Janne Liimatainen

Kvanttikoneohjelmointi

Tietotekniikan kandidaatintutkielma

16. toukokuuta 2016

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Janne Liimatainen

Yhteystiedot: jasaliim@student.jyu.fi

Työn nimi: Kvanttikoneohjelmointi

Title in English: Quantum programming

Työ: Kandidaatintutkielma

Sivumäärä: 24+0

Tiivistelmä: Työssä tutkittiin kvanttiohjelmointikielten nykytilaa. Tavoitteena oli pintapuolisesti kartoittaa millaisia kieliä on olemassa, mitä niiden erot ovat ja miksi niitä on kehitetty, vaikka käytännön kvanttitietokoneita ei vielä olekaan. Tämä tutkimustavoite toteutui hyvin. Johtopäätöksenä voidaan sanoa, että kvanttiohjelmointikieliä on kehitetty suhteellisen paljon kvanttitietokoneiden puuttumiseen nähden, ja ne eroavat monilta osin. Myös niiden kehittämiseen löydettiin paljon hyviä syitä.

Avainsanat: kvanttitietokoneet, kvanttiohjelmointi, kvanttiohjelmointikielet

Abstract: In this paper the current state of quantum programming languages was studied. The objective was to do a survey of existing languages, their differences, and the reasons for their development, even though practical quantum computers do not yet exist. This goal was achieved. As a conclusion we can say that, considering the absence of real quantum computers, there are many quantum programming languages, and they differ on many parts. Many justifiable reasons for their development were also found.

Keywords: quantum computing, quantum programming, quantum programming languages

Taulukot

Sisältö

1 Johdanto

Kvanttitietokoneiden ajatuksen voidaan katsoa lähteneen vuonna 1982 itsenäisesti kahdelta eri henkilöltä, Richard Feynmanilta ja Paul Benioffilta. Feynmanin lähtökohtana oli ajatus, että kvanttitietokoneilla voitaisiin mallintaa paremmin järjestelmiä, joissa kvanttimekaniikka on päävaikuttajana. Klassisilla tietokoneilla useat näistä simulaatioista vaatisivat järjestelmän koon kasvaessa eksponentiaalisesti kasvavan määrän aikaa, kun taas kvanttitietokoneilla ne olisivat huomattavan paljon nopeampia. Benioffin ajatuksena taas oli vastata pienenevien elektroniikkakomponenttien ongelmaan. Elektroniikkapiirit ovat nykyään niin pieniä, että kvanttimekaniikan ilmiöt alkavat vaikuttaa niissä. Kvanttitietokoneessa näitä ilmiöitä voitaisiin hyödyntää. (Grattage 2006)

Kvanttitietokoneita pidetään joidenkin tehtävien suorittamisessa parempina kuin klassisia. Esimerkiksi luvun (alku)tekijöiden löytäminen sekä jotkin salausalgoritmit, jotka voisivat olla moninkertaisesti nopeampia kuin nykyisin. Kvanttilaskenta ei kuitenkaan muuta sitä, mitä tietokoneella voidaan laskea, vaan ainoastaan sen tapaa (Grattage 2006).

Kvanttiohjelmointikielistä on tehty useita aiempia koosteita (mm. Gay 2006; Selinger 2004; Sofge 2008; Unruh 2006), mutta niissä ei ole kaikkia kieliä eivätkä ne ole kovin uusia. Täten tarvetta uudelle, tähän mennessä kehitetyt kvanttiohjelmointikielet kokoavalle tutkimukselle on. Käytännön kvanttitietokoneita ei kuitenkaan ole vielä olemassa, joten voidaan kysyä, miksi niille ylipäätään on kehitetty kieliä. Myös tätä kysymystä tutkielmassa pohditaan.

Tutkimuksen rakenne tästä eteenpäin on seuraavanlainen: Toisessa luvussa esitellään kvanttitietokoneiden teoriaa ja toimintaa, sekä kuvaillaan kaksi tunnettua kvanttialgoritmia. Kolmannessa luvussa tehdään selkoa kvanttikoneiden ohjelmoinnista ja pohditaan sille kehitettävien ohjelmointikielten tarpeellisuutta, sekä kerrotaan yleisesti kvanttiohjelmointikielistä ja niiden kehittämisestä ja vaatimuksista. Neljännessä luvussa esitellään muutamia jo kehitettyjä kvanttiohjelmointikieliä ja pohdi-

taan, miten hyvin ne täyttävät niille esitetyt vaatimukset.

2 Kvanttitietokoneet

Siinä missä nykyisissä tietokoneissa tietoa tallennetaan bitteinä, joiden arvo on joko 0 tai 1, tallennetaan sitä kvanttitietokoneissa kubitteinä, joiden arvo voi olla 0, 1, tai 0 ja 1 samaan aikaan — tätä kutsutaan superpositioksi (arvona voidaan käyttää esimerkiksi fotonin polarisaatiota tai elektronin spiniä). Tämän takia voidaan ajatella, että kvanttitietokone laskee kaikki mahdolliset bittiyhdistelmät samaan aikaan. Kuitenkin kubitteja mitattaessa niiden tila romahtaa tietyllä todennäköisyydellä joko arvoon 0 tai arvoon 1, todennäköisyysamplitudien neliöiden summan ollessa 1. Useampaa kubittia mitattaessa jokaiselle yhdistelmälle on taasen omat todennäköisyytensä. Tästä syystä ne voivat laskea vain yhden kysymyksen kerrallaan, mutta jos se osataan muotoilla oikein, voi saavutettu ratkaisunopeus olla hyvinkin nopea verrattuna klassiseen tietokoneeseen. Kvanttitietokoneet myös skaalautuvat kubitteja lisättäessä todella hyvin, sillä niiden teho tuplaantuu, kun kubitteja tulee yksi lisää (kts. taulukko 1). (Esim. Vizzotto 2013; Selinger 2004)

