Tampereen yliopisto

Toni Uimonen

TEHOKAS LASKENTA PYTHON-OHJELMOINTIYMPÄRISTÖSSÄ

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintutkielma
Joulukuu 2021

TIIVISTELMÄ

Etunimi Sukunimi: Toni Uimonen
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Joulukuu 2021

Python on nykyään yksi suosituimmista ohjelmointikielistä ja yksi suuri syy sen suosion viimeaikaiseen kasvuun liittyy yhä kasvavaan määrään avoimen lähdekoodin kirjastoja, jotka ovat parantaneet monin tavoin kehittäjien tuottavuutta. Monet tieteenalat ovat omaksuneet Pythonin kielekseen uusille kehitysaskelille, ja tämän vuoksi tiedeyhteisö pyrkii nyt tekemään Pythonista kielen suorituskykyisen koodin kirjoittamiseen. Kehittäjät ovatkin pystyneet parantamaan Pythonin suoritustehoa monin eri keinoin, jotka joissain tapauksissa skaalautuvat jopa aikamme tehokkaimmille supertietokoneille. Tämän tutkielman tavoitteena on esitellä nykyaikaisia keinoja suoritustehon nostamiseksi Python-ohjelmointiympäristössä.

Tutkielma on muodoltaan kirjallisuuskatsaus, joka sisältää kokeellisen osuuden. Tutkielmassa esitellään Pythonia ja sen tulkkeja ja kääntäjiä, joilla voidaan tapauskohtaisesti tehostaa Python-koodin suorittamista. Myös yksi Pythonin tärkeimmistä kirjastoista, NumPy, on tieteellisen Python-ekosysteemin perustana merkittävässä osassa tutkielmassa. Suuri osa suorituskykyisemmän Python-ympäristön kehitystyöstä perustuu NumPyn korvaamiseen niin sanotuilla ”drop-in” -korvaajilla, jotka tekevät siitä monissa tapauksissa entistä suorituskykyisemmän. Näissä menetelmissä käytetään Pythonin rinnakkaisen ja hajautetun suorituksen menetelmiä, joista tutkielma sisältää oman osansa.

Tutkielman kokeellinen osuus on suorituskykyvertailu, jossa suoritustehon havainnollistamiseen käytetään vektoreiden pistetuloa. Tuloksista voidaan havaita, että käytetyillä menetelmillä, eli NumPyn omalla pistetulo-operaatiolla ja Numban JIT-kääntäjän käytöllä pystytään merkittävään suorituksen tehostamiseen verrattuna Pythonin vakiolistan käyttöön silmukassa. Myös mahdollisimman ajankohtaiseen ja vertaisarvioituun lähdemateriaaliin perustuva kirjallisuuskatsaus tuotti kokoelman nykyaikaisia menetelmiä, joilla on tutkimuksien mukaan pystytty suoritustehon nostamiseen Python-ympäristössä. Erityisesti hajautettua ja rinnakkaista suorittamista tukevat, ja lähes mielivaltaiseen määrään GPU-kiihdytettyjä solmuja skaalautuvat menetelmät näyttävät kykenevän NumPy-operaatioiden suorituksen huomattavaan nopeutukseen.

Avainsanat: Python, NumPy, Numba, suorituskyky

Sisällysluettelo

1 Johdanto

Python on nykyään yksi suosituimmista ohjelmointikielistä (O'Grady, 2020; IEEE, 2021). Yksi suuri syy sen suosion viimeaikaiseen kasvuun liittyy yhä kasvavaan määrään avoimen lähdekoodin kirjastoja, jotka ovat parantaneet kehittäjien tuottavuutta (Python, 2021; STX Next, 2021). Monet tieteenalat ovat omaksuneet Pythonin ohjelmointikielekseen avustamaan uusissa kehitysaskelissa ja tämän vuosi tiedeyhteisö pyrkii nyt tekemään Pythonista kielen suorituskykyisen koodin kirjoittamiseen (Ben-Nun et al., 2021).

Erityisesti numeeriseen laskentaan kehitetyn *NumPy*-moduulin (2021a) ympärille on kasvanut tieteellisen laskennan Python-ekosysteemi, joka yhdessä kasvaneiden datamäärien kanssa on nostanut vaatimuksia tehokkaammalle tietojenkäsittelylle Python-ohjelmointiympäristössä. Tähän ovat kehittäjät pyrkineet vastaamaan, ja monin eri keinoin onkin pystytty parantamaan suoritustehoa, joka joissain tapauksissa skaalautuu jopa aikamme tehokkaimmille supertietokoneille. Tässä tutkielmassa analysoidaan näitä keinoja kirjallisuuskatsauksen keinoin, ja pyrkimyksenä on löytää tämän hetken merkittävimmät nykyaikaiset työkalut ja niiden yhdistelmät suoritustehon nostamiseksi.

Tutkielman tutkimuskysymyksenä on, että miten tietojenkäsittelyä pystytään tehostamaan Python-ohjelmointiympäristössä. Tutkielman 2. luvussa kerrotaan tutkimusmenetelmästä. 3. luvussa esitellään Pythonia yleisesti ja tutustutaan sen eri toteutuksiin sekä esitellään NumPy-kirjasto ja rinnakkaisen ja hajautetun suorituksen ratkaisuja. 4. luku pitää sisällään tutkielman kokeellisen osuuden, jossa havainnollistetaan joidenkin menetelmien suorituskykyä tehostavaa vaikutusta vektorilaskennassa. 5. luvussa esitetään tulokset ja 6. luvussa yhteenveto.

2 Tutkimusmenetelmä

Tämä tutkielma on kirjallisuuskatsaus, joka sisältää kokeellisen osuuden. Kirjallisuuskatsaus toteutettiin systemaattisesti vaiheittain. Sen suunnitteluvaiheessa suunniteltiin katsauksen teko ja määriteltiin tutkimuskysymys. Toteutusvaiheessa tutkielman lähdemateriaali koottiin pääosin hakemalla aihepiirin tieteellisiä julkaisuja tietojenkäsittelytieteen keskeisistä tietokannoista, kuten *ACM Digital Library* (2021), *Computer Science Database* (ProQuest) (2021*), IEEE Electronic Library* (IEL) (2021), *ScienceDirect* (Elsevier) (2021) ja *SpringerLink* (2021) -hakutietokannoista. Artikkelien karsintaa suoritettiin erityisesti otsikon, abstraktin ja tuloksien perusteella. Myös julkaisupäivämäärä oli yksi merkittävä tekijä, koska tutkimuksessa pyrittiin mahdollisimman ajankohtaisen aineiston

käyttöön. Myös työkalujen dokumentaatiota on käytetty hyväksi. Tutkielman tulosvaiheessa esitettiin kirjallisuuskatsauksen ja kokeellisen osuuden tulokset ja yhteenveto.

