ŠIAULIŲ UNIVERSITETAS

Informacinių technologijų katedra

Paulius Mikalauskas

# Pjezoroboto judesio trajektorijų formavimo metodo optimizavimas

Magistro darbas

Vadovė dr. A. Drukteinienė

Šiauliai, 2014

ŠIAULIŲ UNIVERSITETAS

Informacinių technologijų katedra

TVIRTINU

IT katedros vedėja dr. A. Slotkienė
2014-05-27

# Pjezoroboto judesio trajektorijų formavimo metodo optimizavimas

Informatikos inžinerijos magistro darbas

**Autorius**
ITM-12 gr. magistrantas P. Mikalauskas
2014 m. gegužės 27 d.

**Vadovė**
IT katedros lektorė dr. A. Drukteinienė
2014 m. gegužės 27 d.

**Recenzentai**
IT katedros docentė dr. S. Ramanauskaitė
2014 m. gegužės __ d.
IT katedros docentas dr. E. Paliulis
2014 m. gegužės __ d.

Šiauliai, 2014

**UŽDUOTIES LAPAS**

## SANTRAUKA

### Pjezoroboto judesio trajektorijų formavimo metodo optimizavimas

Šių laikų visuomenę neabejotinai galime vadinti informacine, nes mūsų aplinkoje kiekvieną dieną atsiranda vis naujų įrenginių, be kurių sunkiai įsivaizduojame savo kasdieninį darbą. Tam labiausiai įtakos turėjo vienas svarbiausių žmonijos išradimų – asmeninis kompiuteris. Taigi naujų įrenginių kūrimas, jų tobulinimas bei pritaikymas plačiam naudojimui – viena pagrindinių šių dienų inžinerijos mokslo krypčių.

Pirmame skyriuje apžvelgiama, kas yra pjezorobotas, kaip jis veikia, kaip formuojamos judesio trajektorijos. Antras skyrius skirtas apžvelgti programinio kodo našumo svarbai. Apibudinami tyrimo kriterijai, eiga, resursai, bei pateikiami neoptimizuoto pjezoroboto judesio trajektorijos formavimo liestinių metodu rezultatai ir jų interpretavimas. Trečiame skyriuje nagrinėjami programinio kodo našumo charakteristikų optimizavimo būdai. Ketvirtame skyriuje pateikiama pjezoroboto judesio trajektorijos formavimo liestinių metodu programinio kodo analizė, siūlomos lygiagretinimo galimybės, pateikiami programinio kodo vykdymo laiko pokyčiai ir optimizuoto metodo efektyvumo įvertinimas.

Mokslinė problema – neoptimizuotas pjezoroboto trajektorijos formavimo algoritmas laiko atžvilgiu, kuris priklauso nuo trajektorijos ilgio bei vingiuotumo.

Ištyrus pjezoroboto trajektorijos formavimo liestinių metodu algoritmą, buvo identifikuota probleminė vieta, kuriai buvo sukurti keli lygiagretinimo būdai bei pasiūlytas efektyviausias problemos sprendimo būdas.

## SUMMARY

### Optimization of motion planning method of piezorobot

These times can certainly be called information era, because we see new devices in our environment every day, which is hard to imagine without their daily work. Basically it is due to one of the most important inventions of humanity - the personal computer. Thus, the creation of new facilities and their improvement and mainstreaming - one of the main current directions of engineering education.

The first chapter provides an overview of what is piezorobot and how it works, how motion trajectories are tuned. The second chapter is devoted to an overview of the importance of productivity software code. Criteria, process and recourses of the studies are described in this chapter, as well as present optimization of piezorobot motion trajectories forming tangents method results and their interpretation. The third chapter explores optimization techniques the characteristics of software code performance. The fourth chapter provides programming code analysis of piezorobot motion trajectories forming tangents method of the proposed parallelization opportunities. This chapter also describes the program code and run-time changes in the method of optimized efficiency rating.

The scientific problem - optimization of trajectory shaping algorithm of piezorobot time wise, which depends on the path length and geometry.

Problem areas of forming tangents method algorithm of piezorobot were identified, a few new methods were created and the most effective solution to the problem was proposed in this work.

# TERMINŲ IR SANTRUMPŲ ŽODYNĖLIS

**CCD -** prietaisas judantis nuo elektros krūvio

**MEMS -** labai mažų įrenginių technologija

**CPU -** procesorius

**MB -** megabaitai

**MATLAB -** daugiaplatformė programinė įranga, skirta įvairių mokslo šakų problemoms spręsti

**HDD -** kietasis diskas

**RAM -** kompiuterio operatyvioji atmintinė

**OS -** operacinė sistema

**ROM -** atminties tipas, kuris pasiekiamas tik skaitymui

**GB -** informacijos kiekio matavimo vienetas

**SATA -** atminties įrenginių prijungimo prie kompiuterio standartas

**DDR2 -** kompiuterio darbinės atminties tipas

**DDR3 -** kompiuterio darbinės atminties tipas

**OpenMP -** programavimo standartas, skirtas realizuoti lygiagretiesiems algoritmams

**MPI -** standartas, naudojamas paskirstytosios atminties lygiagrečiuosiuose kompiuteriuose

**GPU -** grafikos procesorius

**SPMD -** lygiagrečios programos tipas

**TURINYS**

# ĮVADAS

Pjezorobotas, kurių veikimas pagrįstas aukšto dažnio virpesių žadinimu, nėra plačiai paplitęs, nes šio tipo įrenginiai yra išrasti palyginti nesenai ir nepritaikytas plačiam naudojimui dėl ilgai generuojamų judesio trajektorijų. Paprastai judesio trajektorijos yra laužytos, todėl tai apsunkina jų generavimą. Tad natūralu, kad algoritmas, ilgai skaičiuojantis sudėtingas lygčių sistemas, turi būti optimizuotas. Būtent dėl šios priežasties ir yra reikalingas pjezoroboto judesio trajektorijos liestinių metodo optimizavimas.

**Tyrimo objektas** – pjezoroboto judesio trajektorijos formavimo liestinių metodas.

Darbe keliamas **tikslas** – pjezoroboto judesio trajektorijos liestinių metodu formavimo optimizavimas, panaudojant lygiagrečius skaičiavimus.

Tikslo įgyvendinimui keliami šie uždaviniai:

1. Susipažinti su judesio trajektorijos formavimo būdais ir raidos ypatybėmis;
2. Išanalizuoti MATLAB R2013b paketo, bei kitų programavimo kalbų lygiagretaus programavimo galimybės;
3. Išanalizuoti pjezoroboto judesio trajektorijos formavimo liestinių metodą;
4. Nustatyti analizuoto metodo trūkumus bei pasiūlyti kaip jį tobulinti;
5. Sukurti patobulintą metodą ir atlikti jo tyrimą.

Darbe planuojama naudoti literatūros apžvalgą, lyginamąją analizė, sisteminę, skaitinę ir eksperimentinę analizes.

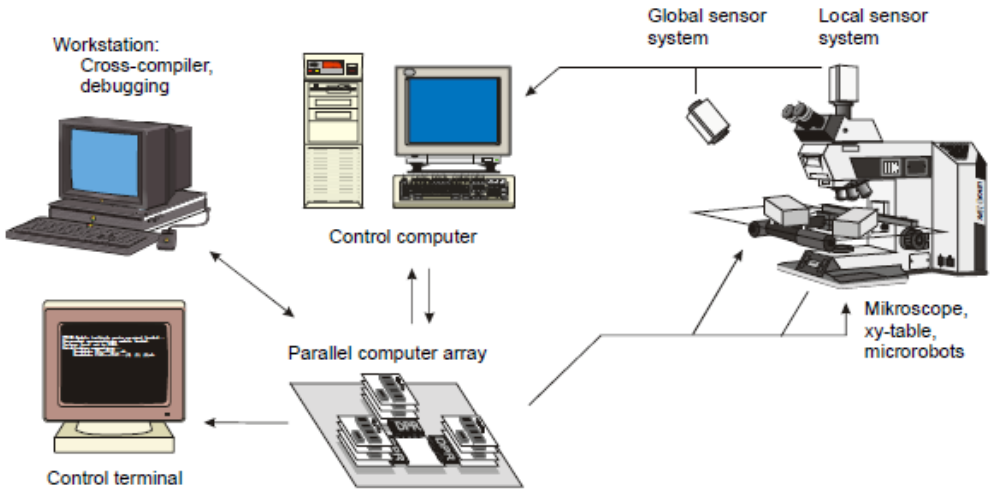
# 1. PJEZOROBOTAI IR JŲ JUDESIO TRAJEKTORIJŲ FORMAVIMAS

## 1. 1 Mechatroninių įrenginių apžvalga

Daugelį metų valdymo sistemų inžinieriai siekia padidinti ar pakeisti mechaniniu sistemų efektyvumą. Daugeliu atveju tokių sistemų funkcionalumo palaikymas negalimas be elektronikos. Tobulėjant apdorojimo sistemoms ir didėjant programinės įrangos kūrimo įgūdžiams, susidarė galimybė kurti mechatroninius įrenginius, kuriuose apjungiamos įvairios mokslo sritys: mechanika, elektronika, informatikos inžinerija, medžiagų inžinerija. Sistemų teorija, valdymo teorija ir informacijos teorija yra technologijų integravimo tarp šių sričių metodikos. [7]

Analizuojant mokslinę literatūrą buvo susipažinta su daug mechatroninių įrenginių, kurių paskirtis gali būti labai įvairi. Nuo medicininės paskirties iki karo paskirties mechatroninių įrenginių.

Kompiuterio valdymas pasitelkiant pjezo mikro robotą. Automatikos manipuliacijos procesas reikalauja universalaus roboto ir pažangios kontrolės sistemos. Šis mikro robotas buvo sukurtas tarpdisciplininės mokslinių tyrimų grupės Karlsruhe universiteto, kuris gali judėti 1 μm tikslumu. Toks tikslumas buvo pasiektas panaudojus naują valdymo principą. Šis robotas yra trikampio formos ir turi tris kojas. Šios kojos pagamintos iš pjezokeraminio vamzdžio ir keturių išorinių elektrodų. Paduodant tam tikrą įtampą tarp vidinių ir išorinių elektrodų vamzdis gali judėti bet kuria kryptimi, priklausomai nuo įtampos pokyčių tarp elektrodų. Labai mažas roboto poslinkis pasiekiamas lenkiant visas tris kojas ta pačia kryptimi. Jeigu įtampa staiga pasikeitė į priešingą, kojos ims lenktis ir robotas pradės judėti priešinga kryptimi. Kartodami šiuos veiksmus robotas gali nujudėti pakankamai didelį atstumą, o didžiausias judėjimo greitis – keletas milimetrų per sekundę. Šis robotas yra pagrindinis komponentas mikro pozicionavimo, kuri atvaizduojama žemiau esančiame paveiksle. [8]

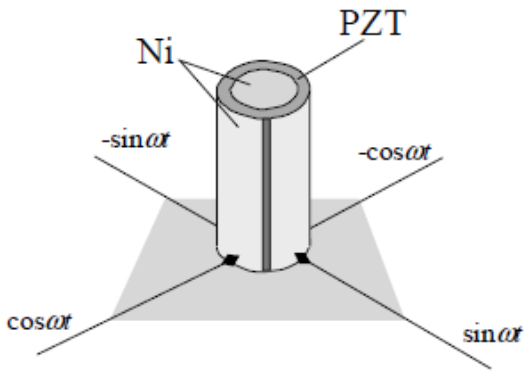



***1 pav.*** *Mikro judesio stotis [8]*

Dėl savo mažo dydžio robotas neturi jokių integruotų daviklių. Jo globali padėtis nustatoma globalaus jutiklio, CCD kameros. Vartotojas gali stebėti monitoriuje, kaip robotas juda.

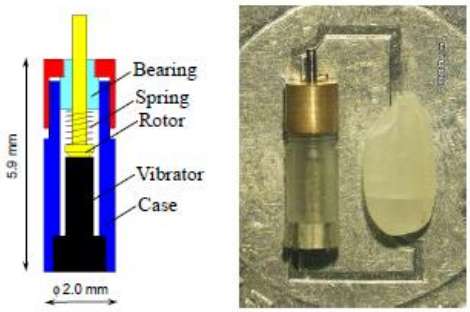
Kitas nagrinėtas įrenginys tai – mikro ultragarsinis variklis, didžiąją dalį įrenginio sudarantis pjezoelektrinis keitiklis. Tai cilindrinės formos pjezoelektrinis vibratorius, kurio diametras 0.8 mm, o aukštis 2.2 mm [9]. Mikro ultragarsinis variklis buvo varomas kintančių bangų nuo cilindro vibratoriaus

pabaigos. Žemiau esančiame paveiksle schematiškai pavaizduotas pjezoelektrinis cilindro keitiklis šiam varikliui. Šis cilindrinės formos pjezoelektrinis vibratorius buvo sukurtas kaip statoriaus keitiklis.



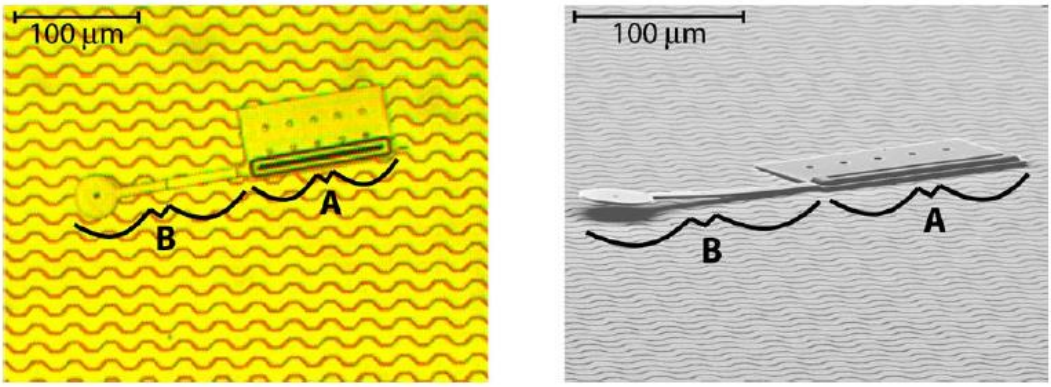

***2 pav.*** *Cilindrinio mikro ultragarsinio variklio vaizdas [9]*

Kaip ir parodyta (žr. 2 pav.) statoriaus keitiklis tvirtinimas prie cilindro galo. Taigi, taip lengviau laikyti vibratorių ir lengva sumažinti ultragarsinį variklį. Ant cilindrinio tipo vibratoriaus paviršiaus išviso yra keturi elektrodai. Vidinis elektrodas veikia kaip lygiavertis žemei. Keturi elektros šaltiniai naudojami virpesiams išgauti ir tai buvo fundamentali vibratoriaus judėjimo priežastis. Sukimosi ar judėjimo kryptį galima keisti pakeitus fazės poslinkį tarp elektros šaltinių (žr. 3 pav.) rodo, kad pjezo elektrinis keitiklis, kuris buvo naudojamas kaip statinis ultragarsiniam varikliui.

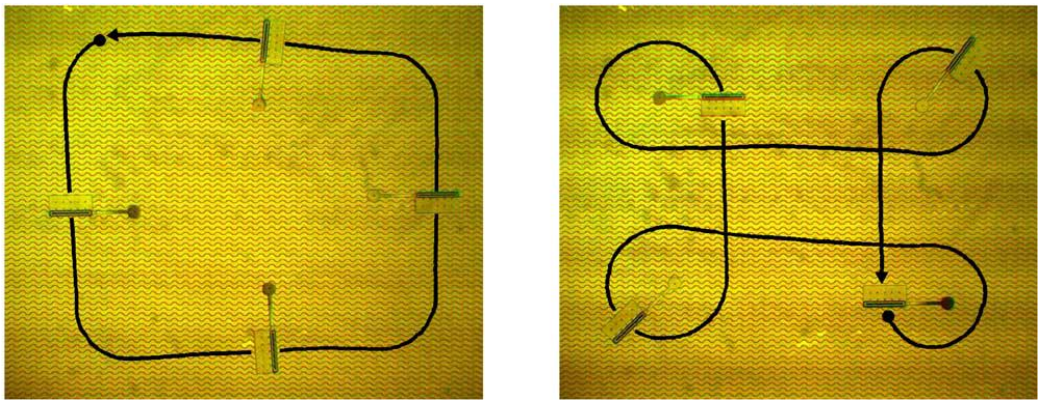



***3 pav.*** *Mikro variklio skersinis pjūvis [9]*

MEMS mikro robotas, kurio matmenys 60 µm x 250 µm x 10 µm susideda iš lenktos vairalazdės, sumontuotos ant nepriklausomos pavaros įrenginyje (USDA). Šie du komponentai pagaminti monolitiškai, iš to paties lapo elektrai laidžios medžiagos, kuri gali gauti bendrą galią ir valdymo signalus per tarpinę movą elektros tinklu. Visos roboto dalys gauna tą pačią galią ir valdymo signalą, todėl prietaisas gali būti valdomas, be žinių, apie savo pradinę judėjimo poziciją. Anksčiau minėti valdikliai suteikia dvi skirtingas judėjimo kryptis, tai judėjimas į priekį ir sukimasis. Šios judėjimo galimybės leidžia visiškai padengti numatytą mikro roboto darbo sritį. Šie MEMS mikro robotai pasižymi labai maža judėjimo paklaida. Remdamiesi bandymais mokslininkai ištyrė, kad šis robotas gali nujudėti iki 35 cm be jokių klaidų [10].




***4 pav.*** *MEMS mikro robotas [10]*



***5 pav.*** *MEMS mikro roboto pavyzdinės judėjimo trajektorijos [10]*

Apžvelgus ankščiau minėtus įrenginius galima teigti, kad visi jie pagrįsti įtampos keitimo pagrindu, norint, kad robotas atliktu vienokį ar kitokį judesį. Tai yra judėtu pirmyn ar atgal, suktųsi aplink savo ašį ar atliktu kitokius iš anksto numatytus veiksmus. Jų gali būti įvairių formų bei paskirties. Šiuo metu daug mokslininkų dirba šia linkme, o mikro robotai, pagrįsti įtampos keitimu turi didelę perspektyvą ateityje.

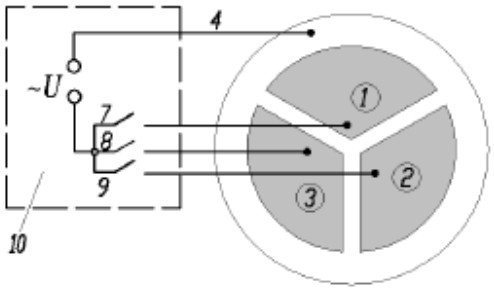
## 1. 2 Pjezoroboto judesio trajektorijos formavimas

### *1. 2. 1 Pjezoroboto ir judesio trajektorijų formavimo metodų apžvalga*

Šiuo metu jau turimi veikiantys skirtingos konstrukcijos pjezorobotai, tačiau metodų, skirtų jų trajektorijų planavimui nėra daug. Didžiausią darbą šioje srityje atliko A. Drukteinienė, G. Kulvietis, R. Bansevičius ir kt., kurie pristatė keletą metodų, skirtų trijų kontaktų pjezoroboto judesio trajektorijos formavimui. Šis pjezorobotas pasižymi itin dideliu tikslumu, mažu dydžiu bei geromis valdymo savybėmis [16]. Žemiau pateikiamas paveikslas, kaip gali atrodyti pjezorobotas (žr. 6 pav.).

