KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
MULTIMEDIJOS INŽINERIJOS KATEDRA

Žygimantas Melinskas

DAUGELIO KŪNŲ SĄVEIKOS LYGIAGREČIŲ ALGORITMŲ TYRIMAS

Baigiamasis magistro projektas

Darbo vadovas:
Doc. Dr. Romas Marcinkevičius

KAUNAS, 2016

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

DAUGELIO KŪNŲ SĄVEIKOS LYGIAGREČIŲ ALGORITMŲ TYRIMAS

Baigiamasis magistro projektas

Informatika M4016L21

Darbo vadovas:
..... Doc. Dr. Romas Marcinkevičius
.....
Recenzentas:
..... Doc. Dr. Stasys Maciulevičius
.....

Projektą atliko:
..... Žygimantas Melinskas
.....

KAUNAS, 2016

Melinskas, Žygimantas. Daugelio kūnų sąveikos lygiagrečių algoritmų tyrimas. Magistro baigiamasis projektas / vadovas doc. dr. Romas Marcinkevičius; Kauno technologijos universitetas, Informatikos fakultetas.

Mokslo kryptis ir sritis: Informatika, Skaičiuojamoji Astrofizika

Reikšminiai žodžiai: gravitacija, CUDA, hierarchinis, N-Body, GPU.

Kaunas, 2016. 75 p.

SANTRAUKA

Skaičiuojamosios astrofizikos srityje dažnai atliekamas daugelio kūnų sistemos modeliavimo eksperimentas. Tobulėjant skaičiavimus atliekančiai techninei įrangai, didelės apimties ($n \geq 10^6$) modeliai tampa prieinamesni. CUDA platforma gali susidoroti su dideliais modeliais, tačiau specialių algoritmų pagalba įmanoma išgauti pastebimai daugiau spartos. Šiame darbe identiškomis sąlygomis yra lyginamos kelios specializuotos medžio kodo algoritmo realizacijos, pritaikytos CUDA platformai. Buvo stebima kaip jos susidoroja su skirtingos formos dalelių erdvėmis esant skirtingiems atidarymo kampams θ (jei taikytina). Darbas yra užbaigiamas pateikiant realizacijų įvertinimą bei kelis įdomius pastebėjimus.

Melinskas, Žygimantas. Analysis of Concurrent N-body interaction algorithms: Master‘s thesis in Informatics / supervisor assoc. prof. Romas Marcinkevičius. The Faculty of Informatics, Kaunas University of Technology.

Research area and field: Informatics, Computational Astrophysics

Key words: gravity, CUDA, hierarchical, N-Body, GPU.

Kaunas, 2016. 75 p.

SUMMARY

An N-Body simulation is commonplace within the field of computational astrophysics. With an ever increasing grade of computational hardware available at our disposal, it is becoming easier to run large simulations ($n \geq 10^6$). CUDA is a platform that can successfully accommodate a large N-Body simulation, but it too benefits from clever algorithms, such as the tree-code. In this paper a black-box comparison is done between several CUDA enabled tree-codes on a single hardware configuration. The algorithms were subjected to a combination of different spatial configurations and opening angles θ (were applicable). The paper concludes with an overall ranking of the compared algorithms and a few interesting observations.

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos Fakultetas

Žygimantas Melinskas

Informatika, M4016L21

Baigiamojo projekto „Daugelio kūnų sąveikos lygiagrečių algoritmų tyrimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

2016m. gegužės 20d.

Patvirtinu, kad mano, **Žygimanto Melinsko**, baigiamasis projektas tema „Daugelio kūnų sąveikos lygiagrečių algoritmų tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

TURINYS

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

TERMINŲ ŽODYNAS

Device	„Įrenginio“ terpė, vykdanti jai būdingą programinį kodą. Darbo kontekste „įrenginys“ dažniausiai reikš CUDA palaikančią nVidia vaizdo plokštę.
Host	„Šeimininko“ terpė, vykdanti jai būdingą programinį kodą. Darbo kontekste „šeimininkas“ dažniausiai reikš kompiuterio pagrindinį procesorių.
CUDA	(angl. Compute Unified Device Architecture) nVidia kompanijos sukurta platforma ir programavimo modelis, leidžiantis kurti programas, kurios yra vykdomos vaizdo plokštėje.
GPU	Graphics Processing Unit. Grafinės plokštės procesorius; „Įrenginio“ terpė.
SIMD	Single Instruction Multiple Data. Lygiagrečių procesorių klasė, kurioje viena instrukcija galima vykdyti daugiau negu vienam duomenų junginiui. Paprasto procesorius SUM instrukcija gali sumuoti tiktai vieną skaičių a su kitu skaičiu b ir taip gauti sumą c. SIMD su SUM instrukcija galima sumuoti masyvą $\left\{A\right\}$ su masyvu $\left\{B\right\}$ ir taip gauti sumą $\left\{C\right\}$. CPU įvykdžius vieną instrukciją galima gauti vieną sumą, o SIMD procesoriuje $\left\lvert\left\{C\right\}\right\rvert$ sumas.
Kernelis	Taip yra vadinama CUDA funkcija, kuri yra vykdoma GPU terpėje ir yra iškviečiama iš CPU terpės.
Gardelė	Nusako blokų kiekį ir kaip bus pasiskirste jų indeksai vykdant kernelį. Paduodama kaip specialus parametras kerneliui.
Blokas	Nusako kiek gijų ir kaip bus pasiskirste jų indeksai, vykdant GPU kodą.

ĮVADAS

Kompiuterinis modeliavimas parodo, kaip bet kokia, formaliai apibūdinama sistema evoliucionuoja laike. Pakankamai tikslaus modelio rezultatai suteikia galimybę daryti prielaidas, kurios gali būti pritaikomos fizinei sistemai. Pavyzdžiui: detalus konstrukcijos vidinių įtempimų modeliavimas gali parodyti, kuriose vietose fizinė konstrukcija yra silpniausia. Taipogi modeliavimas gali padėti įrodyti arba paneigti keliamą hipotezę. Astrofizikos šakoje vis dar vyksta tyrimai, parodantys kaip tiksliai evoliucionuoja įvairios kosminės struktūros (planetos, žvaigždės, žvaigždžių klasteriai ir t. t.). Kosminių struktūrų modeliavimas yra daugialypis procesas, kurio metu reikia spręsti tokias problemas, kaip kūnų orbitų nustatymas, žvaigždžių evoliucijos modeliavimas (temperatūros, dydžio, tankio ir kitų parametrų pokyčiai), kūnų kolizijų išsprendimas, skystųjų kūnų dinamika (žvaigždė nėra kietasis kūnas, dėl to ji turi būti modeliuojama pasitelkus hidrodinamikos principus) ir pan. Šiame magistro darbe bus analizuojami metodai ir algoritmai, skirti nustatyti taškinių kūnų jaučiamoms jėgoms gravitacinio potencialo lauke (t. y. orbitoms modeliuoti), esant dideliam kūnų kiekiui N. Toks daugelio kūnų jėgų įvertinimas yra žinomas kaip daugelio kūnų problemos (angl. *N-Body problem*) sprendimas.

Tyrimo objektas ir aktualumas Vienas iš problemos sprendimo būdų yra sistemos modeliavimas, o tai yra atliekama naudojant specialius algoritmus. Egzistuoja kelios algoritmų klasės, kurios sugeba atlikti daugelio kūnų modeliavimą ir darbe bus sutelktas dėmesys į vieną iš jų — medžio pavidalo kodo (anlg. treecode) klasę. Taipogi bus tyrinėjamos tiktai CUDA platformos realizacijos. CUDA yra labai prieinama technologija ir ją taikant galima skaičiuoti didesnius modelius sparčiau negu bendrojoje platformoje. Šiuo metu nebuvo rastas panašus palyginamasis darbas. Asmeniui, kuris yra suinteresuotas N-kūnų modeliavimu, būtų aktualu matyti, kuris algoritmas kaip veikia esant tam tikroms sąlygoms. Tiktai, kai yra žinomi esminiai algoritmų privalumai ir trūkumai, yra įmanoma pasirinkti tinkamiausią algoritmą esamai užduočiai.

Tikslas ir uždaviniai Darbo tikslas yra palyginti medžio kodo algoritmų realizacijas CUDA platformai. Uždaviniai:

- Atlikti probleminės srities analizę: nustatyti medžio kodo ypatumus ir surasti skirtumus tarp kitų problemos sprendimo metodų.
- Surinkti ir išanalizuoti algoritmo CUDA realizacijas, nustatyti didžiausius egzistuojančius tarp jų skirtumus, išsiaiškinti kuo skiriasi algoritmo realizacijos nuo bendrosios platformos realizacijos.
- Nustatyti esminius palyginimo kriterijus.
- Nustatyti esmines pradinių duomenų formas. T. y. su kokiais duomenimis bus atliekamas palyginimas.
- Atlikti eksperimentą ir interpretuoti rezultatus.

Dokumento struktūra Darbas yra skirstomas į du pagrindinius skyrius — analizės ir eksperimento. Analizes skyriuje yra išanalizuojami pagrindiniai daugelio kūnų problemos sprendimo būdai. Didelis dėmesys yra suteikiamas medžio kodo algoritmui — išsiaiškinami jo principai. Taipogi analizės skyriuje bus apžvelgiamos įvairios medžio kodo realizacijos CUDA platformai. Eksperimento skyriuje yra aprašomas lyginimo eksperimentas. Jame bus aptariama eksperimento eiga, naudojami duomenys, bei kokybės kriterijai pagal kuriuos bus lyginami algoritmai. Eksperimento skyrius bus užbaigiamas eksperimento rezultatų demonstravimu ir interpretavimu.

1. DAUGELIO KŪNŲ PROBLEMOS SPRENDIMO METODŲ ANALIZĖ

Analizės tikslas yra išsiaiškinti skaičiuojamosios astrofizikos srityje padarytus pasiekimus, suformuluotus algoritmus bei kitus sprendimo metodus, skirtus spręsti daugelio kūnų problemą. Tiktai susidarius tvirtą žinių pagrindą, bus galima suformuluoti palyginimo eksperimentą.

Egzistuoja kelios platformos, kuriose galima efektyviai atlikti daugelio kūnų modelius:

- Superkompiuteriai.
- Kompiuterių klasteris.
- Bendroji (CPU).
- Grafinė (GPU).

Darbe analizuojami algoritmai ir jų realizacijos yra skirtos bet kokiam asmeniui, kuris yra suinteresuotas N-kūnų modeliavimu. Dėl to, terpės pasirinkimą priklauso nuo jos prieinamumo. Spartos ir modelio apimties atžvilgiu, superkompiuteris būtų idealiausias variantas, tačiau jis yra mažiausiai prieinamas resursas. Teoriškai, bet koks kompiuterių kambarys (klasė, ofisas) gali būti paverstas į kompiuterių klasterį, tačiau dažniausiai tai yra atliekama tiktai universitetuose arba didesnėse kompanijose. Klasteris yra lengviau prieinamas variantas negu superkompiuteris, bet neaplenkia CPU arba GPU. Bendroji ir grafinė terpės yra pačios prieinamiausios, tačiau grafinė terpė turi besąlygiškai didesnius skaičiavimo išteklius, dėl to darbe yra tyrinėjamos algoritmų realizacijos, skirtos grafinėms plokštėms.

1.1. TYRIMO SRITIS, OBJEKTAS IR PROBLEMA

Sritis N-kūnų problema yra apibendrinta sąvoka, pritaikoma sprendžiant daugelio kūnų gravitacinę sąveiką (astrofizika), molekulių sąveiką (chemija), krūvių sąveiką (elektrostatika) ir kt. Kadangi šis magistrinis darbas yra labiau orientuotas į N-kūnų problemos sprendimą gravitacinei sąveikai, magistrinio darbo tyrimo sritis yra skaičiuojamoji astrofizika.

Objektas Tyrimo objektas yra įvairūs algoritmai ir metodai, skirti modeliuoti gravitacinę sąveiką astrofizikos kontekste (gravitacinė sąveika tarp žvaigždžių, planetų, asteroidų, t. y. bet kokių, didelę masę turinčių, kosminių kūnų).

Problema Problema yra algoritmo sparta. Pats paprasčiausias algoritmas, skirtas gravitacijos modeliavimui, turi $O\left(N^2\right)$ lygio sudėtingumą. Šioje srityje paneigiant ar patvirtinant hipotezę, reikia modelio su dideliu $N \geq 10^6$ ir $O\left(N^2\right)$ algoritmas, dėl savo sudėtingumo, tampa nenaudotinu. Dėl to atsirado aproksimacijos metodai, kurie sugeba panašią sistemą sumodeliuoti sparčiau, tačiau

su prarastu tikslumu, kuris, nors ir nėra labai griežtas reikalavimas, turi priimtiną ribą. Dėl to būtų galima patikslinti problemą taip: Gravitacinės sąveikos modeliavimo algoritmų sparta ir jų atiduodamo rezultato tikslumas.

1.2. ESAMŲ PROBLEMOS SPRENDIMO METODŲ ANALIZĖ

1.2.1. Algoritmai

Egzistuoja nemažai algoritmų, skirtų spręsti daugelio kūnų problemą. Kaip ir kitose algoritmų klasėse, kiekvienas iš jų turi skirtingus kokybinius įverčius (skiriasi sudėtingumas, tikslumas ir pan.). Toliau bus paaiškinti ir apibendrinti keli algoritmai, kurių grafinės terpės realizacijos bus analizuojamos vėlesnėje darbo dalyje.

1.2.1.1. Tiesioginio sumavimo algoritmas

Prieš atsirandant aproksimacijos algoritmams tiesioginis sumavimas (angl. *direct, pair-wise, all-pairs*) buvo vienintelis būdas modeliuoti gravitacines sistemas. Dėl $O(n^2)$ sudėtingumo N dalelių kiekis modeliuose buvo stipriai apribotas, o tai reiškia, kad modeliai buvo grubūs ir interpretuojant jų rezultatus reikdavo labiau „pasinaudoti“ vaizduote. Tačiau tiesioginis sumavimas nėra išnykęs. Jis yra naudojamas praktiškai kiekviename algoritme, sudėtingesniame negu bazinis BH variantas §1.2.1.2., įvertinti labai arti esančių kūnų gravitacinei traukai, nes pagrindinis aproksimacijos komponentas dažniausiai veikia labai prastai esant mažiems tarpdaleliniams atstumams [2, 3, 6, 11]. Tiesioginio sumavimo algoritmas yra pats paprasčiausias algoritmas, kuris gali būti panaudotas modeliuojant sistemą susidedančią iš daugelio taškinių kūnų, sąveikaujančių tarpusavyje kažkokio potencialo lauke. Kadangi darbo sritis yra skaičiuojamoji *astrofizika*, dėmesys bus sutelktas į gravitacinę sąveiką, bus daroma prielaida, kad dalelės sąveikauja gravitacinio potencialo lauke ir pačios formulės vertins gravitacinę trauką.

$$F_{ij} = G\frac{m_i m_j}{\left\|\vec{r}_{ij}\right\|^2} \cdot \frac{\vec{r}_{ij}}{\left\|\vec{r}_{ij}\right\|} \textbf{kur: } \vec{r}_{ij} = \vec{x}_j - \vec{x}_i \tag{1.1}$$

1.1 yra vektoriais išreikšta Niutono gravitacinės sąveikos formulė. Jėga F_{ij}, kuri veikia i-tajį kūną, yra vektorius, nukreiptas nuo i-tojo kūno į j-tajį kūną. Vektoriaus ilgis parodo jėgos stiprį ir yra proporcingas kūnų masių ir gravitacinės konstantos sandaugai ir yra atvirkščiai proporcingas atstumo tarp kūnų *kvadratui*.

$$F_i = \sum_{j=1, j\neq i}^{N} F_{ij} = Gm_i \sum_{j=1, j\neq i}^{N} \frac{m_j \vec{r}_{ij}}{\left\|\vec{r}_{ij}\right\|^3} \tag{1.2}$$

Turint lauką, kuriame yra N dalelių, galime surasti jėgą F_i jaučiamą i-tojo kūno tame lauke pagal lygtį 1.2. Lauko, kuriame yra daugiau negu viena dalelė, gravitacinis potencialas yra nustatomas superpozicijos principu. T.y. atskirų dalelių, esančių tame pačiame lauke, gravitaciniai potencialai gali būti paprasčiausiai susumuojami, norint išgauti viso lauko gravitacinio potencialo lygtį. Analogiškai veikia ir jėgų laukas, dėl to lygtyje 1.2 matome sumavimo operatorių. Modelyje nemodeliuojame gravitacinės sąveikos tarp i-tojo kūno ir jo paties, dėl to turime praleisti $j = i$ (F_{ii}) dėmenį, antraip $\left\|\vec{r}_{ij}\right\|^3 = 0$ ir lygtis tampa neapibrėžiama. Modeliuojant išreikštiniu (angl. *explicit*) būdu, kiekviename integracijos periode yra atnaujinamos visų kūnų pozicijos. Siekiant atnaujinti poziciją $\vec{u}$, reikia žinoti kūno greitį $\vec{v}$, kuris yra sužinomas iš pagreičio $\vec{a}$. Pagreitis $\vec{a}$ yra apskaičiuojamas pagal Niutono antrąjį dėsnį:

$$\overrightarrow{F} = m\vec{a} \;\Leftrightarrow\; \vec{a} = \frac{\overrightarrow{F}}{m} \tag{1.3}$$

Tarp integracijos periodų reikia persinešti kūnų poziciją $\vec{u}$, greitį $\vec{v}$ ir pagreitį $\vec{a}$. Vieno periodo metu yra paskaičiuojamos kiekvieno kūno jaučiamos jėgos, kurios yra konvertuojamos į momentinius pagreičius. Pagal momentinius pagreičius, atsižvelgiant į tai, koks yra naudojamas integravimo metodas, yra vienaip ar kitaip atnaujinami kūnų pagreičiai, greičiai ir yra apskaičiuojami momentiniai poslinkiai. Integracinio periodo gale momentiniai poslinkiai yra prisumuojami prie kūnų pozicijų ir procesas yra kartojamas iš naujo. Reikia atkreipti dėmesį, kad tarp periodų nereikia saugoti kūno jaučiamos jėgos $\overrightarrow{F}$, dėl to lygtį 1.2 galima suprastinti toliau:

$$\vec{a}_i = G \sum_{j=1, j\neq i}^{N} \frac{m_j \vec{r}_{ij}}{\left\|\vec{r}_{ij}\right\|^3} \tag{1.4}$$

Esminis skirtumas tarp 1.4 ir 1.2 yra toks, kad 1.4 nebereikia kreipti dėmesio į i-tojo kūno masę. Būtų galima padaryti pastebėjimą: kūno pagreitis, generuojamas gravitacinio potencialo lauko, nepriklauso nuo to kūno masės.

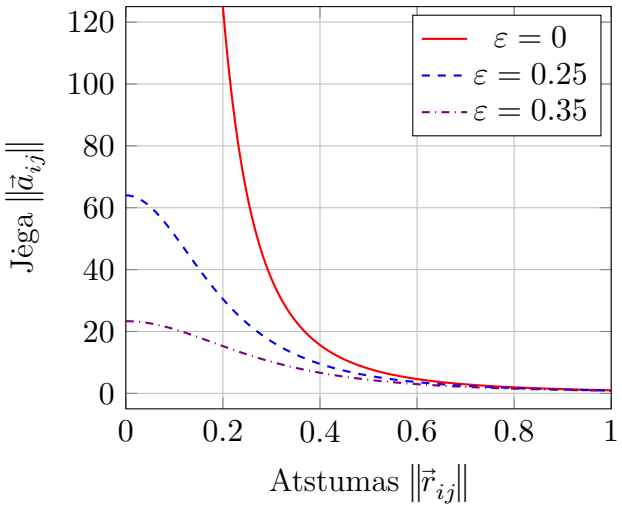
Modeliuojant N-kūnų sistemą tiesioginio sumavimo metodu, vienos iteracijos metu kiekvienam kūnui reikia įvertinti jėgas su kitais $N - 1$ kūnais, dėl to algoritmo sudėtingumas yra $O\left(N^2\right)$. Yra galimybė išnaudoti simetriškumą tarp jėgų: $\overrightarrow{F}_{ij} = -\overrightarrow{F}_{ji}$, taip sumažinant skaičiavimų apimtį per pusę.

Pažvelgus į 1.4 lygtį, būtų galima rasti dar vieną papildomą problemą: sparčiai diverguojanti a_i reikšmė, kai $\left\|\vec{r}_{ij}\right\|$ artėja į 0. Modeliai su dideliais N dalelių kiekiais, bent jau kol kas, yra vykdomi išreikštiniu būdu: pagal esančią sistemos padėtį yra apskaičiuojamos kūnų jaučiamos jėgos ir vėliau poslinkiai. Po integracijos, bet kuri kūnų pora gali atsidurti labai arti. Realybėje tokie kūnai arba susidurtų arba prasilenktų išlaikydami energijos kiekį, tačiau modelyje, jeigu būtų naudojama 1.4 lygtis, jiems būtų paskaičiuojama labai didelė jėga, kas lemtų labai didelį poslinkį. Tokiu atveju, po integracijos, kūnai atsidurtų labai toli vienas nuo kito su *labai* dideliu greičiu — įvyktų neproporcingas energijos prieaugis, kas yra nerealistiška. Kad to būtų išvengta

yra pritaikomas sušvelninimo parametras ε:

$$\vec{a}_i = G \sum_{j=1}^{N} \frac{m_j \vec{r}_{ij}}{\left(\left\| \vec{r}_{ij} \right\|^2 + \varepsilon^2 \right)^{\frac{3}{2}}} \tag{1.5}$$

Kai $\varepsilon = 0$, lygtis 1.5 yra tas pats kas 1.4 ir dėl asimptomiškumo jos reikšmė artėja į begalybę, kai jos argumentas artėja prie 0: $\lim_{\|\vec{r}_{ij}\| \to 0} \left\| \overrightarrow{a_{ij}} \right\| = \infty$. Kai $\varepsilon > 0$, lygties vardiklis nebegali būti 0 ir dėl to jos reikšmė visuomet konverguoja.



1.1 pav. ε įtaka gravitacijos lygčiai

Paveikslas 1.1 iliustruoja ε įtaką lygties 1.5 reikšmei (daroma prielaida kad $N = 1, m_1 = \frac{1}{G}$). Reiktų atkreipti dėmesį, kad nebereikia tikrinti sąlygos $j \neq i$, nes kai $j = i$ dėmuo yra lygus 0.

1.1, 1.2 1.4 ir 1.5 formuluotės yra paimtos iš [9].

1.2.1.2. Barneso-Huto hierarchinis algoritmas

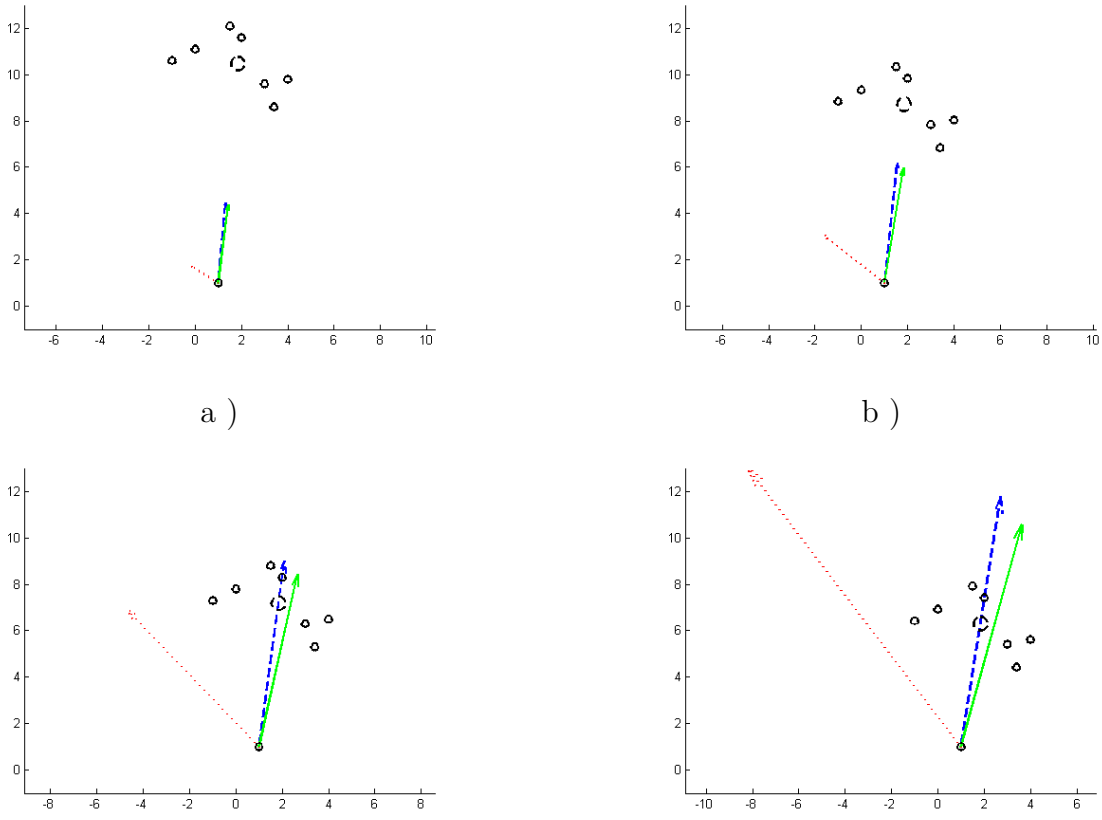
Barneso-Huto (toliau BH) hierarchinis algoritmas aproksimuoja gravitacinę trauką, pasinaudodamas vėlesniame paragrafe apibūdinama savybe [1]. Algoritme yra naudojamas aštuonmedis (kaip dvinaris medis, tačiau kiekvienas mazgas turi 8 pomazgius) sistematiniam erdvės suskaidymui į tiksliai apibrėžtus regionus. Aštuonmedžio struktūra palengvina bendrosios algoritmo paklaidos ir atstumo tarp vieno taško ir erdvės regiono nustatymą. Atstumo nustatymas yra svarbus, norint efektyviai aproksimuoti kiekvieno kūno jaučiamą traukos jėgą. Medžio pavidalo struktūroje kiekvienas mazgas atvaizduoja erdvės regioną. Regionas yra kubo formos ir yra apibūdinamas savo matmenimis, pozicija erdvėje, mase ir masės centro pozicija. Regiono masė ir masės centras yra apskaičiuojami pagal regione esančius kūnus.

