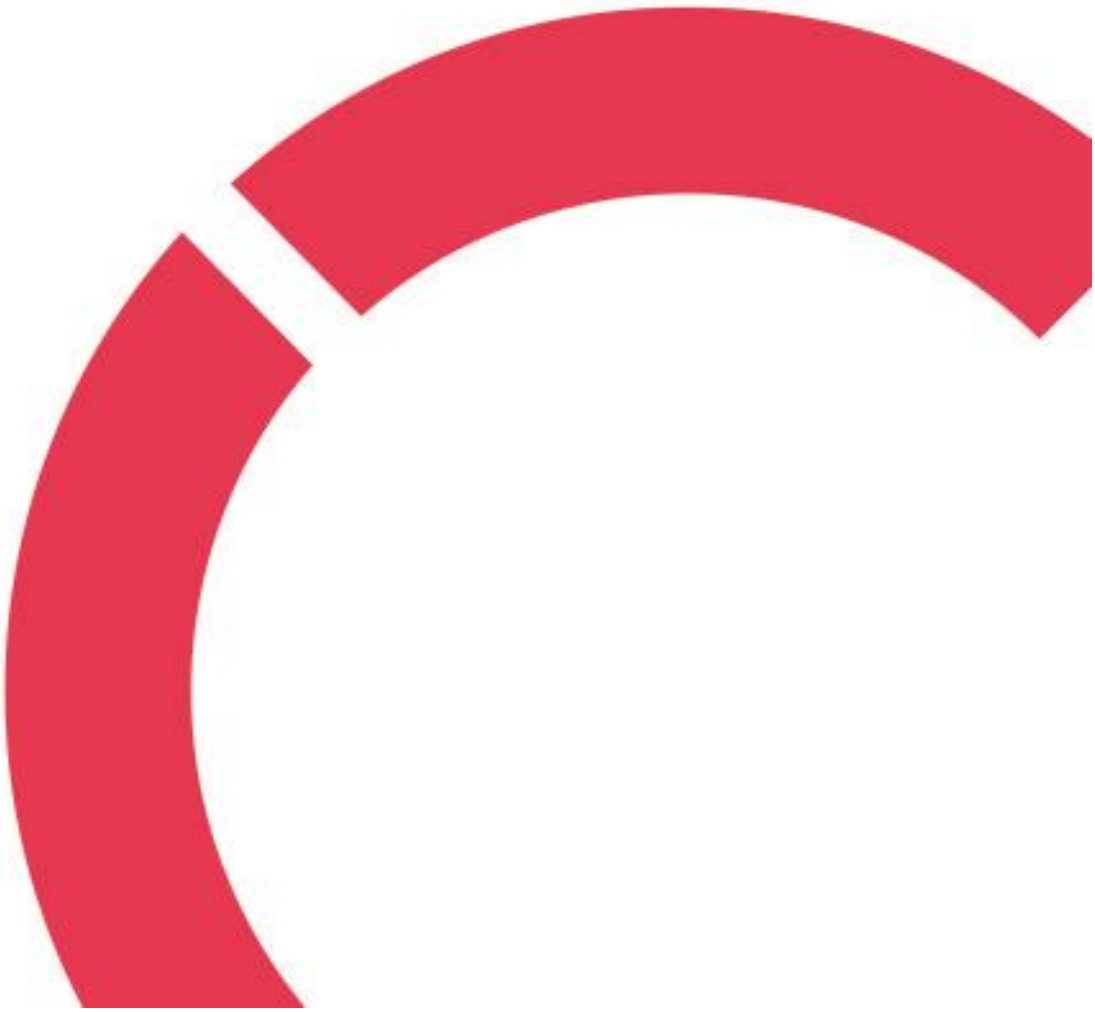
Jesse Sahlström

OPTIMOINTITASOT BINÄRISOINTIOHJELMASSA

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutus
Maaliskuu 2023



TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Maaliskuu 2023	**Tekijä/tekijät** Jesse Sahlström
Koulutus Tieto- ja viestintätekniikka		☒ AMK ☐ YAMK
Työn nimi Optimointitasot binärisointi ohjelmassa		
Työn ohjaaja Jari Isohanni		**Sivumäärä** 12
Työelämäohjaaja -		

Tässä opinnäytetyössä tutkittiin ohjelmointikielen kääntäjän optimointitasojen vaikutusta yksinkertaisen kuvanmuokkausohjelman suorituskykyyn. Ohjelma keskittyi yhteen tiedostomuotoon ja kuvan binärisointiin.