Kvanttiproseduureja ovat alustus (initialisation), kehitys (evolution), ja lopetus (finalisation). Alustuksessa kubitit asetetaan haluttuun alkutilaan, minkä jälkeen kehitysvaiheessa niille tehdään muunnoksia ajettavan algoritmin mukaan. Lopetusvaiheessa kubitit romahdutetaan ja niiden antama lopputulos, eli jokin bittiyhdistelmä, mitataan (Sanders ja Zuliani 2000). Kvanttibittejä voidaan muokata kahdella tavalla, yhtenäisillä muunnoksilla ja mittauksilla (Selinger 2004). Ensin mainitussa kubitit työnnetään yhtenäismatriisin läpi, jolloin ne saavat uudet tilat. Mittauksessa

kubittien määrä	mahdollisuudet	teho
1	0 tai 1	2
2	00,01,10,11	4
3	000,001,010,011,100,101,110,111	8
N		2^N

Taulukko 1. Kubittien määrän vaikutus kvanttitietokoneen tehoon. (Mukaillen (Vizzotto 2013), s. 10)

niiden tilat taas romahdutetaan klassisiin 0:aan ja 1:een.

2.1 Kvanttitietokoneiden jaottelu

Kvanttitietokoneet voidaan jakaa puhtaisiin ja hybrideihin. Näistä ensin mainitussa koko tietokone on kvanttipohjainen, kun taas jälkimmäisessä ohjataan klassisella tietokoneella jotain kvanttitietokonetta tai -laitetta (Ömer 2005). Koska käytännössä toimivia kvanttitietokoneita ei vielä ole olemassa, käytetään usein hyväksi virtuaalisia laitteistomalleja (Virtual Hardware Models) (Selinger 2004). Selinger (2004) mainitsee näistä kolme: *kvanttipiirimallin*, *QRAM-mallin* ja *kvantti-Turingin koneen*.

Kvanttipiirimallissa (quantum circuit model) on kvanttipiirejä, jotka muodostuvat kvanttiporteista samaan tapaan kuin klassinen logiikkapiiri muodostuu logiikkaporteista. Kahdesta peruskvanttioperaatiosta, eli unitaarinen muunnos (unitary transformation) ja mittaus, tämä malli korostaa ensin mainittua ja mittaukset suoritetaan aina viimeisenä askeleena.

QRAM-mallissa kvanttilaitetta ohjataan universaalilla klassisella tietokoneella. Tässä kvanttilaitteessa on suuri, mutta äärellinen määrä yksittäisiä ja erikseen käsiteltäviä kubittejä samaan tapaan kuin tavallisessa RAM-muistissa on klassisia bittejä. Klassinen ohjaustietokone antaa ohjeita kvanttilaitteelle, joka suorittaa nämä käskyt ja palauttaa mahdolliset mittaustulokset. Tässä mallissa peruskvanttioperaatioita voidaan suorittaa vapaasti missä järjestyksessä vain.

Kolmantena mainittua *kvantti-Turingin konetta* ei kuitenkaan pidetä realistisena tulevaisuuden kvanttitietokoneen mallina. Tässä mallissa mittauksia ei suoriteta koskaan.

Näistä QRAM-malli on perustana useimmille ”klassinen ohjaus - kvanttidata” -tavalla toimiville kvanttiohjelmointikielille, mutta koska kvanttitietokoneissa tullee tapahtumaan paljon virheitä ja laskennat tulee suorittaa nopeasti, jotteivät kvanttitilat romahda, on kvanttipiirimalli kenties todennäköisempi ratkaisu. Siinä kaikki laskut voidaan suorittaa samaan aikaan, mutta toisaalta minkäänlaisia ehtolauseita

ei näin ollen voida suorittaa. Tämä voidaan kuitenkin ratkaista esimerkiksi mahdollisuudella säilyttää joitain mittaamattomia kubitteja pitkäaikaisvarastossa suorituskertojen välissä. Loppujen lopuksi ohjelmointikielen kehittämisen kannalta fyysisellä kvanttiarkkitehtuurilla ei tulisi olla paljoa väliä, sillä korkean tason kielen tarkoituksena on eristää ohjelmoija juuri näistä matalan tason yksityiskohdista. (Green, Lumsdaine, Ross, Selinger & Valiron 2013)

2.2 Tunnettuja algoritmeja

Ensimmäisiä ja tunnetuimpia kvanttialgoritmeja on *Shorin algoritmi*, jonka avulla voidaan laskea luvun alkutekijät eksponentiaalisesti nopeammin kuin klassisilla algoritmeilla. Koska luvun tekijöiden laskemista käytetään hyväksi useissa moderneissa salausjärjestelmissä (kuten RSA-protokollassa), tekisi kvanttitietokone näistä käyttökelvottomia, sillä se voisi purkaa ne sekunneissa. Shorin algoritmia pidetäänkin usein tärkeimpänä kvanttitietokonesovelluksena ja se sai aikaan kasvavaa kiinnostusta kvanttitietokoneiden suuntaan. (Grattage 2006; Vizzotto 2013)

Toinen tunnettu kvanttialgoritmi on *Groverin algoritmi*, jolla voidaan nopeasti etsiä järjestämättömästä tietokannasta tietoa. Siinä missä klassisilla algoritmeilla tämä vie N:n kokoisesta tietokannasta noin N operaatiota, Groverin algoritmi pystyy siihen $\sqrt{N}$ operaatiossa. Myös Groverin algoritmia voitaisiin käyttää joidenkin salausten nykyistä paljon nopeampaan purkamiseen. (Grattage 2006; Vizzotto 2013)

3 Kvanttikoneohjelmointi

Kvanttiohjelmointi on tällä hetkellä suunnilleen samalla tasolla kuin klassinen ohjelmointi oli 1940-luvulla, ja laitteisto on olematonta tai toimimatonta (Lampis, Ginis, Papakyriakou & Papaspyrou 2008). Joidenkin arvioiden mukaan käytännöllinen kvanttitietokone voisi olla todellisuutta vuoteen 2030 mennessä (Xu ja Song 2008). Toimivien kvanttitietokoneiden vielä puuttuessa voidaankin kysyä miksi kvanttiohjelmointikieliä edes kannattaa tutkia ja kehittää, kun niille sopivaa laitteistoa ei ole olemassa. Siihen on kuitenkin useita perusteluja.