Panašūs aproksimacijos algoritmai egzistavo jau prieš BH algoritmą, tačiau juose naudojami

erdvės skaidymo metodai nėra tokie lankstūs, kaip BH skaidymo metodas [1]. Pagrindiniai jų trūkumai yra tai, kad sunku apibūdinti jų skaidymo kriterijus (t. y. ne visuomet aišku, *kaip* bus suskaidyta erdvė į regionus) ir dėl to sunku lyginti skaičiavimo rezultatų paklaidas.

Gravitacinio dėsnio savybių išnaudojimas Kai du, masę turintys kūnai, tolsta vienas nuo kito, gravitacinio potencialo kuriama traukos jėga silpnėja. Dėl to analizuojant kūną i, svarbesni yra arti i esantys kūnai. Sakykime, kad erdvėje yra k arti susispietusių kūnų ir pavadinkime juos klasteriu. Į klasterį galima žiūrėti dvejopai: k kūnų su individualia mase ir pozicija erdvėje *arba* vienas kūnas K, kurio masė yra visų klasteryje esančių kūnų masė ir kurio pozicija yra klasterio masės centras. Vertinant traukos jėgą kūnui $i \notin k$ su šiuo klasteriu, mes galime: vertinti traukos jėgą su kiekvienu $j \in k$ kūnu ir ją sumuoti $F_{ik} = \sum_{j \in k} F_{ij}$ *arba* vertinti traukos jėgą tiktai su kūnu K: $F_{ik} \approx F_{iK}$.

Tokia aproksimacija nėra absurdiška, nes vertinant gravitacinę trauką tarp dviejų realių kūnų (pvz., žemės ir saulės), yra dažniausiai apsiribojama šių kūnu centrais ir kiekvieno iš jų mase. Realiai reiktų įvertinti tų kūnų, mažiausiųjų sudedamųjų dalelių jaučiamas jėgas atskirai, ir jas susumuoti. Kadangi Saulė ir Žemė yra pakankamai toli viena nuo kitos, šio supaprastinto spredimo rezultatas praktiškai nesiskiria nuo tikrojo.



a) b)

c) d)

1.2 pav. Aproksimacijos klaidos augimas

Paveikslas 1.2 iliustruoja gravitacinio lauko generuojamą jėgą vienam kūnui. Žalias (vientisas) vektorius — tiesioginio sumavimo metodas, padidintas 50 kartų; Mėlynas (punktyrinis) vektorius — aproksimacijos metodas, padidintas 50 kartų; Raudonas (taškinis) vektorius — skirtumas tarp tiesioginio sumavimo ir aproksimacijos metodų, padidintas 500 kartų; Punktyrinis apskritimas: klasterio masės centras; Klasteris yra toliausiai 1.2a, artėja 1.2b, 1.2c, arčiausiai randasi 1.2d. Klasteriui artėjant, skirtumas tarp aproksimacijos ir tiesioginio sumavimo didėja. Tiesioginis sumavimas suskaičiuoja vektorių labiau paslinkusį į dešinę, nes kūnas, esantis arčiausiai analizuojamojo yra jo dešinėje ir taip turi didesnį indėlį į bendrą traukos jėgą. Reiktų atkreipti dėmesį, kad aproksimacijos vektorius visuomet yra nukreiptas į arba kerta klasterių masės centrą.

Erdvės skaidymas Tarkime, kad turime S kūnų aibę, susidedančia iš s_i kūnų: $s_i \in S$. Kad suskaidyti erdvę, ir taip paruošti modelį tolimesniems skaičiavimams, reikia iš pradžių nustatyti šakninio mazgo centro koordinates $R_c\left(R_{cx}, R_{cy}, R_{cz}\right)$ ir apimtį $R_d\left(R_{dw}, R_{dh}, R_{dd}\right)$. Šakninis mazgas yra kubo formos ir turi aprėpti visus kūnus:

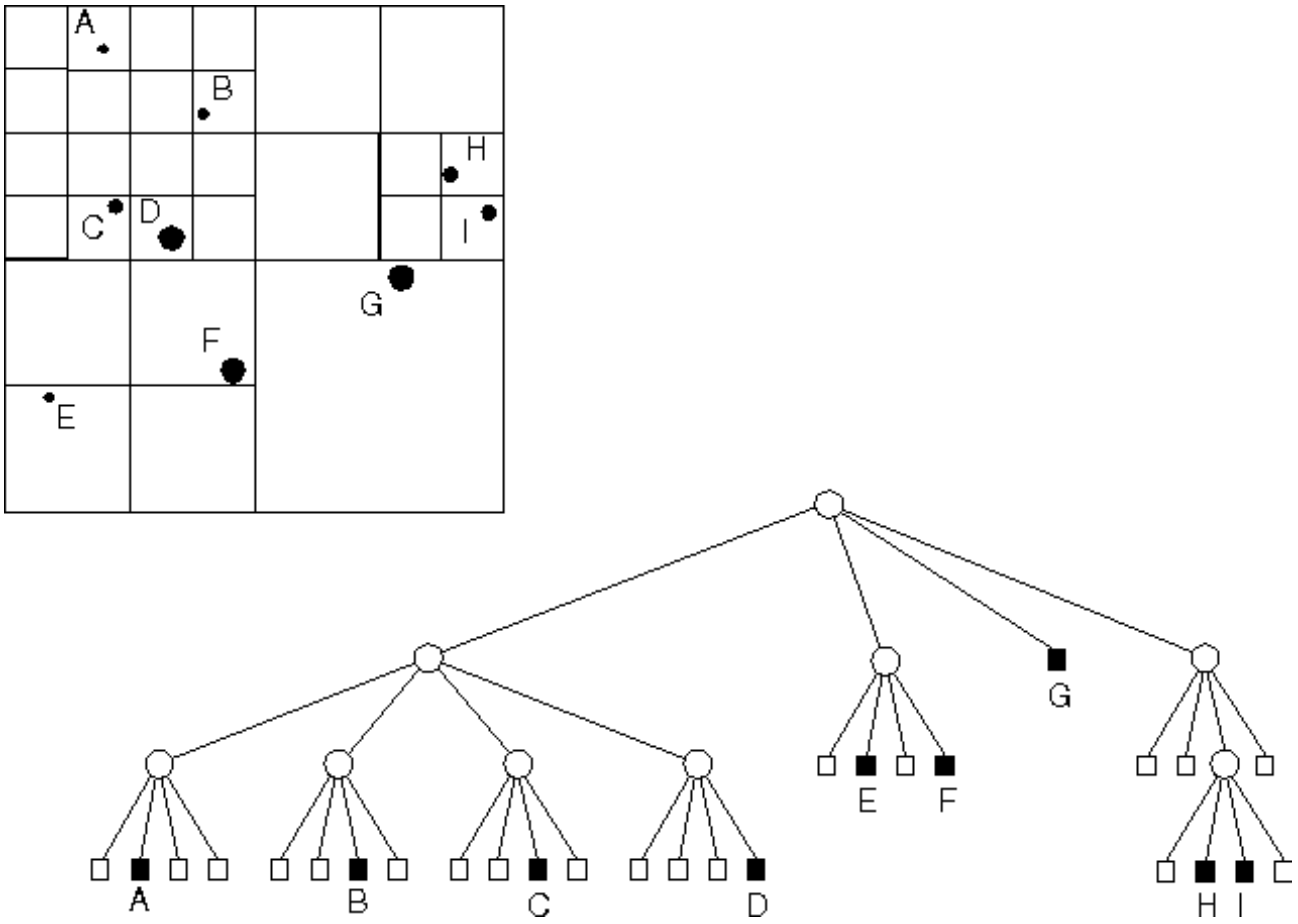
$$R_{dw} = R_{dh} = R_{dd} = \max\left[\left(\max S_x - \min S_x\right), \left(\max S_y - \min S_y\right), \left(\max S_z - \min S_z\right)\right] \quad \textbf{(1.6)}$$

Lygtis 1.6 parodo, kad šakninio mazgo matmenys yra lygūs (i.e. kubas) ir kad briaunos dydis yra *didžiausias* skirtumas tarp didžiausių kūnų s_i pozicijų skirtumų kiekvienos ašies atžvilgiu.

$$R_c\left(\min S_x + \frac{\max S_x - \min S_x}{2}, \min S_y + \frac{\max S_y - \min S_y}{2}, \min S_z + \frac{\max S_z - \min S_z}{2}\right) \quad \textbf{(1.7)}$$

Lygtis 1.7 parodo kaip yra randamos šakninio mazgo centro koordinatės.

Formuojant erdvės paskirstymo medį, į šakninį mazgą yra paeiliui įterpiami visi $s_i \in S$ kūnai. Jeigu bet kokiame mazge yra daugiau negu $n_{crit} = 1$ kūnas, tą mazgą atvaizduojantis kubas yra skeliamas į 8 lygias dalis. Toms dalims yra sukuriami 8 nauji mazgai, kurie yra įstatomi į medį kaip pomazgiai suskaidytam mazgui. Kūnai, kurie buvo perskeltame mazge, yra paskirstomi į pomazgius. Jeigu taip atsitinka, kad tie $n > n_{crit}$ kūnai vistiek patenka į tą patį regioną, skaidymo procedūra yra analogiškai vykdoma tam regionui. Paveiklse 1.3 yra pavaizduojamas erdvės suskaidymo pavyzdys dvimatėje erdvėje. Hierarchinio erdvės skaidymo metodo kontekste dvimatis erdviškumas skiriasi nuo trimačio tik tuo, kad yra dirbama su kvadratais, kurie yra skaidomi į 4 lygias dalis.



1.3 pav. Erdvės suskaidymo pavyzdys dvimatėje erdvėje.

Medžio apėjimas Suskaidžius erdvę, galima paskaičiuoti kiekvieno kūno jaučiamą traukos jėgą. Kiekvienas kūnas yra analizuojamas iš eilės, po vieną. Analizuojant kūną, yra pradedamas aštuonmedžio struktūros apėjimas, pradedant nuo pagrindinio mazgo. Ties kiekvienu žingsniu yra patikrinama, ar mazgas yra pakankamai nutolęs nuo analizuojamo kūno. Jeigu taip – yra paskaičiuojama traukos jėga, jaučiama nuo to mazgo viduje esančio fiktyvaus kūno. Paskaičiuota traukos jėga yra pridedama analizuojamam kūnui ir fiktyvaus kūno mazgas praleidžiamas. Jeigu analizuojamas kūnas yra pakankamai toli nuo mazgo, jis yra "atidaromas" – t. y. atliekamas jo pomazgių apėjimas.

$$\frac{l}{D} < \theta \tag{1.8}$$

Nelygybė 1.8 parodo kada kūnas yra pakankamai toli nuo mazgo: l yra atstumas tarp kūno ir mazgo svorio centro; D yra mazgo kubo briaunos ilgis; θ yra atidarymo kampas. Sumažinant θ yra griežtinamas atstumo reikalavimas, dėl to yra „atidaroma" daugiau mazgų. Kai $\theta = 0$ yra atidaromi visi mazgai ir algoritmo veikimas tampa panašus į tiesioginio sumavimo algoritmą (atlieka tiek pat darbo, kiek ir tiesioginio sumavimo algoritmas, bei visas medžio konstravimo ir apėjimo operacijas) 1.2.1.1.. Atidarymo kampo keitimas leidžia keisti tikslumą į spartą (ir atvirkščiai)

ir daugelis specifinių algoritmo implementacijų turi optimaliausią variantą, pvz., Gaburov et al. naudoja $\theta \approx 0.5$ [6].

Modifikacija Standartinis BH algoritmas atlieka daug operacijų, susijusių su medžio struktūros skaitymu ir rašymu. Šios operacijos, dažniausiai yra realizuojamos pasinaudojant rekursija bei rodyklių gaudymu (angl. *pointer chasing*). Paprastas CPU procesorius šias operacijas vykdo sparčiai, ir dėl to, laikas praleistas manipuliuojant medį, nesukuria akivaizdžios algoritmo našumo problemos.

Tuometiniai (kai buvo aprašyta BH modifikacija) vektoriniai kompiuteriai turėjo daug skaičiuojamųjų resursų, kuriuos buvo galima išnaudoti, spartint daugelio kūnų modeliavimą [2]. Tačiau jų rekursijos ir rodyklių gaudymo operacijos nebuvo tokios sparčios kaip CPU atitikmenys. Dėl to nemodifikuoto BH algoritmo medžio manipuliavimo veiksmai trukdavo daug ilgiau, t. y. tapo pastebimomis spartos problemomis.

Problemą galima spręsti vengiant darbo su medžiu. Tolimesniam modifikacijos paaiškinimui reiktų atlikti medžio apėjimo išskaidymą į dvi fazes [8]:

1: sąveikos sąrašo surinkimas vienai dalelei,
2: sąveikos sąrašo „įvertinimas“/apskaičiavimas.

Tarkime, kad turime erdvės skaidymo medį Q, kuriame yra N dalelių: $k_i \in K; |K| = N$. Pirmosios fazės rezultatas dalelei k_i yra tos dalelės sąveikos sąrašas L_i, kuriame yra medžio mazgai $q_l \in Q$, tenkinantys 1.8. Galima atlikti pastebėjimą: jeigu atstumas tarp dalelės k_i ir mazgo q_l yra daug didesnis, negu atstumas tarp dalelių k_i ir k_j, tuomet yra didelė tikimybė, kad q_l priklauso tiek k_i sąveikos sąraše $q_l \in L_i$, tiek k_j sąveikos sąraše $q_l \in L_j$ [2]. Kitaip tariant, jeigu dalelės k_i ir k_j yra arti, jų sąveikos sąrašai yra labai panašūs: $L_i \approx L_j$. Būtų galima apjungti L_i ir L_j sąrašus į vieną L_c, taip, kad L_c tenkintų 1.8 tiek k_i, tiek k_j atžvilgiu. Sukonstruoti sąrašams L_i ir L_j reikia dviejų dalinių medžio apėjimų, tačiau sukonstruoti L_c reikia vieno truputį ilgesnio dalinio apėjimo: $t(L_i) + t(L_j) \gg t(L_c)$; $\frac{t(L_i)+t(L_j)}{2} < t(L_c)$. Siekiant efektyviai išnaudoti šią savybę, reikia apibrėžti, kokioms dalelių grupėms galima skaičiuoti bendrą sąrašą.

Kaip jau minėta, standartinė BH implementacija medyje suradusi fizinę dalelę k_i, esančią apatiniame mazge, sukonstruoja jai sąveikos sąrašą L_i ir įvertina jame esančių mazgų generuojamą traukos jėgą tai dalelei. Modifikuota BH versija sustoja, suradusi mazgą q_l, kuriame yra $\leq n_{crit}$ dalelių ir joms sugeneruoja bendrą sąveikos sarašą L_c. Taip darant, potencialiai galima sumažinti laiką, praleistą vaikščiojant po medį iki n_{crit} kartų ir tai yra modifikacijos esmė. Reikia atkreipti dėmesį, kad sąrašas L_c savyje *neturi* mazge q_l esančių dalelių, dėl to, įvertinus sąraše esančių mazgų generuojamas traukos jėgas, reikia atlikti ir tiesioginį mazge esančių dalelių sumavimą, kad butų pilnai įvertinta traukos jėga iš visų šaltinių.

Modifikaciją turi kelis poveikius:

- Sumažina laiko kiekį, kurį reikia išleisti dirbant su medžiu.

- Identiški sąveikos sąrašai kelioms dalelėms reiškia, kad traukos vertinimo procedūrą galima idealiai išlygiagretinti ant SIMD principu veikiančio procesoriaus.
- Sąveikos sąrašas kiekvienos dalelės atžvilgiu turi perteklinės informacijos ir tai sąlygoja didesnį tikslumą.
- Atsiranda tiesioginio sumavimo komponentas, kuris irgi yra idealiai išlygiagretinamas.

Šios temos atveju labiausiai įdomus poveikis yra antras: skaičiavimų identiškumas, nes CUDA platformoje 32 gijos (sugrupuotos) gali vykdyti kodą *tuo pačiu metu*, jeigu vykdomas kodas yra identiškas kiekvienai gijai. Vėliau pamatysime, kaip kai kurie realizacijų autoriai gudriai pasinaudoja šia savybe.

1.2.1.3. Apibendrinimas

Egzistuoja daugiau algoritmų, skirtų modeliuoti daugelio kūnų sistemas: tinklelio metodai (jų populiarumas sumažėjo, kai pasirodė BH) ir O(n) sudėtingumo daugiapolių (angl. *multipole*) metodas [7]. Analizė ir magistrinis darbas neliečia daugiapolių metodo, nes jis yra ganėtinai sudėtingas, turi mažiau aprašytų CUDA realizacijų ir jų GPU terpės problemos yra panašios į BH.

Vėliau bus analizuojamos esamos BH algoritmo realizacijos. Kai kurios naudoja modifikuotą versiją, kitos išgauna „modifikuotumą“ kitais būdais ir didžioji dalis specialiai naudoja tiesioginį sumavimą apskaičiuoti smulkiausius regionus [2].

1.1 lentelė. Analizuojami algoritmai

Algoritmas	Sudėtingumas	Išlygiagretinamumas	Modelio apimtis
Tiesioginis sumavimas	$O(n^2)$	Idealus.	Maža
Barnes-Hut	$O(n \log n)$	Sudėtingas.	Didelė
Mod. Barnes-Hut	$O(n \log n)$	Vidutinis.	Didelė

1.2.2. CUDA

Grafinė plokštė, dėl jos teikiamų skaičiuojamųjų resursų ir kainos santykio, yra labai patraukli bendros paskirties skaičiavimų platforma. Sparčiai augant kompiuterinių žaidimų industrijai, kilo paklausa ir grafinėms plokštėms. Grafinių procesorių technologijos nuolat sparčiai tobulėja: žaidimų kūrėjai nuolatos konkuruodavo tarpusavyje dėl grafikos realistiškumo ir tai netiesiogiai sukeldavo iniciatyvą grafinių procesorių gamintojams kurti galingesnius ir spartesnius lustus. Grafinių procesorių sparta iki šiol auga, tačiau lustų kainos išlieka prieinamos. Ankščiau buvo įma-

noma pasinaudoti grafinio lusto skaičiuojamąja galia, jeigu problema, kurią buvo norima spręsti, buvo lengvai pakeičiama į grafinę. Tarkime, kad reikia sumodeliuoti temperatūros pasiskirstymą stačiakampio formos plokštėje. Jeigu plokštė būtų suskaidoma į kvadratinius regionus, ji atrodytų kaip grafinė tekstūra, kur kiekvienas regionas yra pikselis, o jo spalva yra regiono temperatūra. Tuomet, pasinaudojant šešėliavimo paprogramėmis (angl. *shader*), galima tos tekstūros pikselius modifikuoti pagal šilumos srautų dėsnius. Šiais laikais grafinių procesorių gamintojai subendrino savo lustų sąsajas taip, kad jomis įmanoma vykdyti bendros paskirties kodą.

Populiariausias lustų gamintojas nVidia turi savo lygiagrečių skaičiavimų platformą bei programavimo modelį CUDA. CUDA suteikia prieigą prie dešimtmečius plėtotų skaičiuojamųjų resursų, esančių naujesnės laidos nVidia grafinėse plokštėse (GeForce, Tesla, Quadro ir t.t.). CUDA turi ir kitų alternatyvų: Brook bei OpenCL, kurios veikia ne tik ant GeForce grafinių plokščių [4, 10]. Magistriniam darbui buvo pasirinkta CUDA platforma, nes, lyginant su kitomis platformomis, ji veikia sparčiau su GeForce plokštėmis. Papildomos priežastys: CUDA yra labai gerai dokumentuota ir egzistuoja daug literatūros susijusios su darbo tyrimo objektu [3, 5, 6, 9, 11].

1.2.2.1. CUDA C

CUDA C kodo failai yra kompiliuojami į paprastus objektinius failus (*.obj plėtinys Windows operacinėje sistemoje, ir *.o plėtinys Linux operacinėse sistemose), kurie gali būti naudojami standartinėse C ir C++ programose, statinėse bibliotekose bei dinaminėse bibliotekose. CUDA C kalboje yra trys funkcijų tipai: standartinės, globalinės (global) ir įrenginio (device) 1.4 . Standartinės funkcijos yra nepakitusios (paprastos C funkcijos); globalinės funkcijos yra vykdomos vaizdo plokštėje, tačiau gali būti iškviestos tiktai iš šeimininko – įrenginio valdančio vaizdo plokštę; įrenginio funkcijos yra vykdomos vaizdo plokštėje ir gali būti iškviestos tiktai iš globalinių funkcijų arba kitų įrenginio funkcijų.

```
void StandartCFunction( void );
__global__ void GlobalCUDAFunction( void );
__device__ void DeviceCUDAFunction( void );
```

1.4 pav. CUDA funkcijų tipai

1.2.2.2. Programos vykdymo architektūra

CUDA programa vykdymo metu yra suskaidoma į blokus, kurie yra toliau skaidomi į gijas. Blokai yra vykdomi nuosekliai vienam procesoriui. Jeigu vaizdo plokštė turi daugiau negu vieną procesorių, blokai yra apdorojami lygiagrečiai. Kviečiant CUDA funkciją, reikia sukonfigūruoti paleidimo erdvę (t. y. kiek, ir kokio dydžio blokų reikia) – tai galima padaryti su specialiu

<<<>>> operatoriumi (žr. pav. 1.5). Pav. 1.5 pirmasis argumentas (32) parodo blokų kiekį, o antrasis (64) – gijų kiekį/bloko dydį. Funkcijos viduje blokų kiekį galime gauti iš specialaus raktažodžio *gridDim.x*, o bloko dydį su *blockDim.x* . Raktažodžiai turi ir y bei z komponentes, nes blokas ir gardelė (angl. *grid*) gali būti dvimačiai ir trimačiai. Toks architektūrinis sprendimas buvo padarytas, nes taip yra lengviau suplanuoti dvimates ar trimates problemas. Pavyzdžiui: norint nuspalvinti kiekvieną 640x480 bitmap failo pikselį pagal pasirinktą algoritmą, būtų galima iškviesti CUDA funkciją su [x:20, y:15] dydžio gardele, kurioje kiekvienas blokas būtų [x:32, y:32] gijų dydžio.

```
__global__ void Add( int * a, int * b, int * c, int N ) {
  int tid = blockIdx.x * gridDim.x + threadIdx.x;
  if ( tid < N ) {
    c[tid] = a[tid] + b[tid];
  }
}
int main( int argc, char * argv[] ) {
  ...
  Add<<<32,64>>>( vec1, vec2, vec3, 32 * 64 );
  ...
}
```

1.5 pav. CUDA programos pavyzdys

1.2.2.3. Atmintis

Atminties operacijos yra pakankamai brangios (apie 600 ciklų), dėl to programa, kuri neefektyviai naudoja atmintį, praleis daug laiko laukdama kol bus rašoma arba skaitoma iš atminties. CUDA architektūroje yra papildomų darbinės atminties tipų/koncepcijų, suteikiančių galimybę programoms pasiekti vaizdo plokštės teorinį maksimalų duomenų pralaidumą.

- Bendroji Atmintis (angl. *Shared Memory*). Šios atminties yra nedidelis kiekis (jis gali būti konfigūruojamas ir dažniausiai būna apie 48 kilobaitus), nes ji yra L1 kešo viduje (L1 kešo dydis dažniausiai būna 64 kilobaitai). Ši atmintis yra išskiriama, kai GPU pradeda apdoroti bloką ir ja gali naudotis visos gijos esančios tame bloke. Taip pat ši atmintis yra dažniausiai naudojama komunikacijai tarp gijų, nes ji yra prieinama labai greitai (apie 100 kartų greičiau negu standartine DRAM).
- Tekstūrinė Atmintis (angl. *Texture Memory*) pagreitina lokalų skaitymą. Ši atmintis yra *read-only* tipo ir yra užpildoma šeimininko (host) terpėje, vėliau ją galima nuskaityti iš plokštės terpės. Atliekant skaitymo operaciją iš x tekstūros atminties celės, yra užkešuojamas [x - k, x + k] (vienmetės tekstūros atveju) regionas ir sekančios skaitymo operacijos, patenkančios į tą regioną, yra atliekamos daug greičiau.

CUDA tai pat palaiko apjungtą atminties prieigą (angl. *coalesced memory access*) gijoms, esančioms toje pačioje grupėje. Ši optimizacija dviejų arba daugiau gijų globalios atminties skaitymo užklausas apjungia į vieną (t. y. į kiek įmanoma mažiau užklausų). Įvairių autorių algoritmų implementacijose matysime, kad buvo įdėta pakankamai daug pastangų sudėlioti vykdymo seką taip, kad gijos atmintį skaitytų vienu metu.