Opinnäytetyön taustalla oli mahdollisuus perehtyä ohjelmointikääntäjien käyttämiin optimointitekniikoihin, C++-kieleen ja kuvanmuokkaukseen.

Ohjelmakielen kääntäjä pystyi lisäämään ohjelman suorituskykyä huomattavasti. Kääntäjän käyttämien yksittäisten tekniikoiden huomaaminen käännetystä ohjelman koodista oli hyvin vaikeaa. Tämä johtui kääntäjän käyttämien tekniikoiden monipuolisuudesta ja monimutkaisuudesta.

Asiasanat
Binärisointi, C++, Optimointi

ABSTRACT

Centria University of Applied Sciences	**Date** March 2023	**Author** Jesse Sahlström
Degree programme Information and communication technology programme		
Name of thesis Optimatization levels in a binarization program		
Centria supervisor Jari Isohanni		**Pages** 12
Instructor representing commissioning institution or company -		

In this thesis, the influence of the optimization levels of the programming language compiler on the performance of a simple image editing program was investigated. The program focused on one file format and image editing technology.

The background of the thesis was the opportunity to learn about the optimization techniques used by programming compilers, deepening the understating of C++ language and getting to know image editing, particularly image binarization.

The program language compiler was able to increase the performance of the program considerably. It was very difficult to notice the individual techniques used by the compiler in the compiled program code. This was due to the versatility and complexity of the techniques used by the compiler.

Key words
Binarization, C++, optimization

Sisällys

1 JOHDANTO

Opinnäytteen tarkoituksena on tutustua kuvan käsittelyyn, BMP-tiedostomuotoon ja erityisesti kuvankäsittelyn binärisointimenetelmiin sekä C++ ja Assembly ohjelmointikieliin, ja tutustua kääntäjän eri optimointitasoihin. Tässä opinnäytetyössä tehdään C++-kielellä ohjelma, joka avaa BMP-kuvatiedoston, muuntaa sen harmaasävyiseksi ja lopuksi binärisoi sen, eli muuntaa kuvan mustavalkoiseksi ja lopuksi tallentaa sen. Ohjelma myös mittaa kuinka kauan näissä eri vaiheissa menee aikaa, jotta lopuksi voidaan arvioida eri optimointitasojen vaikutusta ohjelman toimintaan.

Binärisointia tarvitaan monessa eri kuvankäsittelymenetelmässä, mutta nykyään erityisesti se on käytössä konenäön ja tekoälyn käytössä. Binärisoimalla pystyy poistamaan mahdollisimman paljon ylimääräistä tietoa kuvasta, jolloin tekoälyn tai konenäön on helpompi erottaa esimerkiksi sarjanumeroita tai löytää pintavikoja tuotteista.

Binärisointia varten pitää löytää sopiva kynnystämisraja, eli arvo, jonka mukaan ohjelma muuttaa yksitätisen pikselin arvon joko täysin valkoiseksi tai täysin mustaksi. Tämän työn tarkoituksena on erityisesti perehtyä Otsu-algoritmiin, joka on kynnystämisalgoritmi, eli se pyrkii löytämään mahdollisimman hyvän kynnystämisarvon kuvan Binärisointia varten.

2 KUVANKÄSITTELY TIETOTEKNIIKASSA

Kuvankäsittely tietotekniikassa on hyvin laaja käsite, joka pitää sisällään monia eri tekniikoita ja käyttötarkoituksia: digitaalisten kuvien muokkaus ja manipulointi, tietokonegrafiikat tai konenäkö kuten pullonpalautusautomaatissa oleva viivakoodinlukija tai teollisuuslaitoksissa pintavikojen löytäminen tai sarjanumeroiden lukeminen (Steger &, Ulrich & Wiedemann 2018). Kuvankäsittelyyn myös kuuluvat eri tekoälyä käyttävät kuvan tutkimis- ja muokkausmenetelmät sekä erilaiset tiedostomuodot, kuten BMP, JPEG, GIF.

Kuvankäsittelyyn myös kuuluu kuvan pakkaaminen niin, että kuvatiedosto vie vähemmän tilaa. Kuvan pakkaaminen onnistuu häviöttömällä menetelmällä, jolloin tietoa ei menetetä. Tällöin kuvan tiedot järjestetään uudelleen niin, että se vie vähemmän tilaa. Kuva voidaan pakata myös häviöllisesti. Häviöllisessä pakkauksessa kuvaa pyritään käsittelemään niin, että ihmissilmä ei sitä huomaa, mutta kuvasta on poistettu tietoa, jolloin säästetään tilaa. (DigiFAQ.)

