TIETOVIRTA OHJELMOINTIMALLINA JA SEN SOVELTUVUUS REAALIAIKAISEEN JATKUVAAN MITTAUKSEEN

Lappeenrannan–Lahden teknillinen yliopisto LUT

Tietotekniikan kandidaatintyö

2023

Konsta Keski-Mattinen

Tarkastaja: Tutkijatohtori Jiri Musto

Tiivistelmä

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT Teknis-luonnontieteellinen

Tietotekniikka

Konsta Keski-Mattinen

TIETOVIRTA OHJELMOINTIMALLINA JA SEN SOVELTUVUUS REAALIAIKAISEEN JATKUVAAN MITTAUKSEEN

Tietotekniikan kandidaatintyö

2023

30 sivua, 3 kuvaa ja 3 taulukkoa

Tarkastaja(t): Tutkijatohtori Jiri Musto

Avainsanat: Tietovirta, graafilaskenta, ohjelmointiparadigma, mittausjärjestelmä, jatkuva mittaus, konstruktiivinen tutkimus

Tässä kandidaatintyössä tutkitaan tietovirtaan perustuvaa ohjelmointimallia jatkuvan reaaliaikaisen mittauksen ympäristössä. Tavoitteena on saada selville, onko malli kykenevä toimimaan edellä mainitussa ympäristössä, sekä mitä ohjelmistokehittäjän on huomioitava kyseistä mallia käyttäessä. Työssä tehtiin kirjallisuuskatsaus tietovirtamallia käyttäviin järjestelmiin, löytääkseen toteutusohjeita ja vauhdittaakseen kehitystä ja työssä rakennettiin uusi mittausjärjestelmä tietovirtamallia käyttäen.

Työssä käytettiin viisivaiheista konstruktiivista tutkimusmallia ja tutkimuksen onnistuminen mitattiin kahdella mittarijoukolla. Ensimmäinen mittarijoukko koostui ohjelmiston ylläpidettävyydestä ja toinen mittarijoukko mittaa rajallisten resurssien käyttöä.

Toteutettu ohjelmisto täyttää sille annetut mittarit ja raja-arvot ja on täten kykenevä toimimaan reaaliaikaisessa jatkuvan mittauksen järjestelmissä tämän työn asettamilla raja-arvoilla. Lisäksi kehitysvaiheessa nousi huomioitavaa, joista tärkeimpänä on minimoida olioiden määrä ja seurata, etteivät rinnakkaiset solmut vaikuta toistensa syötteisiin.

Abstract

Lappeenranta–Lahti University of Technology LUT

School of Engineering Science

Software Engineering

Konsta Keski-Mattinen

DATAFLOW AS PROGRAMMING PARADIGM IN REALTIME CONTINUOUS MEASURING.

Bachelor’s thesis

2023

30 pages, 3 figures, and 3 tables

Examiners: Jiri Musto D.Sc. (Tech.)

Keywords: Dataflow, graph calculation, programming paradigm, measuring system, continuous measuring, constructive research

This thesis investigates a dataflow-based programming model in a continuous real-time measurement environment. The aim is to find out if the model can operate in the above-mentioned environment, and what the software developer needs to consider when using this model. The work conducted a literature review of systems using the dataflow model to find implementation guidelines and to accelerate development and built a new measurement system using the dataflow model.

A five-stage constructivist research design was used, and the success of the research was measured by two sets of metrics. The first set of metrics consisted of the maintainability of the software and the second set of metrics measured the use of limited resources.

The implemented software meets the metrics and thresholds set for it and is thus capable of operating in a real-time continuous measurement system within the limits set by this work. In addition, during the development phase, issues arose, the most important of which is to minimize the number of entities and to monitor that parallel nodes do not affect each other's inputs.

Alkusanat ja kiitokset

Haluan kiittää kaikkia teitä, jotka olette auttaneet ja tukeneet minua tämän kandidaatintyön valmistelussa. Kandiohjaajaani haluan kiittää loistavasta ohjauksesta ja neuvonnasta, vaikkakin ohjaushetket ohjautuivat nopeasti muualle kuin itse aiheeseen. Työn kielen tarkistajaa haluan kiittää huolellisesta ja perusteellisesta tarkistustyöstä sekä jaksamisesta oman aikataulutukseni kanssa. Esimiestäni haluan kiittää tukevasta ja kannustavasta asenteesta ja tietovirtojen konseptin esittämisestä. Opiskelukavereitani haluan kiittää heidän antamasta vertaistuesta ruusuin.

Ilman teidän apuanne ja tukeanne tämä työ ei olisi onnistunut yhtä hyvin tai ollenkaan. Olen erittäin kiitollinen kaikesta tekemästänne panoksesta ja olen iloinen, että olette tukeneet minua tässä projektissa. Kiitos siis kaikille teille paljon!

Symboli ja lyhenneluettelo

3GPP LTE	Kolmannen sukupolven yhteistyöprojekti koskien 4G-verkkoteknologiaa (3rd Generation Partnership Project Long-Term Evolution)
4G	Neljännen sukupolven langaton tiedonsiirtotekniikka (Fourth Generation)
AESA	Elektronisesti skannaava tutkatyyppi (Active Electronically Scanned Array)
ALICE	Koelaitteisto osana CERNin LHC-projektia (A Large Ion Collider Experiment)
ALICE DAQ	Tiedonkeräinjärjestelmä osana ALICE-projektia (Data AcQuisition)
ASIC	Sovelluskohtainen integroitu piiri (Application Specific Integrated Circuit)
CERN	Euroopan hiukkasfysiikan tutkimuskeskus (Consuel Européen pour la Recherche Nucléaire)
CPU	Prosessori (Central Processing Unit)
FIFO	Järjestyksellinen jono (First-in, First-out)
FPGA	Uudelleen ohjelmoitava mikropiiri (Field-programmable gate array)
HEVC	Videonpakkausstandardi (High Efficiency Video Coding)
HLT	Tarkka suodatin osana ALICE DAQ -projektia (High Level Trigger)
IO	Tietokoneen syöte- ja tulostelaitteet (Input and Output)

IoT	Esineiden internet, verkkokytkeytyneiden laitteiden järjestelmä (Internet of Things)
LE	Matalan energian odotustila Bluetooth-järjestelmissä (Low Energy)
LHC	Maailman suurin hiukkaskiihdytin ja hadronitörmäytin (Large Hadron Collider)
MPEG	Työryhmä videonpakkausstandardien luomiseen (Moving Picture Experts Group)

Sisällysluettelo

Tiivistelmä

Abstract

Alkusanat

Lyhenneluettelo

Sisällysluettelo

1. Johdanto

Tässä kandidaatintyössä tarkastellaan suunnattuun graafiin pohjautuvaa tietovirtaohjelmointimallia, sen soveltuvuutta jatkuvaan reaaliaikaiseen mittaukseen ja mallin käyttöönotossa huomioitavia asioita. Tämä kandidaatintyö on tehty konstruktiivisella tutkimusmallilla. Mallin soveltuvuutta ohjelmistoympäristöön tarkastellaan kokeellisesti tuottamalla esimerkkiohjelma ja arvioimalla sen toimivuutta asetetuilla mittareilla. Työssä tuodaan esille ohjelmointimallia toteuttaessa ja käyttäessä huomioitavia asioita ja tarjotaan niille ratkaisuja. Työn tarkoitus on toimia yleiskatsauksena tietovirtaohjelmointimalliin.

1.1 Tausta

Tietovirtaohjelmointimallia käyttäessä ohjelma mallinnetaan yksinkertaisiin osatehtäviin, jotka yhdistellään ylhäältä alas virtaavaksi graafiksi, jossa tieto virtaa syötteestä ulostuloon ohjelmoidun graafin polkua pitkin. Malli mahdollistaa rinnakkaisuuden, sillä tietovirran polut ja solmut ovat riippumattomia toisistaan.

Työn aihe on valittu ohjelmointimallin tuntemattomuudesta opiskelijapiireissä, vaikka malli kasvattaa suosiotaan isoissa tekoälykirjastoissa, kuten PyTorch [1] (Linux Foundation) ja TensorFlow [2] (Google). Mallia on otettu käyttöön suurien tietomäärien analysoimisessa solmujen tulosteiden jakamisen takia ja siksi, että mallia käyttävä ohjelma voi reagoida syötteeseen välittömästi. Analysoinnissa ja ennustuksessa malli mahdollistaa uusien ominaisuuksien kytkemisen lennosta ja eri mallien keskinäisen vertailun.

Isot ohjelmistotalot ovat huomanneet tietovirtamallin hyödyt ja ovat aloittaneet tietovirtaratkaisujen tuottamisen: Data Pipeline [3] (Amazon Web Service), Power Platform [4] (Microsoft) tai Cloud Dataflow [5] (Google) [6].

1.2 Tavoitteet ja rajoitukset

Työn päätavoitteena on tarkastella tietovirtaohjelmointimallin soveltuvuutta jatkuvaan reaaliaikaiseen mittaukseen. Lisäksi tarkoituksena on tuoda esille esimerkkiä hyväksikäyttäen tietovirtamallin tehokkaaseen ja virhevapaaseen käyttöön liittyviä huomioita. Työn kaksi keskeisintä tutkimuskysymystä voidaan määritellä seuraavasti:

1) Soveltuuko tietovirtaohjelmointimalli jatkuvaan reaaliaikaiseen mittaukseen?
2) Mitä on huomioitava tietovirtaohjelmointimallia käyttäessä?