1.2.2.4. Gijų grupavimas

Bloko viduje gijos yra suskirstomos į grupes (angl. *thread warp*) po 32. Gijos grupėje yra vykdomos SIMD principu – t. y. vienu metu, jeigu neįvyksta gijų divergencija. Gijos grupėje diverguoja, jeigu jos nustoja vykdyti tą pačią instrukciją(kiekvienas kodo išsišakojimas (*if*, *while*, *for*, ir t.t.) gali „Išskirti“ gijas). Divergencijos atveju skirtingos kodo šakos yra vykdomos nuosekliai žr. 1.6. Taisyklingas gijų grupių naudojimas daro stiprią įtaką projektuojamo algoritmo spartai.

```
__global__ ThrDiv( int * a, int N ) {
  int tid = threadIdx.x; // Kinta [0..31]
  if ( tid < N ) {
    // Sudetinga operacija trunkanti 1 sekunde
  } else {
    // Skirtinga sudetinga operacija trunkanti 2 sekundes.
  }
}
int main ( int argc, char * argv[] ) {
  ...
  // Nediverguoja, trunka 1 sekunde
  ThrDiv<<<1,32>>>( vec1, 32 );
  // Nediverguoja trunka 2 sekundes
  ThrDiv<<<1,32>>>( vec1, 0 );
  // Diverguoja, trunka 3 sekundes
  ThrDiv<<<1,32>>>( vec1, 4 );
  ...
}
```

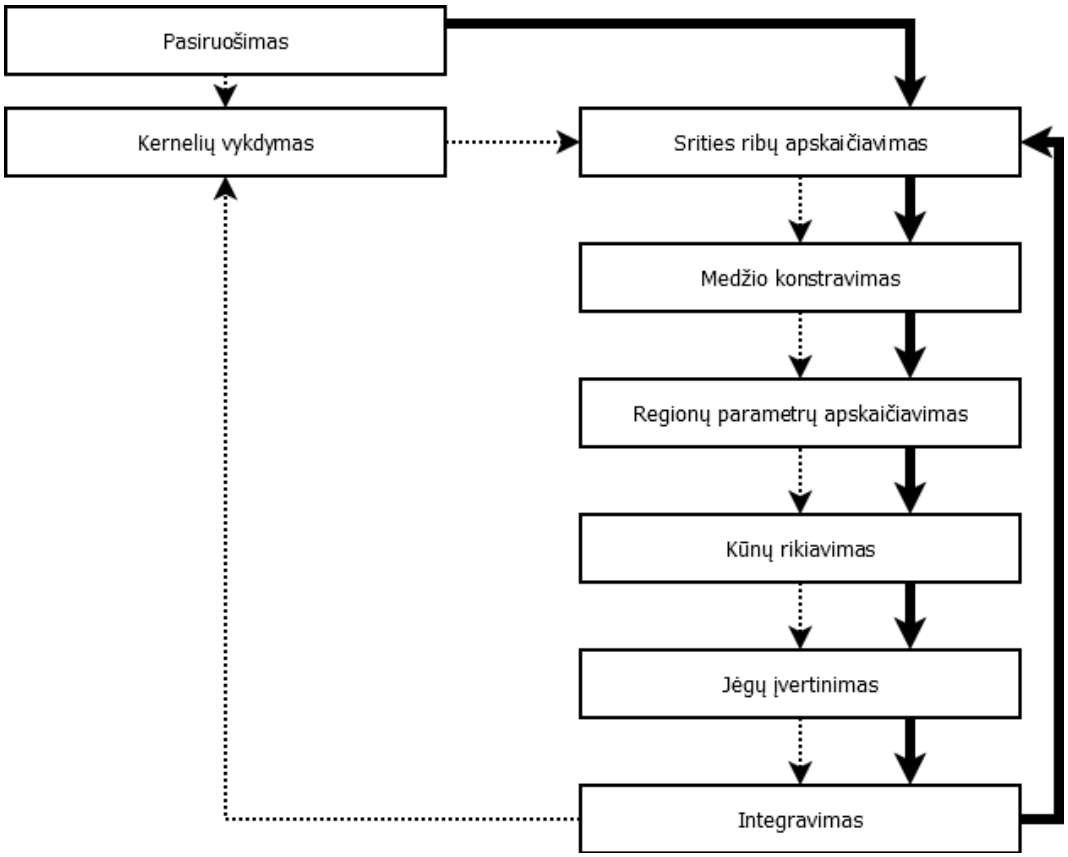
1.6 pav. CUDA gijų divergencijos pavyzdys

1.2.3. Esami problemos sprendimai CUDA terpėje

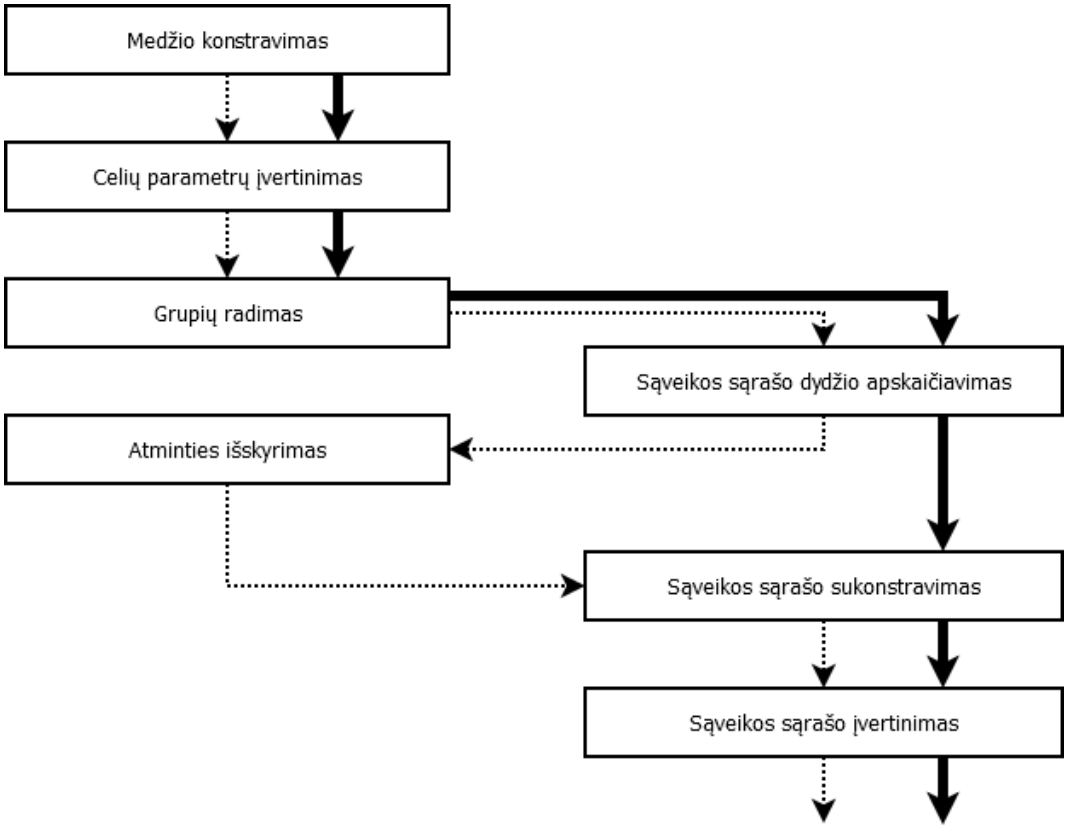
Egzistuoja gerai suformuota N-kūnų problemos sprendimo CUDA terpėje bazė [3, 5, 6, 11]. Kiekviename, toliau einančiame skyrelyje, bus patyrinėjama kaip kiekviena implementacija realizuoja tam tikrą algoritmo aspektą. Visi algoritmai yra savotiški BH algoritmo variantai, modifikuoti autorių tam kad veiktų sparčiau CPU/GPU terpėje.

1.2.3.1. Veiksmų seka

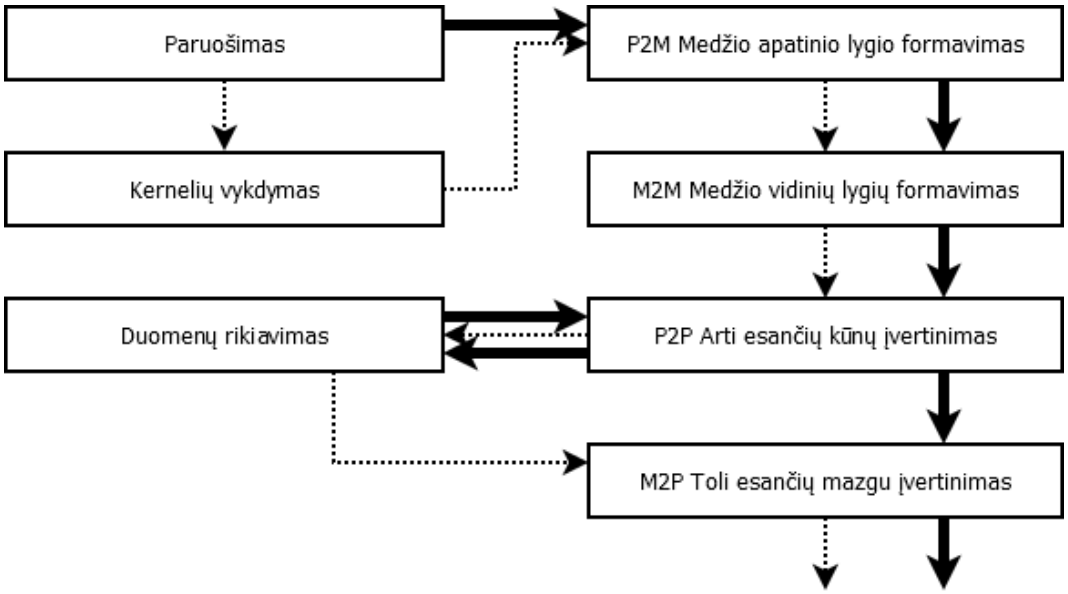
Tyrinėjamos realizacijos turi papildoma uždavinį: surasti balansą tarp veiksmų, atliekamų CPU ir CUDA terpėse (priklausomai nuo įvairių darinių, atskiros algoritmo dalys galį būti vykdomos greičiau CPU terpėje negu GPU), tam kad visumoje vykdymo laikas būtų trumpiausias. Algoritmų veiksmų sekos analizė padės suprasti kodėl ir kada implementacijų autoriai nusprendžia deleguoti darbą į GPU.



1.7) *Burtscher-Pingali* veiksmų seka



1.8) *Gaburov-Bedorf-Zwart* veiksmų seka



1.9) *Yokota-Barba* veiksmų seka

Paveiksluose yra pavaizduotos analizuotos realizacijos: 1.7 iš [5]; 1.8 iš [6]; 1.9 iš [11]. Punktyrinės linijos atvaizduoja algoritmo seką, vientisos linijos atvaizduoja duomenų srautą. Kairėje pusėje esantys veiksmai yra atliekami CPU terpėje, dešinėje — GPU.

Pirmasis BH algoritmo veiksmas, kurį matome paveiksle 1.7, yra pradinio apribojimo (šakninio mazgo kūbo briaunų) suradimas. Šis veiksmas yra išskiriamas tiktai *Burtscher-Pingali* rea-

lizacijoje, ir joje jis yra vykdomas GPU terpėje [5]. Duomenys (kūnai) yra išskaidomi į lygias dalis ir priskiriami blokams, kur kiekvienas iš jų nusiskaito kūnus vieną kartą, pilnai apjungtu būdu, ir redukuoja kūnų pozicijas, pasinaudodamas aparatūrinėje dalyje realizuotomis *min* ir *max* funkcijomis. Galiausiai yra išrenkamas blokas, kuris apjungia kitų blokų redukcijas ir sukuria šakninį mazgą. Kitos realizacijos tai atlieka arba CPU terpėje arba per daug apie tai neišsiplečia.

Sekantis veiksmas yra medžio konstravimas ir *Burtscher-Pingali* bei *Yokota-Barba* tai atlieka tai GPU terpėje [5, 11], o *Gaburov-Bedorf-Zwart* — CPU [6]. *Burtscher-Pingali* realizacijoje kiekvienas kūnas yra priskiriamas gijai (gijų kiekis yra fiksuotas ir gija užbaigusi darbą su vienu kūnų, gauna sekantį), kuri bando tą kūną įterpti į medžio struktūrą [5]. Iš pradžių, gija suranda vietą, kurioje reikia įrašyti kūną (medis saugomas masyve), ją pabando užrakinti ir jeigu tai pavyksta — atlieka įrašymą. Rakinimas yra reikalingas, nes: įsivaizduokime, kad yra dvi gijos vienoje grupėje. Jos vykdo kodą vienu metu ir bando įterpti kūnus, kurių pozicijos yra labai arti (ta pati vieta masyve), tik viena iš gijų gali atlikti įterpimą į tą vietą ir tai yra užtikrinama su rakinimu. Kita gija, radusi užrakintą vietą nutrauks terpimo ciklą ir bandys jį vėliau — dabar jai reikės perskelti tą mazgą, į kurią pirmoji giją įterpė kūną, ir bandyti terpti savo kūną į viena iš pomazgių. Reikia atkreipti dėmesį į tai, kad antroji gija radusi užrakintą vietą, dėl SIMD principo, yra sustabdoma, kol procesorius įvykdo pirmosios gijos divergavusią kodo šaką. Kai pirmoji gija užbaigia įterpimą ir pasiima naują kūną, abi gijos vėl susijungia. Tai yra labai paranku, nes jeigu antroji gija būtų neblokuojama, jinai užtvindytų GPU atmintį užklausomis apie tą vietą, kurioje jinai nori atlikti įterpimą.

Yokota-Barba realizacijoje medžio konstravimas yra atliekamas dvilypiu veiksmu [11]:

1. P2M — Particle To Multipole.
2. M2M — Multipole To Multipole.

P2M veiksmas suskaido erdvę į $4^n; n \in \mathbb{N}^+$ regionus[11]. Kiekvienas regionas yra traktuojamas kaip medžio apatinis mazgas (lapas) ir kiekvienam iš jų, tuo pačiu, yra paskaičiuojami masės centrai ir masės. M2M veiksmas suskirsto erdvę į 4^{n-1} regionus — antras nuo apačios medžio lygis. Kiekvienam šio lygio mazgui irgi yra paskaičiuojamos masės ir jų centrai. M2M yra kartojamas kol yra pasiekiamas reikiamas suskaidymo lygis (tačiau $n = 0$ yra nepasiekiamas), pradinis n ir skaidymų kiekis yra parenkamas pagal modelio apimtį N. Toks „medis“ labiau primena tinklelį negu standartinį nebalansuota BH medį. Tokia struktūra yra pasirenkama, nes realizacija yra apibūdinama kartu su FMM (Fast Multipole) algoritmu (FMM tokia struktūra yra parankesnė).

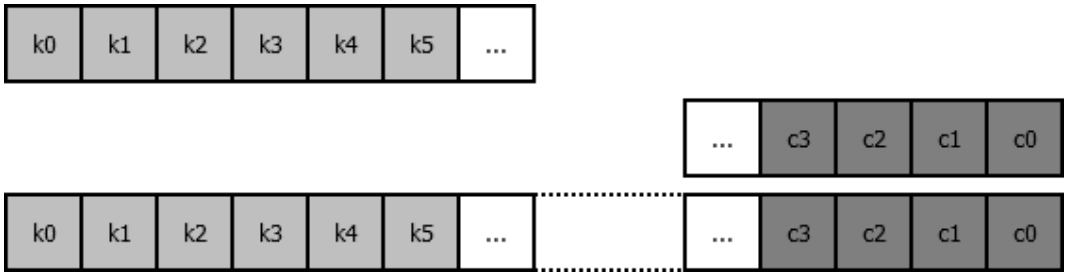
P2M ir M2M taip pat atlieka medžio mazgų/celių parametrų (bendra masė, masės centras) įvertinimą. *Gaburov-Bedorf-Zwart* tai atlieka CPU terpėje [6], o *Burtscher-Pingali* — GPU [5]. *Burtscher-Pingali* realizacijoje medžio apėjimas yra atliekamas nuo apačios [5]. Kiekvienai gijai yra priskiriama tam tikra mazgų imtis, kurią ji apdoroja iš eilės, pradedant nuo mažiausio indekso. Kadangi pagrindiniame masyve pomazgių indeksai yra mažesni negu jų mazgų, nesusidarys situacija, kurioje gija bando apdoroti mazgą, su neapdorotu pomazgiu — yra išvengiama užsikirtimų (angl. *deadblock*). Šis kernelis pasinaudoja vykdomu medžio apėjimu ir atlieka papildomas opera-

cijas: suskaičiuoja kūnų kiekį mazge ir sustumia kiekvieno mazgo pomazgių indeksus į priekį. Tai paspartina sekančių kernelio darbą.

Burtscher-Pingali realizacijoje yra atliekamas rikiavimas, nes jos jėgų apėjimo veiksmas imituoja modifikuotos BH versijos grupavimą [5]. Teoriniame aprašyme kvadratiniame regione, kuriame yra mažiau negu n_{crit} kūnų, visi kūnai turi bendra sąveikos sąrašą, o realizacijoje sąveikos sąrašu dalinasi 32 medžio masyve iš eilės einantys kūnai. Dėl to yra svarbu surikiuoti kūnus taip, kad jų artumas modelio erdvėje (trimatė) atsispindėtų masyvo erdvėje (vienmatė).

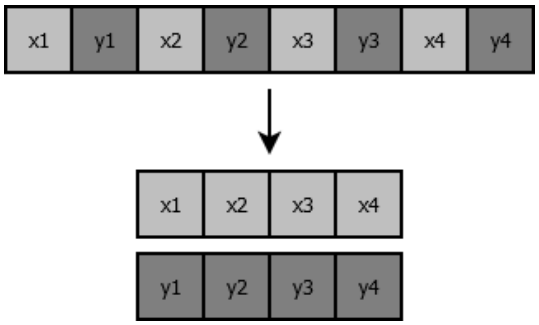
1.2.3.2. Duomenų struktūros

Didžiausia problema, kuria reikia spręsti realizuojant BH algoritmą grafinėje terpėje, yra medžio struktūros manipuliavimas ir saugojimas. Kaip jau minėta, rodyklių sekimas grafinėje terpėje yra labai neefektyvi operacija. Medžio pavidalo struktūros mazgai CPU terpėje būna paskirstomi po atminties aibę (angl. *memory heap*) ir jos skaitymas rašymas reikalauja daug rodyklių sekiojimo. Grafinėje terpėje, kad apeiti šia problemą, yra naudojamas masyvas arba masyvai, kuriuose yra saugomi mazgų duomenys, o rodyklės į atmintį yra pakeičiamos indeksais. Toliau yra išvardinti keli sutikti medžio struktūros saugojimo ypatumai.



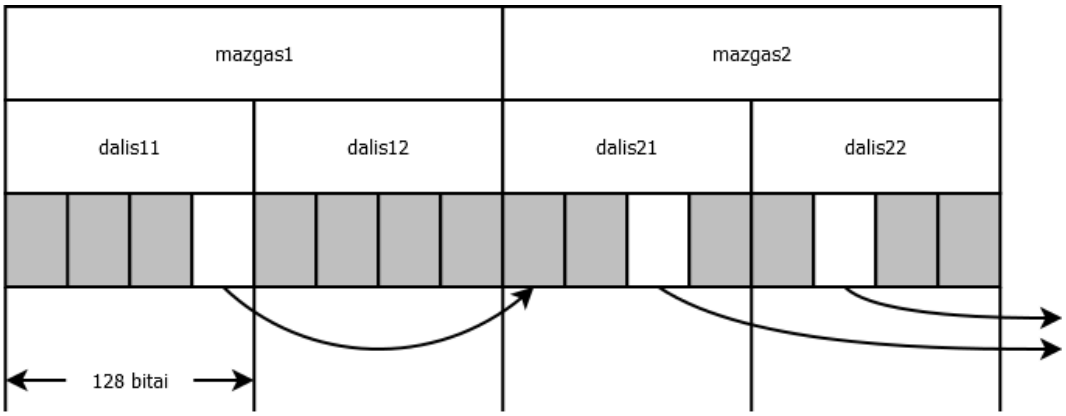
1.10 pav. Kūnų ir celių elementų laikymas viename masyve

Paveiksle 1.10 yra pavaizduota kaip galima apjungti kelis masyvus į vieną, jeigu jų saugojama informacija yra panaši. Kairėje pusėje yra saugomi kūnai, o dešinėje mazgai („c“ — cell/celė). Tokiame išdėstyme nereikia kurti apatinių mazgų (kurie turi tiktai vieną kūna) — užtenka išsaugoti indeksą į kūną. Kadangi kūnai yra priekyje, nereikia turėti papildomos informacijos, nusakančios ar pomazgis yra kūnas ar ne, užtenka indeksą palyginti su esamų kūnų kiekiu: jeigu indeksas mažesnis, pomazgis yra kūnas.



1.11 pav. Struktūrų masyvo išskaidymas į kelis skaliarinius masyvus

Paveiksle 1.11 yra pavaizduojama kaip išskirti struktūrų masyvą į kelis skaliarinius masyvus. Vieną struktūros lauką, išklotą viename masyve, galima skaityti nuosekliai ir išnaudoti atminties prieigos apjungimą. Apjungimas yra neįmanomas dirbant su struktūrų masyvais.



1.12 pav. Atminties struktūros sulyginimas ties 128 bitais

Grafinėse plokštėse su viena instrukcija yra įmanoma užkrauti 128 bitų žodį tiesiai į registrą. Paveiksle 1.12 yra parodoma kaip galima saugoti medžio struktūra pasinaudojant 128 bitų elementais. Kiekvienas mazgas yra atvaizduojamas dviejų 128 bitų žodžiais: pirmasis žodis saugo indeksus į pirmus keturi pomazgius, o antrasis į likusius keturis. Indekso informacija yra išgaunama bitinėmis (angl. *bitwise*) operacijomis.

1.2.3.3. Realizacijų apibendrinimas

Kiekvienos realizacijos autoriai pabrėžia, kad algoritmuose nebuvo pritaikytos visos galimos optimizacijos. Dėl to spartos eksperimento rezultatai nevisiškai tiksliai atspindi algoritmo potencialą. Taisyklingiausia būtų lyginti individualius komponentus, tačiau jie yra per stipriai pririšti prie kitų realizacijos komponentų. Iš realizacijų buvo išimtas integravimo komponentas, nes tik jis gali būti idealiai atskiriamas.

1.3. TYRIMO TIKSLAS IR UŽDAVINIAI

Tyrimo tikslas yra pademonstruoti, kaip įvairūs problemos sprendimo būdai (algoritmai) atrodo tarpusavyje sugretinti.

Analizės uždaviniai Norint atlikti tyrimą reikia:

- Išanalizuoti teorinę algoritmų esmę. Realizuojant algoritmą CUDA terpėje reikia atlikti pakankamai daug modifikacijų, kurios skiriasi su kiekviena realizacija. Norint suprasti kodėl realizacijų autoriai priima tam tikrus projektavimo sprendimus reikia žinoti, su kokiu algoritmo teoriniu aprašu jie dirba.
- Išanalizuoti algoritmų realizacijas. Aprašyti panaudotas optimizacijas, algoritmų sekas, duomenų struktūras.
- Nustatyti kokybės kriterijus, pagal kuriuos bus lyginamos algoritmų realizacijos.
- Pateikti preliminarų eksperimento vykdymo planą.

1.4. SIEKIAMO SPRENDIMO APIBRĖŽIMAS

Darbe yra siekiama aprašyti kaip skirtingos BH algoritmo CUDA realizacijos veikia, esant skirtingoms sąlygoms.

1.5. ANALIZĖS IŠVADOS

Atlikus probleminės srities analizę galima pastebėti:

- Egzistuoja pakankamai daug būdų, skirtų spręsti N-kūnų problemą. Jie naudoja įvairias duomenų struktūras, turi įvairius sudėtingumus (nuo $O(n^2)$ iki $O(n)$) ir įvairiai išnaudoja gravitacinio lauko savybes.
- Realizacijų analizė parodė:
  CUDA terpėje yra įmanoma pakankamai našiai manipuliuoti medžio pavidalo struktūrą.
  Yra daug būdų kaip tą struktūrą galima realizuoti.
  Kiekviena realizacija turi unikalias optimizavimo kryptis.
- Be standartinio algoritmo kokybės parametro — **spartos**, CUDA terpėje reikia atkreipti dėmesį į **atminties** sąnaudas. Taip pat, aproksimacijos algoritmuose yra svarbus rezultatų **tikslumas**.

2. SPRENDIMO REIKALAVIMŲ SPECIFIKACIJA IR PROJEKTAS

2.1. REIKALAVIMŲ SPECIFIKACIJA

2.1.1. Kokybės kriterijai

Aproksimacijos algoritmai atsirado dėl spartos poreikio. Dėl to sparta yra svarbus kokybiškos algoritmo realizacijos kriterijus. Skyriuje 1.2.3.3. yra pabrėžiama, kad darbe tiriamų algoritmų realizacijos nėra pilnai optimizuotos, tačiau vistiek bus įvykdytas pilno vykdymo trukmės matavimas, nes komponentų individualus sulyginimas yra problematiškas.

Algoritmo atminties sąnaudos tiesiogiai apriboja modelio dalelių kiekį N. Operatyviosios atminties srityje grafinė terpė yra ne tokia lanksti kaip bendroji terpė:

- Neturi puslapiavimo (angl. *memory paging*). Dėl to neįmanoma apskaičiuoti modelio, turinčio per didelį N, jo neišskaidžius (skaidymas sumažina spartą).
- Atmintis yra nemoduliška. Norint jos turėti daugiau — reikia pirkti naują grafinę plokštę.

Atminties sąnaudos yra svarbus kokybės kriterijus, nes algoritmo pritaikomumas sumažės, jeigu jis [algoritmas] bus vykdomas tiktai su santykinai mažu N.

Kadangi didžioji dalis algoritmų, tyrinėjamų šiame darbe, labai aproksimacijos algoritmai, yra svarbu nustatyti bei palyginti su kokia paklaida jie apskaičiuoja rezultatą. Jeigu paklaida yra per didelė, modeliavimo erdvė sparčiai mutuoja – įgyja arba nutekina kinetinę energiją, sūkio momentą arba kitaip pažeidžia energijos tvermės dėsnį. Lyginant algoritmus, yra svarbu atkreipti dėmesį į grąžinamo rezultatų paklaidą, nes rezultato kokybė daro tiesioginę įtaką pačio algoritmo kokybei.

