KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Algirdas Žukauskas, Nerijus Lukšys

Tarpląstelinio sąveikavimo imitacinio modelio optimizavimas ir tyrimas

Magistro darbas

Darbo vadovas

Prof. habil. dr. H. Pranevičius

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Algirdas Žukauskas, Nerijus Lukšys

Tarpląstelinio sąveikavimo imitacinio modelio optimizavimas ir tyrimas

Magistro darbas

Recenzentas

2010-05-24

Darbo vadovas

Prof. habil. dr. H. Pranevičius
2010-05-24

Atliko

IFM-4/1 gr. stud.
Algirdas Žukauskas
Nerijus Lukšys
2010-05-24

Kaunas, 2010

Santrauka

Darbe yra nagrinėjama tarpląstelinių plyšinių jungčių modeliavimo sistema (GJM – Gap Juncion Model, sukurta N. Paulausko - Kauno technologijos universiteto, informatikos fakulteto magistrantas). Pagrindinis tyrimo objektas yra optimizavimo uždavinys. Viename iš sistemos modulių, pasinaudojant globalaus optimizavimo metodais, yra ieškoma modelio parametrų, su kuriais modelio ir realių eksperimentų (gautų iš sistemos užsakovų Niujorko, Yeshiva universiteto, Alberto Einšteino medicinos koledžo laboratorijos (prof. Felikso Bukausko)) rezultatai skirtųsi minimaliai arba sutaptų. Šiuo atveju yra modeliuojama tarpląstelinių plyšinių jungčių laidumo priklausomybė nuo įtampos t.y imituojami elektro-fiziologiniai procesai.

Optimizuojant modelio parametrų paiešką, darbe buvo surinkti rezultatai (atliko Nerijus Lukšys), modeliuojant su skirtingu skaičiumi iteracijų bei parenkant skirtingus optimizavimo metodus, realizuotus kitoje sistemoje. Buvo konsultuojamasi su akademiku habil. dr. prof. Jonu Mockumi. Iš gautų rezultatų įvertinamos metodų silpnosios ir stipriosios pusės bei jų pritaikymas Grid sistemai.

Darbo tikslas, remiantis analizės rezultatais, realizuoti modelio parametrų paiešką Grid sistemoje (atliko Algirdas Žukauskas), panaudojant kelis optimizavimo metodus bei palyginti gautus rezultatus.

Summary

In this research we analyze optimization of cells gap junctions simulation model (GJM – Gap Junction Model, designed by N. Paulauskas – Kaunas University of Technology student). The main object of study is the optimization problem. In one module of the system, using global optimization methods, is looking for the model parameters to which the model and real experiments (obtained from the New York, Yeshiva University, Albert Einstein College of Medicine Laboratory (prof. Felikso Bukausko)) results differ minimally or overlap. In this case, we are simulating cells gap junction voltage dependence of conductivity (simulating electro-physiological processes).

Optimizing the model parameters of the search were collected at the results of simulation with different number of iterations (solved by Nerijus Lukšys), and the choice of different optimization methods, realized in another system. There was consulting with academic habil. dr. prof. Jonas Mockus. The results showed methods weak and strong points and their application in Grid systems.

The purpose of this problem was to implement gap junction simulation model on the Grid system (solved by Algirdas Žukauskas). Use several optimization methods and compare the results.

Terminų ir santrumpų žodynėlis

GJM – (angl. Gap Juncion Model) Tarpląstelinio sąveikavimo modelis;

GRID – tinkle sujungti kompiuteriai, naudojami moksliniams bei techniniams tyrimams atlikti;

MPI – (angl. Message Passing Interface) – tai komunikacinių paprogramių bibliotekos (MPL) standartas.

JDL – (angl. Job Description language) – tai specialiai skirta GRID sistemoms sisteminė kalba skirta aprašyti skaičiavimus, darbo stotis ir kitus resursus;

PE – procesoriaus elementas;

AE – atminties elementas;

VO - Virtuali organizacija. Tai vartotojų grupė, kuriai suteikta teisė leisti darbus tam tikrame Grid resursų rinkinyje

Turinys

1. Įvadas

Tarpląstelinės plyšinės jungtys (jungtys tarp neuronų) yra mažai ištirtos, yra daromos įvairios prielaidos, bei kuriami teoriniai matematiniai modeliai, kurie paaiškintų ląstelių kontaktavimą plyšinėmis jungtimis. Šiame darbe nagrinėjamos GJM sistemos vienas iš modulių – parametrų paieška t.y. modeliuojama laidumo, plyšinėse jungtyse, priklausomybė nuo įtampos. Jų radimas pagrįstas globalaus optimizavimo metodais, kuriems užduodamos parametrų galimos kitimo ribos. Tiriama (modeliuojama) tarpląstelinių plyšinių jungčių laidumo priklausomybė nuo įtampos. Gauti duomenys lyginami su realiais eksperimentų rezultatais. Tikimasi rasti atitikmenis.

Sistemoje modeliuojant elektro-fiziologinius procesus yra ieškoma vidutinė kvadratinė paklaida tarp modelio ir eksperimentų rezultatų. Šiuo metu, kad būtų gauta kuo mažesnė paklaida reikalingas didelis skaičius iteracijų, o tai įtakoja modelio parametrų paieškos greitį. Tad, šioje sistemoje, skaičiavimų greitaeigiškumo klausimas tampa aktualus.

Šiuolaikiniai kompiuteriai yra pajėgūs per pakankamai trumpą laiką išspręsti daugelį uždavinių, tačiau šio atveju, gali užrukti valandas ar net paras.

Šiame darbe nagrinėjami globalaus optimizavimo metodai, lygiagrečių skaičiavimų bibliotekos, jų architektūros, privalumai ir trūkumai, nustatomos potencialios modelio programinio kodo lygiagretinimo vietos.

Darbo tikslas yra GJM programos parametrų paieškos modelio greitaeigiškumo optimizavimas.

2. Tarpląstelinių plyšinių jungčių modelio analizė

2.1 Tikslai ir siekiami rezultatai

Modeliuojant tam tikrų objektų elgsenas kompiuteryje yra siekiama, kad modelis atitiktų realaus pasaulio objekto elgsenas. Modelis palengvina ir pagreitina įvairius skaičiavimus susijusius su realiu objektu. Tokiu būdu galima lengvai ir greitai numatyti ar įvertinti kaip realaus pasaulio objektas reaguos į vienokias ar kitokias sąlygas (pakeitimus). Pvz. sumodeliuotoje stogo konstrukcijoje (įvertinus fizikines medžiagų savybes) galima būtų lengvai patikrinti ar stogo konstrukcija atlaikys tam tikrą svori ar geriau keisti stogo dangą į lengvesnę ir pan. Aišku tai, kad realiai tikrinti ar stogas atlaikys ar ne, būtų išnaudoti dideli laiko bei kaštų resursai, nekalbant jau ir apie riziką.

Imitacija, sukurta kompiuterio, tapo naudinga ne tik statybos srityje, bet ir chemijoje, fizikoje, biologijoje ir kitose mokslo srityse. Imitacinis modeliavimas ne išimtimi tapo ir tarpląstelinėje

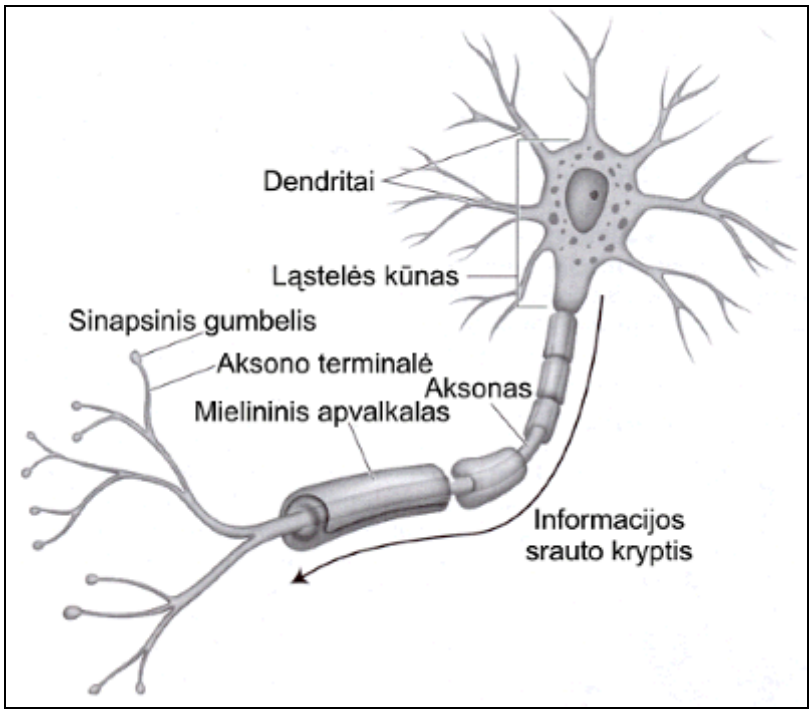
plyšinėje jungtyje. Yra daromos įvairios prielaidos, bei kuriami teoriniai matematiniai modeliai, kurie paaiškintų ląstelių kontaktavimą plyšinėmis jungtimis.

Šios analizės tikslas išsiaiškinti GJM imitacinėje sistemoje sukurto globalaus parametrų optimizavimo modelio veikimą. Išnagrinėti kitus metodus, kuriuos būtų galima pritaikyti nagrinėjamam uždaviniui spręsti, taip pat išnagrinėti modelio pritaikomumą lygiagrečioms sistemoms. Palyginti gautus rezultatus.

2.2 Neurono sandara

Tiriamasis objektas, šiuo atveju – neuronas, yra sudėtingas biologinis darinys, ties kurio dirba daugybę mokslininkų, bandydami išsiaiškinti jame vykstančius procesus, kažkokiais tiksliais apskaičiavimais, formulėmis, patirtimi, stebėjimais, bandymais ar galu gale spėjimais.

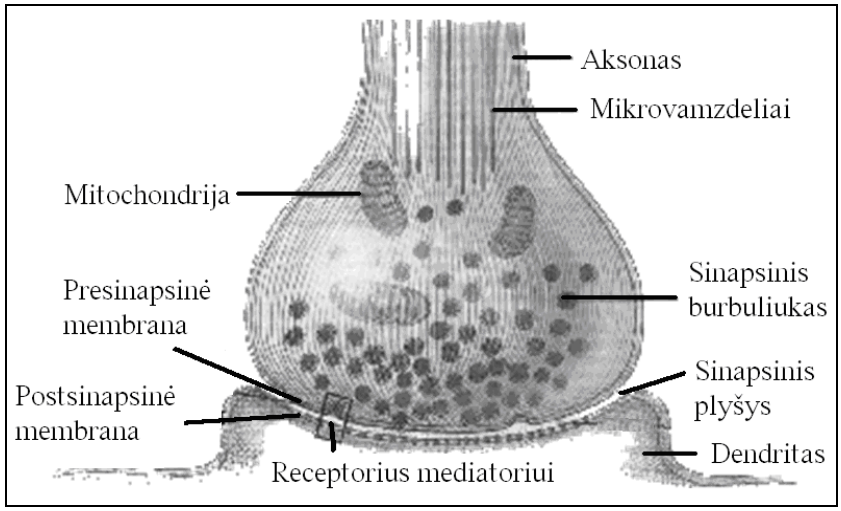
Nervų sistemą sudaro nervinės ląstelės – neuronai. Jie turi kūną ir ataugas. Įprastas neuronas pavaizduotas (žr. 2.2-1 pav). Trumpesnės, nervinius impulsus perduodančios į neurono kūną, ataugos – *dendritai*, ilgesnės – *aksonai*, arba *neuritai*, perduodantys impulsą iš neurono kūno. Jungtis tarp dviejų neuronų – *sinapsė*. Dažniausios sinapsės būna tarp vieno neurono aksono ir kito neurono dendrito, bet egzistuoja ir kitokio tipo sinapsių.



2.2-1 pav. Biologinio neurono sandara

2.3 Nervinis impulsas ir jo sklidimas

Tais pačiais nervais impulsai vienu metu gali būti perduoti net keliomis kryptimis. Juntamųjų arba receptorinių bei judinamųjų (motorinių) neuronų veikla yra darni, koordinuota ir sukelia organizmo atsakomąją reakciją į dirgiklį automatiškai, pavyzdžiui, prie karšto paviršiaus pridėta ranka tuojau atitraukiama, net nespėjus to suvokti. Kontakto vietose neuronų ląstelių membranos nesusilieja, tarp jų visada yra nedidelis tarpas – sinapsinis plyšys (2.3-1 pav.):



2.3-1 pav. Sinapsinis perdavimas

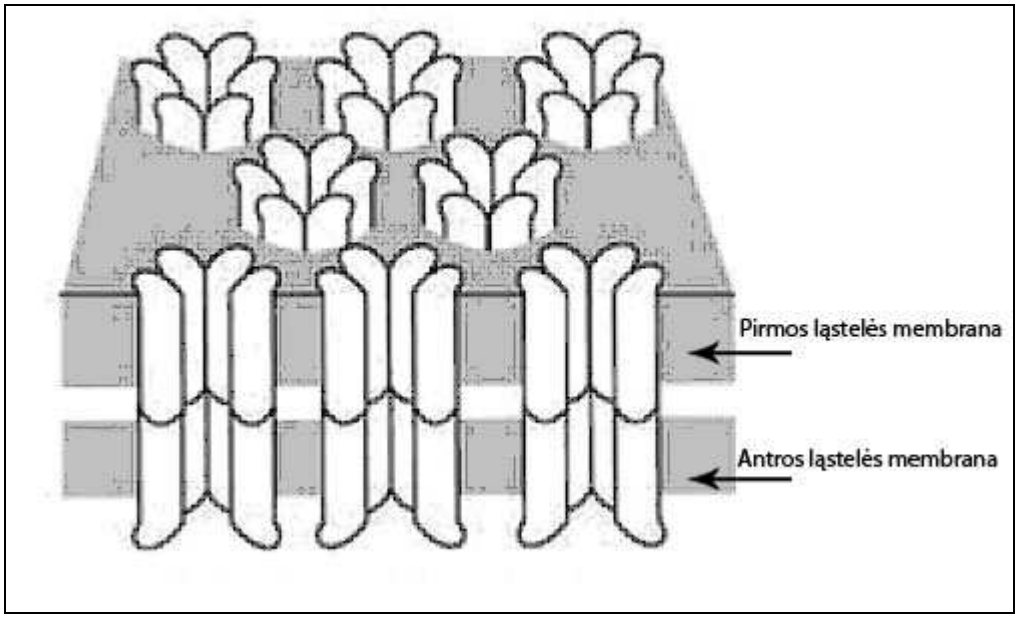
Sinapsinio nervinio impulso sklidimo etapai:

- Nervinis impulsas pasiekia presinapsinę skaidulos dalį.
- Atsidaro kalcio kanalai, Ca++ patenka į presinapsinę citoplazmą.
- Presinapsinėje citoplazmoje pastoviai yra didelis kiekis sinapsinių burbuliukų nešančių neiromediatorių. Neiromediatorius uždaro grandinę, vykdydamas cheminį informacijos perdavimą per sinapsinį plyšį. Dėl patekusių Ca++ jonų, presinapsinės citoplazmos burbuliukai sprogsta, išpurškiant mediatorių į sinapsinį plyšį.
- Patekęs į plyšį mediatorius difunduoja į postsinapsinę plyšio pusę.
- Receptoriai, esantys postsinapsinėje plyšio pusėje pagauna mediatoriaus molekules.
- Padidėja kanalo laidumas Na+ ir K+ jonams abejomis kryptimis.

Kanalų atidarymas postsinapsinėje plyšio pusėje, iššaukia natrio jonų srautą į vidų, o kalio – į išorę. Atsiradusi joninė srovė sužadina nervinį impulsą, kuris sklinda toliau.

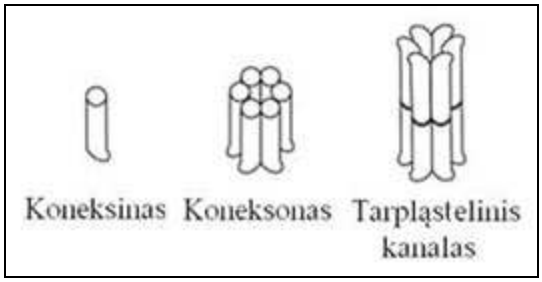
2.4 Tarpląstelinės plyšinės jungtys

Koneksinai (Cx) – tai baltymai, kurie formuoja tarpląstelines plyšines jungtis kontakte tarp kaimyninių ląstelių membranų (žr. 2.4-1 pav.). Koneksinas yra pavaizduotas 2.4-2. pav. Plyšinės jungtys leidžia sujungtoms ląstelėms perduoti elektrinius ir metabolinius signalus.



2.4-1 pav. Susidariusios tarpląstelinės plyšinės jungtys

Kiekvienas plyšinės jungties kanalas (žr. 2.4-2 pav. - Tarpląstelinis kanalas) sudarytas iš dviejų puskanalių (žr. 2.4-2 pav. - Koneksonas) kurie savo ruožtu yra sudaryti iš šešių Cx koneksinų. Tarpląstelinėje sąveikoje dalyvaujančios plyšinės jungtys gali būti homotipinės (ląstelės išreiškia tą patį Cx izotipą), heterotipinės (ląstelės išreiškia skirtingus Cx izotipus) ir heteromerinės (bent viena ląstelė išreiškia du ar daugiau Cx izotipus). PJ kanalai skiriasi elektriniu laidumu, selektyviu pralaidumu, įvairiomis cheminėmis medžiagomis ir kanalų vartų jautrumu įtampai.

