Aapo Peiponen

Gravitaatiosimulaatiot

Tietotekniikan kandidaatintutkielma

24. toukokuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Aapo Peiponen

Yhteystiedot: aaalalpe@student.jyu.fi

Ohjaaja: Sanna Mönkölä

Työn nimi: Gravitaatiosimulaatiot

Title in English: Gravitational simulations

Työ: Kandidaatintutkielma

Sivumäärä: 21+0

Tiivistelmä: Tässä tutkielmassa tarkastellaan gravitaatiosimulaatioita, simulaatioiden tehokkuutta ja algoritmeja, joilla simulaatioita voidaan nopeuttaa. Gravitaatiosimulaatioiden suurin ongelma on laskennallinen vaativuus. Suoraan N kappaleen välisiä vuorovaikutuksia laskemalla laskennallinen vaativuus on luokkaa $O(N^2)$. On selvää, että tämä on ongelma suurilla kappalemäärillä. Erilaiset algoritmit, kuten Barnes-Hut puualgoritmi, voivat vähentää aikavaativuutta tuntuvasti. Parhaimmillaan voidaan saavuttaa nopealla moninapamenetelmällä aikavaativuus $O(N)$. Simulaatioita voidaan lisäksi nopeuttaa hyödyntämällä rinnakkaislaskentaa. Tähän soveltuu esimerkiksi Barnes–Hut-algoritmi hyvin.

Avainsanat: gravitaatiosimulaatio, usean kappaleen järjestelmät, Barnes–Hut-algoritmi

Abstract: In this paper we will look at gravitational simulations, effectiviness of simulations and algorithms that can be used to accelerate simulations. The biggest problem gravitational simulatins have is algorithmic complexity. Calculating interactions directly has the algorithmic complexity of $O(N^2)$. It is clear that this becomes a real problem with large numbers of particles in simulation. Different algorithms such as the Barnes-Hut tree algorithm can reduce algorihtmic complexity a lot. At best algorithmic complexity of $O(N)$ can be achieved with Fast Multipole Method. In addition, to achieve speed it is very important to make use of parallel computing. Barnes–Hut-algorithm is well suited for parallel computing.

Keywords: n-body simulations, gravitational simulations, Barnes–Hut-algorithm

Kuviot

Sisältö

1 Johdanto

Tässä kandidaatintutkielmassa pyritään selvittämään, kuinka gravitaatiosimulaatioissa lasketaan kappaleiden liikkeet ja mitä keinoja laskennan tehostamiseen on. Osa tehostamiskeinoista, joita voidaan hyödyntää nopeuttamaan laskentaa hidastavia tilanteita, kuten kahden kappaleen normittaminen (engl. two body regularizarion), jää tutkielman ulkopuolelle. Samoin suhteellisuusteorian vaikutukset erittäin suuren skaalan simulaatioissa jätetään huomiotta.

Ensimmäisessä luvussa käsitellään gravitaatiosimulaatioiden perusteita. Alussa tutustutaan siihen, kuinka kappaleiden välisiä vuorovaikutuksia voidaan laskea ja mitä ongelmia voi tulla laskennassa vastaan. Ongelmat vaihtelevat epätarkkuuksista simulaation rikkoutumiseen. Epätarkkuuksia voidaan hallita muuttamalla simulaation tarkkuutta, joka määräytyy sen mukaan, kuinka usein simulaation sisältämien kappaleiden sijainnit päivitetään. Voidaan esimerkiksi ohjelmoida simulaatio niin, että kappaleiden sijainteja päivitetään useammin kun kappaleet ovat lähellä toisiaan. Silloin vältetään epätarkkuuksia, joita voi syntyä lähiohitusten aikana. Toinen keino on muokata kaavaa, jolla vuorovaikutuksia lasketaan niin, että lähiohitukset eivät aiheuta yhtä suuria ongelmia. Tällöin kylläkin menetetään jonkin verran tarkkuutta.

Toisessa luvussa tutustutaan siihen, kuinka voidaan laskea kappaleiden välisiä vuorovaikutuksia hieman tehokkaammin. Luvussa käydään läpi Barnes–Hut-algoritmi, jonka perusidea on, että kaukaiset kappaleet eivät vaikuta yhtä vahvasti kuin läheiset kappaleet. Tästä seuraa että kaukaisia kappaleita voidaan kohdella ikään kuin yhtenä suurena kappaleena. Tällainen lähestymistapa säästää laskentatehoa menettämättä liikaa tarkkuutta. Yleisesti menetelmiä, joissa tehdään ero kaukaisten ja läheisten kappaleiden välille, kutsutaan naapurimenetelmiksi.

Kolmannessa luvussa tarkastellaan, kuinka gravitaatiosimulaatioita voidaan nopeuttaa jakamalla laskenta useiden prosessoreiden kesken. Teknisiin yksityiskohtiin ei mennä, vaan aihetta tarkastellaan yleisellä tasolla. Johtuen gravitaatiosimulaatioden suuresta laskentatehon tarpeesta, rinnakkaislaskennan hyödyntäminen on välttämätöntä kaikissa nykyaikaisissa si-

mulaatioissa. Luvun lopussa mainitaan joitakin suurimmista gravitaatiosimulaatioista tähän mennessä. Viimeisenä on lyhyt yhteenveto kaikista esitellyistä asioista.