3 Python

Python on helposti opittava ja tehokas ohjelmointikieli. Siinä on korkean tason tietorakenteet ja yksinkertainen, mutta tehokas lähestymistapa olio-ohjelmointiin. Pythonin syntaksi ja dynaaminen tyypitys yhdessä tulkattavan luonteen kanssa sopivat erityisesti skripteihin ja nopeaan sovellusten kehittämiseen monilla aloilla ja useimmilla alustoilla (Python, 2021). Pythonin ajonaikainen dynaamisuus tulkattuna kielenä tekee siitä hitaamman kuin käännetyt kielet, kuten C tai C++. Ajan mittaan Pythonin nopeusrajoitteisiin on keksitty erilaisia ratkaisuja, kuten suorituskykyä vaativien tehtävien kirjoittaminen C-kielellä Pythoniin käärittynä. (Yegulalp, 2019)

Käännetyn koodin käyttämisen tehokkuusetu on niin merkittävä Pythoniin verrattuna, että olisi suositeltavaa kirjoittaa kaikki merkittävän laskennallisen taakan omaavat työt joko käännetyssä laajennusmoduulissa tai käännetyllä kielellä. Monet tieteellisen laskennan kirjastot tekevät juuri näin NumPy:stä lähtien. On myös mahdollista käyttää Cythonia, joka mahdollistaa ohjelman kääntämisen C-kieliseksi. (Smith, 2016)

3.1 Python-tulkit ja JIT-kääntäjät

Kun puhutaan Pythonista, tarkoitetaan usein paitsi kieltä, myös *CPython*-toteutusta (Python, 2021). Python on itse asiassa spesifikaatio kielelle, joka voidaan toteuttaa monella eri tavalla. Eri toteutukset voivat vaikuttaa suorituksen nopeuteen tai kirjastojen yhteensopivuuteen. Puhtaiden Python-kirjastojen pitäisi toimia Python-toteutuksesta riippumatta, mutta C-kielellä toteutetut eivät välttämättä (Reitz & Sclusser, 2016). Seuraavana tutustumme yleisimpiin toteutuksiin.

3.1.1 CPython

CPython on Pythonin C-kielellä kirjoitettu standarditoteutus. Se kääntää Python-koodin välitavukoodiksi, jonka virtuaalikone sitten tulkitsee ohjelman ajon yhteydessä. CPython tarjoaa korkeimman tason yhteensopivuuden Python-pakettien ja C-laajennusmoduulien kanssa. (Reitz & Sclusser, 2016)

3.1.2 Numba

Numba (Numba. 2021) on Just-in-Time -kääntäjä (JIT) CPythonille, joka on kirjoitettu C-kielellä. Numba on saatavilla kirjastona, joka voidaan ladata CPython-tulkissa. Ndarray tarjoaa ulotteisuuden sekä tyyppi- ja asettelutiedot, joiden avulla Numba voi luoda konekielikoodisia erikoissilmukoita taulukoille. (Lam et al., 2015)

3.1.3 PyPy

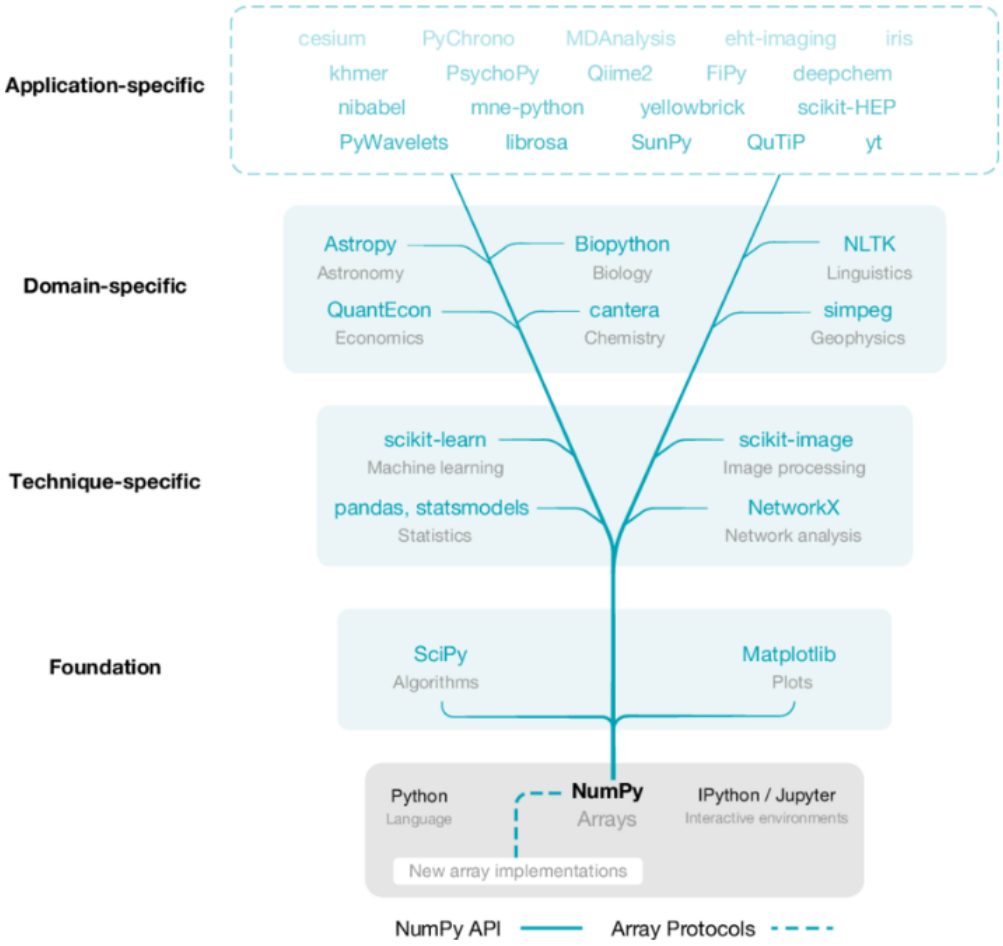
PyPy (2021) on Python-tulkki, joka on toteutettu Python-kielen rajoitetulla staattisesti tyypitetyllä *RPython*-osajoukolla (Reitz & Sclusser, 2016; RPython, 2021). PyPy käyttää JIT-kääntäjää ja tehtävästä riippuen sen vaikutus suoritustehon kasvuun voi olla hyvinkin suuri, koska se suorittaa ajonaikaisia optimointeja, joita CPython-tulkki ei tee. PyPy toimii parhaiten puhtailla Python-ohjelmilla ja se ei ole yhteensopiva kaikkien C-laajennuksien kanssa. Se myös vaatii pidempiaikaista ajoa, jotta se saa kerättyä ajonaikaista informaatiota optimointia varten. (Yegulalp, 2019)

