Tampereen yliopisto

Victor Zhou Jiang

C++- JA JAVA-OHJELMOINTIKIELIEN SUORITUSNOPEUTEEN VAIKUTTAVAT TEKIJÄT

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Huhtikuu 2023

TIIVISTELMÄ

Victor Zhou Jiang: C++- ja Java-ohjelmointikielien suoritusnopeuteen vaikuttavat tekijät
Tampereen yliopisto
Tieto- ja sähkötekniikan tutkinto-ohjelma, Tietotekniikka
Kandidaatintyö
Huhtikuu 2023

Ohjelmistojen kehittäminen on monimutkainen prosessi, joka vaatii eri ohjelmointikielien käyttöä. Kielen valinnassa huomioidaan usein eri tekijöitä kuten helppokäyttöisyys, toimivuus ja suorituskyky. Yksi suorituskyvyn kriittinen näkökohta on suoritusnopeus eli aika, joka ohjelmalta kestää tietyn tehtävän suorittamiseen.

Tässä työssä on tarkoitus selvittää, mitkä tekijät vaikuttavat C++ ja Javan suoritusnopeuksiin, sekä niiden vaikutusta. Kyseisiä ohjelmointikieliä käytetään laajalti ja niiden välistä suoritusnopeutta vertaillaan usein alan julkaisuissa ja keskustelufoorumeilla. Lisäksi työssä selvitetään, miten näiden kahden kielen välisiä suoritusnopeuksia voi vertailla.

Tutkielmassa tarkastellaan seuraavia tekijöitä: Ahead-Of-Time (AOT) kääntäminen, Just-In-Time (JIT) kääntäminen, roskienkeruu ja rinnakkaisuus. Näiden lisäksi tarkastellaan erilaisia menetelmiä suorituskyvyn mittaamiseen ja vertailuun, kuten mikro-, makro- ja mesosuorituskykytestaaminen.

JIT:stä ja AOT:sta todettiin, että JIT voi hidastaa ohjelmaa, koska se varaa resursseja ajon aikaista optimointia varten, mutta samalla tämä mahdollisesti nopeuttaa ohjelmaa. Roskienkeruun osalta todettiin, että optimoidulla koodilla ohjelman suoritusnopeutta voidaan parantaa manuaalisen muistinhallinnan avulla. Sama koskee myös C++:n rinnakkaisuutta.

Lopputuloksena oli, että vaikka Javalla on edut muistinhallinnan ja ohjelmoitavuuden suhteen, C++ on erinomainen suorituskykykriittisissä sovelluksissa matalan tason ohjauksen ja tehokkaan muistinhallinnan ansiosta. Toisaalta C++:n ja Javan suoritusnopeuksien erot eivät ole kovin suuret.

Avainsanat: C++, Java, suorituskyky, suoritusnopeus

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

JIT	Just-In-Time
AOT	Ahead-Of-Time
JVM	Java Virtual Machine
Profilointidata	Suorituskykytietoa, joka on kerätty profilointityökaluilla, jotka mittaavat ja analysoivat ohjelman käyttäytymistä suorituksen aikana.
Lomituslajittelu	Jaa ja hallitse algoritmi, joka lajittelee joukon.
Säie	Ohjelman osa prosessissa, joka voi toimia itsenäisesti ja samanaikaisesti muiden säikeiden kanssa.
Mutex	Synkronointimekanismi, jolla hallitaan pääsyä jaettuihin resursseihin monisäikeisissä sovelluksissa.
Delaunayn-kolmiot	Tapa jakaa pistejoukko kolmioiksi siten, että mikään piste ei ole minkään kolmion ympäri piirretyn ympyrän sisällä.
Bowyer-Watsonin menetelmä	Menetelmä, jolla luodaan Delaunayn kolmioita.

1. JOHDANTO

Java ja C++ ovat tunnetuimpia ja käytetyimpiä ohjelmointikieliä maailmassa [1]. Molemmilla kielillä on omat vahvuutensa ja heikkoutensa, minkä vuoksi ne sopivat erilaisiin sovelluksiin. Yksi keskeisistä tekijöistä, jotka määräävät ohjelmointikielen sopivuuden tiettyyn sovellukseen, on kielellä kirjoitettujen ohjelmien suoritusnopeus.

Suoritusnopeus on tärkeä mittari, joka vaikuttaa suoraan ohjelmistoympäristöjen yleiseen toiminnallisuuteen. Korkea suoritusnopeus on tärkeä vaatimus joissain ohjelmointiympäristöissä, kuten robotiikassa. Ohjelmat, jotka suoritetaan nopeammin, ovat yleensä tehokkaampia ja pystyvät käsittelemään suurempia työkuormia ilman merkittävää viivettä. [2] Tästä syystä ohjelmien suoritusnopeuden optimointi on olennainen osa ohjelmistokehitystä, joissa vaaditaan korkeaa vastausaikaa.

Verrattaessa Javan ja C++:n suorituskykyä toisiinsa mitataan pääsääntöisesti algoritmien ja metodikutsujen suoritusnopeutta. Vertailumenetelmiä on lukuisia, mutta yleisin tapa on hyödyntää suorituskykytestejä. [2] Kyseisiä ohjelmointikieliä on käytetty pitkään, joten niitä on tutkittu paljon ja niistä on kirjoitettu lukuisia artikkeleita.

Koska Java on tulkittu kieli, niin oletetaan, että se on hitaampi kuin käännetty kieli ja että se käyttää enemmän muistia. [2] Javan suorituskyky on kuitenkin tehostunut Just-In-Time (JIT)-kääntäjän käyttöönoton jälkeen. Täten käännösmenetelmien eroilla voi olla merkittävä vaikutus Java- ja C++-ohjelmien suoritusnopeuteen.