Vizzotto (2013) mainitsee, että klassistenkin ohjelmointikielten tutkimus alkoi paljon ennen yleiskäyttöisen klassisen tietokoneen kehittämistä. Samaa aihetta sivuten Gay (2006) muistuttaa, että ohjelmointikielten, joilta puuttuu vankka semanttinen pohja, käyttö on aiheuttanut paljon ongelmia ohjelmistotekniikassa. Käytännön tietojenkäsittelyteknologiat ovat menneet teoreettisten perustojen edelle ja vasta viime aikoina yleisimmillä ohjelmointikielillä on myös hyvin ymmärretty teoreettinen perusta (Gay 2006). Tältä kannalta katsottuna kvanttiohjelmointikielten kehittäminen ennen käytännön sovelluksia on jopa ideaali tilanne (Gay 2006).

Unruh (2006) tuo esille kolme syytä: Ohjelmointikielten avulla voitaisiin tutkia kvanttialgoritmeja teoreettisesti paremmin kuin pseudokielellä tai esittämällä algoritmi suoraan kvanttipiireinä. Toisena hän mainitsee kvanttialgoritmien kokeellisen tutkimisen, esimerkiksi simulaatioiden avulla. Näin voitaisiin tutkia algoritmeja, joita ei voida todistaa teoreettisesti esimerkiksi niiden sisältämien, todistamattomien matemaattisten oletusten takia. Lopuksi Unruh (2006) mainitsee kvanttikryptografisten protokollien määrittelyn ja verifikoinnin.

Lisäksi Gay (2006) huomauttaa markkinoilla olevan jo kvanttikryptografiaan liittyviä komponentteja, joiden käytössä tarvitaan uudenlaista kvanttiohjelmointia. Tämän lisäksi kvanttiohjelmoinnin tutkiminen on antanut uuden näkökulman kvanttiteoriaan itseensä ja siitä voi näin ollen olla hyötyä, vaikkei toimivia kvanttitietokoneita koskaan saataisikaan kehitettyä (Gay 2006).

3.1 Kvanttiohjelmointikielten kehittäminen

Ohjelmointiparadigmoja ovat *imperatiivinen* ja *deklaratiivinen eli funktionaalinen*. Ensin mainittu pyrkii esittämään miten jokin ohjelma toimii, ja siinä käytetään lausekkeita muuttamaan ohjelman globaalia tilaa tai muuttujien arvoja. Imperatiiviset kielet ovat helppoja kääntää käyttämään QRAM-mallia (Selinger 2004). Deklaratiivinen ohjelmointi taas pyrkii esittämään mitä ohjelman halutaan tekevän, ja siinä sisääntulevia funktioita muutetaan jollain tavalla, minkä jälkeen ne laitetaan ulostuloon. Kvanttiohjelmointikieliin funktionaalinen lähestymistapa sopii suhteellisen hyvin, sillä sillä on perin suora operationaalinen merkitysoppi (operational semantics) (Mlnarik 2007). Kokemusten perusteella funktionaalisuus onkin kvanttiohjelmointikieleen sopivampi, mutta sen koodia on vaikeampaa lukea kuin imperatiivisissa kielissä — tämä kuitenkin riippuu pitkälti lukijan matemaattisesta osaamisesta (Xu & Song 2008). Kvanttiohjelmointikieliin onkin kaksi eri lähestymistapaa: toisaalta tarvitaan kieliä, joilla tehdyt algoritmit voidaan todistaa oikeellisiksi; toisaalta taas kieliä, joilla käytännön kvanttitietokoneita voisi ohjelmoida tehokkaasti ja helposti (Unruh 2006). Karkeana jakona voitaisiin pitää sitä, että ensin mainitut, formaalit, kielet olisivat funktionaalisia ja jälkimmäiset taas imperatiivisia. Tavoitteena voitaisiin pitää kieltä, joka yhdistäisi nämä kaksi lähestymistapaa: sama kieli sopisi siis sekä itse ohjelmointiin ja ohjelmien suorittamiseen, että näiden ohjelmien analysointiin ja niiden oikeellisuuden todistamiseen (Unruh 2006).

Vaikka kunnollisia kvanttitietokoneita ei vielä olemassa olekaan, on niille silti kehitetty useita ohjelmointikieliä. Useimmat näistä perustuvat johonkin klassisen tietokoneen ohjelmointikieleen, johon on lisätty päälle kvanttiosioita. Lisäksi ne useimmiten toimivat hybridiperiaatteella, eli vain kvanttilaskenta suoritetaan kvanttitietokoneella ja kaikki muu, kuten ohjelman rakenteet (esimerkiksi silmukat ja ehtolauseet), suoritetaan klassisessa tietokoneessa (esim. Selinger 2004).

Kvanttiohjelmointikielien kehittämiseen liittyy muutamia ongelmia, muun muassa kvanttiteorian itsensä keskeneräisyys, puute kvanttitietokoneista, joilla kvanttialgoritmeja voisi ajaa, sekä kvanttikoneiden käytännön sovelluksien vähäinen määrä (Sofge 2008).

Kvanttiohjelmointikielien kehittämisen haasteita on muun muassa kehittää jollekin korkean tason kielelle esittämisen merkitysoppi (denotational semantics), johon kuuluisivat myös klassiset ominaisuudet ja mittaukset. Tähän mennessä syntaksin ja merkitysopin välille ei ole syntynyt kunnon yhteyttä, ja sekin mitä on, koskee yleensä vain osaa kielestä. Toinen haaste on kehittää teoriaa kvanttiprosessien samanaikaisuudelle, ja klassisen sekä kvanttidatan vaihdolle. Kolmantena haasteena ja tutkimuskohteena on kvanttiohjelmointikielien käyttäminen epätäydellisissä laitteistoissa. Ensimmäiset oikeat kvanttitietokoneet ovat todennäköisesti alttiita satunnaisille virheille ja dekoherenssille, eli hiukkasten lämpöliikkeelle, jolla ne romahduttavat lähellä olevat hiukkaset pois superpositiosta. Kvantti-informaatiolle on olemassa virheenkorjaustekniikoita, mutta nähtäväksi jää, miten ne saadaan automatisoitua esimerkiksi kääntäjään tai käyttöjärjestelmään, tai vaativatko jotkin algoritmit niille erikseen suunniteltuja virheenkorjaustapoja. (Selinger 2004)