2 Usean kappaleen järjestelmät

Gravitaatiosimulaatiot ovat simulaatioita, joissa mallinnetaan ajan myötä kehittyviä järjestelmiä, joissa vaikuttava voima on painovoima. Portegies (2007) listaa, että tällaisia järjestelmiä ovat esimerkiksi aurinkokunnat, tähtijoukot, galaksit, galaksijoukot ja erikokoiset järjestelmät aina koko maailmankaikkeuteen asti. Gravitaatiosimulaatioiden avulla pystytään vastaamaan erilaisiin kysymyksiin tähtitieteellisten järjestelmien kehityksestä ja kehityksen eri vaiheista. Esimerkiksi aurinkokuntia koskevissa simulaatioissa voidaan tarkastella aurinkokunnan syntyä ja kehitystä pienemmistä kappaleista suurempiin kappaleisiin (Sharp 2016). Simulaatiot tähtijoukoista voivat kertoa, kuinka pitkään tähtijoukkojen muodostumisessa kestää ja kuinka kauan ne säilyvät. Suuremmissa simulaatioissa pyritään usein vastaamaan materian jakautumiseen liittyviin kysymyksiin. Keskeisin osa simulaatiossa, huolimatta sen koosta, on kappaleeseen kohdistuvan voiman laskeminen. Järjestelmiä, joissa on enemmän kuin kolme kappaletta, kutsutaan usean kappaleen järjestelmiksi. Tässä luvussa tarkastellaan, kuinka kappaleisiin kohdistuvia voimia voidaan laskea usean kappaleen järjestelmässä.

2.1 Vuorovaikutusten laskeminen

Jokainen kappale simulaatiossa kohdistaa voimaa kaikkiin muihin kappaleisiin. Tämän voiman suuruus määräytyy voimaa kohdistavan kappaleen massasta ja kappaleiden välisestä etäisyydestä. Mitä lähempänä kappaleet ovat toisiaan, sitä suurempi voima on. Vastaavasti pidempi etäisyys heikentää voimaa. Lasketun voiman avulla voidaan laskea kappaleen sijainti jonakin tulevana ajankohtana. Yhteen kappaleeseen kohdistuvan voiman laskeminen vaatii jokaisesta muusta kappaleesta siihen kohdistuvien voimien summauksen. Kappaleeseen kohdistuva voima saadaan seuraavasta yhtälöstä (Gualandris 2007):

$$F_i = m_i a_i = -Gm_i \sum_{j=1;j\neq i}^{N} \frac{m_j(r_i - r_j)}{|r_i - r_j|^3}. \tag{2.1}$$

Kaavassa (2.1) i on tutkittavan kappaleen indeksi, j on sen kappaleen indeksi jonka vaikutusta tutkittavaan kappaleeseen halutaan selvittää, r on kappaleen paikkavektori, G on gravitaa-

tiovakio, m on massa ja a on kiihtyvyys. Laskemalla yksittäiseen kappaleeseen kohdistuva voima, saadaan selville myös siihen kohdistuva kiihtyvyys, nopeus ja tuleva paikka. Kaavan käyttö SI-järjestelmän yksiköillä ei ole tehokasta aurinkokuntaa suuremmissa simulaatioissa, johtuen tähtitieteellisten massojen ja etäisyyksien valtavasta koosta. Monesti lasketaankin etäisyydet parsekeissa tai astronomisissa yksiköissä ja gravitaatiovakio skaalataan sen mukaan.

Kaavan (2.1) nimittäjässä lasketaan kappaleiden välisen etäisyyden kuutio. Tästä seuraa, että kahden kappaleen törmätessä toisiinsa päädytään tilanteeseen, jossa jaetaan nollalla. Tällaisessa tilanteessa voidaan jättää törmäys huomiotta tai tehdä jotain muuta, riippuen siitä kuinka tärkeää on mallintaa törmäyksiä. Yksittäisten törmäysten merkitys kuitenkin vähenee suuremmilla skaaloilla (Capuzzo-Dolcetta 2012).

Johtuen etäisyyksien koosta, ei ole tehokasta laskea kappaleisiin kohdistuvia voimia uudelleen ja uudelleen jatkuvasti. Merkittävät muutokset kappaleisiin kohdistuvissa voimissa tapahtuvat pitkillä aikaväleillä, joten kappaleiden radat eivät koe äkillisiä muutoksia (Aarseth 2003, s. 18). Kuten aiemmin mainittu, kappaleeseen kohdistuvan voiman perusteella saadaan selville kappaleen kiihtyvyys, jonka perusteella voidaan laskea missä kappale tulee olemaan tietyn ajan kuluttua. Johtuen muutosten hitaudesta, on tarpeenmukaista valita jokin sopiva ajanjakso, jonka välein kappaleisiin kohdistuvat voimat ja niiden pohjalta lasketut arvot päivitetään. Tätä ajanjaksoa kutsutaan aika-askeleeksi.