2.1 lentelė. Kokybės kriterijai

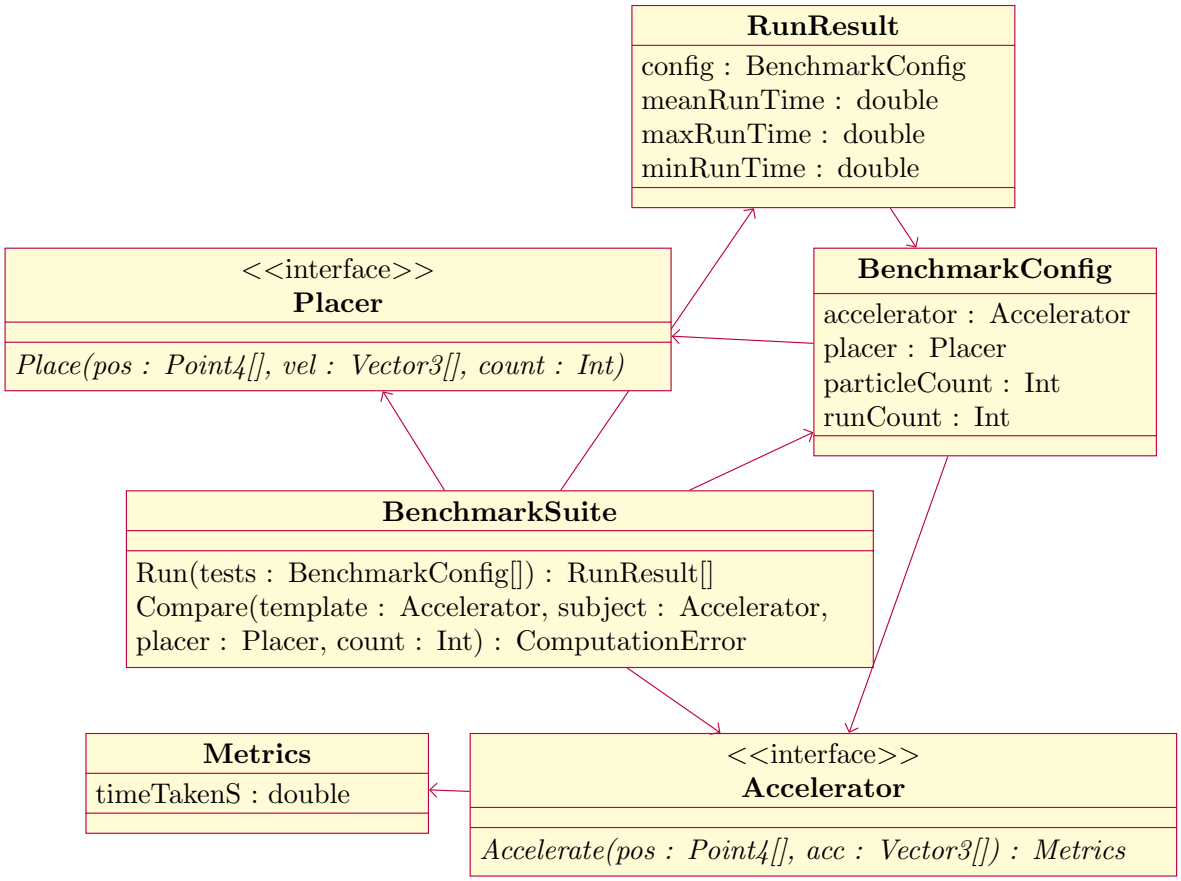
Kriterijus	Matas	Reikšmė
Komponentų sparta	sekundės	Kiek ilgai vyks modeliavimas.
Reikalinga atmintis	baitai	Kokios N apimties modelį yra įmanoma vykdyti.
Tikslumas	koeficientas	Kiek modelio rezultatas skriasi nuo realybės.

3. SPRENDIMO REALIZACIJA IR TESTAVIMAS

3.1. SPRENDIMO REALIZACIJOS IR VEIKIMO APRAŠAS

Darbe buvo sukurtas mažos apimties eksperimentavimo karkasas, su kuriuo galima greitai atlikti pageidaujamos konfigūracijos eksperimentus. Karkasas buvo sukurtas taip, kad būtų galima greitai pakeisti kiekviena iš galimų kintamųjų:

a) Realizacija, kuri atlieka pagreičio vektorių apskaičiavimą
b) pradiniai duomenys
c) dalelių kiekis n
d) jeigu realizacija yra BH algoritmo — atidarymo kampas θ
e) kiek kartų eksperimentas bus kartojamas



3.1 pav. Eksperimentavimo karkaso klasių diagrama

Diagramoje 3.1 yra pavaizduojama apibendrinta eksperimentavimo karkaso struktūra. `BenchmarkSuite` klasės `Run` funkcija yra naudojama atlikti tam tikrą kiekį spartos matavimo eksperimentų. Į šią funkciją yra paduodamas `BenchmarkConfig` objektų masyvas, kur kiekvienas elementas saugo informaciją apie vieną spartos matavimo eksperimentą. Taip pat diagramoje yra

vaizduojamos dvi sąsajos (angl. *interface*): `Placer` ir `Accelerator`. Visi, darbe tiriami algoritmai, yra pasiekiami per `Accelerator` interfeisą, o `Placer` interfeisas yra naudojamas sugeneruoti dalelių erdvę.

Visų Barnso-Huto algoritmo realizacijų, kurios yra tiriamos šiame darbe, išeities kodai yra prieinami internete. Šios programos buvo parsisiųstos ir sukeltos į atskirus Visual Studio projektus. Kiekvienas realizacijos projektas yra sukompiliuojamas į atskirą dinaminę biblioteką (angl. *dyanmic link library*), kuri turi paprastą C stiliaus sąsają. Tuomet kiekviena biblioteka yra susiejama su pagrindiniu projektu, kuriame yra vykdomas eksperimentas.

```
#ifndef _BURTSCHER_PINGALI_API_H_
#define _BURTSCHER_PINGALI_API_H_
#ifdef BPAPI_EXPORTS
#define BPAPI_EXPORTS __declspec(dllexport)
#else
#define BPAPI_EXPORTS __declspec(dllimport)
#endif
#ifdef __cplusplus
extern "C" {
#endif
        void BPAPI_EXPORTS Initialise( float * _mass,
                float * _position_x, float * _position_y, float * _position_z,
                float * _velocity_x, float * _velocity_y, float * _velocity_z,
                unsigned int _particle_count, int _device );
        void BPAPI_EXPORTS Sample(
                float * _position_x, float * _position_y, float * _position_z,
                float * _velocity_x, float * _velocity_y, float * _velocity_z,
                unsigned int _particle_count );
        void BPAPI_EXPORTS Integrate( float _dtime, float _eps_squared );
        void BPAPI_EXPORTS Accelerate(
                float * ax, float * ay, float * az,
                float _eps_squared,
                double _opening_angle,
                double * time_taken,
                unsigned int * bytes_consumed );
        void BPAPI_EXPORTS TearDown( );
#ifdef __cplusplus
}
#endif
#endif
```

3.2 pav. Burtscher-Pingali realizacijos programinė sąsaja

Programinio kodo išraše 3.2 yra pateikiama Burtscher-Pingali realizacijos programinė sąsaja. Kitų realizacijų programinės sąsajos yra labai panašios.

Visą parsiųstą programinį kodą reikėjo minimaliai modifikuoti, kad jis veiktų (kompiliuotųsi ir grąžintų teisingą rezultatą) esamoje aplinkoje:

- Realizacijos naudojo `<time.h>` C antraštę, kurį yra nepalaikoma Visual Studio 2013 versijoje, dėl to ją reikėjo pakeisti.
- GBZ realizacija buvo parašyta, kad veiktų su g++ kompiliatoriumi, kuris skiriasi nuo Microsoft kompanijos c/c++ kompiliatoriaus `cl.exe`. g++ kompiliatorius supranta kai yra norima inicijuoti struktūrą pasinaudojant masyvu: `Point4Double a = {0.0, 0.0, 0.0, 0.0}`, tačiau cl kompiliatorius tokią formuluotę atmeta. Kad apeiti problemą, buvo parašyta dedikuota funkcija, kuri inicijuoja ir gražina struktūros objektą, kuris yra sukonstruojamas iš funkcijai paduodamų argumentų.
- Standartinėje c++ bibliotekoje yra `std::vector<T>` kolekcija, kurios indeksavimas g++ kompiliatoriuje yra „atlaidesnis“. Jeigu norima priskirti i-tąjį elementą, kai kolekcijos dydis yra mažesnis arba lygus i, su g++ kompiliatoriumi programa tai padarys be klaidų (automatiškai praplės kolekciją), tačiau su cl sukompiliuota programa rodys klaidą. Šiuo atveju teko atsisakyti tiesioginio indeksavimo — jis buvo pakeistas kolekcijai priklausančia `push_back(...)` funkcija.

3.2. TESTAVIMO MODELIS, DUOMENYS, REZULTATAI

Eksperimentavimo karkasas buvo testuojamas testais, kurie yra panašūs į vienetų testus:

```
void rng_test( );
void coordinate_tests( );
void placer_tests( );
void integrator_tests( );
void opengl_tests( );
void simulation_test( );
void bh_test( );
void bp_api_test( );
void vector_test( );
void varargs_test( );
void grav_file_stream_test( );
void direct_test( );
void map_test( );
void chrono_test( );
void benchmark_test( );
void progress_bar_test( );
void benchmark_suite_tests( );
void wholemeal_tests( );
void gaburov_10_api_test( );
void yb_api_test( );
void append_time_test( );
void clipboard_test( );
```

3.3 pav. Testavimo funkcijų antraštė

Programinio kodo išraše 3.3 yra pavaizduojama testavimo funkcijų antraštė (angl. *Header*). Iš viso yra 22 testai, skirti izoliuoti specifines eksperimentavimo karkaso dalis. Šie testai yra panašūs į vienetų testus (angl. *Unit Test*), tačiau skiriasi tuo kad juose nėra teiginių (angl. *assertion*) ir dėl to jie nėra automatizuoti. Šie testai buvo parašyti tuo pačiu metu, kai buvo rašomos karkaso klasės, kad jas būtų galima greitai ir patogiai derinti (angl. *debug*).

Pačių realizacijų korektiškumas buvo užtikrinamas atliekant tikslumo matavimo eksperimentą (žr. skyrių 4.3.). Jeigu realizacija paskaičiuoja pagreičio vektorius, kurių santykinė paklaida, lyginant su tiesioginiu algoritmu, yra maža, tuomet yra daroma prielaida kad realizacija yra paruošta eksperimentui. Tikslumo eksperimentas buvo atliekamas su homogeninio paskirstymo duomenimis, nes jie, turėtų sukurti mažiausią paklaidą.

Buvo atvejų kai reikėjo derinti patį CUDA išeities kodą. Norint tai atlikti efektyviai, buvo naudojamas specialus „vieneteinis“ duomenų rinkinys, kuris susideda iš 9 kūnų, kurių pozicijos yra žinomos. Tokia supaprastinta erdvė, turi apibrėžtą apimtį, medžio formą ir pagreičio vektorius. Visa ši informacija buvo naudojama tikrinant ar individualūs realizacijos etapai apskaičiuoja

teisingą rezultatą. Pavyzdžiui, buvo tikrinama ar po rikiavimo etapo dalelės yra surikiuojamos teisingai.

4. EKSPERIMENTINIS TYRIMAS

Darbe buvo eksperimentiškai lyginamos 5 realizacijos: tiesioginė CPU realizacija, tiesioginė lygiagreti CUDA realizacija, Burtscher-Pingali realizacija, Gaburov-Bedorf-Zwart realizacija ir Yokota-Barba realizacija.

4.1 lentelė. Eksperimente dalyvaujančios algoritmų realizacijos

Pavadinimas	Tipas	Terpė	Trumpinys	Spalva	Ženklas
Tiesioginis CPU	Tiesioginis	CPU	HN (angl. **H**ost **N**aive)	Melyna	*
Tiesioginis CUDA	Tiesioginis	GPU	CN (angl. **C**uda **N**aive)	Oranžinė	□
Burtscher-Pingali	Medžio	GPU	BP	Žalia	△
Gaburov-Bedorf-Zwart	Medžio	GPU	GBZ	Geltona	◯
Yokota-Barba	Medžio	GPU	YB	Raudona	◇

4.1 Lentelėje yra pateikiamos darbe analizuojamos algoritmų realizacijos, bei jų unikalus žymėjimas. Tolesniuose skyriuose bus pateikiami eksperimento rezultatai, išreikšti grafikais, kuriuose kiekviena realizacija turės unikalų žymėjimą.

4.1. EKSPERIMENTO PLANAS

4.1.1. Palyginimas

1. Techninės įrangos apibrėžimas.
2. Pradinių duomenų apibrėžimas.
3. Kiekvieno modelio vykdymas ir rezultatų dokumentavimas.

Techninės įrangos apibrėžimas Eksperimentas buvo įvykdytas su Intel Core i5 3570K procesoriumi, kuris dirba 3.8 GHz taktiniu dažniu, bei NVIDIA GeForce GTX 650 Ti grafine plokšte. Grafinės plokštės procesoriaus taktinis dažnis yra 980 MHz, atminties pralaidumas yra 86.4 GB/s, ir ji turi 768 CUDA branduolius.

Pradinių duomenų apibrėžimas BH algoritmas dirba su medžio struktūra. Medžio manipuliavimo sparta gali (ir dažniausiai taip yra) priklausyti nuo to kiek medis yra išbalansuotas. Medis būna idealiai subalansuotas jeigu modeliuojama erdvė yra homogeninė (visos dalelės yra tolygiai pasiskirsčiusios, t. y. — daugelio galaktikų arba dalelių ūko modelis). Kitais atvejais, medis būna išbalansuotas (žvaigždžių pasiskirstymo galaktikoje modelis). Darbe eksperimentas buvo

įvykdytas su trimis dalelių paskirstymo tipais: homogeniniu pasiskirstymu, 1 galaktikos modeliu, 2 susiduriančių galaktikų modeliu. Taip pat atkreiptas dėmesys į tai, kad modelio apdorojimo trukmė priklauso nuo dalelių kiekio, dėl to eksperimentas buvo kartojamas su skirtingais dalelių kiekiais n.

Modelių vykdymas ir rezultatų dokumentavimas Spartos matavimo eksperimentas buvo vykdomas matuojant laiko tarpą tarp algoritmo vykdymo pradžios iki to taško, kuriame yra išgaunami kiekvienos dalelės pagreičio vektoriai a. Į laiko matavimą nėra įtraukiamas integravimas, nes jis yra lengvai atskiriamas ir pakeičiamas komponentas — galima rinktis nuo paprasto (Leapfrog) iki sudėtingesnio (Rungės-Kuto). Esant specifiniam dalelių išsidėstymo tipui ir specifiniam dalelių kiekiui n, duomenys yra identiški. Pavyzdžiui, eksperimentuose

- BP spartos matavimas, kai $\theta = 0.4$, erdvė yra homogeninė ir $n = 32000$
- GBZ tikslumo matavimas, kai $\theta = 0.7$, erdvė yra homogeninė ir $n = 32000$

dalelių pozicijos erdvėje buvo identiškos.

Tikslumo eksperimentui buvo matuojama kaip stipriai skiriasi kiekvienos dalelės pagreitis a, lyginant su etaloniniu pagreičiu a_{CN}. Etaloninis pagreitis a_{CN} yra pagreitis, kuris buvo apskaičiuotas naivaus CUDA algoritmo. Yra daroma prielaida, kad HN ir CN algoritmai sugeba idealiai apskaičiuoti kiekvienos dalelės jaučiamą pagreitį modelio erdvėje. CN algoritmas buvo pasirinktas kaip etalonas, nes jis veikia greičiau už HN.

$$\frac{\Delta a}{a} = \frac{\vec{a} - \vec{a}_{CN}}{\vec{a}_{CN}} \tag{4.1}$$

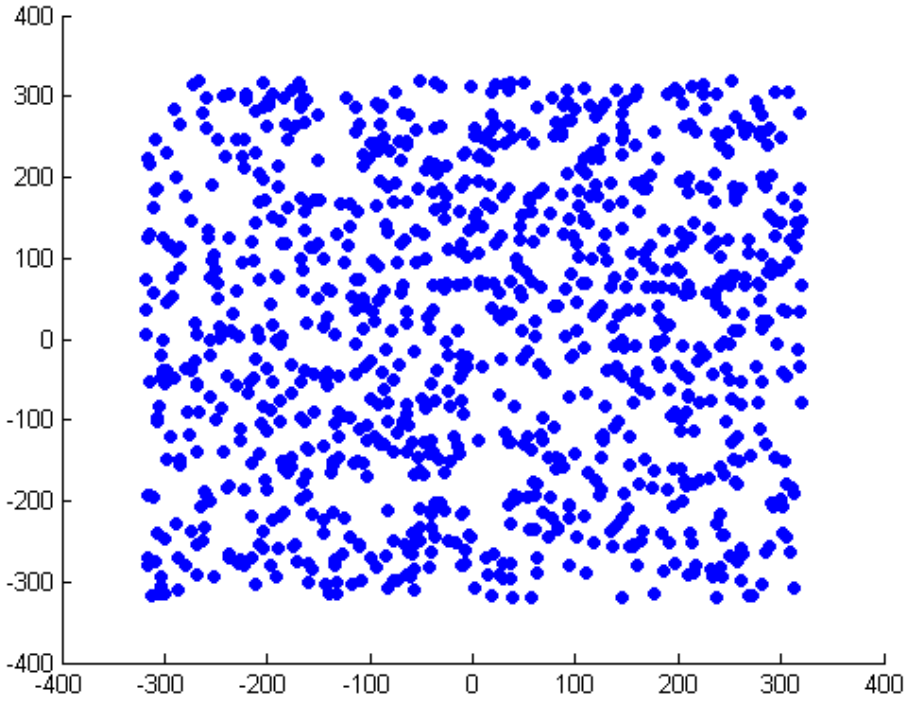
Lygtis 4.1 parodo kaip yra apskaičiuojama vienos dalelės paklaida $\frac{\Delta a}{a}$. Šios paklaidos pasiskirstymas yra vaizduojamas tikslumo grafikuose (pvz., 4.5 paveikslas). $F\left(> \frac{\Delta a}{a}\right)$ funkcija, kuri tikslumo grafike atstoja Oy ašį, parodo kiek dalelių modelyje turi didesnę paklaidą, negu $\frac{\Delta a}{a}$. Pvz.: Jeigu $\frac{\Delta a}{a} = 0.01$ ir $F\left(> \frac{\Delta a}{a}\right) = F\left(> 0.01\right) = 0.30$, reiškia 0.30 modelio dalis (t. y. 30% visų dalelių) turi didesnę paklaidą negu 0.01 (t. y. algoritmo apskaičiuotas pagreičio vektorius $\vec{a}$ skiriasi bent 1%).

Atminties sąnaudų matavimas nebuvo atliktas, nes buvo nuspręsta giliau patyrinėti spartos bei tikslumo priklausomybę nuo atidarymo kampo θ.

4.2. PRADINIAI DUOMENYS

Yra izoliuojami 3 pagrindiniai dalelių pasiskirstymai su kuriais bus atliekamas eksperimentas.

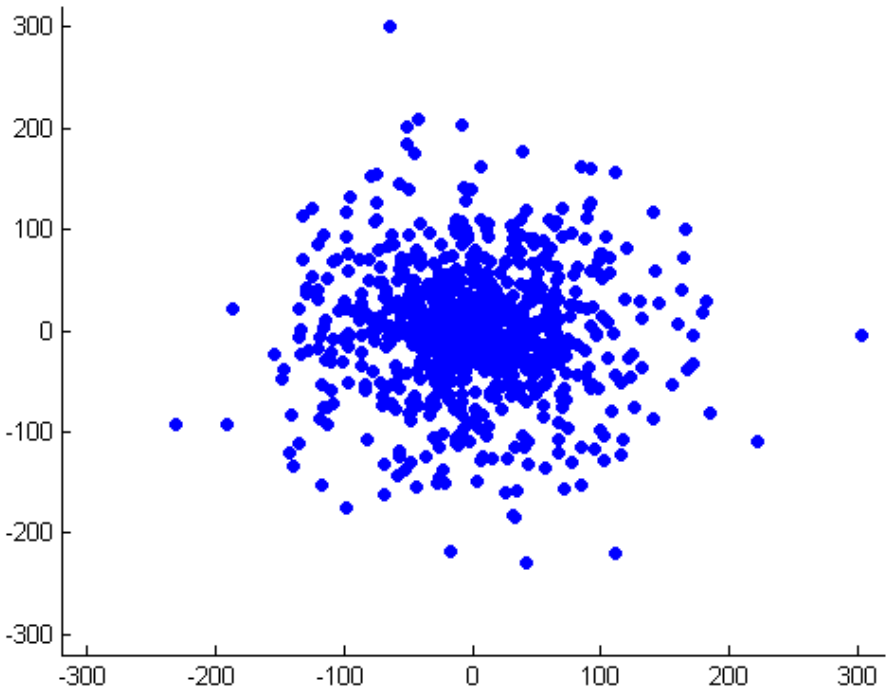
Homogeninis Dalelės yra tolydžiai išsidėsčiusios po tam tikros apimties erdvę.



4.1 pav. homogeninis dalelių pasiskirstymas

4.1 paveiksle yra vaizduojamas pavyzdinis 1000 dalelių išsidėstymas dvimatėje erdvėje. Tokio tipo modeliuose, viena dalelė dažniausiai vaizduoja vieną galaktiką, o dalelių visuma vaizduoja galaktikų superklasterį (angl. *supercluster*) — taip yra tyrinėjamas superklasterio formavimasis. Paveiksle Ox ir Oy ašys vaizduoja neapibrėžtus atstumo vienetus, jie yra sukonkretinami pasirenkant modelio tipą.

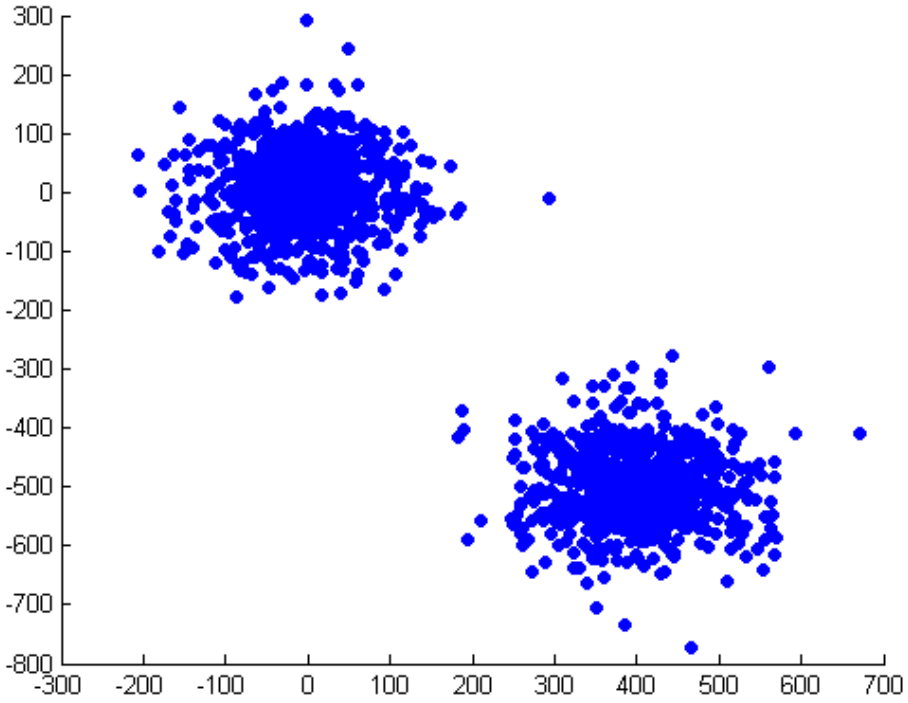
Galaktikos Dalelės yra išdėstomos taip kad sudarytų galaktiką.



4.2 pav. Galaktikos formos dalelių pasiskirstymas

4.2 paveiksle yra vaizduojama pavyzdinė galaktika. Šiuo metu yra sugeneruotas paprastas diskas, kur kiekviena dalelė turi atsitiktinį kampą ir normalini-atsiktinį atstumą nuo disko centro (t. y. arčiau centro yra didesnė tikimybė rasti dalelę). Tokio tipo modeliuose viena dalelė atvaizduoja vieną žvaigždę, ir jos visos turi pradinį judėjimo vektorių. Tokio tipo modelyje yra tyrinėjama galaktikos raida, t. y. stebima kokios formuojasi superstruktūros (angl. *superstructure*), ar jos stabilios, ir ar modelis atitinka realius stebėjimus.

Galaktikų susidūrimas Dalelės yra išdėstomos taip kad būtų imituojamas galaktikų susidūrimas.



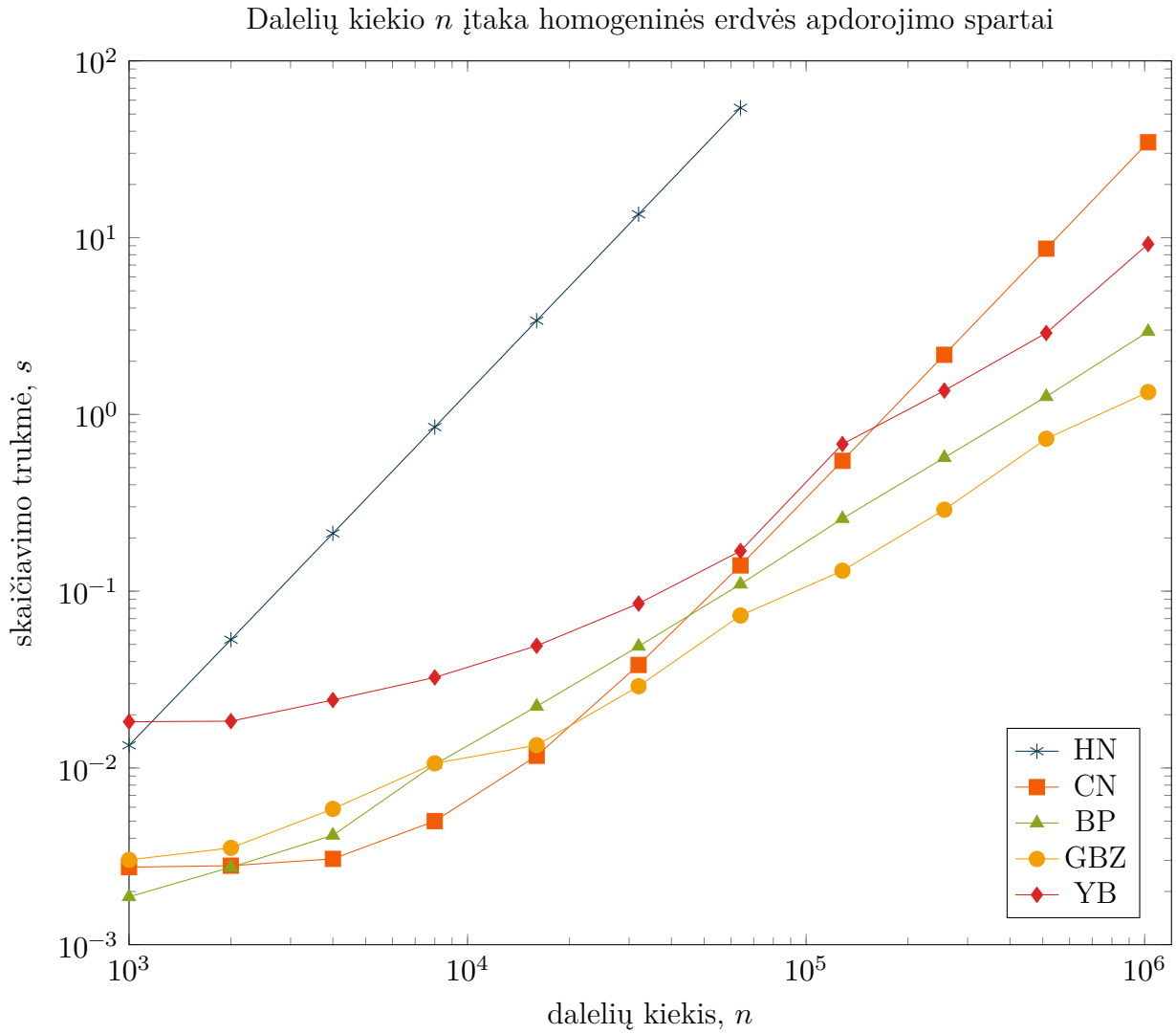
4.3 pav. Dviejų galaktikų susidūrimo dalelių pasiskirstymas

4.3 yra vaizduojamas pavyzdinis dviejų galaktikų susidūrimas. Tokiame modelyje yra tyrinėjama susidūrimo raida, bei rezultatas (kiek masės išlieka, kiek yra prarandama, kokia yra naujos galaktikos struktūra ir pan.).