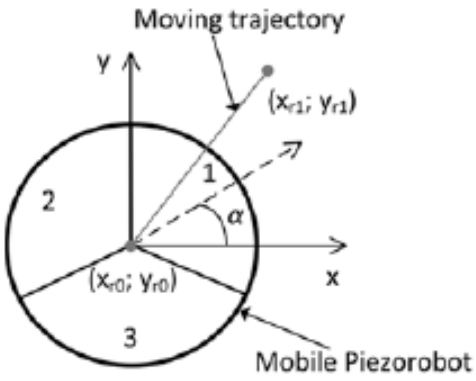

***6 pav.*** *Pjezorobotai [16, 17, 7]*

Pjezorobotus galima suskirstyti į žiedinius, hemisferinius ir cilindrinius [7]. Visų išvardytų įrenginių veikimo principas yra toks pat. Naudojamos dvi žadinimo schemos (žr. 7 pav.).



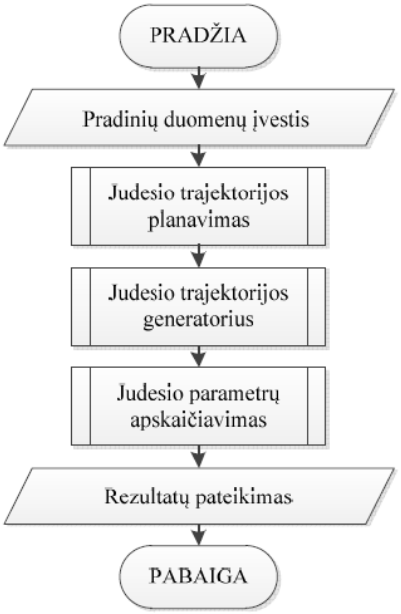

***7 pav.*** *Elektrodų jungimo schema [17]*

Kaip matyti iš žemiau esančio paveikslėlio (žr. 8 pav.), trijų kontaktų cilindrinis pjezorobotas yra sudarytas iš trijų dalių, kurios pažymėtos skaičiais nuo vieno iki trijų, dar kitaip vadinamos elektrodų segmentais. Judėjimo kryptis priklauso nuo aktyvaus segmento. Jie padeda judėti pjezo robotui, pasisukdami tam tikru, reikiamu kampu, paveiksle pažymėta kaip judesio trajektorija.



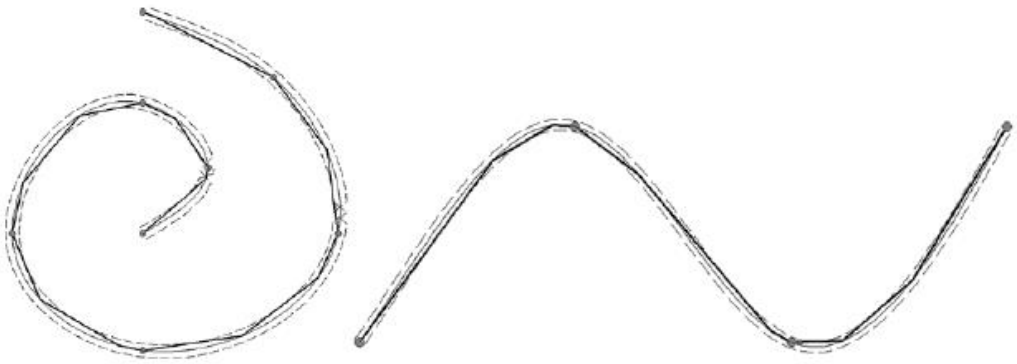

***8 pav.*** *Pjezoroboto konstrukcija [18]*

Trijų kontaktų pjezorobotai turi kelis trajektorijų formavimo algoritmus. Bendrasis trajektorijos formavimo algoritmas pateiktas žemiau esančiame paveiksle (žr. 9 pav.).




***9 pav.*** *Bendras trajektorijos formavimo algoritmas [7]*

Kaip matyti iš pateikto paveikslo (žr. 9 pav.), visų pirma pjezorobotas turi turėti pradinius duomenis, kuriais judės. Sekantis žingsnis aprašomas kaip pačios trajektorijos suplanavimas iš gautų duomenų. Po to su trajektorijos generatoriumi sugeneruojama trajektorija ir galiausiai apskaičiavus visus parametrus pjezorobotas juda iš anksto nustatyta kreive. Gautas rezultatas – tai pjezo roboto teisingas judėjimas iš pasirinkto pradinio taško į galutinį. Žemiau pateikiami suplanuotos trajektorijos pavyzdžiai (žr. 10 pav.).

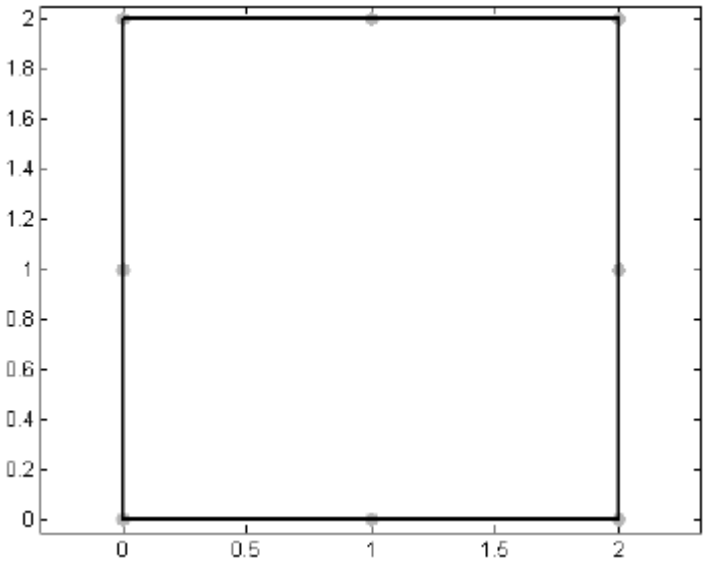


***10 pav.*** *Trajektorijų pavyzdžiai [18]*

Trajektorijos planavimo veiksmai [7]:

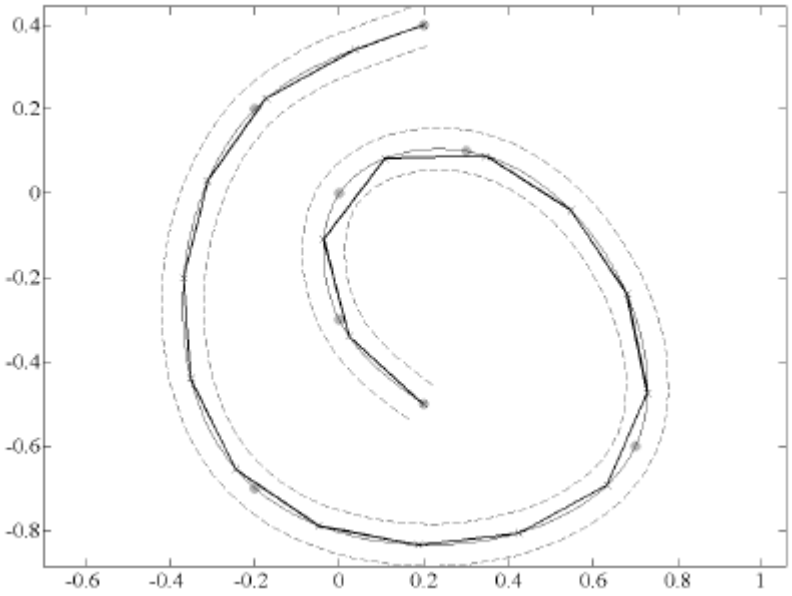
1. Pradinė roboto padėtis;
2. Laikas, per kurį turi būti atliekami tam tikri veiksmai;
3. Veiksmai atliekami tam tikru laiko momentu keičiantys būseną;
4. Galutinė pjezoroboto padėtis.

Trajektorijos formavimas nuo taško prie taško metodu tinka formuoti judesio trajektoriją tik tuo atveju, kai tarp mazgų atliekama tiesinė interpoliacija. Tiesinė interpoliacija yra zoninė pirmo laipsnio polinomo interpoliacija, kur jungtys tarp gretimų zonų sutampa su duomenų taško padėtimi. Šį metodą geriausia taikyti, kai suplanuota trajektorija yra vienoje tiesėje (žr. 11 pav.). [7]




***11 pav.*** *Suformuota trajektorija nuo taško prie taško metodu [7]*

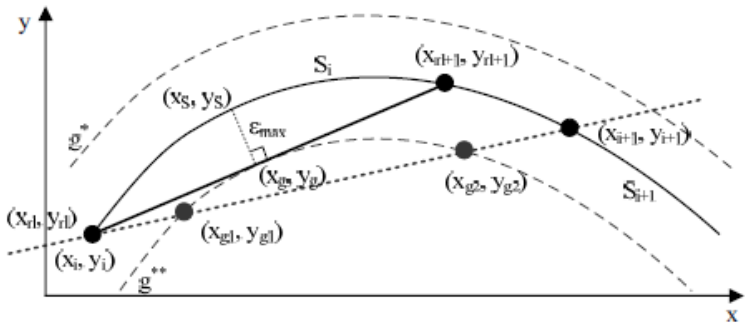
Kontrolinių taškų metodo principas – suskaidyti suplanuotą trajektoriją į vienodo ilgio atkarpas, o taškuose jungiančiuose šias atkarpas atlikti posūkio tikrinimą. Analizuodamas autorius pastebėjo, kad nuokrypis nuo suplanuotos trajektorijos gali būti įvairus. Tai priklauso nuo trajektorijos kreivio. [7]




***12 pav.*** *Suformuota trajektorija kontrolinių taškų metodu [7]*

Liestinių metodas yra labai panašus į metodą nuo taško prie taško, skirtumas tik tas, kad šio metodo suplanuota trajektorija yra kreivė. Šio metodo autorius pastebi, kad liestinių metodas labiau tinka formuoti judesio trajektorijas, kai iš anksto žinoma kreivė, bet gali būti naudojamas ir su tiesėmis. Kadangi šio metodo matematinis modelis yra sudėtingesnis, todėl nepatartina naudoti, kai formuojama trajektorija yra tiesė [7]. Tai užims daugiau laiko, nei taikant metodą, pritaikytą trajektorijoms, kurių pagrindas yra tiesės.

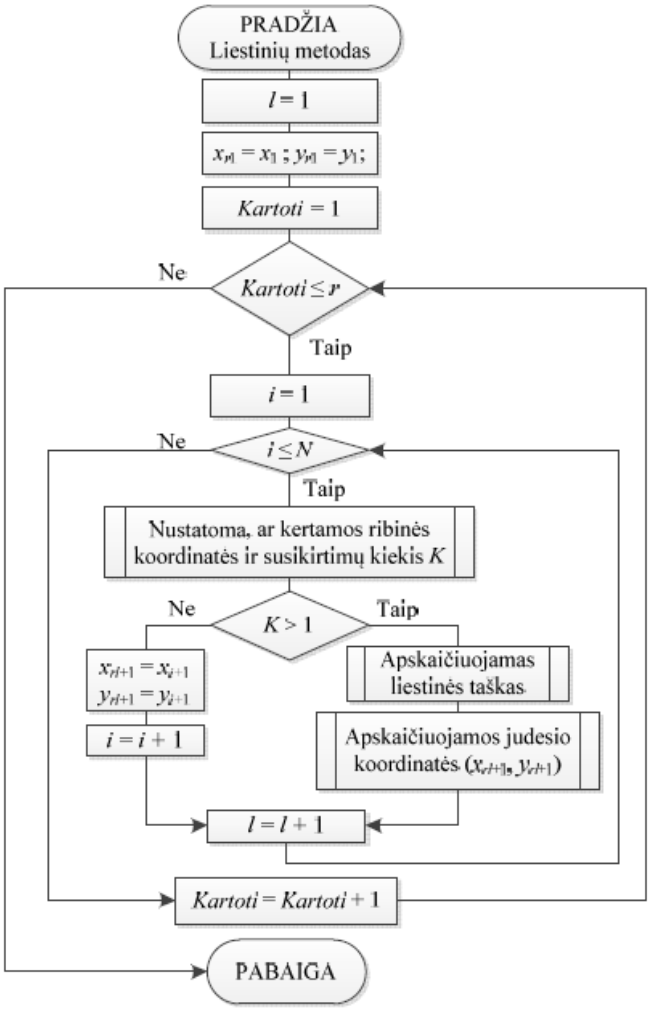
Žemiau esančiame paveiksle (žr. 13 pav.) pateiktas atkarpos skaičiavimo principas, kuris apibudina vienos atkarpos skaičiavimą liestinių metodu.




***13 pav.*** *Atkarpos skaičiavimo principas liestinių metodu [7]*

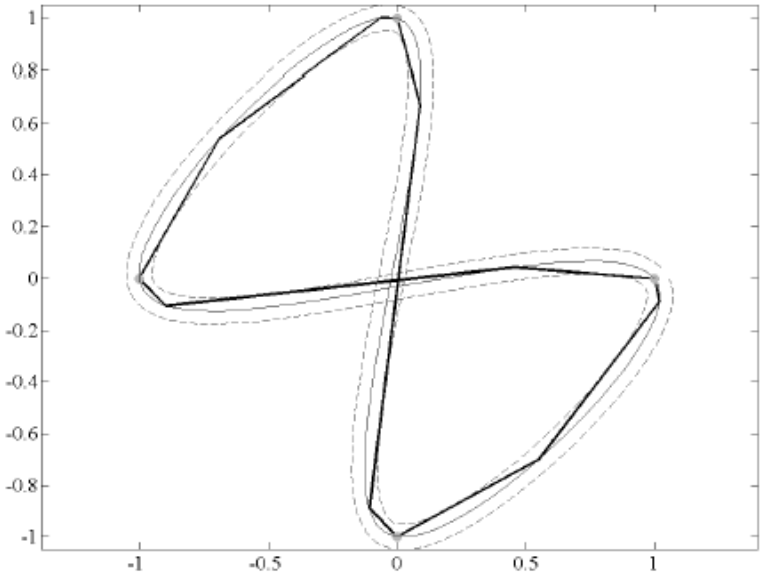
Norint apskaičiuoti atkarpą pirmiausia reikia patikrinti ar nekertamos ribinės koordinatės g* ir g**. Tikrinant ar nekertamos ribinės koordinatės apskaičiuojami susikirtimo taškai. Jei apskaičiavus gaunamas 0, nekertama nei viena ribinė koordinatė, jei 1, tuomet judesio tiesė eina tarp mazgų ribinėje koordinatėje, o jei gaunamas daugiau nei 1, vadinasi kertama ribinė koordinatė ir judėti negalima. Ieškant liestinės taško ribinėse koordinatėse, sprendžiamos dvi lygčių sistemos su kiekviena ribine

koordinate. Suradus tinkamą tašką, apskaičiuojamos judesio koordinatė esanti suplanuotoje trajektorijoje.




***14 pav.*** *Liestinių metodo blokinė schema [7]*

Pagal liestinių metodą yra sukurtas MATLAB kodas, kuris ir atlieka visus numatytus skaičiavimus, o gavęs suformuotą trajektoriją, ją atvaizduoja grafiškai (žr. 15 pav.).




***15 pav.*** *Suformuota trajektorija liestinių metodu [7]*

# 2. PJEZOROBOTO JUDESIO TRAJEKTORIJOS FORMAVIMO ANALIZĖ

## 2. 1 Programinio kodo našumas ir jo svarba

Žmonės nuo seno viską skaičiuoja ir skaičiavimui pasitelkia įvairia skaičiavimo priemones. Per sąlyginai nedidelį laiko tarpą, šioje sferoje žmonės padarė didelį šuolį nuo akmenukų skaičiavimo iki sudėtingų, šiuolaikiškų skaičiavimo įrenginių, dar vadinamu CPU (Central Processing Unit). Šiuolaikiniuose kompiuteriuose dažniausiai yra ne po vieną CPU, todėl galima išgauti dar didesnį skaičiavimų našumą, o nuolat sudėtingėjančios programos reikalauja vis daugiau sisteminių resursų, kurių trūkumas gali sukelti kritinių problemų arba visiškai nutraukti programos darbą. Tad norint kuo veiksmingiau išnaudoti sisteminio bloko resursus, pirmausią reikėtų naudoti kuo mažiau programinės įrangos resursų. Vienas programos resursų naudojimo mažinimo būdų yra atsižvelgti į programos vykdomą kodą ir kaip galima geriau jį optimizuoti. Tai padėtų ne tik veiksmingiau naudoti sistemos resursus, bet ir suteikti programai stabilumo, paspartėtų programos veikimas [13].

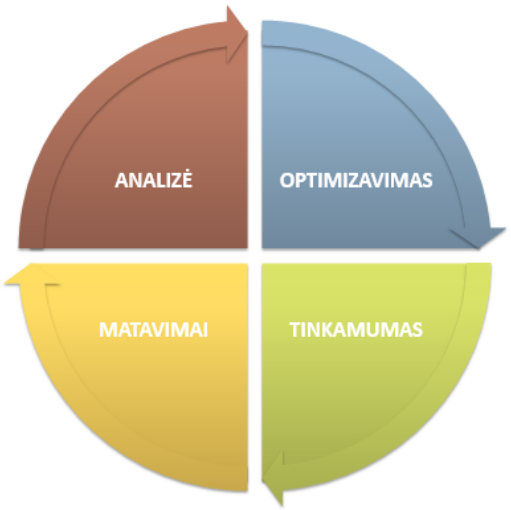
Programinio kodo našumo tyrimas yra aktuali tema tiek mokslininkams, kurie siekia optimizuoti savo siūlomus algoritmus ar naujai kuriamas kalbas, tiek ir pramonės atstovams, kurie suinteresuoti gauti kuo greičiau veikiančią, mažiau sistemos resursų naudojančia sistemą [13].

Kuriant sudėtingas ir daug skaičiavimų atliekančias programines sistemas, programose reikia atkreipti dėmesį į vykdymo laiką ir atminties sunaudojimą. Kuo veiksmingiau bus išnaudojama sisteminio bloko charakteristika, tuo spartesnis ir veiksmingesnis bus programos vykdymo laikas ir atminties užimtumas. Vienas iš programos greitaveikos efektyvumo didinimo būdų yra programinio kodo optimizavimas [13].

Pjezorobotas vienas savaime judėti negali, todėl jam yra sukurta keletas metodų, kurių pagrindu yra formuojamos judesio trajektorijos. Taigi, skaičiuojant sudėtingas lygčių sistemas, kurios užima pakankamai daug laiko ir šiuo metu negali būti taikomos praktikoje, nes trajektorijos skaičiavimas yra ilgas. Todėl skaičiavimus reikia optimizuoti ir priversti pjezorobotą judėti kaip galima greičiau.

$$\begin{cases} \dfrac{\partial S}{\partial t}\left(x_g^{*}-x_i\right)=\left(y_g^{*}-y_i\right); \\ x_g^{*}=g_x^{*}\left(S_{xi}\left(t_S\right),\varepsilon_{\max}\right); \\ y_g^{*}=g_y^{*}\left(S_{yi}\left(t_S\right),\varepsilon_{\max}\right). \end{cases} \qquad \begin{cases} \dfrac{\partial S}{\partial t}\left(x_g^{**}-x_i\right)=\left(y_g^{**}-y_i\right); \\ x_g^{**}=g_x^{**}\left(S_{xi}\left(t_S\right),\varepsilon_{\max}\right); \\ y_g^{**}=g_y^{**}\left(S_{yi}\left(t_S\right),\varepsilon_{\max}\right). \end{cases}$$

Paveiksle (žr. 16 pav.) pavaizduotas iteracinis procesas, kuris apima matavimus, analizę, optimizavimą ir optimizavimo tinkamumą. Toks ciklas leidžia įvardyti ar norimas tikslas pasiektas. Tai vertinga informacija, nes gavus norimus rezultatus, galime įvardyti, kad optimizacija baigta. [15]




***16 pav.*** *Optimizavimo procesas*

### 2. 2 Tyrimo kriterijai

Tyrimo metu norima stebėti įvairias našumo charakteristikas, kurios aprašytos 1 lentelėje.