3.2 Kvanttiohjelmointikielten vaatimukset

Kvanttiohjelmointikielille on niitä suunnitellessa asetettu erilaisia vaatimuksia, joista Bettelli, Calarco & Serafini (2003) mainitsevat *täydellisyyden, klassisen jatkeen, erotettavissa olevuuden, ilmaisuvoimaisuuden ja laitteistoitsenäisyyden*. *Täydellisyydellä* he tarkoittavat, että kielen täytyy olla tarpeeksi vahva ilmaisemaan kvanttipiirimallia. Sillä täytyy siis olla mahdollista esittää jokainen validi kvanttialgoritmi ja jokaisen koodinpätkän täytyy vastata jotain validia kvanttialgoritmia.

Klassisen jatke merkitsee sitä, että kielen täytyy sisältää ja olla lisänä korkean tason klassisen laskennan paradigmaan, jotta sen avulla voidaan yhdistää kvanttilaskenta ja klassinen esi- ja loppuprosessointi mahdollisimman pienellä vaivalla. Kielet, jotka on rakennettu varta vasten kvanttiohjelmointiin eivätkä sisällä klassisia elementtejä, jäisivät jälkeen kun tavalliset ohjelmointiteknologiat kehittyvät. (Bettelli ym. 2003)

Erotettavissa olevuus sisältää ajatuksen, että kielen täytyy pitää klassinen ja kvanttiohjelmointi erillään, jotta klassiselle tietokoneelle voidaan siirtää suoraan kaikki laskennat, jotka eivät tarvitse tai eivät hyödy kvanttilaitteella laskemisesta. *Ilmaisu-*

voimaan kuuluu se, että kielen pitää tarjota korkean tason käsitteitä, jotka tekevät kvanttialgoritmien ohjelmoimisen prosessin samankaltaiseksi kuin tutkimusartikkeleissa esitetty pseudo-koodi ja tuovat sen lähemmäksi ohjelmoijan tapaa ajatella. Lisäksi kielen pitää sallia automatisoitu ja skaalautuva menetelmä korkean tason kielen kääntämiseen ja haluttaessa optimoimiseen matalan tason konekäskyiksi kvanttitietokoneelle. *Laitteistoitsenäisyys* tarkoittaa, että kielen pitää olla itsenäinen laitteistosta, jolla sitä lopulta tullaan ajamaan. Tällöin ohjelma voidaan kääntää eri kvanttiarkkitehtuureille ilman ohjelmoijan väliintuloa. (Bettelli ym. 2003)

Myös Unruh (2006) mainitsee vaatimuksiksi *yksinkertaisuuden ja tehokkuuden* sekä *teknologiaitsenäisyyden*. Lisäksi hän mainitsee, että kieleen tulisi sisältyä *huomaamattomia optimointi- ja virheenkorjaustekniikoita* ja kielellä tehtyjä ohjelmia tulisi *pystyä ajamaan simulaattorissa*. (Unruh 2006)

Erikseen vaatimuksista mainitaan usein myös *helppokäyttöisyys, ymmärrettävyys ja yksinkertaisuus*: pitää pystyä esimerkiksi lisäämään kvanttiluku toiseen kvanttilukuun ja saada ulos kolmas kvanttiluku ilman, että tarvitsee miettiä operaation onnistumista tai kvanttimekaniikan rajoituksia (Lampis ym. 2008). Samoin *tuttuutta* pidetään tärkeänä ja useat kvanttiohjelmointikielet perustuvatkin johonkin klassiseen ohjelmointikieleen (Mlnarik 2007).

4 Kehitetyt kvanttiohjelmointikielet

Tässä luvussa imperatiivisiin ja funktionaalisiin jaoteltuja kvanttiohjelmointikieliä esitellään pintapuolisesti. Näiden kielien taso vaihtelee jonkin verran. Joihinkin kieliin löytyy vankka teoriapohja, semantiikkaa, kääntäjä ja algebraa, kun taas joistakin puuttuu näistä osa tai kaikki.

4.1 Imperatiiviset kielet

Tässä luvussa esitellään kielet QCL, NDQJava, NDQFP, Scaffold, LanQ ja qGCL, jotka ovat kaikki imperatiivisia ohjelmointikieliä, lukuunottamatta funktionaalista kieltä NDQFP. Se esitellään tässä sen vuoksi, että sen kehittäminen liittyy läheisesti NDQJavaan.

QCL (Quantum Computation Language) on imperatiivinen kvanttiohjelmointikieli, joka noudattaa ”kvanttidata, klassinen ohjaus” -paradigmaa (Ömer 1998). Se ei perustu mihinkään klassiseen ohjelmointikieleen, mutta syntaksi on klassisten proseduraalisten kielien, kuten C ja Pascal, kaltaista (Sanders & Zuliani 2000). Sillä on tulkki, joka pystyy implementoimaan ja simuloimaan kaikki tunnetut kvanttialgoritmit (Sanders & Zuliani 2000). Sillä on toteutettu Shorin algoritmi (Ömer 1998). Se oli ensimmäisiä kvanttitietokoneille luotuja ohjelmointikieliä ja se on ollut perustana tai innoittajana monille muille sen jälkeen tulleille kvanttiohjelmointikielille. Siitä huolimatta, että sillä vaikuttaisi olevan vankka teoreettinen pohja, muiden kielien kehittäjät eivät ilmeisesti pidä sitä riittävänä.

Xu & Song (2008) esittelevät artikkelissaan ”Quantum programming languages” kielet *NDQJava* ja *NDQFP*. Näiden kielten kehittäminen aloitettiin vuonna 2005 ja kehittämisen motivaationa on ollut ajatus, että aiemmat kvanttiohjelmointikielet QCL ja QML ovat liian monimutkaisia eivätkä täten täytä ohjelmointikielen yksinkertaisuuden vaatimusta. Koska kvanttialgoritmit ovat deklaratiivisia, mutta klassiset tietokoneet imperatiivisia, päättivät he suunnitella sekä imperatiivisen että funktionaalisen kvanttiohjelmointikielen. (Xu & Song 2008).

