Suunnitteludokumentti

Makromolekyylikompleksien rekonstruktioprosessin rinnakkaistaminen Beowulf-ympäristössä

Antti Perälä

Ilkka Koskenniemi

Matti Kääriäinen

Peter Lamberg

Taneli Mielikäinen

Valtteri Rahkonen

Helsinki 12. toukokuuta 2000

Ohjelmistotuotantoprojekti: Suunnitteludokumentti

Ohjaaja: Satu Sihvo

Asiakkaat:

Janne Ravantti

professori Dennis Bamford

Vastuuhenkilö: Juha Taina

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Sisältö

Kuvat

Kirjallisuutta

[1] How to embed perl in your c program. `http://www.perl.com/pub/doc/manual/html/pod/perlembed.html`.

[2] Macromolecule structure file formats in ifl. `http://www.mih.unibas.ch/Bioimage/iflMacroMol.html`.

[3] Message passing interface forum. `http://www.mpi-forum.org/`.

[4] MPICH – a portable implementation of MPI. `http://www-unix.mcs.anl.gov/mpi/mpich/`.

[5] Perl syntax. `http://www.perl.com/pub/doc/manual/html/pod/perlsyn.html`.

[6] Practical extraction and report language. `http://www.perl.com/pub/doc/manual/html/pod/perl.html`.

[7] Rationale for american national standard for information systems - programming language - C. `http://www.lysator.liu.se/c/rat/title.html`.

[8] Swig 1.1 users manual. `http://www.swig.org/Doc1.1/HTML/Contents.html`.

[9] Anonymous. MPI: A message passing interface. *Proceedings of the Supercomputing Conference* (1993), 878–883.

[10] Aurenhammer, F. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Computing Surveys 23*, 3 (Sept. 1991), 345–405.

[11] Bacon, J. *Concurrent Systems: Operating Systems, Database and Distributed Systems: An Integrated Approach*, 2 ed. Addison-Wesley, 1998.

[12] Baker, T. S., Olson, N. H., and Fuller, S. D. Adding third dimension to virus life cycles: Three-dimensional reconstruction of icosahedral viruses from

cryo-electron micrographs. *Microbiology and Molecular Biology Reviews 63*, 4 (1999), 862 – 922.

[13] Frank, J. *Three-Dimensional Electron Microscopy of Macromolecular Assemblies*. Academic Press, 1996.

[14] Fredriksson, K., and Ukkonen, E. Combinatorial methods for approximate image matching under translations and rotations. *Pattern Recognition Letters 20* (1999), 1249 – 1258.

[15] Frigo, M., and Johnson, S. G. The fastest Fourier transform in the west. Technical Report MIT/LCS/TR-728, Massachusetts Institute of Technology, Sept. 1997.

[16] Gardner, R. J. *Geometric Tomography*, vol. 58 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 1995.

[17] Gonzalez, R. C., and Woods, R. E. *Digital Image Processing*. Addison-Wesley Publishing Company, 1993.

[18] Gropp, W., and Lusk, E. Jumpshot. `http://www-unix.mcs.anl.gov/mpi/mpich/docs/userguide/node40.htm#Node40`.

[19] Gropp, W., and Lusk, E. mpirun. `http://www-unix.mcs.anl.gov/mpi/www/www1/mpirun.html`.

[20] Gropp, W., and Lusk, E. Upshot and nupshot. `http://www-unix.mcs.anl.gov/mpi/mpich/docs/userguide/node39.htm#Node39`.

[21] Gropp, W., and Lusk, E. User's guide for mpe: Extensions for mpi programs. `http://www-unix.mcs.anl.gov/mpi/mpich/docs/mpeguide/paper.html`.

[22] Hawick, K. A., Grove, D. A., and Vaughan, F. A. Beowulf - A New Hope for Parallel Computing? In *Proc. of the 6th IDEA Workshop, Rutherglen* (Jan. 1999). Also available as DHPC Technical Report DHPC-061.

[23] Helgason, S. *The Radon Transform*, 2 ed. Birkhäuser, 1999.

[24] Herman, G. T. *Geometry of Digital Spaces*. Birkhäuser, 1998.

[25] Hoffman, F., and Hargove, W. Parallel computing with Linux. *Crossroads, the ACM student magazine 6*, 1 (1999), 23–27.

[26] Jähne, B. *Practical Handbook for Scientific Applications.* CRC Press, 1997.

[27] Kernighan, B. W., and Ritchie, D. M. *The C programming language.* Prentice-Hall, Englewood Cliffs (NJ), 1988. D.3.2 C, D.3.3.

[28] Kivioja, T. Virusten automaattinen paikantaminen elektronimikroskooppikuvista. Master's thesis, Tietojenkäsittelytieteen laitos, Helsingin yliopisto, 1999.

[29] Kotz, D. *GNUPLOT LaTeX Tutorial Version 2.0.* Computer Science Department, Duke University, Durham, NC, USA, Feb. 1990. See also [40].

[30] Lanzavecchia, S., and Bellon, P. L. Fast computation of 3D Radon transform via a direct Fourier method. *Bioinformatics 14*, 2 (1998), 212 – 216.

[31] Liang, Z.-P., and Lauterbur, P. C. *Magnetic Resonance Imaging: A Signal Processing Approach.* IEEE Press, 1999.

[32] Lichtenbelt, B. B. A. Fourier volume rendering. Tech. rep., Hewlett Packard Laboratories, Visual Computing Department, 1995.

[33] Malzbender, T. Fourier volume rendering. *ACM Transactions on Graphics 12*, 3 (July 1993), 233–250.

[34] Mukundan, R., and Ramakrishnan, K. R. *Moment Functions in Image Analysis.* World Scientific, 1998.

[35] Pressman, R. S. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, 1994.

[36] Ravantti, J. Virusten konstruointiongelma. Esitelmä kurssilla Tietojenkäsittelytieteen esittely, Helmikuu 2000.

[37] Shoemake, K. Euler angle conversion. In *Graphics Gems IV*, P. Heckbert, Ed. Academic Press, Boston, 1994, pp. 222–229.

[38] Sterling, T., Savarese, D., Becker, D. J., Dorband, J. E., Ranawake, U. A., and Packer, C. V. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing* (Oconomowoc, WI, Aug. 1995), pp. I:11–14.

[39] Toft, P. *The Radon Transform: Theory and Implementation.* PhD thesis, Department of Mathematical Modelling, Technical University of Denmark, 1996.

[40] Williams, T., Kelley, C., Campbell, J., Kotz, D., and Lang, R. *GNUPLOT: An Interactive Plotting Program.* Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA, Tel: (617) 876-3296, USA, Aug. 1990. Available in several Internet archives, including the Free Software Foundation collection on `prep.ai.mit.edu`. GNUPLOT can produce output for many different devices, including LaTeX picture mode, PS, and the X Window System. See also [29].

Luku 1

Johdanto

1.1 Katsaus dokumenttiin

Tämä dokumentti käsittelee Orientointijärjestelmän suunnitteluun liittyvät asiat, jonka toteuttaa kevään 2000 Ohjelmistotuotantoprojektiryhmä Beklu.

Luvussa 1 annetaan yleiskuva dokumenttiin ja lisäksi esitellään dokumentissa käytetty teminologia. Luku 2 käsittelee projektin taustaa ja sen suhtautumista muihin alan sovelluksiin. Luku 3 antaa yleiskuvauksen toteutettavasta järjestelmästä. Luku 4 käsittelee määrittelydokumenttiin tehdyt muutokset.

Luvut 5, 6 ja 7 esittelevät järjestelmän arkkitehtuurin, sen toteutuksessa käytetyt tietorakenteet ja toteutettavat algoritmit. Luku 8 käsittelee toteutettavaa hajautusstrategiaa, luku kymmenen järjestelmän 9 ja luku 10 esittelee järjestelmän toteutukseen liittyviä käytäntöjä. Luku 11 esittelee alustavasti järjestelmän testausta. Tämä luku kuitenkaan ei ole testaussunnitelma.

Liitteeksi A dokumentin loppuun on liitetty järjestelmän suorituskykyyn liittyviä laskelmia.

1.2 Terminologiaa

Seuraavassa on esitetty tässä dokumentissa yleisesti käytettyä terminologiaa ja käsitteistöä:

makromolekyylikompleksi Makromolekyyleistä koostuva partikkeli, esimerkiksi virus

elektronimikroskooppikuva Elektronimikroskoopilla otetusta kuvasta poimittu pala, jossa on yhden makromolekyylikompleksin kuva

pizza Ks. elektronimikroskooppikuva

kolmiulotteinen malli Kolmiulotteinen tiheyskartta, jonka perusteella elektronimikroskooppikuvien orientointi tapahtuu

projisointi Kolmiulotteisen mallin läpivalaisun simulointi.

projektio Projisoinnin tuloksena saatu mallin läpivalaisukuva

MRC Tiedostoformaatti, jossa elektronimikroskooppikuvat ja mallit ovat [2]

PGM Kuvankäsittelyohjelmien ymmärtämä tiedostoformaatti, jota käytetään esim. testattaessa erilaisten suotimien vaikutusta elektronimikroskooppikuviin

suunta Mallin katselusuunta kolmiulotteisessa avaruudessa

kulma Ks. suunta, myös kaksiulotteisten kuvien välinen kiertokulma

orientaatio suunnan esitystapa kolmiulotteisessa avaruudessa, katso 6.2

hyvyysinformaatio Vertailun onnistumista kuvaavaa tietoa, jonka tarkempi tulkinta riippuu käytettävästä vertailualgoritmista

MPI Message Passing Interface, rinnakkaisuuden hallintaan käytetty kirjasto [9, 3]

mpich Projektin käyttämä vapaa MPI-toteutus, vastaa lähinnä MPI-standardia 1.2

MPE MPI Extension -kirjasto, käytetään järjestelmän monitoroinnissa [21]

FFTW Fastest Fourier Transform in the West, Fourier-muunnosten laskemiseen käytetty kirjasto [15]

klusteri Erillisistä tietokoneista koottu tehokas rinnakkaislaskentayksikkö

Beowulf-klusteri Linux-työasemista koottu klusteri [22, 25]

Perl Practical Extraction and Report Language, konfiguraatiotiedostossa käytetty ohjelmointikieli [6]

C Ohjelmointikieli, jolla järjestelmä pääasiassa toteutetaan [7, 27]

SISD Single Instructions, Single data, kaikkein perinteisin tietokoneen malli, jossa yhdellä prosessorilla on oma yksityinen muistiavaruutensa [11]

DM-MIMD Distributed Memory - Multiple Instructions Multiple Data, malli, joka sopii kuvaamaan työasemaklusterin rinnakkaisuutta, vaihtoehtoisesti NUMA (Non Uniform Memory Access) [11]

SIMD Single Instructions Multiple Data, eri prosessorit ajavat samaa ohjelmaa ns. 'lock-step' tyylillä. Tämä tarkoittaa, että prosessorit suorittavat samat konekieliset käskyt siten, että jokainen käsky suoritetaan kullakin prosessorilla samalla hetkellä [11]

ei-blokkaava lähetys lähetysoperaatio tapahtuu samanaikaisesti muun ohjelman suorituksen kanssa, katso luku 8.2.3

synkroninen viestinvälitys Lähettäjä ei aloita lähetystä ennen kuin vastaanottaja on ilmoittanut valmiutensa, katso luku 8.2.4

white-box -testaus algoritmin testaus kaikilla suorituspoluilla, katso luku [35]

black-box -testaus järjestelmän testaaminen aineistolla, josta saatavat tulokset tiedetään, kaso [35]

Luku 2

Projektin tausta

2.1 Makromolekyylikompleksien rekonstruointi

Makromolekyylien rekonstruktioprosessin tavoitteena on muodostaa makromolekyylikompleksista kolmiuloitteinen malli elektronimikroskoopilla tuotetuista läpivalaisukuvista [13]. Elektronimikroskooppi suurentaa yli 30000-kertaiseksi, joten se on riittävä esimerkiksi viruksien rakenteen tutkimiseen. Ennen läpivalaisukuvan ottamista on makromolekyylikompleksit sisältävä näyte preparoitava, esimerkiksi jäädyttämällä vedessä oleva näyte nopeasti upottamalla nestemäiseen etaaniin, jottei veteen ei ehtisi muodostua jään kiderakennetta. Kun näyte on jäädytetty, siihen kohdistetaan elektronisuihku, joka muodostaa filmille läpivalaisukuvan. Näytteen kuvaamisen jälkeen voidaan filmi digitoida kuvanlukijalla tietokoneen ymmärtämään muotoon [36].

Digitoidusta läpivalaisukuvasta voidaan makromolekyylikompleksin partikkelit poimia joko manuaalisesti tai käyttäen sitä varten suunniteltua ohjelmaa, esimerkiksi ETHAN-ohjelmaa [28]. Osakuvista pyritään poistamaan mahdollisimman paljon kohinaa, esimerkiksi rajaamalla kuva mahollisimman pieneksi ja poistamalla osa taustasta, sillä usein kuvassa on enemmän kohinaa kuin itse signaalia.

Partikkelit sisältävistä osakuvista oletetaan, että ne esittävät samaa kolmiulotteista kappaletta, mutta kaksiulotteisina projektioina eri suunnalta katsottuina. Radonin teoreemaan pohjautuen näistä kaksiulotteisista projektioista voidaan periaatteessa rekonstruoida makromolekyylikompleksin alkuperäinen kolmiulotteinen rakenne [16]. Usein oletetaan myöskin, että rekonstruoitava kappale on hyvin symmetrinen, jolloin mallin rekonstruoiminen helpottuu. Symmetria-oletuksella voidaan helpottaa mallin rekonstruointia, sillä malli koostuu useasta toistuvasta osiosta (katso kuva 2.1) [12].

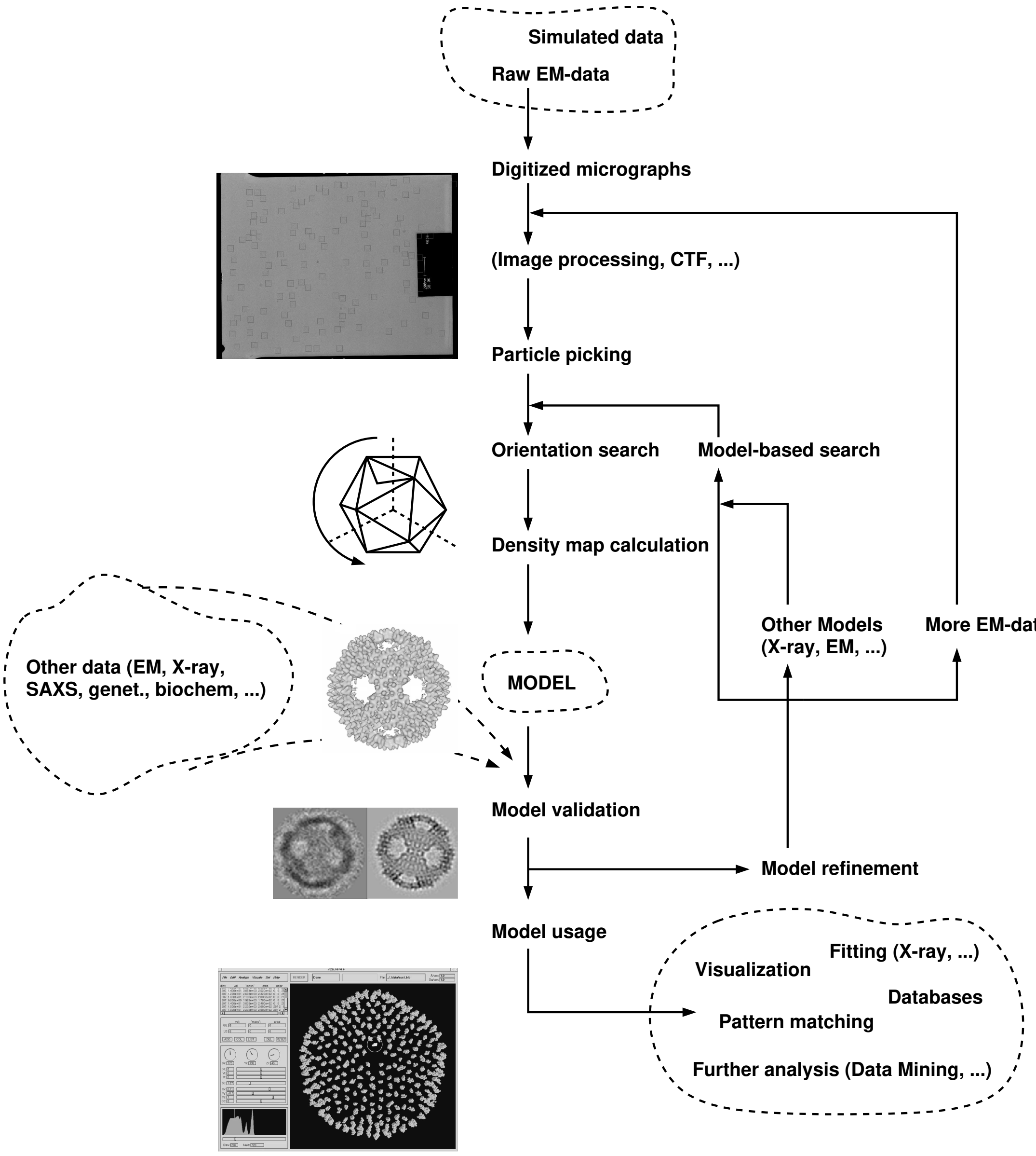
2.2 Mallin tarkentamiseen perustuva rekonstruointi

Mallin tarkennukseen perustuvassa rekonstruonnissa muodostetaan ensiksi rekonstruoitavan kappaleen alustava malli. Alustava malli voidaan muodostaa jakamalla valitut kaksiuloitteiset läpivalisukuvat samantapaisten joukoiksi ja laskemalla niiden sisältämien partikkeleiden perusteella malli, tai käyttämällä hyväksi jo olemassa olevia malleja. Alustavan mallin luomisen jälkeen mallia ruvetaan tarkentamaan laskemalla mallista projektiota eri orientaatiossa tietyn kulma-askeleen välein. Projektioita verrataan valittuun partikkeliin ja valitaan sopivin orientaatio partikkelin orientaatioksi, jonka jälkeen lasketaan uusi parannettu malli. Mallin tarkentamista jatketaan haluttu määrä iteraatioita.

2.3 Orientointijärjestelmä rekonstruointijärjestelmän osana

Orientointijärjestelmän tehtävänä on selvittää syötteenään saamiensa elektronimikroskooppikuvan palasien sisältämien partikkeleiden orientaatio annettuun kolmiulotteiseen malliin nähden. Orientaatio selvitetään laskemalla mallista projektioita eri kulmissa ja vertailemalla niitä partikkeleihin. Vertailuilla määritetään partikkeliin sopiva projektio, eli orientaatio pallokoordinaatistossa. Koska partikkelien kuvissa esiintyy paljon kohinaa ja malli ei välttämättä ole kovinkaan tarkka, orientointijärjestelmä tuottaa partikkeleille useita sopivimpia orientaatiovaihtoehtoja, sekä hyvyysinformaatiota ilmoittaakseen kuinka hyvin kyseinen orientaatio sopii partikkeliin (jonkin mitan mukaan).

Orientaatiojärjestelmän avulla voidaan mallin muodostavien partikkeleiden orientaatiota tarkentaa, jolloin niiden pohjalta voidaan laskea uusi tarkempi malli. Virheellisten orientaatioiden merkitystä pyritään vähentämään käyttämällä riittävän



Kuva 2.1: Makromolekyylikompleksin rekonstruointiprosessi

paljon partikkeleita laskennan pohjana. Tällöin tarkan orientaation määrittäminen on hankalaa, sillä orientaation laskeminen on raskasta ja tarkan orientaation selvittämiseksi projektioiden muodostuksessa tulee olla riittävän pieni kulma-askel, joten tuloksien saamiseen järkevässä ajassa vaati paljon laskentatehoa. Suuren laskentatehon käytön vuoksi on laskenta järkevää hajauttaa usean eri koneen laskettavaksi [38].

Luku 3

Järjestelmän yleiskuvaus

Orientaatiojärjestelmä selvittää viruksesta elektronimikroskoopilla otetun kuvan orientaation laskemalla kolmiuoitteisesta mallista projektioita ja vertaamalla näitä kaksiulotteisissa kuvissa oleviin viruksiin. Järjestelmä toimii Beowulf-klusterissa Linux-ympäristössä ja käyttää hajautukseen MPI-kirjaston (Message Passing Interface) vapaasti levitettävää versiota.

Järjestelmä saa syötteenä yhden kolmiulotteisen mallin viruksesta, useita kaksiulotteisia elektronimikroskooppikuvia viruksesta ja käyttäjän haluamakseen muokkaaman konfiguraatiotiedoston, josta luetaan esimerkiksi monestako kulmasta projektioita lasketaan. Lopputuloksena järjestelmä tulostaa laskemansa hyvyysinformaation perusteella parhaat orientaatiot kuville. Parhaita orientaatioita voi yhdellä kuvalla olla useampia, jos käyttäjä näin haluaa.

Prosessi on laskennallisesti hyvin vaativa, jonka takia järjestelmä on hajautettu klusteriin, jolloin järjestelmä tuottaa tuloksia järkevässä ajassa. Orientointijärjestelmä on rakenteeltaan suhteellisen yksinkertainen, aluksi lasketaan viruksen kolmiulotteisesta mallista projektio määrätystä kulmasta ja tätä verrataan kaksiulotteiseen kuvaan viruksesta, vertailun tuloksena saadaan hyvyysinformaatiota, josta päätellään onko kaksiulotteinen kuva otettu tästä kulmasta, tämän jälkeen lasketaan projektio seuraavasta kulmasta ja verrataan, näin edetään kunnes kaikki halutut kulmat on käyty läpi. Tässä prosessissa käytettävät algoritmit ovat kuitenkin laskennallisesti vaativia ja tutkittavia kulmia on runsaasti samoin kuin kaksiulotteisia kuvia, joten laskentatehoa tarvitaan runsaasti, jolloin klusteri on hyvä laitteistovaihtoehto, sillä klusterissa laskenta voidaan hajauttaa MPI-kirjastoa käyttäen kaikille käytettävissä oleville laskentayksiköille.

Järjestelmän vikasietoisuuden kannalta klusteri on kuitenkin ongelmallinen laitteisto, sillä missä tahansa laskentayksikössä voi tapahtua järjestelmästä riippumattomaton virhe, jonka seurauksena kyseinen yksikkö on käyttökelvoton eli laskentayksikkö on kaatunut, jonka jälkeen järjestelmä pystyy jatkamaan lasketaan tästä huolimatta eli hajauttaamaan laskennan jäljellä oleviin laskentayksiköihin. Lisäksi järjestelmän selviää tilanteesta, jossa käyttökelvoton laskentayksikkö tulee jälleen käyttökelpoiseksi, tämä on ongelmallinen tilanne, sillä käyttettävä MPI-kirjastosta käytössä oleva versio ei mahdollista uusien prosessien luontia järjestelmän ollessa toiminnassa. Tämä tilanne on ratkaistu pitämällä yllä lokitiedostoa laskennan tilasta, jolloin järjestelmä käynnistetään uudestaan ja jatketaan siitä tilasta johon äsken jäätiin.

Luku 4

Määrittelydokumentti ja siihen tehdyt muutokset

4.1 Muutokset suorituskykyvaatimuksiin

Suunnitteluprosessin aikana on käynyt ilmi, ettei järjestelmälle asetettuja suorituskyky- ja keskusmuistivaatimuksia todennäköisesti kaikilta osin pystytä täyttämään.

4.1.1 Keskusmuisti

Keskusmuistia tarvitaan ainakin projektioita laskevissa solmuissa yli 128 megatavua, ehkä jopa 512 megatavua. Tähän ovat vaikuttaneet muun muassa vaatimukset projektion laskemisesta myös Fourier-avaruuden kautta sekä tehokkamman projektioalgoritmin käyttö, jossa mallia käännetään keskusmuistissa [30].

4.1.2 Suorituskyky

Suorituskyvyn osalta tavoitteena on edelleen se, että järjestelmä pystyy orientoimaan 1000 elektronimikroskooppikuvaa n. 7 päivässä, mutta tämän tavoitteen toteutuminen ei ole varmaa. Laskelmia suorituskykyvaatimusten toteutumisista eri lähtöarvoilla ja suoritusympäristökokoonpanoilla on liittessä A.

Luku 5

Arkkitehtuurin kuvaus

5.1 Järjestelmän toimintaperiaate

Orientointijärjestelmä saa syötteenään joukon elektronimikroskooppikuvia ja kolmiulotteisen mallin. Tulosteenaan järjestelmä antaa jokaiselle elektronimikroskooppikuvalle hyvyysinformaatiolla varustetun listan orientaatioista, joissa kuvat parhaiten sopivat malliin. Tässä luvussa kerrotaan, miten järjestelmä laskee tulosteensa syötteiden perusteella.

Järjestelmän toiminnan perusidea on seuraava: kolmiulotteisesta mallista lasketaan ensin projektioita eri suunnista. Sitten kutakin orientoitavaa elektronimikroskooppikuvaa verrataan jokaiseen laskettuun projektioon. Tulosteenaan järjestelmä palauttaa jokaista orientoitavaa elektronimikroskooppikuvaa kohden listan niistä suunnista, joista lasketut projektiot parhaiten sopivat orientoitavaan kuvaan. Lisäksi järjestelmä tulostaa hyvyysinformaatiota, joka kuvaa sitä kuinka hyvin mallin projektiot ja elektronimikroskooppikuvat sopivat toisiinsa. Orientaation löytämiseksi järjestelmän on siis vain laskettava mallin projektiot kaikista eri suunnista, verrattava orientoitavaa kuvaa kuhunkin projektioon ja palautettava ne suunnat, joista otetut projektiot käytetyn vertailumenetelmän mielessä vastaavat parhaiten orientoitavaa elektronimikroskooppikuvaa.

Valitettavasti edellä kuvatun menetelmän toteuttaminen käytännössä on hankalaa. Ensinnäkään mallista ei edes periaatteessa voi laskea projektioita kaikista suunnista, sillä suuntia on äärettömän monta. Toisekseen orientoitavien kuvien ja projektioiden vertailu ei ole aivan yksinkertaista, muun muassa siksi, että elektronimikroskooppikuvissa on erittäin paljon kohinaa.

Järjestelmälle on asetettu suorituskykyvaatimuksia, joiden täyttämiseksi laskenta hajautetaan ryhmän käytössä olevaan Beowulf-klusteriin. Hajautuksesta on kerrottu tarkemmin luvussa 8.1.

Seuraavissa aliluvuissa esitellään lyhyesti järjestelmän tärkeimpiä toimintoja, projisointia ja vertailua.