*1 lentelė. Tyrimo kriterijai*

| **Kriterijaus pavadinimas** | **Matavimo vienetas** | **Kriterijaus svarba ir pagrindimas** |
|---|---|---|
| Kodo vykdymo laikas | s | Kodo vykdymo laikas yra būtinas norint įvertinti ar programa gebės per tam tikrą laiką įvykdyti tam tikrą užduotį. Šis kriterijus svarbus realaus laiko sistemose, kur prieš pasirenkant tam tikrą kodą yra vertinamas galimas blogiausias vykdymo laikas ir tokiu atveju nustatoma ar šis kodas tiks naudoti, ar bus per lėtas, kad aptarnautų visus vartotojus per numatytą laiko tarpą. |
| RAM naudojimas | MB | Priklausomai nuo skaičiavimo pobūdžio, darbinė kompiuterio atmintis yra labai svarbus veiksnys, galintis turėti įtakos skaičiavimams. |
| CPU naudojimas | % | Procesorius yra pagrindinis kompiuterio skaičiavimo įrenginys, todėl jo apkrova tiesiogiai priklauso nuo programinio kodo vykdymo laiko. |

### 2. 3 Tyrimo eiga

Norint tinkamai išanalizuoti, identifikuoti ir pasiūlyti tinkamą problemos sprendimą reikalingi tam tikras veiksmų planas. Tyrimui buvo naudojama:

- MATLAB R2013b programinė įranga;
- Du skirtingi kompiuteriai;
- Pjezoroboto judesio trajektorijos formavimo liestinių metodu MATLAB failai.

Įdiegus į kompiuterius MATLAB paketą buvo analizuota, kaip veikia kodas, metodo struktūra bei kur būtų galima įterpti papildomą kodą, kuris, paleidus liestinų metodą fiksuotų kiek laiko buvo vykdomas trajektorijos formavimas. Tam puikiai pasitarnavo integruotos tic() ir toc() funkcijos, kurių pagalba buvo fiksuotas vykdymo laikas. Kitas įrankis, kurio pagalba galima gauti informaciją apie visas įvykdytas funkcijas, jų laiką ir apkrautumą yra „Run and time“ mygtukas. Jo pagalba galima identifikuoti kokiose vietose programa ilgiausiai užtrunka. Taip buvo identifikuotos probleminės vietos. Išanalizavus gautus duomenis, buvo sprendžiama kaip būtų galima išspręsti iškilusias problemas.

### 2. 4 Tyrimo resursai

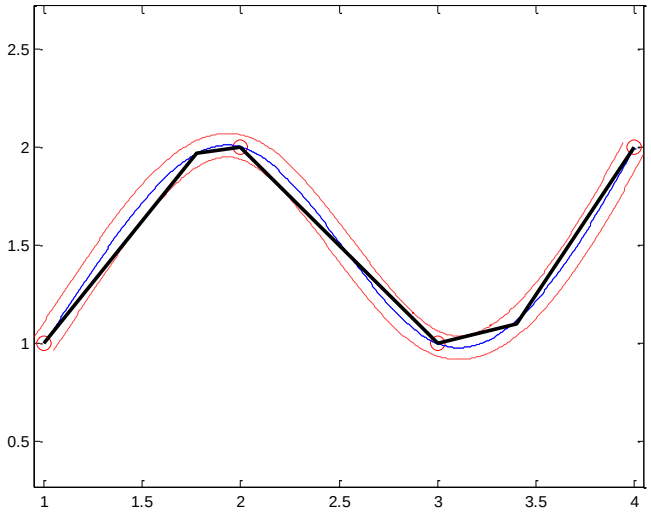
Testavimo resursai:

- Pavadinimas: 1 kompiuteris;
- CPU: AMD Athlon X2 5000+ 2.60 GHz (2 branduoliai);
- HDD: 150 GB SATA;
- RAM: 2 DDR2 GB;
- OS: Windows 7 Ultimate 64-bit;
- Programinės įranga: MATLAB R2013b 64-bit;

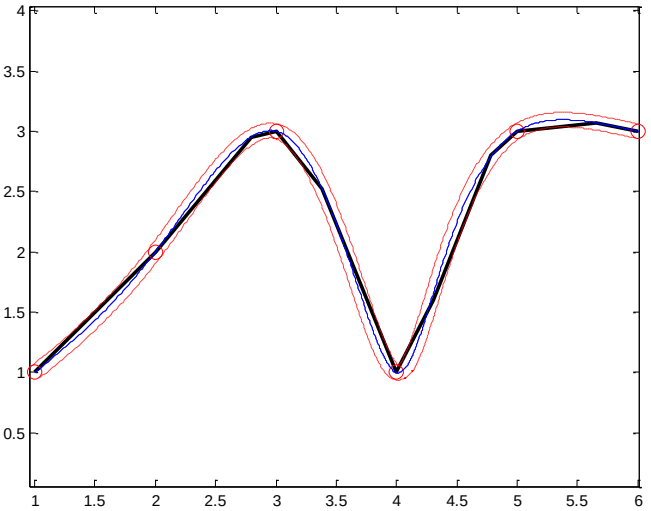
- Pavadinimas: 2 kompiuteris;
- CPU: Intel Core i5-2430M 2.4 GHz (4 branduoliai);
- HDD: 500 GB SATA;
- RAM: 4 GB DDR3 GB;
- OS: Windows 7 Ultimate 64-bit;
- Programinės įranga: MATLAB R2013b 64-bit.

### 2. 5 Tyrimo rezultatai ir jų interpretavimas

Įvykdžius trijų ir penkių atkarpų pjezoroboto formavimo algoritmus gautos suformuotos judėjimo trajektorijos:

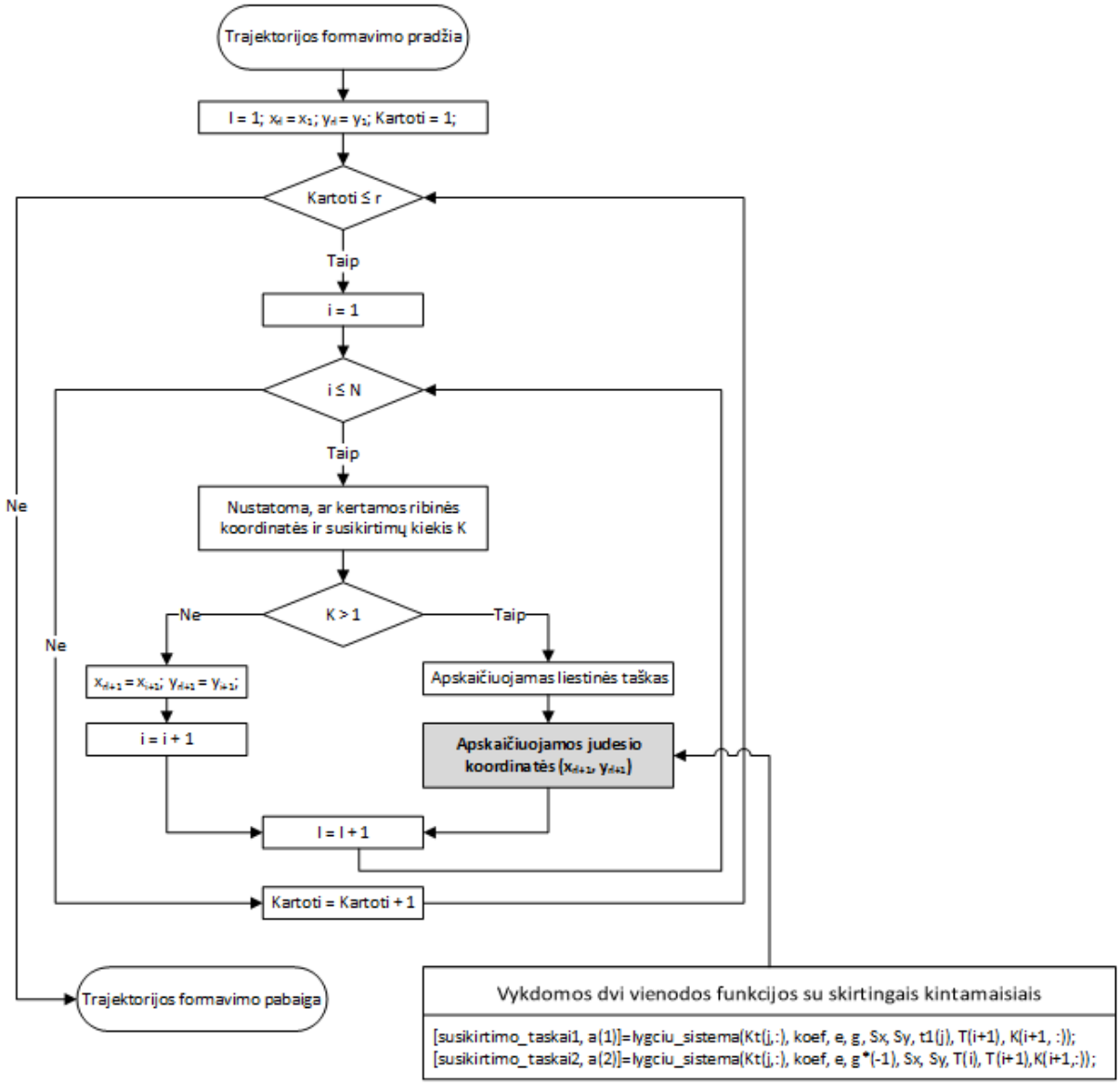



***17 pav.*** *3 atkarpų pjezoroboto judėjimo trajektorija*



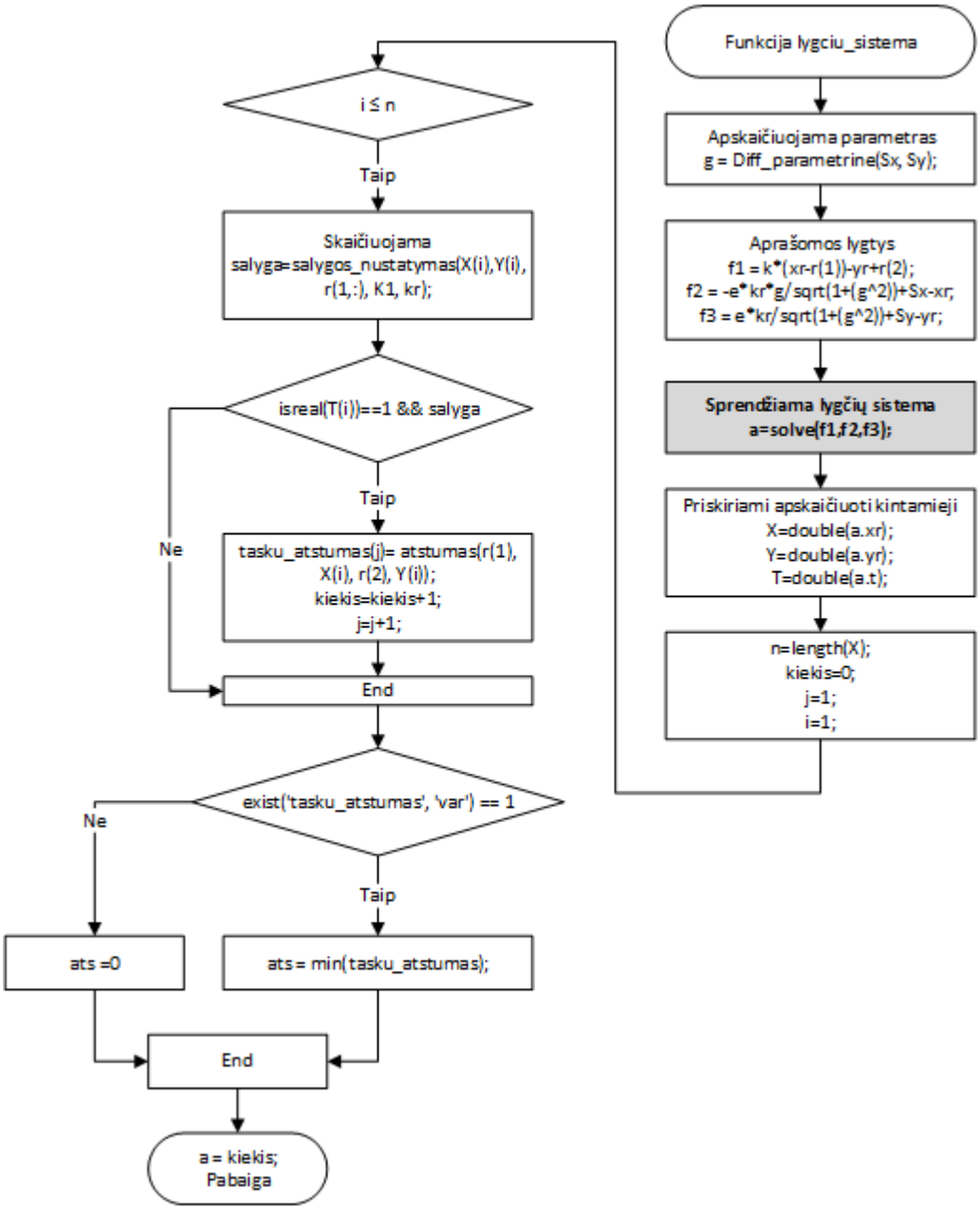

***18 pav.*** *5 atkarpų pjezoroboto judėjimo trajektorija*

Išanalizavus pjezoroboto judesio trajektorijos formavimo algoritmą (žr. 19 pav.) buvo identifikuota probleminė vieta ir įvardyta galima programinio kodo optimizavimo vieta, kuri diagramoje pažymėta paryškintomis raidėmis. Šios probleminės vietos vykdymo laikas, pagal MATLAB duomenis užima 99% viso kodo vykdymo laiko. Papildomame lange diagramoje atvaizduojamos dvi vienodos funkcijos, turinčios tuos pačius įvesties parametrus, kurios gali būti vykdomos lygiagrečiai.



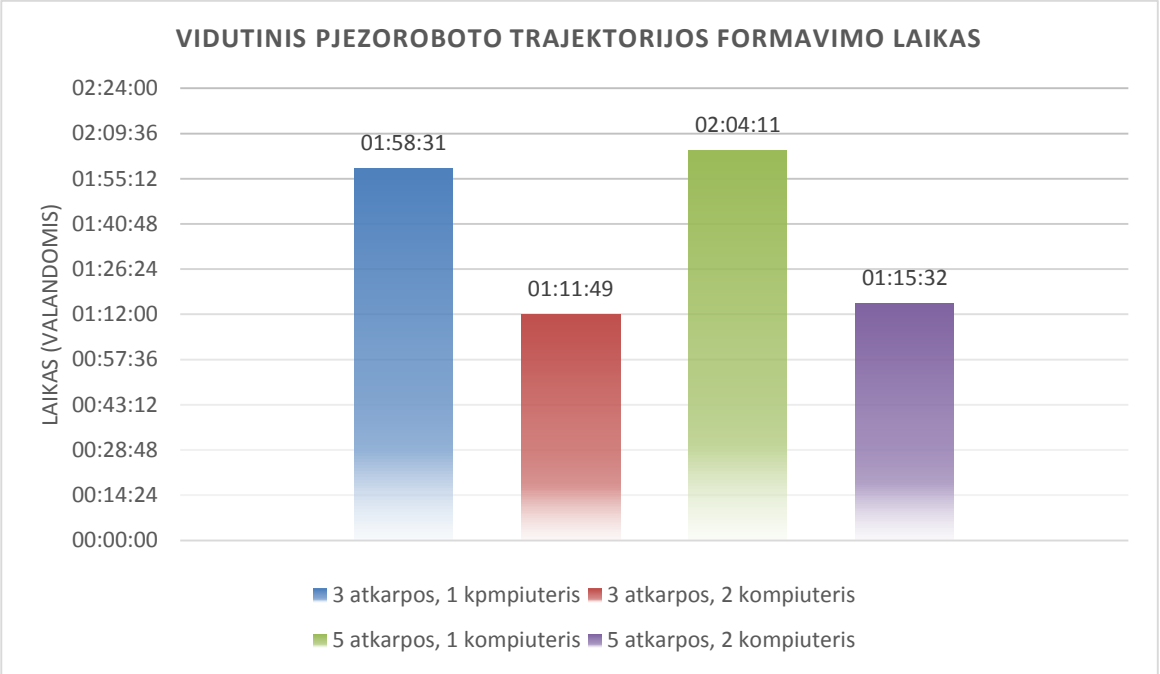

***19 pav.*** *Pjezoroboto trajektorijos formavimo algoritmo liestinių metodu vaizdas*

Šiame paveiksle (žr. 20 pav.) atvaizduojama probleminės funkcijos lygciu_sistema() bendras algoritmo vykdymo vaizdas. Paryškintomis raidėmis pažymėta probleminė funkcijos vieta, sudaranti didžiąją dalį funkcijos vykdymo laiko.




***20 pav.*** *Funkcijos „lygciu_sistema“ algoritmo vykdymo vaizdas*

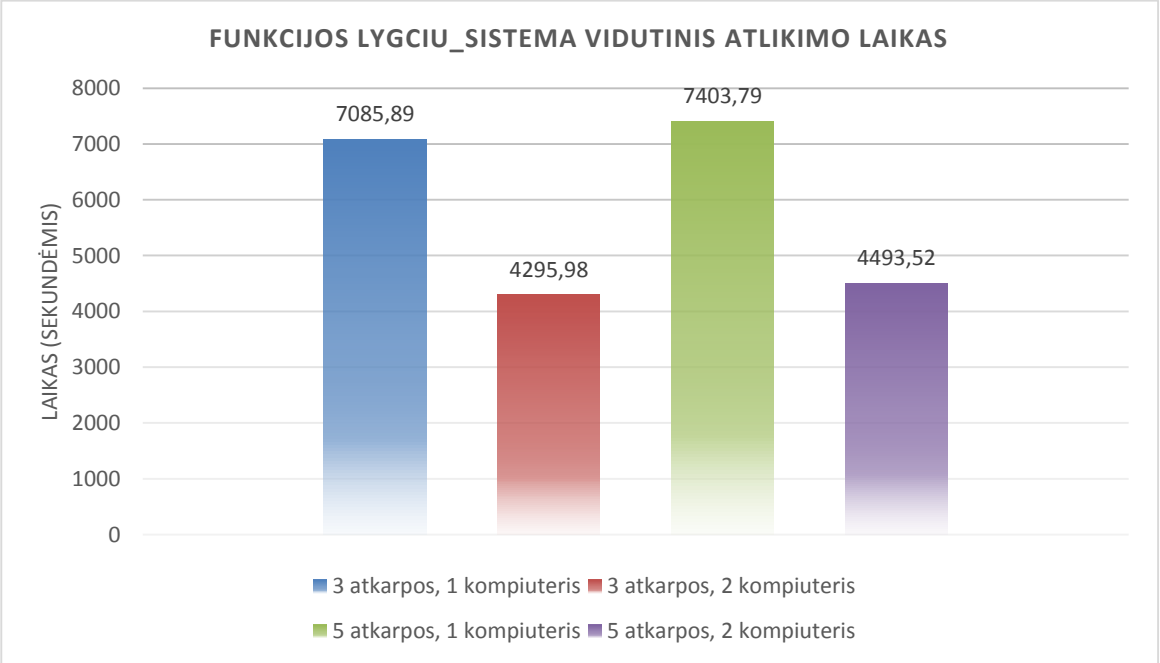
Pjezoroboto trajektorijos formavimas neoptimizavus algoritmo, naudojant skirtingus testavimo resursus bei skirtingo ilgio trajektorijas, kurios nurodytos paveikslėlio apačioje.