Yksi menetelmä, jota käytetään useassa kuvankäsittelytekniikassa, on kuvan binäriasaatio, eli kuvan muuttaminen binääriseen muotoon, jolloin se sisältää vain kahta arvoa. Binärisaatiossa tarkoituksena on poistaa kuvasta ylimääräistä tietoa, jota esimerkkisi konenäkö tai tekoäly eivät tarvitse kuvan tulkitsemisessa. Tällaista ylimääräistä tietoa voi kutsua tässä tapauksessa kohinaksi. (CharterGlobal.)

Tämä tutkielma keskittyy pakkaamattoman BMP-kuvatiedoston avaamiseen, kuvan muuntamiseen harmaaksi, binärisaation toteuttamiseen ja kääntäjän optimointitekniikoihin.

2.1 Binärisaation käyttökohteet

Värillinen kuva sisältää paljon tietoa, jota konenäkö tai tekoäly ei tarvitse, joten tarvitaan menetelmiä niiden poistamiseen niin, että niille oleellinen tieto jää talteen mutta ylimääräinen kohina poistetaan. Binärisaatio auttaa tekoälyä tai konenäköä erottamaan esineen tai tekstin reunat paremmin (KUVIO 1).

KUVIO 1. Reunojen erottaminen binärisoinillä.

Tekoälyn käytössä binärisaation käyttö on hieman vähentynyt, koska on helpompi kouluttaa neuroverkko tunnistamaan kuvia. Neuroverkon kouluttaminen vaatii paljon saatavilla olevaa tietoa, joten binärisointia edelleen käytetään mm. lääketieteellisissä tarkoituksissa, joissa tietoa on vain rajallisesti tarjolla neuroverkon kouluttamista varten. (CharterGlobal.)

2.2 BMP-kuvatiedosto

Tiedostomuotoa BMP käytetään digitaalisten kuvien tallentamiseen. BMP-tiedostomuoto on riippumaton grafiikkalaitteista ja siksi siitä käytetään myös nimeä DIB, joka on lyhenne sanoista device independent bitmap. Laiteriippumattomuus tarkoittaa sitä, että BMP-tiedoston voi avata monilla eri käyttöjärjestelmillä, kuten Microsoft Windows tai Mac OS. (Fileformat.)

Tehdessä ohjelmaa, joka lukee BMP-tiedostoja tärkeimpiä asioita, pitää lukea BMP-tiedoston otsikko-osiosta, headeristä tiedostotyyppi. Tiedostotyyppi löytyy heti ensimmäisestä kahdesta tavusta. BMP-tiedoston tunnistaa otsikossa olevasta tunnuksesta 0x4D42 (KUVIO 2), joka on heksadesimaaliluku ja tarkoittaa ASCII-merkeissä kirjaimia BM. Jos tunnus löytyy, on kyseessä BMP-tiedosto. (Fileformat.)

```
void read(const char* fname) {
    std::ifstream inp{ fname, std::ios_base::binary };
    if (inp) {
        inp.read((char*)&file_header, sizeof(file_header));
        if (file_header.file_type != 0x4D42) {
            throw std::runtime_error("Error!! Unrecognized file format.");
        }
```

KUVIO 2. BMP-tiedoston tiedostotyypin lukeminen.

BMP-tiedosto-otsikko on kooltaan 14 tavua. Tämän jälkeen on DIB-otsikko-osio. Molemmat näistä sisältävät tietoa BMP-tiedostosta. Koska tutkielma ei keskity niinkään kuvan avaamiseen kuin binärisoinnin optimoinnin tutkimiseen, yksinkertaisuuden vuoksi ei ohjelma avaa BMP-tiedostoja, jotka käyttävät pakkaustekniikoita. BMP-tiedoston itse kuvatiedon taulukko on aina neljän kerrannainen, joten kuvarivin loppuun saatetaan lisätä ylimääräinen kenttä, jotta kerrannaisuus säilyy. (Fileformat.)