5.1.1 Projisointi

Kolmiulotteisen mallin projektioita voidaan ajatella mallin diskreetteinä Radonmuunnoksina [23]. (Radon-muunnoksen toteuttamisesta on kerrottu tarkemmin luvussa 7.2.) Tässä luvussa tarkastellaan projisointialgoritmin käyttöä orientointijärjestelmän osana.

Elektronimikroskooppikuvan tarkka orientoiminen malliin nähden edellyttää sellaisen suunnan etsimistä, josta otettu mallin projektio sopii mahdollisimman hyvin orientoitavaan kuvaan. Tämän suunnan löytämiseksi olisi tarkasteltava kaikkia mallin eri projektioita, mikä kuitenkin olisi laskennallisesti aivan liian raskasta. Orientointijärjestelmän on siis tyydyttävä tarkan orientoinnin sijasta approksimoimaan todellista orientaatiota käytettävän laskenta-ajan rajoissa.

Orientointijärjestelmässä käytetään menetelmää, jossa malli projisoidaan jonkin suunta-asteikon kaikista suunnista ja orientoitavan kuvan orientaatio etsitään vertaamalla sitä kaikkiin näin saatuihin projektioihin. Tässä menetelmässä projektioita laskettaessa käytettävä suunta-asteikko määrää siis ylärajan sille, kuinka tarkkoja orientointituloksia järjestelmä voi saavuttaa. Tulosten tarkentamiseksi asteikkoa voidaan tihentää, mutta tällöin laskenta-aika vastaavasti kasvaa – koko järjestelmän laskenta-aika on suoraan verrannollinen projisoinnissa käytetyn suunta-asteikon suuntien lukumäärään.

Projektioiden laskeminen mallista on huomattavasti raskaampaa kuin kaksiulotteisten projektioiden kiertäminen tasossa keskipisteensä ympäri. Siksi on edullista laskea mallista projektioita pitäen yhtä projisointisuunnan vapausastetta kiinnitettynä ja suorittaa tämän vapausasteen ilmaisemat kierrot vasta valmiille projektioille tasossa. Jos projisointia ajatellaan valokuvan ottamisena mallista, tarkoittaa edellinen siis sitä että kameraa ei kuvatessa kierretä kuvaussuuntaa vastaavan akselinsa ympäri, vaan vastaava kierto suoritetaan vasta kehitetyille valokuville. Mallista otettavien projektioiden määrä vähenee tätä menettelyä käytettäessä merkittävästi, tyypillisesti $\Theta(N)$ projektiosta $\Theta(N^{2/3})$ projektioon.

5.1.2 Vertailu

Elektronimikroskooppikuvien vertaaminen kolmiulotteisen mallin projektioihin on järjestelmän laskennallisesti raskain osa. Tämä johtuu siitä, että sekä elektronimikroskooppikuvia että projektioita on hyvin paljon[1], ja kutakin elektronimikroskooppikuvaa on verrattava kuhunkin projektioon. Lisäksi osa vertailualgoritmeista on sinänsä laskennallisesti melko raskaita.

Vertailualgoritmit jakautuvat karkeasti ottaen kahteen eri luokkaan, tavallisiin vertailualgoritmeihin ja rotaatioinvariantteihin vertailualgoritmeihin. Tavalliset vertailualgoritmit mittaavat kuvien etäisyyttä toisiinsa esimerkiksi laskemalla kuvien välisen neliövirheen. Rotaatioinvariantit vertailualgoritmit pystyvät vertaamaan kuvia järkevästi toisiinsa, vaikka toista olisikin kierretty keskipisteensä ympäri. Vertailualgoritmeista on kerrottu tarkemmin luvussa 7.3.

Rotaatioinvariantteja vertailualgoritmeja käyttäen voidaan mallista otettuja projektioita verrata suoraan elektronimikroskooppikuviin ottamatta ensin huomioon kameran kiertoa (katso luku 6.2). Vertailuja tarvitaan tällöin vain yksi jokaista mallista otettua projektiota kohden, jolloin vertailujen kokonaismäärä pienenee kameran kiertokulmien määrää vastaavalla kertoimella. Koska vertailujen lukumäärä on tärkein yksittäinen järjestelmän laskenta-aikaan vaikuttava tekijä, on edellämainitulla säästöllä ratkaiseva vaikutus järjestelmän suorituskykyyn.

Vertailutulosten parantamiseksi elektronimikroskooppikuvia esikäsitellään esimerkiksi suodattamalla niistä pois tutkittavia makromolekyylikomplekseja ympäröivä vesi ja normittamalla ne saman painoisiksi kuin kolmiulotteisen mallin projektiot. Näistä operaatioista on kerrottu tarkemmin luvuissa 5.2.4.3.11 ja 5.2.4.3.13.

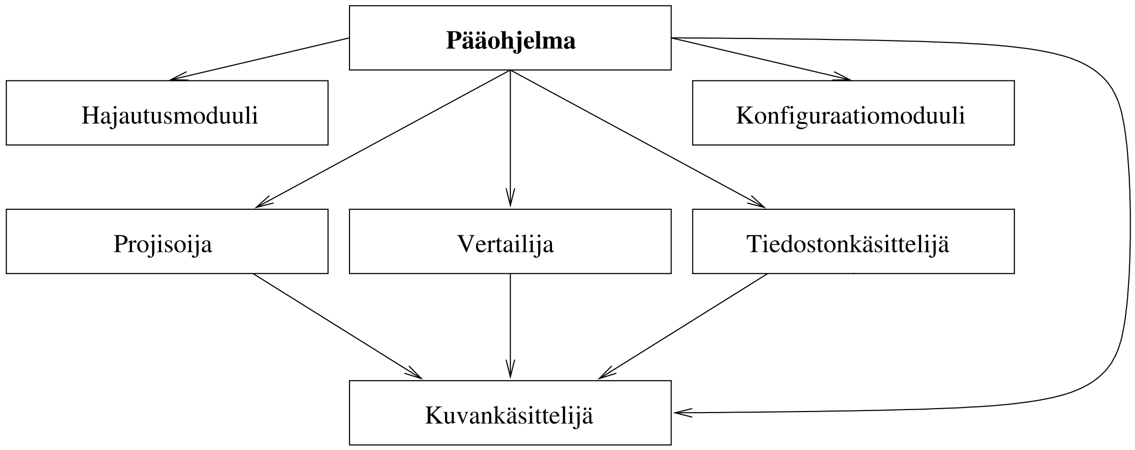
[1] Elektronimikroskooppikuvia on tyypillisesti noin tuhat, projektioita satoja tuhansia.

5.2 Moduulit ja niiden väliset suhteet

5.2.1 Moduulihierarkia

Järjestelmä koostuu pääohjelmasta ja kuudesta moduulista: Hajautusmoduuli, Konfiguraatiomoduuli, Projisoija, Vertailija, Tiedostonkäsittelijä ja Kuvankäsittelijä. Näiden moduulien kutsusuhteet on esitetty kuvassa 5.1. Lisäksi järjestelmässä on toteutettu sekalaisia apufunktioita liittyen esimerkiksi muistinhallintaan ja virhetilanteisiin. Näitä käyttävät kaikki moduulit, joten siitä ei ole joka moduulin kohdalla erikseen mainittu.

Hajautusmoduuli tarjoaa hajautuksen toteutukseen tarvittavat rutiinit. Konfiguraatiomoduulissa toteutetaan konfiguraatiotiedoston käsittelyyn liittyvät rutiinit. Projisoija-moduuli tarjoaa kolmiulotteisen mallin projisointirutiinit ja Vertailija-moduuli kaksiulotteisten kuvien vertailurutiinit. Tiedostonkäsittelijä-moduulissa toteutetaan eri kuvatiedostoformaattien lataus- ja talletusrutiinit. Kuvankäsittelijä-moduuli tarjoaa kaksi- ja kolmiulotteisten kuvien käsittelyyn tarvittavat tietorakenteet ja rutiinit. Moduulien tarkemmat kuvaukset ovat jäljempänä.



Kuva 5.1: Moduulien väliset suhteet

5.2.2 Projisoija

5.2.2.1 Yleiskuvaus

Projisoija-moduuli koostuu funktioista, joita käytetään laskettaessa kaksiulotteisia projektioita kolmiulotteisesta mallista. Moduuli on läheisessä yhteistyössä muun muassa Kuvankäsittelijä-moduulin kanssa.

Projisoinnissa käytetyt algoritmit on kuvattu luvussa 7.2.

5.2.2.2 Toiminnot

5.2.2.2.1 Reaalimallin projisointi

```
image_real *project_real(image_real *in,
 image_real *out,
 orientation o);
```

Funktio `project_real` laskee kolmiulotteisesta mallista `in` projektion suunnasta `o`. Projektion talletukseen käytetään `out`:in osoittamaa kuvatietorakennetta, jonka kokoa muutetaan tarvittaessa. Jos `out` on `NULL`, funktio luo uuden kuvatietorakenteen tulokselle ja palauttaa osoittimen siihen. Projisointiin käytetään luvussa 7.2.1 esiteltyä algoritmia, joka laskee projektiot ampumalla kolmiulotteisen mallin `in` läpi projisointisuunnan `o` suuntaisia säteitä kääntämättä mallia.

```
image_real *project_around_X_real(image_real *in,
  image_real *out,
  double angle);
```

```
image_real *project_around_Y_real(image_real *in,
  image_real *out,
  double angle);
```

```
image_real *project_around_Z_real(image_real *in,
  image_real *out,
  double angle);
```

Funktiot `project_around_X_real`, `project_around_Y_real` ja `project_around_Z_real` laskevat kolmiulotteisesta mallista `in` suunnista (`alpha`, 0, 0), (0, `alpha`, 0) ja (0, 0, `alpha`). Tulokset talletetaan `out`:n osoittamaan kuvatietorakenteeseen, mikäli `out != NULL`. Tässä tapauksessa oletetaan, että taulukon alkioiden osoittamat kuvatietorakenteet ovat kaksiulotteisia ja x- ja y-dimensioiltaan yhteneviä mallin `in` dimensioiden kanssa. Jos `out == NULL`, varataan tulokselle kuvatietorakenne `image_real`. Projisointialgoritmina käytetään luvussa 7.2.1 esiteltyä menetelmää, jossa valmiiksi käännetyn kolmiulotteisen mallin `in` läpi ammutaan projisointisäteitä `angle`:n ilmoittamassa suunnassa.

5.2.2.2.2 Kompleksimallin projisointi

Kompleksimallin projisointifunktiot ovat kutsurajapinnoiltaan vastaavat kuin reaalimallin projisointifunktiot:

```
image_real *slice_real(image_real *in,
                       image_real *out,
                       orientation o, double r);

image_complex *slice_complex(image_complex *in,
     image_complex *out,
     orientation o, double r);
```

Funktio `slice_real` poimii orientaation `o` osoittamasta suunnasta `r` etäisyydeltä syötteen keskipisteestä siivun tulostaulukkoon, jolle varataan muisti tarvittaessa.

Funktio `slice_complex` toimii ulkoisesti täsmälleen samoin kuin reaaliversionsa, paitsi että sen palauttamat siivut ovat kompleksisia. (Tätä funktiota voitaisiin periaatteessa käyttää asymptoottisesti nopeampaan projisointiin, jos osattaisiin interpoloida vakioajassa.)

5.2.2.2.3 Maximum Intensity Projection -visualisointi

```
image_uchar *MIP_uchar(image_uchar *in,
       image_uchar *out,
       orientation o);
```

```
image_real *MIP_real(image_real *in,
     image_real *out,
     orientation o);

image_real *MIP_around_X_real(image_real *in,
      image_real *out,
      double angle);

image_real *MIP_around_Y_real(image_real *in,
      image_real *out,
      double angle);

image_real *MIP_around_Z_real(image_real *in,
      image_real *out,
      double angle);

image_real *halfMIP_real(image_real *in,
 image_real *out,
 orientation o);
```

Funktiot tuottavat Maximum Entropy Projection -visualisointeja (MIP -visualisointjea kolmiulotteisista kuvista. Jos osoitin tuloskuvaan on `NULL`, funktiot varaavat itse muistin sitä varten. Funktio `halfMIP_real` toimii kuten muutkin MIP-visualisoijat, mutta huomioi syötteestä vain katselusuuntaa lähinnä olevan puolikkaan. MIP-visualisointi on kuvailtu mm. raportissa [32].

5.2.3 Vertailija

5.2.3.1 Yleiskuva

Vertailija-moduuli muodostuu kokoelmasta kaksiulotteisten kuvien vertailuun tarkoitettuja funktioita ja niihin liittyviä tietorakenteita. Vertailija-moduuli käyttää muun muassa Kuvankäsittelijän palveluita.

5.2.3.2 Tietorakenteet

Moduulissa Vertailija on seuraavat julkiset tietorakenteet:

```
typedef struct {
  double angle;
  double quality_info;
} angle_quality;

typedef struct {
  int index;
  double quality_info;
} index_quality;

struct sect {
  struct sect *next;
  int index;
  double weight;
};

typedef struct sect sector;

typedef struct {
  int w, h, steps;
  sector *first;
} sectors;
```

Tietorakennetta `angle_quality` käytetään rotaatioinvarianttien vertailualgoritmien palautusarvojen välityksessä. Kenttä `angle` ilmoittaa kulman, jonka hyvyyttä tietorakenne kuvaa. Kentässä `quality_info` välitetään vertailun tulosta kuvaavaa hyvyysinformaatiota.

Tietorakennetta `index_quality` käytetään rotaatioinvarianttien vertailualgoritmien palautusarvojen välityksessä. Kenttä `index` ilmoittaa indeksin, jonka hyvyyttä tietorakenne kuvaa. Kentässä `quality_info` välitetään vertailun tulosta kuvaavaa hyvyysinformaatiota.

Tietorakenne `sectors` määrittelee `w` $\times$ `h` -kokoisen yksiulotteisen taulukon, jonka alkiot ova tyyppiä `sector`. Nämä alkiot sisältävät tiedon siitä, mihin sektoreihin kuvapisteen massa jakautuu (`index` on sektorin järjestysnumero ja `weight` on kuvapisteen paino sektorissa), ja muodostavat linkitetyn listan, jos piste jakautuu useampaan kuin yhteen sektoriin (`next` -kenttä). Kenttä `sectors.steps` ilmoittaa kuinka moneen sektoriin kuvan ala on jaettu. Tietorakennetta käytetään kuvan massan tehokkaaseen laskemiseen sektoreittain, mitä voidaan käyttää apuna vertailtaessa kuvia rotaatioinvariantisti.

5.2.3.3 Toiminnot

Vertailualgoritmit jakautuvat karkeasti kahteen luokkaan. Ensimmäisen luokan muodostavat tavanomaiset kaksiulotteisten kuvien vertailualgoritmit, jotka eivät pyri ottamaan huomioon kuvien kiertoja toisiinsa nähden. Toiseen luokkaan kuuluvat rotaatioinvariantit algoritmit. Nämä algoritmit ovat itse asiassa filttereitä, joilla etsitään vertailtavien kuvien kiertoasentoja joissa ne sopivat hyvin toisiinsa. Vertailualgoritmeja on kuvattu tarkemmin luvussa 7.3.

5.2.3.3.1 Tavalliset vertailut

```
real square_distance_real(image_real *in1, image_real *in2);
real correlation_coefficient_real(image_real *in1, image_real *in2);
```

Funktio `square_distance_real` laskee kuvien `in1` ja `in2` neliövirheen. Funktio `correlation_coefficient_real` laskee kuvien `in1` ja `in2` neliövirheen.

5.2.3.3.2 Rotaatioinvariantit vertailualgoritmit

```
image_complex *compare_polar_prepare_real(image_real *in,
                                          image_complex *out,
                                          int steps,
                                          real *scale,
                                          fftw_plan fft);

index_quality *compare_polar_real(image_complex *in1,
```

```
                                            image_complex *in2,
                                            index_quality *out,
                                            int n,
                                            fftw_plan ifft);
```

Funktio `compare_polar_prepare_real` valmistaa kuvan tehokkaammin vertailtavaan muotoon ja `compare_polar_real` vertaa napakoordinaattimuodossa olevia Fourier-kuvia `in1` ja `in2` käyttäen luvussa 7.3.1 esiteltyä menetelmää. Taulukossa `scale` on skaalauskertoimet säteen mukaan. Vertailtavien kuvien pitää olla kaksiulotteisia ja niiden dimensioiden pitää olla samat. Parametri `n` ilmoittaa kuinka monta parasta löydettyä vaihtoehtoa palautetaan. Taulukkoa `out`, jonka koon oletetaan olevan vähintään `n`, käytetään palautusarvon välittämiseen: taulukkoon talletetaan parhaat löydetyt kulmat hyvyysinformaation mukaan järjestettynä. Jos `out == NULL`, funktio varaa tuloksen tallettamista varten kokoa `n` olevan `index_quality`-taulukon. Parametri `fftw_plan` on FFTW-kirjaston [15] generoima suunnitelma Fourier-muunnoksen laskemiseksi.

Funktio palauttaa osoittimen tulostaulukkoon.

```
image_complex *filter_polar_prepare_real(image_real *in,
                                         image_complex *out,
                                         int steps,
                                         real *scale,
                                         fftw_plan fft);

index_quality *filter_polar_real(image_complex *in1,
                                 image_complex *in2,
                                 index_quality *out,
                                 int n,
                                 fftw_plan ifft);
```

Funktio `filter_polar_prepare_real` muuttaa kuvat tehokkaammin vertailtavaan muotoon ja `filter_polar_real` vertaa napakoordinaattimuodossa olevia yksiulotteisia Fourier-kuvia `in1` ja `in2` laskemalla niiden korrelaation. Muiden parametrien ja palautusarvojen käsittely hoidetaan samoin kuin ylläkuvatussa funktiossa `compare_polar_real`.

```
sectors *create_sectors(int steps, int w, int h);

angle_quality *compare_sectors_image_real(sectors *sector_table,
                                          image_real *in1,
                                          image_real *in2,
                                          int n_best,
                                          angle_quality *result_array);

image_complex *polar_distribution(sectors *sect,
  image_real *in,
  image_complex *out,
  fftw_plan fftw);

index_quality *compare_polar_distributions(image_complex *in1,
   image_complex *in2,
   index_quality *out,
   int n,
   fftw_plan ifftw);
```

Funktio `create_sectors` luo `w` × `h` -kokoisen sektoritaulukon (katso luku 5.2.3.2). Parametri `steps` ilmoittaa kuinka moneen tasaväliseen sektoriin taulukon ala jaetaan. Keskipisteenä sektorijaossa käytetään taulukon "keskipistettä" `(w/2, h/2)`. Funktio palauttaa osoitteen luotuun sektoritauluun. Taulukon dimensioiden `w` ja `h` sekä sektorien määrän `steps` on oltava positiivisia.

Funktio `compare_sectors_image_real` vertaa syötekuvia `in1` ja `in2` rotaatioinvariantisti käyttäen apuna parametrina annettua sektoritaulukkoa `sector_table`. Keskipisteinä vertailussa käytetään syötekuvien keskipisteitä. Paluuarvon ja parametrien `n_best` ja `result_array` merkitys ja käyttö ovat samat kuin funktiossa `compare_polar_complex`. Kuvien on oltava kaksiulotteisia ja niiden dimensioiden on oltava samat kuin annetussa sektoritaulukossa.

Funktio `polar_distribution` laskee paramatrina annetun kuvan `in` massajakauman sektoreittain, laskee tästä Fourier-muunnoksen ja sijoittaa tuloksen kompleksiseen tuloskuvaan `out`. Jakauman laskemiseen se käyttää parametrina annettua sektoritaulukkoa `sect`. Funktio palauttaa osoittimen tuloskuvaan. Jos parametri `out` on `NULL` funktio varaa muistin tulokselle.

Funktio `compare_polar_distributions` vertaa kahta annettua funktiolla `polar_distribution` laskettua massajakaumaa `in1` ja `in2` laskemalla niiden välisen korrelaatiokuvan ja etsimällä siitä parametrin `n` ilmoittaman määrän maksimiarvoja. Nämä maksimiarvot sijoitetaan tulosrakenteeseen `out`. Jos parametri `out` on `NULL` funktio varaa muistin tulokselle.

5.2.3.3.3 Muut algoritmit

```
real *find_minimums_real(image_real *in, int n, real *result_array);
```

Funktio `find_minimums_real` etsii kuvasta `in` `n` pienintä intensiteettiä. Pienimmät intensiteetit talletetaan suuruusjärjestykseen taulukkoon `result_array`, jonka koon oletetaan olevan vähintään `n`. Jos `result_array == NULL`, funktio varaa tuloksen tallettamista varten kokoa `n` olevan `real`-taulukon. Funktio palauttaa osoittimen tulostaulukkoon. Jos kuvassa on alle `n` pistettä, palautetaan kuitenkin `NULL`.

5.2.4 Kuvankäsittelijä

5.2.4.1 Yleiskuvaus

Kuvankäsittelijä-moduuli muodostuu kokoelmasta kaksi- ja kolmiulotteisten kuvien käsittelyyn tarkoitettuja toimintoja ja tietorakenteita. Kuvankäsittelijän toiminnot jakautuvat kahteen eri luokkaan: geometrisiin muunnoksiin ja kuvien intensiteettejä muokkaaviin toimintoihin. Koska edellämainittu jako on melko karkea ja keinotekoinen, on molemmat toimintoluokat päätetty sisällyttää samaan moduuliin.

5.2.4.2 Tietorakenteet

Moduuli sisältää seuraavat julkiset tietotyypit ja tietorakenteet:

```
typedef unsigned char uchar;
```

```
typedef fftw_real real;
```

```
typedef fftw_complex complex;

typedef struct {
  int type;
  int x, y, z;
  double x0, y0, z0;
  uchar *data;
} image_uchar;

typedef struct {
  int type;
  int x, y, z;
  double x0, y0, z0;
  real *data;
} image_real;

typedef struct {
  int type;
  int x, y, z;
  double x0, y0, z0;
  complex *data;
} image_complex;

typedef struct {
  int type;
  int x, y, z;
  double x0, y0, z0;
  void *data;
} image;

typedef struct {
  double a1, a2, a3;
} orientation;

typedef struct {
```

```
  double x, y, z;
} point;
```

Tietotyyppejä `uchar`, `real` ja `complex` käytetään kuvien pisteiden tallettamiseen. Tietotyypit `fftw_real` ja `fftw_complex` on lainattu FFTW-kirjastosta [15] yhteensopivuuden varmistamiseksi.

Tietorakenteita `image_uchar`, `image_real` ja `image_complex` käytetään kaksi- ja kolmiulotteisten reaali- ja imaginäärikuvien esittämiseen muistissa. Kuvien tyyppi on talletettu kenttään `type`, jonka mahdollisia arvoja ovat `IMAGE_UCHAR`, `IMAGE_REAL` ja `IMAGE_COMPLEX`. Kentät `x`, `y` ja `z` ilmoittavat kuvan dimensiot; jos `z == 1` kuva on kaksiulotteinen. Kentät `x0`, `y0` ja `z0` ilmoittavat kuvan keskipisteen sijainnin. Kenttä `data` on osoitin muistialueelle, johon kuvan pisteiden intensiteetit on talletettu. Pisteen, jonka koordinaatit ovat (i, j, k), intensiteetti talletetaan paikkaan `data[i + j*x + k*x*y]`. Tietorakenteiden `image_uchar`, `image_real` ja `image_complex` ainoa ero on kentän `data` tyypissä, joka on `uchar`, `real` tai `complex` vastaavasti.

Tietorakenne `image` on edellisessä kappaleessa esitettyjen kuvatietorakenteiden "yleistys" tietoliikenteen helpottamiseksi. Erona aikaisempiin on vain se, että kentässä `type` täytyy oikeasti kertoa tyyppi ja kentän `data` tyyppi on `void`.

Tietorakennetta `orientation` käytetään orientaatioiden esittämiseen. Tarkempi kuvaus löytyy luvusta Tietorakenteet (6.2).

Tietorakennetta `point` käytetään sijaintien esittämiseen kolmiulotteisessa avaruudessa.

Edellä mainittujen tietorakenteiden lisäksi moduulin toteutuksessa käytetään joitakin yksinkertaisia paikallisia tietorakenteita, joiden dokumentaatio löytyy lähdekoodin yhteydestä.

5.2.4.3 Toiminnot

Seuraavassa moduulin toiminnot toteuttavien C-funktioiden kuvaus:

5.2.4.3.1 Kuvatietorakenteiden muistinhallinta

```
image_uchar *create_image_uchar(int x, int y, int z);
```

```
image_uchar *copy_image_uchar(image_uchar *in);

void delete_image_uchar(image_uchar *in);

image_real *create_image_real(int x, int y, int z);

image_real *copy_image_real(image_real *in);

void delete_image_real(image_real *in);

image_complex *create_image_complex(int x, int y, int z);

image_complex *copy_image_complex(image_complex *in);

void delete_image_complex(image_complex *in);
```

Funktio `create_image_uchar` luo uuden kuvatietorakenteen, jonka dimensiot ovat `x`, `y` ja `z`, ja palauttaa osoittimen luotuun tietorakenteeseen. Kuvan keskipisteeksi asetetaan piste, jonka koordinaatit ovat `(x-1)/2`, `(y-1)/2` ja `(z-1)/2`.

Funktio `copy_image_uchar` luo kuvatietorakenteen, jonka sisällöksi kopioidaan kuvan `in` sisältö, ja palauttaa osoittimen luotuun kopioon.

Funktio `delete_image_uchar` vapauttaa kuvatietorakenteen `in` käyttämän muistin.

Funktiot `create_image_real`, `copy_image_real`, `delete_image_real`, `create_-image_complex`, `copy_image_complex` ja `delete_image_complex` toimivat muuten kuten yllä kuvatut vastaavat funktiot, mutta käsittelevät kuvan dataa muodoissa `real` ja `complex`, vastaavasti.

5.2.4.3.2 Tyypittämättömien kuvatietorakenteiden muistinhallinta

```
int image_dims(image *in);

size_t image_pix_size(image *in);
```

```
size_t image_data_size(image *in);

image* image_create(int type, int x,int y,int z);

image* image_copy(image *in);

void image_delete(image *in);
```

Funktio `image_dims` palauttaa kuvan `in` dimension.

Funktio `image_pix_size` palauttaa kuvan `in` intensiteettien tallettamiseen käytetyn tietotyypin koon.

Funktio `image_data_size` palauttaa kuvan `in` `data`-taulukon koon.

Funktio `image_create` luo tyyppiä `type` olevan kuvatietorakenteen, jonka dimensiot ovat `x`, `y` ja `z`.

Funktio `image_copy` luo kopion kuvasta `in` ja palauttaa osoittimen luomaansa tietorakenteeseen.