***21 pav.*** *Vidutinis pjezoroboto trajektorijos formavimo laikas*

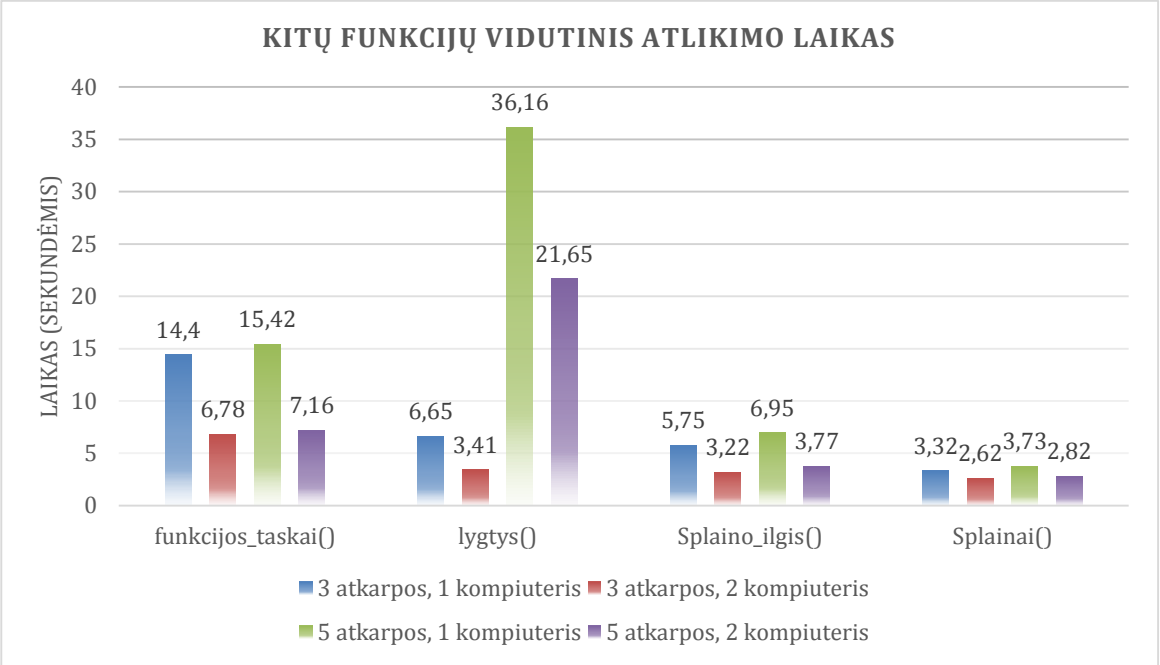
Kaip matyti iš pateiktų rezultatų trajektorijos formavimas buvo atliekamas su dviem skirtingais kompiuteriais, formuojant tas pačias trijų ir penkių atkarpų trajektorijas pastebėta, kad keturis branduolius turintis antras kompiuteris žymiai greičiau susidorojo su pateikta užduotimi, nei pirmasis kompiuteris, turintis du branduolius. Taigi galime daryti išvadą, kad branduolių kiekis turi įtakos bendram algoritmo vykdymui.

Vidutinis funkcijos lygciu_sistema() atlikimo laikas, generuojant skirtingo ilgio trajektorijas. Pateikti rezultatai rodo, kad daugiau branduolių turintis antras kompiuteris greičiau vykdo skaičiavimus, nei pirmasis, turintis du branduolius.




***22 pav.*** *Funkcijos lygciu_sistema vidutinis atlikimo laikas*

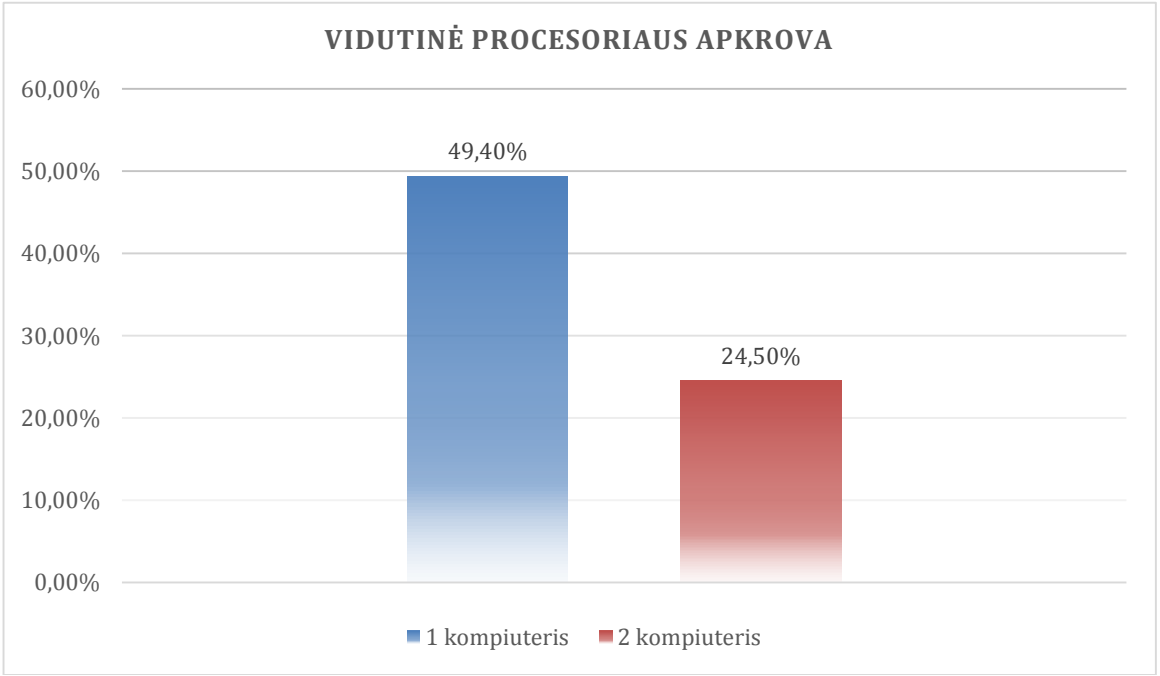
Žemiau atvaizduojamas keturios funkcijos, užimančios ilgiausią laiko tarpą. Likusios funkcijos nesudaro esminės reikšmės, todėl į tyrimą nebuvo įtrauktos kaip probleminės.




***23 pav.*** *Kitų funkcijų vidutinis atlikimo laikas*

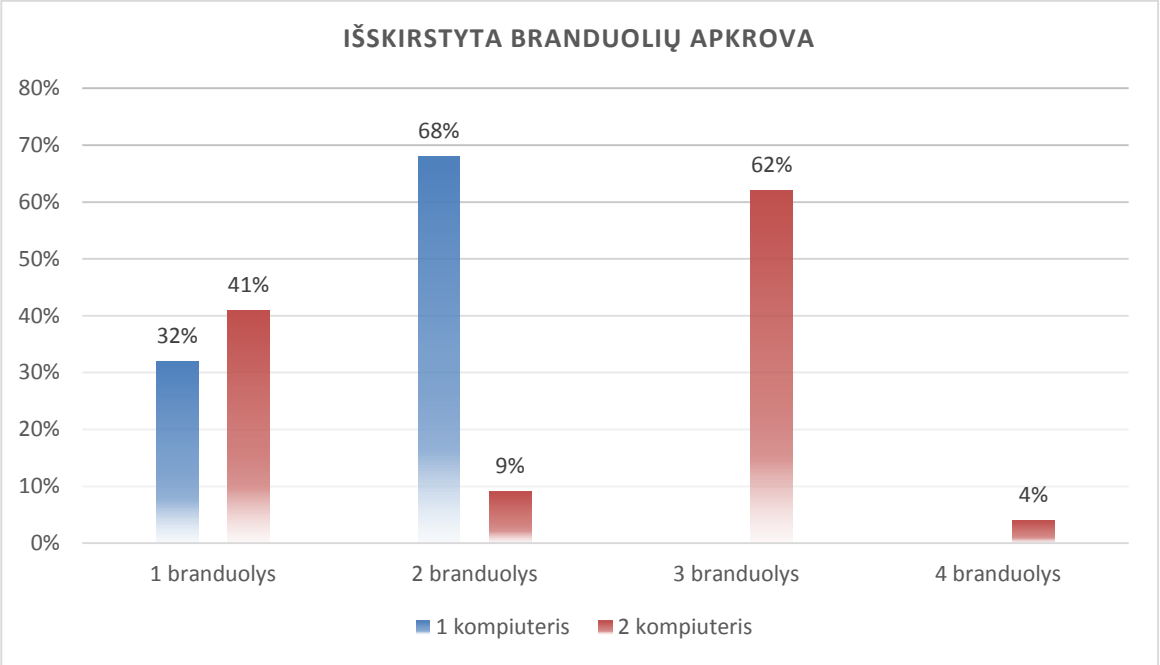
Kaip matyti iš pateikto grafiko, visos funkcijos, išskyrus lygtys(), beveik nepriklauso nuo atkarpų skaičiaus, tuo tarpu funkcija lygtys() yra priklausoma, nes skaičiuojant ilgesnę trajektoriją, turinčia daugiau atkarpų, matyti kelis kartus ilgesnis funkcijos užimamas laikas. Todėl tikėtina, kad skaičiuojant labai ilgas trajektorijas, turinčias daug atkarpų, šis parametras gali turėti įtakos bendram algoritmo vykdymui laiko atžvilgiu.

Vykdant tyrimą nebuvo apsiribojama tik funkcijų vykdymo laikais, nes kaip reikšmingas parametras, galintis įtakoti bendrą trajektorijos formavimo algoritmą gali būti ir procesoriaus apkrova kuri pateikiama žemiau esančiame paveiksle.




***24 pav.*** *Vidutinė procesoriaus apkrova*

Norint sužinoti ir įvertinti, kaip pjezoroboto trajektorijos formavimo algoritmas apkrauna atskirus branduolius, buvo užfiksuoti šie rezultatai.




***25 pav.*** *Išskirstyta branduolių apkrova*

Iš pateiktos diagramos matyti, kad 2 kompiuteris turi keturis branduolius, bet jie nėra pilnai išnaudojami, nes beveik visas krūvis dalinamas dviem branduoliams. Tuo tarpu pirmame kompiuteryje su dviem branduoliais matomas apkrovos netolygumas, bet jis ne toks ryškus.

# 3. PROGRAMINIO KODO NAŠUMO CHARAKTERISTIKŲ OPTIMIZAVIMO BŪDAI

## 3. 1 Programinės įrangos vykdymo charakteristikų optimizavimo būdai

### *3. 1. 1 Techninės įrangos, kurioje naudojama sistema resursų padidinimas*

Kompiuteris sudarytas iš skirtingų komponentų, tokių kaip procesorius, procesoriaus spartinančioji atmintis (cache), darbinė atmintis (RAM), įvestis, išvestis. Bendrai paėmus procesorius ir procesoriaus spartinančioji atmintis yra palyginti greitos, o kompiuterio atmintis lėta. Todėl kiekviena kompiuterio dalis gali būti tobulinama sekančiai būdais:

- Daugiau branduolių / procesorių;
- Spartesnis procesorius;
- Didesnė darbinė atmintis (RAM);
- Didesnė procesoriaus spartinančioji atmintis (cache).

Taigi padarius išvardytus pakeitimus, galima tikėtis didesnės skaičiavimų spartos, nes kiekviena kompiuterio dalis turi didelę įtaką skaičiavimų rezultatams.

### *3. 1. 2 Algoritmo optimizavimas*

Algoritmo optimizavimą galima apibrėžti kaip procedūrą, kuri ieško priimtinų sprendimų, kol randamas optimalus variantas, kuris tenkina iškeltas sąlygas. Renkant optimaliausią variantą priimtini sprendimai yra lyginami vienas su kitu, kol išrenkamas tinkamiausias.

Žemiau pateikiami galimi būdai, užtikrinantys algoritmo optimizavimą:

- Reikia įsitikinti, kad nėra nereikalingų skaičiavimų arba tie skaičiavimai yra vykdomi pakartotinai, nors ankščiau buvusiose iteracijose kintamieji jau buvo apskaičiuoti;
- Jeigu yra galimybė, tam tikrus veiksmus vykdyti lygiagrečiai;
- Programiškai apsibrėžti atminties ribas, nes tai įtakoja algoritmo vykdymo spartą;
- Pašalinti perteklinį kodą, prieš tai įsitikinant, kad jis nenaudojamas;
- Sudėtingai parašytos funkcijos turi būti perrašomos paprasčiau bei suprantamiau kompiliatoriui;
- Būtina įsitikinti, kad funkcijos nėra kviečiamos kelis ar keletą kartų, jeigu tai nereikalinga.

Atsižvelgus į išvardytus punktus galima tikėtis, kad optimizuotas algoritmas veiks greičiau.

### *3. 1. 3 Programinio kodo optimizavimas*

Programinio kodo optimizavimo procesą galima įvardinti kaip kodo dalies transformavimą į efektyvesnį, nepakeitus galutinių rezultatų. Pagrindiniai kriterijai, kuriais remiantis įvertinamas efektyvumas yra tai, kad programa turi veikti greičiau. Optimizavimo tikslas yra pagreitinti kodo vykdymo laiką, bet ne gauti tobulą rezultatą.

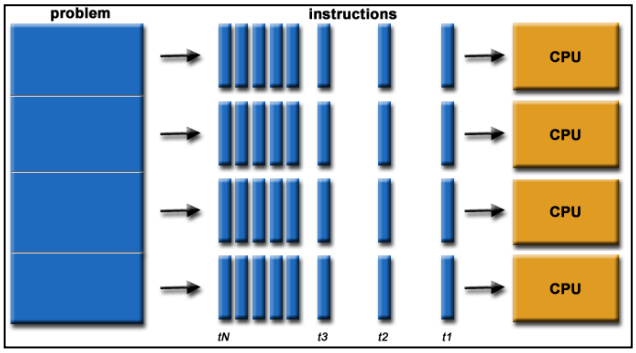
Paprasčiausias būdas tai padaryti yra matuoti kodo vykdymo laiką arba grafiškai atvaizduoti programinio kodo vykdymą, t. y. kokia funkcija ar kokia nors kita kodo dalis užėmė laiko. Turėdami tokius duomenis gali teoriškai spręsti, kur turime daryti pataisymus, kad vykdymo laikas būtų priimtinas. Kai algoritmas optimizuojamas vienam procesoriui, bet neveikia kaip tikėtasi, bandoma

turimą algoritmą pritaikyti vykdyti keletui procesorių vienu metu. Tokia užduotis gali būti realizuojama pasitelkus vidinės atminties technikas, tokias kaip OpenMP, MPI ir pan.

### 3. 2 Lygiagretus programavimas algoritmo optimizavimui

Lygiagretusis programavimas – tai keleto skaičiavimo išteklių naudojimas tuo pačiu metu, sprendžiant uždavinius tokiu būdu:

1. Eksploatuojama naudojant kelis procesorius;
2. Skaičiavimus suskirstyti į kelias lygiagrečias dalis, kurie atliekami kartu;
3. Kiekviena dalis toliau suskirstoma į keletą instrukcijų;
4. Instrukcijos, iš kiekvienos dalies, toliau vykdomos vienu metu skirtingais procesoriais.




***26 pav.*** *Lygiagretaus skaičiavimo schema [1]*

Skaičiavimo resursais gali būti [1]:

1. Vienas kompiuteris turintis kelis procesorius;
2. Tam tikras kompiuterių, sujungtų į vieną tinklą (*network*), skaičius;
3. Abiejų minėtųjų resursų kombinavimas.

Skaičiavimo užduoties sprendimo galimybei turi būti sudarytos sąlygos:

1. Užduotis suskirstyta į lygiagrečias dalis, kurias galima spręsti vienu metu;
2. Vykdyti keletą programos instrukcijų bet kuriuo metu;
3. Laiko atžvilgiu visų uždavinių sprendimas, padalinus juos į dalis, turi būti trumpesnis, nei su vienu skaičiavimo resursu.

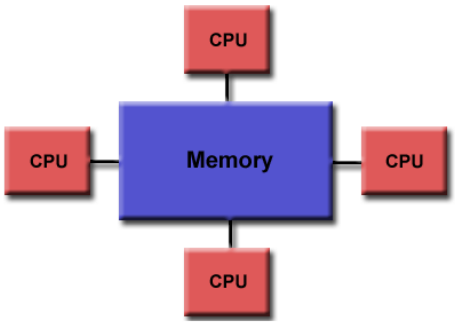
Pagrindinės lygiagretaus programavimo priežastys yra šios [1]:

- Laiko ir pinigų taupymas: teoriškai, naudojant daugiau resursų užduočiai išspręsti taip sutrumpinamas jos įvykdymo laikas ir tuo pačiu sutaupoma lėšų. Be to, lygiagretūs kompiuteriai gali būti sudaryti iš pigių komponentų;
- Didelių uždavinių sprendimas: daugelis uždavinių yra labai dideli ir/ar sudėtingi, todėl tampa ne praktiška ar neįmanoma juos išspręsti vieno kompiuterio pagalba, ypatingai su ribota kompiuterio atmintimi. Pavyzdžiui: interneto paieškos mašinos/duomenų bazės, kur įvyksta milijonas veiksmų per sekundę.
- Konkurencijos kūrimas: vieno kompiuterio resursai sugeba atlikti tik vieną veiksmą vienu metu, tuo tarpu keli skaičiavimo resursai sukuria galimybę atlikti daug veiksmų vienu metu. Pavyzdžiui, the Access Grid (www.accessgrid.org) sistema sukuria pasaulinio bendradarbiavimo tinklą, kur žmonės iš viso pasaulio gali susitikti ir atlikti darbą „iš esmės“.

- Ne lokalių resursų naudojimas: galimybė naudoti plataus tinklo ar net internetinės erdvės skaičiavimo resursus, kuomet lokalūs resursai yra labai riboti.
- Nuoseklaus skaičiavimo ribotumas: dėl fizinių ir praktinių priežasčių sudaryti didelius konstruktyvus nuoseklių kompiuterių, padidinant efektyvumą ir uždavinių sprendimo paprastumą bei atsižvelgiant į tai, jog nuoseklūs kompiuteriai turi savus bruožus:
- Perdavimo sparta - nuoseklaus kompiuterio greitis tiesiogiai priklauso nuo to, kaip greitai duomenys gali pereiti per techninę įrangą (hardware). Absoliuti riba yra šviesos greitis (30 cm / nanosekundę) ir perdavimo ribotumas per varinį laidą (9 cm/ nanosekundę). Didėjantys greičiai reikalaus didėjančio apdorojimo elementų „atvirumo“.
- Ribotos miniatiūrizacijos - procesoriaus technologija leidžia padidinti tranzistorių skaičių čipe (chip). Vis dėlto, net esant molekulinio arba atominio lygio komponentams, riba vis tiek bus pasiekta dėl riboto jų dydžio (pasiekiant minimalų galimą jų dydį).
- Ekonominiai apribojimai - tai didėjanti vieno procesoriaus pagreitinimo kaina. Naudojant didesnį skaičių vidutiniškai greitų procesorių, galima pasiekti tą patį efektyvumą (ar net didesnį) daug pigiau.
- Šiuolaikinė kompiuterių sandara, siekiant pagerinti veiklos rezultatus, vis dažniau pasirenkamas techninės įrangos lygiagretumo principas: keletas sprendimo padalinių, konvejerinės (pipelined) instrukcijos, daugiabranduolinės (multi-core) struktūros.