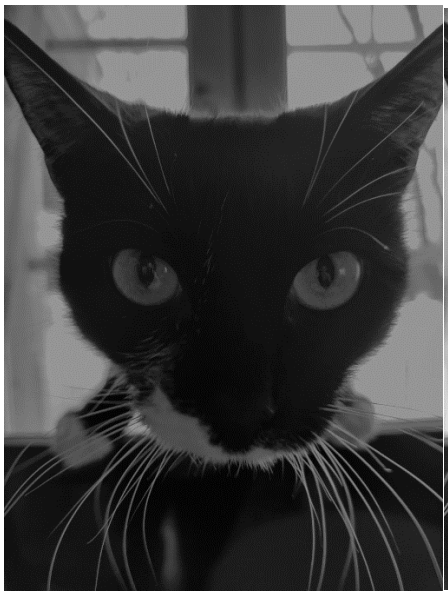
3 KUVAN BINÄRISAATIO

Ohjelman täytyy lukea ensin BMP-tiedostosta pikselitiedot tietotaulukkoon ennen kuin niitä voi alkaa käsittelemään. Tämän jälkeen kuva täytyy muuttaa harmaaksi ja lukea harmaaksi muutetut tiedot, jotta niistä voidaan laskea kynnystämis-raja binärisointiä varten.

Ennen kuvan binärisointiä ja muita binärisointi optimointioperaatioita tulee kuvan olla harmaa. Tämä operaatio tehdään siksi, että värillinen kuva sisältää paljon tietoa, jota kuvan binärisointioperaatiossa ei tarvita. Tekoälyn kannalta myös tärkeimmät piirteet kuvassa ovat rajat ja tekstuurit, jotka eivät niinkään aina hyödy väristä. Asiaa on hyvä ajatella esimerkiksi niin että jos tehtävänä on tehdä ohjelma, joka lukee tekstiä paperilta, ei paperin värillä ole sinänsä mitään merkitystä lopputuloksen kannalta. Ohjelma jopa toimii nopeampaa, koska siinä on vähemmän tietoa kuin ohjelma joutuu käymään läpi. (Charter-global.)

BMP-kuvatiedoston muuttaminen harmaaksi tapahtuu niin, että BMP-kuvatiedoston bit_count, eli bittimäärä jaetaan kahdeksalla, jotta saadaan 4 kanavaa. Oleellisia näistä ovat kolme ensimmäistä, joihin ladataan värien arvot, punainen, vihreä ja sininen. Nämä kolme käydään läpi kuvan korkeuden ja leveyden avulla for loopeilla ja jokainen arvo muutetaan harmaaksi kertomalla punainen 0,3lla, vihreä 0,59lla ja sininen 0,11lla. (TutorialsPoint.)

Äkkiseltään voisi kuvitella, että koska yksi pikseli koostuu kolmesta väristä, jakamalla pikselin luku kolmella, sen voisi muuttaa mustavalkoiseksi. Tämä toimii, mutta lopputulos on hyvin pimeä kuva. (KUVIO 3.) Tämä johtuu siitä, että jokaisella kolmella värillä on eri aallonpituus. Siksi käytetään niin sanottua painotettua menetelmää tai luminositeetti-menetelmää. (KUVIO 4.) Tällä menetelmällä kuvasta tulee oikean näköinen harmaa kuva. (TutorialsPoint.)




KUVIO 3. Keskiarvo menetelmä KUVIO 4. Painotettu menetelmä.

Thresholding eli kynnystäminen on kuvan binärisaatio prosessissa olennainen vaihe ja yksi optimoinnin tärkein kohde. Kynnystämisen tarkoituksen on etsiä niin sanottu raja, jonka mukaan päätetään, onko kuvan pikseli binärisoinnin jälkeen musta vai valkoinen. Pikselit, jotka ovat kynnyksin yli ovat valkoisia ja kynnyksen ali, mustia.

Kynnystämisen optimointia varten tässä työssä käytettiin Otsu-algoritmia, joka on nimetty menetelmän kehittäjän mukaan Nobuyaki Otsu (Otsu 1979). Esimerkiksi kun kuva koostuu vain harmaista väreistä, joiden intensiteetit vaihtelevat 0–255 (eli 255 täysin valkoinen ja 0 täysin musta) valitaan jokin arvo kynnysarvoksi, vaikka 127. Binärisaatio muuttaa arvot, jotka ovat alle kynnysarvon 127, nollaksi eli mustaksi ja arvot yli 127 valkoisiksi.