Funktio `image_delete` vapauttaa kuvan `in` varaaman muistin.

5.2.4.3.3 Muunnokset kuvatietorakennetyyppien välillä

```
image_uchar *image_to_image_uchar(image* in);

image_real *image_to_image_real(image* in);

image_complex *image_to_image_complex(image* in);

image_real *image_uchar_to_image_real(image_uchar *in, image_real *out);

image_uchar *image_real_to_image_uchar(image_real *in, image_uchar *out);

image_real *image_complex_to_image_real(image_complex *in,
                                        image_real *out);
```

Funktiot `image_to_image_uchar`, `image_to_image_real` ja `image_to_image_uchar` muuntavat `image`-kuvatietorakenteita `image_uchar`, `image_real` ja `image_complex` muotoisiksi kuvatietorakenteiksi. Jos muunnettava kuva `in` on `type`-kenttänsä perusteella väärää tyyppiä, aiheutetaan ohjelman suorituksen keskeyttäminen `bk_error`-makrolla. Alkuperäisen kuvan muistialue ei ole käytettävissä muunnoksen jälkeen.

Funktio `image_uchar_to_image_real` kopioi tyypiä `uchar` olevan kuvan reaalikuvaksi varaten tarvittaessa muistin tulokselle.

Funktio `image_complex_to_image_real` kopioi kompleksikuvan reaaliosan reaalikuvaksi varaten tarvittaessa muistin tulokselle.

Funktio `image_real_to_image_complex` kopioi reaalikuvan kompleksi kuvan reaaliosaksi varaten tarvittaessa muistin tulokselle ja nollaten tuloskuvan imaginääriosan.

5.2.4.3.4 Kuvan koon muuttaminen

```
void resize_image_uchar(image_uchar *in, int x, int y, int z);
```

```
void resize_image_real(image_real *in, int x, int y, int z);
```

```
void resize_image_complex(image_complex *in, int x, int y, int z);
```

Funktiot muuttavat kuvien dimensiot parametreina annettujen mukaisiksi.

5.2.4.3.5 Kierrot

Kuvien kierrosta toteutetaan tehokkuussyistä lukuisia eri versioita:

```
point rotate_point(point p, orientation o);
```

```
image_real *rotate_real(image_real *in, image_real *out,
                                  orientation o);
```

```
image_real *rotate_around_X_real(image_real *in, image_real *out,
                                  double alpha);
```

```
image_real *rotate_around_Y_real(image_real *in, image_real *out,
                                 double alpha);

image_real *rotate_around_Z_real(image_real *in, image_real *out,
                                 double alpha);
```

Funktio `rotate_point` kiertää pisteen `p` origon ympäri parametrin `o` ilmoittamaan asentoon.

Funktio `rotate_real` kiertää kuvan `in` keskipisteensä ympäri parametrin `o` ilmoittamaan asentoon. Jos parametri `out` on `NULL`, funktio luo uuden kuvatietorakenteen kierron tuloksen tallettamista varten ja palauttaa osoittimen siihen. Muuten funktio käyttää tuloksen tallettamiseen parametrin `out` osoittamaa kuvatietorakennetta ja palauttaa osoittimen `out` arvon. Tietorakenteen `out` dimensioita kasvatetaan tarvittaessa.

Funktiot `rotate_around_X_real`, `rotate_around_Y_real` ja `rotate_around_Z_real` kiertävät kuvaa `in` keskipisteensä kautta kulkevan X-, Y- tai Z-akselin suuntaisen akselin ympäri kulman `alpha` ilmoittaman määrän. Palautusarvo ja parametrin `out` tulkinta on vastaava kuin funktion `rotate_real` tapauksessa.

```
image_complex *rotate_complex(image_complex *in, image_complex *out,
                              orientation o);

image_complex *rotate_around_X_complex(image_complex *in,
                                       image_complex *out,
                                       double alpha);

image_complex *rotate_around_Y_complex(image_complex *in,
                                       image_complex *out,
                                       double alpha);

image_complex *rotate_around_Z_complex(image_complex *in,
                                       image_complex *out,
                                       double alpha);
```

Kompleksiset kierrot toimivat kuten reaaliset vastineensakin ollen kuitenkin kompleksisia.

5.2.4.3.6 Peilaukset

```
void mirror_X_real(image_real *in);
```

```
void mirror_Y_real(image_real *in);
```

```
void mirror_Z_real(image_real *in);
```

Funktio `mirror_X_real` peilaa kuvan `in` X-akselin, `mirror_Y_real` Y-akselin ja `mirror_Z_real` Z-akselin.

5.2.4.3.7 Polaarimuunnokset

Polaarimuunnoksesta ja sen käänteismuunnoksesta toteutetaan ainoastaan kaksiulotteiset versiot:

```
image_real *cartesian_to_polar_2d_real(image_real *in, image_real *out,
                                        int r, int steps);
```

```
image_real *polar_to_cartesian_2d_real(image_real *in, image_real *out);
```

Funktio `cartesian_to_polar_2d_real` muuntaa parametrin `in` osoittaman karteesisen kuvan polaarimuotoiseksi käyttäen muunnoksen napana kuvan `in` keskipistettä. Parametrilla `r` ilmoitetaan, kuinka etäälle kuvan keskipisteestä muunnos ulotetaan. Parametri `steps` ilmoittaa, kuinka moneen tasaväliseen osaan kulmaparametrin arvoalue $[0, 2\pi[$ muunnoksessa jaetaan.

Parametrin `out` semantiikka on seuraava: Jos `out` on `NULL`, funktio luo tuloksen talletusta varten uuden kuvatietorakenteen, jonka leveys on `r` ja korkeus `steps`. Muuten funktio käyttää tuloksen palauttamiseen `out`:in osoittamaa kuvatietorakennetta, jonka dimensioita muutetaan tarvittaessa. Funktio palauttaa osoittimen tuloskuvaan.

Funktio `polar_to_cartesian` on polaarimuunnoksen käänteismuunnos. Jos osoittimen `out` arvo on `NULL`, funktio luo tulokselle uuden kuvatietorakenteen, jonka leveys ja korkeus ovat kaksi kertaa kuvan `in` leveys. Muuten tuloksen tallettamiseen käytetään kuvatietorakennetta, johon `out` osoittaa (dimensioita muutetaan tarvittaessa).

Molemmat muunnokset käyttävät interpoloinnissa oletusinterpolointia.

Rajoituksia: Funktio `cartesian_to_polar_2d_real` jättää määrittelemättä palauttamansa kuvan keskipisteen.

5.2.4.3.8 Transpoosi

Transpoosioperaatioita tarvitaan ainakin vertailussa:

```
void transpose_2d_real(image_real *in);
```

Funktio `transpose_2d_real` transponoi parametrin `in` osoittaman kuvan. Kuvan ajatellaan koostuvan pinosta kaksiulotteisia XY-tason suuntaisia matriiseja, joista kukin transponoidaan erikseen pinorakenteeseen koskematta. Tuloksena saatu transponoitu kuva korvaa syötekuvan, joka tuhoutuu operaation yhteydessä.

5.2.4.3.9 Karkeistus

Tehokkuussyistä karkeistuksesta toteutetaan erilliset versiot kaksi- ja kolmiulotteisten kuvien käsittelyä varten:

```
image_real *coarse_real(image_real *in, image_real *out,
                        int coarse_factor);

image_real *coarse_2d_real(image_real *in, image_real *out,
                           int coarse_factor);
```

Funktio `coarse_real` karkeistaa parametrin `in` osoittaman kuvan skaalaamalla sen kertoimella `coarse_factor`$^{-1}$. Jos parametrin `out` arvo on `NULL`, funktio luo uuden kuvatietorakenteen tuloksen tallettamista varten. Muuten funktio käyttää tuloksen tallettamiseen parametrin `out` osoittamaa kuvatietorakennetta.

Funktio `coarse_2d_real` toimii muuten samoin kuin edelläkuvattu funktio `coarse_real`, mutta se skaalaa kuvan ainoastaan leveys- ja korkeussuunnissa, ei syvyyssuunnassa.

```
image_complex *coarse_complex(image_complex *in, image_complex *out,
                              int coarse_factor);
```

```
image_complex *coarse_2d_complex(image_complex *in, image_complex *out,
                                 int coarse_factor);
```

Kompleksiset karkeistukset toimivat kuten reaaliset vastineensakin ollen kuitenkin kompleksisia.

5.2.4.3.10 Neljännesten vaihto

```
void swap_2d_real(image_real *in);
```

```
void deswap_2d_real(image_real *in);
```

```
void swap_3d_real(image_real *in);
```

```
void deswap_3d_real(image_real *in);
```

```
void swap_2d_complex(image_complex *in);
```

```
void deswap_2d_complex(image_complex *in);
```

```
void swap_3d_complex(image_complex *in);
```

```
void deswap_3d_complex(image_complex *in);
```

Funktio `swap_2d_real` vaihtaa kaksiulotteisen kuvan neljännekset keskenään seuraavasti: $\begin{array}{cc} \lceil n/2 \rceil \times \lceil m/2 \rceil & \lfloor n/2 \rfloor \times \lceil m/2 \rceil \\ \lceil n/2 \rceil \times \lfloor m/2 \rfloor & \lfloor n/2 \rfloor \times \lfloor m/2 \rfloor \end{array} \Rightarrow \begin{array}{cc} \lfloor n/2 \rfloor \times \lfloor m/2 \rfloor & \lceil n/2 \rceil \times \lfloor m/2 \rfloor \\ \lfloor n/2 \rfloor \times \lceil m/2 \rceil & \lceil n/2 \rceil \times \lceil m/2 \rceil \end{array}$
Funktio `deswap_2d_real` toimii päinvaistaisesti.

Funktio `swap_3d_real` vaihtaa ensin kaikki XY-tasonsa funktion funktion `swap_2d_real` avulla ja tämän jälkeen Z-akselin puolikkaat: $\lceil l/2 \rceil \quad \lfloor l/2 \rfloor \Rightarrow \lfloor l/2 \rfloor \quad \lceil l/2 \rceil$
Funktio `deswap_3d_real` toimii päinvaistaisesti.

Kompleksiset vaihdot toimivat kuten reaaliset vastineensakin ollen kuitenkin kompleksisia.

5.2.4.3.11 Veden poisto

```
void remove_water_image_real(image_real *image, double r1, double r2);
```

Funktio laskee ensin niiden kuvan `image` pikseleiden keskiarvon `a` joiden (kolmiulotteinen euklidinen) etäisyys kuvan keskipisteestä on suurempi kuin `r1` ja korkeintaan `r2`. Sitten funktio nollaa ne pikselit, joiden etäisyys kuvan keskipisteestä on suurempi kuin `r1`. Niistä pikseleistä, joiden etäisyys kuvan keskipisteestä on korkeintaan `r1`, vähennetään `a`.

5.2.4.3.12 Intensiteetin venytys

```
void scale_real(image_real *in, real imin, real imax, real omin, real omax);
```

Funktio laskee kuvan intensiteetille kuvauksen $[\mathtt{imin}, \mathtt{imax}] \to [\mathtt{omin}, \mathtt{omax}]$.

5.2.4.3.13 Intensiteetin normalisointi

```
void normalize(image_real *image, double m);
```

Funktio laskee kuvan `image` pikseleiden summan s. Tämän jälkeen kuvan pikselit p normalisoidaan operaatiolla $p := (\mathtt{m}/s) * p$.

Rajoituksia Jos kuvan pikseleiden summa `s` on tasan nolla, jättää funktio kuvan ennalleen.

5.2.4.3.14 Keskipisteen määrittäminen

```
point find_centroid_real(image_real *image);
```

Funktio `find_centroid_real` etsii kolmiulotteisen reaalikuvan `image` painopisteen. Tässä yhteydessä kuvaa ajatellaan kolmiulotteisena hilana, jonka pisteiden massat vastaavat niiden intensiteettejä [34].

5.2.4.3.15 Tyhjien reuna-alueiden poisto

```
image_real *crop_image_real(image_real *in, real epsilon);
```

Funktio pienentää kuvan kokoa poistamalla kuvan reunoilta suorakaiteen muotoiset lähes tyhjät alueet. Tarkemmin sanoen funktio poistaa kuvan reunoiltaan maksimaalisen suuret suorakaiteen muotoiset alueet, joiden minkään pikselin intensiteetti ei poikkea enempää kuin `epsilon`:in nollasta.

5.2.4.3.16 Fourier-muunnokset

```
image_complex *fft_2d_complex(image_complex *in,
                              image_complex *out,
                              fftwnd_plan fft);

image_complex *fft_3d_complex(image_complex *in,
                              image_complex *out,
                              fftwnd_plan fft);

image_complex *ifft_2d_complex(image_complex *in,
                               image_complex *out,
                               fftwnd_plan ifft);

image_complex *ifft_3d_complex(image_complex *in,
                               image_complex *out,
                               fftwnd_plan ifft);
```

Funktiot laskevat Fourier-muunnokset ja sen käänteismuunnokset kaksi- ja kolmiulotteisille kuville.

5.2.4.3.17 Minimien ja maksimien etsintä

```
int *find_minimums_real(image_real *in, int n, int *out);

int *find_maximums_real(image_real *in, int n, int *out);
```

Funktio `find_minimums_real`

5.2.4.3.18 Korrelaatio ja konvoluutio

Korrelaatiot ja konvoluutiot on kuvattu mm. teoksessa [17].

```
image_complex *correlate_complex(image_complex *in1,
                                 image_complex *in2,
                                 image_complex *out);

image_complex *convolve_complex(image_complex *in1,
                                image_complex *in2,
                                image_complex *out);

image_real *autocorrelate_translational_real(image_real *in,
                                             image_real *out,
                                             fftwnd_plan fft,
                                             fftwnd_plan ifft);

image_real *autocorrelate_rotational_real(image_real *in,
                                          image_real *out,
                                          int r, int steps,
                                          fftw_plan fft,
                                          fftw_plan ifft);

image_real *doubleautocorrelate_real(image_real *in,
                                     image_real *out,
                                     int r,
                                     int steps,
                                     fftwnd_plan fft_2d,
                                     fftwnd_plan ifft_2d,
                                     fftw_plan fft_1d,
                                     fftw_plan ifft_1d);

image_real *variance_real(image_real *in, image_real *out, int r);
```

Funktio `correlate_complex` laskee kahden kuvan Fourier-muunnosten korrelaation.

Funktio `convolve_complex` laskee kahden kuvan Fourier-muunnosten konvoluution.

Funktio `autocorrelate_translational_real` laskee kuvan siirtymällisen autokorrelaation.

Funktio `autocorrelate_rotational_real` laskee kuvan kierrollisen autokorrelaation.

Funktio `doubleautocorrelate_real` laskee kuvan kaksinkertaisen autokorrelaation.

Funktio `variance_real` laskee kuvan varianssikuvan halutun säteisellä ikkunalla.

5.2.4.3.19 Interpolointi

Interpolointifunktioista toteutetaan sekä reaaliset että kompleksiset versiot.

Reaaliversiot:

```
real interpolate_nn_real(image_real *in, point p);

real interpolate_linear_real(image_real *in, point p);

real interpolate_nn_around_X_real(image_real *in,
                            double y, double z, int i);

real interpolate_nn_around_Y_real(image_real *in,
                            double z, double x, int j);

real interpolate_nn_around_Z_real(image_real *in,
                            double x, double y, int k);

real interpolate_linear_around_X_real(image_real *in,
                                double y, double z, int i);

real interpolate_linear_around_Y_real(image_real *in,
                                double z, double x, int j);

real interpolate_linear_around_Z_real(image_real *in,
                                double x, double y, int k);
```

```
real interpolate_sinc_around_X_real(image_real *in,
                              double y, double z, int i);

real interpolate_sinc_around_Y_real(image_real *in,
                              double z, double x, int j);

real interpolate_sinc_around_Z_real(image_real *in,
                              double x, double y, int k);

real interpolate_sinc_trunc_real(image_real *in, point p);

real interpolate_sinc_truncated_real(image_real *in,
                                     point p, int radius);
```

Imaginääriversiot:

```
complex interpolate_nn_complex(image_complex *in, point p);

complex interpolate_linear_complex(image_complex *in, point p);

complex interpolate_nn_around_X_complex(image_complex *in,
                                  double y, double z, int i);

complex interpolate_nn_around_Y_complex(image_complex *in,
                                  double z, double x, int j);

complex interpolate_nn_around_Z_complex(image_complex *in,
                                  double x, double y, int k);

complex interpolate_linear_around_X_complex(image_complex *in,
                                      double y, double z, int i);

complex interpolate_linear_around_Y_complex(image_complex *in,
                                      double z, double x, int j);

complex interpolate_linear_around_Z_complex(image_complex *in,
```

```
                                    double x, double y, int k);

complex interpolate_sinc_around_X_complex(image_complex *in,
                                    double y, double z, int i);

complex interpolate_sinc_around_Y_complex(image_complex *in,
                                    double z, double x, int j);

complex interpolate_sinc_around_Z_complex(image_complex *in,
                                    double x, double y, int k);

complex interpolate_sinc_trunc_complex(image_complex *in, point p);
complex interpolate_sinc_truncated_complex(image_complex *in,
                                           point p, int radius);
```

Interpolointifunktioiden reaali- ja kompleksiversiot toimivat analogisesti, joten seuraavassa rajoitutaan tarkastelemaan ainoastaan reaalista tapausta. Interpolointialgoritmin toimintaa on käsitelty tarkemmin luvussa 7.1.

Funktio `interpolate_nn_real` palauttaa sen kuvan `in` pisteen intensiteetin, joka on pisteen `p` lähin naapuri. Jos pisteen `p` lähin naapuri on kuvan ulkopuolella, palautetaan `0`.

Funktio `interpolate_linear_real` palauttaa lineaarisen interpoloinnin mukaisen approksimaation pisteen `p` intensiteetille. Funktio `interpolate_sinc_real` puolestaan käyttää interpolointiin sinc-interpolointimenetelmää ja `interpolate_sinc_truncated_real` (kutsurajapintanaan `interpolate_sinc_trunc_real`) katkaisua sinc-interpolointia.

Edellä mainitut funktiot interpoloivat kolmiulotteisesti. Projisoinnissa tarvitaan myös kaksiulotteista interpolointia, jota varten kustakin interpolointifunktiosta `interpolate_I_real` on myös versiot `interpolate_I_around_X_real`, `interpolate_I_around_Y_real` ja `interpolate_I_around_Z_real`. Nämä funktiot interpoloivat kaksiulotteisesti interpolointimenetelmää `I` käyttäen tasojen YZ, ZX ja XY suuntaisissa pisteen `p` sisältävissä tasoissa.

Kompleksi-interpolointifunktiot toimivat samoin kuin reaaliset vastineensa, paitsi että ne huomioivat erikseen kompleksikuvan reaali- ja imaginääriosat.

5.2.4.3.20 Sinogrammeihin liittyvät funktiot Sinogrammit ovat kaksiulotteisia diskreettejä Radon-muunnoksia ja niitä voidaan hyödyntää projisoinnissa ja vertailussa.

```
image_real *sinogram_2d_real(image_real *in,
     image_real *out,
     int steps, double *angles);

void normalize_sinogram(image_real *in, real mass);

image_real *coarse_sinogram_lines(image_real *in,
                                  image_real *out,
                                  int coarse_factor);

image_real *coarse_sinogram_columns(image_real *in,
                                    image_real *out,
                                    int coarse_factor);

image_real *match_sinograms_square_distance_real(image_real *in1,
 image_real *in2,
 image_real *out);

image_real *match_sinograms_correlation_coefficient_real(image_real *in1,
 image_real *in2,
 image_real *out);

image_real *match_rotation_symmetries_real(image_real *in,
   image_real *out);

image_real *match_mirror_symmetries_real(image_real *in,
 image_real *out);
```

Funktio `sinogram_2d_real` laskee kuvasta sinogrammin [39].

Funktio `normalize_sinogram` normittaa jokaisen sinogrammirivin intensiteettien summan parametrin `mass` osoittamaksi.

Funktio coarse_sinogram_lines karkeistaa sinogrammin rivejä ja funktio coarse_sinogram_lines sinogrammin sarakkeita karkeistuskertoimella coarse_factor.

Funktiot match_sinograms_square_distance_real ja match_sinograms_correlation_coefficient_ mittaavat kahden sinogrammin etäisyyttä toisistaan, edellinen neliövirheellä ja jälkimmäinen korrelaatiokertoimella.

Funktiot match_rotation_symmetries_real ja match_mirror_symmetries_real laskevat sinogrammista yksiulotteisen projektion symmetrioiden tarkastelua varten.

5.2.5 Tiedostonkäsittelijä

5.2.5.1 Yleiskuva

Tiedostonkäsittelijä-moduuli sisältää funktioita, joita käytetään kuvien lukemiseen kiintolevyltä ja tallentamiseen kiintolevylle. Lataus ja tallennus rutiinit on toteutettu MRC- ja PGM-formaateille.

5.2.5.2 Tietorakenteet

Moduulissa on esitelty MRC:n otsake tietoja varten tietorakenne MRChead (määrittely C-kielellä). MRChead tietorakenne sisältää MRC-tiedoston lukemiseen tai tallentamiseen liittyvää informaatiota.

```
typedef struct {
  int nx;
  int ny;
  int nz;
  int mode;
  int nxstart;
  int nystart;
  int nzstart;
  int mx;
  int my;
  int mz;
  float xlen;
```

```
  float ylen;
  float zlen;
  float alpha;
  float beta;
  float gamma;
  int mapc;
  int mapr;
  int maps;
  float amin;
  float amax;
  float amean;
  int ispg;
  int nsymbt;
  char extra[116];
  float xOrigin;
  float yOrigin;
  int nlabl;
  char label[10][80];
} MRChead;
```

5.2.5.3 Toiminnot

5.2.5.3.1 MRC-tiedostojen käsittely

```
image_uchar *mrc_load_uchar(image_uchar *out, char *filename);
```

```
image_real *mrc_load_real(image_real *out, char *filename);
```

```
image_complex *mrc_load_complex(image_complex *out, char *filename);
```

Funktiot `mrc_load_uchar` ja `mrc_load_real` lataavat reaalimuotoisen MRC-kuvan ja funktio `mrc_load_complex` kompleksimuotoisen MRC-kuvan parametrin `filename` osoittamasta tiedostosta tulostaulukkoon `out`, jolle varataan muisti tarvittaessa.

```
int mrc_save_uchar(image_uchar *in, char *filename);
```

```
int mrc_save_real(image_real *in, char *filename);

int mrc_save_complex(image_complex *in, char *filename);

int mrc_save_components_complex(image_complex *in,
                                char *filename_magnitude,
                                char *filename_phase);
```

Funktiot `mrc_save_uchar` ja `mrc_save_real` tallentavat reaalimuotoisen MRC-kuvan ja funktio `mrc_save_complex` kompleksimuotoisen kuvan parametrin `filename` osoittamaan MRC-kuvatiedostoon. Funktio `mrc_save_components_complex` tallettaa kompleksikuvan amplitudin ja vaiheen parametrien `filename_magnitude` ja `filename_phase` osoittamiin MRC-kuvatiedostoihin.

5.2.5.3.2 PGM-tiedostojen käsittely

```
image_uchar *pgm_load_uchar(image_uchar *out, char *filename);

image_real *pgm_load_real(image_real *out, char *filename);

image_complex *pgm_load_complex(image_complex *out,
                                char *filename_magnitude,
                                char *filename_phase);
```

Funktiot `pgm_load_uchar` ja `pgm_load_real` lataavat reaalimuotoisen PGM-kuvan ja funktio `pgm_load_complex` kompleksimuotoisen PGM-kuvan parametrin `filename` osoittamasta tiedostosta tulostaulukkoon `out`, jolle varataan muisti tarvittaessa.

```
int pgm_save_uchar(image_uchar *in, char *filename);

int pgm_save_real(image_real *in, char *filename);

int pgm_save_complex(image_complex *in,
                     char *filename_magnitude,
                     char *filename_phase);
```

Funktiot `pgm_save_uchar` ja `pgm_save_real` tallentavat reaalimuotoisen kuvan parametrin `filename` osoittamaan PGM-kuvatiedostoon. Funktio `pgm_save_complex` tallettaa kuvan amplitudin ja vaiheen parametrien `filename_magnitude` ja `filename_phase` osoittamiin PGM-kuvatiedostoihin.

5.2.5.3.3 Muiden tiedostojen käsittely

```
int gnuplot_save_1d_real(image_real *in, char *filename);
```

Funktio `gnuplot_save_1d_real` tallettaa yksiulotteisen reaalilukukuvan `gnuplot`:in [40] ymmärtämään muotoon parametrin `filename` osoittamaan tiedostoon.

5.2.6 Konfiguraatiomoduuli

5.2.6.1 Yleiskuva

Konfiguraatiomoduuli koostuu joukosta ohjelman toimintaa ohjaavia globaaleja muuttujia ja C-funktioista, joilla niiden arvoja voi muuttaa. Varsinainen konfiguraatiotiedosto on Perl-kielinen ohjelma, josta voidaan kutsua edellämainittuja C-kielisiä funktioita ja asettaa ohjelman globaaleille muuttujille halutut arvot. Tässä luvussa käsiteltävä Konfiguraatiomoduuli muodostuu konfiguraation käsittelyn C-kielisestä osuudesta; Perl-kielellä toteutettu käyttäjärajapinta on kuvattu luvussa 9.