Toinen suorituskykyyn vaikuttava tekijä on roskienkeruu. Java käyttää roskienkeruuta, jonka toiminta perustuu siihen, että se vapauttaa automaattisesti muistia tietyn ajan jälkeen, kun kyseiseen muistiosoitteeseen ei enää viitata. Vastaavasti C++:ssa roskienkeruuta ei ole, jolloin ohjelmoija joutuu itse huomioimaan muistin vapautumisen. Tätä ohjelmoijat usein kritisoivat. [3]

Tutkielma on toteutettu kirjallisuuskatsauksena, jossa vertaillaan C++- ja Java-ohjelmointikielien suoritusnopeuksia ja siihen vaikuttavia tekijöitä. Tarkoituksena on selvittää, mitkä tekijät vaikuttavat suoritusnopeuteen ja mitä ohjelmointikieltä kannattaa käyttää missäkin tilanteessa. Tutkielmassa ei käsitellä kaikkia suoritusnopeuteen vaikuttavia tekijöitä vaan käännösmenetelmiä, roskienkeruuta ja rinnakkaisuutta.

Luvussa 2 käydään läpi tutkimusmenetelmät ja lähteiden etsintä. Luvussa 3 käsitellään käännösmenetelmiä, roskienkeruuta ja rinnakkaisuutta, sekä niiden vaikutusta suoritusnopeuteen. Luvussa 4 käydään läpi eri tapoja tutkia ja verrata suoritusnopeuksia Java ja

C++ ohjelmointikielien välillä. Luvussa 5 käydään edellisten tutkimusten tuloksia. Lopuksi luvut 6 ja 7 ovat keskustelu ja yhteenveto.

2. TUTKIMUSMENETELMÄ

Tutkielma on toteutettu kirjallisuuskatsauksena. Lähteitä on etsitty Tampereen yliopiston Andor-palvelusta, Google Scholarista ja ACM Digital Librarysta. Haut on tehty suurimmaksi osaksi englanniksi aiheen luonteen vuoksi. Hakutermejä olivat muun muassa ”Java”, ”C++”, ”performance”, ”execution speed”, ”JIT”, ”garbage collection” ja ”concurrency”. Termejä yhdisteltiin käyttäen sulkuja, sekä AND- ja OR-operaatioista.

Lähteitä etsittiin aluksi Andorista tai Google Scholarista, koska niiden kautta löytyi aineiston oikea julkaisuosoite. Hakutulosten laajentamiseksi ja tarkentamiseksi hyödynnettiin myös ACM Digital Librarya. Tuloksia löytyi erittäin paljon, joka johti siihen, että karsintoja joutui tekemään jo varhaisessa vaiheessa.

Lähteen valintakriteerinä toimi sen otsikko, tiivistelmä ja julkaisusivusto. Jos nämä vaikuttivat lupaavilta ja sopivilta, lähde tallennettiin jatkotarkasteluja varten. Andorista tai ACM Digital Librarystä löydetyt lähteet oletettiin olevan käyttökelpoisia, kun taas Google Scholarista löydettyjen lähteiden luotettavuus tarkistettiin erikseen hakemalla ne Andorista tai lukemalla ne läpi.

Tämän jälkeen arviointiin lähteiden johdannot ja lopputulokset. Niiden avulla karsittiin lähteet, jotka eivät vaikuttaneet sopivilta tutkielmaan. Karsinnan jälkeen lähteet luettiin läpi, minkä pohjalta valittiin aihealueeseen sopivat aineistot.

3. SUORITUSNOPEUTEEN VAIKUTTAVAT TEKIJÄT

Useat eri tekijät vaikuttavat merkittävästi ohjelmointikielien suoritusnopeuteen. Vaikka Java ja C++ ovat suosittuja ja laajalti käytettyjä, niillä on merkittäviä eroja, jotka vaikuttavat näiden kahden suoritusnopeuteen. Seuraavissa alaluvuissa käydään läpi muutamia tärkeitä tekijöitä ja niiden vaikutuksia.

3.1 JIT ja AOT

Just-In-Time (JIT) ja Ahead-of-Time (AOT) ovat tapoja kääntää koodia. Ne vaikuttavat merkittävästi modernien ohjelmointikielien suorituskykyyn ja -nopeuteen. JIT määrittelee useimmin suoritetut koodialueet tulkin profilointidatan avulla. Tämän jälkeen se luo uuden version koodista, joka toimii samalla tavalla, mutta nopeammin, koska se on jo käännetty. JIT luo koodia ajon aikana, mikä antaa enemmän joustavuutta, mutta kuluttaa resursseja, jotka olisi muuten käytetty itse ohjelman suoritukseen. AOT-kääntäjä muuttaa koodin konekielelle ja optimointi tehdään ennen ohjelman suoritusta. Tällöin ohjelman käynnistysaika pienenee. [4]

Yleisesti oletetaan, että tulkittu kieli on hitaampi kuin käännetty kieli. Java on tulkittu kieli ja C++ on käännetty kieli. Vuonna 1998 Javaan lisättiin JIT-kääntäminen, ja nykyisin se on tärkeä osa Java-virtuaalikonetta (Java Virtual Machine, JVM). Koska JIT kääntää koodin konekielelle lennosta, se parantaa Java-ohjelman suorituskykyä ja -nopeutta. Se myös mahdollistaa Java-ohjelman alustariippumattomuuden, koska JVM on vastuussa koodin kääntämisestä. [2]

C++ käyttää AOT-kääntämistä. Tästä syystä sen suoritusnopeus on korkea, koska se pystyy varaamaan kaikki resurssit ohjelman ajamiseen. Ongelmana on, että ohjelma joudutaan kääntämään jokaiselle alustalle erikseen. [5]