Tämän kynnysarvon valintaan on olemassa monia eri menetelmiä. Jos kuvassa esiintyvistä arvoista tehdään histogrammi, voidaan hyvä kynnysarvo valita kokeilullisesti valitsemalla silmämääräisesti kynnysarvo, jos histogrammissa näkyy selkeä raja harmaidenvärien voimakkuuden välillä. Tämän jälkeen voi kokeilla binärisoidä kuvan tällä arvolla ja katsoa miltä kuva näyttää. Tällainen menetelmä on kuitenkin hyvin hidasta ja työlästä varsinkin, jos tätä pitää tehdä toistuvasti ja se pitäisi saada tehtyä nopeasti.
Tätä varten on kehitetty useampi algoritmi, joka pyrkii laskennallisesti löytämään hyvän kynnysarvon kuvan binärisoinnilleja Otsu-algoritmi on näistä globaaleista kynnystysmenetelmistä suosituin. (Athimethphat 2011.)

4 BINÄRISOINTIALGORITMIN VALINTA

Sopivan binärisointi-algoritmin valinnassa tulee ottaa huomioon seuraavia asioita: Vaatiiko kuva ”global” vai ”local” -kynnyksen eli koko kuvan kattavan yhden kynnyksen vai paikallisen tai niin sanotun adaptiivisen, joka valitsee kuvan eri alueille sopivan binärisointi-kynnyksen. Esimerkiksi jos kyseessä on vanha dokumentti, josta on otettu huonolla valaistuksella kuva, jossa kuvan varjostus vaihtelee paljon, voi olla tarpeen käyttää paikallista adaptiivista algoritmia. Lääketieteellisissä kuvissa taas varjostus on yleensä hyvin tasainen ja kuvasta on helppo erottaa etu- ja taka-ala, joten niissä taas käy parhaiten algoritmi, joka etsii koko kuvalle yhden kynnysarvon. (Saha, Basu, & Nasipuri 2011.)
Tässä opinnäytetyössä käytetään yksinkertaisuuden vuoksi vain koko kuvan kattavaa Otsu-algoritmia.

Otsu-algoritmi kokonaisuudessaa on:

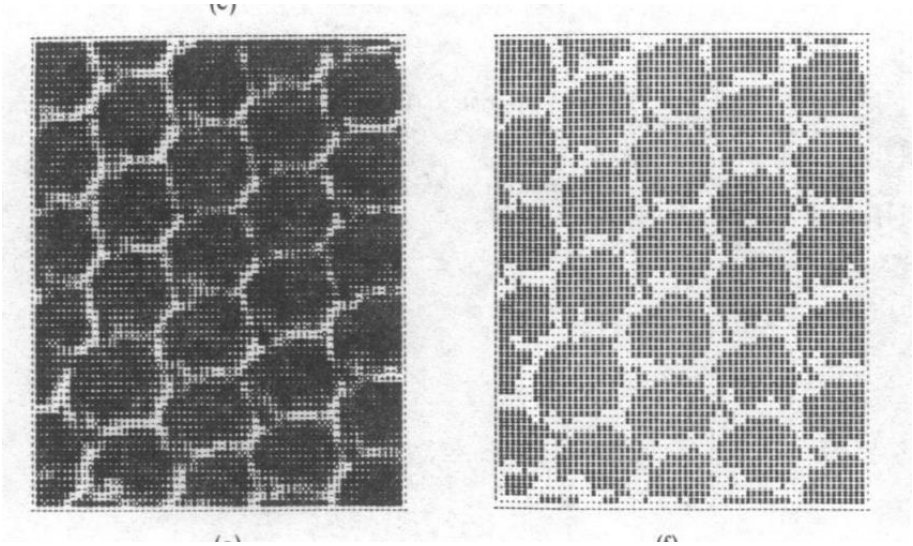
$$\sigma_B^2 = W_b W_F (\mu_b - u_F)^2$$

Sigma b on taustan paino(W), kertaa etualan paino(W), kertaa taustan varianssi miinus etualan varianssin neliö. Pieni b tarkoittaa background(tausta) ja pieni f foreground(etuala). (Otsu 1979.)

Käytännössä algoritmissa käydään läpi kaikki kynnysarvot 0–255. Jaetaan kuva jokaisella kynnysarvolla painoihin W_b ja W_f. Painot lasketaan laskemalla yhteen, kuinka monta pikseliä esiintyy kussakin osassa ja jakamalla se kuvan kokonaispikselimäärällä. Näin saadaan W_b ja W_f. Seuraavaksi tarvitaan u_b ja u_f eli taustan ja etualan väliset varianssit. Arvot u_b ja u_f saadaan laskemalla summa, kuinka usein arvo esiintyy ja jakamalla molemmat taustan tai etualan pikseleiden määrällä. (Otsu 1979.)