Aika-askeleen pituus määrittää simulaation tarkkuuden. Liian suuri aika-askel johtaa tarkkuuden menetykseen, kun taas liian pieni aika-askel johtaa turhaan laskentaan. Aika-askeleen pituutta ei myöskään kannata vakioida, koska siitä voi seurata epätarkkuuksia. Esimerkiksi, jos kaksi kappaletta ohittaa toisensa lähietäisyydeltä, niiden lähestyessä toisiaan niihin kohdistuvat voimat kasvavat erittäin suuriksi. Seurauksena tästä kappaleiden nopeudet kasvavat hallitsemattomasti. Vastaavasti kappaleiden etääntyessä toisistaan niiden nopeudet pienenevät, koska kappaleet vetävät edelleen toisiaan puoleensa. Liian suurella aika-askeleella saatetaan kokonaan hypätä ohi vaiheesta, jossa kappaleet etääntyvät toisistaan, koska kappaleen tuleva sijainti lasketaan kiihtyvyyden ja vauhdin perusteella. Tästä seuraa tilanne, joka näyttäytyy simulaatiossa kappaleen nopeuden hallitsemattomana kasvuna aina kun tapahtuu lähiohitus. Tätä voidaan lieventää muuttamalla aika-askelta aina kappaleiden ollessa riittä-

vän lähellä toisiaan. Koko simulaation aika-askeleen muuttaminen kahden kappaleen ohittaessa toisiaan olisi kuitenkin erittäin tehotonta. Tehokkaampaa onkin määritellä jokaisella kappaleelle yksilöllinen aika-askel (engl. *individual time-step*), joka määrää kappaleelle sen ajan, jonka välein sen sijaintia päivitetään (Aarseth 2003, s. 22). On kuitenkin tehokkaampaa käyttää lohkoaika-askelta (engl. *block time-step*), koska se sallii usean kappaleen käsittelemisen yhtä aikaa. Lohkoaika-askel perustuu siihen, että jokainen aika-askel on jokin kakkosen kerroin, jolloin muodostuu ikään kuin ryhmiä, joilla on sama päivitysaika (Gualandris 2007). Tästä on hyötyä erityisesti rinnakkaislaskennassa, jossa on ideana jakaa tehtävä useiden prosessoreiden kesken suoritettavaksi samanaikaisesti. Ryhmittelemällä kappaleet lohkoihin edellä mainitulla tavalla saadaan hyödynnettyä rinnakkaislaskentaa paljon tehokkaammin, kuin määrittelemällä jokaiselle kappaleelle oma aika-askel.

Huolimatta aika-askeleen muuttamisesta dynaamisesti, eli tilanteen mukaan, saatetaan silti törmätä ongelmiin kappaleiden ohittaessa toisensa mielivaltaisen läheltä. Yksi keino estää nopeuden kasvaminen hallitsemattomasti tällaisessa tilanteessa, on lisätä kaavan nimittäjään pehmennysparametri, joka on jokin suurehko luku (Capuzzo-Dolcetta 2012). Sen tarkoituksena on kasvattaa nimittäjän arvoa, ja siten pienentää kokonaisvoimaa. Luonnollisesti tämä antaa epärealistisia tuloksia ohituksen suhteen, mutta mikäli simulaatiossa ei ole olennaista mallintaa lähiohituksia tarkasti niin tämä on hyvä vaihtoehto. Toinen vaihtoehto on hyödyntää analyyttisia ratkaisumalleja, jotka pätevät kahden kappaleen välisiin vuorovaikutuksiin, ja jättää muut kohteet ohituksen ajaksi huomiotta. Analyyttiset ratkaisut kahden kappaleen välisille vuorovaikutuksille voivat olla tarpeen esimerkiksi tilanteissa, joissa muodostuu binäärijärjestelmiä tai useamman kappaleen muodostamia vakaita järjestelmiä (Portegies Zwart 2007). Jos taas lähiohituksilla ei ole mitään merkitystä simulaation kannalta, voidaan jättää liian läheisten kappaleiden toisiinsa kohdistamat voimat kokonaan huomiotta ja siten ohittaa koko ongelma.

2.2 Ratkaisujen tarkkuus

Kappaleiden liike on jatkuvaa, mutta tietokoneella ei kyetä muunlaiseen kuin diskreettiin laskentaan. Toisin sanoen kappaleen sijainti voidaan tietää täydellisellä tarkkuudella ainoastaan aloitushetkellä. Poikkeuksena tähän sääntöön on järjestelmät, joissa on kaksi kappaletta.

Lisäksi kyetään laskemaan kappaleiden liikkeitä tarkasti erikokoisissa järjestelmissä, joissa säilyy tasapaino, tai joissa on tietynlaisia symmetrioita. Tällaisesta järjestelmästä esimerkki on kolmen kappaleen järjestelmä, jossa kolmas kappale pysyy levossa kahden suuremman kappaleen suhteen. Paikkoja, joissa kolmas kappale voi pysyä levossa kolmen kappaleen järjestelmässä, on viisi kappaletta. Niitä kutsutaan Lagrangen pisteiksi. Reaalimaailman esimerkkeinä näistä pisteistä ovat Jupiterin kiertoradalla sijaitsevat asteroidit, jotka pysyvät levossa Jupiterin ja Auringon suhteen (Boekholt 2015). Suuremmissa järjestelmissä joudutaan tyytymän numeerisiin ratkaisuihin. Numeerinen laskenta sisältää aina virheitä, jotka kertaantuvat ajan myötä.