5.2.6.2 Tietorakenteet

Konfiguraatiomoduulissa on seuraavat ohjelman toiminnan ohjaamiseen tarkoitetut globaalit, Perliin näkyvät, määrittelyt ja muuttujat (listassa näkyvät myös oletusarvot):

```
#define MAX_PROCESS_NUM 30

int projection_buffer_size=16;
int max_projection_angle_group_size=128;
int max_best_results=10;
int state_saving_interval=60*60/4;
```

```
char *projection_angles_file_name=NULL;
char *completed_projection_angles_file_name=NULL;
char *projection_angle_groups_file_name=NULL;
char *results_file_name=NULL;
char *log_file_name=NULL;
char **pizza_file_names = NULL;
char *model_file_name=NULL;
int projection_image_type=IMAGE_REAL;
int projection_image_x=256;
int max_results_from_comparison=16;
int remove_water_r1=100;
int remove_water_r2=110;
int comparison_steps_2d=100;
```

Seuraavassa lyhyet kuvaukset konfiguraatiomoduulin globaaleista muuttujista (ja määrittelystä):

MAX_PROCESS_NUM määrittelee eräiden konfiguraatiomoduulin prosessikohtaista informaatiota sisältävien taulukoiden maksimikoon.

projection_buffer_size on kokonaisluku, joka ilmoittaa projektionvälittäjän projektiopuskurin koon.

max_projection_angle_group_size on kokonaisluku, jota käytetään projektioryhmien oletuskokona mikäli projektioryhmätiedostoa ei ole annettu.

max_best_results on kokonaisluku, joka ilmoittaa kuinka monta parasta tulosta elektronimikroskooppikuvaa kohden talletetaan tulostiedostoon.

state_saving_interval on kokonaisluku, joka ilmoittaa kuinka monen sekunnin välein muistissa olevat laskennan tulokset talletetaan kovalevylle.

projection_angles_file_name on merkkijono, joka kertoo käytettävän projektiokulmatiedoston nimen.

completed_projection_angles_file_name on merkkijono, joka ilmoittaa käytettävän käsiteltyjen orientaatioiden tunnisteet sisältävän tiedoston nimen. Tiedoston nimen lopun tulee olla muotoa `.DDDDD.bin`, missä kukin `D` on joku numeraali väliltä `0-9`.

projection_angle_groups_file_name on merkkijono, joka ilmoittaa, mistä tiedostosta projektiokulmatiedoston ryhmäjako luetaan. Jos `Xprojection_angle_groups_file`== X, blokkien oletetaan koostuvan peräkkäisistä `max_projection_angle_group_size`-kokoisista yhtenäisistä segmenteistä.

results_file_name on merkkijono, joka ilmoittaa käytettävän tulostiedoston nimen. Tiedoston nimen lopun tulee olla muotoa `.DDDDD.bin`, missä kukin `D` on joku numeraali väliltä `0-9`.

log_file_name on merkkijono, jolla ilmoitetaan lokitiedoston nimi.

pizza_file_names on merkkijonotaulukko, joka sisältää pizzojen eli makromolekyylikomplekseista otettujen elektronimikroskooppikuvien tiedostojen nimet. Ohjelman sisällä pizzoihin viitataan tämän taulukon indeksien mukaisilla kokonaisluvuilla.

model_file_name on merkkijono, jolla ilmoitetaan minkänimisestä tiedostosta tarkasteltava kolmiulotteinen malli ladataan.

projection_angle_groups_file on merkkijono joka ilmoittaa, mistä tiedostosta orientaatioiden blokkijako luetaan. Jos `Xprojection_angle_groups_file`== X, blokkien oletetaan koostuvan peräkkäisistä `max_angle_group_size`-kokoisista yhtenäisistä segmenteistä.

pizza_file_names[] sisältää pizzojen eli makromolekyylikomplekseista otettujen elektronimikroskooppikuvien tiedostojen nimet. Ohjelman sisällä pizzoihin viitataan tämän taulukon indeksien mukaisilla kokonaisluvuilla.

projection_buffer_size on kokonaisluku joka ilmoittaa projektionvälittäjän käyttämän puskurin koon.

projection_image_type ilmoittaa projektiokuvien tyypin.

projection_image_x on kokonaisluku, joka ilmoittaa projektioiden x- ja y-dimensiot.

max_results_from_comparison on kokonaisluku jolla ilmoitetaan kuinka monta tulosta vertailija voi kerrallaan lähettää tulosten kerääjälle.

remove_water_r1 ja `remove_water_r2` ovat kokonaislukuja, jotka ilmoittavat vedenpoistofiltterin sisemmän ja ulomman säteen (katso 5.2.4.3.11).

comparison_steps_2d ilmoittaa, kuinka moniaskelista tasavälistä asteikkoa rotaatioinvariantit vertailualgoritmit käyttävät.

Edellisten Perliin näkyvien muuttujien lisäksi moduulissa on myös seuraava vain C-kieliseen Beklun osaan näkyvä globaali muuttuja:

```
int process_num;
```

Tämä muuttuja sisältää luotujen prosessien lukumäärän.

5.2.6.3 Toiminnot

Konfiguraatiomoduulin funktiot jakautuvat karkeasti kahteen osaan, ohjelman sisällä käytettäviin funktioihin ja Perl-rajapintaa varten tarvittaviin funktioihin.

5.2.6.3.1 C-kielisen ohjelman sisäiset funktiot

```
int *get_proc_numbers(int procType, int *count);

void write_mpich_machine_file(char *filename, char *full_path_to_binary);

int get_proc_type(int proc_num);
int get_pizza_count(int proc_num);
int get_pizza_count_comparator(int comp_num);

void perl_load(char *code);
void perl_call_func(char *subname);
void perl_clean(void);

void reorder_processes(void);
char* get_proc_machine(int proc_num);

void print_process_table(void);
```

Funktio `get_proc_numbers` palauttaa kokonaislukutaulukon, joka sisältää kaikkien tyyppiä `procType` olevien prosessien numerot. Parametri `count` on tulosparametri, jonka avulla välitetään kutsujalle palautettavan tulostaulukon koko. Parametrin `procType` arvon pitää olla jokin luetellun tyypin `enum_process_types` arvoista.

Funktio `write_mpich_machine_file` kirjoittaa prosessien oikeilla koneilla käynnistämiseen tarvittavan `mpich`:n konfiguraatiotiedoston, jonka nimeksi tulee `filename`. Käynnistettävän ohjelman sijainti ja nimi ilmoitetaan parametrin `full_path_to_binary` avulla. Tarkempi kuvaus tuotettavan tiedoston rakenteesta löytyy `mpich`:n dokumentaation yhteydestä [4].

Funktio `get_proc_type` palauttaa prosessin `proc_num` tyypin, joka on jokin luetellun tyypin `enum_process_types` arvoista.

Funktio `get_pizza_count` palauttaa prosessille `proc_num` funktiolla `set_pizza_count` asetetun elektronimikroskooppikuvien määrän.

Funktio `get_pizza_count_comparator` toimii kuten edellä kuvattu funktio `get_pizza_count` paitsi että vertailija, jonka elektronimikroskooppikuvien lukumäärä halutaan tietää, identifioidaan sen vertailija-tiedostonlataaja -kommunikaattorin sisäisen asteen mukaisella tunnisteella `comp_num`.

Funktio `perl_load` suorittaa merkkijonossa `code` olevan Perl-ohjelman.

Funktio `perl_call_func` kutsuu Perl-funktiota jonka nimi on `subname`.

Funktiota `perl_clean` kutsutaan, kun integroitua Perl-tulkkia ei enää tarvita. Funktio suorittaa joitakin puhdistusoperaatioita.

Funktio `reorder_processes` järjestää prosessitaulukon prosessinumeroiden mukaiseen järjestykseen.

Funktio `get_proc_machine` palauttaa sen koneen nimen, jolla prosessia numeroltaan `proc_num` ajetaan.

Funktio `print_process_table` tulostaa standarditulostusvuohon `stdout` prosessitaulukon silmää miellyttävässä muodossa.

5.2.6.3.2 Perl-rajapintafunktiot

```
int define_process(char *machine, int process_type);
```

```
void set_pizza_count(int process_number,int pizza_count);
```

Funktio `define_process` määrittelee uuden tyyppiä `process_type` olevan prosessin käynnistettäväksi koneella, jonka DNS-nimi on `machine`. Funktio palauttaa käynnistettävän prosessin prosessinumeron eli prosessin tunnisteen `MPI_COMM_WORLD`:ssä. Funktio ei varsinaisesti käynnistä prosessia, vaan liittää sen myöhemmin käynnistettävien prosessien taulukkoon.

Funktio `set_pizza_count` asettaa prosessille `process_number` lähetettävien pizzojen eli elektronimikroskooppikuvien määräksi parametrin `pizza_count` arvon. Jos prosessi `process_number` ei ole tyypiltään `PROC_COMPARATOR`, aiheutetaan järjestelmän suorituksen keskeytys makrolla `bk_error`. Elektronimikroskooppikuvien määrät eivät ole absoluuttisia vaan määräävät ainoastaan kuvien määrien suhteet.

5.2.7 Hajautus

5.2.7.1 Yleiskuva

Hajautus-moduulissa on esitelty järjestelmän hajautukseen liittyviä aliohjelmia ja tietorakenteita. Aliohjelmat kuvaavat järjestelmän viestienvälityksen rajapinnan ja tietorakenteet kuvaavat prosessien välisessä kommunikoinnisssa käytettyjen viestien rakenteen.

5.2.7.2 Tietorakenteet

5.2.7.2.1 Luetellut tyypit

```
typedef enum {
  PROC_FILE_LOADER=0,
  PROC_PROJECTION_RELAY,
  PROC_RESULT_COLLECTOR,
  PROC_PROJECTION_PRODUCER,
  PROC_COMPARATOR,
} enum_process_types;

typedef enum {
```

```
  MSG_PROJECTION_REQS=0,
  MSG_MODEL,
  MSG_PIZZA,
  MSG_PROJECTION,
  MSG_RESULT,
  MSG_CMD
} enum_message_types;

typedef enum {
  EVT_PROJECTION_BEGIN=0,
  EVT_PROJECTION_END,
  EVT_COMPARISON_BEGIN,
  EVT_COMPARISON_END
} enum_log_event_types;
```

Yksittäisen prosessin tyyppi määritellään `enum_process_types` tyypillä. Järjestelmän prosessit ovat joko alustustiedoston käsittelijöitä `PROC_FILE_LOADER`, projektioden välittäjiä `PROC_PROJECTION_RELAY`, projektioden tuottajia `PROC_PROJECTION_PRODUCER`, vertailijoita `PROC_COMPARATOR` tai tulostiedon keräilijöitä `PROC_RESULT_COLLECTOR`.

Lueteltu tyyppi `enum_message_types` määrittelee lähetettävän tai vastaanotettavan viestin tyypin. Järjestelmässä on toteutettu viestirakenteet makromelkyylin mallille `MSG_MODEL`, elektronimikroskooppikuvalle `MSG_PIZZA`, peojektiolle `MSG_PROJECTION`, projektion pyytämiselle välittäjältä tai tuottaja prosessilta `MSG_PROJECTION_REQS`, tulostiedon välittämiselle `MSG_RESULT`, sekä komentoviesteille `MSG_CMD`.

Järjestelmä ylläpitää käynnissä olevista toiminnoistaan kirjaa lokitiedostoon, lokiviestien tyypit on lueteltu `enum_log_event_types` tyypissä. Lokiin kirjataan ylös uusien projektioden laskennan sekä vertailuoperaatioden aloitus ja lopetus.

5.2.7.2.2 Varsinaiset tietorakenteet

```
typedef struct {
  image *img;
} model;

typedef struct {
```

```
  int id;
  orientation orient;
  image *img;
} projection;

typedef struct {
  int id;
  image *img;
} pizza;

typedef struct {
  int id;
  orientation orient;
} projection_req;

typedef struct {
  int pizzaId;
  int orientId;
  orientation orient;
  double goodness[MAX_GOODNESS_VALS];
} result;
```

Hajautus-moduulissa määriteltyt tietorakenteet kuvaavat järjestelmän prosessien väliset viestit. Tietorakenteet on määritelty kolmiulotteiselle mallille (`model`), mallista projisoidulle projektiolle (`projection`), sekä elektronimikroskooppikuvalle (`pizza`). Malli viestirakenne sisältää `image` tietorakenteen ja projektio ja elektronimikroskooppikuva viestirakenteet sisältävät `image` tietorakenteen lisäksi tunnistetiedot ja projektio vielä orientaatiotunnisteen.

Lisäksi tarvitaan viestirakenne `projection_req`, jolla tiedostojenlataajaprosessi ilmoittaa projektiontuottajaprosesseille mistä kulmista niiden oletetaan tuottavan projektioita mallista. Tiedostojenlataaja lähettää näitä mahdollisesti useita yhdessä viestissä projektioidentuottajaprosessille ei kuitenkaan enempää kuin globaalin muuttujan `max_angle_group_size` ilmoittaman määrän.

Vertailujen tulokset välitetään tuloksien kerääjälle `result` tietorakenteella, joka sisältää mikroskooppikuvan tunnuksen, projektion tunnuksen, projektion orientaarion sekä hyvyys arvoja.

5.2.7.3 Toiminnot

Hajautuksen tarjoamat toiminnot voidaan jakaa viestin tyypin mukaisesti. Viestit ovat elektronimikroskooppikuvia, kolmiulotteisia malleja, mallista laskettuja projektiota, kuvia, laskettavia kulmia välitetään projektioden tuottajaprosesseille tai vertailuoperaatioiden tuloksia. Lisäksi lokiviestien välitystä varten on myöskin omat toimintonsa, ja MPI-ryhmätunnusta tarvitaan edellmämainittujen hajautustoimintojen toteutuksessa.

Kaikki lähettävät funktiot saavat parametreinään varsinaisen lähetettävän tiedon lisäksi MPI:n kommunikointiryhmän (`MPI_Comm comm`), vastaanottajan kommunikointiryhmän sisäisen tunnisteen (`int dest`) ja MPI-viestinhallinnan mahdollistava viestin tunnistetieto (`MPI_Request *req`). Poikkeiksen tekee `send_pizza` ja `send_model`, jotka eivät tarvitse viestin tunnistetietoa, koska ne lähettävät estävästi. `MPI_Comm comm` määrittää kommunikaatioryhmän MPI:n hallitsemille prosesseille, jotta prosessit pystyisivät kommunikoimaan keskenään tulee niiden kuulua samaan kommunikaatioryhmään. Vastaanottajan prosessitunnus `int dest` on vastaanottajan kommunikaatioryhmän sisäinen tunnus. `MPI_Request *reg` on viestin kahva, joka mahdollistaa viestilähetyksen kontrolloimisen (peruutus / kulun seuranta). Vastaanottavat funktiot ottavat vastaavat parametrit, ainoastaan `int source` kuvaa vastaanottavan sijasta lähettäjän kommunikaatioryhmän sisäisen tunnuksen ja funktiot palauttavat osoittimen viestin mukana tulleeseen tietorakenteeseen.

5.2.7.3.1 Elektronimikroskooppikuvan välitys

```
void send_pizza(char *fileName,int dest,MPI_Comm comm);
```

```
pizza *recv_pizza(int source,MPI_Comm comm,MPI_Request *req);
```

Funktio `send_pizza` lähettää `fileName` merkkijonon osoittaman elektronimikroskooppikuvan `dest` prosessille, `comm` on MPI-kommunikointiryhmä johon lähettäjä ja vastaanottaja kuuluvat. Elektronimikroskooppikuvia pystyy vastaanottamaan `recv_pizza` funktiolla, jolle annetaan parametreinä keneltä kuva saadaan `source`, MPI-kommunikointiryhmä ja viestinvälitysoperaation tunnistetiedot `req`.

5.2.7.3.2 Projektioden välitys

```
void send_projection(projection *proj,int dest,MPI_Comm comm,
MPI_Request *req);
```

```
projection* recv_projection(int source,MPI_Comm comm,MPI_Request *req);
```

```
void delete_projection(projection *proj);
```

Funktiolla `send_projection` projektioden välittäjä tai tuottaja pystyy lähettämään eteenpäin projektioita. Parametreinä funktio saa projektion `proj`, `dest` osoittaman vastaanottavan prosessin tunnuksen, MPI-kommunikointiryhmän johon projektio lähettetään sekä osoittimen `MPI_Request` viestioperaation tunnisteeseen. Vastaavasti projektiota voidaan vastaanottaa välittäjältä tai tuottajalta `recv_projection` funktiolla, jolle määritellään projektion lähettäjä, MPI-kommunkointiryhmä ja viestivälitysoperaation tunnus.

Funktiota `delete_projection` käytetään `projection`-tietorakenteiden tuhoamiseen. Funktio vapauttaa tietorakenteen varaaman muistin ja suorittaa joitakin jälkiensiivousoperaatioita.

5.2.7.3.3 Projektioiden tuottamiskulmavälien välitys

```
void send_projection_reqs(projection_req *reqs,int *count,int dest,
MPI_Comm comm,MPI_Request *req);
```

```
projection_req *recv_projection_reqs(int *count,int source,
MPI_Comm comm,MPI_Request *req);
```

Funktiolla `send_projection_reqs` tiedostojenlataajaprosessi välittää projisointiprosessille kulmavälin, jolla tuottaa projektioita. Funktio saa parametreinään kulmavälit `reqs`, kulmavälien lukumäärän `count`, vastaanottavan prosessin tunnuksen `dest`, kommunikointiryhmän `comm`, sekä viestinvälitys tunnisteen. Projektiodentuottaja prosessi vastaanottaa kulmavälit funktiolla `recv_projection_reqs`, joka palauttaa osoittimen kulmavälitaulukkoon.

5.2.7.3.4 Mallin välitys

```
void send_model(char *modnm,int dest,MPI_Comm comm);
```

```
model* recv_model(int source, MPI_Comm comm);
```

Tiedostejenlataaja prosessi välittää makromelekyylikompleksin kolmiulotteisen mallin sisältävän tiedostonimen projisointiprosesseille funktiolla `send_model` ja projisointiprosessi vastaanottaa mallin tiedostoimen funktiolla `recv_model`. Funktio `send_model` tarvitsee parametrikseen MPI-kommunikaatioryhmän ja vastaanottajan järjestysnumeron MPI-kommunikaatioryhmässä.

5.2.7.3.5 Vertailutulosten välitys

```
void send_result(result *res,int *count,int dest,MPI_Comm comm,MPI_Request *req);
```

```
result *recv_result(int *count,int source,MPI_Comm comm,MPI_Request *req);
```

Vertailijaprosessi lähettää funktiolla `send_result` tuloksenkeräilijäprosessille vertailuoperaation tulokset. Osoitin `res` viittaa taulukkoon, joka sisältää muuttujan `count` osoittaman määrän `result` tietueita.

Vartailuoperaatioden tuloksia vastaanotetaan funktiolla `recv_result`, joka palauttaa osoittimen vertailijaprosessin lähettäämään tulostaulukkoon ja asettaa parametrin `res` osoittamaan kokonaislukumuuttujaan vastaanotettujen tulosten määrän.

5.2.7.3.6 Kuvien välitys

```
void send_image(image *img,int dest,MPI_Comm comm,MPI_Request *req);
```

```
image *recv_image(int image_data_size,int source,MPI_Comm comm,
MPI_Request *req);
```

Funktiolla `send_image` prosessi pystyy lähettämään kuvan `img` prosessille `dest` MPI-kommunikaatioryhmässä `comm`, lisäksi parametri `req` sisältää viestitunniste tietoa. Prosessi pystyy vastaanottamaan kuvatietorakenteita funktiolla `recv_image`.

5.2.7.3.7 Lokiviestien välitys

```
void log_event(int event,int data,char *string);

char *log_event_to_str(int i);
```

Funktio `log_event` kirjaa viestin keskuslokiin ja funktio `log_event_to_str` konvertoi tapahtuman tunnuksen merkkijonoviestiksi. Projisointiprosessit lähettävät lokiin projektion laskennan aloituksen ja lopetuksen. Vertailijaprosessit viestittävät vastaavasti vertailu operaation aloituksen ja lopetuksen lokiin.

5.2.7.3.8 Konfiguraatiotiedon välitys

```
void bcast_string_array (char ***p_to_arr,int myrank,int root,MPI_Comm comm);
void bcast_process_array (process_descriptor *arr,
                          int count,int root,MPI_Comm comm);

void distribute_configuration(int root,MPI_Comm comm);
typedef char node_str[MAX_DOMAIN_NAME];
node_str *get_nodes(int root,MPI_Comm comm);
void remap_processes(node_str nodes[],int root,int count);
```

Funktio `bcast_string_array` levittää merkkijonotaulukon `*p_to_arr` prosessilta, jonka tunniste on sama kuin `root` kaikille muille kommunikaattorin `comm` prosesseille käyttäen `MPI_Bcast`-funktiota. Funktio `bcast_process_array` toimii muuten samoin, mutta levittää merkkijonotaulukon sijasta `count` `process_descriptor`-tyyppistä alkiota sisältävän taulukon.

Funktio `distribute_configuration` levittää konfiguraatiotietoa prosessilta, jonka tunniste on `root` muille kommunikaattorin `comm` prosesseille.

Tietotyyppiä `nodes_str` käytetään koneiden nimien talletustietorakenteena.

Funktio `get_nodes` palauttaa prosessille, jonka tunniste on `root`, `nodes_str`-tyyppisistä alkioista muodostuvan taulukon, joka sisältää kaikkien kommunikaattorin `comm` prosessien ajokoneiden nimet.

Funktio `remap_processes` uudelleenkartoittaa prosesseja pyrkimyksenään sovittaa konfiguraatiotiedostossa asetetut prosessien käynnistymiskoneet ja mpirunin arpomat käynnistymiskoneet toisiinsa siten, että konfiguraatiotiedoston asetukset otetaan mahdollisimman hyvin huomioon.

5.2.8 Sekalaisia apufunktioita

5.2.8.1 Yleiskuva

Tässä luvussa esitellään joukko sekalaisia yleiskäyttöisiä apufunktioita, jotka eivät luontevasti sovi mihinkään edellä kuvatuista moduuleista.

5.2.8.2 Toiminnot

5.2.8.2.1 Muistinhallinta

```
__malloc_ptr_t bkmalloc (size_t __size);
```

```
__malloc_ptr_t bkrealloc (__malloc_ptr_t __ptr, size_t __size);
```

```
__malloc_ptr_t bkfree (__malloc_ptr_t __ptr);
```

Funktiot `bkmalloc`, `bkrealloc` ja `bkfree` toimivat muuten samoin kuin vastaavat C-kirjaston funktiot `malloc(3)`, `realloc(3)` ja `free(3)`, mutta eivät koskaan palauta `NULL`-osoitinta. Virhetilanteet käsitellään makrojen `bk_error` ja `bk_perror` avulla, jotka tulostavat virheilmoituksen ja lopettavat ohjelman suorituksen.

5.2.8.2.2 Virheenkäsittely

```
#define bk_error(str) { ... }
```

```
#define bk_perror(str) { ... }
```

Makro `bk_error` tulostaa standardivirhevuohon (`stderr`) virheilmoituksen, josta on luettavissa virheen aiheutumiskohta lähdekoodissa ja merkkijono `str`. Lisäksi `bk_error` kutsuu MPI-kirjaston funktiota `MPI_Abort`, joka lopettaa kaikkien orientointijärjestelmän prosessien toiminnan.

Kaikki prosessit lopetetaan, koska orientointijärjestelmän vikasietoisuus perustuu ajatukselle, että onnettumuuden sattuessa nopein ja varmin tapa toipua on lopettaa ohjelma ja käynnistää se uudestaan siitä, mihin laskenta edellisellä kerralla jäi (katso vikasietoisuus 8.3).

Makro `bk_perror` toimii muuten kuten edellä kuvattu `bk_error`, mutta täydentää virheilmoitusta lisäämällä siihen C-kirjaston viimeisen virheen virheilmoituksen kirjastofunktion `perror(3)` tapaan.

5.2.9 Pääohjelma

Koska orientointijärjestelmä jakautuu useaan erilliseen prosessiin, joiden roolit laskennassa poikkeavat toisistaan täysin, on järjestelmän pääohjelma rakennettu siten, että se pystyy joustavasti käynnistämään prosessin roolista riippuen oikean alipääohjelman. Alipääohjelmat muodostavat itse asiassa jokainen oman erillisen alimoduulinsa.

Orientointijärjestelmä Beklun pääohjelmamoduuli koostuu pienestä yhteisestä osasta ja joukosta vaihtoehtoisia alipääohjelmia. Yhteiseen osaan viitataan termillä pääohjelma ja se on C-kielelle perinteisesti muotoa `int main(int argc,char *argv[])`. Tämä pääohjelma suoritetaan erikseen jokaisella orientointijärjestelmään kuuluvalla prosessilla.

5.2.9.1 Pääohjelmamoduulin palvelut

```
typedef int t_proc_func(int proc_num);
```

Tämä on funktiotyyppi, joka määrää orientointijärjestelmän eri prosessityyppien pääohjelmien parametrit ja palautusarvon. Parametri on prosessin prosessitunnus (katso kohtaa prosessin tunnus alempana 5.2.9.7). Palautusarvo poikkeaa nollasta, jos alipääohjelma on keskeytynyt epänormaalisti, muutoin palautusarvo on nolla.

```
t_proc_func *get_type_func(int proc_type);
```

`get_type_func` palauttaa funktio-osoittimen alipääohjelmaan annetulle prosessityypille. Parametri on tyyppiä `enum_process_types`, joka viittaa alipääohjelman tyyppiin (katso kohtaa alipääohjelmat 5.2.9.11). Palautusarvo on alipääohjelman funktio-osoitin.

```
char *proc_type_name(int proc_type);
```

Parametri on tyyppiä `enum_process_types`, joka viittaa alipääohjelman tyyppiin (katso kohtaa alipääohjelmat 5.2.9.11). Palautusarvo on merkkijono, joka sisältää orientointijärjestelmän sisäisen prosessityypin selkokielisenä tekstinä. Tätä palvelua voidaan käyttää hyödyksi esimerkiksi virheilmoitusten tulostamisessa.

5.2.9.2 Perl konfiguraatio

Pääohjelma suorittaa kullakin prosessilla konfiguraatiotiedoston sulautetun Perl-tulkin avulla. Konfiguraatiotiedoston suorittaminen saattaa jokaisen orientointijärjestelmän prosessin konfiguraatiomoduulin sisältämät tiedot yhtenäiseen tilaan. Voidaan ajatella, että konfiguraatiotiedostoon on koodattu orientointijärjestelmän kaikille osille yhteiset globaalien muuttujien arvot (katso Konfiguraatiomoduuli 5.2.6.1 ja Käyttöliittymä 9).

5.2.9.3 MPICH:n konetiedosto

Konetiedosto on tekstitiedosto, joka määrittää ne koneet ja niillä ajettavat binääritiedostot, jotka muodostavat MPI-sovelluksen eri prosessit.

Tiedosto koostuu riveistä, joista kukin määrittää yhden laskentaan osallistuvan prosessin. Yksi rivi sisältää sen solmun DNS-nimen, jolla prosessi on tarkoitus käynnistää ja polun ajettavaan binääritiedostoon kyseisessä solmussa. Lisäksi rivillä on mahdollista määritellä kullekin prosessille yksilöllisiä käynnistysparametreja.

5.2.9.4 Äitiprosessi

Ensimmäinen suoritettava MPI-prosessi on kaikkien muiden prosessien äitiprosessi. Äitiprosessi on MPI:n mpich-toteutukselle ominainen ratkaisu. Äitiprosessi käynnistää muut prosessit oikeille koneille ja jatkaa tämän jälkeen suoritustaan yhtenä laskentaprosessina.

5.2.9.5 Muiden prosessien käynnistäminen

Äitiprosessi lukee aluksi konfiguraatiotiedoston ja kirjoitetaan sen perusteella konetiedoston. Kun konetiedosto on valmis, äitiprosessi suorittaa `MPI_Init`-kutsun (katso MPI-kirjaston alustus 5.2.9.6).