NDQJava on imperatiivinen, Java-ohjelmointikieleen perustuva kvanttiohjelmointikieli. Se koostuu klassisesta osasta ja kvanttiosasta. Ensin mainittu on puhdasta Javaa, jälkimmäisessä on kvanttidatatyyppi QType, kvanttimuuttuja, kvanttimääritelmä ja kvanttilauseke. (Xu & Song 2008) *NDQFP* on funktionaalinen ja modulaarinen, FP-ohjelmointikieleen perustuva kvanttiohjelmointikieli. (Xu & Song 2008)

Abhari, Faruque, Dousti, Svec, Catu, Chakrabati, Chiang, Vanderwilt, Black, Chong, Martonosi, Suchara, Brown, Pedram & Brun (2011) esittelevät *Scaffold*-kvanttiohjelmointikielen, jonka tarkoituksena on helpottaa monia eri osia, sekä klassisia että kvanttisia, sisältävän kvanttialgoritmin esittämistä. Tämän lisäksi sen avulla on tarkoitus saada lopputulos, josta resurssien tarvetta ja muita analyysejä on helppo tehdä. Scaffoldin koodin klassinen osa on hyvin lähellä C-kielen koodia ja kvanttiosuudet ovat tässä kielessä erityisiä funktiokutsuja. (Abhari ym. 2011)

Tutkijoiden Abhari ym. (2011) mukaan Scaffoldiin haluttiin seuraavat ominaisuudet: *täydellisyys ja ilmaisukyvykkyys, tuttuus ja helppokäyttöisyys, integraatio, ja valmiiden kääntäjien käyttö*. *Täydellisyys ja ilmaisukyvykkyys* tarkoittaa, että pitäisi olla mahdollista ilmaista kokonaisia kvanttiohjelmia eikä kieli saisi rajoittaa mitään osia niistä. Sen tulisi tarjota käyttökelpoisia ja ilmaisuvoimaisia toimintoja, joilla ohjelmoija voisi suhteellisen helposti ilmaista tavallisia kvanttilaskennan tekniikoita. Koska kvanttialgoritmit usein ilmaistaan kvanttiporttijonoina (quantum gate sequences) tai imperatiivisena koodeina (imperative codes), kielen pitää mahdollistaa laskujen esittäminen näillä tavoin. Esimerkiksi kvanttialgoritmeissa usein esiintyvät kvanttioraakkelit (quantum oracle) ilmaistaan klassisella koodilla, joten nämä tulisi olla helppoja esittää. Samoin kvanttialgoritmi voi olla esitetty piirinä, joka myös pitää olla ilmaistavissa helposti. Näin ollen kieleen tehtiin C2QG-moduuli (Classical code to Quantum Gates), joka sallii ohjelmoijan esittää haluttu kvanttitoiminta klassisella koodilla. Ohjelmoijan ei siis tarvitse esittää algoritmia kvanttiporttien avulla, vaan voi esittää sen korkeammalta tasolta.

Tuttuus ja helppokäyttöisyys merkitsee sitä, että koska kvanttilaskenta on jo itsessään tarpeeksi vaikeaa, ei kieli saisi sitä enää vaikeammaksi tehdä. Jos käyttäjä tuntee kvanttilaskennan perusteet ja osaa ohjelmoida klassisilla kielillä, tulisi Scaffoldin

oppimisen ja käytön sujua helposti. Ratkaisuna käytetään tuttuja ratkaisuja, kuten syntaksia ja ohjelmointiparadigmoja, aina kun se on mahdollista, esimerkkinä C- ja C++ -kielien rakenteet. Täten Scaffoldista päätettiin tehdä imperatiivinen kieli, koska sen käyttökohteet tarvitsevat muun muassa silmukoita, iteratiivisia laskuja ja argumentteja käyttäviä funktiokutsuja, ja nämä kaikki sopivat hyvin imperatiivisuuteen. Lisäksi koska kieleen kuuluu klassinen ohjausosa, on siinä hyvä käyttää tuttuja rakenteita. Scaffoldissa on vaikutteita muun muassa korkean tason imperatiivisista ohjelmointikielistä C/C++ ja Java, laitteiston mallinnus -kielestä Verilog, ”C:stä laitteistoon” -kielestä (C-to-hardware) System-C, ja aikaisemmasta kvanttiohjelmointikielestä QCL. Erityisesti syntaksi on lähellä C-kieltä, koska se on tuttua suurimmalle osalle ohjelmoijista, ja täten toteuttaa vaatimusta tuttuudesta ja helppokäyttöisyydestä. Toisaalta C-kielen ei-tarvittujen ominaisuuksien, kuten osoittimien, käyttöä minimoidaan ja estetään bugien välttämiseksi. Tämä toteutetaan mukautetulla kääntäjän toiminnalla, joka kääntämisvaiheessa tarkistaa syntaksia hyvin voimakkaasti.

Edelleen Abhari ym. (2011) mukaan *integraatio* tarkoittaa, että kielen pitäisi sopia hyvin yhteen kvanttitietokoneen sisältävän laitteistoketjun kanssa. Sen pitäisi mahdollistaa algoritmien korkean tason matemaattisten ilmaisumuotojen muuttaminen matalan tason koodiksi saumattomasti ja niin, että laitteisto pystyy sen antaman tiedon perusteella kubitit, kvanttiportit ja muut piirin komponentit rakentamaan. Lisäksi kääntäjä ei saisi olla riippuvainen laitteistosta vaan sen pitäisi pystyä toimimaan erilaisten kvanttitietokonearkkitehtuurien kanssa. Näin ollen kielestä tehtiin modulaarinen, mikä mahdollistaa klassisen koodin käsittelyn erikseen kvanttikoodista.