Suorituskyvyn suhteen JIT:llä on se etu, että se pystyy optimoimaan koodin lennossa tietyn laitteiston ja järjestelmäympäristön perusteella. Näin Java voi hyödyntää kaikkia saatavilla olevia suorituskykyä tehostavia parannuksia ja optimointeja. [3] On kuitenkin mahdollista, että Java-ohjelma kärsii yleiskustannuksen lisääntymisestä, koska se kääntää koodia suorituksen aikana, mikä johtaa hitaampaan yleiseen suoritusaikaan. [4]

Toisaalta AOT voi johtaa nopeampiin suoritusaikoihin, koska koodi on jo käännetty alkuperäiseksi konekieleksi. Tämä tarkoittaa, että muutosten jälkeen koodi on käännettävä uudelleen, mikä saattaa olla työlästä ja monimutkaista. Lisäksi AOT ei välttämättä pysty

hyödyntämään käytettävissä olevia suorituskykyä tehostavia parannuksia ja optimointeja, koska koodi on jo käännetty eikä sitä voi muuttaa. [5]

C++:lle on kehitetty eri lisäyksiä, jotka mahdollistavat JIT-kääntämisen jollain tasolla. Näitä ovat muun muassa Cling ja ClangJIT. Cling toimii samalla tavalla kuin Java-virtuaalikoneen JIT. Vastaavasti ClangJIT mahdollistaa määrittyjien funktiomallien kääntämisen JIT:llä. Näin saadaan yhdistettyä AOT:n sekä JIT edut. Sen sijaan, että harvoin käytetyt funktiomallit käännettäisiin ennen ajoa, ne käännetään vasta ajon aikana. Näin edellisessä tilanteessa säästetään aikaa. Tällöin oletetaan, että funktiomallit ovat ennakolta määritelty JIT-käännettäviksi. [6]

Finkelin et al. selvittivät, että ClangJIT:tia hyödynnettäessä C++:n suorituskyky oli jopa kahdeksan kertaa parempi. Joissain tilanteessa se taas ei tuottanut ollenkaan parannuksia. [6] ClangJIT:ssä on seuraava rajoitus. Tyyppipäättelymekanismeja, esimerkiksi ”*auto*”, ei voida hyödyntää funktiomalleissa, jotka käyttävät JIT kääntämistä. [6] ClangJIT julkaistiin vuonna 2019 joten se vakaudessa ja luotettavuudessa on edelleen parannettavaa, verrattaessa sitä AOT-kääntämiseen tai muihin vastaaviin lisäyksiin. Lisäksi se kärsii hieman JIT-kääntämisen huonoista puolista, mutta ei merkittävästi, koska vain osa ohjelmaa on JIT käännetty.

3.2 Muistinhallinta roskienkeruun avulla

Roskienkeruuta käytetään useissa ohjelmointikielissä. Tämä auttaa hallitsemaan muistia. Tarkoituksena on tyhjentää muistia sen mukaan, kun sitä varannutta kohdetta ei enää käytetä ohjelmassa. Muistia vapautetaan automaattisesti ilman, että ohjelmoija joutuu itse erikseen sen ohjelmoimaan. [3]

Javassa roskienkeruu on sisäänrakennettu JVM:ään, ja se hallitsee automaattisesti muistin varaamisen ja purkamisen. Se käyttää useita eri algoritmeja ja tekniikoita määrittämään, mitä objekteja ei enää tarvita, ja vapauttaa niiden muistin. Roskienkeruu on suorituskyvyltään tehokas, koska se hyödyntää sukupolvitekniikkaa. Muistia varanneet kohteet jaetaan eri sukupolviin (nuoret, vanhat ja pysyvät). Muistia vapautetaan sukupolvien mukaan. [7]

C++ ei sisällä roskienkeruuta ja muistinhallinta on jätetty kokonaan ohjelmoijan vastuulle [3]. Tämä tarkoittaa, että kehittäjän pitää itse varata ja poistaa muistia käyttäen ”new”- ja ”delete”-operaatioita [8]. Vaikka tämä mahdollistaa hienorakeisen muistinhallinnan, se voi myös johtaa vakaviin ongelmiin, jos muistia ei hallita oikein. Esimerkiksi jos ohjelma ei vapauta muistia kunnolla, se voi johtaa muistivuotoon. Tämä on tilanne, jossa ohjelma kuluttaa yhä enemmän muistia, kunnes se kaatuu. C++:n versioon 11 on lisätty luokkia,

jotka auttavat ohjelmoijaa muistinhallinnassa. Näistä eräät ovat niin kutsuttuja älykkäitä osoittimia (smart pointer). Nämä eivät täydellisesti estä muistivuotoja tai muita muistinhallintaan liittyviä ongelmia. Lisäksi ne eivät ole täysin automaattisia, koska ohjelmoija joutuu määrittämään ne itse. [3]

Molemmissa lähtökohdissa esiintyy kompromisseja. Javan automaattinen roskienkeräys helpottaa sovelluskehitystä, mutta se voi myös heikentää suorituskykyä. Esimerkiksi JVM:n sukupolvitekniikassa, kun muistia vapautetaan sukupolvien mukaan, se pysäyttää kaikki säikeet muistin vapautuksen ajaksi. Tämä on niin sanottu ”Stop the World Event”, ja se voi aiheuttaa ohjelman väliaikaista pysähtymistä ja yleiskustannusten lisääntymistä. [3,7]

Nämä negatiiviset puolet puuttuvat C++:sta, joten osaava ohjelmoija kykenee kehittämään sovelluksia, joiden ei tarvitse pysäyttää säikeitä muistin vapauttamisen takia. Tämä mahdollistaa nopeamman sovelluksen kuin vastaava Java-ohjelma. Pitää kuitenkin huomioida se, että jos ohjelmassa esiintyy muistivuotoja tai muistinhallinta on tehty huonosti, niin se voi vaikuttaa negatiivisesti sovelluksen suorituskykyyn ja -nopeuteen. [3]