3.1.4 Muut

Muita merkittäviä Python-tulkkitoteutuksia ovat esimerkiksi *Jython* (2021*)*, joka kääntää Python-koodin Java-tavukoodiksi, jonka Java-virtuaalikone sitten suorittaa. *IronPython* (2021) on Python-toteutus *.NET*-kehykselle (Microsoft .NET, 2021). *Python.Net* (2021) tarjoaa puolestaan lähes saumattoman integroinnin Python-asennukselle .NET *Common Language Runtime* (CLR) -sovellukseen (Reitz & Sclusser, 2016; Microsoft, 2021). JavaScript-toteutuksia ovat muun muassa *pyjs* (2021) ja *Skulpt* (2021). Lisäksi tulkeista voidaan vielä mainita *Stackless-Python* (2021) ja *Nuitka* (2021).

3.2 NumPy

NumPy on yksi Python-ekosysteemin tärkeimpiä kirjastoja ja se tarjoaa tehokkaan numeerisen laskennan perustaksi moniulotteisen taulukko-objektin, eli ndarray-objektin. Se tarjoaa myös erilaisia johdettuja objekteja, kuten matriisit ja peitetyt matriisit, ja valikoiman nopeita rutiineja matriisien käsittelyyn, kuten matemaattisiin ja loogisiin operaatioihin, muodon käsittelyyn, lajitteluun, valintaan, I/0:hon, diskreetteihin Fourier-muunnoksiin, lineaariseen algebraan, tilastollisiin perusoperaatioihin, satunnaissimulaatioihin ja paljon muuhun. (NumPy, 2021a)



Kuva 1. NumPy on tieteellisen Python-ekosysteemin perusta (Harris et al., 2020)

Tieteellinen laskenta sisältää usein suuria tietomääriä ja monimutkaisia laskutoimituksia, joten tällöin myös NumPyn käyttöön saatetaan kaivata tehostusta, jotta operaatioita voitaisiin suorittaa nopeammin. Yksi yleinen tapa NumPy-ohjelmien suorituskyvyn parantamiseksi ovat niin sanotut ”drop-in” -korvaajat, joista lisää myöhemmin. (Bauer et al., 2021)

3.3 Rinnakkaisuus ja hajautus

3.3.1 GIL ja moniprosessitila

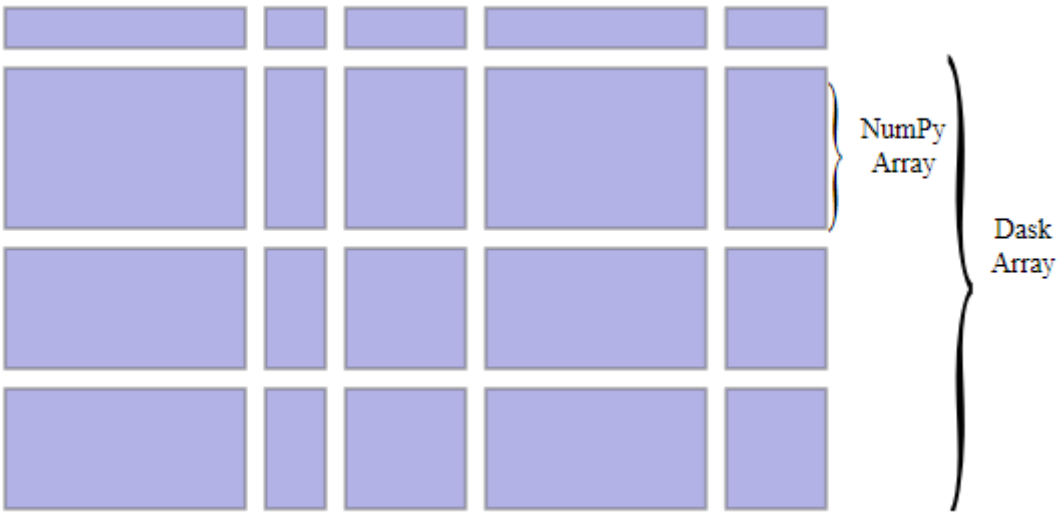
Nykyaikaisten ohjelmointikielien on tarjottava välineet rinnakkaiseen suorittamiseen, mutta säikeiden samanaikainen suorittaminen ei kuitenkaan ole Pythonin nykyisten virtuaalikoneiden vakio-ominaisuus (Gross & Remigius, 2019). Vaikka voimme luoda useita säikeitä Pythonin *Threading*-kirjaston (Python, 2021) avulla, suoritus on edelleen rajoitettu globaalin tulkkilukon (GIL) kautta. Tulkkilukko siis sallii suoritustilaan vain yhden säikeen kerrallaan (Ajitsaria, 2021). Tulkkilukon kiertämiseksi Pythonin kehittäjät

kääntyivätkin moniprosessiratkaisujen puoleen rinnakkaisen monisäikeistyksen sijaan. (Straßel et al., 2020)

Moniprosessitilan avulla useilla Python-prosesseilla on omat itsenäiset toisiinsa vaikuttamattomat GIL-tunnuksensa. Tämän tilan suurin etu on sen korkea vakaus, koska kaatuva aliprosessi ei vaikuta pääprosessiin ja muihin aliprosesseihin. Tilan haittana on prosessin korkeat luomiskustannukset ja ns. zombiprosessien mahdollisuus (Feng et al., 2021). Numba sallii GIL:n ohituksen tapauksissa, joissa Numba optimoi Python-koodin natiivikoodiksi, joka toimii vain natiivityypeillä ja -muuttujilla. GIL vapautetaan asettamalla ”nogil=True”, mutta tällöin on syytä huomioida monisäikeisyyden tavanomaiset sudenkuopat, kuten esimerkiksi synkronointi ja yhtenäisyys (Numba, 2021).