Konetiedosto kirjoitetaan konfiguraatiomoduulin funktiolla `write_mpich_machine_file` (katso Konfiguraatiomoduuli 5.2.6.1).

Orientointijärjestelmän muut prosessit käynnistetään MPICH:n konetiedoston mukaan äitiprosessin `MPI_Init`-kutsun aikana.

5.2.9.6 MPI-kirjaston alustus

MPI-ympäristö toimii vasta kun MPI:n alustusfunktioita `MPI_Init(&argc,&argv)` on kutsuttu. Ennen `MPI_Init`-kutsua muut MPI-kirjastofunktiot eivät toimi ja muita laskentaan liittyviä prosesseja kuin äitiprosessi ei ole käynnistetty.

Äitiprosessin `MPI_Init`-kutsu käynistää muut prosessit MPICH:n konetiedoston perusteella. Muiden prosessien vastaavat `MPI_Init` -kutsut, eivät tarvitse konetiedstoa.

Vasta `MPI_Init`-kutsun jälkeen prosessi tietää oman identiteettinsä muiden MPI-prosessien joukossa (katso Prosessin tunnus 5.2.9.7). Kaikki orientointijärjestelmän prosessit kutsuvat `MPI_Init`-funktiota.

5.2.9.7 Prosessin tunnus

Prosessin tunnus orien-taatiojär-jestelmässä on prosessin järjestysnumero MPI:n äiti-prosessiryhmässä. Äitiprosessiryhmä koostuu kaikista prosesseista, jotka käynnistetään yhdellä kertaa MPI-ohjelman suorituksen alkaessa.

`MPI_COMM_WORLD` on kaikkien äitiprosessiryhmään kuuluvien prosessien MPI-kommunikaatioryhmä, jonka kautta ne voivat kommunikoida keskenään.

Prosessin tunnuksen saa siis selville kutsumalla `MPI_Comm_rank( MPI_COMM_WORLD, &myrank )`, jolloin muuttujan `myrank` arvoksi tulee prosessin tunnus eli kutsujan järjestysnumero äitiprosessiryhmässä.

5.2.9.8 Alipääohjelman valitseminen

Kun MPI-kirjasto on alustettu `MPI_Init` kutsulla ja konfiguraatiotiedosto on käsitelty, pääohjelma valitsee prosessin tunnuksen ja konfiguraatiomoduulin tietojen perusteella kyseisen prosessin tehtävän ja käynnistää oikean alipääohjelman.

Pääohjelma valitsee oikean alipääohjelman kutsumalla ensin funktiota `get_proc_type`, joka ottaa parametrikseen prosessin tunnuksen ja palauttaa prosessille konfiguraatiotiedostossa `define_process`-komennolla määrätyn tyypin. Seuraavaksi pääohjelma kutsuu funktiota `get_type_func`, joka ottaa parametrikseen prosessin tyypin ja palauttaa funktio-osoittimen oikeaan alipääohjelmaan. Seuraavaksi pääohjelma siirtää kontrollin alipääohjelmalle kutsumalla funktio-osoittimen osoittamaa funktiota.

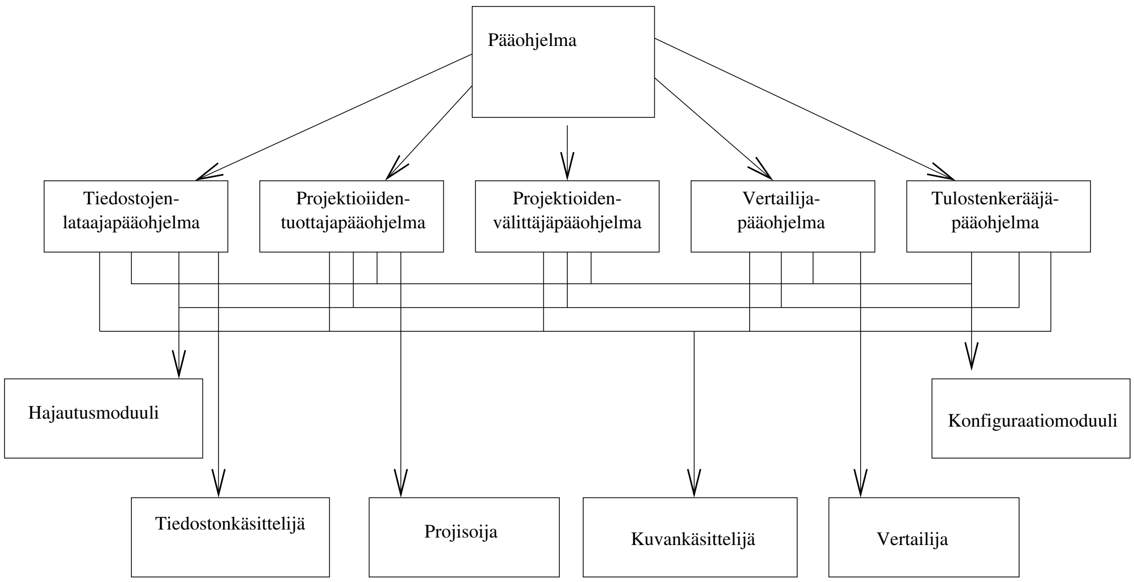
5.2.9.9 Alipääohjelman ennenaikainen keskeytyminen

Kun alipääohjelma lopettaa suorituksensa, pääohjema tarkistaa alipääohjelma palautusarvon. Jos alipääohjelma palauttaa nollasta poikkeavan arvon, pääohjelma tulkitsee, että laskenta on keskeytynyt epänormaalisti. Jos laskenta on keskeytynt epänormaalisti, pääohjelma kutsuu `bk_error` funktiota, joka välittää virheilmoituksen käyttäjälle ja lopettaa kaikkien orientointijärjestelmän prosessien toiminnnan.

5.2.9.10 Kommunikaation vastapuolen selvittäminen

Jotta MPI-kirjastofunktioilla voisi kommunikoida prosessien välillä täytyy tietää MPI-kommunikaatioryhmä ja kohteen tunnus ryhmän sisällä. Konfiguraatiomoduuli tarjoaa tähän palvelun. Funktiolla `get_proc_numbers` saa taulukon tiettyyn orientointijärjestelmän prosessityyppiin kuuluvien prosessien tunnuksista (katso Prosessin tunnus 5.2.9.7). Näistä pro-ses-si-tun-nuk-sis-ta voi-daan muo-dos-taa MPI-kommu-ni-kaa-tio-ryh-mä kut-su-mal-la MPI-kir-jas-to-funk-ti-oi-ta.

Vastapuolen selvittäminen perustuu siihen ajatukseen, että tietty prosessi tietää minkätyyppisten muiden prosessien kanssa se haluaa kommunikoida. Nyt prosessi



Kuva 5.2: Pääohjelman suhteet muihin moduuleihin

voi kysyä kaikki haluamansa tyypin prosessit ja muodostaa niistä ja itsestään MPI-kommunikaatioryhmän.

Joissakin tapauksissa halutun tyyppisiä prosesseja on vain yksi (esimerkiksi projektioiden välittäjä -prosessi), jolloin kommunikoidaan ryhmän ainoan toisen jäsenen kanssa. Jos ryhmässä on useita jäseniä (esimerkiksi vertailijaprosessit), kommunikoija voi laittaa itsensä ryhmän ensimmäiseksi (indeksi 0) jäseneksi ja kommunikoida vuorollaan kunkin ryhmän jäsenen kanssa.

5.2.9.11 Alipääohjelmat

Alipääohjelmat muistuttavat toisiaan, vaikka jokaisella niistä on oma tarkoitus ja toimintatapa. Kaikki alipääohjelmat kutsuvat konfiguraatiomoduulia ja rinnakkaistamismoduulia ja kaikki alipääohjelmat kommunikoivat ainakin yhden toisentyyppisen alipääohjelman kanssa (katso tietovuokaavio 8.2.5.1).

Alipääohjelmat poikkeavat toisistaan monissa suhteissa. Eri alipääohjelmat käyttävät kukin niitä muita orientointijärjestelmän moduuleita, jotka tukevat niiden omaa erityistehtävää. Ohessa on laajennettu moduulien riippuvuusuhteita esittävä kaavio, jossa jokainen alipääohjelma on esitetty omana moduulinaan (kuva 5.2).

Alipääohjelmat on määritelty pääohjelmamoduulissa seuraavasti:

```
int file_loader(int myRank);
int result_collector(int myRank);
int projection_relay(int myRank);
int projection_producer(int myRank);
int comparator(int myRank);
```

Kukin alipääohjelma vastaa yhtä orientointijärjestelmän prosessityyppiä. Alipääohjelma `file_loader` vastaa tiedostojen lataaja -pro-ses-si-a, `result_collector` vastaa tulosten kerääjä -pro-ses-si-a, `projection_relay` vastaa pro-jek-ti-oi-den välittäjä -prosessia, `pro-jecti-on_pro-ducer` vastaa pro-jek-ti-oi-den tuot-taja -prosessia ja `com-pa-ra-tor` ver-tailijaprosessia.

Parametri `myRank` kertoo alipääohjelmalle sen prosessitunnuksen. Palautusarvo on erisuuri kuin nolla, jos alipääohjelman laskenta loppuu normaalisti. Palautusarvo on nolla, jos laskenta keskeytyy epänormaalisti. Alipääohjelmista kerrotaan tarkemmin alempana.

5.2.9.12 Tulosten kerääjä -prosessi

Tulosten kerääjä -prosesseja on orientointijärjestelmässä ajon aikana vain yksi. Tämä prosessi kuuntelee laskennan tuloksia ja säilyttää parhaiten sopineet orientaatiot kullekin elektronimikroskooppikuvalle. Tulostenkerääjä kirjoittaa jaksoittain kovalevylle tietoja, joita käytetään orientointijärjestelmän keskeytyneen laskennan uudelleenkäynnistämiseen (katso Vikasietoisuus 8.3).

Tulostenkerääjä kuuntelee kaikkia vertailijaprosesseja rinnakaistamismoduulin palvelulla `recv_result`. Tulostenkerääjä pystyy kuuntelemaan uusia tuloksia samalla kun se käsittelee aikaisemmin vastaanotettua tulosta. Ajoittaisen uudellenkäynnistysinformaation kirjoittamisen yhteydessä tulostenkerääjä joutuu katkaisemaan tulosten keräämisen hetkeksi, jotta levylle saadaan kirjoitettua yhtenäinen tila.

5.2.9.12.1 Tulosten kirjanpito Tulostenkerääjä joutuu pitämään kirjaa siitä, onko tiettyä projektiota verrattu kaikkiin elektronimikroskooppikuviin. Kirjanpidon avulla tiedetään koska laskenta on valmis ja koska jokin tietty projektio on käsitelty kokonaan ja tieto sen käsittelystä voidaan kirjoittaa kovalevylle keskeytyneen laskennan uudelleenaloittamista varten.

Koska projektioita on kesken enintään projektioiden välittäjä -prosessin puskuriin mahtuva määrä, tulosten kerääjä ei joudu kerralaan pitämään kirjaa kaikista mahdollisista projektiosta. Kirjanpitoon riittää, että tulosten kerääjä pitää yllä ympyräpuskuria laskureista siten, että ympyräpuskuriin mahtuu yhtä monta laskuria kuin projektioiden välittäjän puskuriin mahtuu projektioita.

Kun vertailijalta saapuu tulostietue, tulosten kerääjä lisää projektion tunnuksen perusteella oikean ympyräpuskurin sisältämän laskurin arvoa yhdellä . Kun laskurin arvo on saavuttanut elektronimikroskooppikuvien lukumäärän, tiedetään että laskurin kuvaamaa projektiota on verrattu kaikkiin elektronimikroskooppikuviin. Tässä vaiheessa voidaan projektion tunnus kirjoittaa käsitellyt pro-jek-ti-o-kul-mat -tie-dos-toon.

5.2.9.12.2 Tietorakenteet Tulostenkerääjän tärkein tietorakenne on dynaamisesti varattu kaksiulotteinen taulukko, joka sisältää käyttäjän määrittelemän määrän parhaita tulostietueita kutakin syötteenä annettua elektronimikroskooppikuvaa kohti. Tulostietue on tyyppiä `result`, joka on määritelty tarkemmin kohdassa Tietorakenteet (5.2.7.2.2). Taulukossa on siis eletronimikroskooppikuvien määrä kertaa käyttäjän määrittelemä `max_best_results` arvo (katso kohta konfiguraatiomoduuli 5.2.6.1) `result` tyypistä alkiota.

Toinen olennainen tietorakenne on kirjanpitoon käytetty ympyräpuskuri. Se on selitetty kohdassa Tulosten kirjanpito 5.2.9.12.1.

5.2.9.13 Tiedostojen lataaja -prosessi

Tiedostojen lataaja -prosesseja on orientointijärjestelmässä ajon aikana vain yksi. Tiedostojen lataajan tehtävä on syötetietojen lataaminen eri tiedostoista ja niiden välittäminen eteenpäin muille prosesseille.

5.2.9.13.1 Kolmiulotteisen mallin välittäminen Tiedotojen lataaja -prosessi lähettää aluksi makromolekyylikompleksin kolmiulottesen mallin kaikille projektioiden tuottaja -prosesseille. Malli lähetetään kullekin projisoijalle rinnakkaistamismoduulin funktiolla `void send_model(char *modnm,int dest,MPI_Comm comm)` (katso kohta mallin välitys 5.2.7.3.4).

5.2.9.13.2 Elektronimikroskooppikuvien jakaminen Kun kolmiulotteiset mallit on lähetetty prosessi siirtyy lähettämään elektronimikroskooppikuvia vertailijaprossesseille. Kullekin vertailijaprosessille lähetetään eri elektronimikroskooppikuvat.

Käyttäjä on määritellyt konfiguraatiotiedostossa elektronimikroskooppista saatujen kuvien tiedostonimet, joita on tarkoitus käsitellä, ja kullekin vertailijaprosessille tarkoitettujen kuvien lukumäärän. Tiedostojen lataaja -prosessi jakaa elektronimikroskooppikuvat käyttäjän antamien lukumäärien suhteessa. Kuvat jaetaan käyttäjän antamien lukumäärien suhteessa, jotta käyttäjän ei tarvitsisi huolehtia siitä, että lukumäärien summa on tarkalleen elektronimikroskooppikuvien lukumäärä.

5.2.9.13.3 Laskettavien kulmien välittäminen Tiedostojen lataajan -prosessin viimeinen tehtävä on huolehtia laskettavien kulmien välittämisestä projisoijille. Yksi tavoite on, että kaikki projektioiden tuottaja -prosessit saavat ne kulmat joista projektioita pitää laskea mahdollisimman optimaalisissa ja sopivan kokoisissa ryhmissä. Lisäksi on vielä huolehdittava siitä, että mahdollisesti konfiguraatiomoduulin ilmoittamien lasketut kulmat -tiedostojen sisältämiä kulmia ei välitetä enää projektioiden tuottajille.

Tiedostojen lataaja -prosessil lukee konfiguraatiomoduulin muuttujan `pro-jec-ti-on_ang-les_file_na` osoittaman tiedoston muistiin. Tämä laskettavat kulmat -tiedosto kertoo kaikki kulmat, jotka on tarkoitus käsitellä laskennan aikana.

Seuraavaksi prosessi lukee kaikki `completed_projection_angles_file_name` muuttujan osoittamat valmiit projektiokulmat -tiedostot ja merkitsee niiden osoittamat kulmat erityiseen käsitellyt kulmat -muistikarttaan.

Kolmas tiedosto, jonka tiedostojen lataaja -prosessi lukee, on konfiguraatiomoduulin muuttujan `projection_angle_groups_file` osoittama kulmaryhmät -tiedosto. Kulmaryhmät -tiedostossa on merkittynä ne jaksot kulmatiedostossa, jotka on tarkoitus laskea aina kerrallaan yhdellä projektioiden tuottaja -prosessilla. Jos tämä tiedosto puuttuu, kaikkien jaksojen oletetaan sisältävän konfiguraatiomoduulin muuttujan `max_projection_angle_group_size` osoittaman määrän kulmia.

Näiden kolmen tietorakenteen perusteella tiedostojen lataaja -prosessi siirtyy tarjoamaan projektioiden tuottajille laskettavia kulmia. Menetelmä jolla kulmaryhmät muodostetaan on yksinkertainen. Käsiteltävät kulmat -tiedostoa käydään järjestyksessä läpi poistaen ne kulmat jotka on merkitty käsitellyt kulmat -muistikarttaan.

Aina kun on käyty läpi yhden kulmaryhmätiedostossa määritellyn ryhmän kulmat, jäljelle jääneet kulmat sijoitetaan yhteen puskuriin, jonka lähetys aloitetaan sille ensimmäiselle projektioiden tuottaajalle, joka on jo vastaanottanut edellisen joukon kulmia. Kulmajoukot lähetetään kutsumalla rinnakkaistamismoduulin funktiota `send_projection_reqs`.

Kun käsiteltäviä kulmia ei ole enää jäljellä tiedostojen lataaja -prosessi siirtyy odottamaan, että tulostenkerääjä prosessi -lopettaa koko järjestelmän toiminnan saatuaan viimeisen tuloksen kirjattua levylle.

5.2.9.14 Projektioiden välittäjä -prosessi

Projektioiden välittäjän toimintaperiaate on yksinkertainen. Se kuuntelee projektioita projektioiden tuottajilta ja kirjoittaa niitä järjestyksessä puskuriinsa. Lisäksi se tarjoaa kullekin vertailijalle ensimmäistä projektiota, jota kyseinen vertailija ei vielä ole käsitellyt. Koska lähetys ja vastaanotto-operaatiot ovat synkronisia projektioiden välittäjä pakottaa projektioiden tuottajat odottamaan, jos puskurissa ei ole tilaa uudelle projektiolle. Vastaavasti jos vertailijaprosessi on kiireinen, projektioiden välittäjän lähetysoperaatio kestää kunnes vertailija on valmis ottamaan vastaan. Katso lisää kohdasta: Synkroninen viestinvälitys 8.2.4.

Projektioiden välittäjällä on yksi olennainen tietorakenne eli rengaspuskuri, joka sisältää projektioita, joiden laskenta on vertailijoilla kesken tai ei ole vielä alkanutkaan. Käyttäjä voi määritellä puskurin enimmillään sisältämien projektioiden määrän konfiguraatiomoduulin muuttujalla `projection_buffer_size` (katso lisää projektioiden välittäjä -prosessista luvusta Hajautuksen toteutus 8.2.2.1)

Projektioiden välittäjä kuuntelee projektioita projektioiden tuottajilta rinnakkaistamismoduulin funktiolla `recv_projection` ja lähettää projektioita vertailijoille funktiolla `send_pro-jec-tion`.

5.2.9.15 Projektioiden tuottaja -prosessi

Projektioiden tuottaja -prosesseja on orientointijärjestelmässä ajon aikana määrä, joka saadaan selville konfiguraatiomoduulin funktion `get_proc_numbers` avulla.

Funktiolla `get_proc_numbers` saadaan selville tietyn tyyppisten prosessien lukumäärä.

Prosessi vastaannottaa aluksi kolmiulotteisen mallin rinnakkaistamismoduulin funktiolla `recv_model`. Seuraavaksi projektioden tuottaja vastaanottaa joukon kulmia, joista tarvitaan projektioita ja käynnistää ei blokkaavasti seuraavan vastaanotto-operaation.

Projektioiden tuottaja -prosessi ottaa mallista tarvittavien kulmien mukaisia projektioita kutsumalla projisointimoduulin (katso kohtaa 5.2.2) palveluita.

Prosessi lähettää valmiita projektioita projektioiden välittäjälle sitä mukaan kun niitä valmistuu. Lähettäminen tapahtuu rinnakkaistamismoduulin funktiolla `send_projection`.

5.2.9.16 Vertailijaprosessi

Vertailijaprosessi vastaanottaa projektioita projektioiden välittäjältä ja vertailee niitä elektronimikroskooppikuviin. Vertailujen tulokset vertailijaprosessi lähettää tulosten kerääjälle, joka valitsee niistä parhaat ja huolehtii, että laskennan keskeytyessä olennaiset tulokset eivät häviä.

5.2.9.16.1 Elektronimikroskooppikuvien vastaanottaminen Vertailijaprosessi aloittaa toimintansa vastaanottamalla joukon elektronimikroskooppikuvia tiedostojen lataaja -prosessilta. Vastaanotettavien elektronimikroskooppiviuvien määrän vertailijaprosessi saa selville samalla tavoin kuin tiedostojen lataaja -prosessi sen laskee (katso tiedostojen lataaja 5.2.9.13). Elektronimikroskooppikuvat vastaanotetaan hajauttamismoduulin funktiolla `recv_pizza` yksi kerrallaan.

5.2.9.16.2 Projektioiden vastaanottaminen Kun vertailijaprosessi on vastaanottanut elektronimikroskooppikuvansa ja tallettanut ne muistiin, prosessi siirtyy kuuntelemaan projektioiden välittäjältä projektioita hajauttamismoduulin funktiolla `recv_projection`. Prosessi vastaanottaa seuraavan joukon projektiota ei blokkaavasti. Samalla se laskee vertailuja edelliselle projektiolle ja lähettää niiden tuloksia tulosten kerääjä -prosessille.

5.2.9.16.3 Vertailu Vertailu toteutetaan kutsumalla vertailijamoduulin palveluita (katso kohtaa Vertailijamoduuli 5.2.3). Vertailijaa kutsutaan erikseen jokaista projektiota ja elektronimkroskooppikuvaa kohden. Vertailija palauttaa joukon parhaita orientaatioita hyvyysarvoinen.

5.2.9.16.4 Tulosten lähettäminen Vertailijan palauttamat tiedot kootaan `result` tietueeseen yhdessä projektion ja elektronimikroskooppikuvan tunnisteiden kanssa ja lähetetään eteenpäin tulosten kerääjä -prosessille jatkokäsittelyä varten. Tulokset lähetetään kutsumalla hajauttamismoduulin funktiota `send_result`

Luku 6

Tietorakenteet

6.1 Kuvan esitysmuoto muistissa

Kuvat voivat olla yksi-, kaksi- tai kolmiulotteisia kokonais-, reaali- tai kompleksilukutaulukoita, joissa on tieto kuvan dimensioista ja sen keskipisteestä:

```
typedef struct {
 int x, y, z;
 double x0, y0, z0;
 uchar *data;
} image_uchar;

typedef struct {
 int x, y, z;
 double x0, y0, z0;
 real *data;
} image_real;

typedef struct {
 int x, y, z;
 double x0, y0, z0;
 complex *data;
} image_complex;
```

Origon ollessa $(0, 0, 0)$ se osoittaa kuvan vasemmassa alanurkassa syvimmällä olevan pikselin (tai vokselin) keskipisteeseen. Oletamme pikselien (ja vokselien) olevan pisteitä. Kuvat sijoitetaan muistiin sanakirjajärjestykseen niin, että akselit vähiten merkitsevästä eniten merkitsevään ovat X, Y ja Z, jolloin kolmiulotteinen kuva koostuu jonosta kaksiulotteisia Z-akselin kanssa kohtisuorassa olevia kuvia (katso kuva 6.1).

Reaali- ja kompleksiluvut peritään FFTW-kirjastosta ja kokonaislukutyyppinä käytetään kahdeksanbittistä etumerkitöntä kokonaislukua `unsigned char`.

```
typedef unsigned char uchar;
typedef fftw_real real;
typedef fftw_complex complex;
```

FFTW:n kyseiset tietotyypit on määritelty seuraavasti:

```
typedef double fftw_real;

typedef struct {
 fftw_real re, im;
} fftw_complex;
```

Nämä tietorakenteet on esitelty moduulissa Kuvankäsittelijä 5.2.4.

6.2 Orientaation esitys

Katselusuunta (eli orientaatio) kolmiulotteisessa avaruudessa esitetään ohjelmistossamme kolmen kulman avulla alla olevan tietorakenteen mukaisesti.

```
typedef struct {
 double a1, a2, a3;
} orientation;
```

Orientaatio voidaan esittää kolmiulotteisessa avaruudessa kolmen kulman avulla kahdellakymmenelläneljällä tavalla, joista jokaista kutsutaan Eulerin kulmiksi [37].

Kappaletta käännetään aina X- Y- ja Z-akselien ympäri. Kiertomatriisit määritellään seuraavasti:

$$R_x(\Theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Theta & \sin -\Theta \\ 0 & \sin\Theta & \cos\Theta \end{bmatrix}$$

$$R_y(\Theta) = \begin{bmatrix} \cos\Theta & 0 & \sin\Theta \\ 0 & 1 & 0 \\ \sin -\Theta & 0 & \cos\Theta \end{bmatrix}$$

$$R_z(\Theta) = \begin{bmatrix} \cos\Theta & \sin -\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

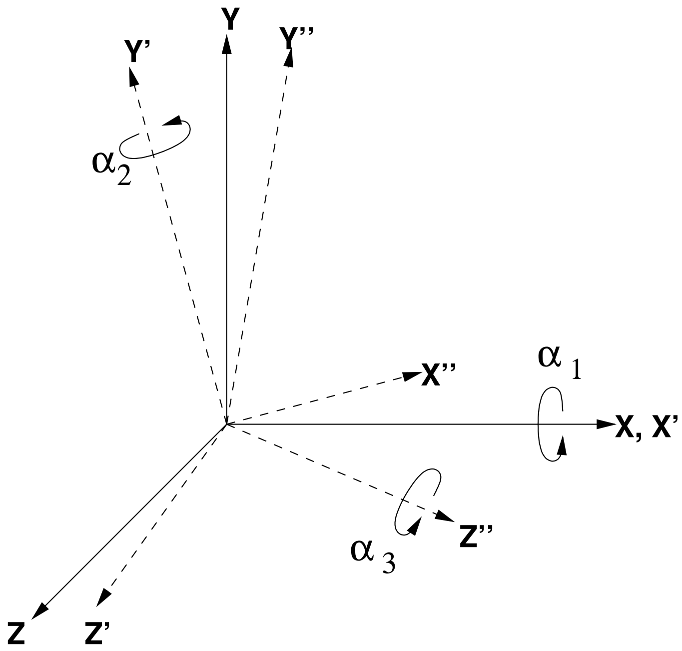
Me käytämme Ken Shoemaken termistöä käyttäen XEDR-järjestelmää [37]. Z tarkoittaa sitä, että ensimmäisenä käännetään Z-akselin ympäri, E kertoo kulman Z, Y, X olevan negatiivinen, D:n perusteella tiedetään, että ensimmäinen ja viimeinen rotaatiomatriisi eivät ole samoja ja R kertoo, että kappaletta käännetään käännettyjen akselien suhteen. Käytännössä kääntäminen tapahtuu kertomalla vektori matriisilla $R_x(\alpha_1)R_y(\alpha_2)R_z(\alpha_3)$. (Tämä voidaan tulkita ZODS-järjestelmäksi kuvan 6.1 mukaisesti.)

$$\begin{bmatrix} \cos\alpha_2\cos\alpha_3 & -\cos\alpha_2\sin\alpha_3 & \sin\alpha_2 \\ \sin\alpha_1\sin\alpha_2\cos\alpha_3 + \cos\alpha_1\sin\alpha_3 & -\sin\alpha_1\sin\alpha_2\sin\alpha_3 + \cos\alpha_1\cos\alpha_3 & -\sin\alpha_1\cos\alpha_2 \\ -\cos\alpha_1\sin\alpha_2\cos\alpha_3 + \sin\alpha_1\sin\alpha_3 & \cos\alpha_1\sin\alpha_2\sin\alpha_3 + \sin\alpha_1\cos\alpha_3 & \cos\alpha_1\cos\alpha_2 \end{bmatrix}$$

6.3 Viestirakenteet

Viestirakenteet tarvitaan kaikien yleisimpien tietorakenteiden välitykseen(makromolekyylikompleksin kolmiulotteiselle mallille, mallista projisoidulle projektiolle ja elektronimikroskooppikuvalle), laskettaville kulmäväleille ja vertailu operaatioden tulostiedolle:

```
typedef struct {
```



Kuva 6.1: Beklu -järjestelmän orientaationesitysmuoto.

```
  image *img;
} model;

typedef struct {
  int id;
  orientation orient;
  image *img;
} projection;

typedef struct {
  int id;
  image *img;
} pizza;

typedef struct {
  int id;
  orientation orient;
} projection_req;

typedef struct {
```

```
  int pizzaId;
  int orientId;
  orientation orient;
  double goodness[MAX_GOODNESS_VALS];
} result;
```

Hajautustietorakenteet on esitelty tarkemmin moduulissa Hajautus 5.2.7.