Usein simuloitaessa erittäin suuria kokonaisuuksia jätetään pois yksittäiset pienet kappaleet, ja simuloidaan sen sijaan massoja, jotka vastaavat suurta määrää pienempiä kappaleita. Simulaation tarkkuus ei kärsi tällaisesta yksinkertaistuksesta, ja samalla säästyy kallisarvoista laskentatehoa. Hyviä esimerkkejä tällaisesta kappaleiden vähentämisestä ovat simulaatiot, joissa mallinnetaan galaksijoukkoja. Yksittäisillä tähdillä ei tällaisessa skaalassa ole suurta merkitystä. Vielä tätä suurempia ovat simulaatiot, joissa mallinnetaan materian jakautumista universumissa (Springel 2005). Tällaisissa simulaatioissa yksittäinen kappale saattaa vastata jopa kokonaista galaksia.

Virheiden vaikutusta voidaan arvioida tarkistamalla noudattaako simulaatio energian säilymisen periaatetta. Mittaamalla simulaation alussa järjestelmän kokonaisenergia, ja vertaamalla sitä kokonaisenergiaan simulaation päätyttyä, saadaan selville kuinka paljon numeeriset virheet ovat vaikuttaneet lopputulokseen. Järjestelmän kokonaisenergia saadaan kaavasta (Aarseth 2003, s. 7)

$$E = T + U + W, \tag{2.2}$$

jossa E on järjestelmän kokonaisenergia, T on järjestelmän kaikki kineettinen energia, U on järjestelmän potentiaalienergia ja W on järjestelmään ulkopuolelta kohdistuva energia. Useissa simulaatioissa järjestelmään ulkopuolelta kohdistuva energia on 0.

Simulaatioita voidaan tarkastella myös tilastollisesta näkökulmasta. Siinä missä yksittäinen

simulaatio saattaa olla laskentavirheiden johdosta väärässä, suuri joukko simulaatioita samasta tilanteesta kerääntyy todellisen ratkaisun ympärille (Boekholt 2015). Gravitaatiosimulaatioiden nopeuttamiseksi on kehitetty useita eri algoritmeja, joita sovelletaan myös muihin osa-alueisiin kuin tähtitieteeseen. Seuraavassa luvussa tutustutaan yhteen näistä.

3 Algoritmit

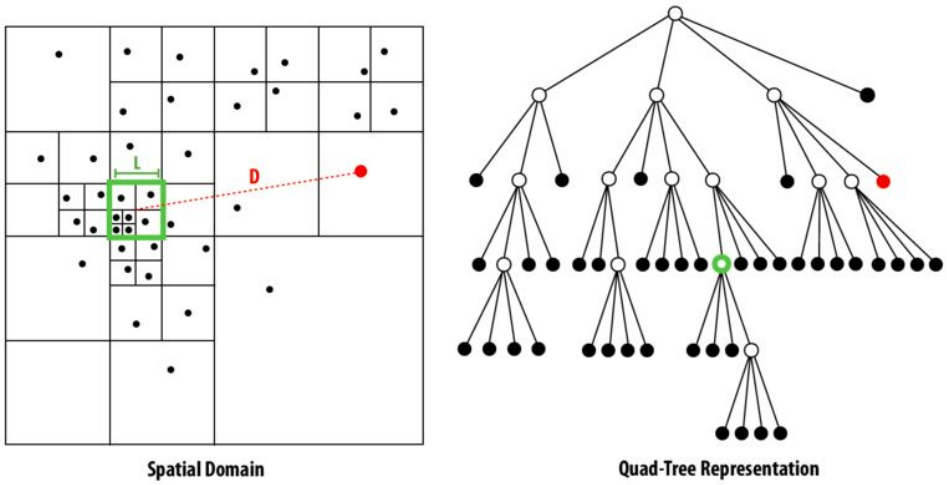
Gravitaatiosimulaatioiden nopeuttamiseksi on useita erilaisia algoritmeja. Suurimmassa osassa perusidea on sama. Vähennetään laskemista vähentämällä laskettavia vuorovaikutuksia. Kaikkien tähtien ja kappaleiden kohdistamat voimat on laskettava tarkkojen tulosten takaamiseksi, mutta niitä ei ole pakko laskea yksitellen. Sen sijaan voidaan lajitella tutkittavaa tähteä ympäröivät tähdet läheisiin ja kaukaisiin tähtiin. Lasketaan läheisten tähtien kohdistama vaikutus yksitellen, ja kaukaisten tähtien vaikutus ryhminä. Barnes–Hut-algoritmi hyödyntää tätä naapuriperiaatetta. Toinen algoritmi, joka hydyntää tätä, on nopea moninapamenetelmä (engl. *Fast Multipole Method*). Nopea moninapamenetelmä on lyhyemmin kirjoitettuna FMM. Tässä menetelmässä ei lasketa suoraan voimia, vaan lasketaan voimien sijaan potentiaaleja (Grama 1998). FMM hyödyntää myös naapuriperiaatetta, kuten Barnes–Hut-algoritmi. Molemmissa jaotellaan tähdet läheisiin ja kaukaisiin ryhmiin, ja lasketaan vuorovaikutukset sen perusteella.