4.3. EKSPERIMENTO REZULTATAI

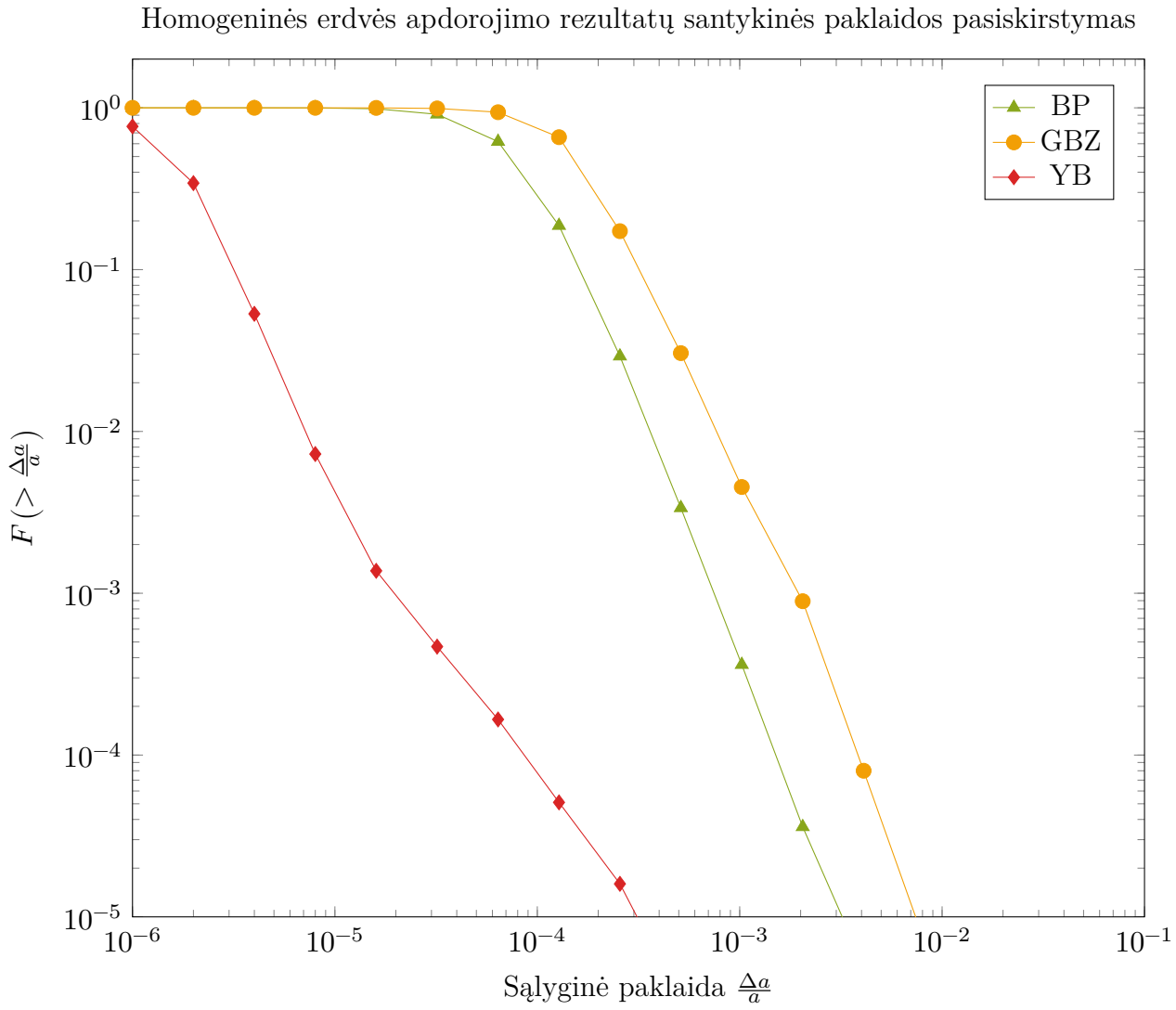
4.3.1. Homogeninis išsidėstymas

Pirmasis atliktas eksperimentas yra bendras spartos palyginimas. Visi algoritmai, išskyrus HN, buvo matuojami nuo $1000 \leq n \leq 1024000$. BP ir GBZ algoritmų atidarymo kampai θ yra 0.5.



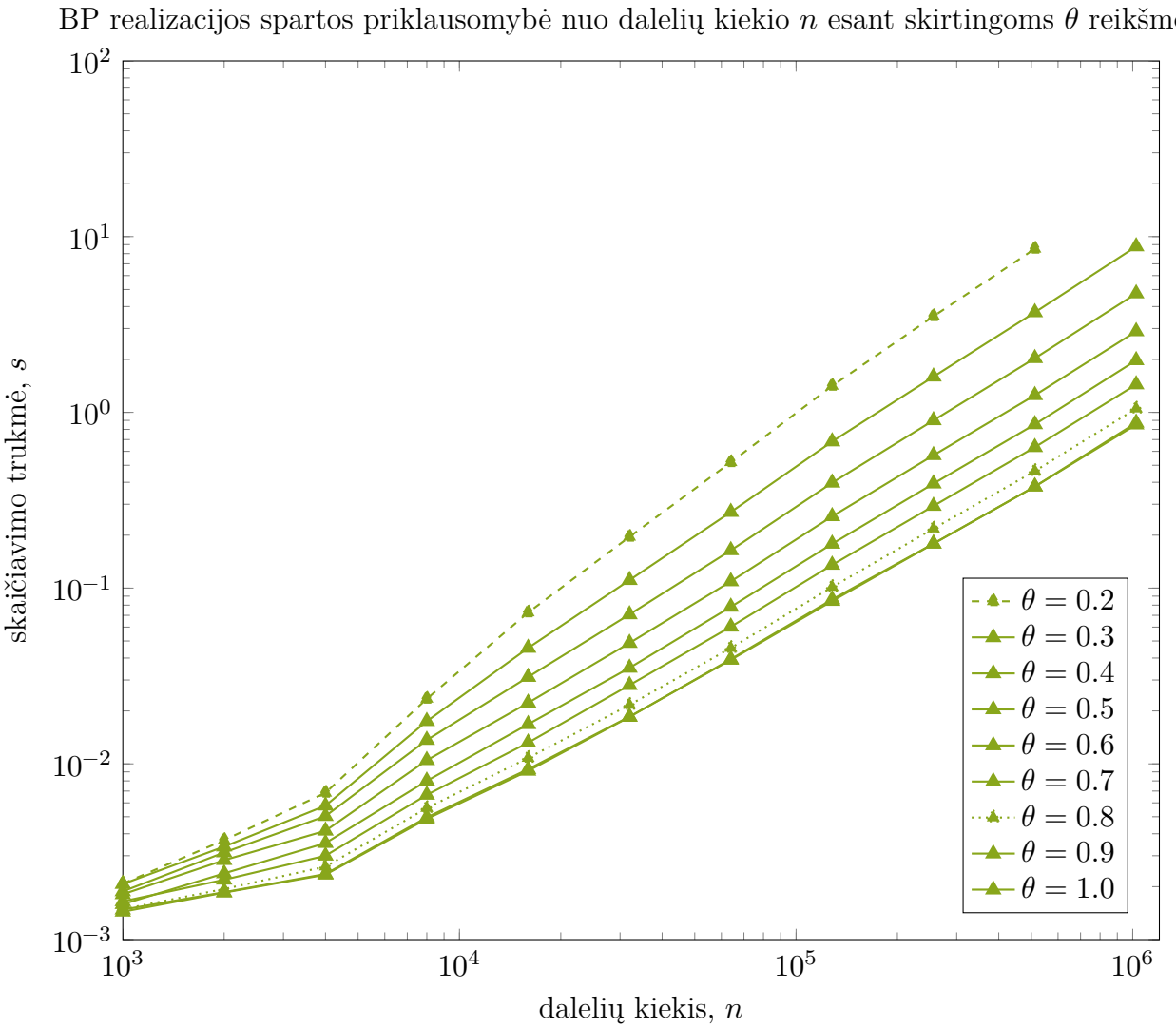
4.4 pav. Dalelių kiekio n įtaka homogeninės erdvės apdorojimo spartai

Paveiksle 4.4 ir lentelėje 7.1 pavaizduota kaip greitai kiekvienas algoritmas apskaičiuoja dalelių pagreičius, kai jos [dalelės] kai jos pasiskirsčiusios homogeniškai. HN algoritmas yra pats lėčiausias ir ties visais n išlaiko $O(n^2)$ sudėtingumą t. y. dvigubinant n, apdorojimo trukmė didėja 4 kartus. CN algoritmo kvadratinis sudėtingumas pradeda ryškėti kai $n \geq 16000$ ir sekant rezultatus atgal, matome kaip apdorojimo trukmė konverguoja iki 0,0027 sekundės ir galima teigti, kad tai yra CUDA posistemės valdymo išlaidos (angl. *overhead*). Kai $16000 \leq n \leq 1024000$ YB realizacija yra lėčiausia, BP yra vidutinė, o GBZ — greičiausia. BP realizacijos apdorojimo trukmės didėjimas yra stabiliausias: apie 2,25 karto, tačiau iš grafiko matyti, kad GBZ realizacija turi panašią tendenciją. YB realizacijos apdorojimo trukmės prieaugis yra neaiškus, nes rezultatai nėra tokie stabilūs kaip kad BP arba netgi GBZ realizacijos. Tačiau yra įmanoma pastebėti, kad YB algoritmo valdymo išlaidos yra apie 0,018s.



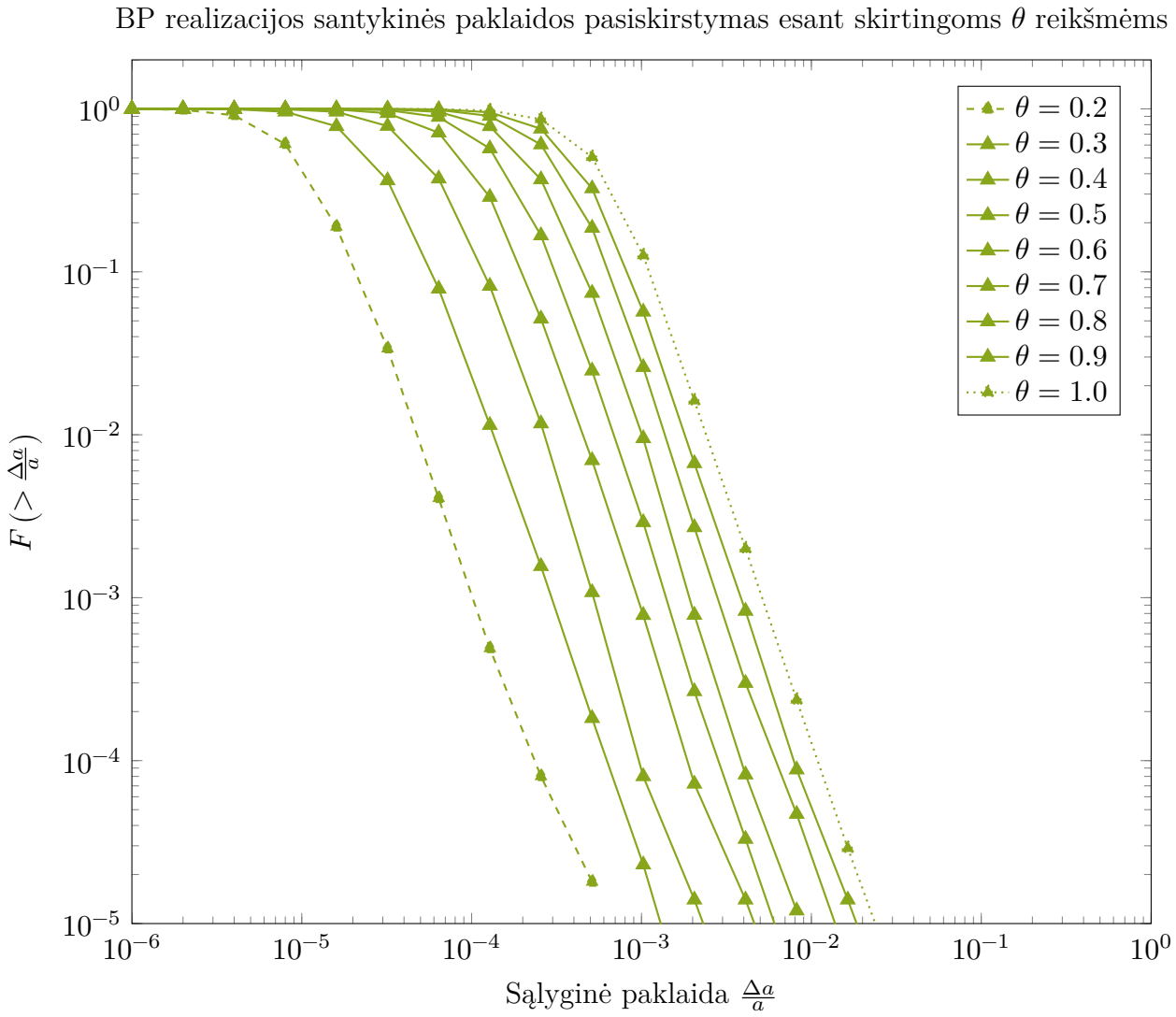
4.5 pav. Homogeninės erdvės apdorojimo rezultatų santykinės paklaidos pasiskirstymas

Paveiksle 4.5 ir lentelėje 7.2 pavaizduota kiekvieno algoritmo realizacijos tikslumas, apdorojant homogeninę erdvę. YB algoritmas yra tiksliausias, po to eina BP ir galiausia GBZ. YB realizacijos rezultate tiktai apie 0,7% visų dalelių turi paklaidą, didesnę negu 0,0008%, o tai yra labai tikslu. BP rezultate praktiškai visos dalelės turi paklaidą nemažesnę negu 0,0016%, bet nedidesnę negu 0,0256%. GBZ realizacijos rezultatų paklaida yra panaši į BP, tačiau turi skirtingą intervalą: nuo 0,0032% iki 0,0512%.



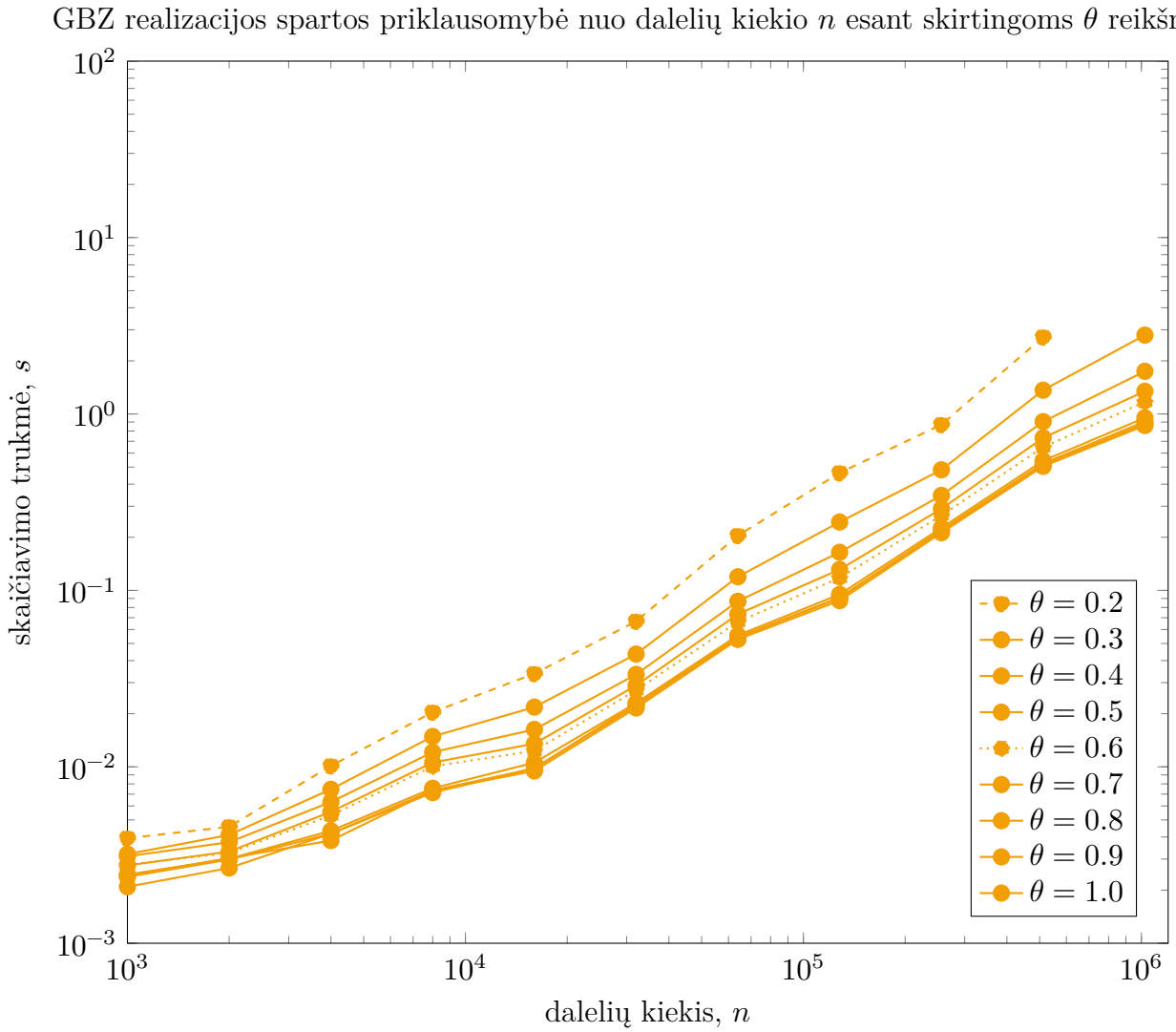
4.6 pav. BP realizacijos homogeninės erdvės apdorojimo spartos priklausomybė nuo dalelių kiekio n esant skirtingoms θ reikšmėms

Paveiksle 4.6 ir lentelėje 7.3 pavaizduota BP algoritmo spartos priklausomybė nuo dalelių kiekio n, esant skirtingiems atidarymo kampams θ. Aukščiausia kreivė (mažiausia sparta) yra $\theta = 0.2$, o žemiausios kreivės (sparčiausios) yra $\theta = 0.9$ ir $\theta = 1.0$ — grafike sunku įžvelgti, tačiau jos yra susiliejusios. Kaip ir buvo tikimasi, didinant atidarymo kampą, sparta irgi didėja. Taip pat matyti, kad spartos prieaugis geometriškai mažėja — einant iš $\theta = 0.2$ į $\theta = 0.3$ sparta padidėja apie 2,2 karto, o iš $\theta = 0.9$ į $\theta = 1.0$ spartos prieaugis yra praktiškai nepastebimas.



4.7 pav. BP realizacijos santykinės paklaidos pasiskirstymas esant skirtingoms θ reikšmėms

Paveiksle 4.7 ir lentelėje 7.4 vaizduojama BP realizacijos sąlyginės paklaidos pasiskirstymas, esant skirtingiems atidarymo kampams θ. Kairiausia kreivė yra $\theta = 0.2$ ir ji parodo, kad labiausiai tikėtina rasti dalelė, kurios paklaida yra nuo 0,0004% iki 0,0032%. Dešiniausia kreivė yra $\theta = 1.0$ ir ji reiškia, kad tokiame rezultate labiausiai tikėtina rasti dalelę, kurios paklaida yra nuo 0,0128% iki 0,2%.



4.8 pav. GBZ realizacijos spartos priklausomybė nuo dalelių kiekio n esant skirtingoms θ reikšmėms

Paveiksle 4.8 ir lentelėje 7.5 vaizduojama GBZ realizacijos spartos priklausomybė, nuo dalelių kiekio n, esant skirtingiems atidarymo kampams θ. Aukščiausia kreivė yra $\theta = 0.2$, o žemiausia $\theta = 1.0$. Kaip ir BP eksperimente GBZ spartos kreivė ne tik kad mažėja, bet ir išlaiko savo „formą“. Taip pat matyti, kad spartos prieaugis praktiškai sustoja, ties $\theta = 0.7$.

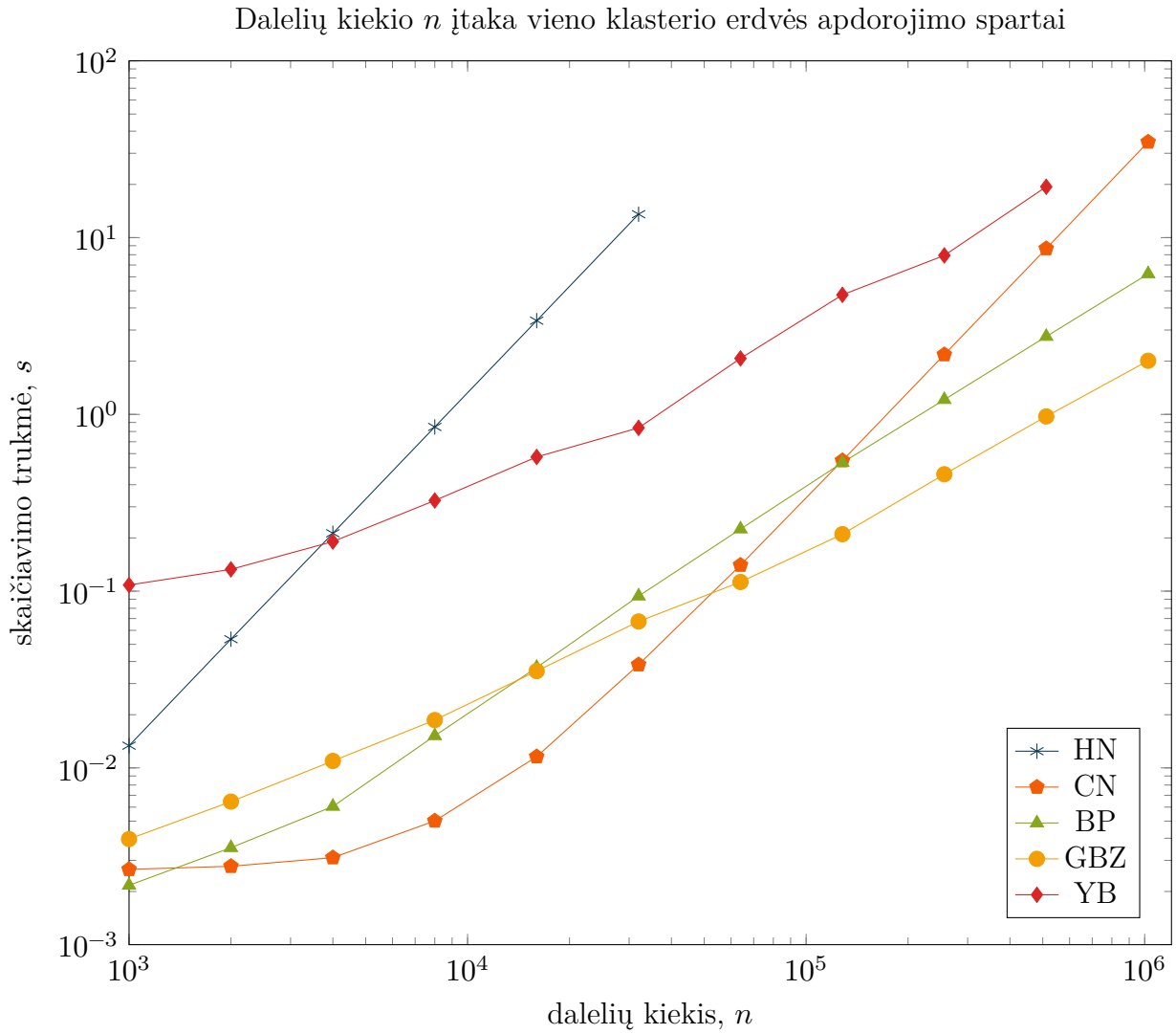
4.9 pav. GBZ realizacijos santykinės paklaidos pasiskirstymas esant skirtingoms θ reikšmėms

Paveiksle 4.9 ir lentelėje 7.6 vaizduojama GBZ realizacijos sąlyginės paklaidos pasiskirstymas, esant skirtingiems atidarymo kampams θ. Kai $\theta = 0.2$ (kairiausia kreivė) labiausiai tikėtina dalelės paklaida yra apytikriai nuo 0,0002% iki 0.0016%, o kai $\theta = 1.0$:

- labiausiai tikėtina paklaida yra apytikriai nuo 0,2% iki 3,27%
- modelyje egzistuoja dalelių, kurių paklaida yra didesnė negu 26,2%, bet nedidesnė negu 52,4%
- Apytikriai nuo 7% iki 31% visų dalelių turi paklaida didesnę negu 1%

4.3.2. Vieno klasterio išsidėstymas

Vykdant vieno klasterio išsidėstymo eksperimentą, yra tikimasi pamatyti algoritmų sulėtėjimą, dėl to kad konstruojamas medis yra išsibalansavęs.