Henriquesin ja Bernardinon tekemässä tutkimuksessa todettiin, että C++:n muistinhallinta on tehokkaampi kuin Javan. Lisäksi selvisi, että C++ oli noin neljä sekuntia nopeampi muistin vapauttamisessa. [3]

3.3 Rinnakkaisuus

Rinnakkaisuuden avulla ohjelma voi suorittaa useita tapahtumia samanaikaisesti. Se mahdollistaa ohjelman tehokkuuden nousun ja näin eri tapahtuminen suoritusnopeuksien kasvun. [9]

C++:ssa ja Javassa rinnakkaisuuden toteutukset ovat erilaiset. C++:n rinnakkaisuus saavutetaan käyttämällä säikeitä, jotka ovat kevyitä prosesseja ja joita voidaan suorittaa itsenäisesti. Nämä säikeet luodaan itse ja säikeet voivat kommunikoida keskenään jaetun muistin avulla. Synkronointimekanismeja, kuten mutexeja, voidaan käyttää estämään kilpailuolosuhteet ja muut synkronointiongelmat. C++11 sisältää vakiokirjaston säikeitä varten, mikä tarjoaa kätevän tavan luoda ja hallita säikeitä. [9]

Java käyttää ylemmän tason rinnakkaisuusmallia, joka perustuu säieallasmalliin. Tässä mallissa ohjelman alussa luodaan kiinteä määrä säikeitä ja tehtävät lisätään jonoon. Kun säie tulee saataville, se hakee seuraavan tehtävän jonosta ja suorittaa sen. JVM tarjoaa sisäänrakennetun tuen tälle mallille java.util.concurrent-paketin kautta, joka sisältää

säievarastot, mutexit ja muut rinnakkaisuustyökalut. [10] C++:ssa säieallas pitää itse toteuttaa, jos sellaista haluaa käyttää [9].

Javassa on myös mahdollista käyttää samankaltaista rinnakkaisuustekniikkaa kuin C++:ssa säiealtaiden sijaan, jos kehittäjä niin haluaa. Tämä nostaa muistinhallinnan yleiskustannusta, muistin varaamisesta ja vapauttamisesta johtuen. Javassa säieallas voi johtaa tehokkaampaan resurssien käyttöön ja parempaan skaalautumiseen erityisesti sovelluksissa, jotka vaativat paljon samanaikaisia, mutta lyhytaikaisia tehtäviä. [10,11]

Vaikka sekä C++ että Java tarjoavat eri tapoja käsitellä rinnakkaisuutta, ne eroavat lähestymistavassa ja suorituskyvyssään. C++ tarjoaa matalamman tason lähestymistavan rinnakkaisuuteen, mikä voi olla tehokkaampaa tietyissä tilanteissa [9]. Se vaatii kuitenkin huolellisempaa hallintaa jaettujen resurssien kanssa ja voi olla virhealttiimpi. Javan korkeamman tason rinnakkaisuusmalli on helpompi käyttää, ja se on vähemmän altis virheille, mutta voi kärsiä säiealtaiden luomisesta ja hallinnasta johtuvista lisäkustannuksista [10].

4. JAVAN JA C++ VÄLINEN SUORITUSNOPEUS

Suoritusnopeuden tutkimiseen on useita eri tapoja. Seuraavissa alikappaleissa avataan eri tutkimustapoja ja miten niitä voidaan hyödyntää suoritusnopeuksien tutkimuksessa. Lisäksi käydään myös läpi edellisiä tutkimuksia ja miten suoritusnopeutta voidaan parantaa.

4.1 Suoritusnopeuden mittaustavat

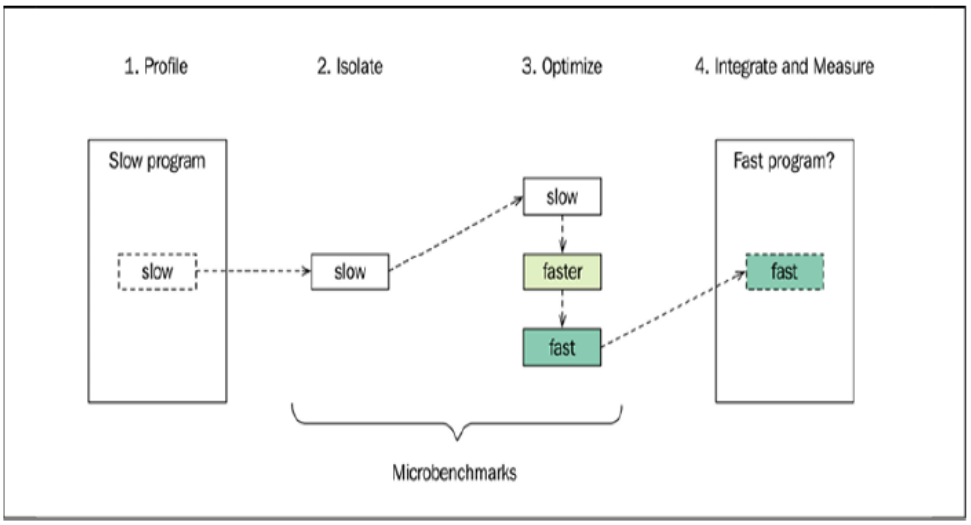
Suoritusnopeutta voi tutkia useilla eri tavoilla. Yksinkertaisin tapa mitata suoritusnopeutta on käyttää ajastinta. Ajastin mittaa ohjelman suoritukseen kulunutta aikaa. Menetelmän etuna on sen yksinkertaisuus. Se ei vaati vaadi ylimääräisiä ohjelmistoja tai työkaluja. Menetelmä ei kuitenkaan ole tarkka, koska se riippuu useista tekijöistä, kuten käyttöjärjestelmästä, laitteistosta ja muista samanaikaisesti käynnissä olevista taustaprosesseista. [12]