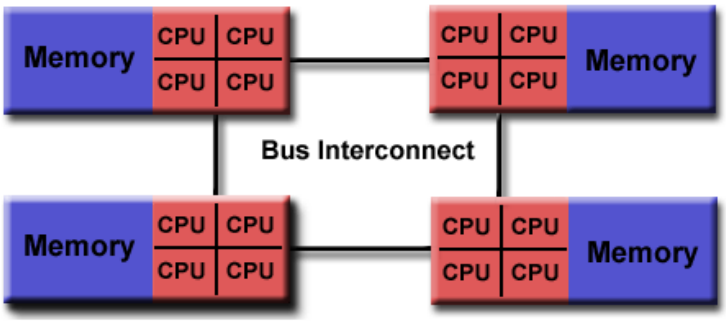
Sudarydami lygiagrečiuosius algoritmus sprendžiame kelis pagrindinius uždavinius, tarp jų svarbiausias – uždavinio skaidymas į mažesnes užduotis, jų paskirstymas procesoriams ir duomenų persiuntimas tarp procesorių. Būtent duomenų persiuntimo algoritmai esmingai priklauso nuo kompiuterio atminties architektūros. Galima išskirti tris atminties architektūros tipus:

1. Dalinama atmintis. Dalinamos atminties lygiagretūs kompiuteriai labai skiriasi, tačiau paprastai turi galimybę, kad visi procesoriai pasiektų visą atmintį (kaip pasaulio adresų erdvė). Multiprocesoriai gali veikti nepriklausomai, tačiau dalinasi tais pačiais atminties resursais. Be to, pokyčiai atminties pasiskirstyme suketi vieno procesoriaus yra matomi visiems kitiems procesoriams. Dalinamos atminties mašinos gali būti suskirstytos į dvi pagrindines klases (jos pagrįstos atminties pasiekimo laiku): UMA ir NUMA.
    a. Tolygus atminties pasiekimas (UMA) daugiausiai šiuo metu naudojamas simetrinių multiprocesorių (SPM) mašinų, kur procesoriai yra identiški. Atminties pasiekimas yra lygus kiekvienam procesoriui, taip pat ir lygiais laiko tarpais (žr. 27 pav.) [1].



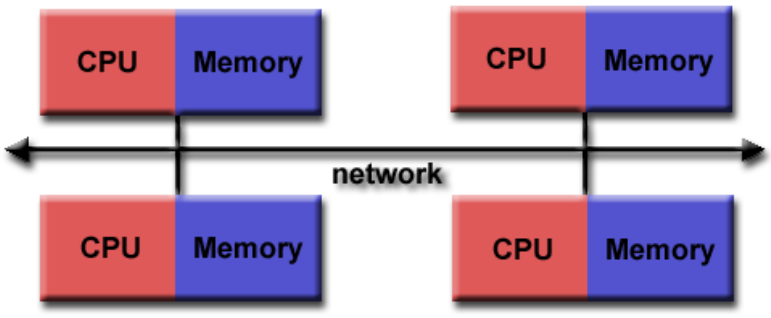

***27 pav.*** *Tolygios atminties pasiekimo procesorių schema [1]*

b. Netolygus atminties pasiekimas (NUMA) dažniausiai būna tarp fiziškai sujungtų dviejų ar daugiau SMP (žr. 28 pav.). Vienas SMP gali tiesiogiai pasiekti atmintį kito SMP. Tačiau ne visi procesoriai turi lygų visų atminčių pasiekimą laike. Atminties pasiekimas aplink sujungimą yra lėtesnis.




***28 pav.*** *Netolygaus atminties pasiekimo procesorių schema [1]*

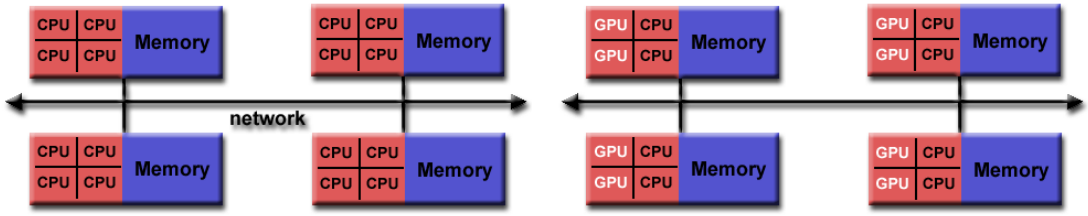
2. Paskirstyta atmintis. Kaip ir dalinamos atminties sistemos, paskirstytos atminties sistemos labai skiriasi, tačiau turi bendrą, plačiai paplitusią charakteristiką. Tam kad sujungti tarpprocesorinę (inter-processor) atmintį, paskirstytos atminties sistemos reikalauja komunikacijos tinklo – Network (žr. 29 pav.). Procesoriai turi savo lokalią atmintį. Atminties adresas viename procesoriuje netinka kitam. Tokiu būdu pasaulio adresų erdvės koncepcija aplink visus procesorius čia neegzistuoja. Kiekvienas procesorius turi savo atmintį, dėl šios priežasties jie veikia nepriklausomai. Pokyčiai atmintyje atsiradę dėl vieno procesoriaus veiklos niekaip neatsispindi kito procesoriaus aplinkoje [1].




***29 pav.*** *Paskirstytos atminties procesorių, sujungtų tinkle, schema [1]*

Kuomet procesoriui reikalingas priėjimas prie kito procesoriaus duomenų, iškyla programuotojui užduotis tiksliai nustatyti kaip ir kada duomenys bus perduodami. Tinklo gijos, naudojamos duomenų perdavimui, taip pat skiriasi (gali būti paprasčiausiai Ethernet'as) [1].

3. Hibridinė paskirstyta dalinama atmintis. Galingiausi ir greičiausi šiuolaikiniai pasaulio kompiuteriai naudoja būtent abi – paskirstytos ir dalinamos atminčių architektūras. Dalinamos atminties komponentas gali būti nuoseklių SMP mašinos ir/arba grafinio apdorojimo elementų (GPU) talpykla (žr. 30 pav.). Paskirstytos atminties komponentą – tai sudėtinio tinklo iš SMP/GPU mašinų sukūrimas, kurios žino tik apie savo atmintį. Vis dėlto, tinklo komunikacija reikalauja perduoti duomenis iš vieno SMP/GPU į kitą. Paskutinės tendencijos rodo, jog šis atminties architektūros tipas išliks vyraujančiu ir dar labiau pasitarnaus aukšto lygio skaičiavimams ateityje [1].




***30 pav.*** *Hibridinės paskirstytos ir dalinamos atminties architektūros schema [1]*

Svarbu paminėti ir lygiagretaus programavimo modelius, kurie yra šiuo metu yra dažnai naudojami [1]:

- Dalinamos atminties (be gijų);
- Gijų;
- Paskirstytos atminties / duomenų perdavimo;
- Lygiagrečių duomenų;
- Hibridiniai;
- Vienos programos daugialypių duomenų (SPMD);
- Daugialypės programos daugialypių duomenų (MPMD).

Lygiagrečių skaičiavimų problemos [2]:

- Sunku naudoti: reikalinga patirtis ir aukšta kvalifikacija, norint paruošti užduotis lygiagretinimui;
- Ribotas lygiagretinamų uždavinių skaičius: lygiagretinimui "pasiduoda" tik nedidelis skaičius pagrindinių uždavinių, trūksta patyrusių specialistų;
- Labai imli darbui lygiagretinimo procedūra: reikia išnagrinėti esamą programinę įrangą, jos modelius ar kodą; intelektinės nuosavybės problema;
- Sunku įvesti pakeitimus į lygiagretų kodą: išlygiagretinus ką nors modifikuoti ar keisti;
- Trūksta priemonių ir įrankių lygiagretinimo efektyvumui įvertinti;
- Kartais sudėtinga sinchronizuoti procesus, dėl ko gali lėtėti lygiagrečiai vykdomi skaičiavimai.

*3. 2. 2 Programavimo kalbos naudojančios lygiagretųjį programavimą*

Yra daug populiarių programavimo kalbų, kuriomis užrašome nuosekliuosius algoritmus, pavyzdžiui C, C++, Fortran, Pascal kalbos. Tokiu būdu reikia nuspręsti, ar realizuodami lygiagrečiuosius algoritmus kursime naujas programavimo kalbas, ar užteks papildyti jau egzistuojančias. Į šį klausimą galima atsakyti tik tada, kai nustatoma, kokių papildomų priemonių reikia, jei norima užrašyti bet kokį lygiagretųjį algoritmą. Taigi buvo suburta specialistų grupė, kuri sudarė paskirstytųjų skaičiavimų standartą ir rekomendavo jį naudoti paskirstytosios atminties (žr. 1 skyrius) lygiagrečiuosiuose kompiuteriuose. Ši duomenų persiuntimo sąsaja buvo pavadinta MPI (Message Parsing Interface). Ją sudarant buvo pasitelkta kitų programavimo bibliotekų (PVM, Chameleon, PARMACS) ypatybės bei įvertinti naujausi teoriniai siūlymai.

Svarbiausios MPI ypatybės [2]:

- MPI yra biblioteka, o ne nauja programavimo kalba. Ji apibrėžia tik paprogramių vardus, parametrus ir jų funkcinę paskirtį. Šias paprogrames galima naudoti Fortran 77 ir C kalbomis parašytose programose; parengtos paprogramių Fortran 95 ir C++ versijos. Vartotojas programą rašo standartine programavimo kalba tik kompiliuodamas prijungia MPI biblioteką.
- MPI realizuotas išreikštinis duomenų siuntimo modelis. Apibrėžtos ne tik būtinosios priemonės, be kurių negalime užrašyti lygiagrečiojo algoritmo, realizuojamo paskirstytosios atminties lygiagrečiuoju kompiuteriu, bet ir daug papildomų paprogramių, lengvinančių algoritmo realizaciją arba padidinančių algoritmo efektyvumą. MPI leidžia atlikti skaičiavimus su heterogeniškais kompiuteriais.
- MPI standartas apibrėžia tik funkcinę paprogramės paskirtį, bet nereglamentuoja jos realizacijos. Todėl skirtingų tipų kompiuterių gamintojai gali šias paprogrames realizuoti efektyviausiu šiam kompiuteriui būdu.
- MPI standartu parašyta programa be pakeitimų gali būti perkelta iš vieno tipo kompiuterio į kito tipo kompiuterį. Tai ypač svarbu kuriant matematinių algoritmų bibliotekas.
- Išsamų MPI standartą sudaro 125 pagrindinės paprogramės ir funkcijos. Be to, jau parengtas ir MPI-2 praplėtimas.

HPF (High Performance Fortran) – yra dar vienas standartas, praplečiantis Fortrano 90 galimybes ir leidžiantis efektyviai realizuoti daugelį lygiagrečiųjų algoritmų. Ši Fortrano versija ypač patogi, kai sprendžiame uždavinius, kuriems būdingas duomenų lygiagretumas. Tada lygiagretusis algoritmas gaunamas iš nuosekliosios Fortrano 90 programos naudojant tik kelias papildomas direktyvas. HPF direktyvos yra įtrauktos į naujausią Fortrano 95 standartą. HPF yra naudojamas tada, kai algoritmui būdingas duomenų lygiagretumas. Jis leidžia išskirti lygiagrečiąsias algoritmo dalis (duomenų lygiagretumo naudojimas) ir paskirstyti duomenis procesoriams. Programuotojui nereikia pačiam rūpintis duomenų perdavimo procesoriams procedūromis, šį darbą automatiška atlieka HPF. Tačiau gautojo lygiagrečiojo algoritmo efektyvumas iš esmės priklauso nuo pasirinkto duomenų paskirstymo būdo be nuo nuosekliajame algoritme egzistuojančio duomenų lygiagretumo [2].

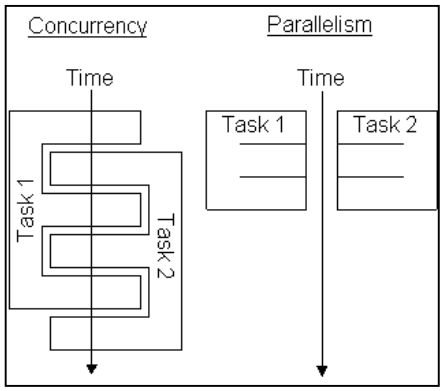
OpenMP yra vienas iš bandymų suformuluoti programavimo standartą, skirtą lygiagretiesiems algoritmams realizuoti bendrosios atminties kompiuteriuose. Siekiama sudaryti nedidelį rinkinį programavimo konstrukcijų, leidžiančių efektyviai realizuoti lygiagretųjį algoritmą. Kadangi OpenMP yra skirtas kompiuteriams, naudojantiems bendrąją atmintį, tai programuotojui nereikia pačiam rūpintis duomenų pasikeitimu tarp skirtingų procesorių. Pagrindinis jų tikslas – išskirti tas algoritmo (ar juos realizuojančios programos) dalis, kurios gali būti vykdomos vienu metu. OpenMP programavimo modelyje pagrindinis programavimo įrankis yra baigtinio skaičiaus procesų (threads) naudojimas. Čia naudojamas terminas threads, kuris reiškia giją dažnai vadinamą srautu.

Pagrindiniai OpenMP programavimo modelio etapai [2]:

- Algoritmą pradeda vykdyti vienas procesas, kurį vadiname šeimininku. Ši algoritmo dalis yra nuoseklioji;
- Nuoseklioji algoritmo dalis nutraukiama, kai naudojama direktyva lygiagrečioji sritis. Tada sukuriama vienu metu dirbančių procesų grupė. Ši algoritmo dalis pažymėta Fork direktyva;
- Kai visi procesai baigia vykdyti lygiagrečiosios algoritmo srities užduotis, jie sustabdomi ir toliau skaičiuoja tik šeimininkas – procesas (Join direktyva).

Panašiai kaip ir HPF programavimo įrankyje, visos OpenMP direktyvos yra informatyvios tik šios programavimo priemonės transliatoriams, o standartiniai C ir Fortrano kalbų transliatoriai jas vertina kaip komentarus. Todėl OpenMP programos be pakeitimų gali būti vykdomos ir nuosekliaisiais kompiuteriais [2].

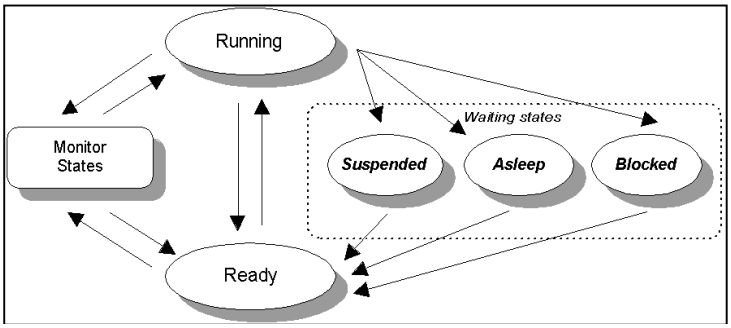
Java – viena iš nedaugelio programavimo kalbų, kurios pilnai palaiko konkurencinį - lygiagretų skaičiavimą (concurent/parallel computing). Skirtumą tarp konkurencinio ir lygiagretaus skaičiavimo gerai atvaizduoja žemiau pateikta diagrama (žr. 31 pav.) [5].




***31 pav.*** *Konkurencinis ir lygiagretus skaičiavimas [6]*

Tam, kad skaičiavimai vyktų tikrai lygiagrečiai reikia, kad kompiuteris turėtų kelis CPU ir lygiagrečiam darbui pritaikytą architektūrą. Tuo tarpu paprastame asmeniniame kompiuteryje dažniausiai yra vienas CPU, todėl užduotys konkuruoja dėl CPU laiko [5].

Java konkurencinio / lygiagretaus darbo mechanizmas realizuotas gijomis (threads) ir sinchronizacijos mechanizmu bei virtualios mašinos struktūra. Didžiausia konkurencinio / lygiagretaus darbo palaikymo dalis yra gijų klasėje. Galimos gijų būsenos pavaizduotos 32 paveiksle [5].

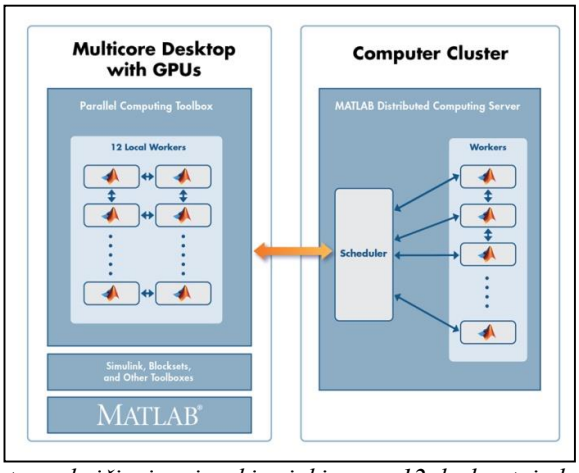



***32 pav.*** *Perėjimai tarp gijų būsenų Java kalboje [5]*

*3. 2. 3 Lygiagretus programavimas ir MATLAB*

Lygiagretaus skaičiavimo įrankių rinkinio produktas siūlo keletą priemonių, kurios palengvina lygiagretaus programavimo operacijas MATLAB. Čia siūlomos programavimo konstrukcijos, tokios kaip lygiagrečios gijos ir suskaidyti masyvai, kurie leidžia išplėsti nuoseklų programavimą į lygiagretų domeną. Šias konstrukcijas galima naudoti be būtinybės mokytis sudėtingas lygiagrečias kalbas ar vykdant reikšmingus pokyčius turimame kode. Įrankių rinkinys palaiko interaktyvų plėtinį, kuris leidžia susijungti su klasteriu tiesiai iš MATLAB sesijos į interaktyvaus veikimo lygiagrečius skaičiavimus. Tokios integracijos kaip Optimization Toolbox, Global Optimization Toolbox ir System Test leidžia naudoti lygiagretaus skaičiavimo galimybes tiesiogiai be jokio lygiagretaus kodo rašymo [4].

Lygiagretaus programavimo įrankių rinkinys MATLAB leidžia išspręsti skaičiavimo ir didelės duomenų apimties uždavinius naudojant daugiabranduolinius procesorius, GPUs bei kompiuterių klasterius. Aukšto lygio lygiagretūs ciklai (for-loops), specialių matricų rūšys bei lygiagretūs skaitmeniniai algoritmai leidžia lygiagrečiai spręsti MATLAB uždavinius nenaudojant CUDA ar MPI programavimo (žr. 2 skyrius). Tam kad paleisti daugkartines simuliacijas esančias lygiagrečiame modelyje, galima naudoti įrankių rinkinį. Skaičiuojant uždavinį lokaliai vieno branduolio darbalaukyje šis įrankių rinkinys siūlo 12 darbuotojų (workers) (MATLAB skaičiavimo mašinų). Nekeičiant kodo, galima paleisti tas pačias aplikacijas kompiuterio klasteryje arba tinkliniame skaičiavimo servise (naudojant MATLAB Distributed Computing Server). Galima paleisti lygiagrečias paraiškas interaktyviai arba grupėmis [4].




***33 pav.*** *MATLAB lygiagretaus skaičiavimo įrankių rinkinys su 12 darbuotojų bei kompiuterio klasteris su paskirstyto skaičiavimo serveriu [4]*

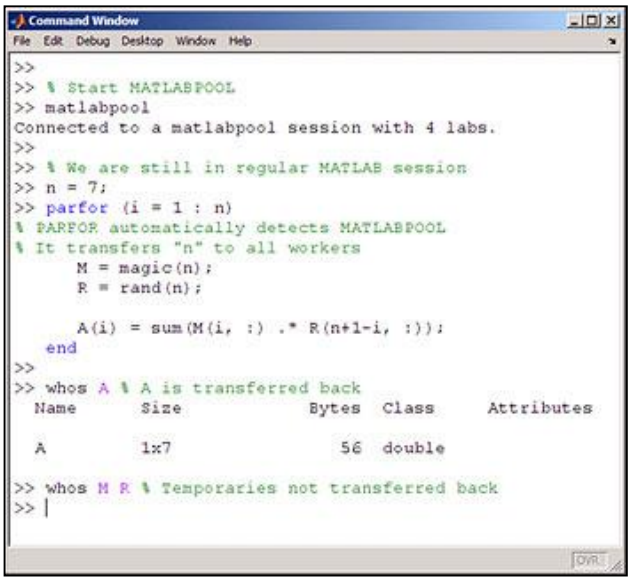
SPMD - Single Program / Multiple Data – tai lygiagrečios programos tipas, kai kiekvienas procesorius turi ir vykdo savo instrukcijų ir duomenų srautus, tačiau visi procesoriai vykdo tą pačią programą, tik skirtingas jos šakas.