3.3.2 Dask

Dask (2021) on suosittu Pythonilla kirjoitettu tehtäväpohjainen ja ajonaikainen järjestelmä, joka tukee sekä ohjelmien hajautettua että rinnakkaista suorittamista. Dask tarjoaa monia erilaisia kirjastoja, joista erityisesti sen taulukkokirjasto tarjoaa käyttöliittymän, joka on lähes identtinen NumPyn kanssa (Bauer et al., 2021). Kuten kuvassa 2 nähdään, Dask-taulukot koordinoivat monia NumPy-taulukoita, jotka on järjestetty paloiksi ruudukon sisällä. Ne tukevat suurta NumPyn ohjelmointirajapinnan osajoukkoa. (Dask, 2021)



Kuva 2. Dask-taulukko koostuu useista NumPy-taulukoista (Dask, 2021)

Daskin ainoa laajennus NumPy-käyttöliittymään on tarve valita palatyyppi (chunk tuple), kun luodaan taulukoita. Palatyyppi kuvaa kuinka taulukko tulee osioida taulukon jokaisen ulottuvuuden mukaan. Palat voidaan valita eksplisiittisesti, tai antaa Daskin valita sisäisen heuristiikkansa avulla (Bauer et al., 2021). Dask skaalautuu joustavasti klustereihin,

joissa voi olla jopa tuhansia ytimiä, mutta se skaalautuu myös alas yhdelle tietokoneelle yhdellä prosessilla. (Dask, 2021)

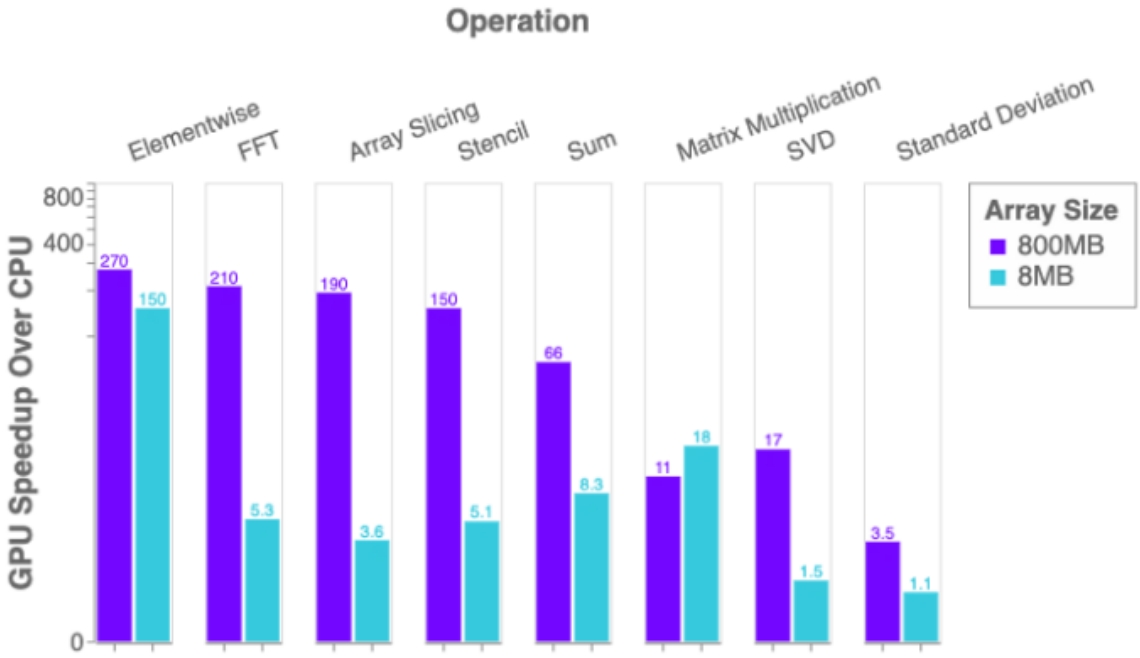
3.3.3 Legate

NumPyn korvaava *Legate*-kirjasto (2021) vaatii vain yhden koodirivin vaihdon, ja se voi skaalautua jopa lähes mielivaltaiseen määrään grafiikkasuoritin-, eli GPU-kiihdytettyjä solmuja. Verrattuna Dask-kirjastolla kirjoitettuihin vastaaviin ohjelmiin, Legate saavuttaa 10-kertaisen nopeutuksen 1280 CPU:lla ja 100-kertaisen 256 GPU:lla (Bauer & Garland, 2019). Kaksi NumPyn merkittävää käyttöliittymää rinnakkaista suoritusta varten yhden solmun koneissa ovat *Intel Python* (2021), joka rinnakkaistaa NumPy -toiminnot monien CPU-ytimien yli ja *CuPy* (2021), joka nopeuttaa NumPy -toimintoja yhdellä grafiikkasuorittimella. Legate NumPy -moduuli tarjoaa hajautettuja ja nopeutettuja toteutuksia monille yleisimmin käytetyille NumPy -operaatioille, jotka toimivat useiden solmujen ja grafiikkasuorittimen välillä. Legaten nykyinen toteutus sisältää kirjastot, jotka toisintavat NumPyn sekä datan manipulointiin ja analysointiin kehitetyn *Pandasin* (2021) käyttöliittymät suorittaessaan laskentaa niin monella CPU- ja GPU-suorittimella kuin käyttäjä asettaa saataville. (Bauer et al., 2021)

3.3.4 CuPy

CuPy on avoimen lähdekoodin kirjasto GPU:lla kiihdytettyyn laskentaan Pythonilla (CuPy, 2021). CuPy on ”drop-in” -korvaaja NumPy:lle, joka jakaa toiminnot yhdelle GPU:lle. CuPy myös tukee usean GPU:n suorittamista, mutta ei ”drop-in” -NumPy-liittymällä. Kuvassa 3 esitetään taulukko, jossa esitetään Entschevin (2019) toteuttaman suorituskykyvertailun tulokset, kun verrattiin yhden CPU:n NumPyn ja yhden GPU:n CuPy:n yksilöllistä suorituskykyä eri operaatioilla ja taulukkokooilla. Tuloksena saatiin

jopa 270-kertainen nopeutus elementtikohtaisille operaatioille, mutta on kuitenkin huomattava, että toimenpiteen luonne vaikuttaa nopeuteen.