Toinen menetelmä suoritusnopeuden tutkimiseen on profilointi. Profilointi perustuu kussakin funktiossa tai koodilohkossa käytetyn ajan mittaamiseen. Menetelmä tarjoaa yksityiskohtaisen analyysin ohjelman suorituskyvystä, jolloin kehittäjät voivat tunnistaa tietyt alueet, jotka voivat aiheuttaa suorituskykyongelmia, niin sanottuja pullonkauloja. Nämä ovat sellaisia kohtia koodista, jotka hidastavat ohjelman toimintaa. Menetelmän haittana on, että sen suorittaminen saattaa vaatia lisäohjelmistoja ja -työkaluja. [12,13] Toisaalta se on osa JIT-kääntäjää, jonka avulla JVM-virtuaalikone optimoi ajonaikaista koodia. [4]

Suorituskykytestaaminen on kolmas menetelmä suoritusnopeuden mittaamiseen ja vertailuun. Se muistuttaa ajastimen käyttöä ja ajastin onkin osa suorituskykytestaamista. Suorituskykytestaaminen sisältää ohjelman tai koodin suorittamista useita kertoja eri olosuhteissa ja suoritukseen kuluvan keskimääräisen ajan mittaamisen. Erona pelkän ajastimen kanssa mittaamiseen on se, että suorituskykytestaamisessa käytetään keskimääräistä aikaa. Menetelmä on hyödyllinen vertailtaessa eri ohjelmien tai koodien suorituskykyä samoissa olosuhteissa. Tätä voi hyödyntää profiloinnin jälkeen tutkimalla mikä toteutustapa tuottaa parhaimman suoritusnopeuden pullonkaulojen korjaamiseksi. [12]

Suorituskykytestaaminen voidaan jakaa eri versioihin. Yksi versio siitä on mikrosuorituskykytestaaminen. Se perustuu tiettyjen pullonkaulojen löytämiseen koodista esimerkiksi profiloimalla, jonka jälkeen osat eristetään ja niille luodaan erillinen mikrosuorituskykytesti niille. Tämän jälkeen eristettyä osaa interpoloidaan eri toteutuksilla ja valitaan se toteutus, joka antaa parhaimman tuloksen. Mikrosuorituskykytestaamisessa on kuitenkin

sudenkuoppia. Tästä on esimerkkinä tilanne, jossa kääntäjä optimoi eristetyn koodin eri tavalla sen ollessa osa koko ohjelmaa tai JVM-virtuaalikoneessa. JIT ja roskienkeruu tekevät mikrosuorituskykytestien toteuttamisesta hankalaa, koska se voi johtaa epätarkkoihin tuloksiin. [13,14] Kuvassa 1 on esitetty mikrosuorituskykytestaamisen osuus ohjelman optimoimisessa.



Kuva 1. *Mikrosuorituskykytestauksen osuus [14].*

Toinen versio suorituskykytestaamisesta on makrosuorituskykytestaus. Siinä mitataan koko ohjelman suorituskykyä, tyypillisesti suorittamalla joukko ennalta määrättyjä tehtäviä tai skenaarioita. Menetelmä tarjoaa kattavamman arvion ohjelman suorituskyvystä, mutta se ei välttämättä tunnista tiettyjä koodiongelmia. [13]

Lopuksi on mesosuorituskykytestaus, joka sijoittuu mikro- ja makrosuorituskykytestauksen väliin. Se sisältää suuremman koodiosan, kuten moduulin tai toiminnon, suorituskyvyn mittaamisen. Koko ohjelman suorituskyvyn mittaaminen ei kuitenkaan sisälly siihen. Esimerkiksi palvelimen vastaus REST pyyntöön ja sen nopeuden mittaaminen olisi mesosuorituskykytestausta. Siinä mitataan vastauksen lukemiseen ja sen kirjoittamiseen käytettyä aikaa. Mesosuorituskykytestaus tarjoaa yksityiskohtaisemman analyysin kuin makrosuorituskykytestaus, mutta siinä on vähemmän sudenkuoppia kuin mikrotestauksessa. [13]

4.2 Suoritusnopeuden vertailu Javan ja C++:n välillä

Javalla ja C++ kirjoitetun ohjelman suoritusnopeutta voidaan vertailla keskenään esimerkiksi suorituskykytestauksella tai mieluummin makrosuorituskykytestauksella, koska mitataan koko ohjelman suoritusnopeutta. Profilointia voidaan hyödyntää ohjelmien opti-

moinnissa, koska sen tarkoitus on etsiä ohjelmasta raskaista koodialueita. Tämän jälkeen raskaat kohdat voidaan testata, koska ne kertovat paljon ohjelman suoritusnopeudesta. [15]

Ohjelma voi olla HTTP-palvelin tai joku algoritmi, joka esimerkiksi järjestää merkkijonot hyödyntäen lomituslajittelu. Kun molempien ohjelmien suorituskyky on testattu, voidaan niiden tuloksia vertailla ja näin määritellä kumpi kieli on suoritusnopeudeltaan tehokkaampi. [15]

Jos toteutukset ovat riittävän samanlaiset, niin voidaan hyödyntää myös mikrosuorituskykytestausta. Vertaillun pääpiirteet pysyvät suurin piirtein samoina kuin makrotestauksessa, mutta tällöin testataan vain pientä osaa koodista, kuten funktiota tai metodia esimerkiksi. [15]