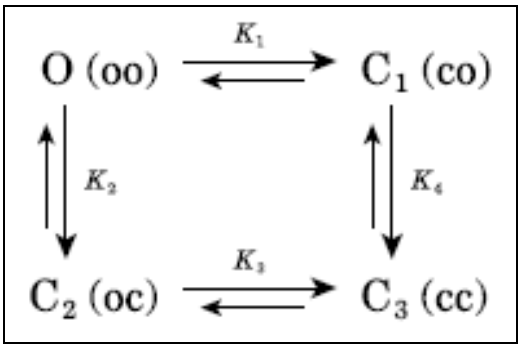


2.4-2 pav. Susidariusios tarpląstelinės plyšinės jungtys

Koneksinai vadinami tapačiais, jeigu jie gali sudaryti funkcinius heterotipinius plyšinės jungties kanalus. Dauguma Cx porų gali sudaryti funkcines plyšines jungtis, bet egzistuoja kai

kurios išimtys. Pavyzdžiui: du pagrindiniai širdies koneksonai, Cx40 ir Cx43, kurie yra ekspresuoti širdyje ir kraujagyslėse, nėra tapatūs ir negali sudaryti funkcinių plyšinies jungties kanalų

Bendra plyšinės jungties savybė būdinga visiems Cx koneksinams yra kanalo vartų būsenos priklausomybė nuo įtampos. Jungties laidumas mažėja sudarius tarpląstelinės plyšinės jungties potencialą, V_j. Manoma, kad kiekvienas puskanalis turi du skirtingus vartų valdymo mechanizmus: lėtus, kuris uždaro PJ pilnai ir greitus, kuris perveda vartus į pereinamą būsena ar būsenas su liekamuoju laidumu. Laikome, kad kiekvienas puskanalis gali būti dviejose būsenose: atviroje su laidumu G_o ir uždarytoje su laidumu G_c. Vartų valdymo mechanizmai gali skirtis poliariškumu, tai yra vartai gali užsidaryti arba atsidaryti, jeigu citoplazmos pusėje potencialas didėja arba mažėja. Jeigu puskanalių vartai turi priešingą poliariškumą, vienas V_j poliariškumas atidarys abu puskanalius, o priešingas V_j potencialas uždarys abu puskanalius. 2.4-3 pav. pavaizduota pagrindinė schema, kuria buvo sudaromos perėjimo tikimybės iš [5] literatūros straipsnio:



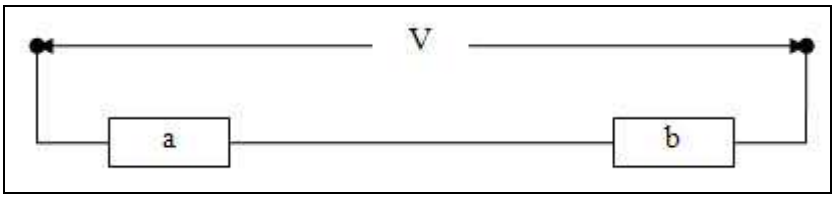
2.4-3 pav. Kanalo būsenų modelio schema

Čia K_i *(i = 1, 2, 3, 4)* yra perėjimo procesų proporcingumo konstantos. Kanalas gali būti vienoje iš keturių galimų būsenų: 1) *O(oo),* kurioje abu kanalo galai yra atidaryti, 2) *C_1(co),* kurioje kanalo galas *L* yra uždarytas, o *R* - atidarytas, 3) *C_2(oc),* kurioje kanalo galas *L* yra atidarytas, o *R* – uždarytas, 4) *C_3(cc),* kurioje abu kanalo galai yra uždaryti.

2.5 Dviejų vartų modelis

Žemiau pateiktas dviejų vartų modelis, kurio pagrindu buvo realizuota sistema. Nagrinėsime tik šį modelį, nes sistemoje parametrų paieška yra realizuota tik dviejų vartų modeliui.

Šis modelis sudarytas iš dviejų nuosekliai sujungtų vartų a ir b (2.5-1 pav.), kurių kiekvienas turi savo kintantį laidumą. Kiekvienas vartas gali būti atviroje arba uždaroje būsenoje.



2.5-1 pav. 2 vartų modelio struktūra

Kiekvienas kanalo vartas apibūdinamas šiais parametrais: κ_j- perėjimo konstanta, reguliuojanti varto būsenų kitimą, A_j - įtampos jautrumo koeficientas, V_{0j} - įtampos koeficientas, $G_{aj}(v)$ - atidaryto varto laidumo funkcija nuo įtampos, $G_{uj}(v)$ - uždaryto varto laidumo funkcija nuo įtampos.

Vartai gali būti vienoje iš dviejų būsenų: a arba u, kur a – atviri vartai, u – uždari vartai. Vartų būsenų kitimas vyksta diskrečiais laiko momentais t_i, $t_{i+1} = t_i + \Delta t$, $i = \overline{1,l}$, $l\,\Delta t = T$, kur T - modeliavimo laiko trukmė. Vartų būsenų kitimas parodytas 2.5-2 pav. Perėjimo tikimybės priklauso nuo vartų būsenos $s_j(t)$ ir įtampos $v_j(t)$, $j = \overline{1,2}, t \in [0,T]$:

$$K_j(t_{m+1}) = e^{A_1(P_1 V_j - V_{01})},$$

$$Paa_j(t_{m+1}) = 1 - \frac{\kappa_j K_j(t_{m+1})}{1+K_j(t_{m+1})},$$

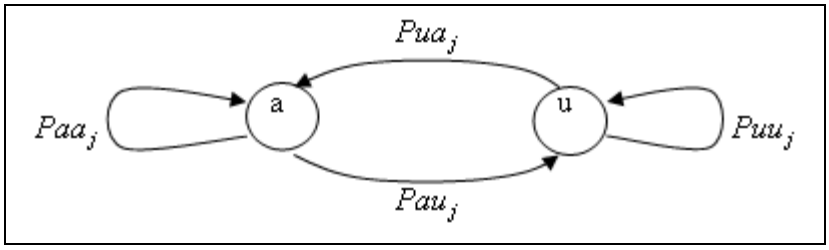
$$Pau_j(t_{m+1}) = \frac{\kappa_j K_j(t_{m+1})}{1+K_j(t_{m+1})},$$

$$Pua_j(t_{,+1}) = \frac{\kappa_j}{1+K_j(t_{m+1})},$$

$$Puu_j(t_{m+1}) = 1 - \frac{\kappa_j}{1+K_j(t_{m+1})},$$

$$Paa_j(t_{m+1}) + Pau_j(t_{m+1}) = 1,$$

$$Pua_j(t_{m+1}) + Puu_j(t_{m+1}) = 1.$$



2.5-2 pav. Vartų būsenų grafas

Vartų laidumas priklauso nuo vartų būsenos $s(t)_j$ ir pridėtos įtampos $V(t)$:

$$ga_j(t) = \begin{cases} Ga_j\left(Va_j(t)\right), jei\ Sa_j(t) = a \\ Gu_j\left(Vu_j(t)\right), jei\ Su_j(t) = u \end{cases};$$

Čia $G(V)$ – varto laidumo funkcija, $S_j(t)$ - varto būsena, $V(t)$ - pridėta įtampa.

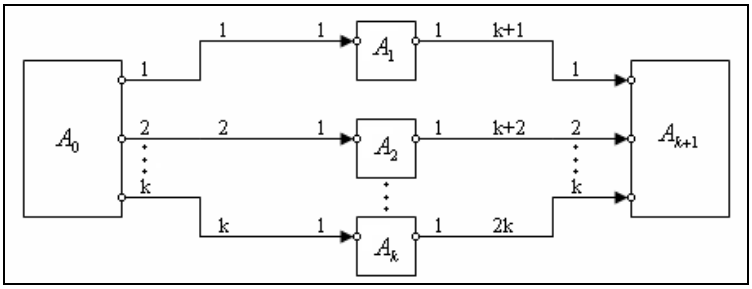
Vidutinis kanalų laidumas:

$$G_i(t) = \frac{\sum_{i=1}^{n} G_i(t)}{n} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{g_{ij}}}{n};$$

Čia $G_i(t)$ - kanalo laidumas t laiko momentu, n – kanalų skaičius, g_{ij} - i- tojo kanalo j-jų vartų laidumas.

Sudarytas modelis leidžia paskaičiuoti sistemos laidumą $G(t_i)$ laiko momentais t_i, $i = \overline{1,m}$, $t_{i+1} = t_i + \Delta t, m\ \Delta t = T$.

2.6 Agregatinis modelis



2.6-1 pav. Agregatinės sistemos schema

$$R = \begin{pmatrix} 1 & 2 & \dots & k & k+1 & k+2 & \dots & 2k \\ 1 & 2 & \dots & 1 & 1 & 2 & \dots & k \end{pmatrix}$$

$$H = \begin{pmatrix} 1 & 2 & \dots & k \\ k+1 & - & - & - \\ k+2 & - & - & - \\ \dots & \dots & \dots & \dots \\ 2k & - & - & - \end{pmatrix}$$

Agregato A_0 aprašymas

- Išėjimo signalų aibė $Y = \{y_1, y_2, ..., y_k\}$,

čia y_i - išduodama įtampa, $i = \overline{1,k}$.

- Vidinių įvykių aibė $E'' = \{e''_1\}$,

čia e''_1 - išduodamas y_i signalas, $i = \overline{1,k}$.

- Valdymo sekos $e''_1 \to \{\Delta t\}$,

čia Δt - laiko kitimo žingsnis.

- Diskrečioji agregato būsenos dedamoji $\nu(t_m) = \{V(t_m)\}$,

čia $V(t_m)$ - įtampa t_m laiko momentu.

- Tolydžioji agregato būsenos dedamoji $z_\nu(t_m) = \{w(e''_1, t_m)\}$,

čia $w(e''_1, t_m)$ - signalo išdavimo pabaigos momentas.

- Pradinė būsena $t_0 = 0$, $z(t_0) = \{0, \Delta t\}$.
- Parametrai:

$V(t_l) : T_l \to R$

- Perėjimo ir išėjimo operatoriai

$H(e''_1)$:

$\nu(t_{m+1}) = V(t_{m+1})$,

$t_m + \Delta t = t_{m+1}$,

$w(e''_1, t_{m+1} = t_m + \Delta t)$.

$G(e''_1)$:

$y_i = V(t_{m+1})$, $i = \overline{1,k}$.

Agregato $A_i, i = \overline{1,k}$ **aprašymas**

- Įėjimo signalų aibė $X = \{x_1\}$,

čia x_1 - paduodamos įtampos reikšmė.

- Išėjimo signalų aibė $Y = \{y_1\}$,

čia y_1 - kanalo laidumo reikšmė.

- Išorinių įvykių aibė $E' = \{e'_1\}$,

čia e'_1 - atėjo x_1 signalas.

- Diskrečioji būsenos dedamoji $\nu(t_m) = \{Sa_j(t_m), t_m, ga_j(t_m)\}$, $j = \overline{1,4}$,

čia $Sa_j(t_m)$ - a_j vartų būsena t_m laiko momentu, $ga_j(t_m)$ - a_j vartų laidumas.

- Pradinė būsena $t_0 = 0$, $\nu(t_0) = \{a, a, a, a, 0\}$
- Parametrai:

κ_j - vartų perėjimo koeficientas, $j = \overline{1,4}$,
A_j - vartų jautrumo koeficientas, $j = \overline{1,4}$,
$V0_j$ - įtampos koeficientas, $j = \overline{1,4}$,
$Ga_j(v)$ - atidaryto vartų laidumo funkcija, $j = \overline{1,4}$,
$Gu_j(v)$ - uždaryto vartų laidumo funkcija, $j = \overline{1,4}$.

- Perėjimo ir išėjimo operatoriai

$H(e'_1)$:

$$Sa_j(t_{m+1}) = \begin{cases} a,\ jei\ (Sa_j(t_m) = a) \wedge (\xi \le Paa_j(t_{m+1})) \\ u,\ jei\ (Sa_j(t_m) = a) \wedge (Paa_j(t_{m+1}) < \xi \le 1) \\ a,\ jei\ (Sa_j(t_m) = u) \wedge (\xi \le Pua_j(t_{m+1})) \\ u,\ jei\ (Sa_j(t_m) = u) \wedge (Pua_j(t_{m+1}) < \xi \le 1) \end{cases}$$

$$Va_j(t_{m+1}) = x_1 \frac{\frac{1}{ga_j(t_{m+1})}}{\sum_{k=1}^{4} \frac{1}{ga_k(t_{m+1})}}$$

$$ga_j(t_{m+1}) = \begin{cases} Ga_j(Va_j(t_{m+1})), jei\, Sa_j(t_{m+1}) = a \\ Gu_j(Va_j(t_{m+1})), jei\, Sa_j(t_{m+1}) = u \end{cases}$$

$G(e'_1)$:

$$y_1 = \sum_{k=1}^{4} \frac{1}{ga_k(t_{m+1})}.$$

- Perėjimo ir išėjimo operatoriai

$H(e'_1)$:

$$Sa_j(t_{m+1}) = \begin{cases} a, jei (Sa_j(t_m) = a) \wedge (\xi \le Paa_j(t_{m+1})) \\ u, jei (Sa_j(t_m) = a) \wedge (Paa_j(t_{m+1}) < \xi \le 1) \\ a, jei (Sa_j(t_m) = u) \wedge (\xi \le Pua_j(t_{m+1})) \\ u, jei (Sa_j(t_m) = u) \wedge (Pua_j(t_{m+1}) < \xi \le 1) \end{cases}$$

$$Va_j(t_{m+1}) = x_1 \frac{\frac{1}{ga_j(t_{m+1})}}{\sum_{k=1}^{4} \frac{1}{ga_k(t_{m+1})}}$$

$$ga_j(t_{m+1}) = \begin{cases} Ga_j(Va_j(t_{m+1})), jei\, Sa_j(t_{m+1}) = a \\ Gu_j(Va_j(t_{m+1})), jei\, Sa_j(t_{m+1}) = u \end{cases}$$

$G(e'_1)$:

$$y_1 = \sum_{k=1}^{4} \frac{1}{ga_k(t_{m+1})}.$$

Agregato A_{k+1} aprašymas

- Įėjimo signalų aibė $X = \{x_1, x_2, ..., x_k\}$,

čia x_i - i-tojo kanalo laidumas, $i = \overline{1,k}$.

- Išėjimo signalų aibė $Y = \{y_1\}$,

čia y_1 - išduodamas laidumas.

- Išorinių įvykių aibė $E' = \{e'_1, e'_2, ..., e'_k\}$,

čia e'_i - atėjo x_i signalas.

- Diskrečioji agregato būsenos dedamoji $\nu(t_m) = \{G_1(t_m), G_2(t_m), ..., G_k(t_m)\}$,

čia $G_i(t_m)$ - i-tojo kanalo laidumas t_m - laiko momentu, $i = \overline{1,k}$.

- Pradinė būsena $t_0 = 0,\ \nu(t_0) = \{0,0,...,0\}$

Perėjimo ir išėjimo operatoriai

$H(e'_1)$:

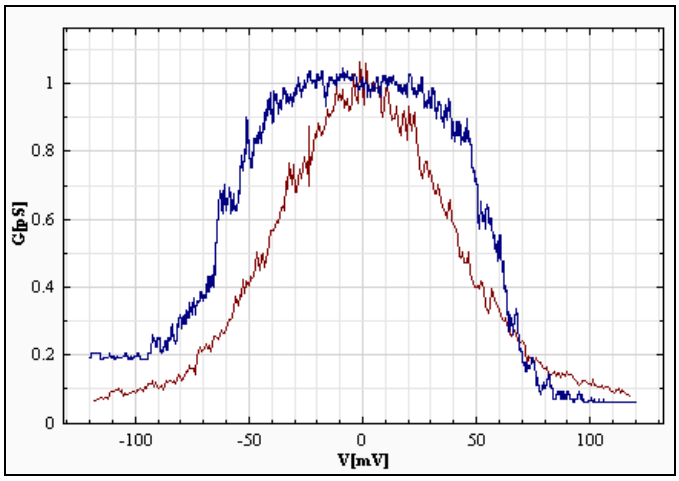
$$G_i(t_{m+1}) = G_i(t_m) + x_i$$

$G(e'_1)$:

$$G(t_{m+1}) = \frac{\sum_{i=1}^{k} G_i(t_m)}{n}, \text{ kai } t_m = t_{m+1}$$

3. Modelio parametrų optimizavimas

Vienas iš sukurtos sistemos (GJM – Gap Juncion Model, sukurta N. Paulausko - Kauno technologijos universiteto, informatikos fakulteto magistrantas) uždavinių yra pagrįsti įvairias tarpląstelinių plyšinių jungčių kontaktavimo prielaidas pasinaudojant eksperimento bei imitavimo rezultatais. Todėl yra svarbu automatizuoti šį procesą, tai yra pasinaudojant eksperimentiniais rezultatais surasti imitacinės sistemos atitikmenį ir jos įėjimo parametrus. Sistemos parametrų paieška yra labai svarbus uždavinys šioje sistemoje. Ji padėtų pagrįsti įvairias prielaidas ir surastų nežinomus parametrus įvairiems kanalų tipams. Tarkime mes turime eksperimentinius nufiltruotus duomenis (3.1 pav. – raudona kreivė) ir imitavimo rezultatus su tam tikrais įėjimo parametrais (3.1 pav. - mėlyna kreivė). Mums reikia rasti imitavimo parametrus su kuriais imitavimo ir eksperimento kreivių skirtumo kvadratas bus mažiausias.



3.1 Pav. Eksperimentiniai duomenys

Į šį uždavinį galima žiūrėti kaip į funkcijos aproksimavimą, t.y. regresiją, kai reikia surasti nežinomos funkcijos parametrus pagal taškų aibę. Sistemoje realizuota galimybė, kuri randa mažiausią skirtumo kvadratą tarp eksperimentinių duomenų ir imitacinės sistemos. Žemiau pateikėme suformuluoto uždavinio formalią specifikaciją:

$P_i = \{P_{i1}, P_{i2}, ..., P_{i8}\}$, čia i – i-toji iteracija

$l_i(P_i, u_i)$, $e(u_K)$, čia K=1, … , N – eksperimentų skaičius

$$\varepsilon = \sum_{K=1}^{N} (l_i(P_i, u_K) - e(u_K))^2 ,$$

čia P_i – i–tosios iteracijos parametrų rinkinys, l_i – i–tosios iteracijos laidumas, u_i - i-tosios iteracijos įtampa, e - eksperimentas, u_K – eksperimento įtampa, ε - skirtumo kvadratas.

3.1 Globalus optimizavimo uždavinys

Funkcijos parametru radimas buvo realizuotas pasitelkus globalaus optimizavimo algoritmus. Tokio tipo algoritmai yra naudojami įvairiems optimizavimo uždaviniams spręsti (Mokyklos tvarkaraščių sudarymas, keliaujančio pirklio uždavinys, ir pan.). Iš šių algoritmų realizacijai buvo pasirinkti ,,Exkor“, ,,Bayes“ ir „Monte Carlo“ algoritmai.

3.2 Optimizavimo metodai

3.2.1 Monte Carlo metodas

Šis skaičiavimo algoritmas, pagrįstas statistiniu modeliavimu ir gautų rezultatų apdorojimu statistiniais metodais. Šis metodas leidžia brangiai kainuojančius bandymus pakeisti modeliavimu

kompiuteriais ir labai sumažina tyrimų trukmę. Monte Karlo metodai dažniausiai naudojami fizikinių ir matematinių sistemų modeliavimui, kai neįmanoma gauti tikslių rezultatų naudojant deterministinį algoritmą**.**

Norint atlikti labai sudėtingą skaičiavimą, reikalaujantį ištyrinėti didelę duomenų erdvę, galima tą patį skaičiavimą atlikti tik su keletu atsitiktinai pasirinktų duomenų. Atsitiktinai parinkti duomenys dažniausiai būna „tipiški“, todėl natūralu tikėtis, kad ir atliktas skaičiavimas ne itin daug skirsis nuo tikslaus. Pavyzdžiui, nežinodami kaip apskaičiuoti apskritimo, nubrėžto kompiuterio ekrane, plotą, galėtume atsitiktinai išdėlioti keliasdešimt taškų. Suskaičiavę, kokia dalis taškų pateko į apskritimą, galėtume apytiksliai pasakyti ir jo plotą. Žinoma, tam turime suprasti, kaip generuojami atsitiktiniai taškai.