3.1 Barnes–Hut-algoritmi

Suurilla kappalemäärillä yksittäisten vuorovaikutusten merkitys vähenee. Törmäyksettömät algoritmit ovat siis hyödyllisiä suurilla kappalemäärillä. Törmäyksettömän algoritmin olisi myös hyvä olla nopeampi kuin suora laskenta, mutta kuitenkin riittävän tarkka realistisuuden säilytykseen. Tässä luvussa käsitellään algoritmia, joka täyttää nämä vaatimukset. Kyseessä on puualgoritmi, nimeltä Barnes–Hut-algoritmi.

Algoritmin perusidea on käsitellä kaukaisia tähtimassoja yhtenä pisteenä, jonka massa vastaa kyseisen alueen tähtien massoja. Tämä on toimivaa, koska yksittäisten kaukaisten tähtien vaikutus tutkittavaan kappaleeseen on melko heikko. Yksittäisten tähtien sijaan pitkillä etäisyyksillä vaikuttaa enemmän se, missä sijaitsee kaukaisten tähtien massakeskipiste. Kaukaisten tähtien tai muiden kappaleiden vaikutus tutkittavaan kappaleeseen voidaan sitten laskea tämän massakeskipisteen perusteella.

Aarsethin (2003, s. 95) mukaan algoritmin kulku on seuraavanlainen. Ensin luodaan kuutio, jonka sisälle mahtuvat kaikki simulaation sisältämät tähdet. Sen jälkeen tämä laatikko jaetaan



Kuvio 1. Barnes-Hut puun rakenne[1]

kahdeksaan yhtä suureen osaan. Mikäli kyse on kaksiulotteisesta avaruudesta jaetaan neliö neljään osaan. Sen jälkeen käydään jokainen uusi solu läpi, ja mikäli kyseisen solun sisällä on enemmän kuin yksi tähti, jaetaan se jälleen kahdeksaan osaan. Tätä jatketaan, kunnes jokainen tähti on oman solunsa sisällä. Lopputuloksena on kahdeksanpuu (engl. *octree*).

Algoritmi 1 Puun rakentaminen ja solujen tietojen tallennus

if tähtien lukumäärä alueella > 1 **then**

Jaa alue kahdeksaan osaan ja suorita tämä funktio jokaiselle osalle erikseen.

Tallenna alueen massaksi alempien alueiden massan summa.

Tallenna alueen massakeskipisteeksi alempien alueiden painotettu keskiarvo.

else

Tallenna alueen massaksi tähden massa ja massakeksipisteeksi tähden sijainti.

Jokainen puun solu tallentaa tiedon sen sisältämästä massasta ja massakeskipisteestä (Grama 1998). Tämä tarkoittaa, että puun rakenteen luomisen jälkeen, puu käydään rekursiivisesti läpi. Solujen massakeskipiste määräytyy solun sisältämien massakeskipisteiden perusteella, ja vastaavasti solun massa on sen sisältämien massojen summa. Alimmalla tasolla solun massakeskipiste ja massa määräytyvät solun sisältämän ainoan tähden perusteella. Nyt, kun jokaisella solulla ja niiden sisältämällä alisolulla, on valmiiksi tiedossa massakeskipiste ja massa, ei niitä tarvitse laskea joka kerta uudelleen yksittäiseen kappaleeseen kohdistuvaa voimaa laskettaessa (Aarseth 2003, s. 95).

Voimia laskettaessa valitaan tähti ja lähdetään liikkeelle juurisolusta. Mikäli käsiteltävän solun massakeskipisteen etäisyys tutkittavasta kappaleesta on riittävä, lasketaan vaikuttava voima kyseisen solun massakeskipisteen ja massan perusteella, jättäen huomiotta sen mahdollisesti sisältämät muut solut. Sopiva etäisyys määräytyy luvun θ perusteella, joka on solun koon l ja tutkittavan kappaleen ja solun massakeskipisteen välisen etäisyyden D suhde l/D (Aarseth 2003, s. 95). Mikäli näiden lukujen suhde on pienempi kuin θ, käsitellään kyseisiä tähtiä yhtenä kappaleena. Muutoin tutkittavaa solua avataan pidemmälle. Valittu arvo määrää simulaation tarkkuuden. Mikäli θ on nolla, simulaatio vastaa perinteistä suoraa laskentaa, eikä toivottua nopeutumista saavuteta. Mikäli se taas on suuri, menetetään tarkkuutta. Sopiva arvo sijaitsee jossain yhden ja nollan välillä. Mitä lähempänä nollaa arvo on, sitä enemmän menetetään aikaa simulaatiossa. Jos taas arvo on yksi tai vielä sitä suurempi, simulaatioon syntyy epätarkkuuksia jotka summautuvat ajan myötä.