Luku 7

Algoritmit

Seuraavissa luvuissa d tarkoittaa kuvan dimensioiden lukumäärää ja n kuvan (pisimmän) sivun pituutta.

7.1 Interpolointialgoritmit

Interpolointi noudattaa pääpiirteittäin teoksen Practical Handbook on Image Processing for Scientific Applications [26] esitystä.

7.1.1 Lähimpään naapuriin pyöristys

Kuva voidaan siis ajatella pistejoukkona. Pisteen neliöesitys saadaan laskemalla jokaiselle pisteelle sen Voronoi-alue kuvassa (olettamalla kuvapisteiden olevan vakioarvoisia kuvan ulkopuolella) [10]. (Kuvankatseluohjelmat usein näyttävätkin suurennettaessa pisteiden sijasta niiden Voronoi-alueita!) Lähimpään naapuriin pyöristettämisessä on siis kyse geometrisesta etsinnästä. Siinä katsotaan minkä pisteen Voronoi-alueelle kysytty piste kuuluu ja palautetaan Voronoi-alueen virittävän pisteen intensiteetti.

Pyöristys onnistuu siis ajassa $\Theta(d)$.

7.1.2 Lineaarinen interpolointi

Lähimpään naapuriin pyöristettäessä oletettiin signaalin olevan diskreetti. Yksinkertaisin tapa jatkuvuuden tuottamiseksi on lineaarinen interpolointi, jossa lasketaan useamman pisteen painotettu keskiarvo ratkaisemalla ongelma rekursiivisesti jakamalla avaruus kahteen osaan alla olevan kaavan mukaisesti, kunnes jäljellä on vain yksiulotteinen tapaus:

$$b = (\lceil x \rceil - x) a_{\lceil x \rceil} + (x - \lfloor x \rfloor) a_{\lfloor x \rfloor} \tag{7.1}$$

Lineaarinen interpolointi onnistuu ajassa $\Theta(d)$.

7.1.3 Sinc-interpolointi

Tasavälisesti otetuista näytteistä koostuvaa kuvaa voidaan voidaan käsitellä taajuusrajoitettuna signaalina. Tällöin siis tietystä taajuudesta lähtien kaikki sitä suuremmat taajuudet kuvan Fourier-muunnoksessa ovat nollaa (eli käytetään suorakaiteen muotoista ikkunointifunktiota).

Diskreettiä kuvaa voidaan interpoloida (periaatteessa) mielivaltaiselle tarkkuustasolle lisäämällä kuvan diskreettiin Fourier-muunnoksen riittävästi nolla-arvoisia korkeampitaajuuksisia kertoimia ja laskemalla käänteismuunnos.

Pisteen x arvo voidaan interpoloida kuvasta g seuraavasti:

$$g_{sinc}(x) = \sum_{i \in I} g(x_i) h(x - x_i) \tag{7.2}$$

I on kuvapisteiden indeksien joukko ja h painokerrointen funktio.

Painokertoimet lasketaan seuraavasti:

$$h(x) = \prod_{0 \le j \le d} \frac{\sin \pi x_j}{\pi x_j} \tag{7.3}$$

Kaava voidaan johtaa Fourier-avaruuden kautta taajuusrajoitetun signaalin oletuksista.

Sinc-interpolointi on siis laskennallisesti varsin raskasta laskenta-ajalla $\Theta(n^d)$.

7.2 Projisointialgoritmit

Tehtävänämme on projisoida diskreettejä kolmiulotteisia tiheyskarttoja kaksiulotteisiksi “läpivalaisukuviksi”.

Kaksiulotteiset läpivalaisukuvat muodostuvat, kun kolmiulotteisen kappaleen läpi ammutaan yhdensuuntaisia säteitä[1], jotka mittaavat kohtaamansa massan kokonaismäärän. Diskreetissä tapauksessa jokaista läpivalaisukuvan pikseliä kohti ammutaan yksi säde, joka laskee kohtaamiensa pisteiden intensiteettien summan. Ei ole kuitenkaan aivan yksikäsitteistä, mitkä pisteet kohdataan.

7.2.1 Projektioiden laskeminen suoraviivaisesti

Suoraviivaisin tapa olisi ajatella pisteet pieninä kuutioina ja laskea kuinka paljon massaa jää kunkin läpivalaisukuvan pikselin alle. Pieni kuutio ei kuitenkaan ole todenmukaisin yksinkertaistus.

Toinen tapa projisoida olisi laskea niiden pisteiden summa, joiden Voronoi-aluetta säde leikkaa [24]. Tällöin ongelmaksi muodostuu se, että säde leikkaa eri määrän pisteiden Voronoi-alueita suunnasta riippuen.

Kolmas tapa olisi kääntää kolmiulotteinen kappale ensin sellaiseen asentoon, että summaus voidaan tehdä jonkin akselin suuntaisesti.

Kun halutaan laskea kaikki mahdolliset projektiot (jollakin tarkkuudella), aiemmat ratkaisut ovat varsin tehottomia. Tehokkaampi tapa on kääntää ensin kolmiulotteista kappaletta yhden akselin ympäri ja tämän jälkeen laskea pyörähdysakselin ja toisen valitun akselin virittämien kuvien kaksiulotteisten siivujen kaikki yksiulotteiset projektiot sinogrammien avulla ja yhdistää nämä yksiulotteiset projektiot joukoksi kaksiulotteisia projektioita.

7.2.2 Projektioiden laskeminen Fourier-avaruuden kautta

Ns. Fourier-viipale-teoreeman [16] mukaan kolmiulotteisen kappaleen kaksiulotteisen projektion Fourier-muunnos on sama kuin saman kolmiulotteisen kappaleen

[1] Tämän operaation englanninkielinen nimitys on *X-Ray Tranform* tai *partial Radon transform* [31].

Fourier-muunnoksen kautta samassa kulmassa kulkevan tason leikkaama osa. Samoin kaksiulotteisen läpivalaisukuvan sinogrammi saadaan laskettua sen Fourier-muunnoksesta laskemalla ensin Fourier-muunnoksen napakoordinaattiesitys ja tämän jälkeen laskemalla napakoordinaattiesityksestä joukko yksiulotteisia käänteisiä Fourier-muunnoksia, jolloin koko laskenta tapahtuu ajassa $\Theta(n^2 \log n)$ reaaliavaruudessa tarvittavan $\Theta(n^3)$ sijasta (olettaen, että halutaan käyttää jotain interpolointia).

Nopean Fourier-muunnoksen ansiosta menetelmä on (asymptoottisesti) varsin tehokas erityisesti suurempia projektiojoukkoja laskettaessa [33].

7.3 Vertailualgoritmit

Yksinkertaisin vertailualgoritmi vertaa ovatko kaksi kuvaa identtiset [14]. Tämä ratkaisu ei selvästikään toimi kohinaisella aineistolla vaan on verrattava esimerkiksi kuvien välistä euklidista etäisyyttä. Kumpikin algoritmi toimii selvästi ajassa $\Theta(n^2)$.

Aina kuvien keskitys ei ole kuitenkaan täydellinen vaan tarvitaan siirtymäinvarianttia vertailua. Tämä voidaan tehdä laskemalla kahden kuvan välinen korrelaatiokuva Fourier-avaruuden kautta ajassa $\Theta(n^2 \log n) + \Theta(n^2) + \Theta(2n^2 \log n)$ ja etsimällä siitä suurin korrelaatio ajassa $\Theta(n^2)$. Periaatteelliseksi ongelmaksi muodostuu se, että Fourier-muunnos käsittelee kuvaa jaksollisena funktiona ja se voi teoriassa tuottaa virheellisiä tuloksia korrelaatiokuvaan. Käytännössä ongelmia ei kuitenkaan todennäköisesti esiinny käyttämässämme aineistossa, sillä virusten oletetaan olevan selvästi ympäröivää jäätä raskaampia (eli kuvissa tummempia).

Huomioitaessa vain pieniä siirtymiä ($\Delta \leq \log n$) voidaan vertailu toteuttaa myös laskemalla kuvien väliset euklidiset etäisyyden pienillä siirtymillä. Ikosaedristen virusten tapauksessa tarvittava siirtymä ei olekaan suuri, jolloin tämä yksinkertaisempi versio voi olla harkinnan arvoinen.

Vertailu voidaan tehdä myös hierarkisesti nelipuurakennetta tai aallokkeita käyttäen, jolloin saatetaan saavuttaa pieniä säästöjä laskennassa, kun mahdottomia polkuja ei tarvitse käydä läpi loppuun asti.

7.3.1 Rotaatioinvariantit vertailualgoritmit

Läpivalaisukuvien vertailussa joudutaan kahta kaksiulotteista kuvaa vertaamaan kaikissa mahdollisissa kulmissa. Tämä on varsin raskasta, kun kulmia on kuvan sivuun verrannollinen määrä [23]. (Pessimistisempien arvioiden mukaan niitä on sivun pituuden neliöön verrannollinen määrä [14].)

Laskennan työläyttä voidaan pyrkiä vähentämään mm. hierarkisella vertailulla, polaarimuunnoksella ja sinogrammien vertailulla.

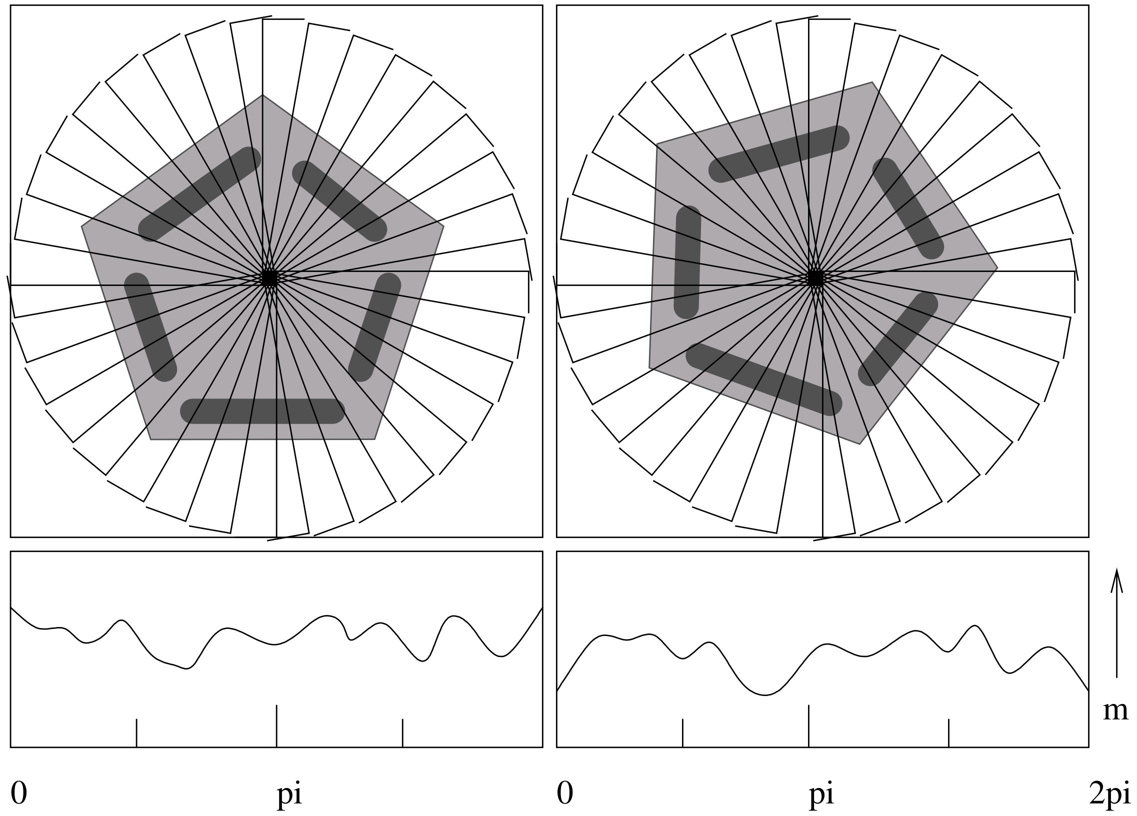
Hierarkisessa vertailussa verrataan kuvia kaikissa kulmissa ensin karkeampina versioina karsien tarkemmista vertailuvaiheista pois ne orientaatiot, jotka eivät voi tulla kysymykseen.

Napakoordinaattiesitykseen muunnettaessa voidaan jokaista kehää verrata yksiulotteisen korrelaation avulla rotaatioinvariantisti. Kuva voidaan muuntaa napakoordinaatistoesitykseen ajassa $\Theta(n^2)$ ja $\Theta(n)$ vertailua voidaan tehdä kukin ajassa $\Theta(n \log n)$, jolloin koko kuvan rotaatioinvariantti vertailu onnistuu ajassa $\Theta(n^2 \log n)$. Menetelmän suurin ongelma on se, että siinä oletetaan (ainakin periaatteessa) keskipisteen olevan kohtalaisen tarkka.

Toinen napakoordinaattimuunnokseen perustuva vertailualgoritmi on sektorien vertailu. Kuva voidaan muuttaa napakoordinaattiesitykseen ajassa $\Theta(n^2)$, sektorit voidaan laskea ajassa $\Theta(n^2)$ ja vertailu onnistuu ajassa $\Theta(n \log n)$, jolloin koko algoritmin suoritusajaksi tulee $\Theta(n^2)$. Myös tässä algoritmissa tehdään oletus kuvan keskipisteestä, mutta se ei ole kuitenkaan yhtä herkkä kuin edellinen. Se soveltuu mahdollisesti esisuotimeksi, jolla poimitaan esimerkiksi $\log n$ parasta ehdotusta.

Sinogrammien avulla kuvia voidaan vertailla rotaatioinvariantisti seuraavasti: lasketaan kuvista Radon-muunnokset (eli sinogrammit) ajassa $\Theta(n^2 \log n)$ ja vertaillaan sinogrammin jokaista riviä toisen sinogrammin jokaiseen riviin ajassa $\Theta(n^3 \log n)$. Algoritmi ei siis tuo parannusta suoritusaikaan, mutta se vähentää dramaattisesti kohinaa ja on siis luotettavampi vertailumenetelmä. Yksinkertaisemmilla vertailuilla (esim. rivien euklidisen etäisyyden laskemisella) päästään kokonaissuoritusaikaan $\Theta(n^2 \log n)$. Käyttämällä vertailukohteena mallin kolmiulotteista Radon-muunnosta päästään suoritusaikaan $\Theta(n^3 \log n)$ mikä on kolmiulotteisen Radon-muunnoksen laskemiseen kuluva aika.

Eräs tapa vertailla kahta kuvaa rotaatioinvariantisti tai ainakin löytää hyviä ehdokkaita niiden väliseksi kiertokulmaksi on laskea kummankin kuvan massajakauma sektoreittain ja vertailla näitä jakaumia toisiinsa. Kuvassa 7.1 on havainnollistettu algoritmin toimintaa, ylhäällä on vertailtavat kuvat ja niiden alapuolella massajakaumat, joita sitten vertaillaan toisiinsa. Jakaumien laskeminen tapahtuu käyttämällä valmiiksi laskettua taulukkoa, joka on vertailtavien kuvien kokoinen, ja jossa jokaista kuvapistettä kohden on painokertoimilla varustettu lista niistä sektoreista, joihin kuvapisteen massa jakautuu. Jos esimerkiksi kuvan keskellä olevien alueiden merkitystä vertailussa halutaan vähentää voidaan se tehdä painottamalla tässä taulukossa niitä vähemmän. Jakaumien laskeminen tämän taulukon avulla onnistuu ajassa $\Theta(n^2)$. Tämän jälkeen etsitään ne vaihesiirtymät, joilla laskettujen massajakaumien neliöpoikkeama on pienimmillään. Tämä voidaan tehdä ajassa $\Theta(A \log A)$, missä A on sektorien määrä. Jos kuvien keskipisteen sijainnissa on pieniä epätarkkuuksia, voidaan toisesta kuvasta laskea haluttu määrä massajakaumia myös sen keskipistettä ympäröivältä alueelta ja valita parhaat tulokset näiden vertailuista ensimmäisen kuvan jakaumaan. Menetelmän etuna on sen nopeus ja se, että summattaessa koko sektorin massa yhteen pienenee kohinan merkitys vertailussa. Haittana on oletus keskipisteen suhtellisen oikeasta sijainnista ja summaamisen aiheuttama informaation menetys tilanteissa, joissa kuvan massa jakautuu kulman suhteen tasaisesti.



Kuva 7.1: Massajakaumien vertailu

Luku 8

Hajautus

8.1 Hajautusstrategiat

Tässä luvussa käsitellään erilaisia mahdollisuuksia hajauttaa orientaatiojärjestelmän vaatimaa laskentaa. Mahdollisuudet esitellään heikkouksineen ja vahvuuksinen, jotta lukija voisi ymmärtää perusteet, miksi orientointijärjestelmän hajautus toteutetaan juuri tietyllä tavalla.

8.1.1 Laskennan hajauttaminen Beowulf-klusterissa

Laskennan hajauttamisessa Beowulf-klusteriin määräävin piirre on yksittäisen prosessorin laskentatehon suhde prosessorien väliseen kommunikointiin. Monissa hajautetuissa algoritmeissa laskennan pullonkaulaksi muodostuu juuri prosessorien välinen kommunikointi. Beowulf klusteri on parempi laskentaintensiivisissä hajautetuissa algoritmeissa,kuin kommunikaatiointensiivisiin algoritmeihin.

Orientointi järjestelmäntoteutuksessa voitaisiin hyötyä toisenlaisista hajautetuista ympäristöistä. Esimerkiksi vektoriprosessoreita (Computer Architectures [11]) olisi mahdollista käyttää laskettaessa suurista tietomääristä matemaattisia muunnoksia (esim. projektion muunnos Fourier-avaruuteen).

Beowulf-klusterissa ei ole yleensä saatavilla vektoriprosessoreiden kaltaisia erikoislaitteistoja vaan yksittäiset prosessorit ovat SISD [11, sivu=20] arkkitehtuurin mukaisia ja hajautus toteutetaan jakamalla suurehkoja laskentakokonaiuuksia prosessoreiden kesken.

8.1.1.1 MPI

Orientointijärjestelmän hajautus toteutetaan MPI-kirjastoa käyttäen. MPI-kirjasto on tarkoitettu yksinkertaistamaan sellaisten laskentatehtävien hajauttamista, joiden kulku voidaan määrätä melko tarkasti etukäteen. MPI standardin versiossa 1.2 määritellään primitiivejä, joilla voidaan luoda uusia MPI-prosesseja laskennan aikana. Nämä mahdollistavat joustavan mukautumisen laskennan aikana muuttuvaan ympäristöön.

MPI 1.2-standardin täyttävää toteutusta ei ole saatavilla klusteriin, jossa orientointijärjestelmä on tarkoitus toteuttaa. Tämä rohkaisee pitämään orientointijärjestelmän hajautuksen mahdollisimman ennaltamäärättynä.

Toinen vaihtoehto olisi tehdä orientointijärjestelmän prosesseista joustavia siten, että ne voivat muuttaa hajautusparametreja laskennan aikana ja jopa vaihtamaan rooliaan laskennassa.

Joustavuus kuitenkin hankaloittaa toeutusta paljon. Tällöin laskennan parametrien alustaminen prosessien käynnistyessä ei riitä vaan niiden muutosten vaikutus on pysytyttävä huomioimaan myös kesken laskennan.

8.1.2 Mahdolliset hajautusstrategiat

Tässä vertailulla tarkoitetaan yhden projektion vertaamista yhteen elektronimikroskooppikuvaan. Vertailualgoritmi voi olla rotaatioinvariantti tai rotaatiovariantti. Projektio voi siis olla käännetty asteverkon mukaisiin kulmiin tai sitten ei.

Riippumatta siitä ovatko vertailut rotaatioinvarianttieja, niitä suoritetaan laskennan aikana useita miljoonia kappaleita. Yhteen vertailuun käytetty aika verrattuna koko laskentaan käytettyyn aikaan on häviävän pieni.

8.1.2.1 Monta prosessoria yhtä vertailua kohti

Yksi tapa on Lähestyä hajauttamista on se, että yksittäinen vertailu jaettaisiin useiden prosessoreiden kesken. Klusterissa olisi meneillään vain vähän vertailuja kullakin hetkellä.

Tässä mallissa verkkoliikennettä syntyy paljon jokaista yksittäistä vertailua kohti ja lisäksi yhden hajautetun vertailun hallintaan kuluu ylimääräistä prosessoriaikaa. Lisäksi hajautetun vertailun toteuttaminen saattaa olla monimutkaista, sillä hajautus on tässä melko matalalla tasolla.

8.1.2.2 Yksi vertailu yhtä prosessoria kohti

Tässä yksittäisen vertailun suorittaa yksi määrätty prosessori, samalla kun muut prosessorit suoritavat toisia vertailuja. Koska yhteen vertailuun tarvitaan yksi projektio ja yksi elektronimikroskooppikuva ja elektronimikroskooppikuvien määrä on useita satoja ja projektioiden määrä useita satoja tuhansia, on olemassa ilmeisiä vaihtoehtoja vertailujen jakamiseen.

8.1.2.2.1 Yksi mikroskooppikuva ja yksi prosessori Tässä versiossa tietty mikroskooppikuva on vain yhden prosessorin työmuistissa ja tämä prosessori suorittaa vertailut tämän mikroskooppikuvan ja kaikkien projektioiden välillä.

Tällöin yksi mikroskooppikuva on vain yhden kerran klusterin muistissa. Koska projektioiden laskeminen on hidasta joudutaan huolehtimaan siitä että järjestelmä laskee kaikki tiettyyn projektioon liittyvät vertailut sinä aikana kun projektio on mahdollista pitää järjestelmän muistissa.

Koska projektioiden kokonaismäärä ylittää moninkertaisesti klusterin muistikapasiteetin, aika jonka yksi projektio voi olla muistissa on verrattain lyhyt.

8.1.2.2.2 Yksi projektio ja yksi prosessori Tässä yhdellä prosessorilla tulee olla muistissa kaikki mikroskooppikuvat, jotta se voi suorittaa tehokkaasti yhteen projektioon liittyvät vertailut. Mikroskooppikuvien säilytyksen vaatima muistimäärä voi kuitenkin olla helposti suurempi kuin solmussa saatavilla oleva muistin määrä.

Koska vain yksi prosessi tarvitsee tiettyä projektiota, valmista projetiota ei tarvitse säilyttää keskitetysti sen jälkeen kun oikea prosessori on saanut sen työmuistiinsa.

8.1.2.2.3 Välimuodot Olisi myös mahdollista antaa yksi mikroskooppikuva esimerkiksi kahdelle prosessorille. Tällöin toinen laskee mikroskooppikuvan vertailut kohti osaa projektioista kohti ja toinen prosessori toista puolta kohti.

Tällaisten välimuotojen mahdollistaminen vaatii järjestelmältä joustavuutta ja on vaikeampaa toteuttaa kuin yksinkertaisemmat mallit. Toisaalta vertailujen jakaminen joustavasti tarkoittaa, että järjestelmä on mahdollista sopeuttaa ääritilanteisiin, kuten klusteriin, jonka solmujen määrä tai niiden ominaisuudet saattavat vaihdella laskennan aikana.

8.1.2.3 Muunnoksien hajauttaminen

Riippuen vertailualgoritmista syötteille saatetaan haluta tehdä erilaisia muunnoksia (Esimerkiksi muunnos Fourier-avaruuteen). Myös laskennan syöteenä saatavalle kolmiulotteiselle mallille voi olla tarpeen tehdä laskennallisesti vaativia muunnoksia.

On olemassa kirjastoja, jotka mahdollistavat yleisesti käytettyjen muunnosten jakamisen eri prosessoreille. Esimerkiksi Fourier-muunnoksen hajauttaminen Beowulf-ympäristössä on suoraan mahdollista FFTW-kirjaston avulla.

Fourier-muunnoksen hajauttaminen lisää operaation suorittamiseen tarvittavaa kokonaislaskenta-aikaa ja aiheuttaa verkkoliikennettä.

Koska yksi kolmiulotteisen mallin kaksiulotteinen projektio on pieni ja niitä on lukumääräisesti paljon, on helpompaa hajauttaa muunnokset siten, että yksi prosessori laskee yhden kokonaisen muunnoksen yhdelle projektiolle ja hajautus toteutetaan jakamalla kokonaisia projektioita prosessoreille.

Jos koko kolmiulotteiselle mallille on tarpeen tehdä esimerkiksi Fourier-muunnos, on mahdollisesti järkevää käyttää FFTW:tä ja sen MPI-yhteensopivaa hajauttamismekanismia.

8.1.2.4 Projektioiden laskennan hajauttaminen

Prototyyppikokeilut ovat osoittaneet, että projektioiden laskeminen voi olla huomattavan hidasta varsinkin, käytettäessä hienostuneita interpolointimenetelmiä, kuten Nearest Neighbour -algoritmia. Tällöin on mahdollisesti tarpeen käyttää useita prosessoreita samanaikaisesti projektioiden laskentaan.