Norint atlikti išėjimo parametro reikšmių išsibarstymo tyrimą, reikia turėti ryšio funkciją $y = f(x_1, x_2, \ldots, x_n)$.

Tikslumo tyrimas susideda iš tokių etapų:

1. Modeliuojami elementų parametrų skirstiniai $W(xi)$;
2. Skaičiuojamos išėjimo parametro y reikšmės, esant atsitiktinėms xi reikšmių kombinacijoms, atitinkančioms $w(xi)$ dėsnius, t. y. modeliuojamas kūrybinis procesas;
3. Modeliavimo rezultatai apdorojami statistiniais metodais.

Šio apdorojimo tikslas yra įvertinti skaitines išėjimo parametro charakteristikas (vidutinę reikšmę ir dispersiją $D(y)$, nustatyti išėjimo parametro skirstinį $w(y)$ arba surasti tikimybę, kad išėjimo parametro reikšmės bus duotosiose ribose, kintant elementų parametrų reikšmėms pagal skirstinius.

Tiriant išėjimo parametrų tikslumą statistinių bandymų metodu, reikalingos elementų parametrų atsitiktinės reikšmės. Šių reikšmių modeliavimui naudojami atsitiktinių skaičių generatoriai. Didžiausią praktinę reikšmę turi vienodos tikimybės skaičiai intervale [0, 1]. Skaičiai su kitokiais norimais skirstiniais w(x) gaunami naudojant vienodos tikimybės skaičius ir sprendžiant lygtį parametro x atžvilgiu, esant įvairioms tikimybės P reikšmėms: $(0 \leq P \leq 1)$ [11].

Tokiu būdu gaunami pseudoatsitiktiniai skaičiai *xi*, pasiskirstę pagal skirstinį $w(xi)$. Naudojami ir kitokie atsitiktinių skaičių gavimo būdai. Visos šiuolaikinės elektroninės skaičiavimo mašinos turi programas, leidžiančias generuoti pseudoatsitiktinius skaičius, pasiskirsčiusius pagal norimą skirstinį. Nepriklausomai nuo atsitiktinių skaičių gavimo būdo apie jų kokybę galima spręsti iš gauto statistinio skirstinio sutapimo su norimu teoriniu skirstiniu. Apie skirstinių sutapimo laipsnį

sprendžiama iš sutapimo kriterijų. Praktikoje plačiausiai naudojami Pirsono ir Kolmagorovo sutapimo kriterijai [10].

Statistinių bandymų metodo pagrindiniai privalumai yra šie:

- Galima tirti išėjimo parametrų tikslumą, esant bet kokiems elementų parametrų skirstiniams;
- Galima gauti rezultatus su norimai maža paklaida; kai bandymų skaičius *N* artėja prie begalybės, skaičiavimų paklaida artėja prie nulio;
- Galima paskaičiuoti kiekybines išėjimo parametrų charakteristikas (vidutinę reikšmę, dispersiją), rasti skirstinį *w*(*y*) arba tikimybę, kad išėjimo parametras bus duotose ribose;
- Palyginus su natūrinių bandymų metodu, atsitiktinių bandymų metodas reikalauja mažai lėšų ir laiko išėjimo parametrų *y* tikslumo tyrimui atlikti.

Statistinių bandymų metodo trūkumas – sunku generuoti tarpusavyje priklausomų atsitiktinių dydžių reikšmes, t. y. sunku tyrinėti tikslumą, kai elementų parametrai yra priklausomi atsitiktiniai dydžiai [12].

3.2.2 Bayes‘o metodas ir jo modifikacijos

Tradicinė skaitinė analizė apibrėžia optimizavimo algoritmus, kurie garantuoja tam tikrą visų optimizuojamų funkcijų tikslumą. Tai apima tikslius algoritmus. Tai yra blogiausio atvejo analizė. Norint apriboti klaidų skaičių reikia daugiau skaičiavimo bandymų, dažniausiai didėjančių eksponentiškai. Tai pagrindinis blogiausio atvejo analizės trūkumas.

Kaip alternatyva yra naudojama vidutinio atvejo analizė. Čia vidutinė paklaida neapribojama, bet sumažinama tiek kiek įmanoma. Vidurkis imamas optimizuojamų funkcijų rinkiniui. Tokia vidutinio atvejo analizė vadinama Bayes'o metodu [9].

Bayes'o optimizacijos taikymui yra keli būdai. Pirmasis yra tiesioginis Bayes'o algoritmas. Jis apibrėžiamas fiksuojant ankstesnio išsibarstymo P funkcijų f(x) rinkiniui ir minimizuojant Bayes'o rizikos funkciją. Rizikos funkcija R(x) yra planuojamas skirtumas iš globalaus minimumo fiksuotame taške x. P išsibarstymas yra apgalvotas kaip stochastinis modelis funkcijai f(x), $x \in R$, kur f(x) gali būti determinuota arba stochastinė funkcija. Gauso atveju, laikant kad (n+1) stebėjimas yra paskutinis

$$R(x) = \frac{1}{\sqrt{2\pi s_n(x)}} x_{-\infty}^{+\infty} \min(c_n, z) e^{-\frac{1}{2}\left(\frac{y-m_n(x)^2}{s_n(x)}\right)} dz$$

Čia $c_n = \min_i z_i$ -, $z_i = f(x_i)$. $m_n(x)$ yra sąlyginė tikimybė atsižvelgiant į stebėjimų reikšmes z_i, i = 1, …, n. $s_n^2(x)$ yra sąlyginis pasiskirstymas ir $\varepsilon > 0$ yra taisymo parametras. Rizikos funkcijos R(x) minimumas yra surandamas taške:

$$x_{n+1} = \arg\max_x \frac{s_n(x)}{m_n(x) - c_n}$$

Tiesioginio Bayes'o algoritmo tikslas, naudojamas daugeliu atvejų, yra pateikti kiek įmanomai mažą paklaidą laikantis konvergavimo sąlygų.

Wienerio procesas yra bendras stochastinis modelis vienmačiu atveju m = 1. Šis modelis laiko, kad skirtumai $f(x_4) - f(x_3)$ ir $f(x_2) - f(x_1)$, $x_1<x_2<x_3<x_4$, yra stochastiškai nepriklausomi. Čia f(x) yra Gauso $(0, \sigma x)$ kiekviename fiksuotame x > 0.

Taip pat šis modelis yra išplėstas į daugiamatį atvejį. Tačiau paprasti apytiksliai stochastiniai modeliai pageidautini, jei m > 1. Apytiksliai modeliai yra suprojektuoti pakeičiant tradicines Kolmogorovo tikslumo sąlygoms. Šios sąlygos reikalauja n-tojo laipsnio matricų inversijos, kad surasti lyginę tikimybę $m_n(x)$ ir pasiskirstymą $s_n^2(x)$.

Keičiant įprastas tikslumo sąlygas (rizikos funkcijos R(x) tęstinumu, x_n konvergavimu iki globalaus minimumo, išraiškų $m_n(x)$ ir $s_n(x)$ supaprastinimu) R(x) išraiška apskaičiuojama naudojant formulę:

$$R(x) = \min_{1 \le i \le n} z_i - \min_{1 \le i \le n} {}^{x - x_i^2}_{z_i - c_n}.$$

Kitas būdas yra Bayes'o euristinis metodas. Jis fiksuoja ankstesnį pasiskirstymą P pagalbinėms funkcijoms $f_K(x)$. Šios funkcijos apibrėžia geriausias reikšmes gautas K kartų, naudojant kokią nors euristiką h(x). Laikoma, kad euristika h(x) priklauso nuo tęstinių parametrų $\to x \in R^m$. Euristika padeda optimizuoti originalią funkciją C(y) kintamiesiems $y \in R^n$, kur m < n. Paprastai, komponentai y yra diskretinės reikšmės. Euristika yra paremta ekspertų nuomonėmis apie sprendimo prioritetus.

Bayes'o euristinis metodas naudojamas optimizuojant funkcijas su dideliais kintamųjų skaičiais. Tai daugelio diskretinių optimizavimo problemų atvejai. Ekspertų nuomonėmis pagrįstos euristikos dažnai įtraukia atsitiktinių skaičių generavimo procedūras.

Bayes metodo modifikacija

Beyes‘o modifikacija vadinamas Exkor metodas, kuris atlieka daugiamatę paiešką. Paieška prasideda nuo pradinio taško. Optimizavimas atliekamas, naudojant Wiener modelį, atskirai keičiant kintamuosius. Pirmasis ciklas:

$$x_1^{n+1} = \arg\min_{x_1} f(x_1, x_2^n, ..., x_m^n),$$

$$x_2^{n+1} = \arg\min_{x_2} f(x_1^{n+1}, x_2, x_3^n, ..., x_m^n),$$

.....

$$x_m^{n+1} = \arg\min_{x_m} f(x_1^{n+1}, x_2^{n+1}, ..., x_{m-1}^{n+1}, ..., x_m).$$

Jei $x^{n+1} = x^n$ exkor metodas baigia darbą, priešingu atveju vykdomas kitas ciklas:

$$x_1^{n+2} = \arg\min_{x_1} f(x_1, x_2^{n+1}, ..., x_m^{n+1}),$$

$$x_2^{n+2} = \arg\min_{x_2} f(x_1^{n+2}, x_2, x_3^{n+1}, ..., x_m^{n+1}),$$

.....

$$x_m^{n+2} = \arg\min_{x_m} f(x_1^{n+2}, x_2^{n+2}, ..., x_{m-1}^{n+2}, ..., x_m).$$

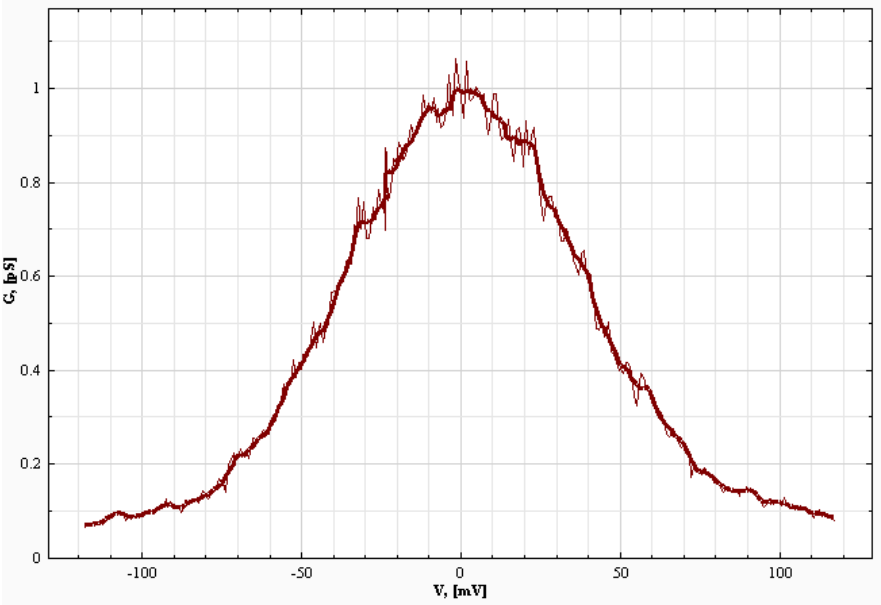
Čia $\arg\min_{x_i}$ žymi kintamojo x_i vertę, gautą naudojant Wiener optimizavimo algoritmą [8]. Metodas konverguoja traukos taške priklausomai nuo pradinio taško. Tam tikras ε-tikslumas gaunamas optimizuojant monotonišką transformaciją $v(f(x))$:

$$v(f(x)) = \sum_{i=1}^{m} v_i(x_i) + \varepsilon v_0(x).$$

Čia $\varepsilon > 0$, komponentas $v_0(x)$ priklauso nuo visų kintamųjų, komponentas $v_i(x_i)$ priklauso nuo vieno kintamojo x_i. Pavyzdžiui, jei $v(f(x)) = f(x)$, tai $f(x) = \sum_{i=1}^{m} f_i(x_i) + \varepsilon f_0(x)$. Jei $v(f(x)) = \log f(x)$, tai $f(x) = \varepsilon f_0(x) \prod_{i=1}^{m} f_i(x_i)$.

3.3 Eksperimentinė dalis

Pagrindinis eksperimento tikslas yra palyginti skirtingus globalaus optimizavimo algoritmus realizuotus sistemoje bei interneto svetainėje realizuota programa [7]. Globalaus optimizavimo arba paieškos eksperimentai yra labai svarbūs šiame darbe. Visų pirma, pabandysime surasti realių eksperimentų atitikmenis. Realūs eksperimentų duomenys buvo gauti iš užsakovų. Prieš tai jie buvo nufiltruoti t.y. dalinai panaikintas triukšmas, tuo tikslu, kad būtų gauti kuo tikslesni globalaus optimizavimo rezultatai (žr. 3.3–1 pav.). Plona linija vaizduoja gautus duomenis, o paryškinta vaizduoja filtravimą pritaikius slenkantį kablelį.

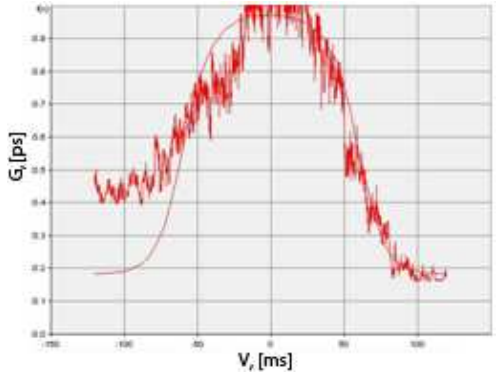


3.3-1 pav. Eksperimentiniai duomenys

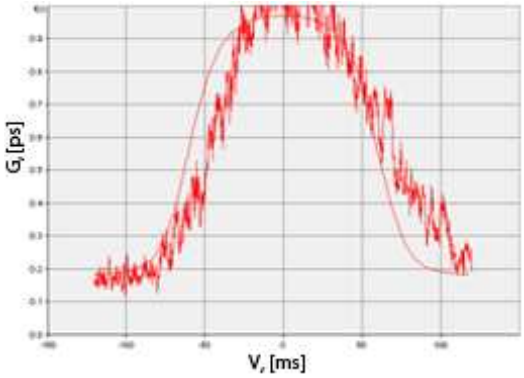
Visų pirma atliksime eksperimentus su optimizavimo uždavinio realizacija Java programavimo kalboje. Šioje versijoje programa papildyta kitais algoritmais („Monte Karlo“ bei „Bayes“) [7]. Naudosime pradinius duomenis, su tikslu įvertinti paieškos tikslumą, bei skaičiavimo laiką. Galiausiai eksperimentuosim su tokiais pat duomenimis GJM sistemoje, kuriems bus taikytas minimalus filtravimas (3.3-1 pav.).

Eksperimentai su “Monte Karlo” algoritmu

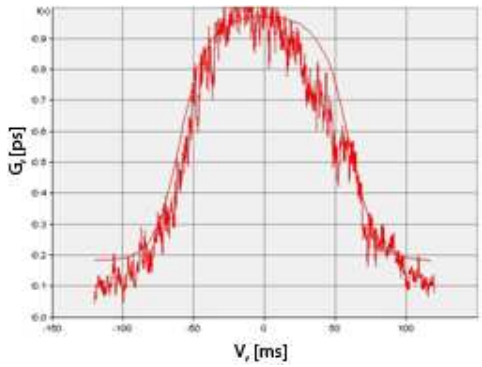
Taigi, pirmų eksperimentų pradiniai duomenys parinkti tokie, kad būtų kuo mažiau triukšmų (plona raudona linija žemiau pateiktuose grafikuose). Pirmuoju eksperimentu buvo bandomas „Monte Karlo“ metodas – Mg1. Gauti rezultatai:



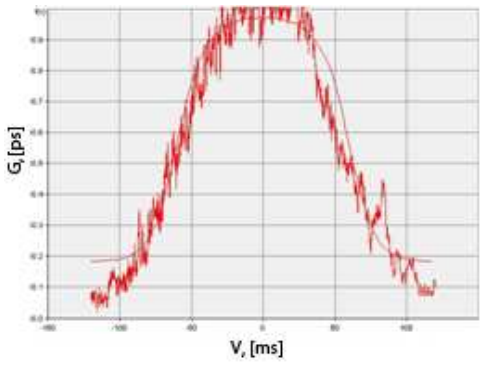
3.3-2 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 1000 iteracijų



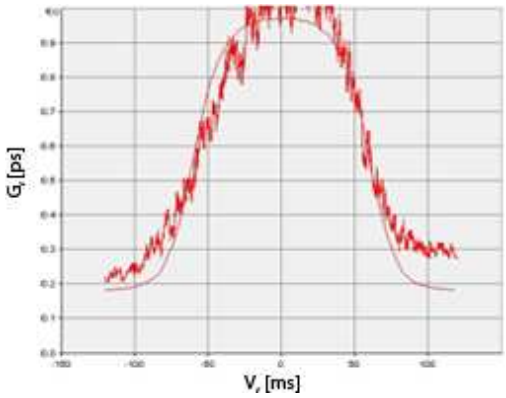
3.3-3 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 2000 iteracijų



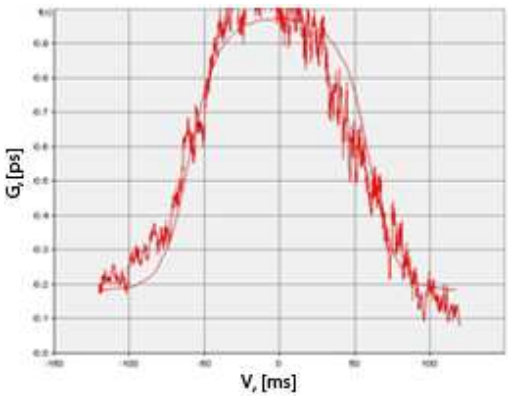
3.3-4 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 3000 iteracijų