Algoritmi 2 Vuorovaikutusten laskeminen

if $l/D < \theta$ **then**

Laske vuorovaikutus suoraan alueen massakeskipisteen ja massan perusteella.

else

Suorita sama tutkittavan alueen sisältämille alueille.

1. Lähde: http://15418.courses.cs.cmu.edu/spring2013/article/18

4 Suurlaskenta ja tehokkuus

4.1 Laskennan rajat

Usean kappaleen järjestelmään liittyviä differentiaaliyhtälöitä ei voida ratkaista analyyttisesti, joten niissä täytyy käyttää numeerisia menetelmiä. Tästä seuraa, että simulaatio on pakko suorittaa askeltaen. Ei voida siis laskea suoraan alkuasetelman perusteella, miltä tähtijoukko näyttää esimerkiksi kymmenen miljoonan vuoden kuluttua. Lisäksi simulaatiot vaativat huomattavan monta askelta, ottaen huomioon, että tähtitieteessä kiinnostavat aikaskaalat lasketaan miljoonissa vuosissa. Näin suurissa aikaskaaloissa syntyy varsin suuri määrä laskettavia askeleita simulaatioon.

Aikavaativuus usean kappaleen järjestelmien suorassa simuloinnissa on luokkaa O(N^2). Tämä johtuu siitä, että jokaisen tähden liikkeiden laskeminen vaatii kaikkien muiden tähtien läpikäynnin laskennassa. Tähtijoukkojen simulointi yhdellä tietokoneella realistisella tarkkuudella on siis todella raskasta. Rinnakkaislaskenta antaa kuitenkin mahdollisuuden simuloida varsin suuria määriä kappaleita tehokkaammin.

Rinnakkaislaskennan lisäksi tehokkaammat algoritmit tarjoavat mahdollisuuden suorittaa suurempia simulaatioita nopeammin. Winkelin (2012) mukaan esimerkki tällaisesta tehokkaasta algoritmista on Barnes–Hut-algoritmi, joka vähentää aikavaativuuden luokkaan $O(NlogN)$. Tätä algoritmia hyödynnetään esimerkiksi tähtijärjestelmien kertymäkiekkojen simuloinnissa. Vielä nopeammin voidaan suorittaa simulaatio, jos järjestelmässä on joukko suurimassaisia kappaleita, ja huomattavasti suurempi määrä pienempiä kappaleita. Tätä asetelmaa voidaan lähestyä jättämällä pois pienten kappaleiden väliset vuorovaikutukset, jolloin simulaation laskennallinen vaativuus putoaa luokkaan $O(MN)$, jossa M on massiivisten kappaleiden määrä, ja N on pienten kappaleiden määrä. Toisessa versiossa tästä otetaan huomioon simulaatiossa myös pienten kappaleiden vaikutus suuriin kappaleisiin. Tällöin laskennallinen vaativuus kasvaa kaksinkertaiseksi verrattuna versioon, jossa pienet kappaleet eivät vaikuta suuriin kappaleisiin.

Laskenta supertietokoneilla eri puolilla maailmaa verkon välityksellä on myös varteenotet-

tava vaihtoehto. Suorittamalla simulaatio ympäri maailmaa sijaitsevilla supertietokoneilla, menetettiin vain noin 13 % tehosta viiveen vuoksi, verrattuna vastaavaan määrään laskentatehoa vain yhdessä paikassa (Groen 2011).

4.2 Rinnakkaislaskenta

Rinnakkaislaskenta on tärkeimpiä kehityssuuntia gravitaatiosimulaatioissa. Algoritmien tehokkuus voi auttaa vain tiettyyn pisteeseen asti. Kehittämällä algoritmeja, jotka ovat sekä tehokkaita, että soveltuvat rinnakkaislaskentaan, voidaan saavuttaa entistä suurempia simulaatioita. Myös yksittäisten vuorovaikutusten mallintamisen tarkkuus on merkityksellistä simulaatioissa, riippuen minkäkokoista skaalaa simuloidaan. Vuorovaikutusten tarkkaa simulointia varten on kehitetty erilaisia menetelmiä, jotka mahdollistavat vuorovaikutusten suoran laskennan hyödyntäen rinnakkaislaskentaa. Rinnakkaislaskenta kykenee lieventämään suoran laskennan suurta aikavaativuutta, mutta luonnollisesti nopeammat algoritmit, kuten edellisessä luvussa esitelty Barnes–Hut-algoritmi, sallivat suurempien simulaatioiden ajamisen samassa ajassa, kuin suora laskenta.