Lygiagretaus skaičiavimo įrankių rinkinys siūlo keletą aukšto lygio programavimo konstruktyvų, kurie leidžia konvertuoti turimas paraiškas pasinaudojant kompiuterių su keleto branduolių procesoriumi ir GPU pranašumais. Konstruktyvai, tokie kaip lygiagrečių gijų (PARFOR) ir specialūs matricų tipai perskirstytam darbui bei GPU skaičiavimas, palengvina lygiagretumo kodo vystymą toliau atsiribojant nuo sudėtingumo valdant skaičiavimus ir duomenis tarp savų MATLAB sesijų ir naudojamų skaičiavimo resursų. Galima paleisti tą pačia paraišką skirtingais skaičiavimo resursais be jų perprogramavimo. Lygiagrečios konstrukcijos funkcionuoja tuo pačiu būdu, neatsižvelgiant į resursus, kurių pagrindu vykdome paraišką – daugiabranduolinio darbalaukio (naudojant įrankių rinkinį) ar didesni resursai tokie kaip kompiuterių klasteriai pagrindu [4].

Pagrindinės lygiagretaus MATLAB programavimo funkcijos:

Lygiagrečių gijų (PARFOR) – atliekami lygiagretūs skaičiavimai, naudojant vienu metu veikiančias gijas aplink keletą darbuotojų (workers).

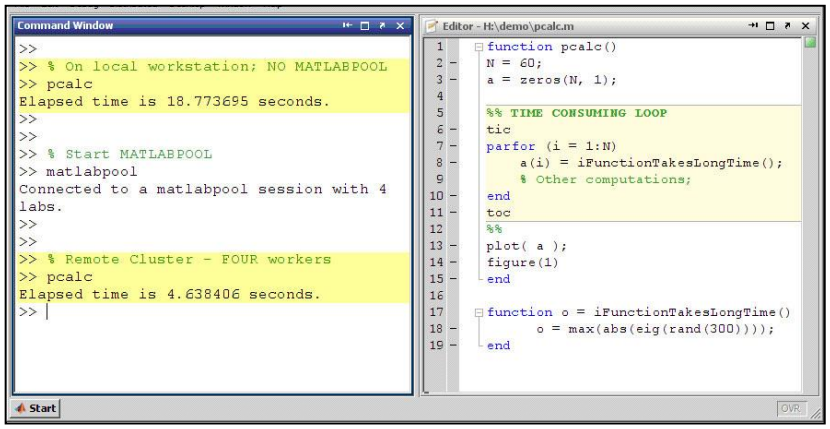
Lygiagrečios gijos leidžia paskirstyti nepriklausomų užduočių rinkinius keliems rinkiniams darbuotojų. PARFOR konstrukcija naudoja panašią for-loop sintaksę ir idealiai tinka parametrų tikrinimui bei panašioms užduotims. PARFOR konstrukcija turi mechanizmą aptikti ir pakeisti būtinus duomenis bei kodus tarp kliento MATLAB sesijos ir darbuotojų. Ji taip pat automatiškai aptinka darbuotojų buvimą. Dėl šios priežasties nereikia sudarinėti bei pateikti sudėtingų grupinių darbų klasteriui [4].




***34 pav.*** *PARFOR gijos algoritmo pavyzdys [4]*

Šiame pavyzdyje (žr. 34 pav.), PARFOR gija stengiasi perskirstyti darbus keletui procesorių paskirstant iteracijas keturiems skirtingiems darbuotojams. Vienintelis reikalavimas – paskirstant užduotis naudojant PARFOR tai, kad iteracijos turi būti nepriklausomos viena nuo kitos bei tarp darbuotojų negali atsirasti jokios komunikacijos per visą užduoties vykdymo lauką. Darbo paskirstymas yra dinamiškas. Vietoj to kad būtų paskirtas fiksuotas iteracijos diapazonas, darbuotojams paskirstomos

naujos iteracijos tiko po to, kai jie pabaigia veikimą su jų dabartine iteracija, kuri yra tolygaus darbo krūvio paskirstymo rezultatas. Su darbuotojais sąveikaujama per matlabpool komandą tiesiogiai iš MATLAB komandų lango. Ši komanda nustato interaktyvią vykdymo aplinką lygiagrečioms konstrukcijoms, tokioms kaip PARFOR ar SPMD. PARFOR gijos gali būti iššaukiamos per komandų eilutę, taip kaip skriptai [4].

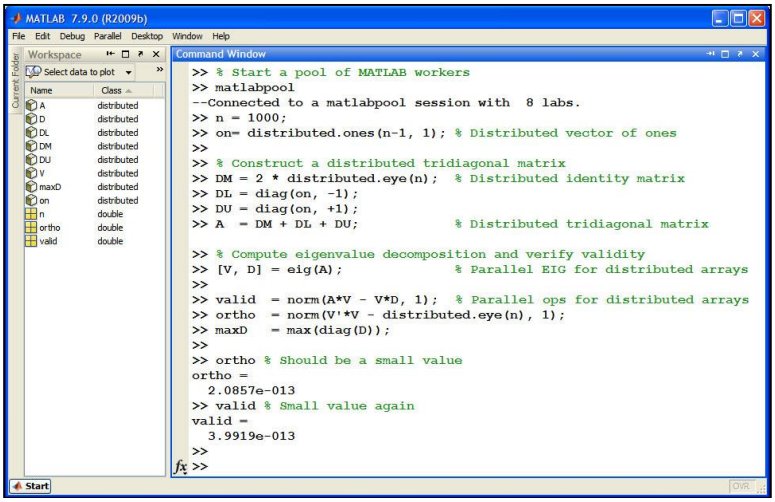



***35 pav.*** *PARFOR komandų eilutė [4]*

Didelių duomenų rinkinių tvarkymas naudojant paskirstytų masyvų ir lygiagrečias MATLAB funkcijas – aplink darbuotojus išskirti bet kokių duomenų tipo matricas ir vykdyti lygiagrečius skaičiavimus šiose duomenų struktūrose.

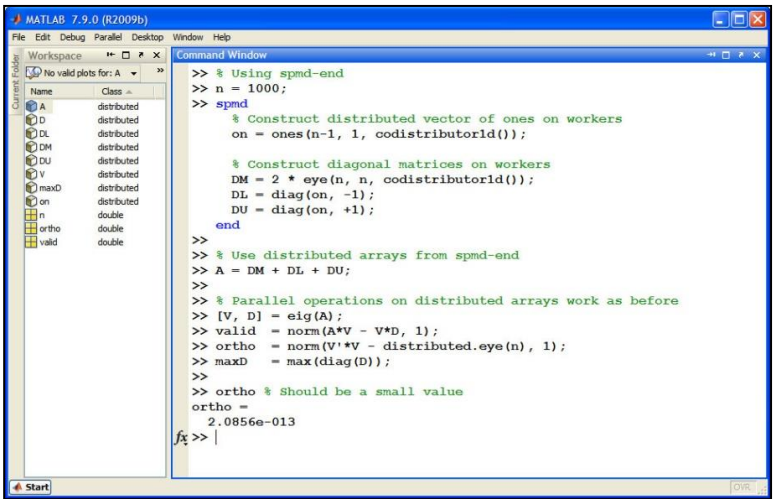
Paskirstyti masyvai yra specialūs masyvai, kurie kaupia MATLAB darbuotojų, kurie dalyvauja lygiagrečiame skaičiavime, duomenų segmentus. Dėl to, jog paskirstyti masyvai palaiko duomenis daugeliui darbuotojų, mes galime tvarkyti didelius duomenų rinkinius, vienoje MATLAB sesijoje. Paskirstytų masyvų duomenys, kurie reziduoja darbuotojus, gali būti išgauti iš MATLAB kliento sesijos. MATLAB automatiškai perduoda naudojimo instrukcijas darbuotojams, kad dirbtų vienu metu visuose masyvuose. Paskirstytus masyvus galima sukurti keliais būdais [4]:

- Naudojant konstrukcijos funkcijas tokias kaip rand(), ones() ar zeros();
- Jungiant masyvus su tuo pačiu vardu, bet skirtingais duomenimis skirtingose laboratorijose;
- Padalinant didelę matricą.




***36 pav.*** *MATLAB komandų langas masyvams paskirstyti [4]*

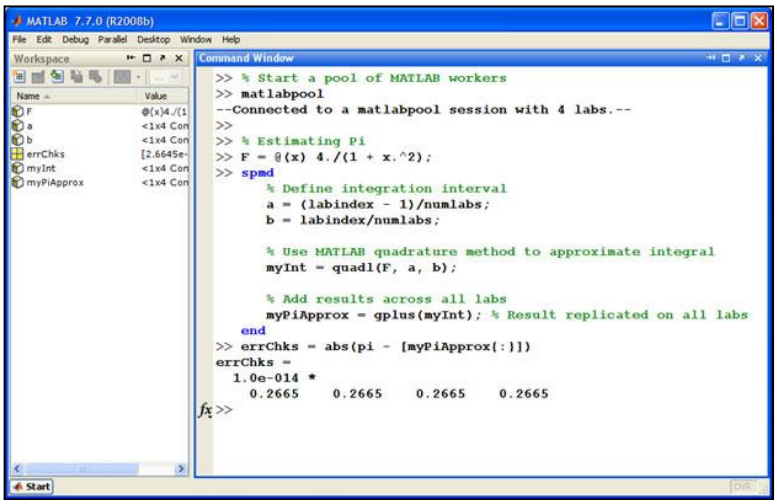
Taip pat yra galimybė tiesiogiai anotuoti savojo kodo sekcijas naudojant *SPMD* (atskiros programos daugialypiai duomenys) konstrukciją lygiagrečioms užduotims, keliems darbuotojams. Kuomet MATLAB susisiekia su *SPMD* formuluote, ji siunčia visas komandas pateiktas tarp *SPMD*-end (žr. 37 pav.) veiksmams MATLAB darbuotojams. MATLAB užtikrina būtinų kintamųjų perdavimą iš vartotojo darbo aplinkos link darbuotojų. Kintamieji esantys MATLAB darbuotojų darbo aplinkoje gali būti išgaunami iš MATLAB kliento, kai tuo tarpu duomenys esantys šiuose kintamuosiuose tęsia nuotoliniu būdu gyvuoti MATLAB darbuotojų darbo aplinkoje [4].




***37 pav.*** *SPMD-end funkcijos naudojimas [4]*

Interaktyvi aplinka – interaktyviai išvystytas algoritmas artimoje MATLAB aplinkoje naudojant MATLAB darbuotojų erdvę (pool) arba lygiagrečių komandų langą.

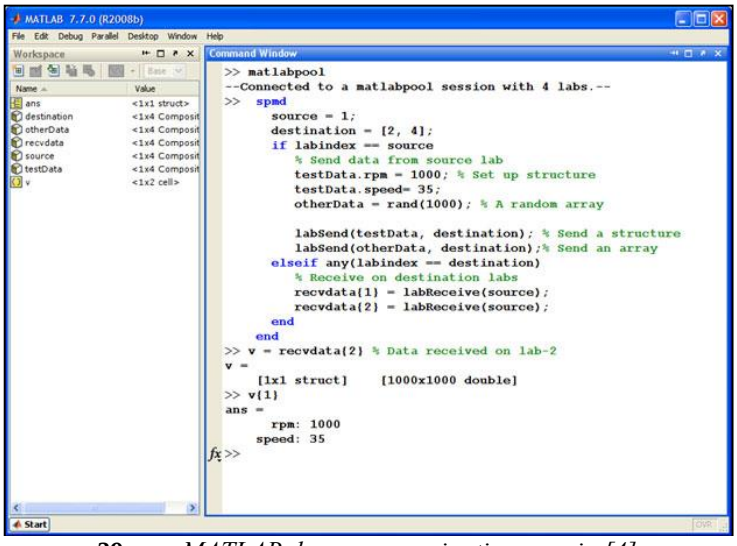
Matlabpool komanda palaiko daug MATLAB darbuotojų ir sukuria interaktyvią aplinką vykdant lygiagretų MATLAB kodą. Šioje aplinkoje, PARFOR ir SPMD konstrukcijos gali sukurti duomenis ir MATLAB kodo apsikeitimą su MATLAB kliento sesija bei MATLAB darbuotojais. Šios operacijos įvyksta automatiškai ir vartotojams nereikia sudarinėti ir pateikinėti darbų klasteriams. Kodas, kuris vykdomas aplinkoje, kuriamas naudojant matlapool komandą gali būti sukurtas ir ne ryšio režime (off-line), naudojant grupės komandą (MATLAB skriptams) arba sukuriant MatlabpoolJob funkciją (MATLAB funkcijoms) [4].




***38 pav.*** *MATLABpool funkcija [4]*

Lygiagrečių komandų langas pateikia MATLAB komandų lango plėtinį, vykdant MATLAB lygiagrečių duomenų kodą tiesiogiai darbuotojo dalyvavime interaktyvioje lygiagrečioje sesijoje. Šis įrankis palengvina derinimo procesą, leisdamas pamatyti rezultatus ir sąveiką tarp laboratorijų bei kiekvieno žingsnio [4]. Duomenų persiuntimo sąsaja – naudoja žemo lygio duomenų persiuntimo sąsajas sudėtingiems lygiagretiems algoritmams.

Lygiagretaus skaičiavimo įrankių rinkinys pateikia prieigą prie duomenų persiuntimo sąsajų paprogramės, prieinamos per Message Passing Interface (MPI-2), realizaciją. Šis patogus aptarnavimo kodas rekomenduojamas geriau suprasti naudojamas aukštesnio lygio konstrukcijas, išskaidytus masyvus bei lygiagrečias gijas. Norint turėti gerą lygiagretaus programavimo schemos kontrolę reikia naudoti duomenų persiuntimo sąsajas. Kuomet naudojamos šios funkcijos, įgauname galimybę valdyti sinchronizaciją tarp turimo algoritmo sekcijų. Funkcijos komandoms: išsiųsti, priimti, transliuoti, kliūtis ir tiriamoms operacijoms prieinamos įrankių rinkinyje. Duomenų persiuntimo sąsajų įrankių rinkinyje pateikiama kelių sudėtingų detalių santrauka, kuri padaro jas paprastesnes naudojimui. Tas pats funkcijų rinkinys tinkamas visiems MATLAB duomenų tipams, įskaitant struktūras ir gardelių masyvus, be jokių papildomų nustatymų. Aklavietės nustatymui padeda nesutampančių išsiųsti-priimti iškvietimų laiko identifikacija, sutaupo laiko, kurį sugaištume šioms sunkiai derinamoms problemoms spręsti. Kaip ir su išskaidytais masyvais bei susijusiomis lygiagrečiomis funkcijomis, duomenų persiuntimo sąsajos gali būti naudojamos su SPMD formuluotėmis [4].




***39 pav.*** *MATLAB duomenų persiuntimo sąsaja [4]*

Vis dėlto, MATLAB lygiagretaus programavimo funkcijos turi ir trūkumų, pavyzdžiui, kuomet naudojame PARFOR funkciją jei yra priklausomybė tarp gijos iteracijų ir priklausomybė gali būti aptikta kodo analizės pagalba, tuomet PARFOR gijos veikimas rodys klaidą. Jei priklausomybė negali būti aptikta, tuomet vienintelis problemos suradimo būdas yra neteisingas rezultato gavimas [4].

Žemiau pateikiama lentelė, kurioje nurodomi apibendrinti lygiagretinimo būdai MATLAB aplinkoje:

*2 lentelė. Lygiagretinimo būdai MATLAB aplinkoje*

| Pavadinimas | Parallel for-Loops (parfor) | Batch Processing | GPU Computing | Distributed Arrays and SPMD |
|---|---|---|---|---|
| Reikalinga NVIDIA CUDA GPU | - | - | + | - |
| Galimybė naudoti kelis CPU | + | + | - | + |
| Ciklo lygiagretinimo galimybė | + | - | - | - |
| Paralelus iteracijų vykdymas | + | + | + | + |
| Galimybė sukurti užduotį (job) | - | + | - | - |
| Paskirstyto skaičiavimo išteklių panaudojimas | - | + | - | - |
| Galimybė išskirstyti masyvą į kelias dalis | - | - | - | + |

*3. 2. 4 Lygiagretaus programavimo praktinis pritaikymas*

Lygiagretusis programavimas – serijos skaičiavimų išvystymas, kurie bando pamėgdžioti tai kas iš tiesų vyksta daugelyje sferų realybėje: daug sudėtingų, tarpusavyje susijusių įvykių vyksta tuo pačiu metu laiko atžvilgiu (laiko sekoje). Pavyzdžiui: planetų judėjimas, procesai atmosferoje, piko valandų transporto spūstys, reaktyvinių lėktuvų konstrukcija ir daug kitų. [1]

Tokiu būdu lygiagretųjį programavimą naudoja daugelis mokslo ir inžinerijos sričių, tokių kaip atmosferos, žemės fizikos, įvairios kitos fizikos sritys (branduolinė, taikomoji, aukšto slėgio ir kt.), biotechnologijos, genetika, chemija, molekuliniai mokslai, geologija, seismologija, mechanika, elektronikos inžinerija, kompiuterinės technologijos, matematika. Taip pat labai plačiai lygiagretusis programavimas paplitęs tarp pramonės ir komercijos sferų, tokių kaip: duomenų bazės, duomenų gavyba, naftos išgavimas, internetinių naršyklių mašinos, medicinos diagnostika, finansų ir ekonomikos modeliavimas, multimedijos technologijos ir daug kitų. [1]

## 3. 3 Programinio kodo optimizavimo būdai

*3. 3. 1 Baziniai programavimo kalbų našumo tyrimai*

Šiais laikais, kai programuojamų įrenginių poreikis auga didžiuliai tempais, labai aktuali tema tampa kaip greitai įrenginys gali susidoroti su iš anksto numatyta užduotimi. Todėl šių įrenginių kūrėjai susiduria su problema, kaip efektyviai optimizuoti jų darbą. Viena iš to priežasčių yra nekokybiškas programavimas arba programavimo kalbų subtilybės, su kuriomis susiduria kiekvienas, norintis parašyti kokybišką programinę įrangą.

Tirdamas ir ieškodamas informacijos, kaip galėčiau išspręsti susidariusią situacija pastebėjau, kad yra nemažai mokslinių straipsnių, kuriose kalbama apie programavimo kalbų efektyvumą ir kaip su tam tikra programavimo kalba parašyti efektyvią programinę įrangą. Štai S. Gilmore kalba apie tai, kad efektyvumas yra paliekamas nuošalyje. Todėl, kad pagrindinis tikslas, kad programinė įranga atlieka savo darbą, nesvarbu kokiais kaštais. Todėl, kad sukurti kokybišką programą reikia nemažai tyrimų, o tai atsiremia į lėšas ir laiką. Optimizuojant didelės apimties kodą neišvengiamai susiduriama su sunkumais, kaip programuotojui reikia perprasti ir padaryti kodą efektyvesnį. Ypač kai tai daroma su žemo lygio kalbomis, nes šiuolaikiniai procesoriai geba apdoroti didelius duomenų kiekius. [12]

Štai viename straipsnyje buvo lyginamos C, C#, Java, Calm, ir Cyclone programavimo kalbos bei sukurtas vienas ir tas pats algoritmas matuojamoms kalboms bei matuojami rezultatai. Tyrimo duomenys atskleidė, kad Cyclone programavimo kalba geriausiai susitvarkė su duota užduotimi, todėl galime daryti išvadą, kad pasitelkti vien intuiciją yra klaidinga. Todėl į būtinai į pagalbą reikia pasitelkti matavimus. [11]

Kitoje publikacijoje taip pat kalbama apie efektyvų C/C++ programavimą įterptinėms sistemoms. Jose yra dideli ROM/RAM apribojimai, todėl nepaslaptis, kad verčiant C/C++ kodą kompiliatoriaus pagalba į mašininį yra sunaudojama daugiau ROM/RAM resursų, todėl ypač svarbu tinkamai sudėlioti kodo eilutes taip, kad gautume maksimalią naudą ir funkcionalumą. [11]

*3. 3. 2 MATLAB našumo savybės ir optimizavimo būdai*

Norint gauti informaciją apie tai kurias programos sritis reikia tobulinti MATLAB turi tokius įrankius kaip Profiler, chronometro vaidmenį atliekančias funkcijas tic(), toc(), bei timeit(). Profiler įrankis yra labai naudingas matuojant santykinį kodo vykdymo laiką, bei nustatant konkrečias kodo veiklos kliūtis. Funkcija timeit() gali išmatuoti konkrečios funkcijos veikimo laiką, o tic(), toc() funkcijos naudojamos apskaičiuoti mažesnių kodo gabalų veikimo trukmę, taip pat kodą, kuris nėra funkcija.