Saadut arvot syötetään yhtälöön, saadaan niin sanottu luokkien välinen varianssi. Se millä kynnysarvolla saadaan suurin luokkien välinen varianssi, on optimaalinen kynnysarvo kyseiselle kuvalle. (Otsu 1979.)

Otsu-algoritmi toimii parhaiten, jos kuvalla on bi-modaalinen jakauma eli taustan ja etualan välillä on selvä ero. (KUVIO 5.) Kuvan histogrammia katsoessa siinä tulisi näkyä kaksi selvää huippua. Otsu-algoritmi toimii heikoiten silloin kun kuvan valaistus on epätasapainoinen. (Otsu 1979.)



KUVIO 5. Selvä ero taustan ja etualan välillä. (Otsu 1979.)

5 OHJELMAKÄÄNTÄJÄN OPTIMOINTITASOT

Gcc on kääntäjäkokoelma, joka on osa GNU-projektia. Gcc:hen kuuluvat mm. C, C++, Fortran, Go jne. Ohjelmointikääntäjän tehtävänä on kääntää yksi ohjelmointikieli toiseen, yleensä korkeamman tason kielestä matalimmille. (GCC.) Tässä tutkielmassa käytettiin Gcc:n g++ -kääntäjää eli C++-kielen kääntäjää.

Kääntäjästä löytyy eri optimointitasoja, jota pystyy käyttämään käyttämällä komentoa -O*taso*, jossa tason kohdalle laitetaan numero yhden ja kolmen välillä. Tässä työssä vertailtiin kahta eri optimointitasoa niin suorituskyvyn tasolta kuin myös kielen tasolta. Vertailtavat tasot olivat vakiotaso O0 ja O3-taso.

Tasot yhdestä kolmeen lisäävät optimointitekniikoita. Taso 0 on vakiotaso, jossa ei ole ollenkaan käytössä optimointitekniikoita. Taso 1 laittaa päälle yleisiä optimointitekniikoita. Kääntäjä pyrkii luomaan nopeampaa ja vähemmän tilaa vaativaa koodia, ilman että kääntämiseen kuluva aika kasvaa. Vaativampia optimointitekniikoita, kuten käskyjen aikatauluttaminen, ei ole käytössä tällä tasolla. (Linuxtopia.)

Taso 2 käyttää kaikkia aikaisempien tasojen optimointeja, mutta se myös pyrkii aikatauluttamaan käskyjä, mutta ohjelman kokoon nämä optimoinnit eivät vielä vaikuta, eli käytössä ei ole niin sanottuja tila-aika-kompromisseja. Taso 2 on paras taso ohjelmille, jotka ovat julkaisuvalmiita, koska se tarjoaa parhaat optimointiasetukset eikä muuta ohjelman kokoa. (Linuxtopia.)

Taso 3 optimointi voi muuttaa ohjelman kokoa ja joissain tapauksissa se saattaa myös hidastaa ohjelmaa koska optimointiasetukset ovat niin laajat. Tämä taso lisää kääntämiseen tarvittavaa aikaa ja muistin käyttöä. Taso 3 ottaa käyttöö mm. ftree-vectorize -komennon, joka laittaa päälle vektorisoinnin. Vektorisoinnilla muutetaan algoritmi vaikuttamaan kerralla muuttujajoukkoon sen sijaan että vaikutettaisiin vain yhteen muuttujaan kerrallaan. (Intel.)

Tasoon kolme kuuluvat myös kaikki aikaisemmat tasot, mutta siitä silti puuttuu silmukoiden purkaminen. Jos kaikki ohjelman silmukat halutaan purkaa, tulee käyttää komentoa -funroll-loops. Silmukoiden purkaminen kasvattaa tiedoston kokoa, mutta siinä ei ole takeita, että se nopeuttaa ohjelmaa. Silmukoiden purkamista tulee siis miettiä joka ohjelman kohdalla erikseen. (Gentoo linux.)

Yleinen väärinkäsitys optimoinnissa on että, mitä enemmän käyttää kääntäjän optimointitekniikoita, sitä nopeampia ja pienikokoisempia ohjelmia saadaan tehtyä, mutta huolimattomalla optimointitekniikoiden käytöllä voi luoda ohjelmia, jotka ovat hitaampia, suurempia ja sisältävät koodivirheitä, joita on todella vaikea korjata. Kääntäjän optimointitekniikoita tulee siis käyttää harkiten. (Gentoo linux.)