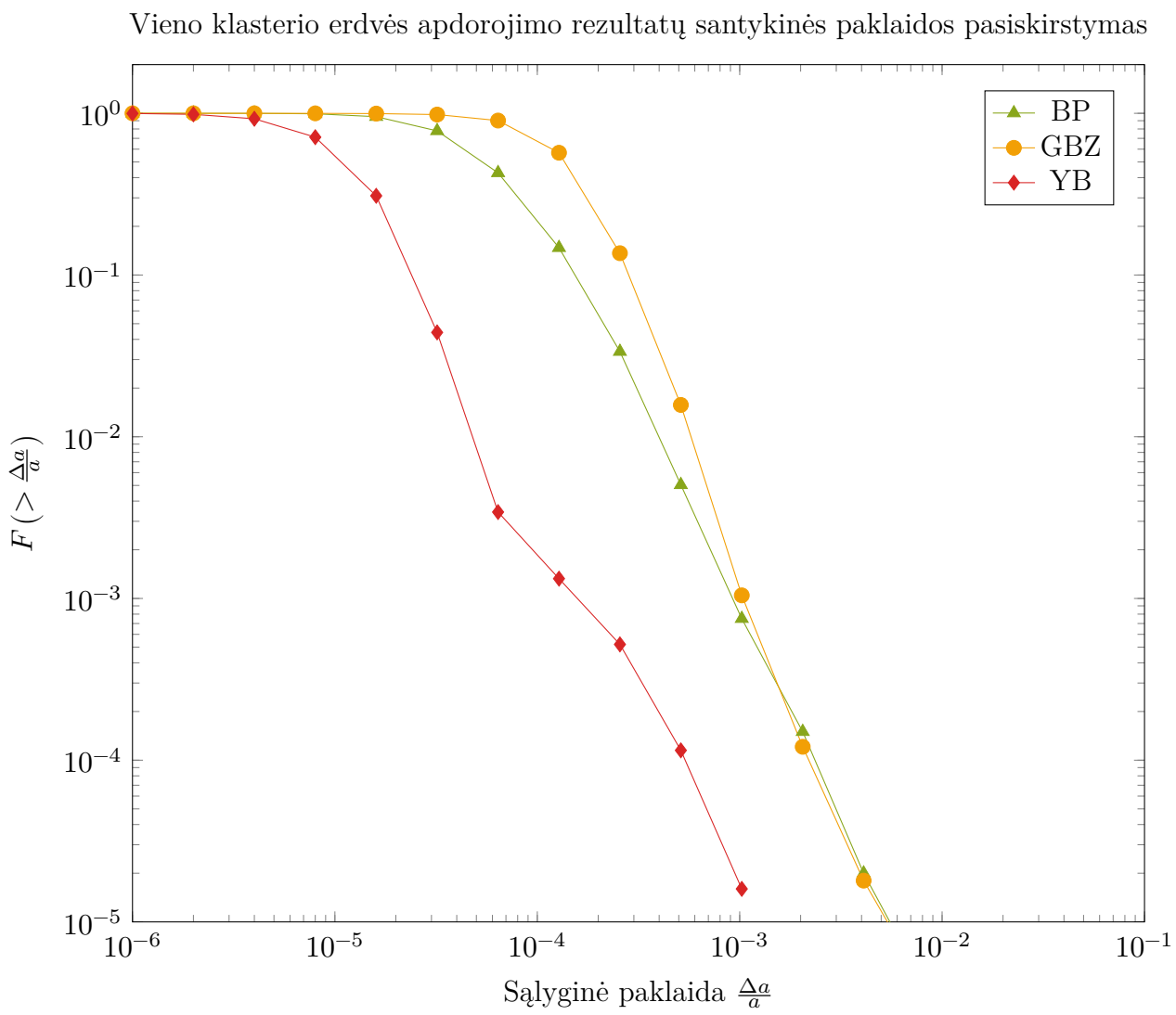


4.10 pav. Dalelių kiekio n įtaka vieno klasterio erdvės apdorojimo spartai

Paveikslas 4.10 ir lentelė 7.7 vaizduoja kaip priklauso skirtingų realizacijų spalvos nuo dalelių kiekio n. Rezultatai yra labai panašūs į homogeninės erdvės, tačiau yra pastebimas žymus sulėtėjimas GB, GBZ ir YB realizacijose. HN ir CN realizacijos yra nepakitusios. Lyginant su homogeninės erdvės spartos rezultatais:

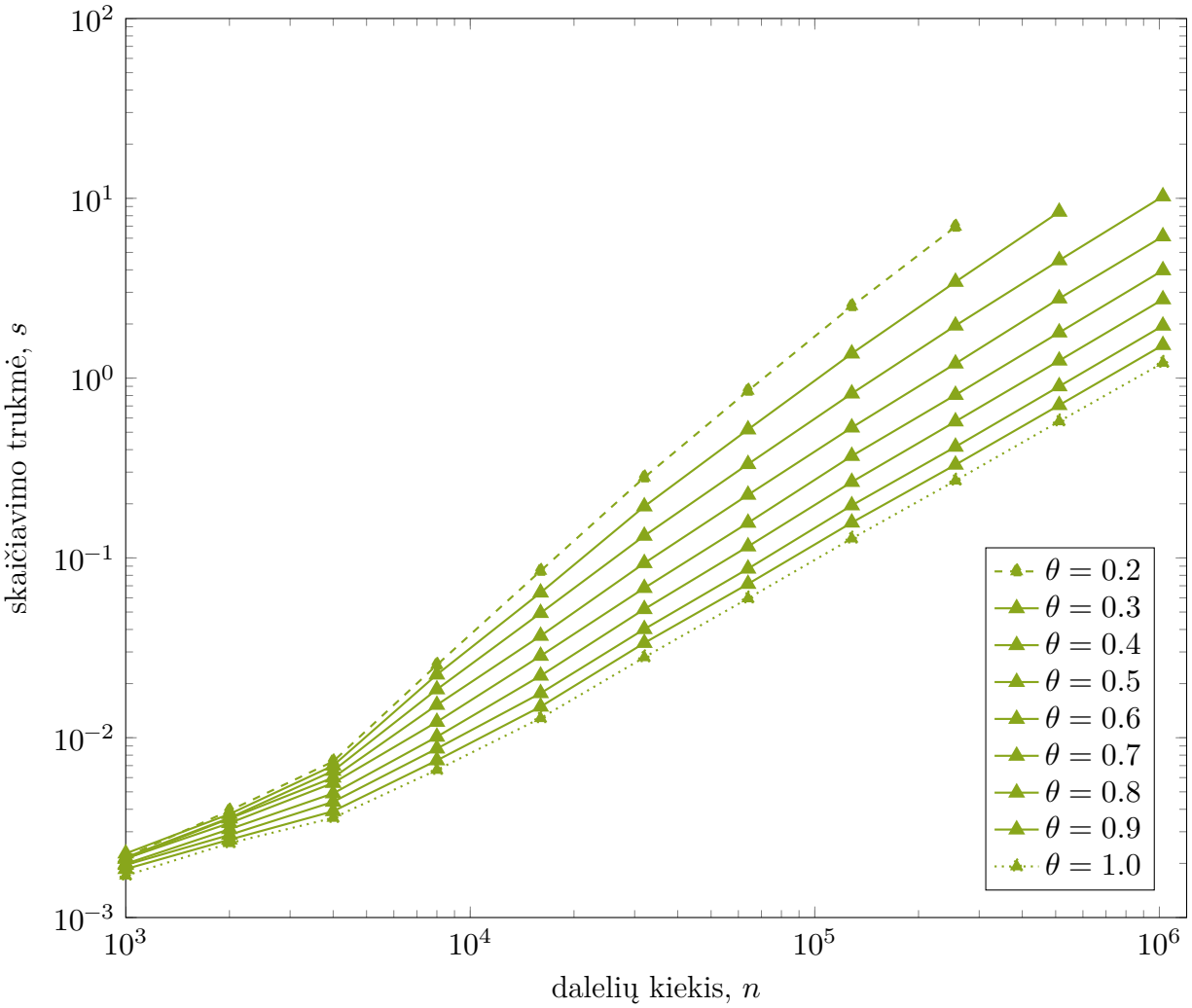
- BP — sulėtėjimas mažiau pastebimas kai $n = 1000$ (apie 1,16 karto) ir labiausiai pastebimas kai $n = 512000$ (apie 2,18 karto).
- GBZ — vidutiniškai turi mažiausiai pastebima sulėtėjimą (apie 1,75 karto). Kai $n = 1024000$ realizacija sulėtėja apie 1,5 karto.
- YB — vidutiniškai turi didžiausią sulėtėjimą (apie 8.43 karto). Kai $n = 512000$ realizacija sulėtėja apie 6,71 karto.

Taip pat pastebėta, kad GBZ realizacija kai $16000 \leq n \leq 32000$ patiria didžiausią spartos praradimą, o YB realizacijai šis efektas matomas kai $8000 \leq n \leq 64000$.



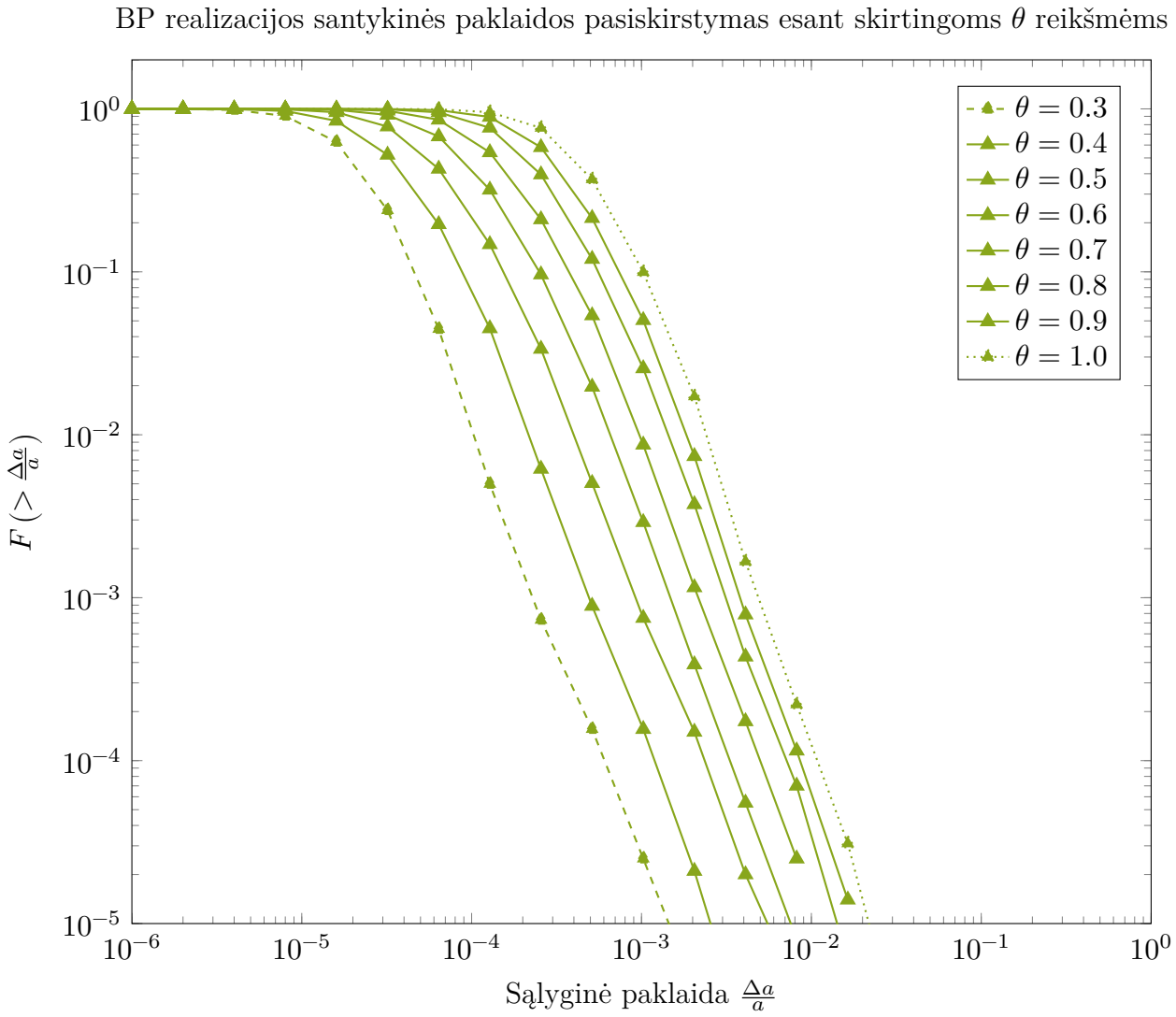
4.11 pav. Vieno klasterio erdvės apdorojimo rezultatų santykinės paklaidos pasiskirstymas

Paveiksle 4.11 lentelėje 7.8 vaizduojama kiekvieno algoritmo realizacijos paklaidos pasiskirstymas, apdorojant vieno klasterio erdvę. YB realizacija yra tiksliausia — dalelių labiausiai tikėtina paklaida yra nuo 0,0002% iki 0,0032%. Po to, eina BP realizacija — dalelių labiausiai tikėtina paklaida yra nuo 0,0016% iki 0,0256%. Mažiausiai tiksli realizacija yra GBZ: nuo 0,0032% - 0,0512%.



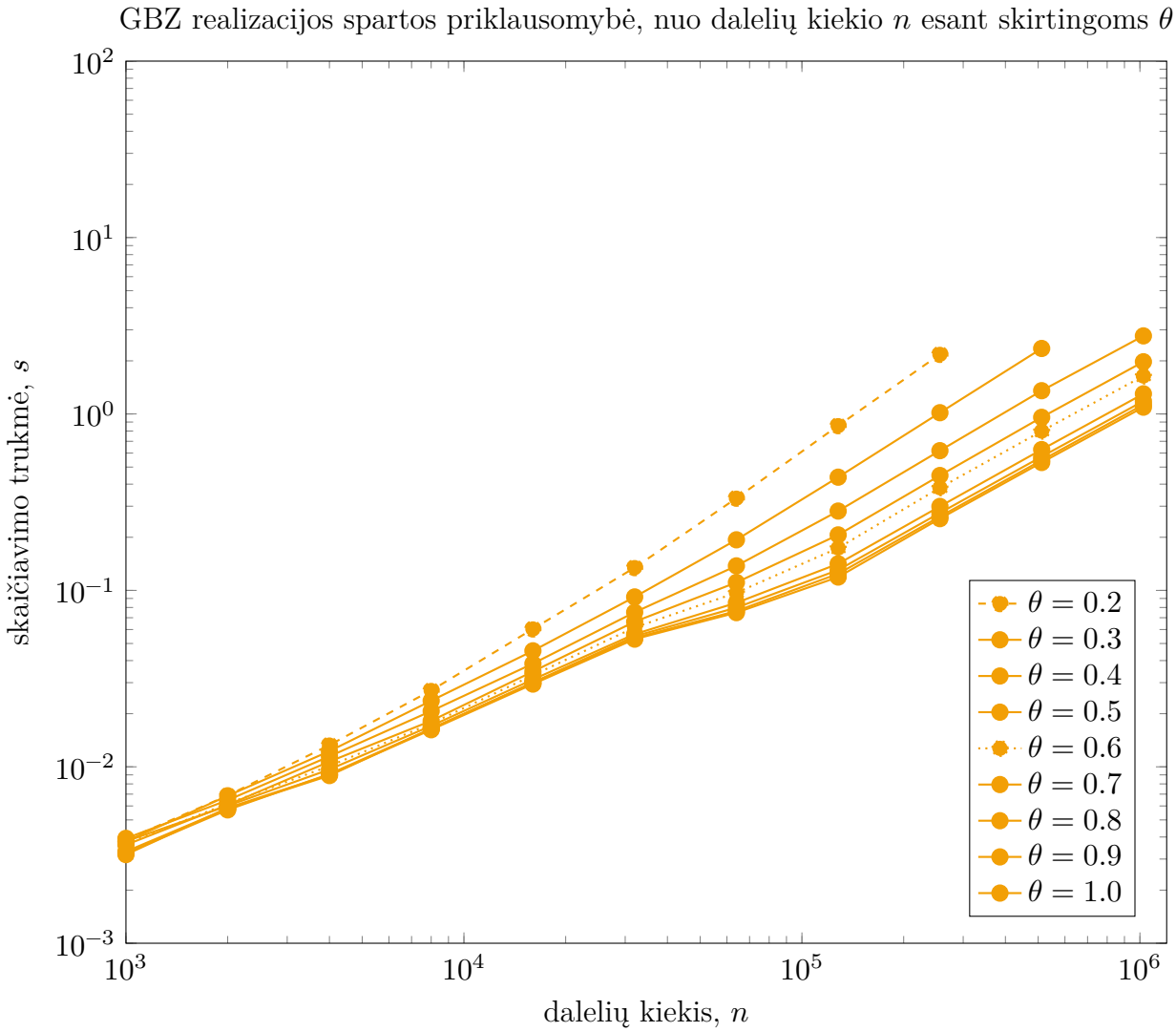
4.12 pav. BP realizacijos spartos priklausomybė nuo dalelių kiekio n esant skirtingoms θ reikšmėms

Paveiksle 4.12 ir lentelėje 7.9 vaizduojama BP realizacijos spartos priklausomybė, nuo dalelių kiekio n, esant skirtingiems atidarymo kampams θ. Spartos yra panašios į homogeninės erdvės, tačiau nėra matoma, kad sparta nustotų augti ties $\theta = 0.9$. Paskaičiavus, kiek kartų krinta sparta, lyginant su homogenine erdve, ties kiekviena n ir θ kombinacija, pastebėta, kad daugiausia spartos yra prarandama (iki 2,5 karto), kai θ yra mažesnis, o n yra didesnis.



4.13 pav. BP realizacijos santykinės paklaidos pasiskirstymas esant skirtingoms θ reikšmėms

Paveiksle 4.13 ir lentelėje 7.10 vaizduojama BP realizacijos sąlyginės paklaidos pasiskirstymas, esant skirtingiems atidarymo kampams θ. Paklaidos pasiskirstymas yra labai panašus i homogeninės, tačiau vidutinė paklaida yra nežymiai didesnė.

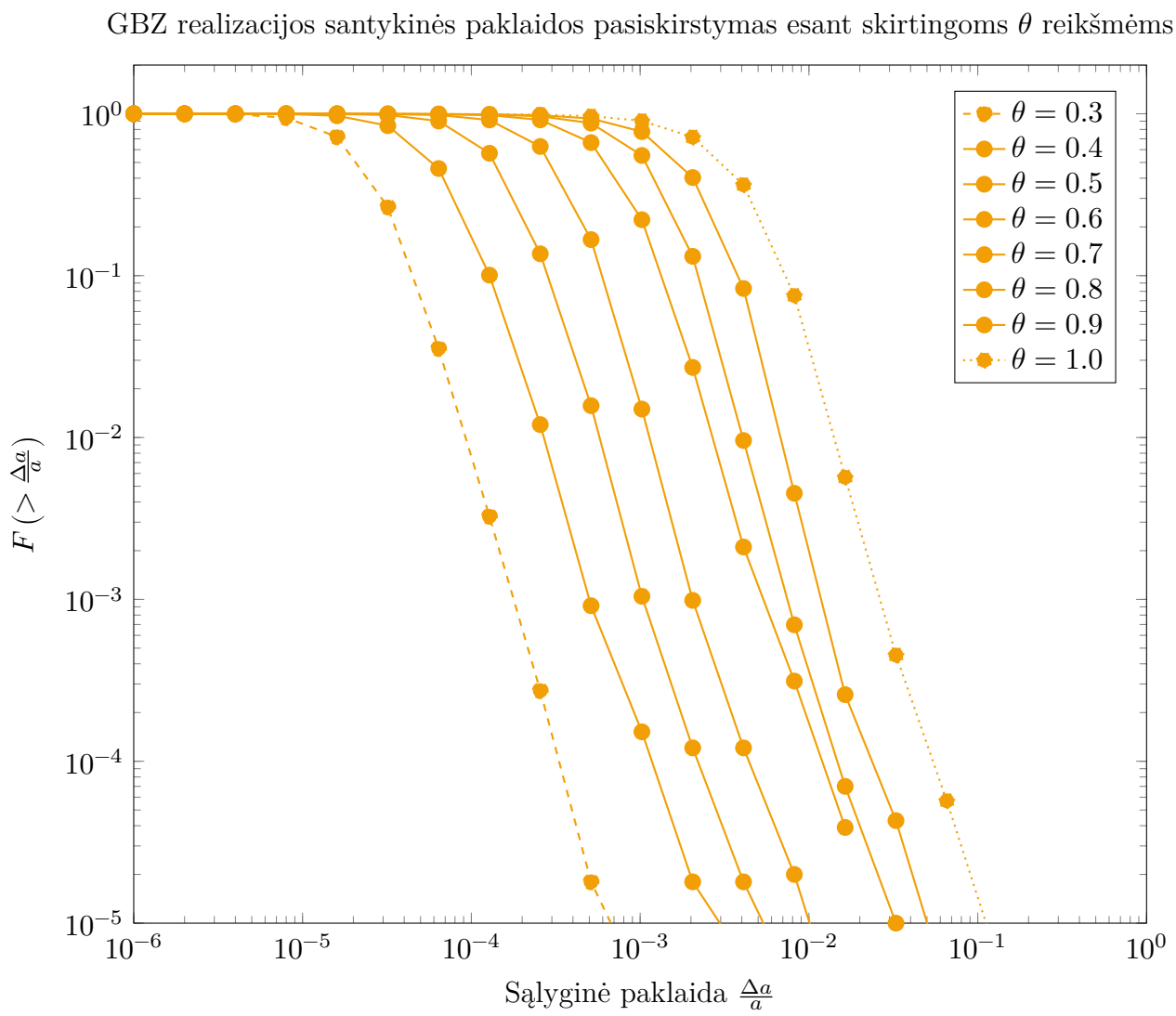


4.14 pav. GBZ realizacijos spartos priklausomybė, nuo dalelių kiekio n esant skirtingoms θ

Paveiksle 4.14 ir lentelėje 7.11 vaizduojama GBZ realizacijos spartos priklausomybė nuo dalelių kiekio n, esant skirtingiems atidarymo kampams θ. Lyginant su homogeninės erdvės grafiku, matyti:

- Ties žemais θ grafikas yra lygesnis.
- Yra pastebimas tarpas tarp $\theta = 0.7$ ir $\theta = 0.8$.
- Apatinė spartos riba (kai θ yra 0.8, 0.9, 1.0) turi specifinę forma.

Patikrinus kiek kartų skiriasi spartą ties kiekviena n ir θ kombinacija, matyti, kad daugiausia spartos (apie 2-3 kartus) yra prarandama, kad $8000 \leq n \leq 32000$.

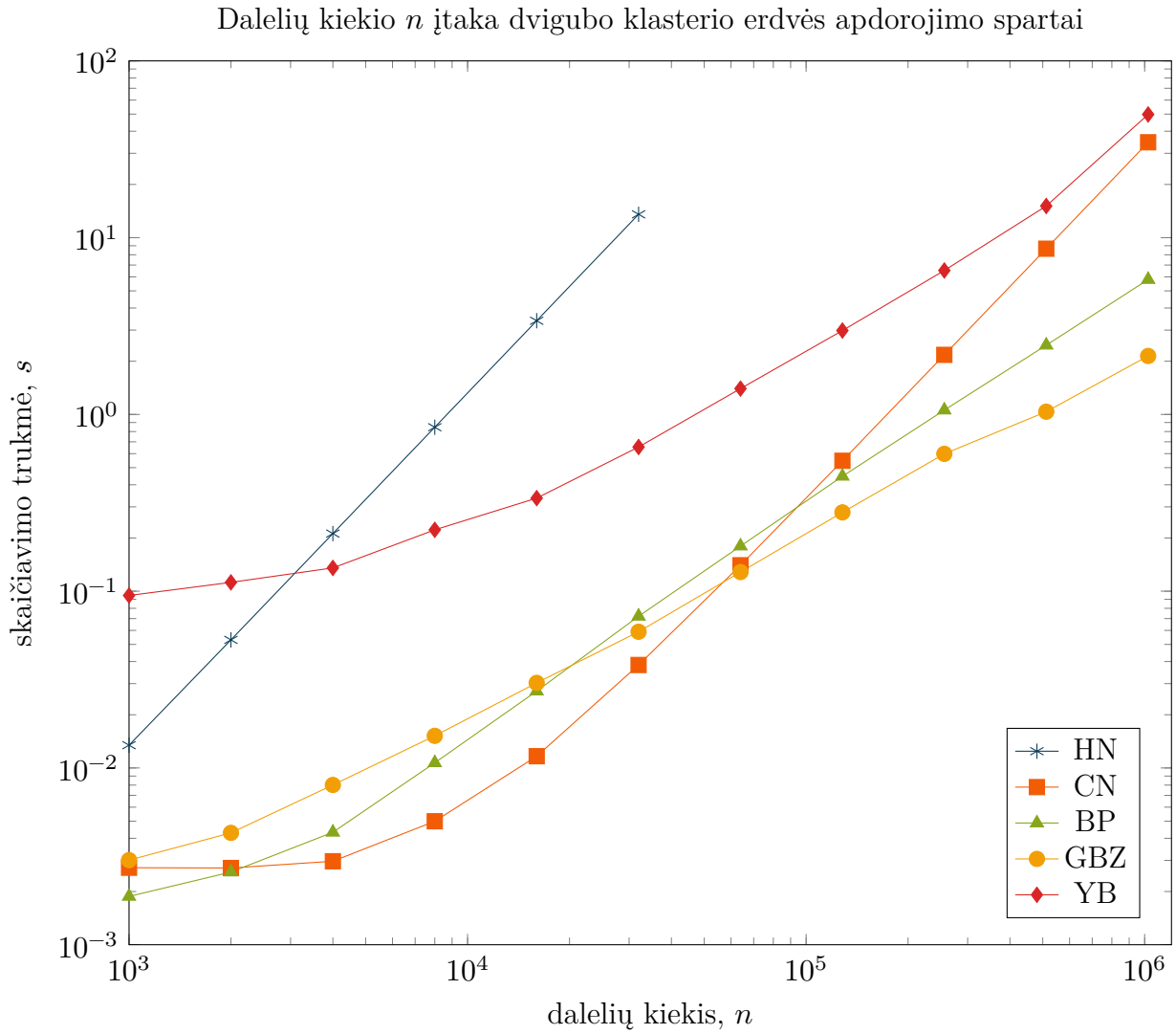


4.15 pav. GBZ realizacijos santykinės paklaidos pasiskirstymas esant skirtingoms θ reikšmėms

Paveiksle 4.15 ir lentelėje 7.12 vaizduojama GBZ realizacijos sąlyginės paklaidos pasiskirstymas, esant skirtingiems atidarymo kampams θ. Kaip ir BP realizacijos, paklaidos pasiskirstymas mažai skiriasi nuo homogeninės erdvės.