8.1.2.4.1 Yksi projektio ja monta prosessoria Koska kolmiulotteinen malli on suuri verrattuna yhden solmun muistiin on jopa mahdollista, ettei yhdellä prosessorilla ole tarpeeksi muistia projektionlaskemiseen. Tällöin kolmiulotteisen mallin hajauttaminen usean prosessorin muistiin on eräs mahdollisuus.

Projektion laskennan hajauttamisen toteuttaminen käsin ja tehokkaasti tulee kuitenkin huomattavatasti vaikeammaksi. Lisäksi verkkoliikenteen kasvu saattaa peittää saavutettavan hyödyn.

Kuten määrittelydokumentissa on todettu, on kohtuullista olettaa, että ainakin yhdellä prosessorilla on tarpeeksi työmuistia koko kolmiulotteisen mallin käsittelyyn.

8.1.2.4.2 Yksi projektio ja yksi prosessori Jos usealla prosessorilla on tarpeeksi muistia kolmiulotteisen mallin käsittelyyn on mahdollista laskea projektioita samanaikaisesti usealla eri prosessorilla monimutkaistamatta turhaan projektion laskentaan käytettyä algoritmia.

Tällöin tarvitaan ylimääräistä kirjanpitoa projektioiden keräämiseseen ja toimittamiseen eteenpäin vertailijaprosesseille.

8.2 Hajautuksen toteutus

Orientointijärjestelmän toiminta hajautetaan laskemalla projektioiden vertailuja elektronimikroskooppikuviin niin monella eri prosessilla, kuin käyttäjä haluaa. Myös projektioiden tuottaminen voi muodostua pullonkaulaksi laskennassa. Tämän takia järjestelmä mahdollistaa myös niin monen prosessin käyttämisen projektioiden tuottamiseen, kuin käyttäjä haluaa.

Orientointijärjestelmän prosessit voidaan käynnistää käyttäjän määrittelemille koneille kukin erikseen.

8.2.1 Orientointijärjestelmän hajautusstrategia

8.2.1.1 Vertailu

Orientointijärjestelmä on hajautettu käyttäen aliluvussa Yksi vertailu yhtä prosessoria kohti 8.1.2.2 käytettyä hajautusstrategiaa. Eli laskenta jaetaan jakamalla vertailualgoritmin syötetiedot eri prosessoreille ja suorittamalla vertailualgoritmia eri prosessoreilla toisistaan riippumatta.

Vertailun syötetiedot jaetaan käyttäen aliluvussa Yksi mikroskooppikuva ja yksi prosessori 8.1.2.2.1 esiteltyä lähestymistapaa. Yksi tietty elektronimikroskooppikuva on vain yhden prosessorin työmuistissa ja kyseinen prosessori laskee kaikki tähän elektronimikroskooppikuvaan liittyvät vertailut.

Koska vertailuja on suuri määrä, ei yksittäisen vertailun rinnakkaistaminen ole mielekästä. Yhden vertailun jakaminen usean prosessorin kesken tarkoittaisi ylimääräistä rasitetta verkolle. Toisaalta jokainen prosessori kykenee suorittamaan yhden vertailun kerrallaan itsenäisesti. Yksi vertailu on sopiva rinnakkaistamisen perusyksiköksi ja myös järkevin vaihtoehto.

8.2.1.2 Projektioiden tuottaminen

Projektioiden tuottaminen on myös hajautettu käyttäen aliluvussa Yksi projektio ja yksi prosessori 8.1.2.4.2 esitettyä mallia. Kullakin projektioita tuottavalla prosessorilla on työmuistissaan kaikki projektion tuottamiseen tarvittavat tiedot. Tämä asettaa vaatimuksen, että projektioiden tuottamiseen käytettävillä prosessoreilla tulee olla riittävästi työmuistia makromolekyylikompleksin kolmiulotteisen mallin säilyttämiseen ja käsittelemiseen.

Nämä hajautustrategiat johtavat järjestelmään, joka nojaa paljolti prosessoreiden laskentatehoon, eikä aseta epärealistisen suuria vaatimuksia prosessorien väliselle kommunikaatiolle.

Järjestelmän mitään osaa ei ole hajautettu jakamalla loogisesti yhtä tehtävää (esimerkiksi yhden kuvan Fourier-muunnosta) usealle prosessorille.

8.2.2 Laskennan etenemisen hallinta

Tieto virtaa järjestelmän läpi olennaisesti yhteen suuntaan (katso luku Tietovuokaavio 8.2.5.1 ja kuva Tietovuokaavio 8.1). Osien väliset kommunikaatiot on toteutettu synkronisilla viestinvälityksillä. Kommunikaatiot on lähes kaikissa kohdissa toteutettu tehokkuussyistä ei-blokkaavaa lähetystä käyttäen (Katso Ei-blokkaava 8.2.3 ja Synkroninen 8.2.4).

Laskennan etenemiseen vaikuttaa siis järjestelmän kaikkien osien nopeus. Järjestelmä on vain niin nopea kuin sen hitain osa.

Jos tulosten kerääjä käsittelee tuloksia hitaammin kuin vertailijat saavat tuloksia, vertailijat odottavat lupaa lähettää seuraava tulos tekemättä mitään. Jos taas vertailijat ovat hitaampia kuluttamaan projektioita kuin mitä projektioiden tuottajat voisivat tuottaa prosesseja, projektioiden tuottaja prosessit odottavat lupaa jatkaa tekemättä mitään.

Luonnollisesti seuraa myös,että jos projektioden tuottajat ovat hitaampia kuin järjestelmän muut osat, kaikki prosessit ovat ajoittain tekemättä mitää odottaessaan, että seuraava prosessi on valmis vastaanottamaan, tai edellinen prosessi on valmis lähettämään uutta tietoa käsiteltäväksi.

8.2.2.1 Projektioiden välittäjä

Projektioiden välittäjä on prosessi joka on tehty hallitsemaan tiedonvälitystä usealta projektioidentuottajaprosesseilta useille vertailijaprosesseille.

Projektioidenvälittäjäprosessi kuuntelee ei-blokkaavasti projektioita projektioiden tuottajilta, jos sen puskurissa on tilaa ylimääräisille projektioille. Samanaikaisesti prosessi lähettää projektioita vertailijaproseseille ja säilyttää projektiota puskurissaan siihen asti kunnes jokainen vertailijaprosessi on saanut kyseisen projektion. Käyttäjä voi halutessaan asettaa puskurin koon (katso Käyttöliittymä 9).

Projektionvälittäjäprosessin tarkoitus ei ole korjata huonosti tasapainotettua laskentaa, vaan sen tehtävänä on tasoittaa pieniä hetkittäisiä vaihteluita projektiontuottajaprosessien ja vertailijaprosessien suorituskyvyssä. Esimerkiksi järjestelmäprosessi tai satunnainen verkkoliikenne voi aiheuttaa hetkittäisiä vaihteluita eri vertailijoiden toimintaan. Jos minkäänlaista puskurointia ei olisi toteutettu, pienikin viivästys yhden vertailijan toiminnassa aiheuttaisi välittömästi sen, että muut joutuvat odottamaan myöhästynyttä prosessia.

Erillinen projektionvälittäjäprosessi poistaa myös ongelmat, joita olisi syntynyt, jos useat projektioidentuottajaprosessit olisivat välittäneet projektioita useille vertailijoille. Samoin projektionvälittäjäprosessi varmistaa sen, että projektiontuottajaprosessi voi jatkaa toimintaansa välittömästi saatuaan valmiiksi projektion joutumatta odottamaan vertailijaa, joka ei välttämättä ole valmis vastaanottamaan.

8.2.2.2 Tasapainotus

Koska orientointijärjestelmän mahdolliset ajoympäristöt poikkevat paljon toisistaan ja ohjelmistotuotantoryhmällä ei ole riittävää kokemusta dynaamisesta kuormanjakamisesta hajautetussa ympäristössä, suunnittelussa on päädytty yksinkertaiseen ratkaisuun. Käyttäjälle ja järjestelmän mukauttajalle tai jatkokehittäjälle tarjotaan joustavat mahdollisuudet toteuttaa juuri hänen järjestelmäänsä sopivat kuormantasausmekanismit ja säädöt.

Laskennan oikea tasapainotus on kuitenkin sen valmistumisen kannalta erittäin olennaista, joten olisi kohtuutonta vaatia, että käyttäjä osaisi konfiguroida laskennan oikein ensi yrittämällä. Klusterin rakenne ja ominaisuudet voivat myös muuttua laskennan aikana.

Orientointijärjestelmän vikasietoisuusmekanismia voidaan myös käyttää hajautuksen tasapainottamiseen, vaikka laskenta on ollut käynnissä pitkäänkin. Periaate on, että käyttäjä keskeyttää laskennan, muuttaa konfiguraatiotiedostoa ja käynnistää laskennan uudestaan. Laskenta käynnistyy lähes siitä kohdasta, missä se oli, kun se pysäytettiin.

Käyttäjä voi määritellä konfiguraatiotiedoston avulla järjestelmän kaikkille prosesseille erikseen prosessorin ja määritellä haluamansa määrän vertailija- ja projektioiden tuottaja -prosesseja. Käyttäjä voi myös määritellä vapaasti elektronimikroskooppikuvien jaon vertailijaprosessien kesken. (katso 9.2)

Ohjelmiston mukauttaja tai jatkokehittäjä on otettu huomioon siten, että ohjelma on rakenteeltaan modulaarinen ja erityyppisten laskentaprosessien välinen kommunikaatio on pyritty pitämään mahdollisimman yksinkertaisena ja yksiselitteisenä.

8.2.3 Ei-blokkaava lähetys

Ei-blokkaava lähetys tarkoittaa, että lähetysoperaatio ei varsinaisesti lähetä tietoa vaan käynnistää lähetyksen siten, että se tapahtuu samanaikaisesti rinnakkain muun ohjelman suorituksen kanssa. Näin prosessi voi suorittaa laskentaa samalla kun erillinen viestinvälitysprosessori siirtää tietoa muille prosesseille.

8.2.4 Synkroninen viestinvälitys

Synkroninen viestinvälitys tarkoittaa sitä, että lähettäjä ei varsinaisesti aloita lähetystä ennen kuin vastaanottaja on ilmoittanut valmiutensa julkaisemalla lähetyksen tunnistetietoihin täsmäävän vastaanotto-operaation.

8.2.5 Tiedonkulku orientointijärjestelmän sisällä

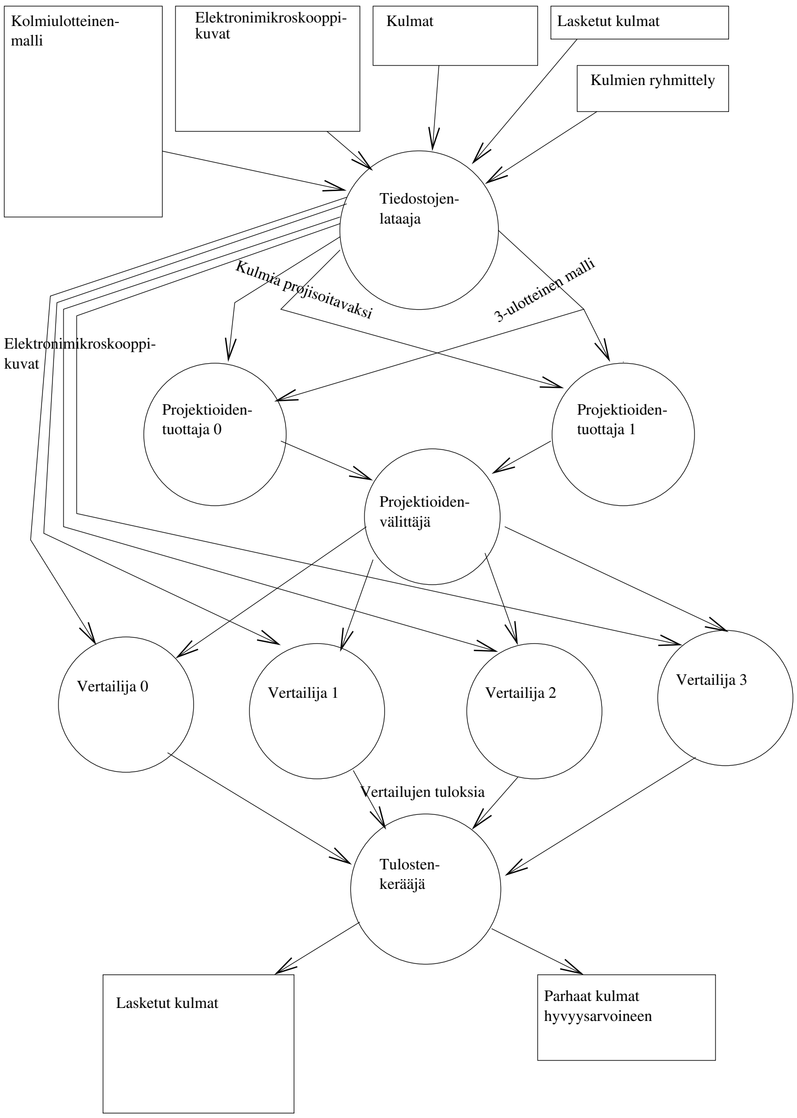
Synkroninen viestintä on ainoa mekanismi, joka säätelee tiedon kulkua orientointijärjestelmän osien välillä ja tieto kulkee osien välillä vain yhteen suuntaan. Orientointijärjestelmän tiedostonlataajaprosessi lukee kaikki syötetiedostot ja lähettää tietoa muille prosesseille, jotka mahdollisesti käsittelevät tietoa jotenkin ja välittävät sen eteenpäin.

Lopulta laskennan tulokset tiivistyvät tulostenkerääjäprosessiin joka kerää parhaat tulokset kaikista verratuista projektioista ja elektronimikroskooppikuvista ja kirjoittaa tuloksia ajoittain levylle käyttäjää varten.

Tulosten kerääjä kirjoittaa myös käsiteltyjä projektiokulmia valmiit projektiokulmat -tiedostoon sitä mukaa kun projektiota on verrattu kaikkiin elektronimikroskooppikuviin ja tulokset on käsitelty. Tulostiedostoa ja valmiit projektiokulmat -tiedostoa voidaan käyttää laskennan uudelleenkäynnistämiseen, jos laskenta on jostain syystä keskeytynyt ennen valmistumistaan (katso Vikasietoisuus 8.3).

8.2.5.1 Tietovuokaavio

Ohessa on järjestelmän tietovuokaavio. Kaavio on järjestetty siten, että tieto kulkee kaavion yläpäästä alapäähän. Kaaviosta on syytä huomata, että projektioiden



Kuva 8.1: Tietovuokaavio

tuottaja- ja vertailijaprosesseja voi olla eri määrä kuin kuvassa. Suurin osa kuvaan merkityistä tiedonkulkuväylistä on käytössä vain kun ohjelma käynnistyy. Laskennan aikana projektiokulmia siirtyy tiedostojen lataajalta projektioiden tuottajalle, projektioita siirtyy projektioiden välittäjältä vertailijoille ja vertailijoilta siirtyy tuloksia tulostenkerääjäprosessille.

8.3 Vikasietoisuus

Orietointijärjestelmän vikasietoisuus perustuu laskennan tilan aika-ajoittaista tallentamista kiintolevylle. Tallennusväli määritellään konfiguraatiotiedostossa. Järjestelmän suoritus saattaa keskeytyä mahdolliseen virhetilanteeseen[1] tai käyttäjän aiheuttamaan keskeytykseen (käyttäjä haluaa tarkentaa järjestelmän konfiguraatiota).

Tuloksien kerääjä tallentaa laskennan tuloksia muistipuskuriin pohjautuen hyvyysarvoihin. Konfiguraatiotiedostossa on määritelty kuinka monta tulosta järjestelmä tallentaa. Lisäksi tuloksien kerää seuraa saamiensa laskennan tuloksien projektio tunnuksia ja tallentaa projektiotunnuksen valmiisiin projektioihin, kun projektio on vertailtu kaikkiin elektronimikroskooppikuviin.

Kun tallennusväli kuluu umpeen järjestelmä luo uuden prosessin tallentamaan valmiit projektiot ja laskennan tulokset kiintolevylle. Järjestelmän suorituksen keskeytyessä menetetään korkeintaan edellisen tallennusvälin aikan tapahtunut laskenta.

Laskennan tila tallennetaan kahteen erilliseen tiedostoon. Toiseen tiedostoon tallennetaan valmiit projektiokulmien tunnukset ja toiseen tallennetaan varsinainen vertailuinformaatio hyvyysarvoineen. Tiedostot tellennetaan siten, että tiedoston nimi sisältää viisinumeroisen tunnisteen (tulostiedostot muotoa `result.XXXXX.bin` ja valmiit kulmat `completed_angles.XXXXX.bin`). Järjestelmä huolehtii, että tiedostoissa ei siirrytä uuteen versioon ennen kuin laskennan tulokset on kokonaan tallennettu kiintolevylle, eli uudelleenjatkamiskohta on viimeisimmän valmistuneen vertaillun kulmavälin kohdalla.

Järjestelmä ei itse poista vanhentuneita tiedostoversiota, vaan niiden poistaminen on ulkoisen roskienkeruutyökalun vastuulla (9.3.3).

8.3.1 Virhetilanteiden havaitseminen

Koska orientointijärjestelmä on hajautettu useaan eri prosessiin usealle eri koneelle, liittyvät virhetilanteet usein hajautukseen. Prosesseja saattaa kuolla tai koneita kaatuilla kesken laskentaa. Johtuen käytössä olevan MPI-toteutuksesta (mpich), joka ei tue prosessien dynaamista luomista tai pysäyttämistä eikä ole säieturvallinen, virhetilanteet aiheuttavat järjestelmän suorituksen päättymisen. Eli käynnistyksen jälkeen konfiguraatiota ei pystytä muuttamaan, ennen suorituksen päättymistä ilman huomattavasti kehittyneenpää prosessien sisäistä kirjanpitoa. Sisäinen kirjanpito aihauttaa kuitenkin merkittäviä ongelmia ja järjestelmän käynnistys uudella kon-

[1]järjestelmän sisällä kaikki virheet aiheuttavat suorituksen keskeytymisen.

figuraatiolla on kohtalaisen pieni operaatio. Mpich huolehtii järjestelmän alasajosta automaattisesti (tappaa kaikki prosessit). Järjestelmä on mahdollista konfiguroida uudestaan ja jatkaa suoritusta keskeytyneestä kohdasta. Lisää hajautusstrategiosta luvussa 8.1.2 ja MPI:istä luvussa 8.1.1.1.

8.3.2 Pullonkaulojen havaitseminen

Beowulf-klusterissa, jossa järjestelmää suoritetaan, on useita suorituskyvyltään vaihtelevia koneita. Lisäksi, jos orientointijärjestelmä on huonosti konfiguroitu, voidaan kuormittaa joitakin koneita huomattavasti enemmän kuin muita koneita. Tämän vuoksi laskennan etenemistä pitää kyetä valvomaan, jotta kyettäisiin tarvittaessa tasapainottamaan laskentaa. Tasapainotus tapahtuu käytännössä siten, että käyttäjä keskeyttää järjestelmän suorituksen, tarkentaa järjestelmän konfiguraatiota ja käynnistää keskeytyneestä kohdasta laskennan uudestaan. Laskennan uudelleenkonfiguroinnilla pystytään vaikuttamaan vertailuoperaatioden (eli kuinka monta elektronimikroskooppikuvaa prosessilla on), millä koneella prosessit suoritetaan (sijoitetaan hitaammalle koneelle kevyempi kuorma), sekä prosessien määrään (voidaan lisätä vaikkapa vertailiprosessien määrää).

8.3.3 Laskennan valvonta

Orientointijärjestelmässä laskennan valvonta perustuu järjestelmän päätteelle tulostamaan informaation laskennan tilasta. Järjestelmä seuraa vertailijaprosessien vartailtavaa projektiota, projektioden välittäjän lähettämiä ja vastaanottamia projektioita, projektioden tuottajien tuottamia projektioita ja tuloksien kerääjän vastaanottamia tuloksia, sekä valmiiksi laskettuja projektioita. Käyttäjä kynenee informaation perusteella havaitsemaan, jos jokin vertailija tai projektioiden tuottaja on muita hitaampi. Jos laskenta on epätasapainossa voi käyttäjä keskeyttää laskennan, ja käynnistää sen uudelleen tasapainotetulla konfiguraatiolla.

Luku 9

Käyttöliittymä

9.1 Yleiskuvaus

Orientointijärjestelmä on melko tyypillinen suodatin. Järjestelmä lukee ensin syötteensä syötetiedostoista, suorittaa raskasta hajautettua laskentaa, ja lopulta kirjoittaa laskennan tulokset tulostiedostoihin. Tästä syystä järjestelmään ei ole tarvetta toteuttaa interaktiivista käyttöliittymää, jolla laskentaan voisi vaikuttaa kesken ohjelman suorituksen.

Edellämainituin perustein järjestelmän käyttöliittymä on päätetty toteuttaa konfiguraatiotiedostopohjaisena. Järjestelmä lukee siis laskentansa alkuvaiheessa komentoriviparametrin avulla nimetyn konfiguraatiotiedoston, jonka sisältämien komentojen perusteilla laskentaa jatketaan. Konfiguraatiotiedoston lisäksi järjestelmän toimintaa ohjataan erilaisin syötetiedostoin sekä ajonaikaisin näppäimistökomennoin. Laskennan tulokset talletetaan tulostiedostoihin.

Tässä luvussa esitellään konfiguraatiotiedoston sekä syöte- ja tulostiedostojen rakenne, komentoriviparametrit ja rinnakkaistamisen seurantaan käytettävät mpich:n monitorointityökalut.

9.2 Konfiguraatiotiedosto

9.2.1 Yleiskuvaus

Orientaatiojärjestelmän konfiguraatiotiedosto on rakenteeltaan erittäin joustava, sillä sen toteuttamiseen on käytetty laajennettua sulautettua Perl-kieltä. Konfiguraatiotiedosto on siis Perl-ohjelma, joka suoritetaan C-kieliseen pääohjelmaan upotettua Perl-tulkkia käyttäen. Perl-ohjelmassa on käytettävissä tavallisten Perlin ominaisuuksien lisäksi C-kielellä toteutettuja laajennusfunktioita, joiden avulla Perl-ohjelma pystyy suoraan muuttamaan C-kielisen pääohjelman toimintaa ohjaavien globaaleiden muuttujien arvoja.

C-kielisten laajennosten liittäminen Perliin toteutetaan käyttäen SWIG-ohjelmistoa (Simplified Wrapper and Interface Generator) [8]. Käytännössä Perlin laajennokset ovat vain rajapinta Konfiguraatiomoduulin C-kieliseen toteutukseen. Perl-tulkin upottaminen C-kieliseen pääohjelmaan hoidetaan puolestaan Perlin standardijakeluun kuuluvien upotusmekanismien avulla [1]. Koska C:n ja Perl:n välisen rajapinnan toteutus noudattaa tarkoin SWIG:n ja Perl:n dokumentaatiossa esiteltyjä yleisiä käytäntöjä, ei siihen tässä dokumentissa tämän tarkemmin puututa.

9.2.2 Konfiguraatiotiedoston rakenne

Konfiguraatiotiedosto on vapaamuotoinen Perl-ohjelma, joten sen rakenne on pääpiirteittäin sama kuin millä tahansa Perl-ohjelmalla. Poikkeuksen tavanomaiseen Perl:iin muodostavat C-kielellä toteutetut SWIG:llä Perl:iin liitetyt laajennokset. Tässä luvussa esitellään näiden laajennosten syntaksi ja semantiikka; Perl-ohjelmista ja niiden semantiikasta löytyy lisätietoa lähteistä [6, 5].

9.2.2.1 Laajennokset Perl-kieleen

Perl-kieleen on liitetty seuraavat C-kielellä toteutetut laajennokset, joiden avulla C-kielellä toteutetun pääohjelman sisäisiä tietorakenteita voidaan muuttaa sulautetusta Perl:stä käsin.

9.2.2.1.1 Laajennusfunktiot

```
define_process MACHINE,PROCESS_TYPE
set_pizza_count PROCESS_NUMBER,PIZZA_COUNT
```

Yllä luetellut funktiot ovat rajapintoja vastaavan nimisiin C-kielisiin Konfiguraatiomoduulin funktioihin, joiden dokumentaatio on luvussa 5.2.6.3.2.

9.2.2.1.2 Rajapinnan läpi näkyvät globaalit muuttujat

```
int projection_buffer_size=16;
char *projection_angles_file_name=NULL;
char *completed_projection_angles_file_name=NULL;
char *projection_angle_groups_file_name=NULL;
int max_projection_angle_group_size=128;
int max_best_results=10;
int state_saving_interval=60*60/4;
char *results_file_name=NULL;
char *log_file_name=NULL;
char **pizza_file_names = NULL;
char *model_file_name=NULL;
int projection_image_type=IMAGE_REAL;
int projection_image_x=256;
int max_results_from_comparison=16;
int remove_water_r1=100;
int remove_water_r2=110;
int comparison_steps_2d=100;
```

Yllämainittuihin C-kielellä toteutetun konfiguraatiomoduulin muuttujiin voi asettaa arvon aivan samaan tapaan kuin tavallisiin Perl-kielen sisäisiin muuttujiin. Muuttujiin, joiden tyyppi on `int`, voi siis asettaa Perl-kielen kokonaislukua esittävän skalaarin ja `char *`-tyyppisiin muuttujiin voi sijoittaa Perl-kielen merkkijonoskalaarin. Hieman hankalampikäyttöinen on `char **`-tyyppinen muuttuja `pizza_file_names`, johon pitää sijoittaa referenssi Perl-kieliseen merkkijonoskalaareja sisältävään listaan eikä listaa itseään.

Edellä mainittujen muuttujien lisäksi Perl-moduuliin näkyvät seuraavat vakiot:

```
int FILE_LOADER = 0;
```

```
int PROJECTION_RELAY = 1;
int RESULT_COLLECTOR = 2;
int PROJECTION_PRODUCER = 3;
int COMPARATOR = 4;
```

Näitä vakioita käytetään `define_process`-funktiokutsun parametreina ilmoittamaan, minkä tyyppistä prosessia ollaan luomassa.

Käytännönläheiset ohjeet konfiguraatiotiedoston käytöstä esimerkkeineen löytyvät erillisestä Käyttöohjeesta.

9.3 Käyttöliittymän muut osat

9.3.1 Komentoriviparametrit

Orientointijärjestelmä käynnistetään komennon `mpirun` avulla seuraavasti:

```
mpirun <mpirunin parametrit> beklu [konfiguraatiotiedoston nimi]
```

Ohjelman `mpirun` toiminta ja parametrit on kuvattu lähteessä [19].