Työ rajataan tarkastelemaan tietovirtamallin käyttämistä ohjelmiston rakenteena osana imperatiivista ohjelmointia yleistietokoneella. Työssä ei tarkastella tietovirtamallin käyttöä osana tietokoneen arkkitehtuuria taikka osana sulautettuja järjestelmiä. Työ on rajattu ohjelmiston tuotantoon eikä sisällä jälkikäsittelyä.

1.3 Rakenne

Kandidaatintyön rakenne seuraa tieteellisen työn rakennetta. Johdannon jälkeen kappaleessa kaksi käydään yleiskatsaus tietovirtaohjelmointimalliin ja eri tietovirtamalleihin. Kolmannessa kappaleessa tutustutaan muihin tietovirtamallia käyttäviin järjestelmiin ja ratkaisuihin. Pohjatietojen jälkeen neljännessä kappaleessa esitetään kandidaatintyössä käytettyä konstruktiivista tutkimusmenetelmää ja esitellään työn mittarit. Viidennessä kappaleessa käydään läpi työn kehittämisvaihetta, sisältäen esimerkkiohjelman toteutusprosessin ja prosessissa nousseet huomiot. Kuudennessa kappaleessa esitellään työn tulokset ja työn suoriutuminen mittareilla. Seitsemännessä kappaleessa esitetään teoreettiset ja tutkimukselliset yhteydet ja tuodaan työ päätökseen yhteenvedolla.

Kandidaatintyön rakenne seuraa tieteellisen työn rakennetta. Johdannon jälkeen kappaleessa kaksi käydään yleiskatsaus tietovirtaohjelmointimalliin ja eri tietovirtamalleihin. Kolmannessa kappaleessa tutustutaan muihin tietovirtamallia käyttäviin järjestelmiin ja ratkaisuihin. Pohjatietojen jälkeen neljännessä kappaleessa esitetään kandidaatintyössä käytettyä konstruktiivista tutkimusmenetelmää ja esitellään työn mittarit. Viidennessä kappaleessa käydään läpi työn kehittämisvaihetta, sisältäen esimerkkiohjelman toteutusprosessin ja prosessissa nousseet huomiot. Kuudennessa kappaleessa esitellään työn tulokset ja työn suoriutuminen mittareilla. Seitsemännessä kappaleessa esitetään teoreettiset ja tutkimukselliset yhteydet ja tuodaan työ päätökseen yhteenvedolla.

2 Tietovirtaohjelmointimalli

Tässä kappaleessa esitetään tietovirtamalli ja työn muut käsiteltävät konseptit. Kappaleessa tarkastellaan yleisesti mallia ja käydään läpi kirjallisuudessa nousseita huomioita mallin toteuttamisesta.

2.1 Tietovirta ohjelmointimallina

Tietovirtamallissa ohjelma pilkotaan rekursiivisesti työperäisiin osiin, joita kutsutaan solmuiksi ja koko ohjelmaa verkoksi tai graafiksi. Halutun tavoitteen saavuttamiseksi solmuja ketjutetaan yhdistämällä yhden solmun tuloste seuraavan solmun syötteeseen. Koska solmut aktivoituvat vasta niiden kaikkien syötteiden ollessa valmiina, ei ole tarvetta ohjelman sisäiselle hallintajärjestelmälle tai vuorottajalle. Tietovirtamalli antaa kyvyn lukea syötettä samaan aikaan kun aikaisempi mittaus olisi vielä kesken.

Malli on luonnostaan sopiva monisäikeiseen laskentaan, koska eri polkujen solmut eivät ole kytköksissä keskenään. Lisäksi jokaiselle solmulle voidaan luoda oma säie, joka pysäytetään solmun odottaessa syötettä. Tietovirtamalli sopii ratkaisuihin, jossa syöte on virtamainen, eli data saapuu nopeasti pienissä paketeissa. Tietovirtaratkaisun on kestettävä pieni viive mittauksen ja osittaisen tuloksen välillä [7]. Tämä johtuu mallin sisäisistä viiveistä, jotka muodostuvat, koska jokainen solmu toimii itsenäisesti. Myös solmujen aktivointi ja säikeiden vuoroitus prosessorille aiheuttavat viivettä malliin.

Koska tietovirtamallin suunnattu graafi on vain abstrakti rakenne, raskaammassa laskennassa voidaan nostaa mallia tietoverkkotasolle, missä solmuja toteuttavat eri tietokoneet [8].

Solmujen käsittelyssä on eroavaisuuksia eri tietovirtamallien välillä. Kahnin tietovirtamallissa solmut toimivat kutsupohjaisesti ja tyhjentävät kaikki valmiit syötejoukot. Denniksen tietovirtamallissa vuorostaan solmu aktivoidaan heti ensimmäisen syötejoukon ollessa valmiina, milloin solmu kuluttaa vain yhden syötejoukon. [9]

2.1.1 Solmu

Matalimman tason solmujen on kulutettava vain yhtä rajallista resurssia, olkoon se IO-operaatioita tai paljon prosessoriaikaa. Solmut on muodostettava erillisiksi, jotta rinnakkaisuus ja asynkronisuus paranisivat.

Tietovirtamallin solmujen interaktiot on rajattu niiden syötteisiin ja tulosteisiin, kun tarkastellaan tiedonkäsittelyä mallin sisällä. Solmulla voi olla useampi syöte ja tuloste, tietenkin huomioiden, ettei nollan syötteen ja tulosteen solmu ole erityisen hyödyllinen.

Solmu toteutetaan, kun sillä oleva syötejoukko, eli solmun kaikki syötteet, on valmiina. Koska solmut toteuttavat eri tehtäviä eri aikaan, on kyettävä säilömään valmiiksi laskettuja syötteitä. Nämä säilöt toteutetaan FIFO-jonona, jottei syötteiden järjestys sekoittuisi. [9]

Muodostamalla käsittelysolmut yleisiksi voi valmiita solmuja jakaa eri tietovirtaratkaisujen välillä [1].

2.1.2 Graafi

Graafin oikean muotoisella rakenteella vähennetään tarvittavaa uudelleen laskemista, kun solmun tuloste voidaan jakaa monelle eri solmulle. Lisäksi graafi voi koostua solmujen joukoista, joita voi mieltää ryhmäksi.

Graafin muodostuksessa käytetyillä solmuilla voi muodostaa yleisiä rakenteita, kuten ketjuttaminen (engl. pipeline), jakaminen (engl. farm), suodatus (engl. filter) ja kerääminen (engl. accumulator). Ketjuttamisessa on asetettu monta erillistä solmua peräkkäin yhdeksi putkeksi. Jakamisrinnastuksessa jaetaan syöte tasaisesti monelle samalle solmulle, jonka jälkeen solmujen tulosteet yhdistetään takaisin yhdeksi virraksi. Suodatuksessa päästetään läpi vain suodattimen hyväksymät tietopaketit. Keräämisessä muodostetaan yksi paketti useammasta peräkkäisestä paketista. [10], [11]

Perusteellisin yleinen rakenne on ketjuttaminen, jossa eri operaatioita tekeviä solmuja ketjutetaan peräkkäin. Ketjuttamisen hyötynä on, että eri ketjut ovat täysin riippumattomia toisistaan. Esimerkkinä ketjuttamisesta olisi lukea tietolähteestä tietoa formaatissa A, kääntää se tallennettavaan formaattiin B ja lopuksi säilöä tietosäilöön.

Jakaminen on tärkeää, jos yksi solmu ei jaksa laskennan vaativuuden tai tietomäärän takia pysy mukana tietomäärässä. Tällöin on jaettava syötteet usealle samalle solulle, jotka laskevat rinnakkain ja lopuksi tulokset on koottava takaisin yhdeksi virraksi. Esimerkkinä jakamisesta voisi olla hajautettu palvelinverkko, jossa kutsu ohjataan kykenevimmälle palvelimelle.

Suodatus on tärkeä rakenne, sillä yleensä analysointi on raskasta ja aikaa vievää. Täten on syötettä suodatettava ja rajattava niin, ettei kalliita laskentasolmuja hukata. Rajaaminen on lähellä suodatusta. Suodattamisessa rajataan tietopaketteja, kun taas raamisessa rajataan käsiteltävien tietopakettien kokoa. Esimerkki suodatusrakenteesta voisi olla riistakamera, jossa suodattimena toimii liiketunnistin. Tällöin analysoitavaksi, tai tässä tapauksessa kuvattavaksi, päätyy vain, jos liiketunnistimen suodattimesta päästään ohitse. Tällä tavalla varmistetaan, että vain mielenkiintoa tarvitsevat kuvat lähetetään.