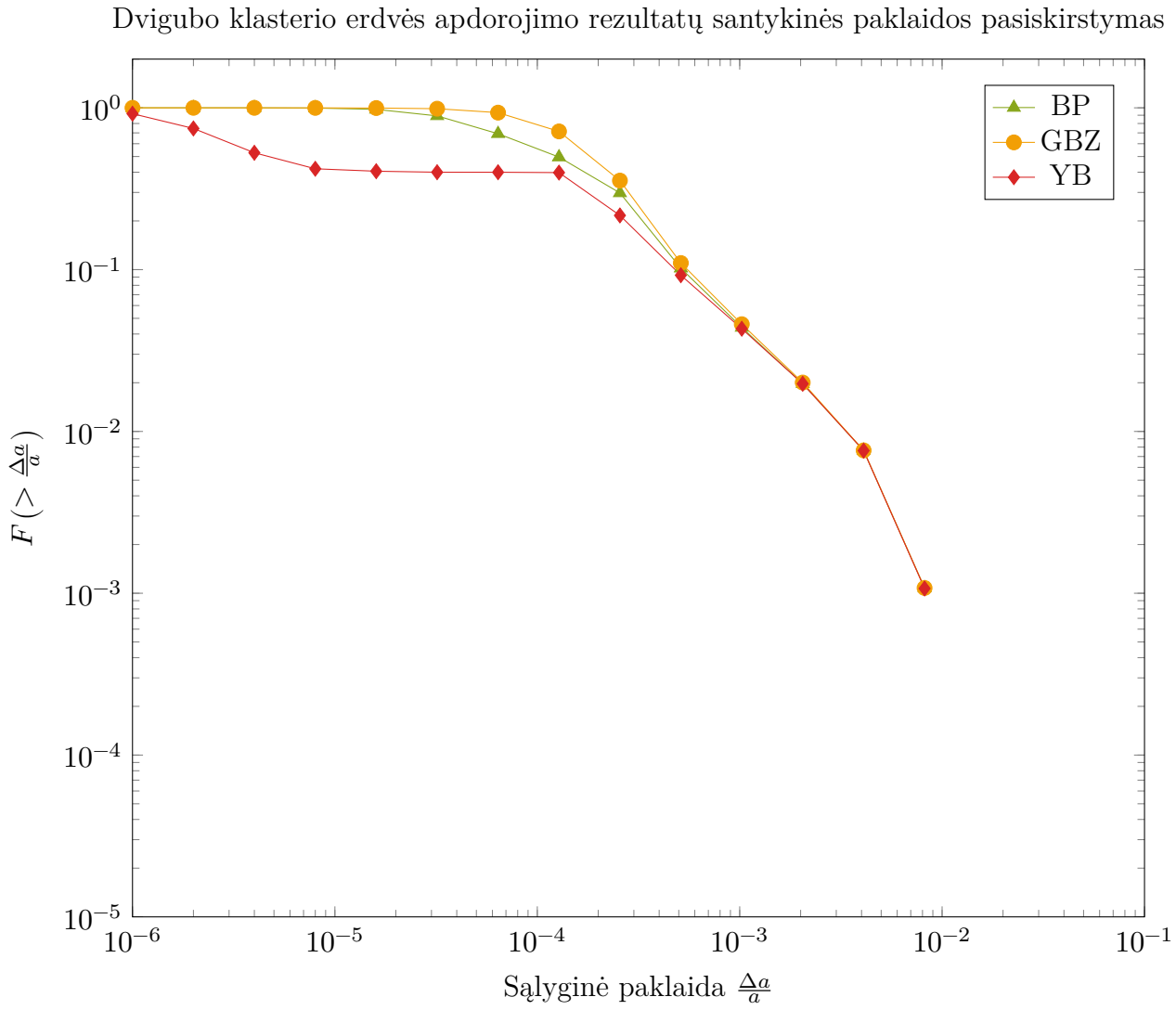
4.3.3. Dviejų klasterių išsidėstymas

Eksperimentas buvo pakartotas esant dviejų klasterių išsidėstymui. Pirmasis klasteris turi 40% visų sistemos dalelių ir 100 atstumo vienetų spindulį; antrasis klasteris turi 60% visų dalelių ir 200 atstumo vienetų spindulį. Abu klasteriai yra nutolę vienas nuo kito apytikriai per 361 atstumo vienetų.



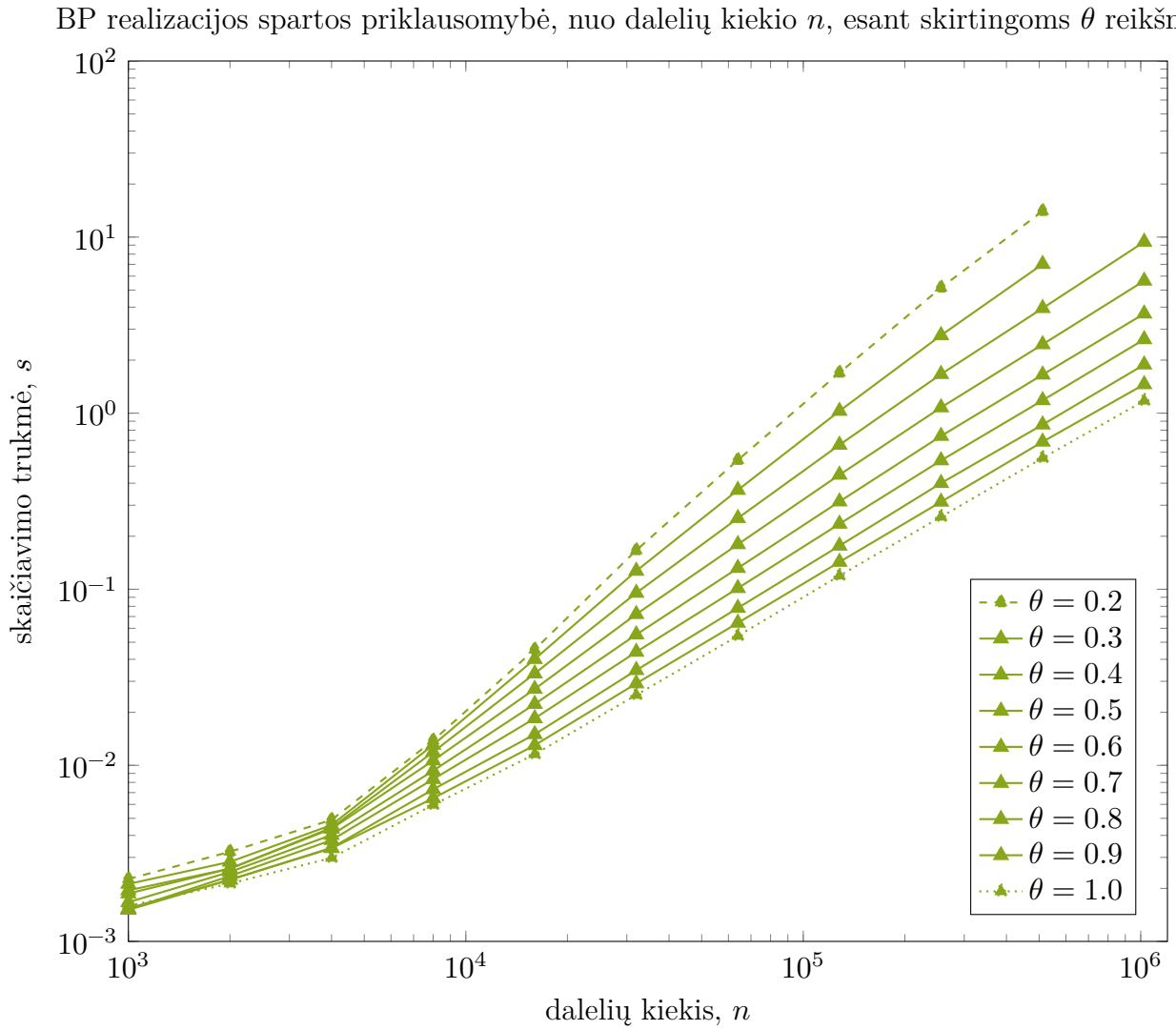
4.16 pav. Dalelių kiekio n įtaka dvigubo klasterio erdvės apdorojimo spartai

Paveiksle 4.16 ir lentelėje 7.13 vaizduojama visu realizacijų spartos priklausomybė nuo dalelių kiekio n, kai apdorojama dviejų klasterių erdvė. Sparta vidutiniškai mažiau sumažėja, einant iš homogeninio išsidėstymo į dviejų klasterių išsidėstymą, negu einant iš homogeninio išsidėstymo į vieno klasterio išsidėstymą. BP realizacija vidutiniškai sulėtėja apie 1,44 karto, o kai $n = 1024000$ — 1,97 karto. GBZ realizacija vidutiniškai sulėtėja apie 1,66 karto, tačiau, ankščiau pastebėtas spartos nuosmukio paaštrėjimas, dabar pastebimas tarp $16000 \leq n \leq 256000$. BP realizacija vidutiniškai sulėtėja apie 6,03 karto.



4.17 pav. Dvigubo klasterio erdvės apdorojimo rezultatų santykinės paklaidos pasiskirstymas

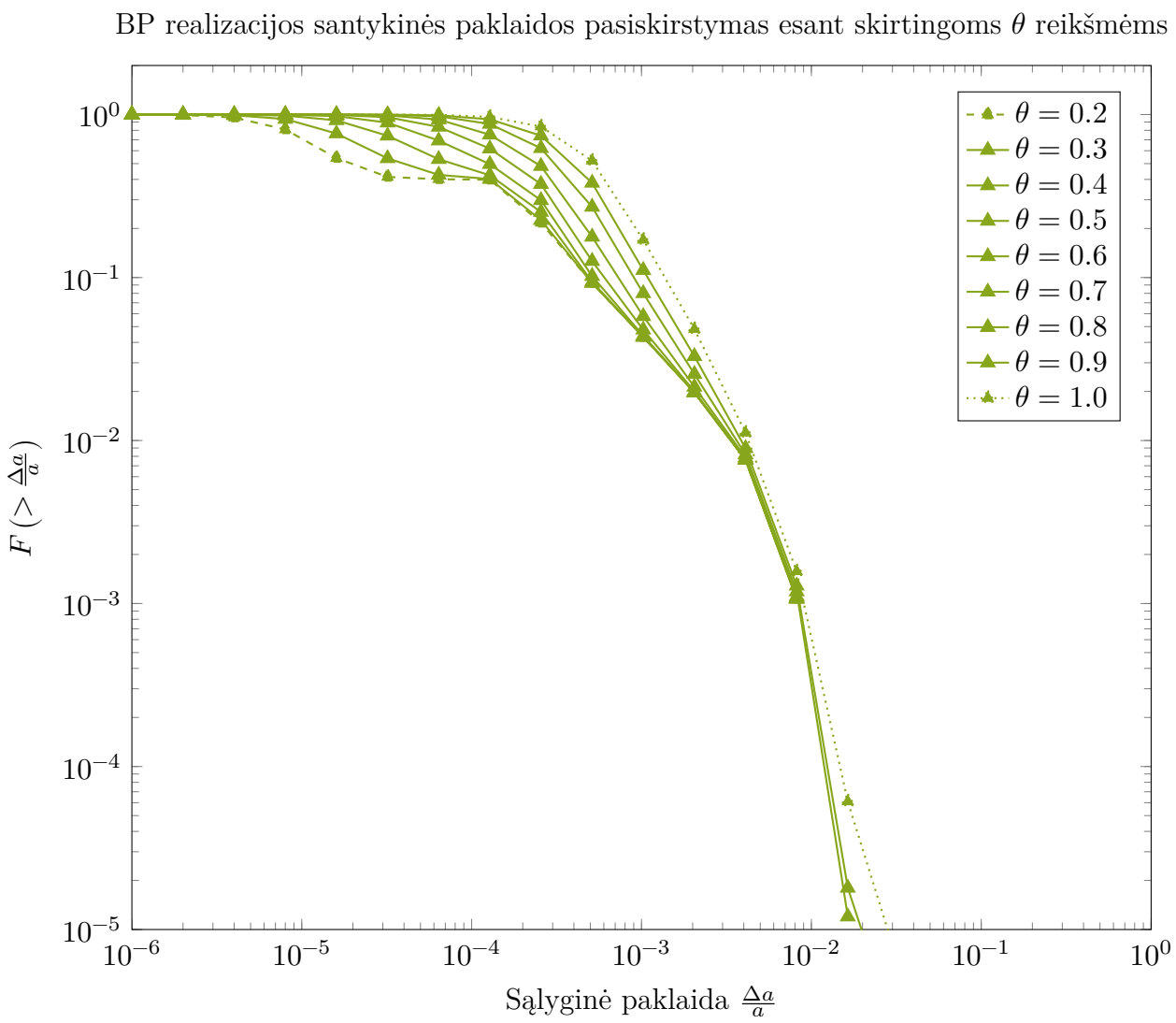
Paveiksle 4.17 ir lentelėje 7.14 vaizduojamas BP, GBZ ir YB realizacijų paklaidos pasiskirstymas, apdorojant dviejų klasterių erdvę. Matyti išlikusi tendencija, kad YB realizacija turi aukščiausia tikslumą, o GBZ mažiausią. Įdomiausia tai, kad matyti minimali paklaidos riba — visose realizacijose apie 40% dalelių turi bent 0,0128% paklaidą, 10% turi bent 0,0512% paklaidą.



4.18 pav. BP realizacijos spartos priklausomybė, nuo dalelių kiekio n esant skirtingoms θ reikšmėms

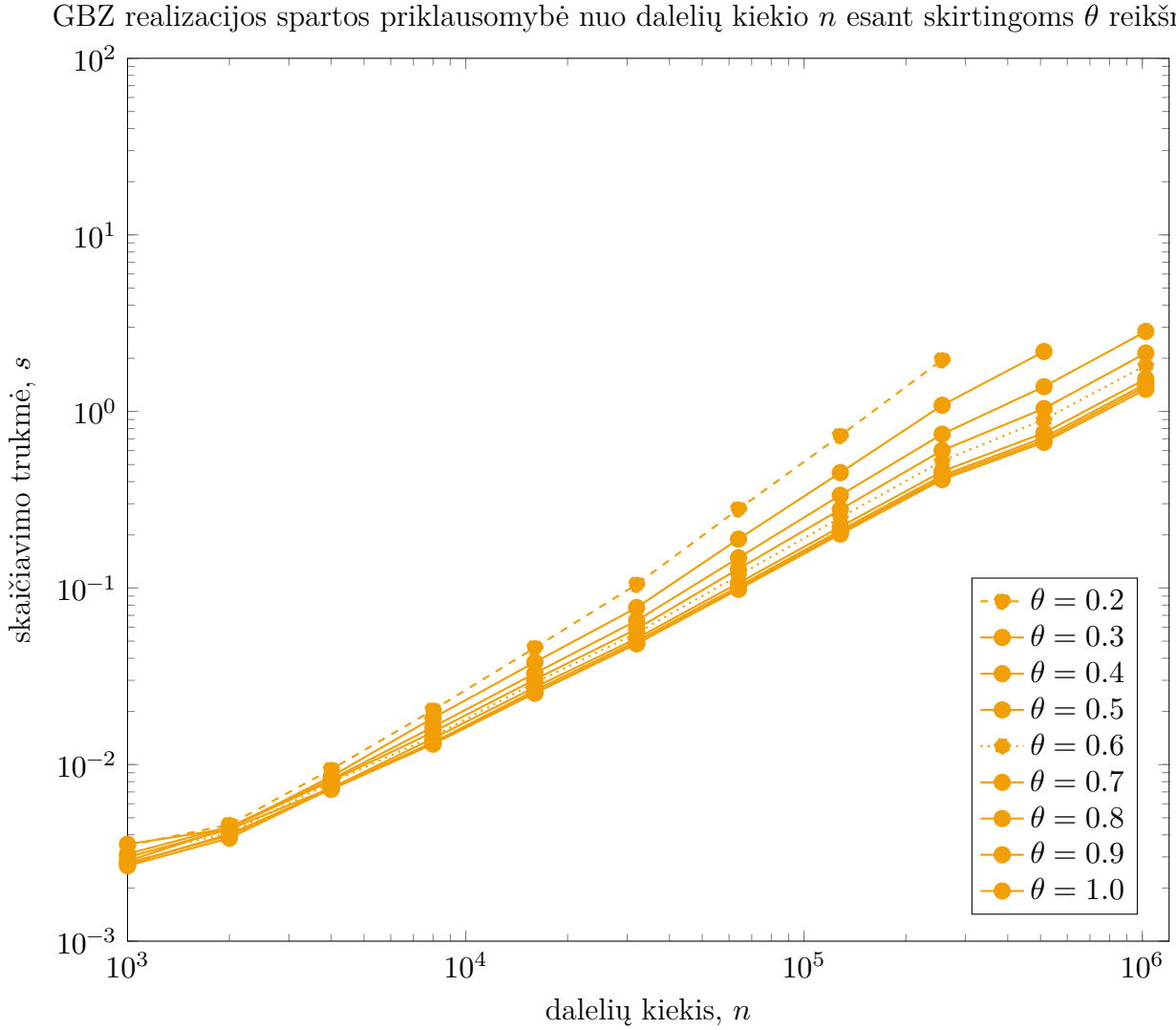
Paveiksle 4.18 ir lentelėje 7.15 vaizduojama BP algoritmo spartos priklausomybė nuo dalelių kiekio n esant skirtingiems atidarymo kampams θ. Spartos priklausomybė yra labai panaši į vieno klasterio erdvės. Skirtumai:

- Dviejų klasterių erdvės apdorojimas yra šiek tiek spartesnis, negu vieno klasterio erdvės.
- Lyginant su homogenine erdve, spartos yra daugiau prarandama, kai n yra didesnis. T. y. *theta* įtaka yra sunkiai pastebima.



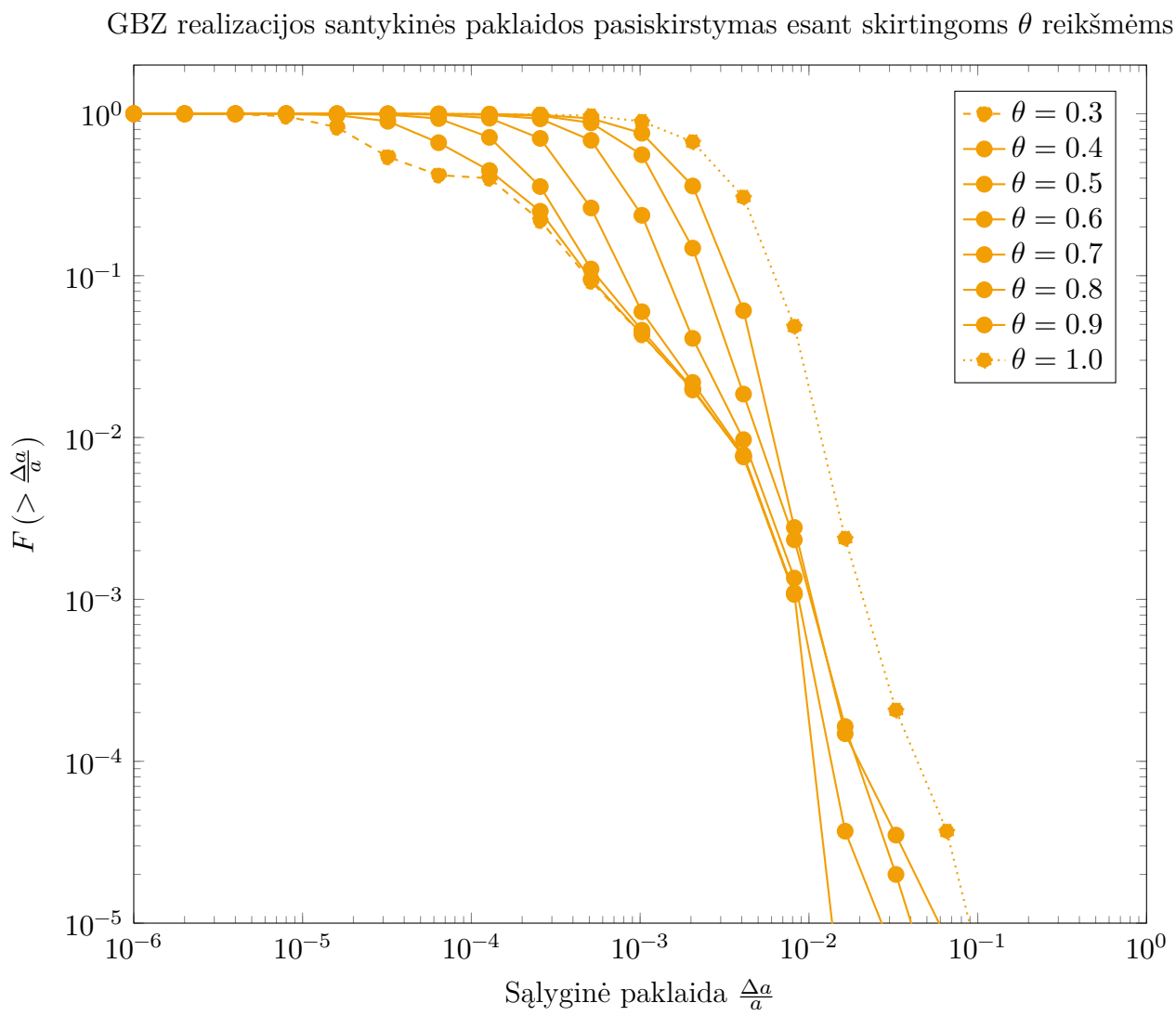
4.19 pav. BP realizacijos santykinės paklaidos pasiskirstymas esant skirtingoms θ reikšmėms

Paveiksle 4.19 ir lentelėje 7.16 vaizduojamas BP realizacijos paklaidos pasiskirstymas esant skirtingiems atidarymo kampams θ. Pagrinde matyti ta pati apatinė riba, kaip ir 4.17.



4.20 pav. GBZ realizacijos spartos priklausomybė nuo dalelių kiekio n esant skirtingoms θ reikšmėms

Paveiksle 4.20 ir lentelėje 7.11 vaizduojama GBZ realizacijos spartos priklausomybė nuo dalelių kiekio n esant skirtingiems atidarymo kampams θ.



4.21 pav. GBZ realizacijos santykinės paklaidos pasiskirstymas esant skirtingoms θ reikšmėms

Paveiksle 4.21 ir lentelėje 7.18 vaizduojamas GBZ realizacijos paklaidos pasiskirstymas, esant skirtingiems atidarymo kampams θ. Pagrinde matyti ta pati apatinė riba, kaip ir 4.17. Didėjant atidarymo kampui paklaidos pasiskirstymas po truputi „atlimpa“ nuo apatinės ribos, ir $\theta = 0.7$ tampa panašus į viengubo klasterio arba homogeninės erdvės paklaidos pasiskirstymus.

4.4. SPRENDIMO VEIKIMO IR SĄVYBIŲ ANALIZĖ, KOKYBĖS KRITERIJŲ ĮVERTINIMAS

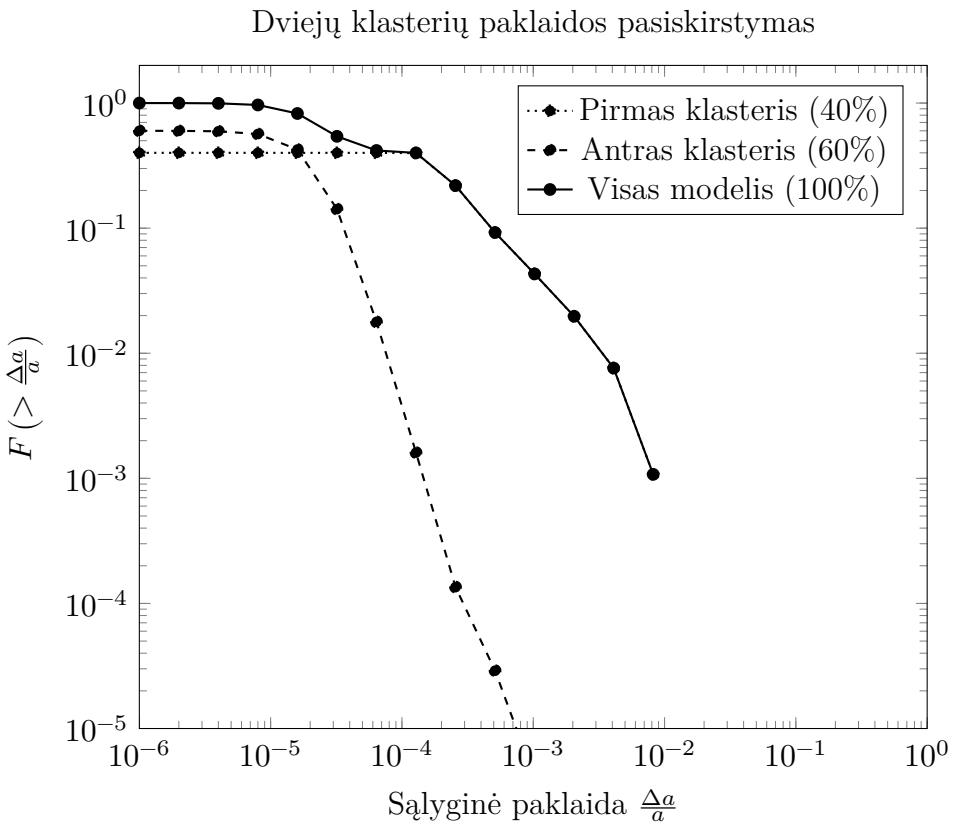
Aukščiau aprašyto eksperimento rezultatai rodo, kad YB realizacija yra pati lėčiausia ir tiksliausia. Taip pat ji pati jautriausia erdvės išsidėstymo pokyčiams — einant iš homogeninės į vieno klasterio išsidėstymą, sparta sumažėja apie 6,71 karto; einant į dvigubo klasterio išsidėstymą, sparta sumažėja apie 6,03 karto. Lyginant homogeninės ir dvigubo klasterio erdvių rezultatus yra pastebimas žymus tikslumo pokytis, o kitos realizacijos keičiasi nežymiai.

