPGA: Programming Metal: Your Brain on Hardware. Apress, 2017.

[5] https://www.academia.edu/36504841/A_Survey_of_ReRAM-based_Architectures_for_Processing-in-memory_and_Neural_Networks. S. Mittal, Machine Learning and Knowledge Extraction, 2018. Viitattu 24.10.2018.

[6] https://patents.google.com/patent/US1641608?oq=1641608. Google patents. Viitattu 7.9.2018

[7] https://tubularinsights.com/video-2021/. G. Jarboe, 14.12.2017. Viitattu 12.10.2018.

[8] Richardson, Iain E. The H.264 Advanced Video Compression Standard, Second Edition. John Wiley & Sons.

[9] https://www.vcodex.com/an-overview-of-h264-advanced-video-coding/. Vcodex Ltd. Viitattu 24.8.2018

[10] http://wiki.webmproject.org/hardware/socs. SoCs Supporting VP8/VP9, WebM Wiki. Viitattu 7.9.2018

[11] Waggoner, Ben. Compression for Great Video and Audio. Focal Press, 2010.

[12] https://www.ffmpeg.org. Ffmpeg. Viitattu 24.8.2018

[13] https://www.videolan.org/vlc/libvlc.html. 2018 VideoLAN, VLC. Viitattu 17.8.2018

[14] https://www.dacast.com/blog/software-vs-hardware-encoders-for-live-video-streams/. Software vs. Hardware Encoders for Live Video Streams, DaCast. Viitattu 7.9.2018

[15] Sherrod, Allen, and Wendy Jones. Beginning DirectX 11 Game Programming. Cengage Learning, 2012.

[16] https://docs.microsoft.com/en-us/windows/desktop/direct3ddxgi/d3d10-graphics-programming-guide-dxgi. DXGI Overview, Microsoft. Viitattu 7.9.2018

[17] https://www.nvidia.com/content/nvision2008/tech_presentations.html. Nvision 08, Nvidia, 2008. Viitattu 18.10.2018

[18] https://docs.microsoft.com/en-us/windows/desktop/medfound/about-dxva-2-0. Microsoft. Viitattu 24.8.2018

[19] Simmonds, Chris. Mastering Embedded Linux Programming - Second Edition. Packt Publishing Ltd, 2017.

[20] Cataldo, Giuseppe Di. Stack Frames: A Look from Inside. Apress, 2016.

[21] Jones, M. Tim. GNU/Linux Application Programming, Second Edition. Cengage Course Technology, 2008.

[22] https://github.com/Alexpux/mingw-w64/blob/master/mingw-w64-doc/howto-build/mingw-w64-howto-build.txt. Alexpux, mingw-w64. Viitattu 17.8.2018

[23] Vohra, Deepak. Pro Docker. Apress, 2016.

[24] https://www.docker.com/resources/what-container. 2018 Docker Inc. Viitattu 17.8.2018

[25] https://docs.microsoft.com/en-us/windows/desktop/api/dxgi/. DXGI, Microsoft. Viitattu 10.9.2018