Barnes–Hut-algoritmin toiminta voidaan jakaa kahteen osaan. Puun rakentamiseen, ja sen läpikäymiseen. Rakennusvaiheessa muodostetaan puun solut, ja tallennetaan jokaiseen soluun sen massa ja massakeskipiste. Läpikäyntivaiheessa lasketaan kappaleiden väliset vuorovaikutukset edellisessä luvussa esitellyllä tavalla. Rinnakkaislaskennassa voidaan hyötyä siitä, että erotetaan puuta läpikäyvä prosessi voimia laskevasta prosessista, ja luodaan oma erillinen prosessi toimimaan kommunikaatioväylänä näiden välille. Tällä tavalla saavutetaan nopeusetu, koska puun läpikäynnin ja voimien laskennan ei tarvitse olla synkronoituna keskenään. Kommunikaatioväylä pitää huolen, ettei ristiriitatilanteita synny. Samaa ideaa voidaan hyödyntää hajautetussa laskennassa, ja vähentää latenssin haitat minimiin (Winkel 2012). Tietokoneet pystyvät silloin kommunikoimaan keskenään ja vaihtamaan tietoja, vaikka laskenta tai puun läpikäynti olisi meneillään.

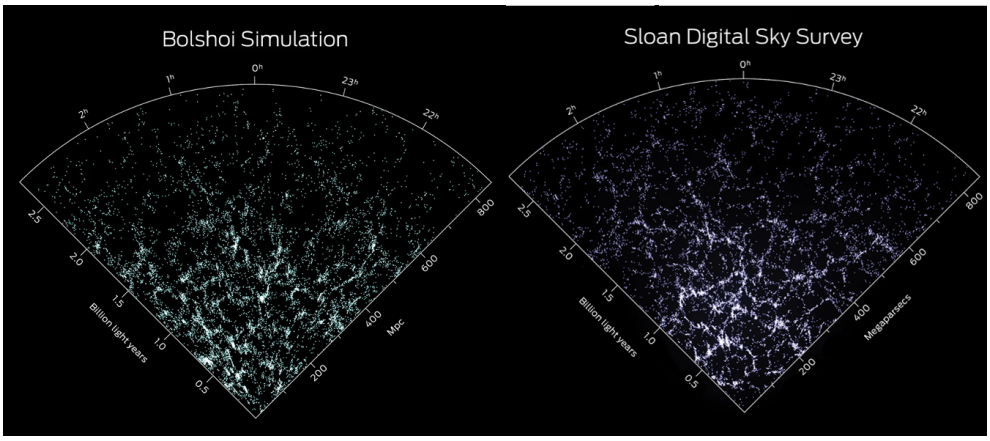
Bédorf (2012) mukaan aiemmin hyödynnettiin laskennassa erityisiä vuorovaikutusten laskemiseen tarkoitettuja järjestelmiä, joista käytettiin lyhennettyä nimeä GRAPE (GRAvity pipE). Tämä järjestelmä on erikoistunut laskemaan kappaleiden välisiä vuorovaikutuksia no-

peasti ja tehokkaasti. Nykyään tavalliset näytönohjaimet pystyvät kuitenkin tehokkaampaan laskentaan.

4.3 Simulaatioiden kasvava koko

Simulaatioiden koko on jatkuvassa kasvussa. Yhä suuremmat simulaatiot mallintavat sekä suurempia skaaloja, että suurempia aikavälejä. Näillä kosmologisilla simulaatioilla pyritään selvittämään maailmankaikkeuden rakenteen kehittymistä ajan myötä, sekä kehitystä ohjanneita voimia.

Suuren skaalan simulaatioiden hyödyllisyyden puolesta puhuu alla oleva kuva, jossa on rinnakkain Bolshoi-simulaatiosta leikattu pala, ja vastaavan kokoinen pala varsinaisia havaintoja. Se että simulaatiot maailmankaikkeuden rakenteesta osuvat näin lähelle varsinaisia havaintoja, osoittaa että käsityksemme maailmankaikkeuden kehitystä muovaavista voimista ovat kohtuullisen tarkkoja. Parannettavaa tietysti on. Kuvista näkeekin, että aivan samalla tavalla materia ei ole jakautunut, vaikka siltä ensi silmäyksellä vaikuttaisikin. Tavoitteena onkin, että tulevaisuudessa gravitaatiosimulaatiot avaisivat esimerkiksi pimeän aineen luonnetta (Springel 2005).



Kuvio 2. Bolshoi-simulaation tulokset verrattuna havaintoihin.[1]

Millenium-simulaatio sisälsi yli 10 miljardia kappaletta. Sen tarkoituksena oli selvittää aineen jakautumista universumissa suuressa skaalassa, kuten galaksien kerääntymistä ja yk-

1. Lähde: http://hipacc.ucsc.edu/Bolshoi/IEEEFeature2012.html

sittäisten himmeiden galaksien syntyprosessia (Springel 2005). Sitä käytettiin esimerkiksi vertailukohtana varsinaisiin havaintoihin galaksien jakautumisesta universumissa. Bolshoi-simulaatio on myöhemmin ajettu simulaatio, jossa pyrittiin tarkempiin tuloksiin kun havaittiin, että Millenium ei täysin vastannut uusimpia havaintoja. Näistä kahdesta simulaatiosta Bolshoi-simulaatio on huomattavasti lähempänä nykyisiä havaintoja (Klypin 2011). Kappaleiden lukumäärän suhteen suurin simulaatio tähän mennessä on suoritettu Kiinassa Sunway TaihuLight -supertietokoneella, jossa simuloitiin noin 10 biljoonan eri kappaleen liikkeitä.[2]