6 MITTAUSTULOKSET

Ajan mittaamisessa tulee ottaa huomioon, onko ajastusfunktio Wall time vai CPU time. Wall time, eli seinäaika mittaa aikaa nimensä mukaisesti samalla tavalla kuin seinäkello, eli ajastus, joka käyttää wall time ei ota huomioon mihin muuhun prosessori käyttää aikaa samaan aikaan kun ohjelmaa ajetaan. Ohjelman aikana prosessori voi tehdä muiden ohjelmien prosesseja jne. CPU time, eli prosessoriaika taas mittaa ainoastaan ajan, joka kuluu kyseisen prosessin suorittamiseen. Tämän ohjelman ajastamiseen riittää seinäaika, joten siihen käytettiin C++ Chrono-kirjastoa. Chronon seinäaika riittää siksi, että ohjelma ajettiin useita kertoja ja näistä ajoista otettiin keskiarvo ja Chrono oli myös siksi hyvä valinta koska se pystyy mittaamaan sekunnin murto-osia. (Ferreira 2020.)

Tulokset on saatu testikoneella, jossa on Intel Core i7-9700K CPU @ 3.60GHz prosessori ja GNU-kääntäjä GCC versio 9.2.0 Windows 10 Home käyttöjärjestelmän versiolla 22H2. Ohjelma ajettiin kymmenen kertaa kummallakin optimointitasolla ja ajat kirjattiin ylös, ja niistä laskettiin keskiarvot ohjelman eri toiminnoille.

Kuva on kooltaan 1309 x 876 pikseliä. Perus eli O0-optimointitasolla kuvan muuttaminen värikuvasta harmaaksi vei testikoneella aikaa keskimäärin noin 0,09 sekuntia. Vaativin tehtävä ja siksi eniten aikaa vievä on Otsu-algortmin kynnysrajaan löytämiseen käytettävä aika. Keskiarvo on noin 25,39 sekuntia. Lopuksi kuvan binärisointi vei aikaa vain 0,03 sekuntia. O3-tasolla värikuvasta kuvan muuttaminen harmaaksi vei vain 0,01sekuntia, Otsu-algoritmi pystyi löytämään parhaan kynnysarvon binärisoinnille keskimäärin 7,5 sekunnissa ja binärisoimaan kuvan noin 0,004 sekunnissa.

7 JOHTOPÄÄTÖKSET

Nykypäivänä kääntäjät pystyvät tekemään yllättäviä asioita optimoinnin suhteen. Kääntäjät ja kääntäjän optimointimenetelmät ovat todella monimutkainen ja syventymistä vaativa aihe. Vaikka osasin odottaa tätä, silti jokaista uutta oppimaani asiaa kohtaan löytyi kymmenen uutta hankalaa, mutta mielenkiintoista asiaa. Yllättävän suuri osa tutkielman tekemisestä meni kääntäjien toimintaan perehtymiseen.

Kääntäjän optimoimaa koodia oli parasta lukea komennolla:
g++ -S -O3 -masm=intel -fno-asynchronous-unwind-tables -fno-dwarf2-cfi-asm main.cpp
Komento -S tallentaa tiedoston asm-tiedostomuotoon, eli assembly-kielellä. Komento masm=intel muuttaa asm-tiedoston AT&T-syntaksista Intel-syntaksiin, eli muotoon ”komento, kohde, lähde”. Komennot - fno-asynchronous-unwind-tables -fno-dwarf2-cfi-asm poistaa ns. ”ylimääräistä” tietoa, jota kääntäjä tarvitsee ja siten tekee koodista helpommin luettavaa ihmiselle (GCC). Assembly-koodia lukemalla oli todella vaikea saada selville mitä menetelmiä kääntäjä on käyttänyt optimoimaan koodia. Yksi suurimpia muutoksia oli koodin järjestys. Vakiotaso pyrkii pitämään koodin samassa muodossa, jossa se on alkuperäisessä tiedostossa, kun taas O3-taso järjestelee koodia uudelleen (Linuxtopia). Kääntäjä myös pyrkii poistamaan monia asioita koodista, jotka ovat siellä koodin luettavuuden tai vikojen löytämisen vuoksi. (Cooper K & Torczon L.)