Kuva 3. CuPyn nopeutuskertoimet NumPyn suoritusnopeuksiin verrattuna (Entschev, 2019)

3.3.5 Muut

NumPy:lle löytyy yhä kasvava määrä muitakin ”drop-in” -korvaajia ja niiden ominaisuudet ja käyttötavat vaihtelevat. Osa toteutuksista tukee hajautettuja CPU- tai GPU-klustereita ja osa vain yksittäisiä CPU:ita ja GPU:ita. Myös toteutuksien tukemat operaatiot saattavat vaihdella ajan myötä. Esimerkiksi alkuperäinen *Bohrium*-toteutus (2021) tuki hajautettuja CPU-klustereita, mutta tämä vaihtoehto ei näytä enää olevan tuettu (Bauer & Garland, 2019). Muita tunnettuja ”drop-in” -toteutuksia NumPy:lle ovat muun muassa *Grumpy* (2021), *JAX* (2021), *Weld* (2021), *Spartan* (2021) ja *NumPywren* (2021). Lisäksi ainakin *Arkouda* (2021), *Bodo* (2021), *Phylanx* (2021) ja *Ray* (2021) tarjoavat mahdollisia ratkaisuja suuren mittakaavan klustereille Python-ympäristössä.

4 Suorituskykyvertailu

Suorituskykyvertailu suoritettiin 64-bittisessä Windows 10 käyttöjärjestelmässä käyttäen *Anaconda*-ympäristöä (2021) ja *JypyterLab*-editoria (2021). Koneessa oli 3.40GHz In-

tel(R) Core(TM) i7-3770 suoritin ja 16 GB keskusmuistia. Vertailussa käytettiin suoritustehon havainnollistamiseen vektorilaskentaa, jolla voidaan analysoida matemaattisesti monia fysiikan ilmiöitä ja ominaisuuksia (Corral, 2021). Esimerkkinä tässä työssä käytettiin vektoreiden skalaari- eli pistetuloa.

Käytetyt ulkopuoliset kirjastot tuodaan 1. ohjelman mukaisesti, eli työ pitää sisällään NumPyn ja Numban laskentaan ja suorituksen tehostamiseen sekä *timeit*-paketin (Python, 2021) ajanottoon. Timeit-paketin käyttöön päädyttiin, koska testin suorituksen aikana roskien kerääminen on poistettu käytöstä ja timeit-funktio ottaa sisäisesti tarkan ajan käytössä olevan käyttöjärjestelmän mukaan (Python, 2021).

```
import numpy as np
from numba import jit
from timeit import timeit
```

Ohjelma 1. Ulkopuolisten kirjastojen tuonti

Ohjelmassa 2 luodaan tuhannen reaaliluvun NumPy-vektorit a_array ja b_array käyttämällä NumPyn random-moduulin randn-funktiota. Funktio luo tässä tapauksessa yksiulotteisen taulukon, joka on täynnä satunnaisia liukulukutyyppisiä arvoja (float), jotka on otettu yksimuuttujaisesta normaalijakaumasta, jonka keskiarvo on 0 ja varianssi 1 (NumPy, 2021b). Samat vektorit muunnetaan Pythonin normaaliin listamuotoon list-funktiolla.

```
1  # Luo vektorit NumPy-pohjaista ajonottoa varten.
2
3  a_array = np.random.randn(10**3)
4  b_array = np.random.randn(10**3)
5
6  # Luo listat NumPy-pohjaista ajonottoa varten.
7
8  a_list = list(a_array)
9  b_list = list(b_array)
```

Ohjelma 2. Vektorien ja listojen luonti

Vektoreiden luonnin jälkeen tarkastetaan, että niiden alkiot ovat halutussa muodossa. Vektoreiden ensimmäisten 10 luvun tulosteet ovat nähtävissä kuvassa 4.

```
# Tulostetaan kaikista ensimmäiset 10, jotta nähdään, että sisältävät lukuja.

print("NumPy a_array (head):\n", a_array[:10])
print("NumPy b_array (head):\n", b_array[:10])
print("\na_list (head):\n", a_list[:10])
print("\nb_list (head):\n", b_list[:10])
```

Ohjelma 3. Taulukoiden ja listojen alkujen tulostus

```
NumPy a_array (head):
 [-0.65021302  0.14391285  0.14741304  1.63629234 -0.49571756 -0.115643
 -0.32677316 -0.07672179 -0.52636041  0.5627449 ]
NumPy b_array (head):
 [ 1.00565732  1.03836827  2.12371791 -1.46477323 -1.19163686  0.04628054
  1.91699408 -1.59641069  0.78499045  1.00267239]

a_list (head):
 [-0.6502130209420357, 0.14391285431573614, 0.1474130428505866, 1.6362923449530045, -
0.49571755643372717, -0.11564299920234668, -0.32677316291733904, -0.07672179487515916,
-0.5263604093699785, 0.5627449016544727]

b_list (head):
 [1.0056573185672428, 1.0383682664991438, 2.123717908389228, -1.4647732271716836, -1.1
91636857377996, 0.04628053977197214, 1.9169940800888798, -1.5964106900626478, 0.784990
449842325, 1.002672385641151]
```

Kuva 4. Ohjelman 3 tulosteet

Tässä vertailussa suorituskykyä testataan siis kahden vektorin skalaari- eli pistetulolla, jossa vektorit ovat yksiulotteisia tuhannen liukulukuarvon sisältäviä taulukoita tai listoja. NumPyn dot-operaatiota voidaan myös käyttää matriisituloihin sekä n-ulotteisten taulukoiden tuloihin.

```
def dot(a, b):
    """
    Takes two vectors as a parameter and returns their dot product.
    Param: array-like floats
    Return: float
    """

    dot_product = 0
    for a, b in zip(a, b):
        dot_product += a * b

    return dot_product
```