Iš anksto nustatomi masyvai for ir while, kurie kiekvieną ciklą didina duomenų struktūros dydį gali neigiamai paveikti vykdymo rezultatus bei atminties naudojimą. Pakartotinis masyvo dydžio keitimas MATLAB priverčia praleisti daugiau laiko ieškant daugiau atminties gretimuose atminties blokuose, juos suradęs masyvas perkeliamas į juos. Dažnas tokios problemos sprendimas, pagerinantis kodo vykdymo laiką yra iš anksto apibrėžiama didžiausia reikalinga vieta, reikalinga masyvui. [14]

*3 lentelė. Iš anksto apibrėžiamas masyvo dydis [14]*

| | |
|---|---|
| tic<br>x = 0;<br>for k = 2:1000000<br>  x(k) = x(k-1) + 5;<br>end<br>toc<br>Elapsed time is 0.301528 seconds. | tic<br>x = zeros(1, 1000000);<br>for k = 2:1000000<br>  x(k) = x(k-1) + 5;<br>end<br>toc<br>Elapsed time is 0.011938 seconds. |

Norint pasiekti geriausių rezultatų reikia iš anksto žinomas reikšmes priskirti kintamiesiems. Jeigu yra poreikis saugoti kito tipo duomenis, patartina sukurti naują kintamąjį. Esamo masyvo ar klasių

esamų kintamųjų keitimas taip pat lėtiną programos darbą, nes tai yra papildomos laiko sąnaudos. Kaip blogą pavyzdį galime įvardyti kintamojo X = 1 keitimą į X = ‘A’. Taip pat negalima keisti realaus skaičiaus į kompleksinį ar atvirkščiai, nes tai taip pat neigiamai veikia programos našumą [14].

Kada naudojamos loginės operacijos AND ir OR reikia tinkamai pasirinkti simbolį, kuris tiksliai atitinka apdorojamus duomenis.

*4 lentelė. Operatorių AND ir OR paaiškinimai [14]*

| Operatorius | Paaiškinimas |
|---|---|
| &, \| | Naudojama paprastam masyvui apdoroti. |
| &&, \|\| | Naudojama skaliarinėms reikšmėms. |

Jeigu programuojant reikalinga panaudoti if ir while sąlygas, rekomenduojama naudoti &&, || operatorius, nes jie dažnai nevertina visos loginės išraiškos. Pavyzdžiui jeigu pirmas įvesties argumentas nėra skaičius, tai MATLAB vykdo tik pirmą programinio kodo dalį. Pvz.: if (isnumeric(varargin{1})) && (ischar(varargin{2})). [14]

Jeigu programos efektyvumui vis tiek iškyla problemų, būtina:

- Padalinti didelius failus į mažesnius;
- Suskaidyti kodą į mažesnes funkcijas, jeigu tai yra įmanoma;
- Jeigu yra labai sudėtingų funkcijų arba išraiškų, išskirstyti ir naudoti paprastesnes;
- Naudoti funkcijas, o ne skriptus, nes jos paprastai greičiau veikia;
- Naudoti nedideles matricos struktūras, nes jos reikalauja papildomos atminties;
- Nevykdyti didelės apimties procesų MATLAB fone.

# 4. LIESTINIŲ METODO OPTIMIZAVIMAS IR JO EFEKTYVUMO TYRIMAS

## 4. 1 Algoritmo optimizavimas, panaudojant lygiagrečius algoritmus

*4. 1. 1 Metodo optimizavimo aprašymas*

Norint lygiagretinti tam tikrą uždavinį, visų pirma yra ieškoma vietų, kuriose būtų vykdomos pakartotinai, bet nepriklausomai viena nuo kitos – ciklų, kurių kiekviena iteracija yra nepriklausoma nuo prieš tai buvusių. Tai paprasčiausia padaryti analizuojant jau turimą programinį kodą ir jame ieškant ciklo sakinių. Pjezoroboto trajektorijos formavimo lietinių metodu programiniame kode buvo vertinami šie ciklo sakiniai (žr. 5 lentelę) ir analizuojama ar jų iteracijos yra priklausomas viena nuo kitos ar ne.

*5 lentelė. Ciklo sakinių analizė lygiagretinimo galimybėms*

| Nr. | Ciklo sakinys | Paskirtis | Tinkamumas lygiagretinti | Tinkamumo/Atmetimo priežastis |
|---|---|---|---|---|
| 1. | `while kartoti<=r` (74 kodo eilutė pagrindas.m faile) | Nustatomas ciklinės trajektorijos kartojimų kiekis | Netinka | Ciklas gali būti nutraukiamas neįvykdžius visų iteracijų (`if liestine==0 kartoti=kartoti+1; else break; end`). To nebūtų galima padaryti, jei visos iteracijos būtų vykdomos lygiagrečiai. |
| 2. | `for i=1:N` (76 kodo eilutė pagrindas.m faile) | Analizuojamas kiekvienas splainas atskirai | Netinka | Šio ciklo viduje yra daug kintamųjų, kurie padaro iteracijas priklausomas viena nuo kitos: (`if j==40 break; end`) – nurodo kada nutraukti ciklą; (`j=j+1;`) – naudojamas kito ciklo viduje, todėl nepavyktų išlaikyti reikšmės nuoseklumo kiekvienoje iteracijoje; (`clear vieta;`) – funkcijos clear naudojimas gali įtakoti kitas iteracijas; ir kt. |
| 3. | `while judeti==1` (95 kodo eilutė pagrindas.m faile) | Analizuojamas kiekvienas žingsnis splaino viduje | Netinka | Šiame cikle iš anksto nežinomas iteracijų skaičius, nes jo kartojimas priklauso nuo kiekvienos iteracijos viduje vykstančių veiksmų. Taip pat šiame cikle naudojamas kintamųjų reikšmių išvalymas (`clear vieta;`), kuris taip pat apsunkintų ciklo išlygiagretinimą. |
| 4. | `for w=1:f` (128 kodo eilutė pagrindas.m faile) | Apskaičiuojama judėjimo kryptis | Netinka | Cikle yra naudojamas sąlygos sakinys, kuris įtakoja m reikšmę, o ji yra naudojama kiekvienoje iteracijoje, todėl būtų sunku išlaikyti |

| | | | | |
|---|---|---|---|---|
| | | | | tinkamą m nuoseklumą kiekvienoje iteracijoje, jas vykdant lygiagrečiai. |
| 5. | `for w=1:f` (136 kodo eilutė pagrindas.m faile) | Ieškomas trumpiausias kelias | Netinka | Cikle yra naudojamas sąlygos sakinys, todėl sakinys negali būti vykdomas lygiagrečiai. |
| 6. | `for q=1:qq` (169 kodo eilutė pagrindas.m faile) | Skaičiuojamas judėjimo atstumas | Tinka | Ciklo viduje vykdoma viena kodo eilutė, nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai.<br>Tačiau reikėtų praktiškai įvertinti ar toks išlygiagretinimas bus efektyvus, nes kiekvienos iteracijos metu yra atliekami labai paprasti, trumpi veiksmai. |
| 7. | `for i=2:tasku_sk` (214 kodo eilutė pagrindas.m faile) | Skaičiuojamas roboto judėjimo kelias | Netinka | Ciklo viduje yra keli sąlygos sakiniai, o juose kinta 1 reikšmė, todėl būtų sunku lygiagrečiose iteracijose užtikrinti jų nuoseklumą. |
| 8. | `for j=1:c` (217 kodo eilutė pagrindas.m faile) | Skaičiuojami roboto kampai su x ašimi | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 9. | `for j=1:c` (226 kodo eilutė pagrindas.m faile) | Apskaičiuojami posūkio kampai | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 10. | `for i=1:N` (264 kodo eilutė pagrindas.m faile) | Grafiko braižymas | Netinka | Ciklo cikle negalima vykdyti lygiagrečiai. |
| 11. | `for j=T(i):0.002:T(i+1)` (265 kodo eilutė pagrindas.m faile) | Grafiko braižymas | Netinka | Ciklo cikle negalima vykdyti lygiagrečiai. |
| 12. | `for i=1:n` (4 kodo eilutė funkcijos_taskai.m faile) | Funkcijos taškų skaičiavimas | Netinka | Ciklo cikle negalima vykdyti lygiagrečiai. |
| 13. | `for j=T(i):0.002:T(i+1)` (4 kodo eilutė funkcijos_taskai.m faile) | Funkcijos taškų skaičiavimas | Netinka | Ciklo cikle negalima vykdyti lygiagrečiai. |
| 14. | `for i=1:n` (11 kodo eilutė kreivumo_spindulys.m faile) | Nustatomas kreivumo spindulys | Netinka | Ciklo cikle negalima vykdyti lygiagrečiai. |

| | | | | |
|---|---|---|---|---|
| 15. | `while rasta==0` (14 kodo eilutė kreivumo_spindulys.m faile) | Ieškomas kreivumo spindulys | Netinka | Ciklo viduje yra ciklas, be to cikle išvalomos reikšmės, kurios turi įtakos bendram algoritmo vykdymui. (`clear a parametras pparam`) |
| 16. | `for ii=1:k` (22 kodo eilutė kreivumo_spindulys.m faile) | Ieškomas kreivumo spindulys | Netinka | Cikle yra sąlygos sakinių. Kintamasis parametras neleidžia vykdyti kiekvienos iteracijos lygiagrečiai. |
| 17. | `for i=1:N-1` (5 kodo eilutė Kryptys_tarp_tasku.m faile) | Nustatoma kryptis tarp taškų | Netinka | Cikle yra sąlygos sakinių. |
| 18. | `for i=1:n` (13 kodo eilutė lygciu_sistema.m faile) | Skaičiuojama lygčių sistema | Netinka | Cikle yra sąlygos sakinys. Kintamieji X, Y, m ir param neleidžia kiekvienos iteracijos vykdyti lygiagrečiai. |
| 19. | `for i=1:n` (47 kodo eilutė lygciu_sistema.m faile) | Skaičiuojama lygčių sistema | Netinka | Cikle yra sąlygos sakinys. Kintamieji j ir tasku_atstumas neleidžia kiekvienos iteracijos vykdyti lygiagrečiai. |
| 20. | `for i=1:n` (17 kodo eilutė lygtys.m faile) | Atrenkami realūs teigiami parametrai | Netinka | Cikle yra sąlygos sakinys, be to kintamieji m, koord ir parametrai neleidžia kiekvienos iteracijos vykdyti lygiagrečiai. |
| 21. | `for i=1:m-1` (27 kodo eilutė lygtys.m faile) | Ieškomas minimalus atstumas tarp taškų | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 22. | `for i=1:n` (49 kodo eilutė lygtys.m faile) | Ieškomas minimalus atstumas tarp taškų | Netinka | Cikle yra sąlygos sakinys. |
| 23. | `for i=1:n` (11 kodo eilutė parametro_apskaiciavimas.m faile) | Apskaičiuojamas parametras | Netinka | Ciklo cikle negalima vykdyti lygiagrečiai. |
| 24. | `for j=1:m` (12 kodo eilutė parametro_apskaiciavimas.m faile) | Apskaičiuojamas parametras | Netinka | Ciklo cikle negalima vykdyti lygiagrečiai. |
| 25. | `for i=1:N` (4 kodo eilutė Santykis.m faile) | Apskaičiuojamas santykis | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 26. | `while i<N` (5 kodo eilutė Skirtumai.m faile) | Skaičiuojami skirtumai | Netinka | Šiame cikle iš anksto nežinomas iteracijų skaičius, nes jo kartojimas priklauso nuo kiekvienos iteracijos viduje vykstančių veiksmų. |
| 27. | `for i=1:N` (5 kodo eilutė Splaino_ilgis.m faile) | Apskaičiuojami splaino ilgiai | Netinka | Negalima vykdyti, nes funkcija naudojama keliose vietose, todėl sunku užtikrinti kintamųjų nuoseklumą. |

| | | | | |
|---|---|---|---|---|
| 28. | `for i=1:n-1` (6 kodo eilutė Splainai.m faile) | Splainų skaičiavimai | Netinka | Kintamasis suma yra priklausomas nuo kitų iteracijų. |
| 29. | `for i=1:n` (10 kodo eilutė Splainai.m faile) | Splainų skaičiavimai | Netinka | Kintamasis suma yra priklausomas nuo kitų iteracijų. |
| 30. | `for i=1:n` (15 kodo eilutė Splainai.m faile) | Splainų skaičiavimai | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 31. | `for i=1:n-1` (23 kodo eilutė Splainai.m faile) | Splainų skaičiavimai | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 32. | `for i=1:n-2` (38 kodo eilutė Splainai.m faile) | Splainų skaičiavimai | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 33. | `for i=1:n-2` (48 kodo eilutė Splainai.m faile) | Splainų skaičiavimai | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 34. | `for i=1:n-2` (58 kodo eilutė Splainai.m faile) | Splainų skaičiavimai | Tinka | Nėra sąlygos sakinių ar kitų kodo eilučių, kurios neleistų kiekvienos iteracijos vykdyti lygiagrečiai. |
| 35. | `for i=1:n-1` (77 kodo eilutė Splainai.m faile) | Splainų skaičiavimai | Netinka | Kintamasis Kx yra priklausomas nuo kitų iteracijų. |

Kitas faktorius, galintis leisti programinio kodo išlygiagretinimą yra vykdomi veiksmai, kurie vienas kito neįtakoja ir jų atlikimo eiga nėra svarbi. Kadangi visame algoritme yra daug vykdomojo kodo, kiekvieną eilutę įvertinti ar galima keisti jos eilės tvarką būtų netikslinga. Todėl analizuojamas tik aprašytų funkcijų naudojimas kode, jų eiliškumas.

*6 lentelė. Funkcijų analizė lygiagretinimo galimybėms*

| **Nr.** | **Funkcijų kvietimas** | **Tinkamumas lygiagretinti** | **Tinkamumo/Atmetimo priežastis** |
|---|---|---|---|
| 1. | `[Sp, T, Atstumas]=Splainai(K, ciklas);`<br>`Sp_ilgis=Splaino_ilgis(Sp, T);`<br>`santykis=Santykis(Atstumas, Sp_ilgis);` | Netinka | Visų 3 funkcijų eiliškumas labai svarbus, nes vienoje gauti rezultatai yra naudojami kitos įvesčiai. |
| 2. | `[susikirtimo_taskai1, a(1)]=lygciu_sistema(Kt(j,:), koef, e, g, Sx, Sy, t1(j), T(i+1), K(i+1, :));`<br>`[susikirtimo_taskai2, a(2)]=lygciu_sistema(Kt(j,:), koef, e, g*(-1), Sx, Sy, T(i), T(i+1),K(i+1,:));` | Tinka | Ta pati funkcija vykdoma 2 kartus, tačiau su skirtingais įvesties parametrais. Antrosios funkcijos parametrai nepriklauso nuo pirmosios rezultatų, todėl funkcijų vykdymo eiliškumas gali kisti. |
| 3. | `jud_t=lygtys(Sx, Sy, Kt(j, :),e, g, tt, T(i+1),K(i+1,:));`<br>`jud_t(size(jud_t,1)+1,:)=lygtys(Sx, Sy, Kt(j, :),e, g*(-1), tt, T(i+1),K(i+1,:));` | Netinka | Ta pati funkcija kviečiama 2 kartus ir jų įvesties parametrai nepriklauso vienas nuo kito. Tačiau antrosios funkcijos kvietimo metu gautas rezultatas yra priskiriamas kintamajam, kurio dydis |

| | | | |
|---|---|---|---|
| | | | priklauso nuo pirmosios funkcijos kvietimo rezultatų. |

Ciklo atveju išlygiagretinimui pakanka for sakinius perrašyti į parfor, nes tai MATLAB pakete reiškia, kad kiekviena tokio ciklo iteracija turi būti vykdoma kaip atskira gija. Tam šias dvi eilutes siūloma perkonstruoti į ciklą, kurio metu kiekvienoje iteracijoje būtų vykdomas atitinkama kodo eilutė. Tam ciklo viduje reikėtų tikrinti kelinta tai iteracija ir atitinkamai pagal iteracijos pavadinimą, priskirti atliekamą veiksmą, pavyzdžiui:

```
parfor lyg_i=1:3
       if lyg_i == 1
           disp('1 iteracija');
           pause(10);
       elseif lyg_i == 2
           disp('2 iteracija');
           pause(5);
       elseif lyg_i == 3
           disp('3 iteracija');
           pause(7);
       end;
end;
```

Kitas galimas atvejis, kada kiekvienoje iteracijoje yra kviečiama ta pati funkcija, bet naudojant kitus parametrus. Tokiu atveju ciklo sakinį galima rašyti be sąlygos sakinių, o prieš ciklą sukurti masyvo tipo parametrų atitikmenį.

*7 lentelė. Ciklo sakinio pavyzdys*

| **Pirminis kodas** | **Pakeistas kodas** |
|---|---|
| `[a] = funkcija(in1, e);`<br>`[b] = funkcija(in2, e);`<br>`[c] = funkcija(in3, e);` | `in(1)=in1;`<br>`in(2)=in2;`<br>`in(3)=in3;`<br>`parfor lyg_i=1:3`<br>`  [out(lyg_i)] = funkcija(in(lyg_i), e);`<br>`end;`<br>`a=out(1);`<br>`b=out(2);`<br>`c=out(3);` |

Tokiu būdu padaugėja kodo eilučių, tačiau jei funkcijos vykdymas yra ilgas, tai įvesties ir išvesties reikšmių priskyrimo sakiniams skirtas vykdymo laikas gali būti kompensuojamas lygiagretaus funkcijų vykdymo atžvilgiu.