3.3-5 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 4000 iteracijų

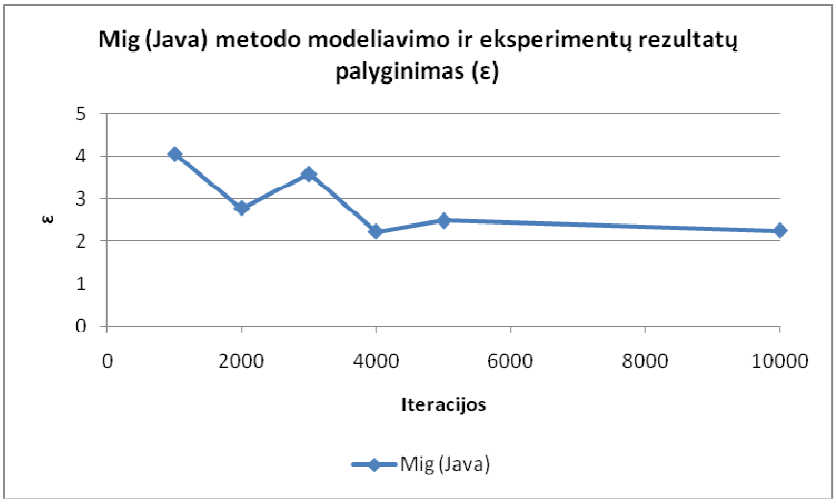


3.3-6 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 5000 iteracijų



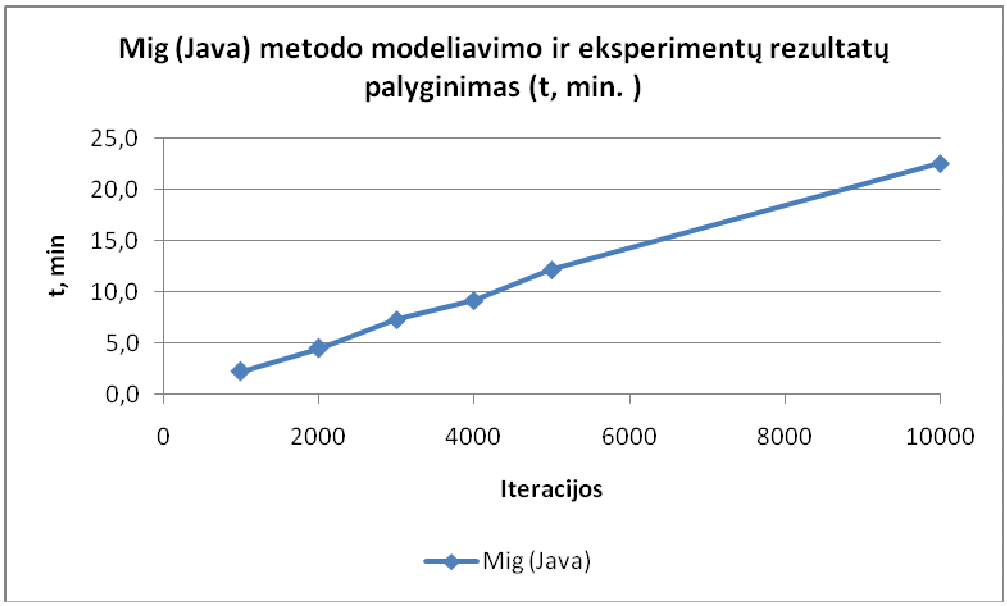
3.3-7 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 10000 iteracijų

Atliekant bandymus buvo įvertintas skirtumo kvadratas tarp modeliavimo ir eksperimentų rezultatų (skirtumo kvadratas nuo iteracijų skaičiaus):



3.3-8 pav. Mig(Java) metodo modeliavimo ir eksperimentų rezultatų palyginimas (ε priklausomybė nuo iteracijų skaičiaus)

Iš gautų duomenų brėžiamas laiko priklausomybės nuo iteracijų skaičiaus grafikas:

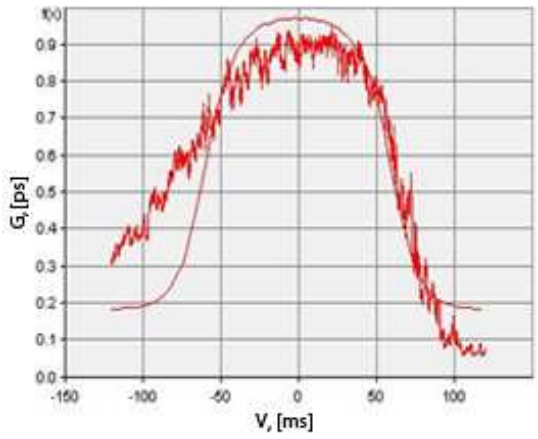


3.3-9 pav. Mig(Java) metodo modeliavimo ir eksperimentų rezultatų palyginimas (laiko priklausomybė nuo iteracijų skaičiaus)

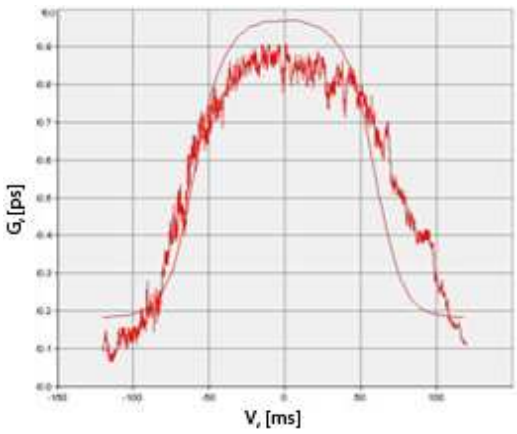
Iš rezultatų galime teigti, kad algoritmas veikia palyginus greitai ir duoda neblogą rezultatą. Silpnoji algoritmo puse yra ta, kad jis pagrįstas atsitiktinio ieškojimo principu. Kiekvieną kartą algoritmas gali rasti skirtingą rezultatą su tuo pačiu iteracijų skaičiumi. Taip pat ir didelis kiekis iteracijų negarantuoja gero rezultato.

Eksperimentai su “Bayes” algoritmu

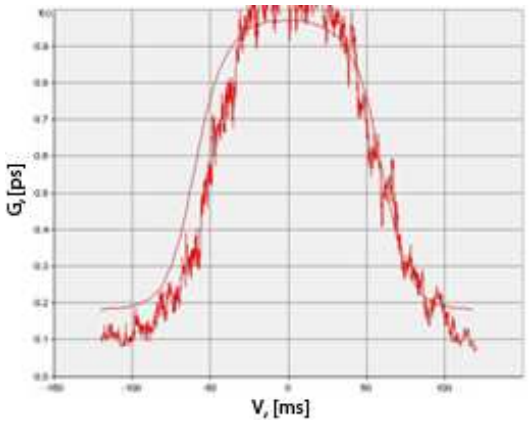
Toliau atliksime tokius pat bandymus, bet naudosime kitą algoritmą (Bayes), kaip ir anksčiau bandysime kas 1000 iteracijų:



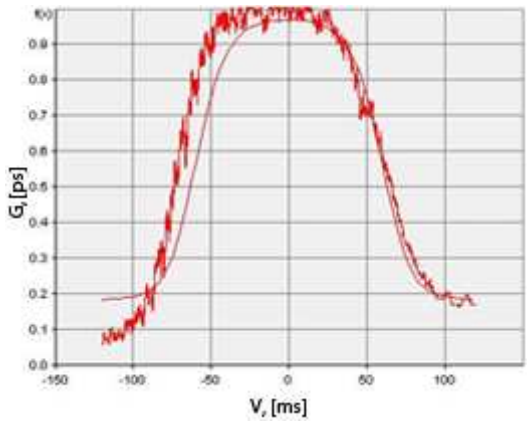
3.3-10 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 1000 iteracijų



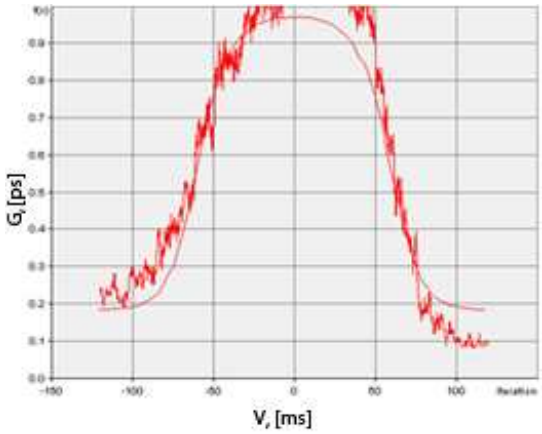
3.3-11 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 2000 iteracijų



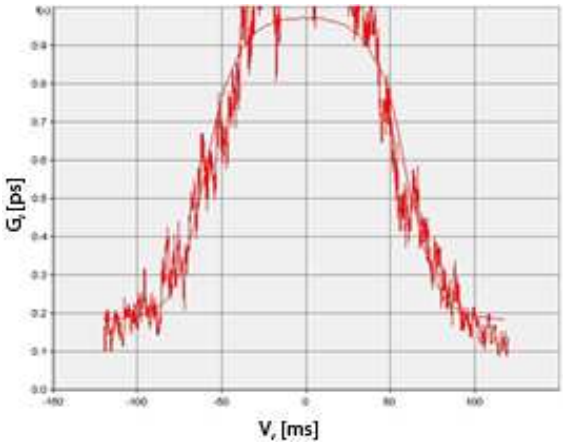
3.3-12 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 3000 iteracijų



3.3-13 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 4000 iteracijų

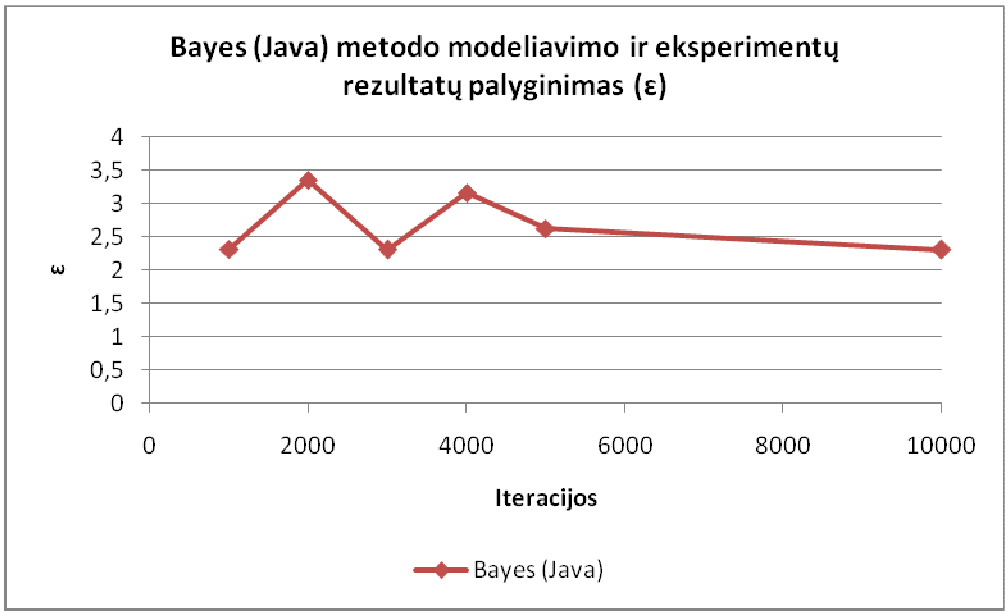


3.3-14 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 5000 iteracijų

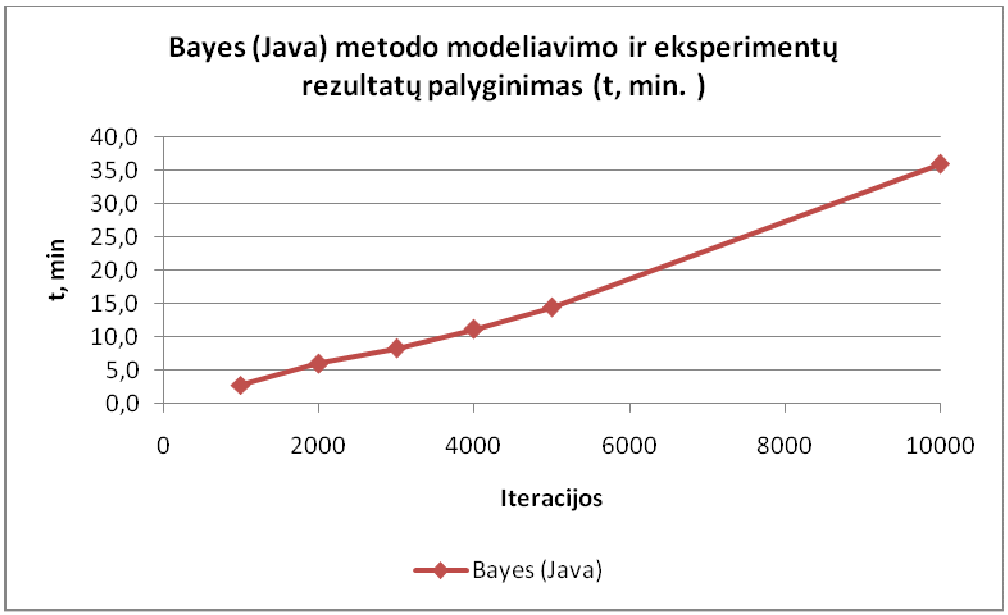


3.3-15 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 10000 iteracijų

Kaip ir ankstesniuose eksperimentuose, atliekant bandymus, buvo įvertintas parametrų radimo laikas, taip pat ir skirtumo kvadratas tarp pradinių duomenų ir modeliavimo rezultatų:



3.3-16 pav. Bayes(Java) metodo modeliavimo ir eksperimentų rezultatų palyginimas (ε priklausomybė nuo iteracijų skaičiaus)

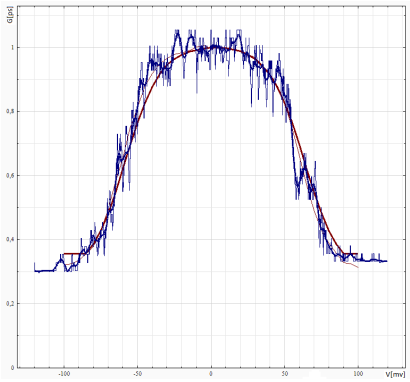


3.3-17 pav. Bayes(Java) metodo modeliavimo ir eksperimentų rezultatų palyginimas (laiko priklausomybė nuo iteracijų skaičiaus)

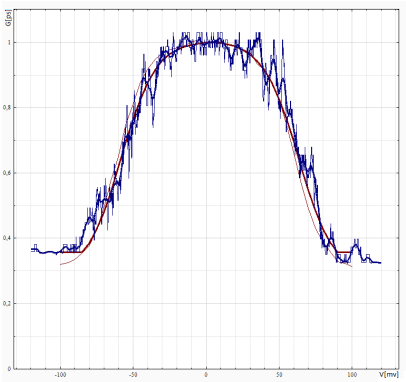
Nors metodo skaičiavimo laikas bei skirtumo kvadratas didesnis, vis tiek matosi, kad didinant iteracijų skaičių skirtumo kvadratas, kad ir ne ženkliai, bet mažėja. Taip pat pastebėta, kad metodas, kaip ir ankstesnis, su didesniu iteracijų skaičiumi gali duodi blogesnį rezultatą nei su mažesniu iteracijų skaičiumi.

Eksperimentai su “Exkor” algoritmu

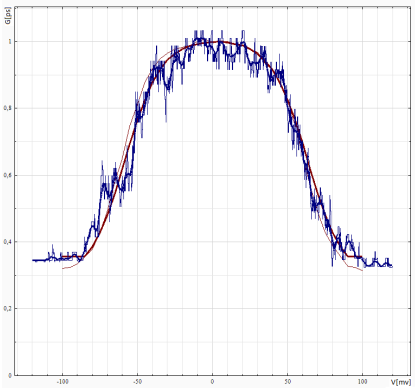
Paskutinis bandymas su GJM sistema. Kaip buvo minėta ankščiau imsime tokius pat pradinius duomenis, kad būtų galima palyginti su prieš tai darytais bandymais. Vėl kartosime bandymus kas 1000 iteracijų ir galiausiai padarysime paskutinį bandymą ženkliai padidinę iteracijų skaičių:



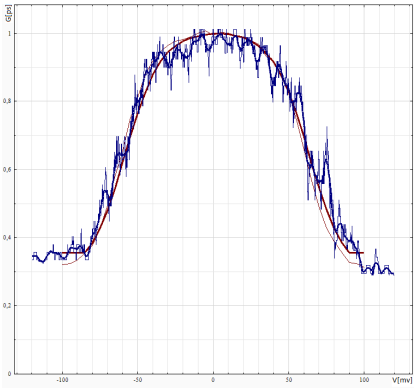
3.3-18 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 1000 iteracijų



3.3-19 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 2000 iteracijų

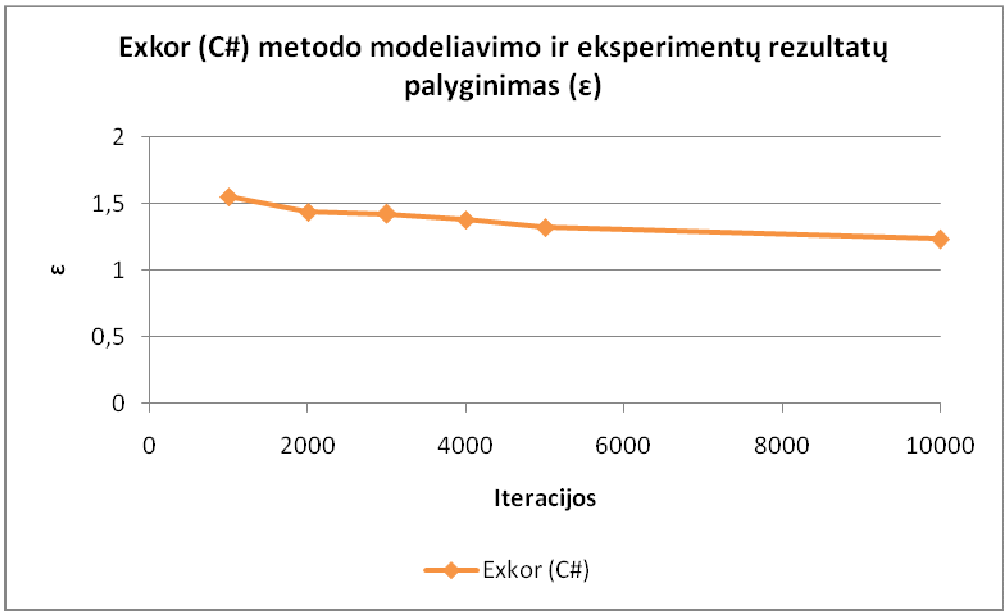


3.3-20 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 3000 iteracijų

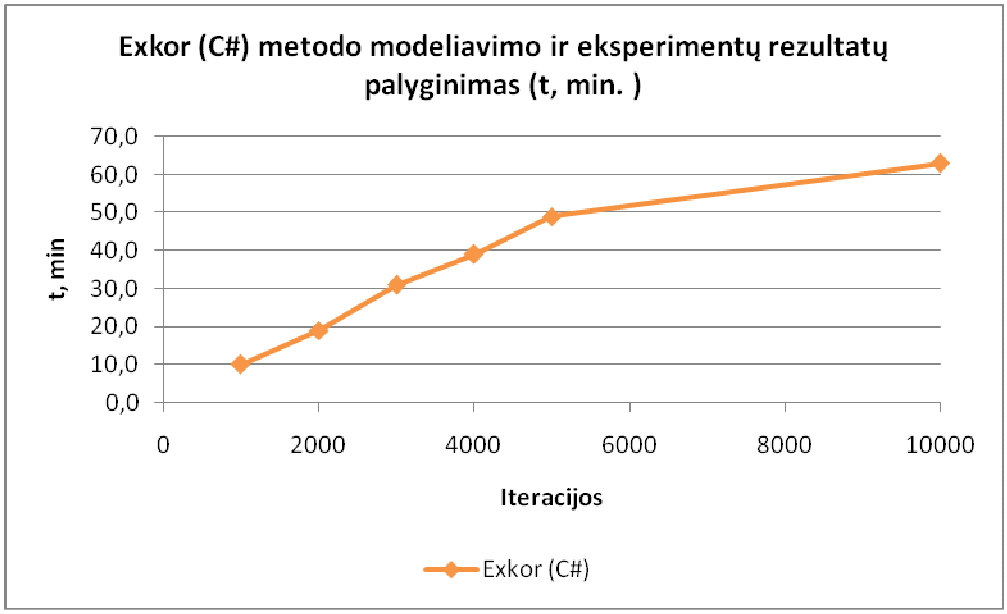


3.3-21 pav. Bandymo grafikas su optimaliu parametrų rinkiniu, gautu atlikus 4000 iteracijų