2. https://www.scmp.com/news/china/society/article/2104297/chinese-scientists-create-biggest-virtual-universe-worlds-fastest

5 Yhteenveto

Gravitaatiosimulaatiot ovat simulaatioita, joissa mallinnetaan ajan myötä kehittyviä järjestelmiä, joissa vaikuttava voima on painovoima. Kappaleiden liikkeet lasketaan selvittämällä niiden toisiinsa kohdistamat voimat, ja laskemalla niiden perusteella jokaiselle kappaleelle kiihtyvyys. Kiihtyvyyden ja kappaleen aika-askeleen avulla voidaan laskea kappaleen uusi sijainti. Merkityksellistä kappaleiden liikkeiden laskennassa on aika-askeleen pituuden valinta. Kappaleille on kannattavaa määritellä dynaaminen aika-askel, joka määritellään sen perusteella, kuinka nopeasti kappale liikkuu, ja kuinka kaukana se on toisista kappaleista. Rinnakkaislaskennan kannalta tehokkainta on käyttää lohkoaika-askelta, jolloin useiden kappaleiden liikkeitä voidaan laskea yhtä aikaa.

Suurin pullonkaula gravitaatiosimulaatioissa on voimien laskemisen aikavaativuus. Laskennan nopeuttamisessa voidaan hyödyntää erilaisia algoritmeja, jotka nopeuttavat kappaleiden välisten vuorovaikutusten laskentaa. Esimerkki tällaisesta algoritmista on Barnes–Hut-algoritmi. Se perustuu läheisten ja kaukaisten kappaleiden erotteluun laskennassa. Barnes–Hut-algoritmin paras aikavaativuus on O($nlogn$). Rinnakkaislaskenta on tehokkaita algoritmejakin parempi tapa nopeuttaa laskentaa. Erilaisia menetelmiä suorittaa laskentaa rinnakkain on toteutettu sekä suoralle laskennalle, että muille menetelmille.

Simulaatioiden koko jatkaa edelleen kasvuaan, vaikkakin ennätyksiä ei rikota joka vuosi. Suurilla simulaatioilla pyritään selvittämään maailmankaikkeuden rakennetta ja kehitystä ajan myötä. Tulevaisuudessa tutkijat pyrkivät simulaatioiden avulla rajaamaan pimeän aineen luonnetta.

Lähteet

Winkel, Mathias. 2012. “A massively parallel, multi-disciplinary Barnes–Hut tree code for extreme-scale N-body simulations” [kielellä eng]. *Computer Physics Communications* 183 (4): 880–889. https://jyu.finna.fi/PrimoRecord/pci.sciversesciencedirect_elsevierS0010-4655(11)00401-2.

Aarseth, Sverre J. 2003. *Gravitational N-body simulations.* Cambridge monographs on mathematical physics. Cambridge ; New York: Cambridge University Press. ISBN: 0511065442.

Bédorf, Jeroen. 2012. “A sparse octree gravitational N-body code that runs entirely on the GPU processor” [kielellä eng]. *Journal of Computational Physics* 231 (7): 2825–2839.

Boekholt, Tjarda. 2015. “On the reliability of N-body simulations” [kielellä eng]. *Computational Astrophysics and Cosmology* 2 (1): 1–21.

Capuzzo-Dolcetta, R. 2012. “A fully parallel, high precision, N-body code running on hybrid computing platforms” [kielellä eng]. *Journal of Computational Physics* 236 (1). https://jyu.finna.fi/PrimoRecord/pci.sciversesciencedirect_elsevierS0021-9991(12)00690-0.

Grama, Ananth. 1998. “Scalable parallel formulations of the Barnes–Hut method for n-body simulations” [kielellä eng]. *Parallel Computing* 24 (5): 797–822. https://jyu.finna.fi/PrimoRecord/pci.sciversesciencedirect_elsevierS0167-8191(98)00011-8.

Groen, Derek. 2011. “High-performance gravitational n -body simulations on a planet-wide-distributed supercomputer” [kielellä eng]. *Computational Science Discovery* 4 (1): 015001.

Gualandris, Alessia. 2007. “Performance analysis of direct N-body algorithms for astrophysical simulations on distributed systems” [kielellä eng]. *Parallel Computing* 33 (3): 159–173.

Klypin, Anatoly A. 2011. “Dark matter halos in the standard cosmological model: results from the bolshoi simulation” [kielellä eng]. *The Astrophysical Journal* 740 (2): 102. https://jyu.finna.fi/PrimoRecord/pci.iop10.1088%2F0004-637X%2F740%2F2%2F102.

Portegies Zwart, Simon F. 2007. “High-performance direct gravitational N-body simulations on graphics processing units” [kielellä eng]. *New Astronomy* 12 (8): 641–650.

Sharp, P. W. 2016. “GPU-enabled N-body simulations of the Solar System using a VOVS Adams integrator” [kielellä eng]. *Journal of Computational Science* 16:89.

Springel, Volker. 2005. “Simulations of the formation, evolution and clustering of galaxies and quasars”. *Nature* 435 (7042): 629. https://jyu.finna.fi/PrimoRecord/pci.nature_a10.1038%2Fnature03597.