Vertailua tehdessä pitää ottaa esimerkiksi kappaleessa 3 esitettyjä tekijöitä. JIT ja AOT muodostavat tärkeän osan ohjelmien suoritusta ja vertailijan pitää tietää, miten JIT voi aiheuttaa korkeamman suoritusajan. Tämä aiheutuu siitä, että se optimoi koodia lennosta ja näin voi lisätä ohjelman yleiskustannuksia [4]. AOT:ssa koodi käännetään ennen ajoa, joten siinä ei esiinny yleiskustannusten lisääntymistä. Toisaalta koodi saattaa olla suoritusnopeudeltaan nopeampi, koska se on optimoitu ennen ajoa. Ajon aikana optimoitu ohjelma voi olla parempi kuin ennen ajoa. [5]

Roskienkeruu vaikuttaa merkittävästi myös suoritusnopeuteen. Lisäksi muistinhallintaan vaikuttavat C++ ohjelmoitsijan taidot. Jos ohjelmoijan taso on korkea, hän kykenee välttämään muistivuotoja ja hallitsemaan muistia tehokkaasti. Tämä johtaa korkeaan suoritusnopeuteen, mikä voi olla parempi kuin roskienkeruuta sisältävä ohjelma [3]. Javan roskienkeruussa pitää myös huomioida sen toiminta, koska se hyödyntää ”Stop the World Event” toimintaa, joka voi aiheuttaa ohjelman väliaikaisen pysähtymisen [7]. Tämä taas nostaa sen suoritusaikaa.

Kääntämistekniikan ja roskienkeruun painoarvot suoritusnopeuteen ovat merkittävät, koska ne ovat suuri osa ohjelman ajoa. Toisaalta rinnakkaisuuden painoarvo riippuu kokonaan siitä, kuinka paljon sitä on hyödynnetty ohjelman toteutuksessa, jos lainkaan. Jos sitä on hyödynnetty, niin vertailijan pitää ottaa huomioon miten rinnakkaisuus on toteutettu ohjelmissa. Mikäli Java ohjelmassa on hyödynnetty säiealtaita, niin säikeiden luominen ja hallinta voi tuottaa lisäkustannuksia ja hidastaa ohjelmaa [10]. Toisaalta jos C++ ohjelmassa rinnakkaisuus on epätehokasta, se johtaa alempaan suoritusnopeuteen [9].

5. TULOKSET

Vertailua Javan ja C++ välillä on suoritettu vuodesta 1999 alkaen. Bernadin et al. tekemässä tutkimuksessa analysointiin Javaa symbolisen laskennan työkaluna. Tutkimuksessa vertailtiin, että pystyykö Javalla luomaan tehokkaita algebrallisia algoritmeja, jotka kykenevät kilpailemaan C++:lla tai muilla tietokonealgebrajärjestelmillä luotujen toteutusten kanssa. [16]

Tutkimuksessa toteutettiin polynomi kertolaskuja eri asteilla molemmilla kielillä ja niiden suoritusnopeus mitattiin. Tuloksena oli, että C++ on noin 2,5–2,8 kertaa nopeampi riippuen asteluvusta. [16] Nämä tulokset on esitetty taulukossa 1, jossa n on asteluku.

Taulukko 1. *Vertailutulokset [16].*

n	Java	C++	Java vs. C++
1000	2,6 s	0,9 s	2,8
2000	9,1 s	3,6 s	2,6
3000	20 s	8 s	2,5
4000	36 s	14 s	2,6
5000	57 s	22 s	2,6
6000	82 s	31 s	2,6
7000	111 s	42 s	2,6
8000	146 s	57 s	2,6

Java on kuitenkin tehostunut erittäin paljon Bernadin et al. tutkimuksen jälkeen. Tämä näkyy Grerardin et al. tutkimuksen tuloksessa vuodelta 2012. Tutkimuksessa analysoitiin Javan suorituskykyä robotiikassa ja tarkoituksena oli selvittää, että voiko se olla korvaava vaihtoehto C++:lle. Siinä vertailtiin molempien kielien kykyä luoda Delaunay-kolmioita hyödyntäen algoritmia, joka perustuu Bowyer-Watsonin menetelmään. He kehittivät testistä kolme eri versiota. Ensimmäisessä versiossa jokainen testikerta käännettiin yksitellen, toisessa ja kolmannessa versiossa testi käännettiin vain kerran. Toisessa versiossa ohjelmat käännettiin paikallisesti ja kolmannessa versiossa testattiin vain Java-ohjelmaa, joka käännettiin JVM-palvelin asetuksilla. Kolmannen testin tulokset vertailtiin toisen testin C++ tuloksiin. Lisäksi toisessa ja kolmannessa versiossa testit ajettiin Linux ja Windows ympäristöissä. Jokaisessa testissä kolmiointia toistettiin 50 kertaa. [2]

Heidän tutkimuksessa selvisi, että versio kolme tuotti parhaimmat tulokset ja Java oli vain 1,09–1,51 kertaa hitaampi kuin C++, kun ohjelmat ajettiin Windows laitteella. Linuxilla Java oli taas 1,21–1,91 kertaa hitaampi. Jos taas käytettiin paikallista kääntäjää,

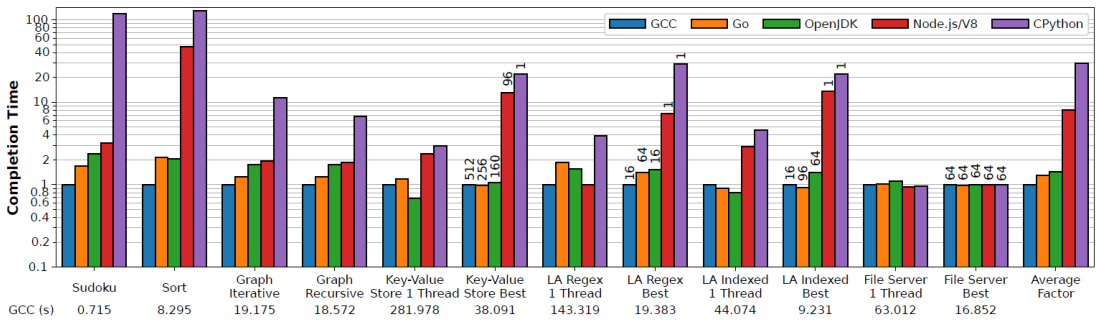
niin Java oli 2,72–5,61 kertaa hitaampi Windowsilla ja 2,81–4,04 kertaa hitaampi Linuxilla. Heidän lopputuloksensa oli, että Java on toimiva vaihtoehto robotiikassa sen nopeuden ja ominaisuuksien takia.