Valmiiden kääntäjien käytön vaatimus näkyy siinä, että koodissa, joka ei eroa klassisesta, pitäisi voida käyttää valmiita kääntäjiä sekä ajan säästämiseksi, että koska ne ovat toimiviksi todettuja. Näin ollen syntaksin pitää olla lähellä jotain klassista ja paljon käytettyä ohjelmointikieltä. Valmiiden kääntäjien käyttö ei kuitenkaan saa rajoittaa kvanttikielen kehitystä, vaan sitä tulee tarpeen mukaan muokata. Koska Scaffoldin syntaksi on lähellä C-kieltä, voidaan sen kääntäjiä käyttää lähtökohtana.

Mlnarik (2007) esittelee imperatiivisen kvanttiohjelmointikielen *LanQ*, joka on suunniteltu tukemaan kvantti- ja klassisen ohjelmoinnin yhdistelmää, sekä prosessien perusoperaatioita — prosessin luontia ja prosessien välistä kommunikaatiota. Monen muun kielen tapaan myös sen syntaksi muistuttaa C-kielen syntaksia. Sillä on vankka teoreettinen perusta ja sillä voidaan todistaa implementoidun kvanttialgoritmin oikeellisuus (Mlnarik 2007).

Kun Dijkstran GCL-kieleen (Guarded-command language) lisätään probabilismi, saadaan aikaan ohjelmointikieli pGCL. Kun tähän lisätään vielä mahdollisuus kutsua kvanttiproseduureja, saadaan aikaan *qGCL*. Koska GCL soveltuu ohjelmien oikeellisuuden tarkistamiseen ja sille on olemassa tarkkaan määritelty teoria ja formaali semantiikka, ovat nämä ominaisuudet myös qGCL:llä. Täten myös sillä on vankka pohja. Sillä voidaan ohjelmoida ”universaali” kvanttitietokone ja sillä on toteutettu sekä Shorin että Groverin algoritmit. (Sanders & Zuliani 2000; Bettelli ym. 2003; Zuliani 2004)

qGCL:stä löytyy muun muassa seuraavat ominaisuudet (Sanders & Zuliani 2000): *Ilmaisuvoima*, eli sillä voidaan esittää olemassaolevat kvanttialgoritmit. *Yksinkertaisuus*, eli kieli vaikuttaa olevan niin yksinkertainen kuin mahdollista, silti sisältäen nondeterminismin ja todennäköisyyden. *Abstraktio*, eli kielen ohjaus- ja datarakenteet ovat samalla abstraktiotasolla kuin nykyisten imperatiivisten klassisten kielten. *Laskenta (calculation)*, eli kielellä on formaali semantiikka, hyvässä kunnossa olevat lait, ja se tarjoaa kehittelylaskennan (refinement calculus), joka tukee kvanttiohjelmien verifikointia ja derivointia. Lisäksi kieli tarjoaa *”observoinnin” yhtäläisen käsittelyn* (uniform treatment of ”observation”).

4.2 Funktionaaliset kielet

Tässä luvussa käsitellään funktionaalisia kieliä QML, Quipper, nQML ja QPL. Funktionaalinen kieli *NDQFP* on poikkeuksellisesti esitelty imperatiivisten NDQJava-kielen yhteydessä.

Altenkirch ja Grattage (2005) esittelevät funktionaalisen kvanttiohjelmointikielen *QML*, jonka suurimpia eroja useimpiin muihin kieliin on se, että siinä datan lisäksi myös ohjausrakenteet ovat kvanttipohjaisia. Sen kehittämisen motivaationa on ollut mahdollistaa kvanttialgoritmien esittäminen korkealla tasolla, matalan tason kvanttipiirien ja muiden vastaavien sijasta (Altenkirch & Grattage 2005).

QML:lle on lisäksi esitelty sen teoria ja algebra (Altenkirch, Grattage, Vizzotto & Sabry 2005), sekä luotu kääntäjä (Grattage & Altenkirch 2005). Näin ollen se on kvanttiohjelmointikielistä yksi valmiimpia ja vankimmalla pohjalla olevia. Toisaalta se ei perustu mihinkään tunnettuun ohjelmointikieleen vaan on luotu alusta asti itse, ja tästä syystä se ei varmaankaan ole helpoimmasta päästä oppia.

Green ym. (2013) esittelevät funktionaalisen, skaalautuvan ja ilmaisuvoimaisen kvanttiohjelmointikielen *Quipper*in, joka pystyy luomaan kvanttiporttiesityksiä, joissa on biljoonia portteja. Skaalautuvuus tarkoittaa tässä sitä, että kieli ei jää vain esimerkkialgoritmien ja pienten todistuksien tasolle, vaan sillä pystyy esittämään oikeita, kirjallisuudessa esitettyjä kvanttialgoritmeja, jotka ovat näitä monta kerta-astetta monimutkaisempia. Tekijöiden mukaan useat muut kielet eivät tähän pysty. Quipper suunniteltiin oikeellisuus, skaalautuvuus ja käytettävyys edellä. Useimpien muiden kielten tapaan sekin on suunniteltu systeemiin, jossa klassinen tietokone ohjaa kvanttilaitetta, eikä se ole riippuvainen mistään tietystä kvanttilaitteiston mallista. Quipper on osaltaan piirikuvauskieli ja se on suunniteltu tukemaan sekä yksittäisten porttien että kokonaisten piirien operaatioiden esittämistä luonnollisella tavalla. (Green ym. 2013)

Quipper on upotettu kieli, eli se perustuu johonkin jo olemassa olevaan ohjelmointikieleen, tässä tapauksessa Haskelliin. Se voidaan nähdä kokoelmana datatyyppejä ja kombinatoreja (combinators), sekä funktiokirjastona Haskellin sisällä. Näiden lisäksi sillä on idiomi, eli suositeltu tyyli kirjoittaa sulautettuja (embedded) ohjelmia. Haskell valittiin siksi, että osa Quipperin ominaisuuksista pystytään tekemään Haskellin valmiilla työkaluilla, ja ne ovat molemmat vahvasti tyypitettyjä funktionaalisia kieliä. Toisaalta Haskellista puuttuu kaksi ominaisuutta, jotka olisivat hyödyllisiä Quipperille: lineaariset tyypit ja riippuvaiset tyypit (dependent types). Täs-

tä syystä Quipperille saatetaan tulevaisuudessa tehdä erillinen kääntäjä tai ainakin tyyppitarkastaja. (Green, Lumsdaine, Ross, Selinger & Valiron 2013)