Atlikę bandymus, gavome tokius rezultatus:



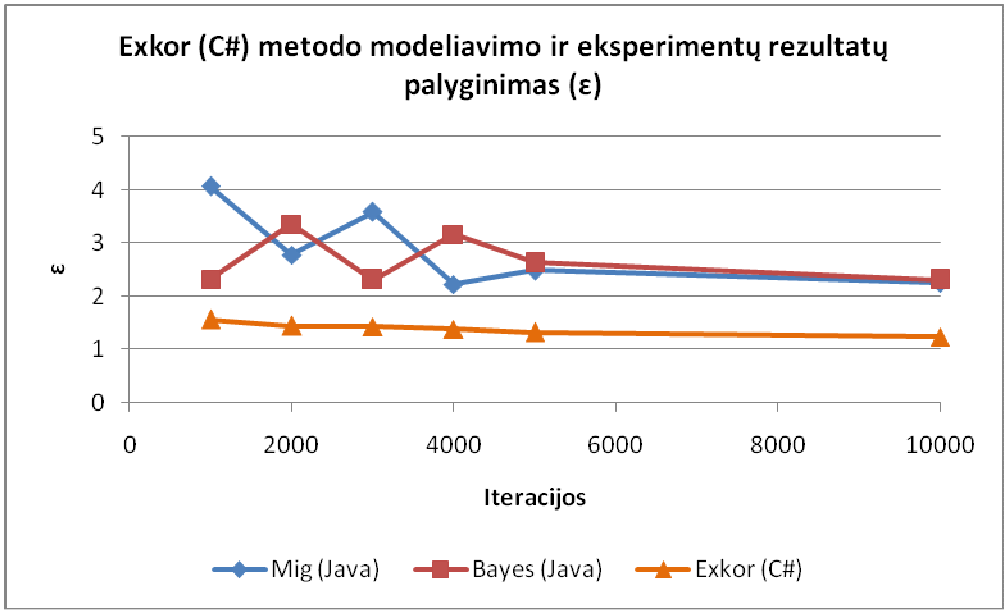
3.3-24 pav. Exkor(C#) metodo modeliavimo ir eksperimentų rezultatų palyginimas (ε priklausomybė nuo iteracijų skaičiaus)



3.3-25 pav. Exkor(C#) metodo modeliavimo ir eksperimentų rezultatų palyginimas (ε priklausomybė nuo iteracijų skaičiaus)

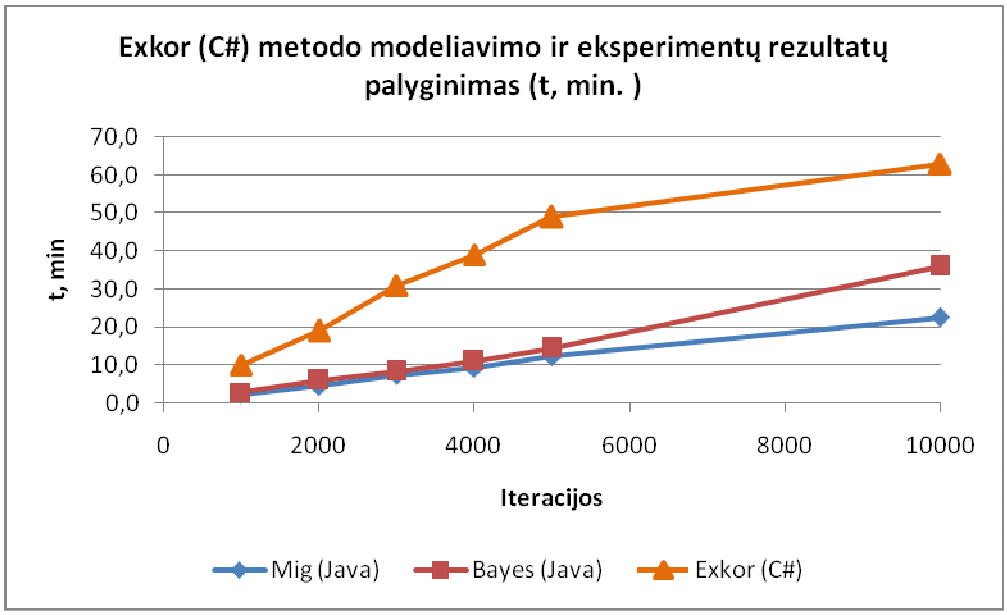
Iš grafikų (3.3-18 – 23 pav.) matome, kad gauti rezultatai, didinant iteracijų skaičių, vis labiau artėja prie realių bandymo rezultatų. Galime daryti išvadą, kad skirtumo kvadratas nuo iteracijų skaičiaus mažėja link 0 (žr. 3.3-24 pav.). Parametrų paieškos laikas stipriai išauga (žr. 3.3-25 pav.), todėl vykdant paiešką yra labai sunku ir reikalauja daug laiko resursų, kad prieiti prie norimo rezultato, ypač jei yra padidėjęs triukšmo lygis tiek eksperimentuose, tiek imitaciniuose rezultatuose. Eksperimentuojant su ,,exkor“ algoritmo suprojektuota parametrų paieška, pastebime,

kad ši sistema duoda geresnius rezultatus, skirtumo kvadrato atžvilgiu, nei prieš tai bandytų algoritmų (žr. 3.3-26 pav.).



3.3-26 pav. Modeliavimo ir eksperimentų rezultatų palyginimas (ε priklausomybė nuo iteracijų skaičiaus)

Iš grafiko (žr. 3.3-26 pav.) matome, kad "Exkor" algoritmo pagrindu skaičiavimai gaunasi tiksliausi, bet laiko atžvilgiu - prasčiausi (žr. 3.3-27 pav.).



3.3-27 pav. Modeliavimo ir eksperimentų rezultatų palyginimas (laiko priklausomybė nuo iteracijų skaičiaus)

3.4 Eksperimentų rezultatai

Matome, kad ,,exkor“ algoritmas, realizuoti šiam uždaviniui, buvo pasirinktas tikslingai. Jis veikia geriausiai lyginant su kitais algoritmais. Blogoji algoritmo pusė yra ta, kad norint gauti kuo tikslesnį rezultatą, reikia daug laiko resursų.

Kai kuriais atvejais užsakovams pasitelkus savo nuojautą, uždavinys buvo išspręstas sunaudojant mažiau laiko resursų nei kompiuteris [6]. Todėl žmogiškoji nuojauta tokioje sistemoje yra labai svarbi ir ieškant sudėtingesnių eksperimentų atitikmenis be žmogaus nesurasime. Galima daryti prielaidą, jog paieška be žmogaus įsikišimo yra nereikšminga sprendžiant sudėtingesnius uždavinius.

Taigi tam tikros žinios gali smarkiai paspartinti paieškos sistemą, nors šiuo metu paieškos greitis yra pakankamai geras, tačiau plečiant modelį (4 ar 6 vartų modelio parametrų paieška) sistema tampa sudėtingesnė ir paieška atitinkamai būtų vykdoma dar ilgiau.

Tiriant tokias sudėtingas sistemas, kaip neuronai, labai svarbu yra laikas. Yra gaunama begales duomenų, kuriuos reikia išanalizuoti, ištirti, palyginti. Parametrų paieškos modelis šiuo atveju nebetenka prasmės, dėl didelio laiko resursų sunaudojimo. Nes modeliuojant yra bandoma ar įvertinama labai daug faktorių, atliekama daug bandymų, kurių pasėkoje, šiuo atveju, tai truktų per ilgai.

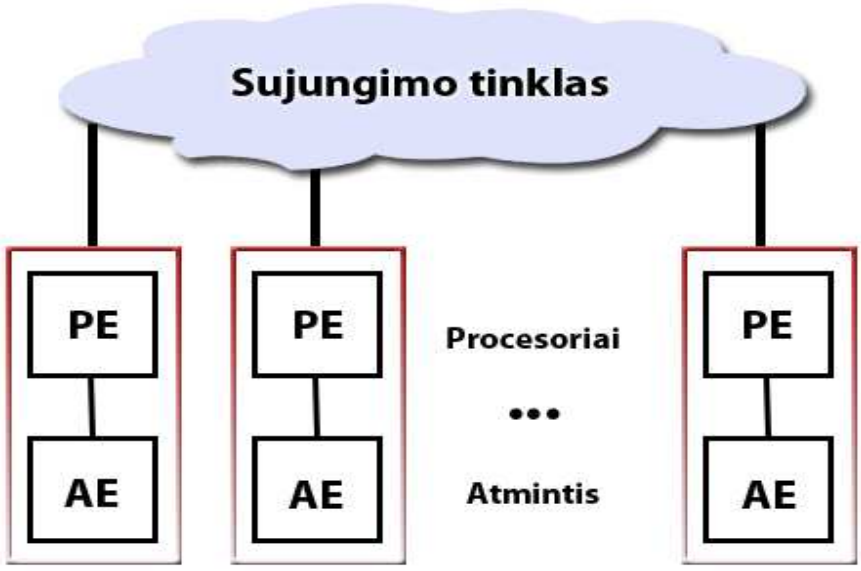
Kai įprastų nuosekliųjų kompiuterių resursų praktiniam uždaviniui spręsti nebeužtenka, vilčių teikia lygiagretieji kompiuteriai. Juose tą patį uždavinį tuo pačiu metu gali spręsti keli procesoriai, todėl sutrumpėtų bendra sprendimo trukmė. Taigi galimas sprendimo būdas (su tikslu sutrumpinti modelio parametrų ieškojimo laiką) – modelio lygiagretinimas.

4. Lygiagrečių skaičiavimų technologijos

Lygiagretieji algoritmai yra vienas iš būdų pagreitinti ilgai trunkančius sudėtingus skaičiavimus. Lygiagretinant programos kodą pirmiausia kyla klausimai, kaip išskaidyti užduotį į nepriklausomas dalis, kiek procesų reikės panaudoti, kaip paskirstyti užduotis procesams, kokią tinkamiausią technologiją pasirinkti lygiagretaus skaičiavimo realizavimui. Šiame skyriuje apžvelgsime lygiagrečių skaičiavimų technologijas, jų architektūras, privalumus ir trūkumus.

4.1 Asimetrinės technologijos

Asimetrinės technologijos lygiagretiems skaičiavimams naudoja sujungtų į tinklą kompiuterių (klasterių), procesorius ir atmintį (žr. 4.1-1 pav.), todėl būtinas duomenų apsikeitimas tarp procesų.



4.1-1 pav. Paskirstyta atmintis tarp procesorių

Šiuo metu paskirstytos atminties (pranešimų siuntimo) paradigma tampa vis daugiau ir daugiau populiari. Viena iš priežasčių yra platus platformų, kurios palaiko pranešimų siuntimo modelį, paplitimas. Programas parašytas šiuo metodu galima paleisti naudojant kelių procesorių: paskirstytos atminties daugiaprocesorinės arba kelias vienprocesorines (sujungtas į tinklą) sistemas. Pranešimų siuntimo paradigmoje procesai komunikuoja siųsdami vienas kitam pranešimus. Prie paskirstytos atminties technologijų priskiriama MPI technologija.

4.2 MPI (Message Passing Interface)

MPI – tai komunikacinių paprogramių bibliotekos (MPL) specifikacija, kuria yra naudojamasi programuojant C/C++ ir Fortran kalbomis. Biblioteką sudaro funkcijų ir paprogramių rinkinys, užtikrinantis komunikaciją tarp procesų. MPI standartas apibrėžia tiesiogines (angl. point to point) komunikacijas, kai komunikacija vyksta tik tarp dviejų procesų, ir kolektyvines (angl. collective) komunikacijas, kai komunikacijos ar sinchronizacijos operacijos įtraukia procesų grupes. MPI gali būti įvairiai realizuotas, pvz. MPICH, LAM-MPI ir kt. bibliotekomis, atitinkančiomis šį standartą.

Visi MPI paprogramių ir konstantų vardai prasideda su prefiksu MPI_, kad išvengti pavadinimų kolizijų. Pagrindinės MPI funkcijos: MPI_Init (inicijuoja lygiagrečiojo algoritmo darbo pradžią), MPI_Finalize (paskelbia lygiagrečiojo algoritmo pabaigą), MPI_Send (naudojama duomenų siuntimui) ir MPI_Recv (naudojama duomenų gavimui). Pastarosios dvi funkcijos įvykdomos tada, kai vienas procesas siunčia duomenis (MPI_Send), o kitas pasiruošęs juos priimti (MPI_Recv), t. y. tuo metu nevykdo jokių kitų veiksmų.

Pirmoji MPI procedūra iškviečiama bet kokioje MPI programoje, turi būti inicializavimo procedūra MPI_INIT. Kiekviena MPI programa turi išsikviesti šią procedūrą vieną kartą, prieš bet

kokias kitas MPI procedūras. MPI_FINALIZE procedūra iškviečiama programos pabaigoje. Ši procedūra išvalo visas MPI duomenų struktūras ir po jos iškvietimo jokia kita procedūra negali būti iškviesta.

Svarbų vaidmenį MPI komunikacijose atlieka komunikatorius – procesų grupė. MPI_COMM-WORLD komunikatorius yra apibrėžtas iš anksto. Jį sudaro visi procesai. Kitus komunikatorius gali sukurti pats vartotojas iš jau egzistuojančių komunikatorių, juos sujungiant ar išskaidant. Kiekvienas komunikatorius turi procesus, kurie komunikatoriaus viduje numeruojami nuo 0 iki N-1, kur N – procesų skaičius komunikatoriuje. Kiekvieno proceso numeris suprantamas kaip jo rangas (angl. rank). Rangas identifikuoja kiekvieną procesą komunikatoriaus viduje. Pavyzdžiui, rangas gali specifikuoti pranešimo šaltinį ar pristatymo vietą. Vartotojas bet kada gali sužinoti procesorių skaičių komunikatoriuje naudodamas MPI_COMM_SIZE (comm, size) paprogramę ir proceso rangą komunikatoriuje – MPI_COMM_RANK (comm, rank) paprogramę, kur comm yra komunikatoriaus pavadinimas.

Pranešimo pagalba perkeliamos kintamųjų reikšmės. Visi MPI pranešimai yra nustatyto tipo (žr. 4.2-1 lentelė), todėl turinio tipas turi būti vienodas išsiuntimo ir gavimo metu. Kai pranešimas išsiųstas, gavimo procesas turi tikėtis gauti tą patį duomenų tipą.

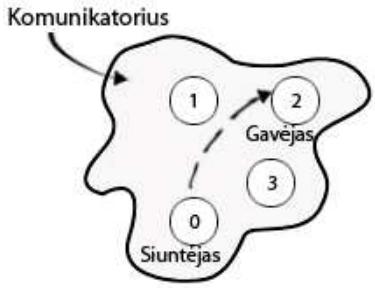
MPI duomenų tipas
MPI_INTEGER
MPI_REAL
MPI_DOUBLE_PRECISION
MPI_COMPLEX
MPI_LOGICAL
MPI_CHARACTER
MPI_BYTE
MPI_PACKED

4.2-1 lentelė. MPI duomenų tipai

Pavyzdžiui, jei procesas siunčia pranešimą su MPI_INTEGER duomenų tipu, gavimo procesas turi gauti duomenų tipą MPI_INTEGER, priešingu atveju komunikacija bus ne tiksli ir funkcionalumas neapibrėžtas. Išimtis yra tik su MPI_PACKED, kuris gali suderinti bet kokius tipus. Panašiai yra su Fortrano ar C/C++ kintamųjų tipais pranešime. Kintamieji turi atitikti MPI duomenų tipą, t.y. jei procesas siunčia pranešimą su duomenų tipu MPI_INTEGER, tai procesas kintamuosius turi apibrėžti INTEGER tipo, priešingu atveju funkcionalumas yra neapibrėžtas. Išimtis yra su MPI_BYTE ir MPI_PACKED.

4.2.1 MPI tiesioginė komunikacija

Tiesioginė komunikacija apima tik du procesus. Vienas procesas siunčia pranešimą kitam (žr. 4.2.1-1 pav.). Tuo tiesioginė komunikacija skiriasi nuo kolektyvinės komunikacijos, kuri apima visą procesų grupę vienu metu.



4.2.1-1 pav. Tiesioginė komunikacija

Kad pasiųsti pranešimą, procesas “siuntėjas“ iškviečia MPI_SEND, kuris nustato proceso “gavėjo“ rangą komunikatoriuje. Pastarasis procesas iškviečia MPI_Recv, kad gauti pranešimą. MPI_Send (message, count, datatype, dest, tag, comm, ierror), kur message – siunčiamas kintamasis, count – siunčiamų kintamųjų skaičius, datatype – MPI duomenų tipas, dest – proceso “gavėjo“ rangas, tag – markeris, naudojamas išskirti pranešimo tipus, comm – komunikatorius, ierror – gražinimo reikšmė.

MPI_Recv(message, count, datatype, source, tag, comm, status, ierror), kur source – proceso “siuntėjo“ rangas, status – gražinimo informacija.

Tiesioginėje komunikacijoje yra keturi siuntimo režimai (žr. 4.2.1-1 lentelė). Šie režimai naudojami tiek blokavimo formose, tiek formose be blokavimo.