Lion et al. julkaisemassa tutkimuksessa vuodelta 2022 todetaan, että Javan suoritusnopeus ei ole laskenut merkittävästi viimeisen kymmenen vuoden aikana. Heidän tutkimuksessa vertailtiin JavaSriptin, Pythonin, Gon ja Javan suoritusnopeutta C++:saan. Tutkimuksen tarkoituksena oli ymmärtää erot suoritusnopeuden ja skaalautuvuuden suhteen. Lion et al. totesivat tutkimuksessa, että he eivät voi käyttää olemassa olevia suorituskykytestejä, koska ne olivat suunniteltu tietyille kielille ja niiden hyödyntäminen tutkimuksessa oli mahdotonta. Tämä johti siihen, että he kehittivät kuusi eri ohjelmaa, joiden tarkoitus oli kattaa pienistä ohjelmista suurempiin ohjelmiin. He pyrkivät siihen, että ohjelmat olivat kirjoitettu samalla tavalla jokaisella kielellä. Nämä kuusi ohjelmaa olivat

1. sudoku-ratkaisija, joka hyödynsi brute-force hakualgoritmia
2. merkkijonon järjestä, joka käytti lomituslajittelua
3. graafiväritys, jonka tarkoitus on luokitella jokainen kärkipiste graafista
4. avainarvovarasto, josta haetaan arvoja avaimen perusteella
5. lokianalysoija, joka erottaa toistuvat muuttujien arvot staattisesta tekstistä
6. tiedostopalvelin, joka hakee staattisia tiedostoja hakemistosta

Näiden ohjelmien avulla he loivat kaksitoista suorituskykytestiä, joista jokainen ajettiin viisi kertaa. Avainarvovarastossa, lokianalysoijassa ja tiedostopalvelimessa käytettiin myös rinnakkaisuutta. Ohjelmat ajettiin Linux laitteella. Suorituskykytestien tulokset näkyvät kuvassa 2. [15]

Heidän tuloksistansa selvisi, että C++ oli nopein melkein kaikissa suorituskykytesteissä. Java oli keskimäärin 1,45 kertaa hitaampi kuin C++:ssa ja se päihitti C++:n kahdessa eri testissä. [15] Tämä tutkimustulos asettuu Gherardin et al. tutkimustulosten väliin.



***Kuva 2**. Kahdentoista eri suorituskykytestin tulokset. Tulokset on esitetty logaritmisessa skaalassa y-akselin suhteen. LA tarkoittaa lokianalyysia [15].*

Näiden tulosten perusteella voidaan sanoa, että Javan suoritusnopeus kasvoi vuosina 1999–2012, mutta ollut sen jälkeen melko muuttumaton. Pitää kuitenkin ottaa huomioon, että jokaisessa tutkimuksessa tutkimustapa oli erilainen ja näin tulokset eivät reflektoi toisiaan täydellisesti.

6. KESKUSTELU

Tutkielman tavoitteena oli selvittää millaiset tekijät vaikuttavat Javan ja C++:n suoritusnopeuksiin ja nopeuksien vertailu. Samalla oli tarkoitus antaa suuntaviivoja kehittäjälle näiden kahden kielen valinnassa. Vaikuttavia tekijöitä löytyi useampia, mutta tässä tutkielmassa päätettiin keskittyä kolmeen: kääntäminen, muistinhallinta roskienkeruulla ja rinnakkaisuus. Samalla käytiin myös läpi erilaisia tapoja mitata kielien välistä suoritusnopeutta.

Tutkielman aikana havaittiin, että kääntämisellä ja roskienkeruulla on suuret vaikutukset suoritusnopeuteen. Toisaalta taas muistinhallinnan vaikutus ilman roskienkeruuta ja rinnakkaisuutta, riippuu kokonaan kehittäjän taidoista ja kyvyistä. Näiden tulosten kaltaisia väitöksiä kuulee useasti internetissä ja nämä tulokset vahvistavat näitä väitöksiä. Tutkielmassa ei kuitenkaan tutkittu kaikkia muuttujia ja näin ollen jatkon kannalta ne olisivat hyvä ottaa tutkittavaksi.

Johdannossa todettiin, että Java on suoritusnopeudeltaan hitaampi kuin C++:ssa, koska se on tulkittu kieli. Tämä hypoteesi osoittautui tutkielmassa oikeaksi. Ero on kuitenkin pieni ja joissain tapauksissa Java jopa peittoaa C++:n. Näin ollen suoritusnopeus ei pitäisi olla pääargumentti, kun pohditaan kielen valintaa, ellei sovellus vaadi korkeaa suoritusnopeutta. Kehittäjän pitäisi sen sijaan keskittyä muihin asioihin, esimerkiksi kielien tarjoamiin ominaisuuksiin ja niiden vahvuuksiin. Jos suoritusnopeus on tärkeä osa ohjelmaa kuten robotiikassa, niin C++:n valitseminen on järkevämpi vaihtoehto. Tällöin on oletettava, että kehittäjän taidot ovat riittävät.