Keräämisessä kerätään joukko syötteitä ja lähetetään joukkona eteenpäin. Video on hyvä esimerkki keräämisestä. Kamera tuottaa paljon kuvia ja kuvat kerätään aikajärjestykseen, joka muodostaa yhden videon. Video voidaan sitten jakaa omana kokonaisuutenaan. Keräämisessä on myös muita hyötyjä kuin konseptuaalinen rakenne. Kun kerätään monta syötettä ja päästäessään eteenpäin vain yhtenä tulosteena, säästetään solmujen välisessä viestinnän määrässä.

2.2 Jatkuva reaaliaikainen mittaaminen

Lopullinen järjestelmä tulee toimimaan osana tuotantolinjaa, jolloin järjestelmän tuottama mittaus ja analyysi on oltava valmiina ennen seuraavaa tuotantolinjan vaihetta. Järjestelmän tulee täten kyetä mittaamaan ja tuottamaan analyysi tuotantolinjan tahdissa, etteivät tulokset ruuhkaudu ja siten myöhästy.

Tietovirtaohjelmointimallissa tämä toteutuu varmistamalla, ettei minkään solmujen syötepuskuri kasva pidemmällä aikamääreellä. On kuitenkin huomioitava, että verkon raskaammat solmut voivat kestää pidempään kuin mittausväli, jolloin mitattua dataa on suodatettava ja ryhmitettävä.

Ohjelmoinnissa yleisimmin kutsuttujen metodien ja funktioiden on oltava nopeita ja tehokkaita. Tietovirtamallissa aktivoituu eniten verkon ensimmäiset solmut, joiden on oltava täten tehokkaita ja paljon nopeampia kuin mittausväli jättäen aikaa niitä seuraaville solmuille. Mittausverkolle on suotava rakentaa syötteen suodattimet etupäähän, jotta raskaammat analyysisolmut eivät ruuhkautuisi ja tekisivät vain vaadittavan minimityön.

3 Muita tietovirtamallia käyttäviä ratkaisuja

Tietovirtamallin abstraktiuden takia on mallia käytetty erilaisissa tarkoituksissa, jopa taiteessa[12] ja eri kertaluokan laitteistoissa. Tarkastelkaamme muutamia erilaisia ja eri kertaluokan ratkaisuja.

3.1 Sulautetut järjestelmät

Tietovirtamallia on käytetty sulautetuissa järjestelmissä mikroprosessorien sisustasta ja kokonaisten laitteiden toteutuksesta jopa IoT-järjestelmiin. Tietovirtamalli on sopiva sulautettuihin järjestelmiin, sillä malli jakaa valmiin tuloksen seuraaville solmuille. Lisäksi tietovirtamalli on helppo toteuttaa sulautetussa järjestelmässä, sillä solmut voidaan toteuttaa yksittäisillä mikroprosessoreilla tai mikroprosessorin sisäisesti.

Pienimmillä sulautettujen järjestelmien tasolla on mikroprosessori. Tietovirtamallin laskentaverkko saadaan mahdutettua mikroprosessoriin kääntämällä solmut omina ohjelmina ennen ajoa, jolloin on mahdollista ajaa tietovirtamallin ohjelmia mikroprosessorissa [13].

Noustessa tasoa ylemmäs, jolloin mikroprosessorit ovat osana sulautettua järjestelmää, on tietovirtamallilla toteutettu laiteohjaimia, pakkaajia ja purkajia. Esimerkkejä tietovirtamallia käyttävistä sulautetuista järjestelmistä ovat Bluetooth LE -ohjain [14] ja 3GPP LTE -projektin, eli 4G-verkkotekniikan, fyysisten viestien pakkaus ja purkaminen [15].

Tietovirtamallilla pystytään käsittelemään suuret määrät tietoa sulautetuissa järjestelmissä. Älykameroiden prosessointia ja tietoliikennettä on tehostettu tietovirtaratkaisulla [16]. Toinen suuren tietomäärän tietovirtamallia käyttävä sulautettu järjestelmä on elektronisen tutkan mittausjärjestelmä AESA. AESA-tutkassa tietovirtamallia käytetään antennien tuottaman tiedon analysoimiseen [17]. Kun antennien välisten tulosten eroavaisuuksia analysoidaan, muodostetaan aluetutka tasaisella mittaristolla [17].

Tietovirtamallilla voidaan esittää IoT-järjestelmiä. IoT-laitteet muodostavat laskentaverkon, jossa yhdistelemällä eri laitteiden jatkuvaa tiedon tuottoa voidaan seurata järjestelmän tilaa sekä seurata ja hallita tietovirtoja [18].

3.2 Yleistietokoneet

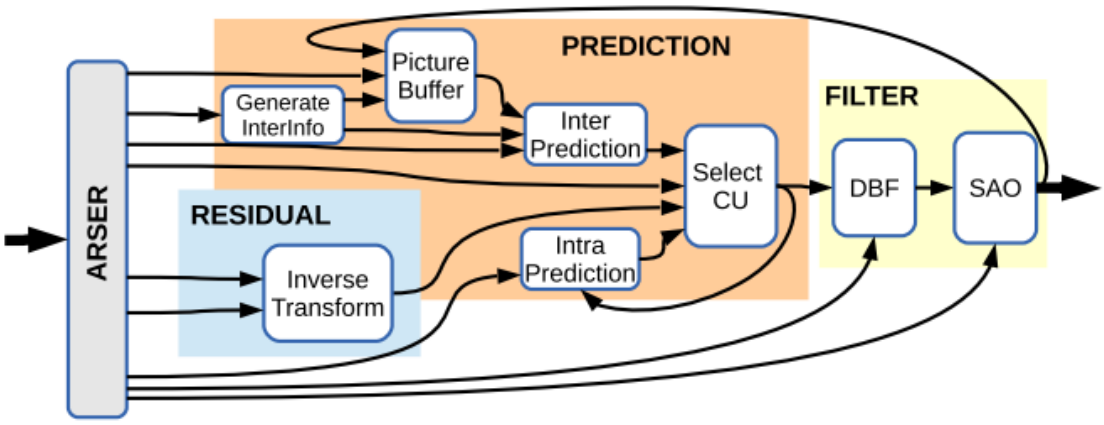
Yleistietokoneissa tietovirtamallia käytetään eniten erillisenä laskentakiihdyttimenä näytönohjaimen tavoin. Kiihdytintä käyttäviä ratkaisuja ovat esimerkiksi suurilla taulukoilla kymmenen kertaa nopeampi binäärihaku [19] tai nopea Fourier-muunnin [20].

Tietovirtakiihdyttimiä voidaan käyttää tekoälyratkaisujen vauhdittaakseen, sillä neuroverkkoratkaisut toteuttavat fundamentaalisesti tietovirtamallin. Täten on luonnollista esittää neuroverkon solmut tietovirtasolmuina ja käyttää ulkoista tietovirtamallikiihdytintä.

Neuroverkkojen kiihdytys voidaan toteuttaa koulutuksessa ja itse mallin ajossa, jolloin tietovirtamallilla kiihdytetty ratkaisu on todettu olevan energiatehokkaampi ja nopeampi kuin yleistietotokoneella laskettuna [21].

TensorFlow vastaa eniten tämän työn tapaa käyttää tietovirtamallia. TensorFlow on tekoälyn koulutusjärjestelmä, joka toteuttaa koulutusprosessin tietovirtoina mallin rinnakkaisuuden vuoksi.

Ohjelmistorakenteen tasolla tietovirtoja on käytetty osana uudempia Moving Pictures Experts Group -työryhmän (MPEG) tarjoamia videokoodausstandardeja. Tietovirroilla toteutettu purkaja on tehokkaampi ja skaalautuvampi kuin vaihtoehtoiset ratkaisut [22]. Täten on luontevaa toteuttaa videokoodausstandardi tietovirtamallilla.



Kuva 1: Tietovirroilla mallinnettu HEVC-purkaja. [22]

3.3 Suuren tietomäärän järjestelmät

Tietovirtamallia on käytetty suurien tietomäärien analysoimisessa solmujen tulosteiden jakamisen takia. Tällöin voidaan kytkeä uusia analyysipolkuja käsiteltyyn syötteeseen. Tämä mahdollistaa uusien ominaisuuksien kytkemisen lennosta ja eri mallien keskinäisen vertailun.

Isojen ohjelmistotalojen tietovirtaratkaisut, kuten Googlen Cloud Dataflow, Amazonin Data Pipeline ja Microsoftin Power Platform ovat tiedonkäsittelypalveluita, joista jokainen tarjoaa tietovirtoja hyödyntäviä ominaisuuksia osana palveluaan. Tietovirtamallia käytetään häilyttämään reunalaskenta (engl. edge computing) loppukäyttäjältä.