BP realizacija yra truputi lėtesnė bei tikslesnė negu GBZ realizacija. Taip pat ji turi vidutinį jautrumą, kuris yra truputi didesnis negu GBZ. Esant homogeniniam išsidėstymui buvo pastebėta spartos riba ties $\theta = 0.9$.

GBZ realizacija yra sparčiausia, tačiau turi mažiausią tikslumą. Taip pat ji mažiausiai reaguoja į dalelių išsidėstymo pokyčius. Kai yra apdorojama homogeninio išsidėstymo erdvė, tampa sunku nuspėti kokia sparta ji bus apdorojama. Kai yra apdorojama vieno arba dvigubo klasterio erdvės, spartos priklausomybė nuo dalelių kiekio n yra lengviau nuspėti. Pastebėta, kad imant reikšmes $\theta > 0.7$, realizacija nespartėja.

Kai yra apdorojama dvigubo klasterio erdvė, visuomet pastebima tam tikra apatinė paklaidos riba. Buvo atlikti papildomi bandymai, kuriuose buvo keičiami dvigubo klasterio erdvės parametrai: klasterių santykiniai dalelių kiekiai, spinduliai, bei atstumas tarp jų. Buvo pastebėta, kad apatinės ribos forma kinta keičiant visus parametrus:

- Didinant atstumą tarp klasterių, riba nyksta — darosi mažiau pastebima.
- Keičiant santykinę dalelių koncentraciją, keičiasi ribos forma. Iš 4.17, 4.19 ir 4.21 matyti, kad 40% modelio dalelių turi tam tikrą paklaidą, ši 40% reikšmė yra tiesiogiai susijusi su pirmo klasterio dalelių kiekiu, kuris irgi yra 40%.



4.22 pav. Atidarymo kampo θ įtaka GBZ realizacijos tikslumui

Paveiksle 4.22 vaizduojama atskirų klasterių dalelių paklaidos pasiskirstymas. Matyti, kad pirmasis klasteris turi apie 16 kartų didesnę paklaidą, negu antrasis. Viso modelio paklaidos pasiskirstymas

yra gaunamas sudedant vieno klasterio paklaidos pasiskirstymą su kito.

4.5. SPRENDIMO TAIKYMO REKOMENDACIJOS

Matuojant GBZ realizacijos spartą pastebėta, kad kai $8000 \leq n \leq 32000$, ji [realizacija] yra jautriausia erdvės konfigūracijos pokyčiams. Galbūt, išsiaiškinus kuo šis n ruožas yra ypatingas, algoritmą būtų galima tobulinti.

Darbe atlikti įvairūs pastebėjimai gali būti panaudoti, kuriant hibridinį algoritmą. Nustačius dalelių išsidėstymo klasę (t. y. ar ji yra homogeninė, vieno klasterio, dviejų klasterių ir pan.) ir parinkus atitinkamą realizacijos ir θ kombinaciją, būtų galima išgauti balansą tarp greičio ir paklaidos.

5. REZULTATŲ APIBENDRINIMAS IR IŠVADOS

Išanalizavus probleminę sritį bei atlikus eksperimentinį tyrimą buvo prieitos išvados:

- Medžio kodo algoritmo sudėtingumas yra $O(n \log n)$ — geresnis už tiesioginio sumavimo $O(n^2)$; prastesnis už daugiapolių metodo $O(n)$. Aštuonmedžio pavidalo struktūra naudojama hierarchiniam erdvės skaidymui realizuoti. Modifikuota versija padaro algoritmą pritaikomą SIMD platformoms.
- Tik viena realizacija algoritmą pilnai vykdo grafinėje platformoje. Labiausiai skiriasi medžio struktūros realizavimas grafinės plokštės atmintyje bei medžio formavimas. Grafinės platformos realizacijos turi būti stipriai modifikuojamos, kad būtų galima prisitaikyti prie platformos trūkumų, bei kad būtų išnaudojami platformos privalumai.
- Svarbiausia algoritmo savybė (ir pagrindinis palyginimo kriterijus) yra sparta. Antriniai lyginimo kriterijai yra:
  a) algoritmo atminties sąnaudos, nes tai riboja modelio duomenų apimtį (dalelių kiekį)
  b) tikslumas, nes tai parodo ar algoritmo rezultatas yra naudingas.
- Atlikus eksperimentą buvo prieitos išvados:

  Vidutiniškai GBZ realizacija yra sparčiausia, o YB yra lėčiausia. BP realizacija mažai atsilieka nuo GBZ.

  Vidutiniškai YB realizacija yra tiksliausia, tuomet eina BP, o GZB realizacijos tikslumas žemiausias.

  Esant dviem skirtingų dydžių klasteriams, mažesniojo klasterio dalelės turės didesnę santykinę paklaidą.

  BP ir GBZ realizacijose pastebėtos atidarymo kampo θ ribos, kurios apriboja maksimalią spartą, tačiau neapriboja tikslumo.

6. LITERATŪRA

[1] Josh Barnes ir Piet Hut. „A hierarchical O(N log N) force-calculation algorithm“. Leidinyje: *NATURE* (1986 m. gruodžio mėn.).

[2] Joshua E. Barnes. „A Modified Tree Code: Don't Laugh; It Runs“. Leidinyje: *Journal of Computational Physics* ().

[3] Jeroen Bédorf, Evghenii Gaburov ir Simon Portegies Zwart. „A sparse octree gravitational N-body code that runs entirely on the GPU processor“. Leidinyje: *Journal of Computational Physics* (2012 m. bal mėn.). arXiv: `1106.1900v2 [astro-ph.IM]`.

[4] Ian Buck ir kt. „Brook for GPUs: Stream Computing on Graphics Hardware“. Leidinyje: *ACM Transactions on Graphics (TOG)* (2004 m. rugpj. mėn.).

[5] Martin Burtscher ir Keshav Pingali. „An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm“. Leidinyje: *GPU Computing Gems*. Hrsg. von Wen mei W. Hwu. 2011 m. vas mėn.

[6] Evghenii Gaburov, Jeroen Bedorf ir Simon Portegies Zwart. „Gravitational tree-code on graphics processing units: implementation in CUDA“. Leidinyje: *Procedia Computer Science*. 2010 m. geg mėn. arXiv: `1005.5384v1 [astro-ph.IM]`.

[7] L. Greengard ir V. Rokhlin. „A fast algorithm for particle simulations“. Leidinyje: *Journal of computational physics* (1987 m.).

[8] Lars Hernquist. „Performance Characteristics of Tree Codes“. Leidinyje: *The Astrophysical Journal* ().

[9] Lars Nyland, Mark Harris ir Jan Prins. „Fast N-Body Simulation with CUDA“. Leidinyje: *GPU Gems 3*. 2007 m.

[10] *The open standard for parallel programming of heterogeneous systems*. URL: `https://www.khronos.org/opencl/`.

[11] Rio Yokota ir Lorena A. Barba. „Treecode and fast multipole method for N-body simulation with CUDA“. Leidinyje: *GPU Computing Gems*. Hrsg. von Wen mei W. Hwu. 2011 m. vas mėn. arXiv: `1010.1482v1 [physics.comp-ph]`.

7. PRIEDAS

7.1. EKSPERIMENTO REZULTATŲ LENTELĖS

Čia yra pateikiami eksperimento, kuris buvo aptartas 4.3. skyriuje, rezultatų duomenys.

7.1 lentelė. Homogeninės erdvės apdorojimo trukmė sekundėmis

Dalelių kiekis	HN	CN	BP	GBZ	YB
1000	0,013427467	0,002746827	0,00186677	0,003020173	0,018256667
2000	0,053189667	0,002800157	0,002743493	0,0035302	0,018396667
4000	0,212345333	0,003060173	0,00416024	0,005867007	0,02422
8000	0,848001	0,005006953	0,010467267	0,0106206	0,03252
16000	3,39359	0,0117207	0,022294633	0,013454133	0,049106667
32000	13,56686667	0,038268867	0,048749467	0,029001667	0,085106667
64000	54,24625	0,139924667	0,109506	0,072854167	0,168866667
128000	-	0,546131	0,257415	0,130507333	0,679366667
256000	-	2,17403	0,569556667	0,289112	1,363666667
512000	-	8,6639	1,260403333	0,728308667	2,888733333
1024000	-	34,6306	2,94157	1,337813333	9,1966

7.2 lentelė. Homogeninės erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F\left(>\frac{\Delta a}{a}\right)$

Santykinė paklaida $\frac{\Delta a}{a}$	BP	GBZ	YB
0,000001	0,999997	1,000000	0,767492
0,000002	0,999976	0,999996	0,342452
0,000004	0,999774	0,999982	0,053302
0,000008	0,998250	0,999868	0,007251
0,000016	0,986607	0,998919	0,001376
0,000032	0,912039	0,991405	0,000468
0,000064	0,618424	0,939277	0,000166
0,000128	0,187073	0,658411	0,000051
0,000256	0,029169	0,172864	0,000016
0,000512	0,003370	0,030479	0,000003
0,001024	0,000362	0,004535	0
0,002048	0,000036	0,000893	0
0,004096	0,000005	0,000080	0
0,008192	0	0,000007	0
0,016384	0	0	0

7.3 lentelė. BP realizacijos homogeninės erdvės apdorojimo sparta (sekundėmis) esant skirtingoms θ reikšmėms

Dalelių kiekis	$\theta = 0,2$	$\theta = 0,3$	$\theta = 0,7$	$\theta = 0,9$	$\theta = 1,0$
1000	0,00208012	0,00207012	0,00165009	0,00144	0,00148009
2000	0,00368021	0,00338019	0,00219012	0,00185	0,00186011
4000	0,00684039	0,00578034	0,00300018	0,00234	0,00236014
8000	0,0234413	0,017481	0,00666039	0,00486	0,00496029
16000	0,0726041	0,0456426	0,0132008	0,009121	0,00932054
32000	0,195691	0,111006	0,0280416	0,018481	0,0185611
64000	0,52008	0,270815	0,0602534	0,039402	0,0390522
128000	1,41478	0,682439	0,135508	0,086205	0,0844048
256000	3,52163	1,59609	0,293302	0,179153	0,178867
512000	8,51889	3,70621	0,633236	0,378222	0,378222
1024000	-	8,7841	1,44068	0,87225	0,851849

7.4 lentelė. BP realizacijos homogeninės erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F(>\frac{\Delta a}{a})$ esant skirtingoms θ reikšmėms

Santykinė paklaida $\frac{\Delta a}{a}$	$\theta = 0,2$	$\theta = 0,6$	$\theta = 1,0$
0,000001	0,998354	1	1
0,000002	0,987043	1	1
0,000004	0,913199	0,999957	1
0,000008	0,607451	0,99968	0,999992
0,000016	0,188732	0,997707	0,999945
0,000032	0,03376	0,982656	0,999633
0,000064	0,004072	0,891641	0,996799
0,000128	0,000488	0,571111	0,977053
0,000256	0,00008	0,167076	0,863682
0,000512	0,000018	0,024707	0,505529
0,001024	0	0,0029	0,125992
0,002048	0	0,000266	0,016184
0,004096	0	0,000033	0,002
0,008192	0	0,000004	0,000236
0,016384	0	0	0,000029
0,032768	0	0	0,000004
0,065536	0	0	0,000002
0,131072	0	0	0

7.5 lentelė. GBZ realizacijos homogeninės erdvės apdorojimo sparta (sekundėmis) esant skirtingoms θ reikšmėms

Dalelių kiekis	$\theta = 0,2$	$\theta = 0,4$	$\theta = 1,0$
1000	0,00394023	0,00311018	0,00241014
2000	0,00457026	0,00374022	0,00304017
4000	0,0101206	0,0063204	0,00382022
8000	0,0204012	0,0121007	0,00732041
16000	0,0337219	0,0162809	0,00948054
32000	0,0668038	0,0334019	0,0215612
64000	0,204412	0,086755	0,052753
128000	0,463427	0,164409	0,087505
256000	0,869335	0,345448	0,212155
512000	2,72916	0,904852	0,504829
1024000	-	1,7449	0,860249

7.6 lentelė. GBZ realizacijos homogeninės erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F(>\frac{\Delta a}{a})$ esant skirtingoms θ reikšmėms

Santykinė paklaida $\frac{\Delta a}{a}$	$\theta = 0,2$	$\theta = 0,6$	$\theta = 1,0$
0,000001	0,992273	1	1
0,000002	0,945604	1	1
0,000004	0,718777	0,99999	1
0,000008	0,266141	0,999932	1
0,000016	0,051775	0,99949	0,999996
0,000032	0,008787	0,996246	0,999994
0,000064	0,001363	0,974709	0,999988
0,000128	0,000123	0,875182	0,999969
0,000256	0,00002	0,586363	0,99984
0,000512	0,000008	0,211891	0,998986
0,001024	0	0,036012	0,99252
0,002048	0	0,004105	0,954711
0,004096	0	0,000623	0,774629
0,008192	0	0,000072	0,310197
0,016384	0	0,000021	0,070242
0,032768	0	0,000002	0,022012
0,065536	0	0	0,002244
0,131072	0	0	0,000051
0,262144	0	0	0,000002
0,524288	0	0	0

7.7 lentelė. Vieno klasterio erdvės apdorojimo trukmė sekundėmis

Dalelių kiekis	HN	CN	BP	GBZ	YB
1000	0,0133608	0,002666817	0,002166793	0,003960227	0,1083
2000	0,053469733	0,002776827	0,003536867	0,006447033	0,132806667
4000	0,212478667	0,003106847	0,006053683	0,010947267	0,19076
8000	0,848906	0,00501362	0,0152009	0,018647767	0,32568
16000	3,390056667	0,011574033	0,036988767	0,035308667	0,574231
32000	13,56633333	0,0383622	0,093712033	0,067363833	0,838794667
64000	-	0,140341333	0,224463	0,112673	2,072953333
128000	-	0,547731333	0,533497	0,210045333	4,746193333
256000	-	2,17822	1,21126	0,458407333	7,929
512000	-	8,669766667	2,75636	0,973522667	19,39556667
1024000	-	34,74426667	6,235156667	2,01058	

7.8 lentelė. Vieno klasterio erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F\left(>\frac{\Delta a}{a}\right)$

Santykinė paklaida $\frac{\Delta a}{a}$	BP	GBZ	YB
0,000001	0,999984	0,999996	0,997723
0,000002	0,999887	0,999982	0,985232
0,000004	0,999104	0,99984	0,924975
0,000008	0,993221	0,99918	0,711748
0,000016	0,953242	0,996104	0,308943
0,000032	0,778625	0,98217	0,044164
0,000064	0,428266	0,901742	0,003422
0,000128	0,147438	0,570145	0,001326
0,000256	0,033613	0,136412	0,00052
0,000512	0,005043	0,015725	0,000115
0,001024	0,00075	0,001045	0,000016
0,002048	0,00015	0,000121	0
0,004096	0,00002	0,000018	0
0,008192	0,000004	0,000004	0
0,016384	0	0	0

7.9 lentelė. BP realizacijos vieno klasterio erdvės apdorojimo sparta (sekundėmis) esant skirtingoms θ reikšmėms

Dalelių kiekis	$\theta = 0,2$	$\theta = 0,3$	$\theta = 0,6$	$\theta = 0,9$	$\theta = 1,0$
1000	0.00217012	0.00227013	0.00213012	0.00186011	0.0017101
2000	0.00393022	0.00376022	0.00335019	0.00271016	0.00259015
4000	0.00738042	0.0070004	0.00560033	0.00390023	0.00358021
8000	0.0254015	0.0224413	0.0122407	0.00748044	0.00662038
16000	0.0846848	0.0641237	0.0285216	0.0148809	0.0128407
32000	0.280176	0.193251	0.0680039	0.0336419	0.0280016
64000	0.849799	0.51783	0.156709	0.0716541	0.0595034
128000	2.51144	1.36888	0.368821	0.157109	0.128107
256000	6.94354	3.42777	0.805903	0.329733	0.269015
512000	-	8.40768	1.7927	0.70744	0.574433
1024000	-	-	3.97523	1.52569	1.21527

7.10 lentelė. BP realizacijos vieno klasterio erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F(>\frac{\Delta a}{a})$ esant skirtingoms θ reikšmėms

Santykinė paklaida $\frac{\Delta a}{a}$	$\theta = 0,3$	$\theta = 0,6$	$\theta = 1,0$
0,000001	0,999689	0,999998	1
0,000002	0,997893	0,999977	1
0,000004	0,984437	0,999773	0,999998
0,000008	0,905367	0,99823	0,999969
0,000016	0,626521	0,986857	0,999836
0,000032	0,238391	0,91801	0,999102
0,000064	0,044506	0,677154	0,993408
0,000128	0,004982	0,318689	0,954564
0,000256	0,00073	0,096162	0,763934
0,000512	0,000156	0,019674	0,369307
0,001024	0,000025	0,002904	0,099494
0,002048	0,000004	0,000389	0,017172
0,004096	0	0,000055	0,001666
0,008192	0	0,000008	0,000221
0,016384	0	0	0,000031
0,032768	0	0	0,000002
0,065536	0	0	0

7.11 lentelė. GBZ realizacijos vieno klasterio erdvės apdorojimo sparta (sekundėmis) esant skirtingoms θ reikšmėms

Dalelių kiekis	$\theta = 0,2$	$\theta = 0,3$	$\theta = 0,6$	$\theta = 0,9$	$\theta = 1,0$
1000	0,00386022	0,00378022	0,00366021	0,00331019	0,00318018
2000	0,0068704	0,00685039	0,00619035	0,00580033	0,00585034
4000	0,0132808	0,0122607	0,0101006	0,00892051	0,00902051
8000	0,0270816	0,0236814	0,017581	0,0164809	0,0162209
16000	0,0604435	0,0454426	0,0330419	0,0299217	0,0294417
32000	0,134568	0,0918053	0,0616435	0,0537631	0,052923
64000	0,333069	0,193511	0,0970056	0,0764044	0,0748043
128000	0,858149	0,438225	0,17391	0,124207	0,119007
256000	2,17598	1,01649	0,381736	0,262015	0,255015
512000	-	2,35073	0,799446	0,546431	0,53063
1024000	-	-	1,64129	1,12986	1,09266

7.12 lentelė. GBZ realizacijos vieno klasterio erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F(>\frac{\Delta a}{a})$ esant skirtingoms θ reikšmėms

Santykinė paklaida $\frac{\Delta a}{a}$	$\theta = 0,3$	$\theta = 0,6$	$\theta = 1,0$
0,000001	0,999871	0,999998	0,999998
0,000002	0,998912	0,999992	0,999992
0,000004	0,992211	0,999922	0,999959
0,000008	0,945385	0,999619	0,999715
0,000016	0,720012	0,998355	0,998889
0,000032	0,265516	0,994982	0,997615
0,000064	0,035457	0,982225	0,99584
0,000128	0,003258	0,915047	0,993174
0,000256	0,000273	0,627908	0,986908
0,000512	0,000018	0,166971	0,966865
0,001024	0,000004	0,014984	0,904145
0,002048	0	0,000986	0,717012
0,004096	0	0,000121	0,364014
0,008192	0	0,00002	0,075354
0,016384	0	0,000002	0,005684
0,032768	0	0	0,000453
0,065536	0	0	0,000057
0,131072	0	0	0,000006
0,262144	0	0	0,000002
0,524288	0	0	0

7.13 lentelė. Dvigubo klasterio erdvės apdorojimo trukmė sekundėmis

Dalelių kiekis	HN	CN	BP	GBZ	YB
1000	0,013454133	0,002726827	0,001876773	0,00301017	0,09457
2000	0,053043033	0,002716823	0,002576817	0,00429358	0,11218
4000	0,211812	0,002966837	0,00432692	0,008013793	0,135426667
8000	0,846381667	0,00499362	0,010693933	0,0152009	0,222373333
16000	3,388523333	0,011654033	0,0272149	0,0302817	0,336077
32000	13,55343333	0,038255533	0,072137433	0,0589634	0,654077
64000	-	0,140174667	0,179943333	0,128340333	1,39986
128000	-	0,546497667	0,446726	0,279182667	2,982143333
256000	-	2,173026667	1,057726667	0,598367667	6,521933333
512000	-	8,660766667	2,463473333	1,036393333	15,10533333
1024000	-	34,63216667	5,795196667	2,141453333	49,7815

7.14 lentelė. Dvigubo klasterio erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F\left(>\frac{\Delta a}{a}\right)$

Santykinė paklaida $\frac{\Delta a}{a}$	BP	GBZ	YB
0,000001	0,999996	0,999998	0,919631
0,000002	0,999963	0,999982	0,745529
0,000004	0,999662	0,999828	0,527459
0,000008	0,996932	0,999201	0,419986
0,000016	0,978043	0,996799	0,405672
0,000032	0,891646	0,98751	0,399971
0,000064	0,691684	0,93376	0,399854
0,000128	0,496068	0,714883	0,397816
0,000256	0,297844	0,354992	0,216199
0,000512	0,10198	0,109834	0,092346
0,001024	0,043902	0,045996	0,043121
0,002048	0,019723	0,020053	0,019707
0,004096	0,007623	0,007637	0,007609
0,008192	0,001078	0,001078	0,00107
0,016384	0	0	0

7.15 lentelė. BP realizacijos dvigubo klasterio erdvės apdorojimo sparta (sekundėmis) esant skirtingoms θ reikšmėms

Dalelių kiekis	$\theta = 0,2$	$\theta = 0,3$	$\theta = 0,6$	$\theta = 0,9$	$\theta = 1,0$
1000	0,00226013	0,00212013	0,00167009	0,00154009	0,0015901
2000	0,00323018	0,00283016	0,00248014	0,00224013	0,00213012
4000	0,00490028	0,00458027	0,00402022	0,0033802	0,00298017
8000	0,0138408	0,0130607	0,00932054	0,00650038	0,00596035
16000	0,0454426	0,0400823	0,0222813	0,0130008	0,0115607
32000	0,16641	0,127007	0,0552832	0,0291617	0,0250814
64000	0,541431	0,365971	0,131858	0,0644037	0,0543031
128000	1,6953	1,02846	0,314618	0,142808	0,119307
256000	5,15758	2,76787	0,741471	0,313589	0,257729
512000	14,0352	7,0216	1,65329	0,687839	0,556432
1024000	-	-	3,67261	1,45688	1,17567

7.16 lentelė. BP realizacijos dvigubo klasterio erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F(>\frac{\Delta a}{a})$ esant skirtingoms θ reikšmėms

Santykinė paklaida $\frac{\Delta a}{a}$	$\theta = 0,2$	$\theta = 0,3$	$\theta = 0,4$	$\theta = 0,6$	$\theta = 1,0$
0,000001	0,999148	0,999846	0,999969	0,999996	1
0,000002	0,99352	0,998668	0,999748	0,999986	1
0,000004	0,957918	0,989812	0,997973	0,999898	0,999998
0,000008	0,80999	0,938408	0,985232	0,999219	0,99999
0,000016	0,540621	0,764811	0,92127	0,994328	0,999904
0,000032	0,414305	0,538457	0,741082	0,963232	0,999334
0,000064	0,40099	0,42643	0,532096	0,839701	0,995059
0,000128	0,398139	0,402119	0,42401	0,617654	0,967379
0,000256	0,216652	0,224965	0,252264	0,374287	0,843713
0,000512	0,092475	0,092469	0,094129	0,126107	0,520629
0,001024	0,043123	0,043193	0,043258	0,04783	0,170354
0,002048	0,019707	0,019705	0,019754	0,02	0,048248
0,004096	0,007607	0,007609	0,007613	0,007707	0,011166
0,008192	0,001072	0,001066	0,00108	0,00107	0,001576
0,016384	0	0	0	0	0,000061
0,032768	0	0	0	0	0,000006
0,065536	0	0	0	0	0

7.17 lentelė. GBZ realizacijos dvigubo klasterio erdvės apdorojimo sparta (sekundėmis) esant skirtingoms θ reikšmėms

Dalelių kiekis	$\theta = 0,2$	$\theta = 0,3$	$\theta = 0,6$	$\theta = 0,9$	$\theta = 1,0$
1000	0,0034702	0,0035402	0,00303017	0,00267015	0,00273016
2000	0,00457025	0,00435025	0,00410024	0,00383022	0,00404023
4000	0,00938054	0,00856049	0,00792045	0,00732042	0,00724042
8000	0,0203612	0,0183611	0,0144408	0,0131608	0,0130607
16000	0,0458826	0,0379622	0,0290417	0,0254015	0,0254414
32000	0,104846	0,0774844	0,0556032	0,0484028	0,0484428
64000	0,280166	0,189261	0,117707	0,0995057	0,0982556
128000	0,729442	0,450126	0,250614	0,206012	0,201912
256000	1,9594	1,08378	0,524459	0,42231	0,411166
512000	-	2,18472	0,899451	0,687039	0,668638
1024000	-	-	1,8263	1,38688	1,33468

7.18 lentelė. GBZ realizacijos dvigubo klasterio erdvės apdorojimo rezultato sąlyginė paklaidos paskirstymas $F(>\frac{\Delta a}{a})$ esant skirtingoms θ reikšmėms

Santykinė paklaida $\frac{\Delta a}{a}$	$\theta = 0,3$	$\theta = 0,4$	$\theta = 0,7$	$\theta = 1,0$
0,000001	0,999904	0,99999	1	1
0,000002	0,999324	0,999938	0,999986	0,999996
0,000004	0,994836	0,999477	0,99993	0,999957
0,000008	0,965643	0,996535	0,999645	0,999811
0,000016	0,825154	0,980066	0,999047	0,999482
0,000032	0,542008	0,899199	0,997682	0,998979
0,000064	0,41766	0,662416	0,994318	0,997797
0,000128	0,399633	0,445771	0,983887	0,995072
0,000256	0,218891	0,249762	0,930793	0,989355
0,000512	0,092277	0,094836	0,68542	0,971002
0,001024	0,043076	0,043422	0,235867	0,896664
0,002048	0,019744	0,019695	0,040953	0,67066
0,004096	0,007607	0,007615	0,009699	0,305207
0,008192	0,001076	0,001076	0,001354	0,048643
0,016384	0	0	0,000037	0,002385
0,032768	0	0	0,000006	0,000207
0,065536	0	0	0	0,000037
0,131072	0	0	0	0,000002
0,262144	0	0	0	0
0,524288	0	0	0	0
1,048576	0	0	0	0