Režimai	Užbaigimo sąlygos	Procedūra
Sinchronizuotas siuntimas	Baigiasi, kai gavimas yra užbaigtas	MPI_SSEND
Buferinis siuntimas	Visada baigiasi (jei neaptikta klaida) nepriklausomai ar buvo užbaigtas gavimas	MPI_BSEND
Standartinis siuntimas	Baigiasi, (jei neaptikta klaida) nepriklausomai ar gavimas yra užbaigtas	MPI_SEND
Pasiruošęs siuntimas	Baigiasi, (jei neaptikta klaida) nepriklausomai ar gavimas yra užbaigtas	MPI_RSEND
Gavimo	Baigiasi, kai pranešimas yra gautas	MPI_RECV

4.2.1-1 lentelė. Komunikacijos režimai

MPI tiesioginės komunikacijos savybės:

1. Pranešimų eilės tvarkos išsaugojimas. Pranešimai ne pralenkia vienas kito. Tarkim, procesas A siunčia du pranešimus procesui B su tuo pačiu komunikatoriumi. Procesas B siunčia du gavimo kvietimus, kurie suderina abu siuntimus. Tuomet du pranešimai bus gauti ta tvarka, kuria buvo išsiųsti (first in first out).

2. Nėra galimybės siųsti ir gauti pora pranešimų iškarto. Jei vienas MPI procesas siunčia pranešimą, o kitas procesas siunčia gavimo derinimą, tuomet bet kuris siuntimas ar gavimas ilgainiui užsitęs. Galimi du scenarijai: siuntimą gauna trečias procesas su suderintu gavimu, kurio atveju siuntimas užbaigiamas, bet antro proceso gavimas neužbaigiamas, arba trečias procesas pasiunčia pranešimą, kurį gauna antras procesas, kurio atveju gavimas užbaigiamas, bet pirmo proceso siuntimas neužbaigiamas.

4.2.2 MPI kolektyvinė komunikacija

Kolektyvinė komunikacija perduoda duomenis tarp visų komunikatoriaus apibrėžtų procesų. Kolektyvinėje komunikacijoje MPI teikia procedūrų įvairovę duomenų paskirstymui, perskirstymui ir surinkimui. Kad būtų atlikta kolektyvinė komunikacija, procesų aibei komunikatoriuje turi būti sukurtas naujas komunikatorius. Kolektyvinės komunikacijos sąvybės:

- kolektyvinė komunikacija netrukdo tiesioginei komunikacijai;
- bendradarbiavimas galimas, kai komunikatorius turi du vidinius komunikatorius, kurių vienas skirtas

  tiesioginei komunikacijai, o kitas – kolektyvinei;

- užbaigimas reiškia, kad buferis gali būti naudojamas toliau arba naudojamas pakartotinai;
- visi procesai komunikatoriuje turi iškviesti kolektyvinę komunikaciją. Tačiau kai kurie procedūriniai argumentai yra neprasmingi kai kuriems procesams ir gali būti apibrėžti kaip “fiktyvi“ reikšmė;
- panašumai su tiesiogine komunikacija:
  - pranešimas yra vieno konkretaus duomenų tipo masyvas;
  - siuntimo ir gavimo duomenų tipas turi sutapti.
- skirtumai:
  - siuntimo pranešimas turi užpildyti apibrėžtą gavimo buferį.

MPI pateikia šias kolektyvinės komunikacijos funkcijas:

- duomenų perdavimas (angl. broadcast) funkcijos pagalba iš vieno proceso siunčiamos pranešimų kopijos į visus kitus procesus;
- duomenų paskirstymo (angl. scatter) funkcijos atveju duomenys iš vieno proceso paskirstomi visiems procesams;
- duomenų surinkimo (angl. gathering) funkcijos atveju duomenys siunčiami iš visų procesų į vieną;
- visiško duomenų rinkimo (angl. allgather) atveju duomenų surinkimas taikomas visiems procesams;
- iš visų procesorių duomenų perdavimo visiems procesams (angl. alltoall) funkcijos pagalba visiprocesai gauna skirtingus rezultatus.

MPI privalumai:

- sujungti į tinklą kompiuteriai gali turėti skirtingas architektūras;
- konstruojant daugiaprocesorines sistemas patiriamos mažesnės išlaidos, nei perkant superkompiuterius.

Trūkumai:

- nėra tiesioginio proceso prieinamumo prie kito proceso atminties;
- informacijos apsikeitimas tarp procesų reikalauja laiko, todėl dideliais kiekiais vykstantis informacijos apsikeitimas neduos efektyvumo;
- reikalinga greita tinklinė įranga, kuri yra brangi.

4.3 Grid sistemos

Grid tinklas - tai geografiškai paskirstytų kompiuterinių resursų visuma, apjungtų į virtualų superkompiuterį. Grid resursus dažniausiai sudaro klasteriai, priklausantys Grid tinklo partneriams, kurie skiria visus arba dalį klasterio mazgų Grid vartotojų uždaviniams skaičiuoti [13]. Natūralu, kad tokiame tinkle ypač aktuali saugumo problema, kuri leistų lokalių resursų administratoriams pasitikėti Grid vartotojais bei jų grupėmis (virtualiomis organizacijomis) ir užtikrintų saugų vartotojų prisijungimą bei tinkamą resursų naudojimą.

Modelio parametrų ieškojimui buvo pasitelktas BalticGrid gridas, kuris jungia Lietuvą, Latviją, Estiją, Lenkiją ir kt. šalis (žr. 4.3-1 pav.).

4.3.-1 pav. BalticGrid padengimo žemėlapis

Norint gauti priėjimą reikia:

- Gauti BalticGrid pasirašytą skaitmeninį sertifikatą. Tam reikia susigeneruoti sertifikato užklausą ir kreiptis į Lietuvoje įgaliotus atstovus, kurie patikina jūsų asmens duomenis ir užklausą persiunčia į BalticGrid sertifikavimo tarnybą. Kelių dienų bėgyje elektroniniu paštu gauname laišką su prisegtu pasirašytu sertifikatu.
- Tapti bent vienos vartotojų grupės (dar vadinamos VO) nariu, nes tik vartotojų grupės, o ne atskiriems vartotojams suteikiami Grid resursai skaičiavimams atlikti.

Virtuali organizacija (VO) - tai vartotojų grupė, kuriai suteikta teisė leisti darbus tam tikrame Grid resursų rinkinyje t.y. tam tikruose Grid tinkle esančiuose klasteriuose. Pavyzdžiui viename moksliniame projekte dalyvaujantys vartotojai gali priklausyti vienai VO ir naudotis vienodais Grid resursais.

4.3.1 JDL

Į GRID siunčiamos užduotys aprašomos **JDL** *(Job Description Language)* failuose. Failo struktūra (pagrindiniai atributai ir jų reikšmės):

4.3.1-1 lentelė. Jdl failų pagrindiniai atributai ir jų reikšmės.

Atributas	Galimos reikšmės	Prasmė
Type	"Job"	
JobType	"Normal", "MPICH", "Interactive", "DAG", "Parametric", "Checkpointable"	Užduoties tipas
Executable	"/kelias/programa", "programa"	Programa, kuri bus vykdoma
Arguments	"arg" arba "arg1 arg2"	Užduoties programos argumentai
StdInput	"failo vardas"	Failas, kuris bus pakištas programai vietoj įvedimo iš konsoles *(stdin)*
StdOutput	"failo_vardas"	Kur bus įsimenami programos išvedami į ekraną rezultatai *(stdout)*
StdError	"failo_vardas"	Kur bus įsimenami programos pranešimai apie klaidas *(stderr)*
InputSandbox	{ "failas1", "failas2" }	Programa ir jos duomenys, kurie persiunčiami kartu su užduotim (**iki 10 MB**)
OutputSandbox	{ "failas1", "failas2" }	Failai, kurie gražinami su užduoties rezultatais
RetryCount	N	N - skaičius, kiek kartų bandyti pakartotinai paleist užduotį, jei užduotis pasibaigė *klaida*.
Requirements	(loginė išraiška)	Reikalavimai, pagal kuriuos atrenkami užduoties vykdymui tinkami klasteriai.
Rank	(aritmetinė išraiška)	Išraiška, pagal kurią rūšiuojami reikalavimus tenkinantys klasteriai
CPUNumber	N	Norimas klasterio *core*'u skaičius (MPICH užduotims, *core*'ai išskiriami viename klasteryje)

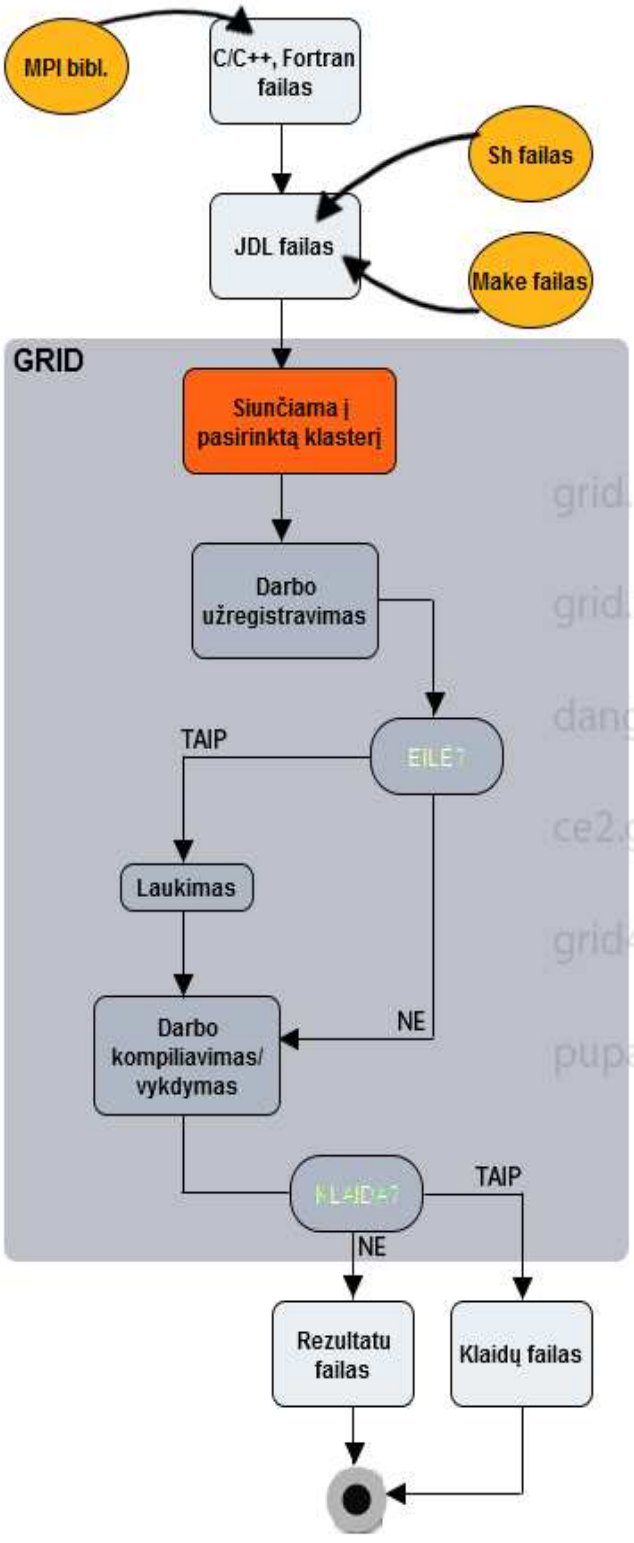
Taigi iš lentelės duomenų matome (žr. 4.3.1 – 1 lentelė), kad pagrinde reikia nurodyti „Job Type“ atributo reikšmę į MPICH – tai reiškia, kad skaičiavimai bus atliekami su nurodytu atribute „CPUNumber“ procesorių skaičiumi. Taip pat patartina nurodyti „Requirements“ atributą, kad darbas patektų į reikalavimus atitinkantį klasterį. Ir be abejo atribute „InputSandBox“ ir „OutputSandBox“ nurodyti failus, kokie bus siunčiami ir gaunami.

4.3.2 Skaičiavimų paleidimas BalticGrid gride

Gavus reikiamą sertifikato patvirtinimą ir įsiregistravus į vieną iš virtualių organizacijų, gavome priėjimą prie grido resursų. Išlygiagretinti C++ modelio failai (ir kiti failai reikalingi uždaviniui paleisti) perkeliami į specialiai išskirtą vietą gride. Serveryje galima pasitikrinti grido klasterių būklę (komanda: lcg-infosites –vo balticgrid all), t.y. kiek procesorių, reikiamame klasteryje naudojama, kiek jų yra laisvų. Taip pat gaunama informacija apie resursų užimtumą. Skaičiavimų paleidimas gride ir rezultatų gavimas (žr. 4.3-1 pav.):

- Pirmas žingsnis - užregistruoti skaičiavimus (darbą) gride. Galima nurodyti reikiamą klasterį savarankiškai (komanda: glite-wms-job-submit –a –o jobs –r ce.grid.lei.lt:2119/jobmanager-pbs-balticgrid job.jdl) arba tai patikėti sistemai. Taigi šiuo žingsniu nusiunčiame klasteriui nurodymus apie darbą ir reikiamus failus.

- Darbas užregistruojamas ir nustatoma eilė (darbo statusas – sheduled). Jei eilės nėra ir yra tinkamas skaičius procesorių darbas paleidžiamas vykdymui (darbo statusas - running).
- Kai darbas baigtas (darbo statusas – done) arba kai darbas nutrauktas (darbo satusas – aborted). Jei darbas baigiasi sėkmingai, visi gauti rezultatai atspausdinami viename faile „job.out“ , o jei darbas, dėl kažkokių priežasčių buvo nutrauktas, faile „job.err“ atspausdinama nutraukimo priežastis. Pataisius klaidas, vėl pradedama nuo darbo užregistravimo.



4.3-1 pav. Darbo realizacijos gride schema

5. Lygiagretaus skaičiavimo realizacija paieškos modelio programos kode

Ilgų diskusijų su užsakovais pasekoje buvo nuspręsta lygiagretinti ir realizuoti GRID aplinkoke, visų pirmą, modelį su „Monte Carlo“ optimizavimo algoritmu - Mig1, dėl jo greito ir paprasto veikimo. Modelis buvo supaprastintas ir perrašytas į C++ kalbą.

Šiuo metu GJM parametru paieškos programiniam kodui yra pritaikyta MPI technologija. Eksperimentiniai skaičiavimai buvo atlikti GRID aplinkoje su 2-8 procesoriais.

5.1 MPI bibliotekos pritaikymas (modeliui su „Monte Carlo“ metodu)

Mūsų atveju, modelio lygiagretinimas realizuojamas OPENMPI-1.2.3-GCC-4.2.0 bibliotekos pagalba. Kaip minėta ankščiau, kad programa būtų paleista GRID aplinkoje, reikalingas „Make“ failas, kuriame nurodoma kokį kompiliatorių reikia naudoti kompiliuojant programą. Taip pat parodytas ir JDL failo pavyzdys, kuriame nurodėme kokio tipo užduotis, kiek naudoti procesorių, ir kokie papildomi failai bus reikalingi darbui atlikti. Taip pat nurodomi reikalavimai klasteriui „Requirements“.

Make failas

```
all: mpi_grid mpi_gridxx

mpi_grid: mpi_grid.c
        mpicc –o mpi_grid mpi_grid.c
mpi_gridxx: mpi_gridxx.cc
        mpicxx –o mpi_gridxx mpi_gridxx.cc
clean:
        rm –f *.o mpi_grid mpi_gridxx
```

JDL failas

```
JobType = “MPICH”;

CpuNumber = 8;
Executable = “job.sh”;
StdOutPut “job.out”;
StdError = “job.err”;
InputSandbox = {“job.sh”, “Makefile”, “mpi_gridxx.cc”,”distribute”,”cleanup”};
OutputSandbox = {“job.err”, “job.out”};
RetryCount = 0;
Requirements=Member(“VO-balticgrid-E-PARALLEL-OPENMPI-1.2.3-GCC-4.2.0”,
other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

MPI kodo fragmentas programoje

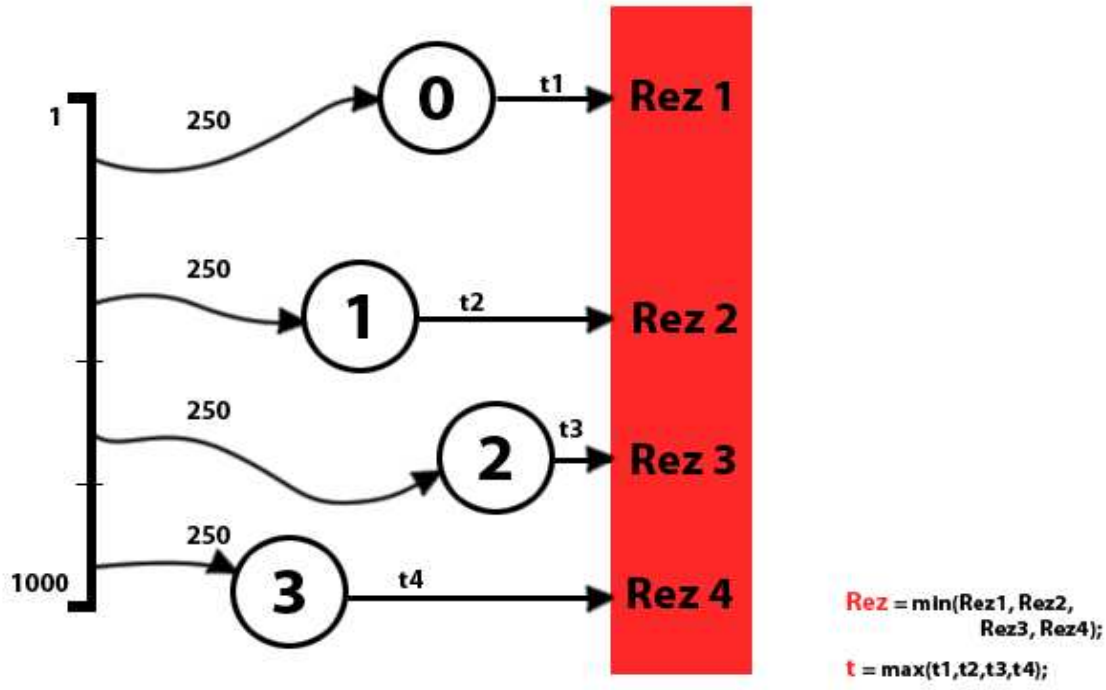
```
//--------------------------------------------------------------------------------------------------