3.3.1 CERN

Osana CERN:n LHC-projektia on koeasema ALICE, jonka mittausjärjestelmä ALICE DAQ on toteutettu tietovirtamallilla. Muista esiteltävistä ratkaisuista eroten ALICE DAQ on hyvin pienen taajuuden järjestelmä, mutta käsittelee suuren määrän dataa pienissä sykleissä. Tämä on toteutettu erottamalla jokainen vaihe omaksi tietokoneeksi ja yhdistelemällä solmut optisin yhteyksin. [23]

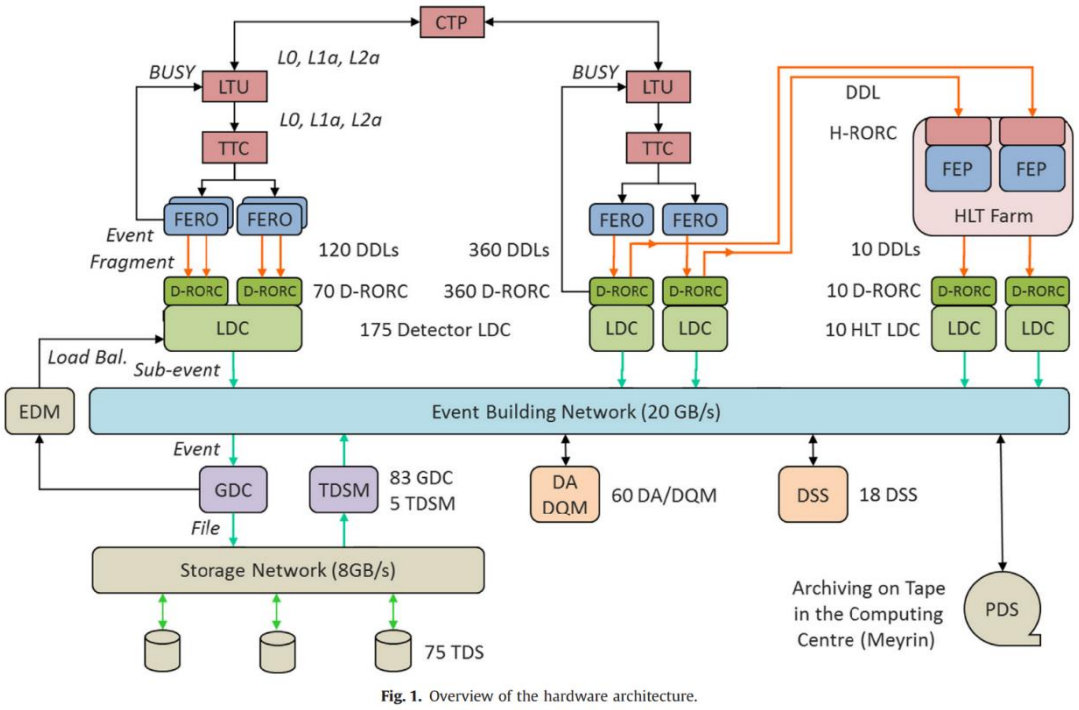
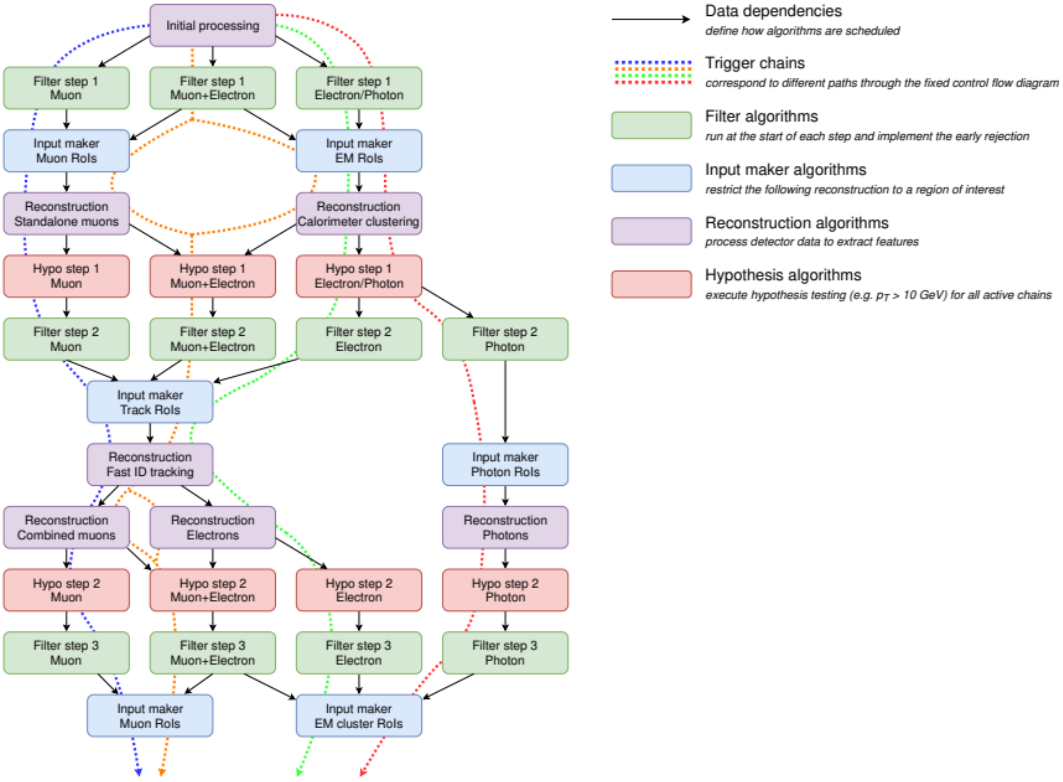


Fig. 1. Overview of the hardware architecture.

Kuva 2: ALICE DAQ tietovirtamalli. Kuvassa punaiset ovat suodattimia, siniset puskureita, vihreät laskentaa ja harmaat säilöntää. [23]



Kuva 3: CERN:n HLT-laskentakeskuksen käyttämä tietovirtamalli. Kuvassa nuolet ja katkoviivat esittävät tietovirtaa. Kuvassa vihreällä suodattimet, sinisellä rajaimet, violetilla ja punaisella ennustusalgoritmit. [24]

Kuvassa 2 näkyy hyvin aikaisemmin mainitut tietovirtaosarakenteet, kuten ketjuttaminen, jakaminen ja suodattaminen mahdollisimman aikaisin. Kuvissa 2 ja 3 näkyy selkeästi tietovirran optimointi, jossa tietovirtaa suodatetaan ja rajataan mahdollisimman aikaisin.

4 Tutkimusprosessi

Tämän työn toteutussuunnitelma on kolmiosainen. Ensimmäisessä osassa tutkitaan muita tietovirtamallin ratkaisuja. Toisessa osassa suunnitellaan ja tuotetaan kokeellinen jatkuvassa mittauksessa toimiva ohjelma tietovirtaohjelmointimallia käyttäen. Työn kolmas osa koostuu empiirisen kokeen havainnoista.

4.1 Kirjallisuustiedon kerääminen

Kirjallisuustieto kerättiin satunnaisella lumipallo-otannalla, jossa valitaan joukko tietolähteitä, joiden lähteistä kerätään rekursiivisesti tietoa. Tapa valittiin tietovirtaohjelmointimallin uuden luonteen takia, sekä kirjallisuusmateriaalin yleisyyden tai etäisyyden takia. Tällöin kirjallisuustieto käsittelee tietovirtamallia joko yleisesti tai esittelee käyttökohteita, jotka eivät vastaa tämän työn tapaa käyttää tietovirtamallia ohjelmiston rakenteena.

4.2 Tutkimusmenetelmä

Tutkimuksessa käytetään konstruktiivista tutkimusmallia, koska tutkimuksessa tuotetaan uusi kokeellinen ohjelma. Konstruktiivinen tutkimusmalli on lähestymistapa, jolla pyritään ymmärtämään ja luomaan uutta merkityksellistä tietoa[25]. Konstruktiivisen tutkimusmallin pääpaino on yksittäisen ongelman ratkaiseminen ja ratkaisun yleisyys[25]. Konstruktiivinen tutkimusmalli voidaan jakaa kuuteen vaiheeseen [25].

1) Tutkimusongelman löytäminen
2) Yleiskäsityksen kerääminen
3) Uuden luominen
4) Toiminnallisuuden tarkistaminen
5) Teoreettiset ja tutkimukselliset yhteydet
6) Tutkimuksen käyttökelpoisuuden arvioiminen

Konstruktiivisen tutkimusmallin ensimmäinen vaihe on tutkimusongelman löytäminen. Tässä työssä tutkimusongelma on selvittää tietovirtamallin toimiminen mittausohjelmiston rakenteessa. Työn tutkimusongelma on esitelty tarkemmin kappaleessa 1.2.

Toisessa vaiheessa kerätään kattava yleiskäsitys tutkittavasta aiheesta ja tutkimuskentästä [25]. Tässä työssä vaihetta vastaa tiedon kerääminen yleisestä tietovirtamallista sekä mihin ja miksi sitä on käytetty. Yleiskäsitystä tietovirtamallista on esitetty kohdassa 2 ja muista tietovirtamalleja käyttävistä ohjelmista on esitetty kohdassa 3.

Tutkimusmallin kolmannes vaihe, uuden luominen, esitellään kokonaisuudessaan kohdassa 5. Vaihe koostuu uuden idean kehittämisestä. Työn toiminnallisuuden tarkistaminen toteutuu osana kehitysprosessia jatkuvana testaamisena. Kehitetyn työn pidempiaikainen toiminta on tämän työn ulkopuolella.