Optimointitasojen välillä oli yllättävän suuri ero. Lähes puolesta minuutista alle kymmeneen sekuntiin on melkoinen hyppy. Ohjelman pienen koon takia ei pystynyt mittaamaan ohjelman kääntöaikoja eri optimointitasoilla. Yllättävintä oli, että kolmannella optimointitasolla myös ohjelman koko pieneni huomattavasti. Perus optimointitasolla ohjelman koko oli 130 kilotavua kun taas suurimmalla optimoinnilla ohjelman koko oli vain 75 kilotavua ja koodirivien määrä tippui noin 8800 rivistä vain 2500 riviin. Modernit kääntäjät eivät vain keskity ohjelmien nopeuteen vaan myös ohjelman koko tai energian käyttö ovat optimoinnin kohteena. (Cooper K & Torczon L.)

Aihe oli mielestäni sopiva tutkinnan kohde. Siinä pääsi tutustumaan uusiin alueisiin eikä se ollut mahdoton toteuttaa vaikka alkuperäisestä suunnitelmasta jouduin hieman poikkeamaan. Alun perin oli tarkoitus pyrkiä itse kirjoittamaan joitain ohjelman funktioita assembly-kielellä, mutta ne osoittautuivat hieman liian vaativiksi. C++-kieltä käyttäessä pyrin samalla tutustumaan ja käyttämään uudempien standardien funktioita ja ohjelmointitapoja. Apuna tässä käytin Stanley Lippmanin kirjoittamaa C++ Primer-kirjaa, joka keskittyy C++11 standardiin. Jos lähtisin tekemään tätä uudestaan, valitsisin aiheen niin että

siinä olisi useampia pieniä yksinkertaisia matemaattisia funktioita, joita olisi helppo tutkia assembly-kielellä ja olisi ehkä helpompi huomata kääntäjän käyttämiä optimointitekniikoita tai kirjoittaa niitä itse.

LÄHTEET

Athimethphat, M. 2011. *A Review on Global Binarization Algorithms for Degraded Document Image*s. *AU J.T.* 14(3), 188-195. January 2011.
Saatavissa: https://www.thaiscience.info/journals/Article/AUJT/10817455.pdf Viitattu: 15.11.2022

Charterglobal. What is image binarization in AI? Saatavissa: https://www.charterglobal.com/what-is-image-binarization-in-ai/ Viitattu: 23.11.2022

Cooper, K. D. & Torczon, L. 2012. Engineering a Compiler. 2nd ed. Boston: Elsevier/ Morgan Kaufmann.

DigiFAQ. Mitä on kuvan pakkaaminen? Saatavissa: http://digifaq.info/digifaq/3_pakkaaminen.html Viitattu: 23.11.2022

Ferreira, C. R. 2020. 8 Ways to Measure Execution Time in C/C++. Saatavissa: https://levelup.gitconnected.com/8-ways-to-measure-execution-time-in-c-c-48634458d0f9 Viitattu 5.12.2022

Fileformat. What is a BMP file? Saatavissa: https://docs.fileformat.com/image/bmp/ Viitattu: 1.12.2022

GCC. GCC, the GNU Compiler Collection. Saatavissa: https://gcc.gnu.org/ Viitattu 2.12.2022

Gentoo linux. Gcc optimization. Saatavissa: https://wiki.gentoo.org/wiki/GCC_optimization Viitattu 21.3.2023

Intel. Vectorization: A Key Tool To Improve Performance On Modern CPUs. Saatavissa: https://www.intel.com/content/www/us/en/developer/articles/technical/vectorization-a-key-tool-to-improve-performance-on-modern-cpus.html Viitattu 21.3.2023

Linuxtopia. Optimization levels. Saatavissa: https://www.linuxtopia.org/online_books/an_introduction_to_gcc/gccintro_49.html Viitattu 21.3.2023

OpenCV. Image Thresholding. Saatavissa: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html Viitattu 21.3.2023

Otsu, N. 1979. A Threshold Selection Method from Gray-Level Histograms. *IEEE transactions on systems, man, and cybernetics.* vol. smc-9, No. 1, January 1979. Saatavissa: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4310076Viitattu: 23.11.2022

Saha, S. Basu, S. & Nasipuri, M. 2011. Automatic Localization and Recognition of License Plate Characters for Indian Vehicles. *International Journal of Computer Science and Emerging Technologies* 2(4), 520-533. https://www.researchgate.net/publication/261760841_Automatic_Localization_and_Recognition_of_License_Plate_Characters_for_Indian_Vehicles Viitattu 23.11.2022.

Steger, C. Ulrich, M. & Wiedemann, C. 2018. *Machine Vision Algorithms and Applications*. 2nd ed. Weinheim: Wiley-VCH.

TutorialsPoint.com. Grayscale to RGB conversion Saatavissa: https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm Viitattu: 23.11.2022