// Divide iterations
//--------------------------------------------------------------------------------------------------
void Divide(int *b, int len, int it)
{
	int i;
	double check = it/len;
	double check2 = floor(check);
	if (check > check2)
	{
		for (i = 0; i < len-1; i ++){
				b[i] = int(check2);
		}
		b[len-1] = it - (int(check2)* (len-1));		}
	else if (check == check2){
		for(i = 0; i < len; i++){
				b[i] = int(check);
		}
	}
}
//--------------------------------------------------------------------------------------------------
// Main method
//--------------------------------------------------------------------------------------------------
int main(int argc, char * argv[])
{
	int Iterations = 10000;
	ModelTask task;

		int cpu_number;
		int portion[1];
		int rank;
		double startwtime = 0.0, endwtime;
	int i;
	double sum = 10000;
		MPI::Init(argc, argv);
.....
		cpu_number = MPI::COMM_WORLD.Get_size();
		rank = MPI::COMM_WORLD.Get_rank();
.....
		int a[cpu_number];
		Divide(a, cpu_number,Iterations);
		MPI::COMM_WORLD.Scatter(a,1,MPI::INT,portion,1,MPI::INT,1);
.....
		srand(rank * (unsigned) time(NULL));
		startwtime = MPI::Wtime();
	for(i = 0; i < portion[rank]; i++){
```

```
                RandomPoint randompoint(task.domain);
                double log = task.f(randompoint.x);
                if (log < sum) sum = log;
                printf("Proccess nr %d  of %d;  VKP %f    Min %f\n", rank, cpu_number, log, sum);
        }
        endwtime = MPI::Wtime();
.....
        printf("Total time took: %f sec. Min Rez:%f  Procesor rank %d  \n", (endwtime-startwtime), sum, rank);
        MPI::Finalize();
.....
        return 0;
}
```

Rezultatų failo pavyzdys

```
/opt/exp_soft/balticgrid/E-PARALLEL-OPENMPI-1.2.3-GCC-4.2.0/bin/mpirun
/opt/exp_soft/balticgrid/E-PARALLEL-OPENMPI-1.2.3-GCC-4.2.0/bin/mpicc
/opt/exp_soft/balticgrid/E-PARALLEL-OPENMPI-1.2.3-GCC-4.2.0/bin/mpicxx
mpicxx -o mpi_gridxx mpi_gridxx.cc
Shared home or single node
 ++++ no copying necessary
.....
Total time took: 21.551556 sec. MinRez:4.018461  Procesor rank 0
Total time took: 21.493830 sec. MinRez:5.469309  Procesor rank 3
Total time took: 22.207225 sec. MinRez:3.091911  Procesor rank 2
Total time took: 22.350022 sec. MinRez:5.539096  Procesor rank 1
Total time took: 24.969002 sec. MinRez:3.947308  Procesor rank 5
Total time took: 25.115013 sec. MinRez:3.481137  Procesor rank 4
rm -f *.o mpihello mpihelloxx
Shared home or single node
 ++++ no cleaning necessary
```

Programa pradžioje nustato, kiek procesorių buvo nurodyta užklausoje. Sekantis žingsnis– padalina iteracijų skaičių po lygiai kiekvienam procesoriui (žr. 5.1-1 pav.):



5.1-1 pav. Modelio lygiagretinimo schema

MPI pagalba siunčiamas iteracijų skaičius, kurį atskiras procesorius turės apdoroti (kintamasis portion). Toliau programoje procesoriai, nepriklausomai vienas nuo kito, atlieka skaičiavimus. Pasibaigus skaičiavimams, procesorius atspausdina į failą savo surastą geriausią rezultatą ir kiek laiko jam užtruko pereiti užduotą iteracijų skaičių. Dėl metodo pritaikomumo lygiagretinimui pavyko išvengti dažno komunikavimo tarp procesų, kurių pasekoje gali išaugti bendras uždavinio skaičiavimo laikas (realizuota kolektyvinė MPI komunikacija). Galiausiai išrenkamas geriausias rezultatas ir didžiausias laikas.

5.2 MPI bibliotekos pritaikymas (modeliui su „Bayes“ metodu)

Šiuo metodu realizuota lygiagrečioji versija yra panaši į prieš tai nagrinėto metodo versiją, tik papildomai kiekvienas procesas dar įvertina paramėtų reikšmes skirtinguose taškuose. Paleisti skaičiavimams naudojamas tas pats kompiliatorius ir prieš tai naudoti papildomi failai (sh, jdl, Make)

```
int main (int argc, char * argv[])
{
        int Iterations = 1000;
        int InitialPoint = 10;
        ModelTask task;
        Result result;
        ResultLogger resultlogger;
```

```
	Point mz;
	Point x2;
	int ix = 0;
	int cpu_number;
	int portion[1];
	int rank;
	double startwtime = 0.0, endwtime;
	MPI::Init(argc, argv);
.....
	cpu_number = MPI::COMM_WORLD.Get_size();
	rank = MPI::COMM_WORLD.Get_rank();
.....
	int a[cpu_number];
	Divide(a, cpu_number,Iterations);

	MPI::COMM_WORLD.Scatter(a,1,MPI::INT,portion,1,MPI::INT,1);
.....
	srand(rank * (unsigned) time(NULL));
	startwtime = MPI::Wtime();