Kielen toimivuudesta kertoo se, että sillä on ohjelmoitu seitsemän ei-triviaalia kvanttialgoritmia, jotka käyttävät laajaa kirjoa kvanttiprimitiiveistä (quantum primitives) ja antavat täten hyvän kuvan nykyisistä kvanttialgoritmeistä. (Kvanttiprimitiivi on jokin kvanttialgoritmeissä yleisesti käytetty ”alkukantainen” kvanttirakennuspalikka, kuten kvantti-”Fourier’n muunnos”.) Lisäksi sillä on hyvässä kunnossa oleva teoreettinen pohja. (Green ym. 2013)

Lampis ym. (2008) esittelevät QML:ään perustuvan kvanttiohjelmointikielensä *nQML*:än, joka pyrkii olemaan sitä yksinkertaisempi kieli. QML:n tavoin myös se perustuu paradigmaan "kvanttidata ja -ohjaus". Se myös estää käyttäjää rikkomasta kvanttilaskennan sääntöjä ja lakeja, mutta sallii kvanttimittaukset missä ohjelman osassa tahansa. Sen avulla yhtenäisiä muunnoksia voi esittää helposti ohjelmana, käyttäen pitkälti samoja merkintätapoja kuin kvanttialgoritmien suunnittelijoilla. Sillä on toteutettu Groverin algoritmi. (Lampis ym. 2008)

Selinger (2004) esittelee kehittämänsä kvanttiohjelmointikielen *QPL*:n, joka noudattaa ”kvanttidata, klassinen ohjaus” -paradigmaa. Se oli ensimmäinen funktionaalinen kvanttiohjelmointikieli (Ying ja Feng 2011), mutta näyttää enemmän imperatiiviselta (Lampis ym. 2008). Siinä on staattinen tyyppisysteemi, mikä takaa ajonaikaisten virheiden puuttumisen (Selinger 2004), ja se sisältää korkean tason ominaisuuksia, kuten silmukoita ja rekursiota (Lampis ym. 2008). Lisäksi se on esitetty vuokaaviokielenä (flow chart language) (Selinger 2004).

4.3 Vertailua

Kielet täyttävät luvun alussa esitettyjä vaatimuksia vaihtelevasti. *Täydellisyys* kaikista kielistä vaikuttaisi löytyvän, vaikka sen todistusta ei kaikkien kielien kohdalla olekaan esitetty. Toisilla, kuten Quipperilla, tämä todistaminen on suoritettu ohjelmoimalla kvanttialgoritmeja ja näyttämällä niiden toimivuus; toisilla, kuten qGCL:llä, todistaminen on teoreettisempaa, ja periaatteessa näin ollen myös oikeel-

lisempaa. Toisaalta se, että käytännön kvanttialgoritmeja voidaan ohjelmoida ja ne myös toimivat, kertoo kielen käytettävyydestä kenties enemmän. Tällöin teorian pitää kuitenkin pysyä perässä ja tarpeeksi hyvällä tasolla eikä jäädä jälkeen, mikä on monien klassisten kielten kohdalla esiin tullut ongelma.

Klassisen jatketta eivät kaikki kielet toteuta. Suurin osa käyttää syntaksinaan jotain tuttua kieltä, useimmiten C/C++:aa, mutta on myös kieliä, joilla on ihan oma syntaksinsa. Esimerkiksi Scaffoldissa on vaikutteita mainituista C-kielistä ja Quipper on upotettu Haskelliin, mutta QML:llä on oma, joskin hyvin yksinkertainen ja tuttu, syntaksinsa.

Erotettavissa olevuutta voi soveltaa vain kieliin, jotka noudattavat ”klassinen ohjaus, kvanttidata” -paradigmaa, minkä suurin osa kielistä tekeekin. Yleisin toteutustapa on jonkinlainen modularisointi, eli kvanttilaskentaa varten on erityisiä funktiokutsuja, ja vain niiden tarvitsema data liikkuu kvanttitietokoneeseen ja takaisin.

Ilmaisuvoiman vaatimus on toteutettu useimmiten niin, että kielen syntaksi ja rakenteet ovat lähellä tuttuja klassisia korkean tason ohjelmointikieliä. Kaikki kielet eivät näin kuitenkaan tee, mutta toisaalta kovin matalan tason kieliä ei koosteesta löydy.

Laitteisto- ja teknologiaitsenäisyys tuntuu olevan kaikilla kielillä, vaikka jotkin esimerkiksi olettavat tietynlaisia muistiarkkitehtuureja. Koska käytännön kvanttikoneita ei kuitenkaan vielä ole olemassa, ei voida sanoa onko yksikään kieli laitteistosta riippumaton. Toisaalta laitteistoriippumattomuus riippuu pitkälti kääntäjästä, jota läheskään kaikilla kielillä ei vielä ole. *Simulaattorissa ajamisen mahdollisuus* löytyy ainakin kielistä QML, Quipper, nQML, LanQ ja QCL (Altenkirch & Grattage 2005; Green ym. 2013; Lampis ym. 2008; Mlnarik 2007; Ömer 1998).

Huomaamaton virheenkorjaus puuttuu käytännössä kaikilta kieliltä. Selinger (2004) mainitsee, että virheenkorjausta voitaisiin tehdä useassa eri vaiheessa: se voisi olla sisäänrakennettuna laitteistossa, käyttöjärjestelmä voisi huolehtia siitä, tai kääntäjä voisi tehdä sen automaattisesti. Näin ollen virheenkorjauksen puuttumista itse kielistä ei välttämättä tarvitse pitää puutteena, sillä näin ne toteuttavat helppokäyttöisyyttä, koska ohjelmoijan ei itse tarvitse huolehtia virheenkorjauksesta.

Helppokäyttöisyys, ymmärrettävyys, yksinkertaisuus ja tuttuus toteutuvat useissa kielissä suurelle osalle ohjelmoijia puhtaasti siksi, että ne käyttävät C/C++:n ja muiden tuttujen klassisten kielten syntaksia ja rakenteita.