Ohjelma 4. dot-funktio

Ohjelmassa 5 on esitetty toiminnallisuudeltaan ohjelmaa 4 vastaava ohjelma, mutta Numba-käännettävänä. Numban käyttöönotto vaatii vain funktion esittelyn yhteyteen komennon @jit(nopython=True), jolloin ajoon käytetään Numban kääntäjää.

```python
@jit(nopython=True)
def dot_numba(a, b):
    """
    Takes two vectors as a parameter and returns their dot product.
    Param: array-like floats
    Return: float
    """

    dot_product = 0
    for a, b in zip(a, b):
        dot_product += a * b

    return dot_product
```

Ohjelma 5. Numba-käännettävä dot-funktio

```python
@jit(nopython=True)
def dot_np_numba(a, b):
    """
    Takes two vectors as a parameter and returns their dot product.
    Param: array-like floats
    Return: float
    """
    return np.dot(a, b)
```

Ohjelma 6. Numba-kiihdytetty NumPyn dot-funktio

Vertailussa on huomioitava, että käytettäessä Numba-kääntäjää ensimmäisellä ajolla suoritukseen kuluu enemmän aikaa, koska koodi käännetään konekieliseksi. Merkittävämpi hyöty suoritusajoissa on nähtävissä vasta toisella suorituksella. Tässä vertailussa funktiot suoritetaan silmukassa 10 000 kertaa, joka on määritetty asettamalla ”number” -attribuutin arvoksi vastaava luku. Timeit-funktion ottama aika käsittää siis kaikki silmukan kierrokset. Ajan tulostuksen yhteydessä sekunnit kerrotaan luvulla 1000, jotta saadaan aikayksikkö tuloksien esittämisen kannalta mielekkäämpään muotoon, eli millisekunneiksi (ms).

```
# dot-funktion suoritus Python-listoilla
list_time = timeit(lambda: dot(a_list, b_list), number=10000)

print((list_time * 1000), "ms")
```

```
2307.059300000002 ms
```

```
# Numba-käännettävän funktion ensimmäinen suoritus
numba_time = timeit(lambda: dot_numba(a_array, b_array), number=10000)

print((numba_time * 1000), "ms")
```

```
284.96110000000033 ms
```

```
# Numba-käännettävän funktion toinen suoritus
numba_time = timeit(lambda: dot_numba(a_array, b_array), number=10000)

print((numba_time * 1000), "ms")
```

```
17.605600000024424 ms
```

```
# NumPyn oman dot-operaation suoritus
numpy_time = timeit(lambda: np.dot(a_array, b_array), number=10000)

print((numpy_time * 1000), "ms")
```

```
28.7343999999905 ms
```

```
# Numba-käännetyn NumPy-dot-operaation ensimmäinen suoritus
numba_plus_numpy_time = timeit(lambda: dot_np_numba(a_array, b_array), number=10000)

print((numba_plus_numpy_time * 1000), "ms")
```

```
167.9825000000008 ms
```

```
# Numba-käännetyn NumPy-dot-operaation toinen suoritus
numba_plus_numpy_time = timeit(lambda: dot_np_numba(a_array, b_array), number=10000)

print((numba_plus_numpy_time * 1000), "ms")
```

```
9.165899999999283 ms
```

Ohjelma 7. Suoritusaikojen mittaukset

```
# Lasketaan nopeutumiskertoimet verrattuna Pythonin list-toteutukseen

numba_x = list_time / numba_time
numpy_x = list_time / numpy_time
numba_plus_numpy_x = list_time / numba_plus_numpy_time

print("Numba:", numba_x)
print("NumPy:", numpy_x)
print("Numba + NumPy:", numba_plus_numpy_x)
```

```
Numba: 131.04121983896044
NumPy: 80.28910643691064
Numba + NumPy: 251.70024765709672
```

Ohjelma 8. Nopeutumiskertoimien laskeminen

5 Tulokset

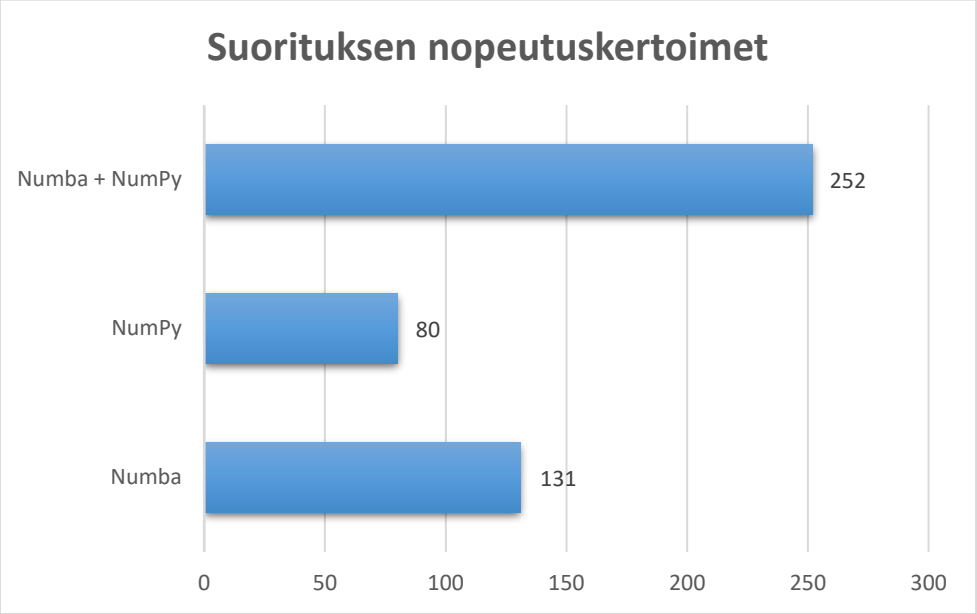
Taulukossa 1 on esitetty suorituskykyvertailun tulokset millisekunteina pyöritettynä kahteen desimaaliin. Tuloksista käy ilmi, että kaikki käytetyt menetelmät Pythonin lista-muotoisen skalaaritulon suorituksen tehostamiseksi toimivat. Tuloksista voidaan nähdä, että Numba-käännetyn suorituksen edut tulevat näkyviin vasta ensimmäisen suorituksen jälkeen, koska ensimmäisen suorituksen aikana Numba kääntää koodin konekieliseksi. Pythonin listatoteutus Numba-käännettynä antoi kuitenkin jo ensimmäisellä suorituksella merkittävästi nopeamman suoritusajan. Numba-käännetyn NumPyn dot-operaation suoritusaika oli sen sijaan ensimmäisellä suorituksella merkittävästi hitaampi, mutta toisella ajolla merkittävästi nopeampi.