Tutkimusmallin neljännes vaihe koostuu uuden tutkimuksen toiminnallisuuden tarkistamisesta. Mallin toiminnallisuutta arvioidaan ja testataan osana kehitysprosessia. Mallin pidemmän ajan toimiminen on tämän työn ulkopuolella. Tietovirtamallin käyttökelpoisuuden arviointi koostuu soveltuvuuden arvioimisesta, joka mitataan kappaleen 4.3 mittareilla kohdassa 6.

Tutkimusmallin viides ja kuudes vaihe koostuu työn tuottaman tiedon osoituksesta osana tutkimusta. Viidennessä vaiheessa esitetään työn teoreettiset ja tutkimukselliset yhteydet muihin alan julkaisuihin [25]. Tutkimusmallin viimeisessä eli kuudennessa vaiheessa esitetään työn tuottaman tiedon hyödyllisyys [25]. Työn teoreettiset ja tutkimukselliset yhteydet sekä tutkimuksen käyttökelpoisuuden arvioiminen esitellään kappaleessa 7.

Taulukko 1: Työn tutkimusmenetelmän vaiheet

Vaihe	Konstruktiivisen tutkimusmalli	Tässä työssä tehty
1	Tutkimusongelman löytäminen	Soveltuuko tietovirtaohjelmointimalli jatkuvaan reaaliaikaiseen mittaukseen? Mitä on huomioitava tietovirtaohjelmointimallia käyttäessä?
2	Yleiskatsauksen kerääminen	Yleisen ymmärryksen kerääminen tietovirtamallista ja sen toiminnasta.
3	Uuden luominen	Luodaan uusi kokeellinen ohjelmisto testaamaan mallin toimivuutta asetetussa ympäristössä.
4	Toiminnallisuuden tarkistaminen	Ohjelmiston toiminnallisuus tarkistetaan jatkuvalla testauksella ja mallin soveltuvuutta arvioidaan viidellä mittarilla. Mallin käyttökelpoisuus arvioidaan työssä käytetyillä mittareilla.
5	Teoreettiset ja tutkimukselliset yhteydet	Aikaisemmassa tutkimuksessa tietovirtamalli oli ulkoistettu erillisille laitekiihdyttimille tai sulautetuille järjestelmille. Työ tutkii mallin toimivuutta laskentaohjelmiston rakenteena.
6	Tutkimuksen käyttökelpoisuuden arvioiminen	Työ esittää, voiko tietovirtamallia käyttää ohjelmointimallina vaativassa ympäristössä. Työ esittää oman asiansa käytettäväksi muille tutkimuksille.

Taulukko 1 näyttää konstruktiivisen tutkimusmenetelmät vaiheet ja lyhyesti mitä eri vaiheessa on tämän työn puitteissa tehty.

4.3 Mallin soveltuvuuden arviointi

Mallin soveltuvuudella tarkoitetaan mallin kykyä toimia sille annetussa ympäristössä ja tehtävissä. Tämä tarkoittaa, että tietojärjestelmä on oltava riittävän tehokas, luotettava, helppokäyttöinen ja yhteensopiva. Tässä työssä keskitytään tehokkuuteen, luotettavuuteen ja yhteensopivuuteen. Mallin soveltuvuutta arvioidaan erilaisten mittarien ja kynnysehtojen avulla. On tärkeää huomioida, että mittarit mittaavat vain mitattavaa suuretta, eivätkä taustalla olevaa kohdetta. Työn mittarit hyväksytään, jos ne toteuttavat niille annetut raja-arvot tai kynnysehdot. Mallin soveltuvuus mitataan seuraavilla mittareilla:

1) Koodirivien määrä
2) Ohjelmiston ylläpidettävyys
3) Ajon aikainen ajallinen suoriutuminen
4) Ajon aikainen muistin käyttö
5) Ajon aikainen laitteiston käyttö

Mittarit ovat osittain kytkeytyneitä, sillä esimerkiksi algoritmi, joka varaa vähän muistia, on nopeampi verrattuna paljon muistia varaavaan algoritmiin. Koodirivien määrä vaikuttaa myös ohjelmiston ylläpidettävyyteen, sillä pienempi rivimäärä on nopeammin luettavissa ja ymmärrettävissä. Pienempi rivimäärä ei kuitenkaan takaa ylläpidettävää ohjelmistoa. Työn mittarit on valittu tietojärjestelmän toteutusympäristön takia ja jakautuu konseptuaalisesti kahteen joukkoon.

Ensimmäisessä joukossa on ohjelmiston ylläpidettävyyteen keskittyvät ominaisuudet: koodirivien määrä ja empiirinen tunne ohjelmiston ylläpidettävyydestä. Tämä osa mittareista on valittu esittämään ohjelmiston jatkettavuutta, jottei kokeellinen ohjelmisto päädy asenna ja unohda -ratkaisuun.

Koodirivi mittari esittää toissijaisena mittarina ohjelmiston luettavuutta. Nykyajan ohjelmointiympäristö on kiireellinen ja muutoksia järjestelmään tekee mahdollisesti kehittäjä, jolle ohjelman rakenne ja toiminta ei ole täysin sisäistyneet. Tämän takia on tärkeää, että ohjelman osasta pystyy nopeasti saamaan selkeän kuvan ja tätä auttaa koodirivien vähyys.

Ohjelmiston ylläpidettävyysmittari ottaa laajemmin kantaa ohjelmiston ylläpidettävyyteen ja sisäistämiseen. Mittari koostuu kehittäjän sisäisistä kannoista ylläpidettävyyteen, jotka varmennetaan ohjelmoinnista tietävällä, muttei kyseistä projektia kehittävällä kontrollihenkilöllä.

Mittarien toinen joukko koostuu resurssien käytön minimoimisesta. Mittarit tässä osassa mittaavat ohjelmiston ajonaikaista rajattujen resurssien käyttöä, kuten muistin ja laskentayksikköjen käyttöä ja ohjelmiston ajallista suoriutumista.

Ohjelmiston ajallinen käyttäytyminen ajon aikana on tämän sovelluksen tärkeimpiä mittareita. Koska ohjelmiston on kyettävä jatkuvaan mittaukseen, on mittausjoukko kyettävä laskemaan seuraavaan vaiheeseen, ettei ruuhkia synny. Asetetaan mittausjoukon syöteajan raja-arvoksi <~1ms. Toisin sanottuna uusi mittaus tapahtuu joka <~1ms, jolloin ohjelman on kyettävä suoriutumaan mittausjoukon laskennasta alle millisekunnissa.

Ajon aikainen muistin käyttö on yksi tärkeistä mittareista, joka määrää mittausjärjestelmän kustannusta ja laajennuskykyä. Mittarin tavoitteena on seurata, pysyykö ohjelmisto verrokkien muistinkäytössä ja jääkö tietopaketteja ohjelmistoon jumiin. Tämä on minimoitava arvo.

Ajon aikainen laitteiston ja laskentayksikköjen käyttö on toinen mittausjärjestelmän kustannukseen vaikuttava mittari. Ohjelma on toteutettava, että pullonkauloja ei muodostu ja laskentatehoa säilyy tulevaisuutta ja muita sivujärjestelmiä varten.

Taulukko 2: Työssä käytetyt mittarit

Mittari	**Tavoite**
Koodirivien määrä	Verrattavissa oleva mittausjärjestelmä
Ohjelmiston ylläpidettävyys	Asiantuntijan ja tutkimusryhmän mielipiteet
Ajon aikainen ajallinen suoriutuminen	Kestettävä <~1ms syöteajan mittareita.
Ajon aikainen muistin käyttö	Vastaavan järjestelmän kokonaismuistinkäyttö.
Ajon aikainen laitteiston käyttö	Mikään resurssin ei saa muodostaa pullonkaulaa ja toteutetulla laitteistolla on mahdollisuus laajentaa laskentaa.

Taulukko 2 esittää työssä käytetyt mittarit ja niiden hyväksymiskynnykset.

5 Kehitysprosessi

Malli on osa kokeellista projektia, jonka tavoitteena on tutustuttaa ja kouluttaa tutkijoita, selvittää mallin ominaisuudet ja luoda uusi laskentajärjestelmä. Tietovirtamallin toteuttamisella halutaan jatkaa mittarilta tulevaa tietovirtaa jatkuvana tietokohteille asti. Ohjelma tehdään käyttäen olio-ohjelmointikieltä käyttäen ohjelmistokehystä, joka toimii tietovirtarakenteen mukaisesti. Kehys mahdollistaa tarkan toteutuksen abstraktoinnin ja ohjaa kehittäjän aikaa itse solmuihin.

Malli toteutetaan yleistietokoneella, joka käyttää x64-käskysarjaa ohjelman samanlaistamiseksi muihin tuotteisiin ylläpito- ja korvauskykyisyys syistä. Tämä tapa on hieman tehottomampi ja hitaampi kuin FPGA- ja ASIC-toteutus [14], mutta tarjoaa helpon kyvyn ylläpitää ja päivittää ohjelmistoa tarpeen ja vaatimusten kehittyessä. Yleistietokoneista on kuitenkin hyötyä, sillä mittausjärjestelmää tukevat ohjelmat löytyvät myös yleistietokoneella.