7. YHTEENVETO

Tutkielman tavoitteena oli selvittää millaiset tekijät vaikuttavat Javan ja C++:n suoritusnopeuksiin ja näiden kahden kielen suoritusnopeuksien vertaileminen. Aiheesta on tehty useita tutkimuksia, joten lähteiden etsimen ei tuottanut ongelmia.

Java- ja C++-kielellä kirjoitettujen ohjelmien suoritusnopeuteen vaikuttavat useat tekijät, kuten roskienkeruu, samanaikaisuus ja Just-In-Time-kääntäminen. Roskienkeruu Javassa voi aiheuttaa suorituskyvyn lisäkustannuksia, mutta se tarjoaa myös kätevän automaattisen muistinhallinnan, mikä voi vähentää virheitä ja lisätä tuottavuutta. C++ puolestaan vaatii manuaalista muistinhallintaa, mikä voi olla aikaa vievää ja virhealtista, mutta tarjoaa paremman hallinnan muistin käyttöön.

Rinnakkaisuus vaikuttaa myös suoritusnopeuteen, sillä Javassa on sisäänrakennettu tuki säiealtaille, kun taas C++:ssa tätä ei ole. Just-In-Time-kääntäminen Javassa parantaa suorituskykyä, koska ohjelma voi olla paremmin optimoitu, mutta se voi myös aiheuttaa ylimääräisiä kustannuksia käännösprosessin aikana. Cling ja ClangJIT ovat uusia avauksia C++ maailmassa, mitkä tarjoavat Just-In-Time-kääntäjän C++ koodille.

Suoritusnopeuden mittaamisessa ja vertailussa voidaan käyttää erilaisia menetelmiä, kuten mikro-, makro- ja mesosuorituskykytestaukset. Lisäksi profilointi ja ajastin voivat avata ymmärrystä tiettyjen koodisegmenttien suorituskyvystä. On kuitenkin huomioitava näiden menetelmien rajoitukset ja mahdolliset väärintulkinnat.

Tutkimustulosten perusteella voidaan sanoa, että Java on kirinyt C++:san etumatkaa kiinni viimeisten kymmenien vuosien aikana ja se on hyvä vaihtoehto sovelluksiin, jotka vaativat korkeaa suoritusnopeutta. Kaiken kaikkiaan ymmärrys suoritusnopeuksiin vaikuttavista tekijöistä ja eri tutkimustulosten tuloksiin voi auttaa ohjelman kirjoituskielen valinnassa C++ ja Javan väliltä. Java voi olla hyvä vaihtoehto sen sisäänrakennetun roskienkeruun ja rinnakkaisuuden johdosta. Suoritusnopeudeltaan se kilpailee hyvin C++:n kanssa. Toisaalta, jos ohjelmoija on erittäin kokenut C++ osaaja niin hän kykenee luomaan erittäin nopeita sovelluksia johtuen siitä, että C++:ssa jättää kaiken ohjelmoijan vastuulle. Näin ohjelmoija voi hallita muistia täydellisesti.

LÄHTEET

[1] Abdulkareem SA, Abboud AJ. Evaluating Python, C++, JavaScript and Java Programming Languages Based on Software Complexity Calculator (Halstead Metrics). IOP conference series. Materials Science and Engineering. 2021.

[2] Gherardi L, Brugali D, Comotti D. A Java vs. C++ Performance Evaluation: A 3D Modeling Benchmark. In Simulation, Modeling, and Programming for Autonomous Robots. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 161-172.

[3] Henriques L, Bernardino J. Performance of Memory Deallocation in C++, C# and Java. In CAPSI 2018 Proceedings. 10. 2018.

[4] Krylov G, Dueck GW, Kent KB, Maier D, D'Souza I. Ahead-of-time compilation in OMR: overview and first steps. In CASCON '19: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering; 2019. p. 299-304.

[5] Thom M, Dueck GW, Kent KB, Maier D. A survey of ahead-of-time technologies in dynamic language environments. In CASCON '18: Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering; 2018. p. 275-281.

[6] Finkel H, Poliakoff D, Camier JS. ClangJIT: Enhancing C++ with Just-in-Time Compilation. In 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC); 2029. p. 82-95.

[7] Oracle. Java Garbage Collection Basics. [Internetti]. [viitattu 15. helmikuuta 2023]. Saatavilla: https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

[8] cppreference.com. New Expression. [Internetti]. [viitattu 10. helmikuuta 2023]. Saatavilla: https://en.cppreference.com/w/cpp/language/new

[9] Williams A. C++ concurrency in action. Shelter Island, New York : Manning; 2019.

[10] Oracle. Lesson: Concurrency. [Internetti]. [viitattu 15. helmikuuta 2023]. Saatavilla: https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

[11] Oracle. java.lang - Class Thread. [Internetti]. [viitattu 15. helmikuuta 2023]. Saatavilla: https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

[12] Akinshin A. Pro.NET Benchmarking: The Art of Performance Measurement. Berkeley, CA: Apress; 2019. p.7-9, 32-34.

[13] Oaks S. Java performance : in-depth advice for tuning and programming Java 8, 11, and beyond. Beijing : O'Reilly; 2020. p. 15, 64-70.

[14] Andrist B, Sehr V, Garney B. C++ High Performance - Second Edition. Birmingham: Packt Publishing; 2020.

[15] Lion D, Chiu A, Stumm M, Yuan D. Investigating Managed Language Runtime Performance: Why JavaScript and Python are 8x and 29x slower than C++, yet Java and Go can be Faster? In 2022 USENIX Annual Technical Conference (USENIX ATC 22); 2022. p. 835-852.

[16] Bernardin L, Char B, Kaltofen E. Symbolic computation in Java: an appraisement. In International Conference on Symbolic and Algebraic Computation: Proceedings of the 1999 international symposium on Symbolic and algebraic computation; 1999. p. 237-244.