Taulukko 1. Suorituskykytestin tulokset

Toteutus	Aika (ms)
Pythonin listatoteutus	2307,06
Pythonin listatoteutus Numba-käännettynä, ensimmäinen ajo	284,96
Pythonin listatoteutus Numba-käännettynä, toinen ajo	17,61
NumPyn oma dot-operaatio	28,73
Numba-käännetty NumPyn dot-operaatio, ensimmäinen ajo	167,98
Numba-käännetty NumPyn dot-operaatio, toinen ajo	9,17

Kuvassa 5 on esitetty käytettyjen menetelmien nopeutuskertoimet verrattuna Pythonin listamuotoisen vektorilaskennan suoritusnopeuteen. Numba-käännettyjen suorituksien osalta on jätetty selkeyden vuoksi kääntämisen sisältävät ensimmäiset suoritukset pois. Kaikkein merkittävimpään nopeutukseen päästiin Numba-käännetyllä NumPyn dot-operaatiolla, joka nopeutti suoritusta jopa 252-kertaisesti. Numba-käännetyllä Python-list-

operaatiollakin saavutettiin 131-kertainen nopeus. NumPyn omalla dot-operaatiolla saavutettiin 80-kertainen nopeutus.



Kuva 5. Suorituksen nopeutumiskertoimet verrattuna Pythonin listatoteutuksen suoritusaikaan

6 Yhteenveto

Tässä tutkielmassa pyrittiin vastaamaan kysymykseen, että miten tietojenkäsittelyä pystytään tehostamaan Python-ohjelmointiympäristössä. Tutkielmassa esiteltiin eri tapoja tehostamiseen kirjallisuustutkimuksen keinoin, ja havainnollistamalla joidenkin keinojen käyttöä suorituskykyvertailun avulla. Aluksi esiteltiin Pythonia ja sen tulkkeja ja JIT-kääntäjiä, joilla voidaan joissain tapauksissa tehostaa Python-koodin suorittamista. Sitten esiteltiin tieteellisen Python-ekosysteemin perustaa, NumPya, joka on optimoitu tarjoamaan tehokkaita tieto- ja kirjastorakenteita. Tämän jälkeen esiteltiin Pythonin rinnakkaisen ja hajautetun suorituksen menetelmiä, joista merkittävä osa perustuu NumPyn korvaamiseen niin sanotuilla ”drop-in” -korvaajilla. Näillä suurille laskentaklustereille skaalautuvilla välineillä on esitettyjen tutkimuksien perusteella pystytty merkittävästi tehostamaan tietojenkäsittelyä Python-ohjelmointiympäristössä. Tehostusta saadaan myös kirjoittamalla kaikki merkittävän laskennallisen taakan omaavat työt joko käännetyssä laajennusmoduulissa tai esimerkiksi Cythonilla. Täytyy myös huomata, että pelkästään yksittäisten algoritmien koodaaminen mahdollisimman tehokkaiksi saattaa olla monissa tilanteissa riittävää, jolloin muita välineitä tehostamiseen ei tarvita.

Tutkielman suorituskykyvertailussa havaittiin NumPyn ja Numban suorituskykyä tehostava vaikutus vektorilaskennan skalaarituloon verrattuna Pythonin vakiolistan käyttöön silmukassa. Vaikka vertailu suoritettiin normaalilla pöytätietokoneella, niin suhteellisen yksinkertaisilla keinolla saavutettiin vektorilaskentaan merkittävää tehostusta. Kaikilla algoritmeilla ja ohjelmilla vastaaviin lukemiin ei kuitenkaan näillä menetelmillä voi päästä ja silloin saattavat muut menetelmät olla parempia.

Joku saattaa olla sitä mieltä, että tässä tutkielmassa on jotain olennaista jäänyt mainitsematta ja toinen sitä mieltä, että jokin on saanut liikaa huomiota. Aihepiiri on varsin laaja ja käytettävien teknologioiden hyöty riippuu käyttökohteesta. Tässä työssä haluttiin esittää mahdollisimman ajankohtaiseen ja vertaisarvioituun tutkimukseen perustuvaa tietoa, koska ala on jatkuvassa nopeassa kehityksessä. Uusia toimijoita syntyy suurin lupauksin ja poistuu sellaista vauhtia, että yksittäisen kehittäjän on vaikeaa pysyä kaiken teknologian kehityksen mukana. Kaikesta päätellen Python halutaan yhä kilpailukykyisemmäksi vaihtoehdoksi tehokkaaseen tietojenkäsittelyyn myös tulevaisuudessa.

Lähdeluettelo

ACM Digital Library. https://dl.acm.org/ (Viitattu 21.11.2021)

Ajitsaria, A., What Is the Python Global Interpreter Lock (GIL)? https://realpython.com/python-gil/ (Viitattu 7.12.2021)

Anaconda. https://www.anaconda.com/ (Viitattu 24.11.2021)

Arkouda 2020.7.7. https://pypi.org/project/arkouda/ (Viitattu 24.11.2021)

Bauer, M. & Garland, M. (2019). Legate NumPy: accelerated and distributed array computing. SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, https://doi.org/10.1145/3295500.3356175

Bauer, M., Garland, M., Lee, W., Papadakis, M. & Zalewski, M. (2021). Supercomputing in Python With Legate. Computing in Science & Engineering (Volume: 23, Issue: 4, July-Aug. 1 2021), https://doi.org/10.1109/MCSE.2021.3088239