Prosessia muuttaa normaali ohjelmisto tietovirtakiihdytintä käyttävään ohjelmistoon, on kuvailtu viisiosaiseksi: [26].

1) Ohjelmistoanalyysi
2) Tietovirtarakenteiden luominen ja vertaaminen
3) Solmujen toteuttaminen
4) Solmujen integroiminen tietovirtarakenteeseen
5) Optimointi ja varmennus

Ensimmäisen vaiheen tavoitteena on tunnistaa ohjelman käytetyimmät koodipolut. Toisen vaiheen tavoitteena on luoda erilaisia toteutuksia mallista huomioiden virtaavan tiedon määrä. Kolmannessa vaiheessa toteutetaan solmut tietoverkkoa varten ja neljännessä vaiheessa otetaan solmut osaksi tietoverkkoa. Viidennessä vaiheessa varmennetaan ohjelmiston toiminta ja optimoidaan rakenne suorituskyvylle. Rakenteen pitäisi käyttää tasaisesti kaikkia rajaavia resursseja, niin ettei mikään resurssi muodosta pullonkaulaa tietoverkkoon. [26]

Koska malli toteutetaan täysin yleistietokonejärjestelmillä ilman erillistä tietovirtakiihdytintä, on käytetty prosessi lyhyempi. Solmut on toteutettu samoilla ohjelmointikirjastoilla kuin niitä yhdistelevä koodi. Käytetty prosessi on lyhyempi, sillä se koostuu vain osittaisesta ohjelmistoanalyysista, tietovirtarakenteiden ja solmujen luomisesta ja vertaamisesta, sekä viimeisestä osasta eli optimoinnista ja varmennuksesta. Kevyempi kehitysprosessi johtuu siitä, ettei uutta tietovirtaratkaisua luodessa tarvitse tai ole järkevää luoda mallia käyttämätön ratkaisu sekä muuttaa se mallia käyttäväksi ja tarkistaa prosessin onnistuneisuutta. Lisäksi kokonaan tietovirtaa käyttävässä ohjelmistoa luodessa ei tarvitse arvioida solmujen ja yhdistelevän koodin rajapintoja.

Tietovirran suunnitteluvaiheessa käytettiin kirjallisuuskatsauksesta löytyneitä optimointitapoja optimoimaan luodun laskentaverkon rakenne.

5.1 Suunnittelu

Käytetyn kehitysprosessin ensimmäinen vaihe eli ohjelmistoanalyysi koostui laskentaverkon tehtävien, mittaustietolähteiden, mittaustiedosta etsittävät ominaisuuksien ja eriteltävän tiedon sijoittamissijainnin selvittämisellä. Laskenta verkon erilliset tehtävät ja mittaustiedosta etsittävät ominaisuudet eritettiin omiksi solmuiksi. Tietolähteet toimivat lähdesolmuina ja sijoituskohteet toimivat sijoitussolmuina. Ohjelmiston analyysivaiheessa on tärkeää myös selvittää erillisten analyysisolmujen käyttämät ominaisuudet ja tiedon muoto. Näitä tietoja käytetään muodostaessa tietoverkkoa.

Kehitysprosessin toinen vaihe, eli tietorakenteiden luominen, tehtiin iteratiivisesti aloittaen muodostamalla ohjelmistoanalyysin tuottamat solmut kolmikerroksiseen verkkoon. Verkon huipulla olivat lähdesolmut, verkon keskellä olivat etsittävien solmujen analyysisolmut ja pohjalla sijoitussolmut. Verkossa tietolähteet tuottavat tietoa, eli tietueita, analyysisolmuille ja analyysisolmujen tulosteet syötetään toisten analyysisolmujen jälkeen tallennussolmuille.

Verkon luomisen seuraavassa iteraatiossa tarkoituksena on vähentää analysointisolmujen laskentalastia. Kun tietolähteiden jälkeen lisätään suodatinsolmut, vähennetään analysointisolmuille päätyviä tietueita ja rajataan tietueista vain tarvittavat tiedot.

Suodattimien ja tiedon rajaamisen jälkeen on analysointisolmuille yhteisten operaatioiden vuoro. Yhteiset operaatiot hoidetaan tiedon rajauksen jälkeen, jotta eri polkujen ei tarvitse toteuttaa poluille yhteisiä muunnoksia erikseen ja solmut voivat jakaa tietueita vähentäen muistipainetta.

Yhteisten operaatioiden jälkeen jaetaan tietueet omille poluilleen ominaisuuksiensa perusteella jakajasolmuilla. Jakajasolmujen jälkeen voi lisätä ensisijaisia analyysisolmuja, joilla eritellään helposti tunnistettavia ominaisuuksia. Ensisijaisia analysointisolmujen tulosteita voidaan yhdistää uusissa analyysisolmuissa, kunnes on kytketty aikaisemmin mainittujen etsittävien ominaisuuksien tunnistavat solmut. Viimeiset kytketyt analyysisolmut voidaan sitoa sijoitussolmuihin. Tässä vaiheessa laskentaverkko on kokonaanut mallinnettuna.

Verkkorakenteen rekursiivisuuden vuoksi voidaan abstraktoida osaverkot omaksi verkokseen [27]. Verkkorakenne antaa mahdollisuuden luoda valmiita mittarikohtaisia osaverkkoja tuleville eri sovelluksille ja mahdollisuuden uusiokäyttää laajemmissa ohjelmistoissa tämän työn toteuttamaa verkkoa.

5.2 Solmujen toteuttamien

Kehitysprosessin kolmannessa vaiheessa, solmujen toteuttamisessa, käytettiin olio-ohjelmoinnista perittyjä tapoja ja yleistämistä jakamaan geneerisen työn solmut, kuten keräimet ja kopioijat. Solmujen operointilogiikka toteutettiin mahdollisimman tilattomana vähentäen yhden solmun operointikustannusta ja mahdollistaen harvoin käytettyjen solmujen sammuttamisen vapauttaen työmuistia.

Solmua toteuttaessa on huomioitava, että solmun tulostetta ei jälkikäteen muokattaisi, parhaimmassa tapauksessa tuloste tuotettaisiin vain luku -tilassa. Lukutilaa käytettäisiin sen vuoksi, että tulosteessa olevaa tietoa ei voisi muokata. Tämä varmistaa, ettei rinnakkaiset solmut muokkaa toisiaan ristiin rikkoen tuloksen ja säästääkseen muistia. Toteuttamalla tulosteet vain luku -tilassa yksi olio voidaan jakaa monelle eri solmulle, jotta muistinkeräimen painetta ja käytettyä muistinmäärää ei synny. Lisäksi ei ole tarpeellista toteuttaa solmujen, jotka toimivat omilla säikeillään, välistä lukitusmekanismia tietueille, jotta säästettäisiin toteutus ja ajonaikaista aikaa. Tämä tehdään, jotta samaa tulostetta käyttävät solmut eivät muokkaa toisiensa syötteitä, mikä rikkoisi ohjelmiston.

5.3 Testaaminen

Kehitysprosessin viimeinen vaihe, eli rakenteen varmentaminen, toteutettiin testaamalla mittausverkkoa. Tietovirtamallin takia testaaminen oli toteutettava yksikkötesteinä solmutasolla ja integraatiotesteinä verkkotasolla. Lisäksi, jos käyttää ohjelmointikehystä, on senkin toimintavarmuus testattava. Testauksen luottamustasoksi valittiin tietovirtakehys. Tätä perusteellisimpia palasia ei testattu, vaan luotettiin ohjelmistokielen ja muiden kehysten toimivan oikeellisesti.

Luottamustasoa kasvatettiin kerros kerrokselta testaamalla ohjelmiston perusteellisimmat osat ensin kasvattaen testausaluetta ja laajuutta ajan myötä. Pohjimmaisin testattava rakenne on ohjelmistokehys, jossa testattiin yksittäisen solmun toiminta ja solmujen välinen liikenne. Solmut testattiin yksikköinä, millä luotiin varmuutta yksittäisten solmujen toimintaan. Koko laskentaverkko testattiin integraatiotestauksella, kun varmennuttiin aikaisempien osien oikeellisuudesta. Järjestelmätestauksessa varmennettiin koko järjestelmän toiminta ja mitattiin ohjelmiston suoriutuminen tämän työn mittareilla.

5.4 Nousseita huomioita

Ensimmäisellä toteutuskierroksella malli ei ollut tarpeeksi nopea pysyäkseen tietolähteiden perässä. Mallia riivasi roskankerääjästä johtuvat monen sekunnin viivästykset. Näitä ongelmia lähdettiin ratkomaan kierrättämällä eniten käytettyjä ja nopeinten vaihtuvia olioita. Lisäksi varmistettiin, että kaikki tietueviittaukset on käyty läpi, ettei käsiteltävää tietuetta vapauteta.