5 Yhteenveto

Tässä tutkielmassa on käyty läpi lyhyesti kvanttitietokoneiden perusteita ja tärkeimpiä kvanttialgoritmeja. Erityisesti läpi käytiin kvanttiohjelmointia ja siihen kehitettyjä ohjelmointikieliä, joista annettiin pieni katsaus. Kvanttiohjelmointi on vielä pieni ala, johtuen erityisesti käytännöllisten kvanttitietokoneiden puutteesta, mutta siitä huolimatta sille on kehitetty useitakin kvanttiohjelmointikieliä. Niiden kehittämisen järkevyyttä ennen kvanttitietokoneiden olemassaoloa voidaan kyseenalaistaa, mutta sille löydettiin useita hyviä perusteluja.

Katsauksessa esitellyt kvanttiohjelmointikielet jakautuivat tasaisesti funktionaalista ja imperatiivista paradigmaa käyttäviin. Molemmille ratkaisuille esitettiin selityksiä: funktionaaliset kielet sopisivat luonnollisemmin kvanttiohjelmointiin ja niillä tehtyjen ohjelmien oikeellisuus olisi helpommin todistettavissa, kun taas imperatiiviset kielet olisivat helppokäyttöisempiä ja useimmille ohjelmoijille tutumpia.

Jos tulevaisuudessa kvanttitietokoneet saadaan kehitettyä sille asteelle, että kvanttiohjelmointikielet tulevat oikeasti tarpeellisiksi, on mielenkiintoista nähdä, mitkä näistä kielistä tulevat olemaan suosituimpia vai nouseeko pinnalle jokin tässä tutkielmassa mainitsematon kieli. Useimmat työssä mainituista kielistä perustuvat johonkin klassiseen ohjelmointikieleen ja ovat suunniteltuja toimimaan kokoonpanossa, jossa klassinen tietokone ohjaa jotain kvanttilaitetta. Koska on todennäköistä, että ensimmäiset kvanttitietokoneet ovat juuri tällaisia hybridejä, on niiden ilmaantuessa täten heti saatavilla helposti omaksuttavia ja käyttökelpoisia ohjelmointikieliä.

Kirjallisuutta

Abhari, A. J., Faruque, A., Dousti, M. J., Svec, L., Catu, O., Chakrabati, A., Chiang, C-F., Vanderwilt, S., Black, J., Chong, F., Martonosi, M., Suchara, M., Brown, K., Pedram, M., & Brun, T. 2012. *Scaffold: Quantum programming language*. PRINCETON UNIV NJ DEPT OF COMPUTER SCIENCE.

Altenkirch, T. & Grattage, J. 2005.*A functional quantum programming language*. Teoksessa Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on (s. 249–258). IEEE.

Altenkirch, T., Grattage, J., Vizzotto, J. K., & Sabry, A. 2005. *An algebra of pure quantum programming*. arXiv preprint quant-ph/0506012.

Bettelli, S., Calarco, T., & Serafini, L. 2003. *Toward an architecture for quantum programming*. The European Physical Journal D-Atomic, Molecular, Optical and Plasma Physics, 25(2), s. 181–200.

Gay, S. J. 2006. *Quantum programming languages: Survey and bibliography*. Mathematical Structures in Computer Science, 16(04), s. 581–600.

Grattage, J. J. 2006. *A functional quantum programming language (Doctoral dissertation, University of Nottingham)*.

Grattage, J., & Altenkirch, T. 2005.*A compiler for a functional quantum programming language*. Submitted for publication.

Green, A. S., Lumsdaine, P. L., Ross, N. J., Selinger, P., & Valiron, B. 2013, June. *Quipper: a scalable quantum programming language*. In ACM SIGPLAN Notices, 48(6), s. 333–342. ACM.

Green, A. S., Lumsdaine, P. L., Ross, N. J., Selinger, P., & Valiron, B. 2013. *An introduction to quantum programming in Quipper*. In Reversible Computation, s. 110–124. Springer Berlin Heidelberg.

Lampis, M., Ginis, K. G., Papakyriakou, M. A., & Papaspyrou, N. S. 2008. *Quantum data and control made easier*. Electronic Notes in Theoretical Computer Science, 210, s. 85-105.

Mlnarik, H. 2007. *Operational semantics and type soundness of quantum programming language LanQ*. arXiv preprint arXiv:0708.0890.

Sanders, J. W., & Zuliani, P. 2000, July. *Quantum programming*. In Mathematics of Program Construction, s. 80–99. Springer Berlin Heidelberg.

Selinger, P. 2004. *Towards a quantum programming language*. Mathematical Structures in Computer Science, 14(04), s. 527–586

Selinger, P. 2004 .*A brief survey of quantum programming languages*. In Functional and Logic Programming, s. 1–6. Springer Berlin Heidelberg.

Sofge, D. A. 2008, February. *A survey of quantum programming languages: History, methods, and tools*. In Quantum, Nano and Micro Technologies, 2008 Second International Conference on, s. 66–71. IEEE.

Unruh, D. 2006. *Quantum programming languages*. Informatik - Forschung und Entwicklung, 21(1), s. 55–63.

Vizzotto, J. K. 2013, October. *Quantum Computing: State-of-Art and Challenges*. In 2013 2nd Workshop-School on Theoretical Computer Science, (WEIT), s. 9–13. IEEE.

Xu, J. & Song, F. 2008. *Quantum programming languages*. Frontiers of Computer Science in China, 2(2) s. 161–166.

Ying, M. & Feng, Y. 2011. *A flowchart language for quantum programming*. Software Engineering, IEEE Transactions on, 37(4), s. 466–485.

Zuliani, P. 2004, July. *Non-deterministic quantum programming*. In Proceedings of the 2nd International Workshop on Quantum Programming Languages, 33, s. 179–195.

Ömer, B. 1998. *A procedural formalism for quantum computing*.

Ömer, B. 2005. *Classical concepts in quantum programming*. International Journal of Theoretical Physics, 44(7), s. 943–955.