Vaihtoehtoisesti ohjelman voi käynnistää `-m`-parametrin avulla suoraan ilman `mpirun`:ia seuraavasti:

```
en muista miten
```

Tällöin beklu käynnistää prosessit konfiguraatiotiedostossa määritellyillä koneilla.

9.3.2 Syötetiedostot

Orientointijärjestelmä lukee konfiguraatiotiedoston lisäksi neljää eri tyyppiä olevia syötetiedostoja: MRC-formaatissa olevia elektronimikroskooppikuvia, MRC-formaatissa olevan kolmiulotteisen mallin, projektiokulmalistatiedoston ja projektiokulmaryhmätiedoston. MRC-tiedostojen rakenne on määritelty spesifikaatiossa [2], joten siihen ei tässä puututa.

9.3.2.1 Projektiokulmalistatiedosto

Projektiokulmalistatiedoston avulla järjestelmälle kerrotaan ne kulmat, joista sen halutaan laskevan kolmiulotteisen mallin projektiot. Tiedosto koostuu peräkkäin talletetuista binäärimuotoisista tietueista, joista kukin sisältää `projection_req` tyyppisen tietueen.

9.3.2.2 Projektiokulmaryhmätiedosto

Projektiokulmaryhmätiedostossa määritellään projektiokulmalistatiedoston sisältämän kulmalistan jako ryhmiin, jota tiedostonlataaja käyttää välittäessään projektiosuuntia projektioiden tuottajien laskettavaksi. Kulmaryhmätiedosto koostuu jonosta binäärimuodossa talletettuja `int`-tyyppisiä arvoja, jotka ilmoittavat projektiokulmalistatiedoston ryhmien alku- ja loppupisteet. Jos siis projektiokulmaryhmätiedoston sisältö on $i_0, i_1, \ldots, i_n$, muodostuu k:nnes B_k kulmaryhmä projektiokulmatiedoston kulmista $i_{k-1}, i_{k-1}+1, \ldots, i_k - 1$ $(1 \leq k \leq n)$. Erityisesti oletetaan, että $i_0 = 0$ ja $i_n = m$, missä m on projektiokulmalistatiedoston tietueiden lukumäärä.

Kun projektioiden tuottaja tarvitsee lisää projisointikulmia, välittää tiedostojen lataaja sille jonkin kulmaryhmän B_k, jota se ei ole vielä välittänyt millekään projektioiden tuottajalle. Kulmaryhmän saanut projektioiden tuottaja laskee sitten projektiot kaikista ryhmän kulmista, jonka jälkeen tiedostojen lataaja välittää sille uuden käsittelemättömän ryhmän.

9.3.3 Tulostiedostot

Orientointijärjestelmällä on kaksi erillistä tulostiedostoa, joista toinen on varsinainen käyttäjälle näkyvä laskennan tulokset sisältävä tiedosto ja toinen laskennan etenemisen seurantaan käytetty valmiit projektiokulmat sisältävä tiedosto.

9.3.3.1 Varsinainen tulostiedosto

Tämä tulostiedosto sisältää laskennan varsinaiset tulokset eli elektronimikroskooppikuville löydetyt orientaatiot hyvyysinformaatioineen. Tulostiedosto on binäärimuotoinen ja koostuu kaksiulotteisesta taulukosta `result`-tyyppisiä tulostietorakenteita. Taulukossa on rivi jokaista orientoitavaa elektronimikroskooppikuvaa kohden. Kullakin rivillä on `max_best_results` saraketta, joihin on tallennettu riviä vastaavan elektronimikroskooppikuvan parhaita löytyneitä orientaatiovaihtoehtoja kuvaavat tulostietorakenteet hyvyysinformaation mukaan järjestettynä. Tulostaulukon rivit on järjestetty siten, että rivillä `i` on tunnuksella `i` varustetun elektronimikroskooppikuvan tiedot.

Orientaatiojärjestelmän vikasietoisuuden parantamiseksi tulostiedostosta pidetään aina levyllä useita kopioita. Näin kaikki laskentatulokset eivät katoa, jos järjestelmä sattuu kaatumaan kesken päivitysoperaation. Eri tulostiedostoversioiden erottamiseksi niiden nimet ovat muotoa `*.DDDDD.bin`, missä `DDDDD` on viisinumeroinen desimaaliluku. Päivitykset luovat uuden tulostiedoston, jonka nimessä edellämainittua versionumeroa on kasvatettu yhdellä. Päivityksiä tehdään `update_interval` sekunnin välein. Järjestelmä ei poista automaattisesti vanhoja versioita, vaan niiden siivoaminen jää erillisen roskienkeruutyökalun vastuulle.

9.3.3.2 Valmiit projektiokulmat sisältävä tiedosto

Järjestelmän vikasietoisuus perustuu pääasiassa siihen, että keskeytynyt laskenta pystytään käynnistämään uudestaan ilman että saavutetut osittaiset tulokset katoavat. Valmiit projektiokulmat sisältävässä tiedostossa pidetään kirjaa siitä, mitkä projektiokulmat on jo kokonaan käsitelty. Projektiokulman esiintyminen valmiit projektiokulmat -tiedostossa ei välttämättä kuitenkaan tarkoita, että sitä vastaavat laskentatulokset olisivat päivittyneet varsinaiseen tulostiedostoon; tarkemmin asiasta luvussa 8.3.

Valmiit projektiokulmat talletetaan binäärimuodossa lisäämällä tiedoston loppuun projektiokulmien `int`-tyyppisiä tunnisteita sitä mukaan kun projektioita valmistuu. Vikasietoisuuden parantamiseksi tiedosto jaetaan kuitenkin useaan osaan siten, että kunkin varsinaisen tulostiedoston päivitysoperaation yhteydessä perustetaan uusi valmiit projektiokulmat -tiedosto, jonka loppuun valmistuneiden projektiokulmien tunnisteita seuraavaan tulostiedoston päivitysoperaatioon asti lisätään. Tiedostoversiot nimetään samaan tapaan kuin varsinaisen tulostiedoston tapauksessa. Tiedoston päivitystavasta johtuen kaikki valmistuneet projektiokulmat valmistumisjärjestyksessä sisältävä tulostiedosto saadaan katenoimalla versionumeroidut osatiedostot versionumeroiden mukaisessa järjestyksessä peräkkäin.

9.3.4 Ajonaikaiset näppäimistökomennot

Orientointijärjestelmää voi ohjata suorituksen aikana Unix:n keskeytysnäppäinkomennolla (tavallisesti `^C`). Tämä komento aiheuttaa järjestelmän kaikkien prosessien keskeytyksen.

9.3.5 mpich:n monitorointityökalut

Orientointijärjestelmän rinnakkaisen laskennan seurantaan ja profilointiin käytetään mpich:n monitorointityökaluja upshot, nupshot ja Jumpshot. Näillä ohjelmilla visualisoidaan lokitiedostoja, joita tuotetaan ohjelman laskennan kuluessa. Lisäksi ohjelman toimintaa seurataan mahdollisesti prototyyppiasteella olevien MPE-kirjastoon kuuluvien animaattoreiden avulla. Näistä työkaluista löytyy lisätietoa lähteistä [20, 18, 21].

Luku 10

Rajoitukset toteutukselle

10.1 Noudatettavat standardit

Toteutuksen C-kielisissä osissa noudatetaan ANSI C -standardia [7] ja Perl-kielisissä osissa käytetään Perlin versiota 5.005 [6]. MPI-rinnakkaisuudenhallintakirjastosta käytetään versiota 1.2.

10.2 Ohjelmointikielet ja käyttöjärjestelmät

Toteutuksessa käytetään pääasiassa C-kieltä. Konfiguraatiotiedostossa käytetään Perl-kieltä. Järjestelmä toteutetaan Mandrake Linux 7.0 -ympäristössä, mutta se toimii muissakin sellaisissa ympäristöissä joihin järjestelmän tarvitsemat kirjastot voidaan asentaa ja joissa on ANSI C -kääntäjä.

10.3 Muut tarvittavat apuohjelmistot

Järjestelmän toteuttamisessa käytetään apuna ohjelmakoodieditoria, kuvankäsittelyohjelmia ja mahdollisesti muitakin apuohjelmistoja. Näitä ei kuitenkaan kiinnitetä tässä, vaan kukin projektiryhmän jäsen valitsee käyttämänsä työkalut tarpeen ja mieltymysten mukaan.

10.4 Arvio toteutuksen koosta ja ohjelmointityyli

Toteutuksen arvioidaan sisältävän C-kielistä ohjelmakoodia n. 5000 - 10000 riviä. Tämä perustuu arvioon, jossa jokainen aliohjelma sisältäisi keskimäärin 50 riviä koodia ja aliohjelmien määrä olisi n. 150.

Ohjelmointityylinä noudatetaan kirjassa The C programming language [27] esitettyä tyyliä sillä erotuksella, että sisennyksen syvyytenä on vain kaksi merkkiä. Kielenä kommentoinnissa sekä muuttujien nimeämisessä käytetään englannin kieltä.

Jokaisen kooditiedoston alkuun kirjoitetaan kommentti, jossa kerrotaan lyhyesti, millaisia rutiineja tiedostossa toteutetaan ja mitä ne tekevät. Jokaisen aliohjelman alkuun kommentoidaan tarkemmin aliohjelman nimi, tehtävä, parametrit, paluuarvo ja tarvittaessa selitys aliohjelman toiminnasta.

Luku 11

Testaus

11.1 Käytettävät testausmenetelmät

Järjestelmää testataan sekä white-box että black-box testausmenetelmillä. Lisäksi järjestelmän kriittisiä osa-alueita (esim. vikasietoisuus ja tulosten oikeellisuus) testataan tarkemmin. Testauksen helpottamiseksi luodaan joitakin testaustyökaluja, joilla järjestelmän toiminnan seuraamista voidaan helpottaa ja laskennan tuloksia tutkia. Myöskin käyttöohjeet ja muu dokumentaatio tullaan testaamaan ennen järjestelmän valmistumista.

11.1.1 White-box testaus

White-box testausmenetelmällä testataan yksittäisten algoritmien toimintaa niin suunnittelu kuin toteutusvaiheessa, tarkoituksena on löytää logiikkavirheitä koodin harvemmin suoritettavista osista. Algoritmista rakennetaan kaavio, josta voidaan testata algoritmin toimintaa kaikkilla suorituspoluilla.

11.1.2 Black-box testaus

Black-box testausmenetelmällä testataan järjestelmän toimintaa aineistolla, josta saatavat tulokset ovat jo valmiiksi tiedossa, jolloin voidaan testata kaikkien algoritmien yhteistoimintaa ja tulosten oikeellisuutta. Tässä testausmenetelmässä on tärkeää että oikeat tulokset ovat etukäteen tiedossa, jotta voidaan tutkia toimiiko järjestelmä oikein.Johtuen elektronimikroskooppikuvissa olevasta häiriöstä testaus pitää aloittaa testitapauksilla, joissa kohinan osuus on mahdollisimman pieni. Tämä on välttämätöntä koska aluksi järjestelmää on testattava aineistolla, joka ei voi aiheuttaa järjestelmän toiminnassa kummallisuuksia, vaan kaikki virheet voidaan jäljittää itse järjestelmään.

11.2 Järjestelmän testaus

11.2.1 Tulosten oikeellisuus

Tulosten oikeellisuus testataan black-box menetelmällä aluksi keinotekoisella aineistolla, josta saatavat tulokset ovat ennalta selvillä ja tämän jälkeen voidaan siirtyä testaamaan järjestelmää oikeilla syötteillä, edelleen saatavat tulokset pitää olla etukäteen tiedossa, jotta niitä voidaan verrata järjestelmän laskemiin.

11.2.2 Vikasietoisuus

Järjestelmän tulee toipua mahdollisemman monesta virhetilanteesta, sillä järjestelmän suoritusaika on pitkä eikä sitä pitäisi kasvattaa turhan takia. Joten järjestelmä pitää yllä lokitiedostoa, johon kirjataan kaikki tapahtumat ja järjestelmä voidaan käynnistää tämän tiedoston perusteella uudestaan, jolloin laskentaa voidaan jatkaa siitä mihin jäätiin. Lokitiedoston toiminta testataan yksinkertaisesti sammuttamalla järjestelmä ja käynnistämällä uudestaan, jolloin laskennan tulisi jatkua siitä mihin se ennen järjestelmän alasajoa jäi. Järjestelmän ollessa suorituksessa poistetaan yksi laskentayksikkö klusterista. Poistettava laskentayksikkö ei voi olla Master. Järjestelmä ei saa seota tässä tilanteessa vaan sen pitää toipua tapahtuneesta jajatkaa toimintaa normaalisti ja kirjata tapahtunut lokitiedostoon. Laskentayksikkö poistetaan sammuttamalla se. Kun järjestelmän on todettu selvinneen tästä, poistetaan klusterista toinenkin laskentayksikkö ja tutkitaan selviääkö järjestelmä tästä. Uuden laskentayksikön ottaminen mukaan järjestelmän toimintaa vaatii järjestelmän uudelleenkäynnistämisen, jolloin järjestelmä jatkaa laskentaa siitä mihin se lokitiedoston mukaan jäi ennen sammuttamistaan, nyt järjestelmä ottaa mukaan myös lisätyn laskentayksikön.

11.2.3 Suorituskyky

Järjestelmää suoritetaan oikealla aineistolla, jossa on noin 10 elektronimikroskooppi kuvaa, jolloin voidaan varmistaa, ettei järjestelmän suorittaminen tässä tapauksessa vie suhteettoman kauan, sillä oikean kokoinen aineisto olisi noin 1000 elektronimikroskooppikuvaa ja aikaa noin viikko. Suorituskykyä voidaan parantaa vähentämällä laskettavien projektioiden määrää, jolloin saatujen tulosten tarkkuus laskee.

11.3 Testaustyökalut

Testityökalut ovat pieniä apuohjelmia testaukseen, joilla testeissä saatuja tuloksien tutkimista voidaan helpottaa ja nopeuttaa. Useita erilaisia ohjelmia on ehdotettu ja tulosten visualisoija toteutetaan orientointijärjestelmän tulosten tutkimisen helpottamiseksi. Apuohjelmat toteutetaan ennen järjestelmän testauksen alaoittamista.

11.3.1 Tulosten visualisoija

Tulosten visualisoija apuohjelma, joka helpottaa orientointijärjestelmän tulosten tutkimista. Visualisoija piirtää näytölle pallon ja merkitsee siihen parhaat orientaatiot halutulle elektronimikroskooppikuvalle. Lisäksi ohjelma kertoo jokaisen orientaatioon hyvyyden. Ohjelma on suunnitteilla eikä sen käyttöliittymää ole vielä määritelty

Liite A

Suorituskykylaskelmia

Tässä osassa on laskelmia, jotka on tehty Microsoft Excel -ohjelmalla. Laskelmat tehtiin alunperin vaatimusdokumenttia varten, mutta niitä kehitettiin suunnittelun aikana edelleen samalla, kun näkemys ohjelman toteutuksesta selkiintyi.

Parametrit				
mallin_sivun_pituus_pixel	256	256	256	256
asteverkon_askelia_kpl	500	500	500	500
pikselin_koko_B	4	4	4	4
mikroskooppikuvia_kpl	1500	1000	500	100
kuvan_koko_KB	256,00	256,00	256,00	256,00
cpu_lkm	10	10	10	10
cpu_nopeus_mhz	700,00	700,00	700,00	700,00
alkeisoperaatio_rotaatio	**10**	**50**	**10**	**5**
alkeisoperaatio_projektio	**128**	**256**	**128**	**64**
alkeisoperaatio_vertailu	**100**	**50**	**25**	**10**
alkeisoperaatio_vertailu_ri	**5000**	**2500**	**1000**	**500**
Tulokset				
ojektioiden_laskeminen_1_solmussa_vrk	8,876833862	17,75366772	8,876833862	4,438416931
otaatioiden_laskeminen_1_solmussa_vrk	1,354497354	6,772486772	1,354497354	0,677248677
verkon_kapasiteetti_GB_h	36,00	36,00	36,00	36,00
dataa_rotatatoimattomat_proketiot_GB	62,50	62,50	62,50	62,50
siirto_rotatoimattomat_projektiot_h	1,74	1,74	1,74	1,74
dataa_rotatatoidut_projektiot_GB	31250,00	31250,00	31250,00	31250,00
rotatoitujen_projektioiden_siirto_vrk	36,17	36,17	36,17	36,17
Rotaatioittain vertailu				
aikaa_kaikki_vertailut_s	175542857,1	58514285,71	14628571,43	1170285,714
aikaa_kaikki_vertailut_vrk	2031,75	677,25	169,31	13,54
aikaa_yhteensä_vrk	2041,98	701,77	179,54	18,66
rotaatioinvariantti vertailu				
vertailua_minuutissa_ri_kpl_min	1281,738281	2563,476563	6408,691406	12817,38281
aikaa_kaikki_vertailut_ri_s	17554285,71	5851428,571	1170285,714	117028,5714
aikaa_kaikki_vertailut_ri_vrk	203,17	67,72	13,54	1,35
aikaa_yhteensä_ri_vrk	213,41	92,25	23,78	6,47

Parametrit	
mallin_sivun_pituus_pixel	256
asteverkon_askelia_kpl	500
pikselin_koko_B	4
mikroskooppikuvia_kpl	1000
kuvan_koko_B	262144
kuvan_koko_KB	256,00
cpu_lkm	10
cpu_nopeus_tcks_s	7,00E+08
cpu_nopeus_mhz	700,00
alkeisoperaatio_rotaatio_tcks_pix	10
alkeisoperaatio_projektio_tcks_pix	50
alkeisoperaatio_vertailu_tcks_pix	50
alkeisoperaatio_vertailu_ri_tcks_pix	2500
Tulokset	
ojektioiden_laskeminen_1_solmussa_vrk	3,46751323
otaatioiden_laskeminen_1_solmussa_vrk	1,35449735
verkon_kapasiteetti_B_s	10485760
verkon_kapasiteetti_GB_h	36,00
dataa_rotatoimattomat_projektiot_B	6,5536E+10
dataa_rotatatoimattomat_proketiot_GB	62,50
siirto_rotatoimattomat_projektiot_s	6250
siirto_rotatoimattomat_projektiot_h	1,74
dataa_rotatoidut_projektiot_B	3,2768E+13
dataa_rotatatoidut_projektiot_GB	31250,00
rotatoitujen_projektioiden_siirto_s	3125000
rotatoitujen_projektioiden_siirto_vrk	36,17
Rotaatioittain vertailu	
operaatio_vertailu_tcks	3276800
vertailuja_yhteensä_kpl	1,25E+11
vertailua_minuutissa_kpl_min	128173,828
aikaa_kaikki_vertailut_s	58514285,7
aikaa_kaikki_vertailut_vrk	677,25
aikaa_yhteensä_vrk	682,07
rotaatioinvariantti vertailu	
operaatio_vertailu_ri_tcks	163840000
operaatio_vertailu_ri_s	0,23405714
vertailuja_yhteensä_ri_kpl	250000000
vertailua_minuutissa_ri_kpl_min	2563,47656
aikaa_kaikki_vertailut_ri_s	5851428,57
aikaa_kaikki_vertailut_ri_vrk	67,72
aikaa_yhteensä_ri_vrk	72,55

Parametrit				
mallin_sivun_pituus_pixel	256	256	256	256
asteverkon_askelia_kpl	**1000**	**750**	**500**	**250**
pikselin_koko_B	4	4	4	4
mikroskooppikuvia_kpl	1000	1000	1000	1000
kuvan_koko_KB	256,00	256,00	256,00	256,00
cpu_lkm	10	10	10	10
cpu_nopeus_mhz	700,00	700,00	700,00	700,00
alkeisoperaatio_rotaatio_tcks_pix	10	10	10	10
alkeisoperaatio_projektio_tcks_pix	50	50	50	50
alkeisoperaatio_vertailu_tcks_pix	50	50	50	50
alkeisoperaatio_vertailu_ri_tcks_pix	2500	2500	2500	2500
Tulokset				
ojektioiden_laskeminen_1_solmussa_vrk	13,87005291	7,801904762	3,467513228	0,866878307
otaatioiden_laskeminen_1_solmussa_vrk	10,83597884	4,571428571	1,354497354	0,169312169
verkon_kapasiteetti_GB_h	36,00	36,00	36,00	36,00
dataa_rotatatoimattomat_proketiot_GB	250,00	140,63	62,50	15,63
siirto_rotatoimattomat_projektiot_h	6,94	3,91	1,74	0,43
dataa_rotatatoidut_projektiot_GB	250000,00	105468,75	31250,00	3906,25
rotatoitujen_projektioiden_siirto_vrk	289,35	122,07	36,17	4,52
Rotaatioittain vertailu				
vertailuja_yhteensä_kpl	1E+12	4,21875E+11	1,25E+11	15625000000
aikaa_kaikki_vertailut_vrk	5417,99	2285,71	677,25	84,66
aikaa_yhteensä_vrk	5442,70	2298,09	682,07	85,69
rotaatioinvariantti vertailu				
vertailuja_yhteensä_ri_kpl	1000000000	562500000	250000000	62500000
aikaa_kaikki_vertailut_ri_vrk	270,90	152,38	67,72	16,93
aikaa_yhteensä_ri_vrk	295,61	164,75	72,55	17,97

Parametrit				
mallin_sivun_pituus_pixel	256	256	256	256
asteverkon_askelia_kpl	500	500	500	500
pikselin_koko_B	4	4	4	4
mikroskooppikuvia_kpl	**1500**	**1000**	**500**	**100**
kuvan_koko_KB	256,00	256,00	256,00	256,00
cpu_lkm	10	10	10	10
cpu_nopeus_mhz	700,00	700,00	700,00	700,00
alkeisoperaatio_rotaatio_tcks_pix	10	10	10	10
alkeisoperaatio_projektio_tcks_pix	50	50	50	50
alkeisoperaatio_vertailu_tcks_pix	50	50	50	50
lkeisoperaatio_vertailu_ri_tcks_pix	2500	2500	2500	2500
Rotaatioittain vertailu				
vertailuja_yhteensä_kpl	1,875E+11	1,25E+11	62500000000	12500000000
aikaa_kaikki_vertailut_s	87771428,6	58514285,71	29257142,86	5851428,571
aikaa_kaikki_vertailut_vrk	1015,87	677,25	338,62	67,72
aikaa_yhteensä_vrk	**1020,70**	**682,07**	**343,45**	**72,55**
rotaatioinvariantti vertailu				
vertailuja_yhteensä_ri_kpl	375000000	250000000	125000000	25000000
aikaa_kaikki_vertailut_ri_s	8777142,86	5851428,571	2925714,286	585142,8571
aikaa_kaikki_vertailut_ri_vrk	101,59	67,72	33,86	6,77
aikaa_yhteensä_ri_vrk	**106,41**	**72,55**	**38,68**	**11,59**

Parametrit	
kuvan_koko_B	=mallin_sivun_pituus_pixel^2*pikselin_koko_B
kuvan_koko_KB	=kuvan_koko_B/2^10
cpu_nopeus_mhz	=cpu_nopeus_tcks_s/10^6
Tulokset	
rojektioiden_laskeminen_1_solmussa_vrk	=mallin_sivun_pituus_pixel^3*alkeisoperaatio_projektio_tcks_pix/cpu_nopeus_tcks_s*asteverkon_askelia_kpl^2/3600/24
rotaatioiden_laskeminen_1_solmussa_vrk	=mallin_sivun_pituus_pixel^2*alkeisoperaatio_rotaatio_tcks_pix/cpu_nopeus_tcks_s*asteverkon_askelia_kpl*asteverkon_askelia_kpl^2/3600/24
verkon_kapasiteetti_B_s	=10*2^20
verkon_kapasiteetti_GB_h	=verkon_kapasiteetti_B_s*3600/(2^20*1000)
dataa_rotatoimattomat_projektiot_B	=kuvan_koko_B*asteverkon_askelia_kpl^2
dataa_rotatatoimattomat_proketiot_GB	=dataa_rotatoimattomat_projektiot_B/2^20/1000
siirto_rotatoimattomat_projektiot_s	=dataa_rotatoimattomat_projektiot_B/verkon_kapasiteetti_B_s
siirto_rotatoimattomat_projektiot_h	=siirto_rotatoimattomat_projektiot_s/3600
dataa_rotatoidut_projektiot_B	=kuvan_koko_B*asteverkon_askelia_kpl^3
dataa_rotatatoidut_projektiot_GB	=dataa_rotatoidut_projektiot_B/2^20/1000
rotatoitujen_projektioiden_siirto_s	=dataa_rotatoidut_projektiot_B/verkon_kapasiteetti_B_s
rotatoitujen_projektioiden_siirto_vrk	=rotatoitujen_projektioiden_siirto_s/3600/24
Rotaatioittain vertailu	
operaatio_vertailu_tcks	=mallin_sivun_pituus_pixel^2*alkeisoperaatio_vertailu_tcks_pix
vertailuja_yhteensä_kpl	=mikroskooppikuvia_kpl*asteverkon_askelia_kpl^3
vertailua_minuutissa_kpl_min	=cpu_lkm*cpu_nopeus_tcks_s/operaatio_vertailu_tcks*60
aikaa_kaikki_vertailut_s	=vertailuja_yhteensä_kpl*operaatio_vertailu_tcks/cpu_nopeus_tcks_s/cpu_lkm
aikaa_kaikki_vertailut_vrk	=aikaa_kaikki_vertailut_s/3600/24
aikaa_yhteensä_vrk	=aikaa_kaikki_vertailut_vrk+rotaatioiden_laskeminen_1_solmussa_vrk+projektioiden_laskeminen_1_solmussa_vrk
rotaatioinvariantti vertailu	
operaatio_vertailu_ri_tcks	=mallin_sivun_pituus_pixel^2*alkeisoperaatio_vertailu_ri_tcks_pix
operaatio_vertailu_ri_s	=operaatio_vertailu_ri_tcks/cpu_nopeus_tcks_s
vertailuja_yhteensä_ri_kpl	=mikroskooppikuvia_kpl*asteverkon_askelia_kpl^2
vertailua_minuutissa_ri_kpl_min	=cpu_lkm*cpu_nopeus_tcks_s/operaatio_vertailu_ri_tcks*60
aikaa_kaikki_vertailut_ri_s	=vertailuja_yhteensä_ri_kpl*operaatio_vertailu_ri_tcks/cpu_nopeus_tcks_s/cpu_lkm
aikaa_kaikki_vertailut_ri_vrk	=aikaa_kaikki_vertailut_ri_s/3600/24
aikaa_yhteensä_ri_vrk	=aikaa_kaikki_vertailut_ri_vrk+rotaatioiden_laskeminen_1_solmussa_vrk+projektioiden_laskeminen_1_solmussa_vrk