Tietovirtamallia käyttämällä tuodaan ohjelmaan uusi alue virheille, jotka ovat solmujen väliset puskurit. Tietovirtamallin solmujen välissä on oltava puskurit solmujen omatoimisuuden takia, jolloin solmun jumittuessa kyseisen solmun puskuri täyttyy ja kaikki solmusta riippuvat solmut nälkiintyvät täyttäen omat puskurinsa. Käytössä on joko rajattomat tai rajatut puskurit. Rajattomilla puskureilla solmujen syötemäärää ei ole rajattu, jolloin yksikin puskuri pystyy varaamaan koko tietokoneen muistin, kunhan ohjelma pyörii tarpeeksi pitkään. Rajatuilla puskureilla on vaarana menettää syötteitä. Syötteitä voi myös kadota virhetilojen takia, jolloin on kyettävä käsittelemään ja merkitsemään osittainen syötejoukko.

Tietovirtamallia käyttäessä muodostuu kokonaan uusi virheluokka. Virheluokka esiintyy tilanteissa, jossa solmu saa ainoastaan osan syötteistään laskettavaksi. Tällöin on ilmoitettava laskun olevan vajava ja virhetila käsiteltävä sopivasti.

Sovelluksen luonteen vuoksi järjestelmän kokonaisvaltainen testaaminen ja eri versioiden tehokkuuden mittaaminen on hankalaa. Erityisesti se, milloin koko ohjelmistoverkko on ajanut rajallisen testiaineiston, koettiin vaikeaksi. Solmut voivat millä tahansa hetkellä odottaa syötteitä, jolloin ne eivät laske aktiivisesti ja solmun jälkeen voi aktivoitua uusia solmuja ketjutuksen takia.

Koko mittausverkon testattavaksi on järjestettävä erillinen simulaatioympäristö, jossa simuloidaan ohjelmistoa käyttävät tietolähteet ja kohteet. Simulaattorien on oltava erillisenä ohjelmana, joka kykenee toimimaan annetuissa raja-arvoissa. Ongelmaksi muodostui ohjelmien välinen kommunikointi, jonka vuoksi viestintäprotokollaa jouduttiin muuttamaan simulaattoreissa tiedon määrän takia.

6 Tulokset

Ohjelmisto on toteutettu onnistuneesti ja se saavutti sille annetut toiveet ja raja-arvot toteutusprosessin jälkeen. Käytetyillä työn mittareilla ohjelma on onnistunut. Mittareissa suoriutuminen oli seuraavanlaista:

Koodirivien määrä on yksi mittareista, jolla seurattiin ohjelmiston ylläpidettävyyttä. Käytössä huomattiin totaalikoodirivien kasvu, joka voidaan selittää tietovirtamallin solmujen, verkon ja viestinnän toteuttavan kehyksen vaatimista riveistä. Vaikka koodirivien kokonaismäärä kasvoi, yksittäisten tiedostojen koko laski roimasti. Jos ei laske mukaan ohjelmointikehyksen vaatimia kiinteitä koodirivejä, on koodirivien määrä laskenut.

Ohjelmiston ylläpidettävyys on toinen ylläpidettävyyden mittareista ja se perustuu enemmän asiantuntijan ja tutkimusryhmän mielipiteisiin ja tuntemukseen. Koska ohjelmisto on jakautunut tietovirtamallin verkon solmuihin, on ohjelmiston rakenne selkeä ylhäältä alas -viivakaavio. Tietovirtamallin ansiosta jokainen solmu on itsenäinen, jolloin solmujen ymmärtäminen ja jälkikäteen muokkaaminen on nopeaa ja selkeää.

Ajon aikainen ajallinen suoriutuminen on ensimmäinen minimoitava tehokkuuden mittari. Ohjelmiston ajallinen suoriutuminen raja-arvoksi asetettiin kestää <~1ms mittausväliä. Ohjelma toimi annetuissa raameissa tietovirtarakenteen ansiosta, sillä yhden mittausjoukon läpikulkemisaika koko laskentaverkosta oli pidempi kuin mittausväli. Tämä onnistui tietovirtamallin kyvystä käsitellä aikaisempaa mittaustietoa myöhäisemmällä solmulla samalla kun uutta mittaustietoa tuotiin järjestelmään. Ohjelmisto oli myös kestävä käyttöjärjestelmän ja laitteiston tuottamaan epävarmuuteen, sillä suurimmat solmun syötejoukon odotusjonot olivat vain pieni osa niille varatuista puskurista.

Ajon aikainen muistin käyttö on toinen tehokkuuden mittari. Mittarin tarkoituksena oli seurata, ettei tietovirtamallin solmujen toteutuksesta aiheudu sietämätöntä muistinkäyttöä. Muistinkäyttöä verrattiin vastaavaan eri rakennetta käyttävään ohjelmistoon. Muistinkäytössä saavutettiin siedettävä tulos, sillä ohjelmiston muistinkäyttö oli verrattavissa verrokkiohjelmiston muistinkäyttöön.

Ajon aikainen laitteiston käyttö on kolmannes minimoitava mittari, jolla seurattiin mittausjärjestelmän kustannusta ja tietovirtamallin käyttämiä laitteistoresursseja. Muiden laiteresurssien käytössä järjestelmä on hyvin säästeliäs kokoisekseen. Isoimpana huomiona oli mittausjärjestelmän CPU:n käyttö, joka oli jakautunut hyvin monelle laskentasäikeelle. Jakautuminen johtuu tietovirtamallin toteutuksesta, jossa jokainen solmu toimii omilla säikeillään, joita vuorottaja voi jakaa eri ytimille. Koska jokainen solmu on omalla säikeellään, raskaammat ja enemmän käytetyt säikeet voivat varata koko ytimen.

Lopuksi katse tulevaisuuteen. Tämä työ on esittänyt, että tietovirtamalli on kykenevä toimimaan työn asettamissa raameissa ja ympäristössä. Olisi tärkeää selvittää, mitkä olisivat tietovirtamallin rajat.

Taulukko 3: Mittareiden tulokset

Mittari	**Tavoite**	**Suoriutuminen**
Koodirivien määrä	Verrattavissa oleva mittausjärjestelmä	Muuttuvien koodirivien määrä väheni, mutta kokonaiskoodirivien määrä kasvoi ohjelmistokehyksen vaatimien koodirivien myötä.
Ohjelmiston ylläpidettävyys	Asiantuntijan ja tutkimusryhmän mielipiteet	Ohjelman rakenne pilkkoutui isosta monoliittisestä luokasta pienempiin yhden solmun luokkiin.
Ajon aikainen ajallinen suoriutuminen	Kestettävä <~1ms syöteajan mittareita.	Toteutettu ohjelma kykeni käsittelemään <~1ms syöteajan mittareita ilman solmuverkon ruuhkautumista.
Ajon aikainen muistin käyttö	Vastaavan järjestelmän kokonaismuistinkäyttö.	Ohjelmisto käyttää hieman verrokkiohjelmaa enemmän muistia.
Ajon aikainen laitteiston käyttö	Mikään resurssin ei saa muodostaa pullonkaulaa ja toteutetulla laitteistolla on mahdollisuus laajentaa laskentaa.	Ohjelmisto käyttää laitteistoa tasaisesti ilman pullonkauloja. Laitteistolla on vaadittu laajennusvara.

Taulukko 3 esittää toteutetun ohjelman suoriutumisen työssä käytetyillä mittareilla. Tämä taulukko on jatkettu taulukosta 2.

7 Yhteenveto

Tässä kandidaatintyössä tutkittiin suunnattuun graafiin pohjautuvaa tietovirtaohjelmointimallia. Mallista selvitettiin sen rakenne ja yleisimmät käyttötavat täysikokoisilla x64-tietokoneilla. Tärkeänä osana tutkimusta oli selvittää, mitä erityistietoa pitää kyseistä mallia toteuttavan kehittäjän ja suunnittelijan tietää.

Työ on tehty pilottihankkeena mallin käyttämiseen ja itsenäisenä ratkaisuna omaan ongelmaansa. Tietovirtamalli virtaominaisuuksien todettiin olevan hyvä jatke jatkuvasti tietovirtaa tuottaville mittareille.

Mallin toteuttava ohjelma on tuotettu ja se täyttää tässä kandidaatintyössä annetut ajalliset ja resurssilliset määreet. Työn ensimmäinen mittarijoukko seurasi raja-arvotettuja ajallisia ja resurssillisia määreitä perustuen vastaavaan ohjelmistoratkaisuun. Työn toinen mittarijoukko koostui ohjelmiston ylläpidettävyydestä. Ylläpidettävyyttä seurattiin koodirivien määrällä sekä empiirisellä tunteella koodin luettavuudesta.

Tämän kandidaatintyön tutkimuskysymykset olivat:

1) Soveltuuko tietovirtaohjelmointimalli jatkuvaan reaaliaikaiseen mittaukseen?
2) Mitä on huomioitava tietovirtaohjelmointimallia käyttäessä?

Työn soveltuvuudella kuvattaan mallin toimimista järjestelmässä, jossa tietoa tuotetaan erillisesti jokaisella mittaushetkellä. Mallin toteuttava ohjelma on oltava tarpeeksi tehokas, luotettava ja yhteensopiva tietoa työntäviin mittareihin. Mallia käyttävä ohjelma soveltui tietovirtojen käsittelyyn solmujen itsenäisen toiminnan ja luonnollisen rinnakkaisuuden takia. Tulokset tarkoittavat, että mallia voidaan hyödyntää jatkuvassa reaaliaikaisessa mittausympäristössä kyseisillä rajauksilla, järjestelmillä ja laitteistolla.