Ben-Nun, T., Calotoiu, A., de Fine Licht, J., De Matteis, D., Hoefler, T., Lavarini, L., Schneider, T. & Ziogas, A. (2021). Productivity, Portability, Performance: Data-Centric Python. SC '21: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. https://doi.org/10.1145/3458817.3476176

Bodo. https://bodo.ai/ (Viitattu 24.11.2021)

Bohrium. https://bohrium.readthedocs.io/ (Viitattu 24.11.2021)

Computer Science Database (ProQuest). https://www.proquest.com/ (Viitattu 21.11.2021)

Corral, M., Vector Calculus (2021). http://www.mecmath.net/ (Viitattu 8.12.2021)

CuPy. https://cupy.dev/ (Viitattu 10.11.2021)

Cython. https://cython.org/ (Viitattu 22.11.2021)

Dask documentation. https://docs.dask.org/ (Viitattu 13.11.2021)

Entschev, P., Single-GPU CuPy Speedups (Julkaistu 23.7.2019). https://medium.com/rapids-ai/single-gpu-cupy-speedups-ea99cbbb0cbb (Viitattu 9.11.2021)

Feng, H., Xuran, H., Shurenm, L., Tao, L., Kaifeng, Z., Xin, B. & Chao, J. (2021). Algorithm for Improving Processor Utilization in Multi-core Processor Environment by Python Language. 2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). https://doi.org/10.1109/IMCEC51613.2021.9481962C

Gross, T. & Remigius, M. (2019). Reflections on the compatibility, performance, and scalability of parallel Python. DLS 2019: Proceedings of the 15th ACM SIGPLAN Inter-national Symposium on Dynamic Languages. https://doi.org/10.1145/3359619.3359747

Grumpy. https://opensource.google/projects/grumpy (Viitattu 24.11.2021)

Harris, C., Millman, K., van der Walt, S., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M., Brett2, M., Haldan, A., del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T., (2020). Array programming with NumPy. Nature volume 585, pages 357–362. https://doi.org/10.1038/s41586-020-2649-2

IEEE Electronic Library (IEL). https://innovate.ieee.org/ieee-electronic-library-iel/ (Viitattu 21.11.2021)

IEEE Spectrum, Top Programming Languages 2021. https://spectrum.ieee.org/top-programming-languages/#toggle-gdpr (Viitattu 11.11.2021).

IronPython. https://ironpython.net/ (Viitattu 24.11.2021)

Intel Python. https://www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-for-python.html (Viitattu 24.11.2021)

JAX Reference Documentation. https://jax.readthedocs.io/en/latest/ (Viitattu 24.11.2021)

JupyterLab. Project Jupyter. https://jupyter.org/ (Viitattu 24.11.2021)

Jython. https://www.jython.org/ (Viitattu 24.11.2021)

Lam, S., Pitrou, A. & Seibert, S. (2015). Numba: a LLVM-based Python JIT compiler. LLVM '15: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. https://doi.org/10.1145/2833157.2833162

Legate. https://github.com/nv-legate (Viitattu 24.11.2021)

Microsoft .NET. https://dotnet.microsoft.com/ (Viitattu 24.11.2021)

Microsoft. Common Language Runtime (CLR) overview (Julkaistu 15.9.2021) https://dotnet.microsoft.com/ (Viitattu 24.11.2021)

Nuitka. https://nuitka.net/ (Viitattu 24.11.2021)

Numba Documentation. https://numba.pydata.org/numba-doc/latest/index.html (Viitattu 1.10.2021)

NumPy. (2021a) v1.21 Manual. https://numpy.org/doc/1.21/ (Viitattu 21.10.2021)

NumPy. (2021b) numpy.random.randn. https://numpy.org/doc/stable/reference/random/generated/numpy.random.randn.html (Viitattu 23.11.2021)

Numpywren. https://github.com/Vaishaal/numpywren (Viitattu 24.11.2021)

O'Grady, S. The RedMonk Programming Language Rankings: January 2020 (Julkaistu 28.2.2020). https://redmonk.com/sogrady/2020/02/28/language-rankings-1-20/ (Viitattu 11.11.2021).

Pandas. https://pandas.pydata.org/ (Viitattu 24.11.2021)

Phylanx. https://phylanx.stellar-group.org/ (Viitattu 24.11.2021)

Pyjs. http://pyjs.org/ (Viitattu 24.11.2021)

PyPy. https://www.pypy.org/ (Viitattu 24.11.2021)

Python 3.10.0 documentation. https://docs.python.org/3/ (Viitattu 21.10.2021)

Python 3.10.0 Standard Library. https://docs.python.org/3/library/ (Viitattu 11.11.2021)

Python.NET. https://pypi.org/project/pythonnet/ (Viitattu 24.11.2021)

Ray. https://www.ray.io/ (Viitattu 24.11.2021)

Reitz, K. & Sclusser, T. (2016). The Hitchhiker's Guide to Python. O'Reilly Media, Inc.

RPython Documentation. https://rpython.readthedocs.io/en/latest/ (Viitattu 24.11.2021)

ScienceDirect (Elsevier). https://www.sciencedirect.com/ (Viitattu 21.11.2021)

Skulpt. https://skulpt.org/ (Viitattu 24.11.2021)

Smith., R. (2016). Performance of MPI Codes Written in Python with NumPy and mpi4py. 2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC). https://doi.org/10.1109/PyHPC.2016.010

Spartan. https://pythonhosted.org/spartan/ (Viitattu 24.11.2021)

SpringerLink. https://link.springer.com/ (Viitattu 21.11.2021)

Straßel, D., Reusch, P. & Keuper, J. (2020). Python Workflows on HPC Systems. 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). https://doi.org/10.1109/PDP50117.2020.00041T

STX Next (2021). The Most Popular Python Scientific Libraries. https://www.stxnext.com/blog/most-popular-python-scientific-libraries/ (Viitattu 11.11.2021)

Stackless-Python Documentation. https://stackless.readthedocs.io/en/3.6-slp/stackless-python.html (Viitattu 24.11.2021)

Weld. https://www.weld.rs/ (Viitattu 24.11.2021)

Yegulalp, S., What is PyPy? Faster Python without pain (Julkaistu 1.5.2019). https://www.infoworld.com/article/3385127/what-is-pypy-faster-python-without-pain.html (Viitattu 20.10.2021).