	for(int i = 0; i < InitialPoint; i++)
	{
		LPPoint lppoint =  LPPoint(task.domain, i);
		Result result1 = Result(i, lppoint, task.f(lppoint.x));
		resultlogger.log(result1);
		if (result1.value < result.value) result = result1;
		//printf("VKP: %f  Iteration:  %d  Min VKP: %f Proc %d of %d \n", result1.value, (i+1), result.value, rank,
cpu_number);
	}
	for (int j = InitialPoint; j < portion[0]; j++)
	{
		Point point = mig2f2(resultlogger, task, result, j, mz, x2, ix);
		Result result2 =  Result(j, point, task.f(point.x));
		resultlogger.log(result2);
		if (result2.value < result.value)result = result2;
		//printf("VKP: %f  Iteration:  %d  Min VKP: %f Proc %d of %d \n", result2.value, (j+1), result.value, rank,
cpu_number);
	}
	//system("PAUSE");
	endwtime = MPI::Wtime();
.....
	printf("Total time took: %f sec. MinRez:%f  Procesor rank %d \n", (endwtime-startwtime), result.value , rank);
	MPI::Finalize();
.....
	return 0;
}
```

6. Modelio greitaeigiškumo analizė

Modelio greitaeigiškumui patikrinti, buvo atlikti eksperimentai naudojant pupa.elen.ktu.lt klasterį, kurio pagrindiniai parametrai surašyti žemiau pateiktoje lentelėje (žr. 6-1 lentelė):

6-1 lentelė. Pupa.elen.ktu.lt klasterio parametrai

Klasterio pavadinimas	Pupa.elen.ktu.lt
OS	ScientificSL 4.6
CPU tipas	Intel P4 3000 MHz
CPU skaičius	10
Atmintis	1024 MB
Benchmark	1160
Mazgo disko dydis	840 GB

Rezultatai su Monte Carlo metodu

Efektyvumui nustatyti gali būti pasirinktos charakteristikos:

- Spartinimo koeficientas Sp = T0/Tp, įvertinantis pagreitėjimą, kurį pasiekiame spręsdami uždavinį lygiagretindami, naudodami p procesorių (T0 – greičiausio nuosekliojo algoritmo vykdymo laikas, Tp – lygiagrečiojo algoritmo vykdymo laikas naudojant p procesorių);
- Efektyvumo koeficientas Ep = Sp/p, parodantis, kokią dalį procesorių pajėgumo pasitelkėme spręsdami uždavinį lygiagrečiuoju algoritmu.

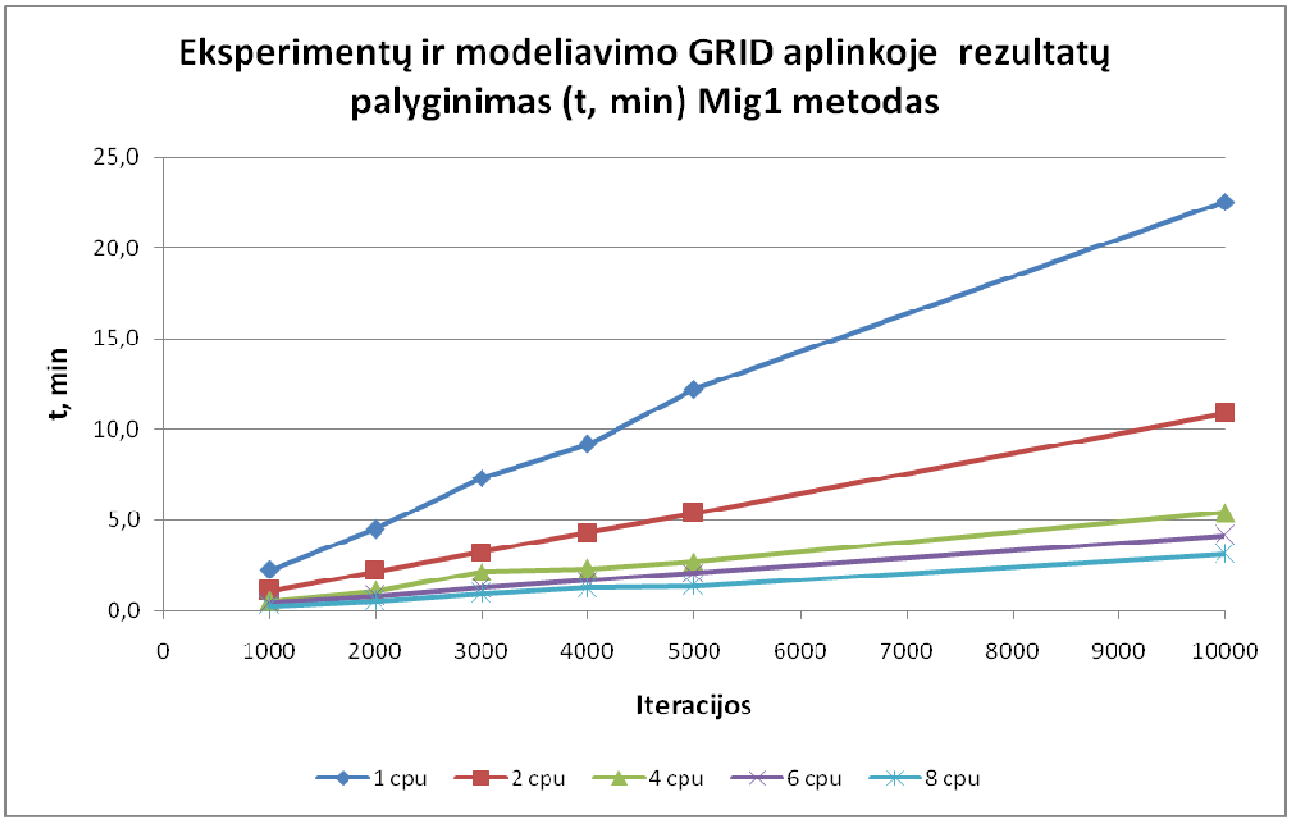
Skaičiavimus atliekant lygiagretinant kodą („Monte Carlo“ metodas) buvo naudojami 2-8 procesoriai. Rezultatai gauti didinant iteracijų skaičių pateikti lentelėje (žr. 6-2 lentelė):

Programos vykdymas	Procesorių skaičius, p	Skaičiavimo laikas s/ Iteracijos						Pagreitėjimas, Sp	Efektyvumas, Ep
		1000	2000	3000	4000	5000	10000		
Nuosekliai	1	135	270	438	550	733	1354		
Naudojant MPI biblioteką	2	65	131	194	258	322	652	2	1
	4	32	65	128	138	161	323	4	0,98
	6	25	49	74	99	124	248	5,6	0,94
	8	16	32	56	74	81	187	7,8	0,9

6-2 lentelė. Skaičiavimo trukmės priklausomai nuo iteracijų skaičiaus (Mig1)

Kaip ir analizės dalyje, tiriame, kiek skaičiavimai užtrunka kas 1000 iteracijų, o paskutinį bandymą su atitinkamu procesoriumi atlikome ženkliai padidindami iteracijų skaičių. Iš lentelės (žr. 6-1 lentelė) duomenų matome, kad su mažu procesorių skaičiumi metodo skaičiavimai pagreitėja

tiek kiek procesorių. Didinant procesorių skaičių 6-8 pagreitėjimas mažėja. Žemiau pateiktame grafike (žr. 6-1 pav.) pavaizduotas eksperimentų ir modeliavimo rezultatų trukmės palyginimas minutėmis.

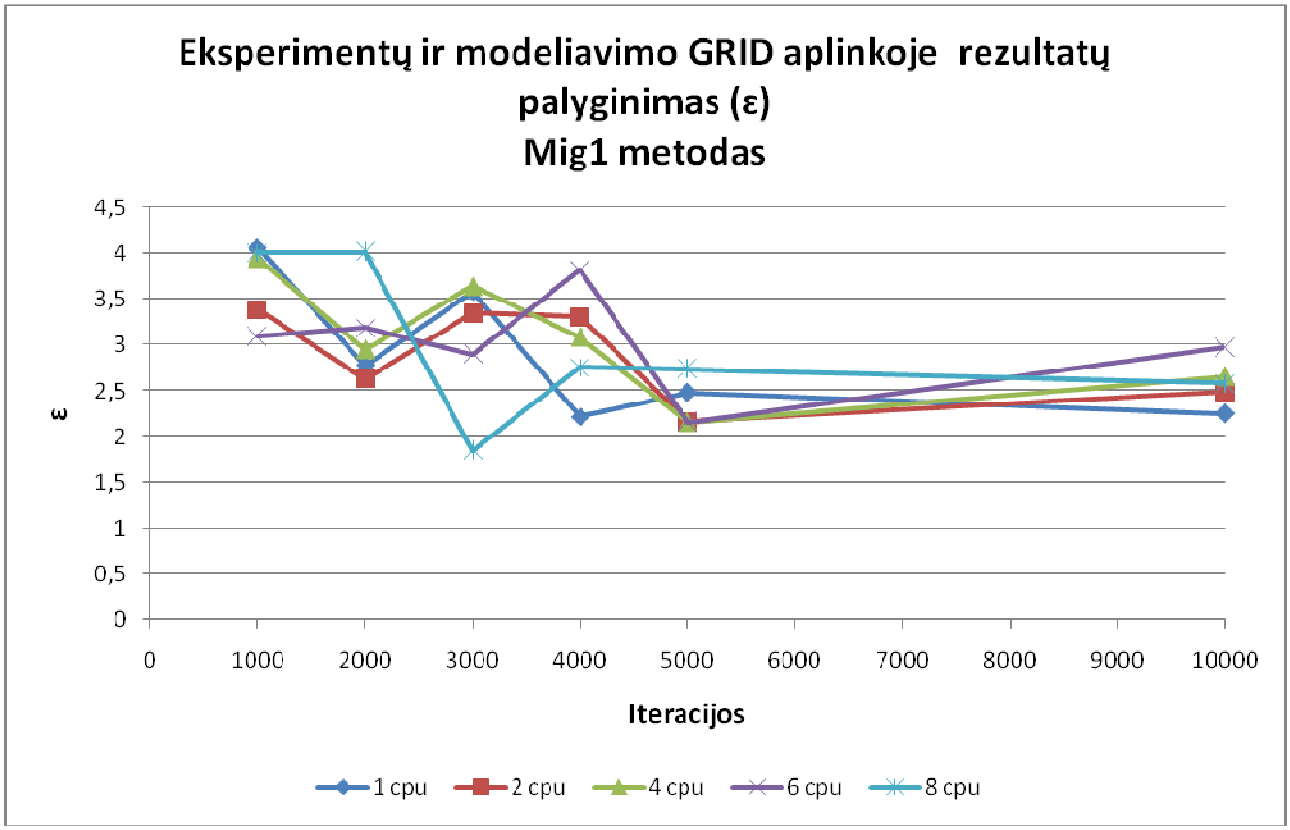


6-1 pav. Eksperimentų ir modeliavimo GRID aplinkoje rezultatų palyginimas (trukmė nuo iteracijų)

Kitoje lentelėje (žr. 6-3 lentelė) surinkti duomenys (skirtumo kvadratas) priklausomai nuo naudotų procesorių skaičiavimams atlikti. Grafiškai pavaizduoti duomenys (žr. 6-2 pav.)

Programos vykdymas	Procesorių skaičius, p	VKP/ Iteracijos					
		1000	2000	3000	4000	5000	10000
Nuosekliai	1	4,065	2,78	3,584	2,22	2,482	2,251
Naudojant MPI biblioteką	2	3,393	2,629	3,35	3,31	2,166	2,486
	4	3,95	2,956	3,644	3,09	2,157	2,655
	6	3,0919	3,189	2,9018	3,8211	2,1577	2,97
	8	4,01	4,018	1,85	2,76	2,74	2,583

6-3 lentelė. Skaičiavimų rezultatai (ε priklausomybė nuo iteracijų skaičiaus)



6-2 pav. Eksperimentų ir modeliavimo GRID aplinkoje rezultatų palyginimas (skirtumo kvadratas nuo iteracijų)

Iš surinktų duomenų galime teikti, kad lygiagretinimas laiko atžvilgiu yra efektyvus, bet jis negarantuoja geresnio rezultato. Kaip ir su nuosekliąja modelio versija, taip ir su lygiagrečiąja, matome, kad metodas gali rasti patenkinamą rezultatą per pirmąjį tūkstantį iteracijų lygiai taip pat sėkmingai, kaip per dešimt kartų ilgesnį iteracijų skaičių. Norėdami įsitikinti ar ši tendencija galioja didiname iteracijų skaičių. Taigi, ženkliai padidinę iteracijų skaičių gavome tokius rezultatus:

Rezultatai	Iteracijos, tūkst.	
	100	1000
Trukmė, h	0,5	5,1
ε	1,45	1,345

*6-4 lentelė.*Rezultatai gauti ženkliai padidinus iteracijų skaičių

Iš lentelėje (žr. 6-4 lentelė) surašytų rezultatų matome, kad lygiagrečioji modelio versija pasitvirtino. Kvadratinė paklaida, tarp eksperimentų ir modeliavimo rezultatų, ženkliai mažėjo. Eksperimentuojant buvo naudojami 8 procesoriai. Po pusvalandžio metodas jau randa rezultatus, kurie teikia vilčių tolimesniems tyrimams.

Rezultatai su Bayes metodu

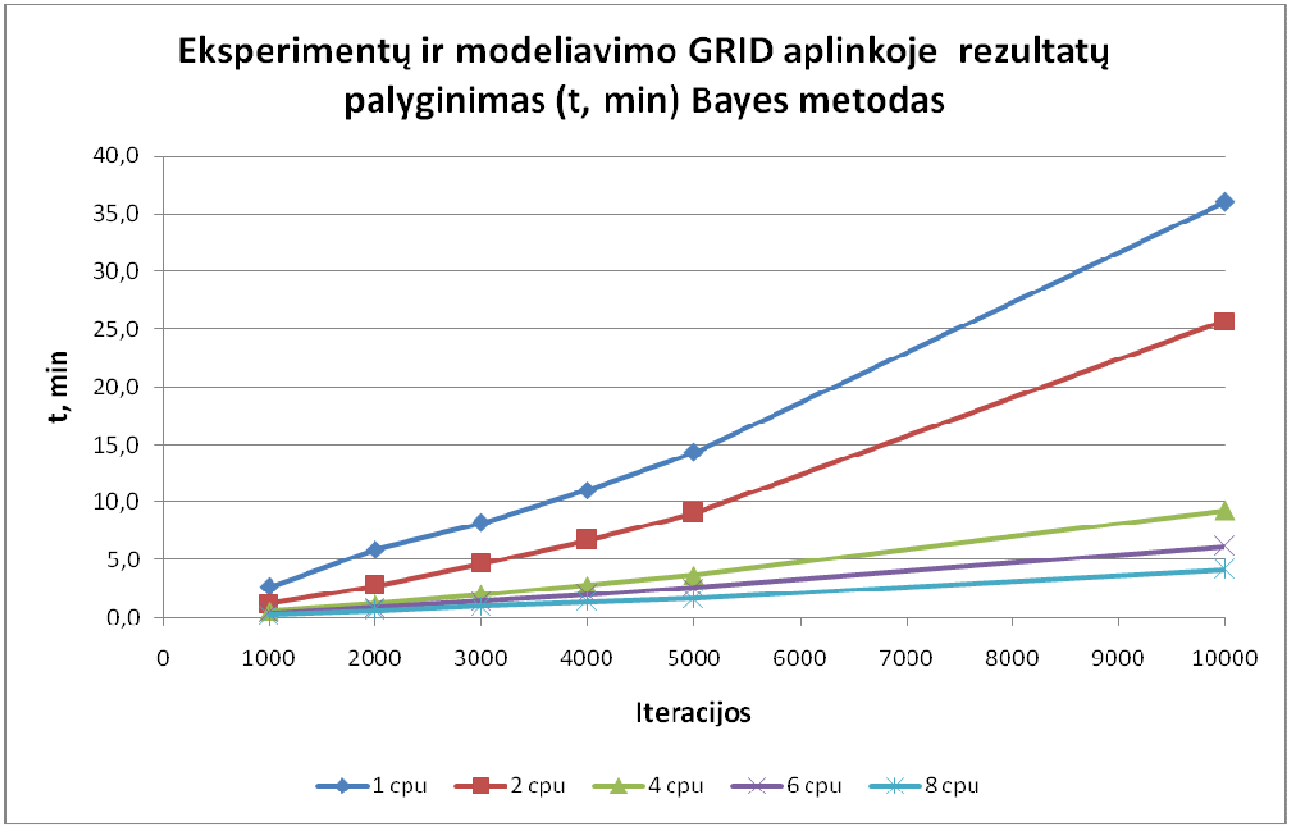
Skaičiavimus atliekant su „Bayes“ metodu buvo naudojami 2-8 procesoriai. Rezultatai gauti didinant iteracijų skaičių, kas 1000 iteracijų, pateikti lentelėje (žr. 6-5 lentelė):

Programos vykdymas	Procesorių skaičius, p	Skaičiavimo laikas s/ Iteracijos						Pagreitėjimas, Sp	Efektyvumas, Ep
		1000	2000	3000	4000	5000	10000		
Nuosekliai	1	163	358	495	666	861	2160		
Naudojant MPI biblioteką	2	76	169	281	404	546	1542	1,72	0,86
	4	36	76	123	172	223	558	3,96	0,99
	6	27	57	89	124	160	373	5,67	0,94
	8	19	40	63	86	100	250	7,70	0,96

6-5 lentelė. Skaičiavimo trukmės priklausomai nuo iteracijos skaičiaus (Bayes)

Paskutinis bandymas atliktas su žymiai didesniu iteracijų skaičiumi. Lentelės duomenys parodė, kad didinant iteracijų skaičių pagreitėjimas mažėja.

Žemiau pateiktame grafike (žr. 6-3 pav.) pavaizduotas eksperimentų ir modeliavimo rezultatų trukmės palyginimas minutėmis.



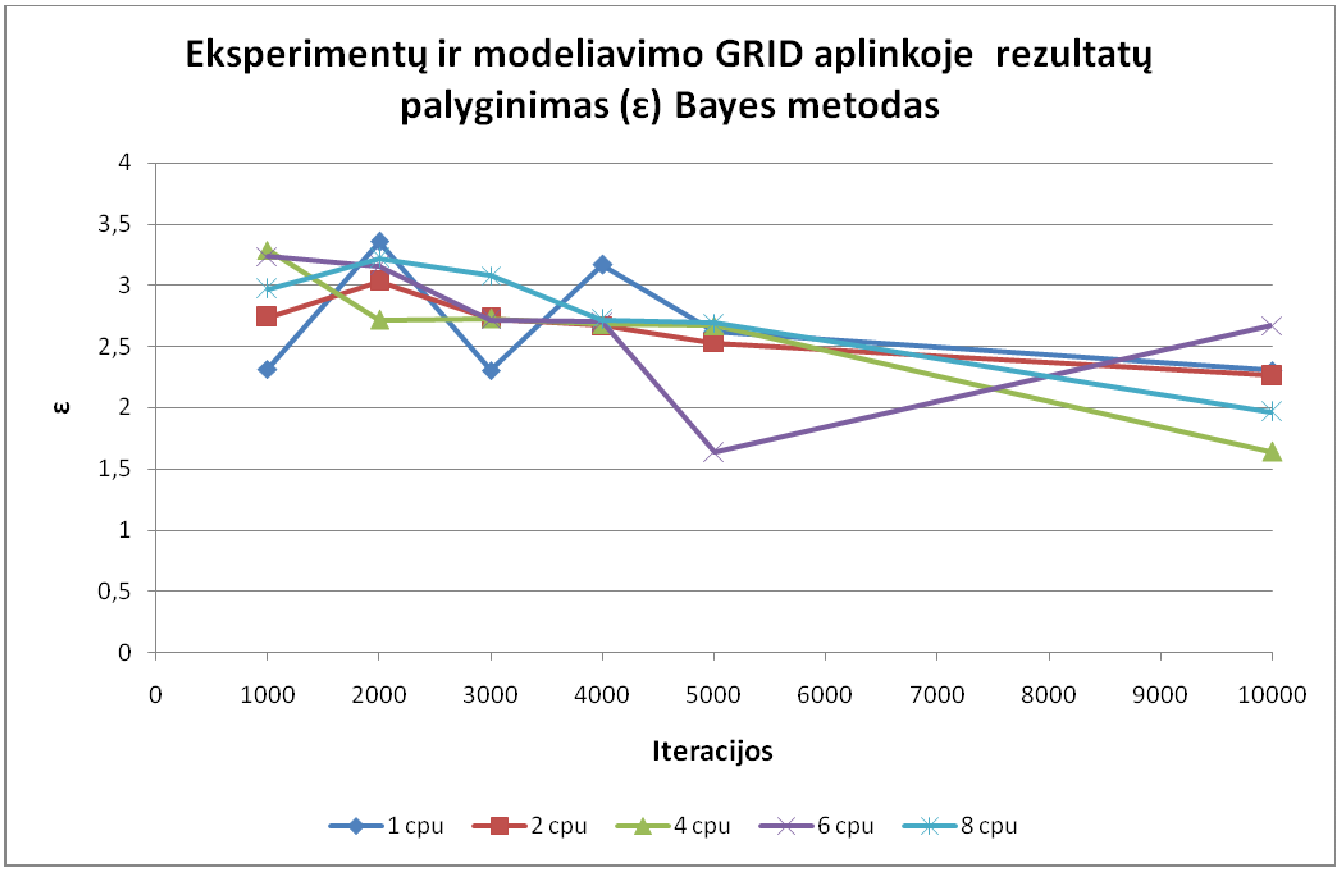
6-3 pav. Eksperimentų ir modeliavimo GRID aplinkoje rezultatų palyginimas (kvadratų skirtumas nuo iteracijų)

Kitoje lentelėje (žr. 6-4 lentelė) surinkti duomenys (skirtumo kvadratas) priklausomai nuo naudotų procesorių skaičiavimams atlikti.

Programos vykdymas	Procesorių skaičius, p	VKP/ Iteracijos					
		1000	2000	3000	4000	5000	10000
Nuosekliai	1	2,314	3,358	2,304	3,166	2,626	2,304
Naudojant MPI biblioteką	2	2,74	3,03	2,73	2,67	2,53	2,26
	4	3,288	2,72	2,729	2,682	2,671	1,6406
	6	3,238	3,156	2,72	2,703	1,632	2,67
	8	2,97	3,219	3,081	2,72	2,69	1,965

6-4 lentelė. Skaičiavimų rezultatai (ε priklausomybė nuo iteracijų skaičiaus, Bayes metodas)

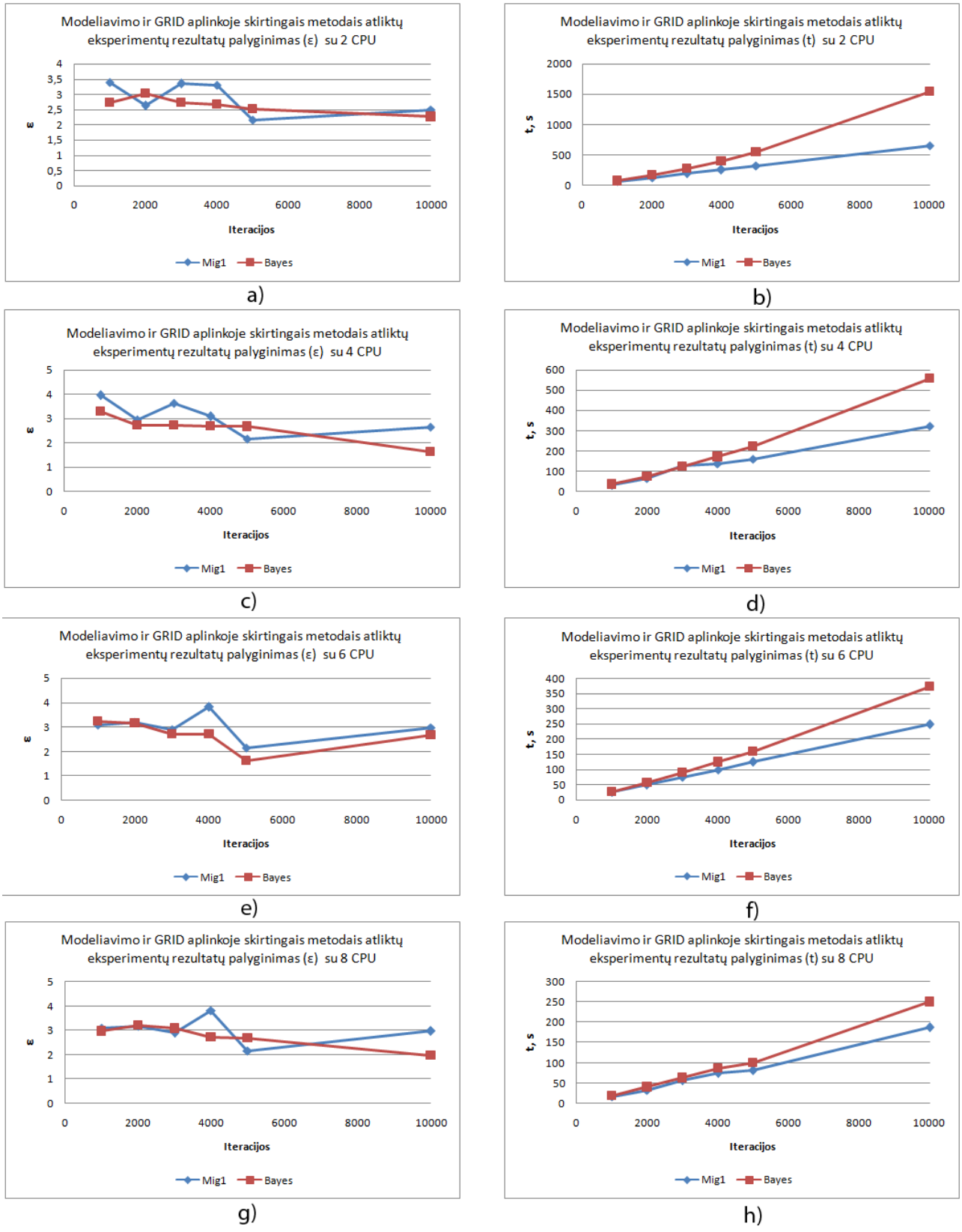
Grafiškai pavaizduoti duomenys (žr. 6-4 pav.) :



6-4 pav. Eksperimentų ir modeliavimo GRID aplinkoje rezultatų palyginimas (trukmė nuo iteracijų)

Iš grafiko (žr. 6-4 pav.) duomenų matome, kad ir šio metodo pagrindu modelis negarantuoja gero rezultato didinant iteracijų skaičiaus. Taip pat pastebime tendenciją, kad gaunamas vis mažesnis skirtumo kvadratas, didinant iteracijų skaičių.

Metodų palyginimas



6-5 pav. Išlygiagretintų modelių,su skirtingais metodais, rezultatų palyginimas, skirtumo kvadrato ir laiko atžvilgiu nuo iteracijų skaičiaus.

a), b) - su 2 CPU; **c), d)** - su 4 CPU; **e), f)** - su 6 CPU; **g),h)** – su 8 CPU

Aukščiau esančiame paveikslėlyje (žr. 6-5 pav.) pavaizduota skirtingų metodų lygiagretaus modeliavimo grid aplinkoje rezultatų palyginimas. Bandymai buvo atliekami su 2 – 8 procesoriais. Lygintas skirtumo kvadratas ir užduoties atlikimo laikas nuo iteracijų skaičiaus.

Paveikslėlio grafikuose (žr. 6-5 pav. a,c,e,g), kur lyginamas kvadratų skirtumas, matome, kad algoritmai išlaiko tendencija, didinant iteracijų skaičių, surasti vis geresnį rezultatą. Taip pat pastebime, kad „Bayes“ algoritmu išlygiagretintas modelis duoda geresnį rezultatą nei ,,Monte Carlo“ metodu atlikti adekvatūs bandymai. Bet, kaip ir analizės dalyje, matome kad ir vienu ir kitu metodu iteracijų didinimas negarantuoja gero rezultato t.y. metodas per pirmą tūkstantį iteracijų gali rasti geresnį rezultatą nei per antrą tūkstantį. Tai įtakoja atsitiktinai parinkti duomenys, kurie ir parodo metodų veikimo principą. Leidžiant daug didesnį iteracijų skaičių, galima pastebėti, kad abu metodai išlaiko tendenciją duoti vis geresnį rezultatą.

Laiko atžvilgiu (žr. 6-5 pav. b,d,f,h) rezultatai geresni „Monte Carlo“ metodo pagrindų atlikti lygiagretūs skaičiavimai. Su mažu iteracijų skaičiumi skirtumas – nežymus, bet didėjant iteracijų skaičiui matoma, kad „Bayes“ algoritmas vis daugiau užtrunka ieškodamas parametrų.

Manome, kad modelio lygiagretinimo plėtojimas yra galimas keičiant kitas lygiagretinimo bibliotekas ir parenkant kitus klasterius su bendros atminties architektūra. Taip pat, jei yra galimybė, skaičiavimams parinkti daugiau procesorių.

7. Išvados

Buvo išnagrinėtas tarpląstelinių plyšinių jungčių 2-jų vartų modelis, bei modelio parametrų optimizavimo algoritmai, kurie maksimaliai leidžia priartinti modeliavimo rezultatus prie realių eksperimentų rezultatų;

Išanalizuoti jau sukurti parametrų optimizavimo moduliai:

- analizė ir bandymai su C# programavimo kalba sukurtu ir realizuotu tarpląstelinės plyšinės jungties modeliu (“Exkor”) - gaunami geriausi rezultatai, bet skaičiavimų trukmė žymiai ilgesnė (lyginant su „Monte Carlo“ metodu, skaičiavimai užtrunka apie 4 kartus ilgiau);
- analizė ir bandymai su Java terpėje sukurtais adekvačiais moduliais („Monte Carlo“ , „Bayes“) - gaunami tarpusavyje panašūs rezultatai, skaičiavimo laikas pastebimai geresnis (lyginant su „Exkor“) , bet gaunamos didesnės kvadratinės paklaidos;

Supaprastintas tarpląstelinės plyšinės jungties modelis. Taip pat perrašytas į C++ programavimo kalbą ir pritaikyta MPI biblioteka lygiagretiems skaičiavimams („Monte Carlo” bei „Bayes” metodams).

Susipažinta su BalticGrid tinklo skaičiavimų paleidimo specifika. Vykdyti skaičiavimų bandymai bei padaryta išsami rezultatų analizė.

8. Literatūra

1. Pranevičius, H. *Sudėtingų sistemų formalizavimas ir analizė*. Kauno technologijos universitetas, 2008. 239 p. ISBN 978-9955-591-51-1;
2. Kirvelis D. *Biofizika*.Vilniaus universiteto leidykla, 2007, p. 296 - 297. ISBN 978-995533-055-4;
3. Verikas, A.; Gelžinis, A. *Neuroniniai tinklai ir neuroniniai skaičiavimai*. Kauno technologijos universitetas, 2008. 241 p. ISBN 978-9955-591-53-5;
4. Žilinskas, A.; *Matematinis Programavimas* [Interaktyvus]. Vytauto didžiojo universitetas, 2005. Prieiga per internetą: [http://www.vdu.lt/ Matematinis Programavimas/#t1];
5. Ye Chen-Izu, Alonso P. Moreno, Robert A. Spangler. Opposing gates model for voltage gating of gap junction channels. 2001;
6. Nerijus, P. *Tarpląstelinių plyšinių jungčių imitacinio modelio kūrimas ir tyrimas.* Magistrinis darbas. Kauno technologijos universitetas 2009;
7. Globalaus optimizavimo pavyzdžiai. Prieiga internete: http://soften.ktu.lt/~mockus/;
8. Diaconis, P. Bayesian numerical analysis. In *Statistical Decision Theory and Related Topics.* [ineraktyvus]. 1988;
9. MOCKUS, J., EDDY, W., MOCKUS, A., MOCKUS, L., REKLAITIS, G. *Bayesian heuristic approach to discrete and global optimization* ;
10. Eidukas D. Elektroninių sistemų metrologija. Leidinys apie šiuolaikines technologijas. 2001;
11. Kruopis J. Matematinė statistika Oficialus matmatikos vadovėlis. 1977;
12. Martinėnas B. Eksperimento duomenų matematinės analizės pagrindai Matematikos ir fizikos vadovėlis. 1999;
13. LitGrid projektas [interaktyvus], 2010; Prieiga per internetą [http://www.litgrid.lt].