Toinen tutkimuskysymys seurasi tietovirtaohjelmointimallin toteutuksen ohjelmointivaihetta. Työn tärkeimpänä huomiona keskitytään tietueiden määrään ja rakenteen selkeyteen. Tietueiden määrän takia on tietueiden oltava arvoja, arvojoukkoja tai vain luettavia olioita, jotta rinnakkaiset solmut muuta toisiensa tuloksia. Tulosteessa voidaan käyttää vain yhtä varausta vähentäen muistinvarauksien määrää. Muistinvarauksia voidaan vähentää niin, että solmujen välinen viestintä toteutetaan kierrätettävillä viesteillä. Siten vähennetään tietovirtamallin käyttämisestä johtuvaa muistinpainetta.

Tietovirtamallin käyttäminen tuo ohjelmistoon uusia virhepaikkoja ja kokonaan uuden virheluokan. Tietovirtamallin solmujen välissä on oltava puskurit solmujen omatoimisuuden takia, jolloin solmun jumittuessa kyseisen solmun puskuri täyttyy ja kaikki solmusta riippuvat solmut nälkiintyvät täyttäen omat puskurinsa. Tällä tavalla ohjelmisto voi helposti viedä huomaamatta koko järjestelmän muistin ja lopuksi kaatua. Uusi virheluokka esiintyy tilanteissa, jossa solmu saa ainoastaan osan syötteistään laskettavaksi. Tällöin on ilmoitettava laskun olevan vajava ja virhetila käsiteltävä sopivasti.

Työn lopputulema esittää tietovirtamallin olevan sopiva käyttöön sille projektissa varatuilla resursseilla sekä mallina kykenevä toimimaan jatkuvassa reaaliaikaisessa ympäristössä. Loppupäätelmä avaa tien muille suuremman laskennan vaativille sovelluksille kyseistä mallia käyttäen. Lisäksi kehitysvaiheessa nousseita huomioita ja kehityssuunnitelmia keräytyi niin seuraavaan hankkeeseen, sekä taidoksi tutkijoille.

Aikaisemmassa tutkimuksessa tietovirtalaskenta on ulkoistettu erillisille laitekiihdyttimille tai erillisille sulautetuille FPGA- tai ASIC-järjestelmille. Itse tietovirran käyttämistä osana ohjelman rakennetta tukee monet tutkitut ohjelmointikehykset, mutta nyt on todistettu, että tässä käyttöympäristössä on hyödyllistä käyttää kyseistä rakennetta.

Jatkossa olisi mielenkiintoista selvittää tietovirtamallin laskennallisten resurssien käyttöä verrattuna muihin ohjelmistomalleihin sekä luoda staattisia analysaattoreita projektissa käytetyn ohjelmistokielen ja -ympäristön tueksi, jotta tietovirtamallin käyttö olisi tehokkaampaa.

Lähteet

[1] ”PyTorch”. https://pytorch.org/ (viitattu joulu 29, 2022).

[2] ”TensorFlow”. https://www.tensorflow.org/ (viitattu joulu 29, 2022).

[3] ”Data Pipeline - Managed ETL Service - Amazon Data Pipeline - AWS”. https://aws.amazon.com/datapipeline/ (viitattu joulu 29, 2022).

[4] ”Business Application Platform | Microsoft Power Platform”. https://powerplatform.microsoft.com/en-us/ (viitattu joulu 29, 2022).

[5] ”Dataflow | Google Cloud”. https://cloud.google.com/dataflow/ (viitattu joulu 29, 2022).

[6] P. Cappellari, M. Roantree, ja S. A. Chun, ”Optimizing data stream processing for large-scale applications”, *Softw Pract Exp*, vsk. 48, nro 9, ss. 1607–1641, syys 2018, doi: 10.1002/SPE.2596.

[7] V. Milutinović, J. Salom, N. Trifunovic, ja R. Giorgi, ”Guide to DataFlow Supercomputing”, ss. 1–39, 2015, doi: 10.1007/978-3-319-16229-4_1.

[8] G. Ortiz, I. Castillo, A. Garcia-De-Prado, ja J. Boubeta-Puig, ”Evaluating a Flow-Based Programming Approach as an Alternative for Developing CEP Applications in IoT”, *IEEE Internet Things J*, vsk. 9, nro 13, 2022, doi: 10.1109/JIOT.2021.3130498.

[9] W. A. Najjar, E. A. Lee, ja G. R. Gao, ”Advances in the dataflow computational model”, *Parallel Comput*, vsk. 25, nro 13–14, ss. 1907–1929, joulu 1999, doi: 10.1016/S0167-8191(99)00070-8.

[10] D. Del *ym.*, ”A generic parallel pattern interface for stream and data processing”, 2017, doi: 10.1002/cpe.4175.

[11] P. Basanta-Val, N. Fernández-García, L. Sánchez-Fernández, ja J. Arias-Fisteus, ”Patterns for Distributed Real-Time Stream Processing”, *IEEE Transactions on Parallel and Distributed Systems*, vsk. 28, nro 11, ss. 3243–3257, marras 2017, doi: 10.1109/TPDS.2017.2716929.

[12] S. A. Mokhov *ym.*, ”Dataflow Programming and Processing for Artists and Beyond”, doi: 10.1145/3355047.3359423.

[13] K. Tanaka, C. Tsujino, ja H. Maeda, ”IoT Software by Dataflow Programming in Mruby Programming Environment”, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vsk. 12252 LNCS, ss. 212–220, 2020, doi: 10.1007/978-3-030-58811-3_15/FIGURES/9.

[14] N. A. A. Latib, A. A. H. A. Rahman, ja M. S. Rusli, ”Design and implementation of bluetooth low energy link layer controller using dataflow

programming", *Lecture Notes in Electrical Engineering*, vsk. 756 LNEE, ss. 593–604, 2021, doi: 10.1007/978-981-16-0749-3_46/TABLES/5.

[15] M. Pelcat, S. Aridhi, J. Piat, ja J.-F. Nezan, "Physical Layer Multi-Core Prototyping", ss. 197–198, 2013, doi: 10.1007/978-1-4471-4210-2_9.

[16] J. Sérot, F. Berry, ja C. Bourrasset, "High-level dataflow programming for real-time image processing on smart cameras", doi: 10.1007/s11554-014-0462-6.

[17] Z. Ul-Abdin ja M. Yang, "A Radar Signal Processing Case Study for Dataflow Programming of Manycores", *J Sign Process Syst*, vsk. 87, ss. 49–62, 2017, doi: 10.1007/s11265-015-1078-1.

[18] V. Milutinovic, M. Kotlar, M. Stojanovic, I. Dundic, N. Trifunovic, ja Z. Babovic, "DataFlow Supercomputing Essentials", ss. 127–148, 2017, doi: 10.1007/978-3-319-66125-4_5.

[19] V. Milutinovic, M. Kotlar, M. Stojanovic, I. Dundic, N. Trifunovic, ja Z. Babovic, "Binary Search in the DataFlow Paradigm", ss. 93–107, 2017, doi: 10.1007/978-3-319-66125-4_3.

[20] V. Milutinović, J. Salom, N. Trifunovic, ja R. Giorgi, "Guide to DataFlow Supercomputing", ss. 73–106, 2015, doi: 10.1007/978-3-319-16229-4_3.

[21] V. Milutinovic, M. Kotlar, M. Stojanovic, I. Dundic, N. Trifunovic, ja Z. Babovic, "DataFlow Supercomputing Essentials", ss. 3–44, 2017, doi: 10.1007/978-3-319-66125-4_1.

[22] H. Yviquel *ym.*, "Embedded Multi-Core Systems Dedicated to Dynamic Dataflow Programs", *J Sign Process Syst*, vsk. 80, ss. 121–136, 2015, doi: 10.1007/s11265-014-0953-5.

[23] F. Carena *ym.*, "The ALICE data acquisition system", *Nucl Instrum Methods Phys Res A*, vsk. 741, ss. 130–162, maalis 2014, doi: 10.1016/J.NIMA.2013.12.015.

[24] "ATLAS High Level Trigger within the multi-threaded software framework AthenaMT", doi: 10.1088/1742-6596/1525/1/012031.

[25] A. Oyegoke, "The constructive research approach in project management research", *International Journal of Managing Projects in Business*, vsk. 4, nro 4, ss. 573–595, syys 2011, doi: 10.1108/17538371111164029/FULL/PDF.

[26] V. Milutinovic, M. Kotlar, M. Stojanovic, I. Dundic, N. Trifunovic, ja Z. Babovic, "DataFlow Supercomputing Essentials", ss. 111–126, 2017, doi: 10.1007/978-3-319-66125-4_4.

[27] A. C. Sena, E. S. Vaz, F. M. G. França, L. A. J. Marzulo, ja T. A. O. Alves, "Graph Templates for Dataflow Programming", *2015 International Symposium on Computer Architecture and High Performance Computing Workshop (SBAC-PADW)*, 2015, doi: 10.1109/SBAC-PADW.2015.20.