Atsižvelgiant į išanalizuotas lygiagretinimo galimybes ir jų įgyvendinimo MATLAB programinėje įrangoje specifiką, siūlomi tokie kodo, o tuo pačiu ir algoritmo pakeitimai:

*8 lentelė. Siūlomi algoritmo pakeitimai*

| **Nr.** | **Buvęs kodas** | **Siūlomas kodas** |
|---|---|---|
| 1. | `for q=1:qq` | `parfor q=1:qq` |
| 2. | `for j=1:c` | `parfor j=1:c` |
| 3. | `for j=1:c` | `parfor j=1:c` |
| 4. | `for i=1:m-1` | `parfor i=1:m-1` |
| 5. | `for i=1:N` | `parfor i=1:N` |
| 6. | `for i=1:n` | `parfor i=1:n` |
| 7. | `for i=1:n-1` | `parfor i=1:n-1` |
| 8. | `for i=1:n-2` | `parfor i=1:n-2` |
| 9. | `for i=1:n-2` | `parfor i=1:n-2` |
| 10. | `for i=1:n-2` | `parfor i=1:n-2` |

| | | |
|---|---|---|
| 11. | [susikirtimo_taskai1, a(1)]=lygciu_sistema(Kt(j,:), koef, e, g, Sx, Sy, t1(j), T(i+1), K(i+1, :));<br>[susikirtimo_taskai2, a(2)]=lygciu_sistema(Kt(j,:), koef, e, g*(-1), Sx, Sy, T(i), T(i+1),K(i+1,:)); | lyg_g(1)=g;<br>lyg_g(2)=g*(-1);<br>lyg_t(1)=t1(j);<br>lyg_t(2)=T(i);<br>lyg_susikirtimo_taskai(1)=0;<br>lyg_susikirtimo_taskai(2)=0;<br>lyg_a(1)=0;<br>lyg_a(2)=0;<br>parfor lyg_i=1:2<br>[lyg_susikirtimo_taskai(lyg_i), lyg_a(lyg_i)]=lygciu_sistema(Kt(j,:), koef, e, lyg_g(lyg_i), Sx, Sy, lyg_t(lyg_i), T(i+1), K(i+1, :));<br>end<br><br>susikirtimo_taskai1 = lyg_susikirtimo_taskai(1);<br>susikirtimo_taskai2 = lyg_susikirtimo_taskai(2);<br>a=lyg_a; |

*4. 1. 2 Optimizavimo metodo efektyvumo įvertinimas*

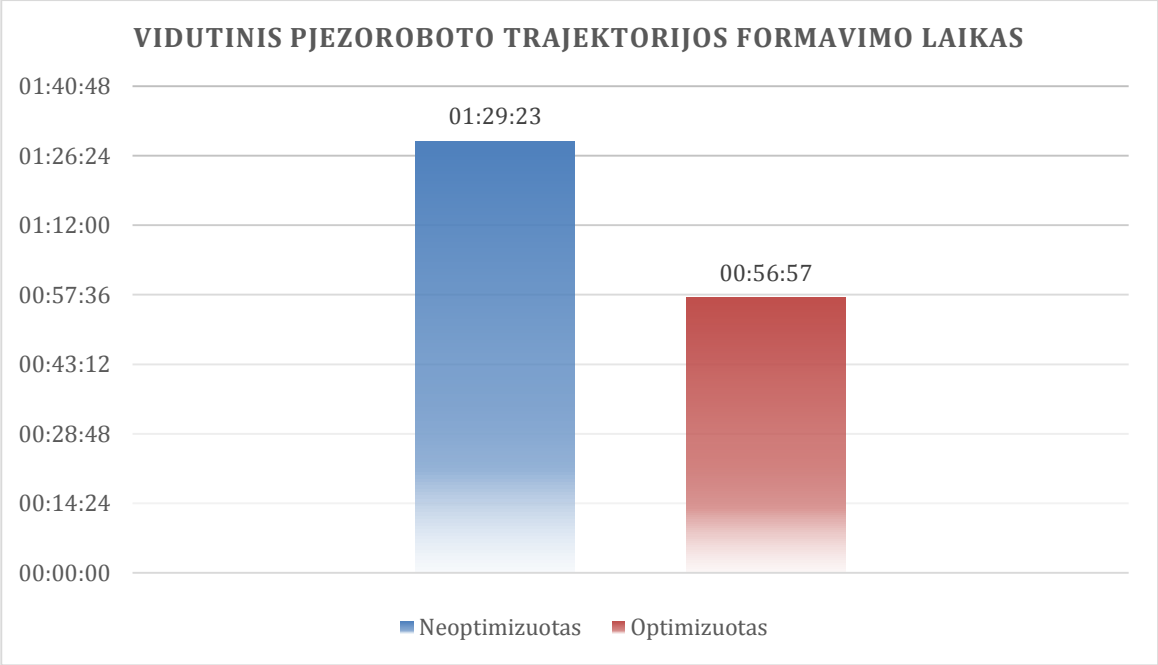
Siekiant įvertinti siūlomų išlygiagretinimo atvejų efektyvumą, buvo atlikti testai, norint įvertinti programinio kodo vykdymo laiko pokyčius (žr. 9 lentelę).

*9 lentelė. Programinio kodo vykdymo laiko pokyčiai*

| **Nr.** | **Testavimo resursas** | **Testinis atvejis** | **Vykdymo laikas prieš išlygiagretinimą** | **Vykdymo laikas po išlygiagretinimo** |
|---|---|---|---|---|
| 1. | 1 kompiuteris | Lygiagretinimo Nr. 1, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 02:01:15 | 02:03:05 |
| 2. | 2 kompiuteris | | 01:15:21 | 01:15:48 |
| 3. | 1 kompiuteris | Lygiagretinimo Nr. 1, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3] (žr. 18 pav.) | 02:10:34 | 02:11:04 |
| 4. | 2 kompiuteris | | 01:15:49 | 01:14:30 |
| 5. | 1 kompiuteris | Lygiagretinimo Nr. 2, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 02:01:15 | 02:02:34 |
| 6. | 2 kompiuteris | | 01:15:21 | 01:16:01 |
| 7. | 1 kompiuteris | Lygiagretinimo Nr. 2, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3] (žr. 24 pav.) | 02:10:34 | 02:09:58 |
| 8. | 2 kompiuteris | | 01:15:49 | 01:16:36 |
| 9. | 1 kompiuteris | Lygiagretinimo Nr. 3, K = [1 1; 2 2; 3 1; 4 2]; (žr. 18 pav.) | 02:01:15 | 02:02:14 |
| 10. | 2 kompiuteris | | 01:15:21 | 01:15:11 |
| 11. | 1 kompiuteris | Lygiagretinimo Nr. 3, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3] (žr. 24 pav.) | 02:10:34 | 02:09:41 |
| 12. | 2 kompiuteris | | 01:15:49 | 01:17:03 |
| 13. | 1 kompiuteris | Lygiagretinimo Nr. 4, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 02:01:15 | 02:00:57 |
| 14. | 2 kompiuteris | | 01:15:21 | 01:16:46 |
| 15. | 1 kompiuteris | Lygiagretinimo Nr. 4, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 02:10:34 | 02:11:34 |
| 16. | 2 kompiuteris | | 01:15:49 | 01:17:24 |
| 17. | 1 kompiuteris | Lygiagretinimo Nr. 5, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 02:01:15 | 02:03:01 |
| 18. | 2 kompiuteris | | 01:15:21 | 01:14:48 |
| 19. | 1 kompiuteris | Lygiagretinimo Nr. 5, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 02:10:34 | 02:14:27 |
| 20. | 2 kompiuteris | | 01:15:49 | 01:16:21 |
| 21. | 1 kompiuteris | Lygiagretinimo Nr. 6, K = [1 1; 2 2; 3 1; 4 2];(žr. 17 pav.) | 02:01:15 | 02:03:25 |
| 22. | 2 kompiuteris | | 01:15:21 | 01:15:41 |
| 23. | 1 kompiuteris | Lygiagretinimo Nr. 6, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 02:10:34 | 02:10:49 |
| 24. | 2 kompiuteris | | 01:15:49 | 01:13:37 |
| 25. | 1 kompiuteris | Lygiagretinimo Nr. 7, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 02:01:15 | 02:01:02 |
| 26. | 2 kompiuteris | | 01:15:21 | 01:14:22 |
| 27. | 1 kompiuteris | | 02:10:34 | 02:14:18 |

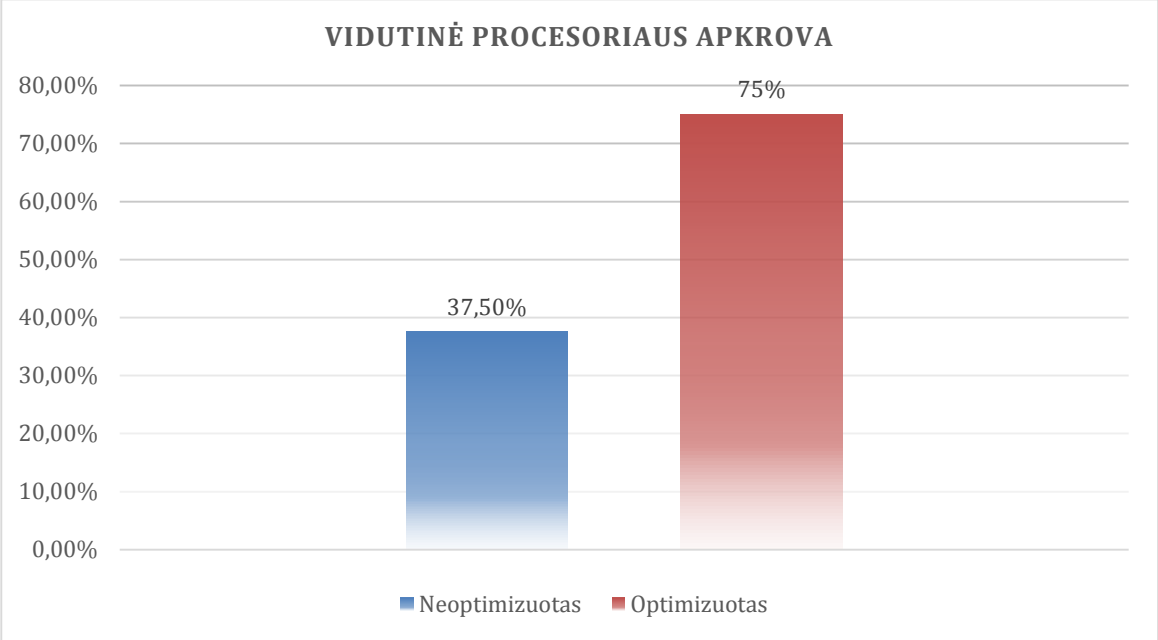
| | | | | |
|---|---|---|---|---|
| 28. | 2 kompiuteris | Lygiagretinimo Nr. 7, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 01:15:49 | 01:16:34 |
| 29. | 1 kompiuteris | Lygiagretinimo Nr. 8, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 02:01:15 | 02:04:16 |
| 30. | 2 kompiuteris | | 01:15:21 | 01:14:45 |
| 31. | 1 kompiuteris | Lygiagretinimo Nr. 8, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 02:10:34 | 02:10:56 |
| 32. | 2 kompiuteris | | 01:15:49 | 01:16:58 |
| 33. | 1 kompiuteris | Lygiagretinimo Nr. 9, K = [1 1; 2 2; 3 1; 4 2];(žr. 17 pav.) | 02:01:15 | 02:00:34 |
| 34. | 2 kompiuteris | | 01:15:21 | 01:14:19 |
| 35. | 1 kompiuteris | Lygiagretinimo Nr. 9, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 02:10:34 | 02:11:55 |
| 36. | 2 kompiuteris | | 01:15:49 | 01:17:13 |
| 37. | 1 kompiuteris | Lygiagretinimo Nr. 10, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 01:15:49 | 01:14:59 |
| 38. | 2 kompiuteris | | 02:01:15 | 02:01:06 |
| 39. | 1 kompiuteris | Lygiagretinimo Nr. 10, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 01:15:21 | 01:17:09 |
| 40. | 2 kompiuteris | | 02:10:34 | 02:12:14 |
| 41. | 1 kompiuteris | Lygiagretinimo Nr. 11, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 01:15:49 | 00:59:50 |
| 42. | 2 kompiuteris | | 01:15:21 | 00:42:55 |
| 43. | 1 kompiuteris | Lygiagretinimo Nr. 11, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 02:10:34 | 01:18:00 |
| 44. | 2 kompiuteris | | 01:15:49 | 00:47:04 |
| 45. | 1 kompiuteris | Lygiagretinimo Nr. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 02:01:15 | 00:59:41 |
| 46. | 2 kompiuteris | | 01:15:21 | 00:44:55 |
| 47. | 1 kompiuteris | Lygiagretinimo Nr. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 02:10:34 | 01:03:12 |
| 48. | 2 kompiuteris | | 01:15:49 | 00:51:03 |

Atlikus testavimą, pastebėta, kad kai kuriais lygiagretumo atvejais (Nr. 1-6, 8, 9, 12, 14-17, 19-23, 27-29, 31, 32, 35, 36, 39, 40) kodo vykdymo laikas pailgėja, o lygiagretinimo atvejis Nr. 11 (žr. 9 lentelę) turi didžiausią teigiamą laiko pokytį. Žemiau pateikiamos diagramos, kaip kito vidutinis pjezoroboto trajektorijos formavimo laikas bei vidutinė procesoriaus apkrova po pakeitimų.




***40 pav.*** *Vidutinis pjezoroboto trajektorijos formavimo laikas*

Tyrimo metu užfiksavus bendrą vidutinę procesoriaus apkrovą (žr. 41 pav.) paaiškėjo, kad vykdant programinį kodą vidutinė apkrova procesoriui padidėjo 100 proc. Pažymėtina, kad kompiuterio darbinė atmintis algoritmo vykdymui įtakos beveik neturėjo.




***41 pav.*** *Vidutinė procesoriaus apkrova*

## 4. 2 Programinio kodo optimizavimas, supaprastinant lygčių sistemos sprendimą

*4. 2. 1 Metodo optimizavimo aprašymas*

Analizuojant pjezoroboto trajektorijos formavimą liestinių metodą buvo pastebėta, kad lygčių sistemos, sprendžiamos funkcijoje lygciu_sistema() (žr. 20 pav.) užima didelę dalį trajektorijos formavimo vykdymo laiko. Todėl buvo pabandyta spręsti problema, apjungti skaičiavimus, laikantis matematinių taisyklių.

1) $f1 = k * (xr - r(1)) - yr + r(2)$

2) $f2 = \frac{-e * kr * g}{\sqrt{1+g^2}} + Sx - xr$

3) $f3 = \frac{e * kr}{\sqrt{1+g^2}} + Sy - yr$

4) $\begin{cases} k * (xr * r(1)) - yr + r(2) = 0 \\ \frac{-e * kr * g}{\sqrt{1+g^2}} + Sx - xr = 0 \\ \frac{e * kr}{\sqrt{1+g^2}} + Sy - yr = 0 \end{cases}$

5) $\begin{cases} xr = -\frac{e * kr * g}{\sqrt{1+g^2}} + Sx \\ yr = \frac{e * kr}{\sqrt{1+g^2}} + Sy \\ k * \left(\left(-\frac{e * kr * g}{\sqrt{1+g^2}} + Sx\right) - r(1)\right) - \left(\frac{e * kr}{\sqrt{1+g^2}} + Sy\right) + r(2) = 0 \end{cases}$

*10 lentelė.* *Pirminė ir pakeista lygčių sistemos*

| **Pirminė lygčių sistema** | **Pakeista lygčių sistema** |
|---|---|
| `f1=k*(xr-r(1))-yr+r(2);`<br>`f2=-e*kr*g/sqrt(1+(g^2))+Sx-xr;`<br>`f3=e*kr/sqrt(1+(g^2))+Sy-yr;`<br>`a=solve(f1,f2,f3);`<br>`X=double(a.xr);`<br>`Y=double(a.yr);` | `f1=k*(-e*kr*g/sqrt(1+(g^2))+Sx-r(1))-e*kr/sqrt(1+(g^2))-Sy+r(2);`<br>`T=double(solve(f1));`<br>`X=double(subs(-e*kr*g/sqrt(1+(g^2))+Sx,T));`<br>`Y=double(subs(e*kr/sqrt(1+(g^2))+Sy,T));` |

Pirminėje lygčių sistemoje pateiktoje su trimis kintamaisiais f1, f2 ir f3 užėmė ilgą laiko tarpą, todėl nuspręsta, suradus bendrą kintamąjį T, visas lygtis sutraukti į vieną lygčių sistemą f1. Teoriškai vienos lygties sprendimas turėtų užimti mažesnį laiko tarpą, nei sprendžiant tokio tipo tris lygtis.

Apskaičiavus parametrą T, prie gautų rezultatų buvo prijungtos lygtis X ir Y apskaičiavimui, panaudojant subs() funkciją, išskaičiuoti xr ir yr kintamieji.

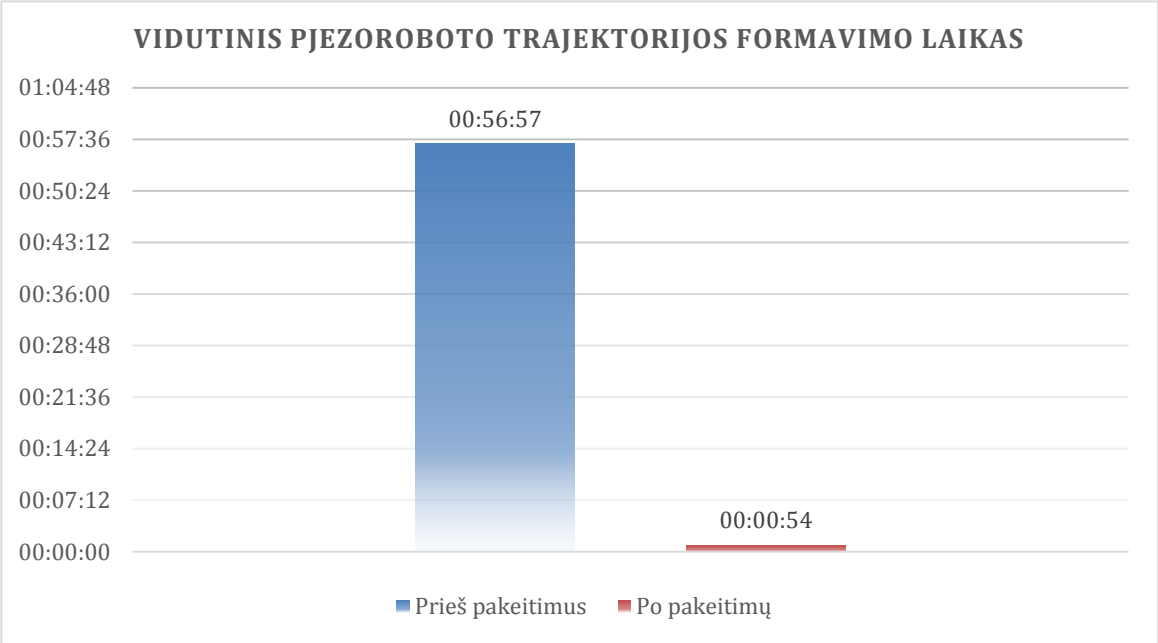
*4. 2. 2 Optimizuoto metodo efektyvumo įvertinimas*

Atlikus programinio kodo pakeitimus buvo atlikta keletas testų, siekiant įvertinti pakeitimų našumą ir teisingumą prie testinio atvejo Nr. 11 pridėjus pakeistą lygčių sistemą. Pateikiamuose testiniuose atvejuose kompiuterio darbinė atmintis kodo vykdymo laikui reikšminės įtakos neturėjo.

*11 lentelė. Optimizuoto metodo efektyvumo įvertinimas*

| **Nr.** | **Testavimo resursas** | **Testinis atvejis** | **Vykdymo laikas prieš pakeitimus** | **Vykdymo laikas po pakeitimų** |
|---|---|---|---|---|
| 1. | 1 kompiuteris | Lygiagretinimo Nr. 11, pakeista lygčių sistema, K = [1 1; 2 2; 3 1; 4 2]; (žr. 17 pav.) | 00:59:50 | 00:00:57 |
| 2. | 2 kompiuteris | | 00:42:55 | 00:00:32 |
| 3. | 1 kompiuteris | Lygiagretinimo Nr. 11, pakeista lygčių sistema, K = [1 1; 2 2; 3 3; 4 1; 5 3; 6 3]; (žr. 18 pav.) | 01:18:00 | 00:01:20 |
| 4. | 2 kompiuteris | | 00:47:04 | 00:00:48 |

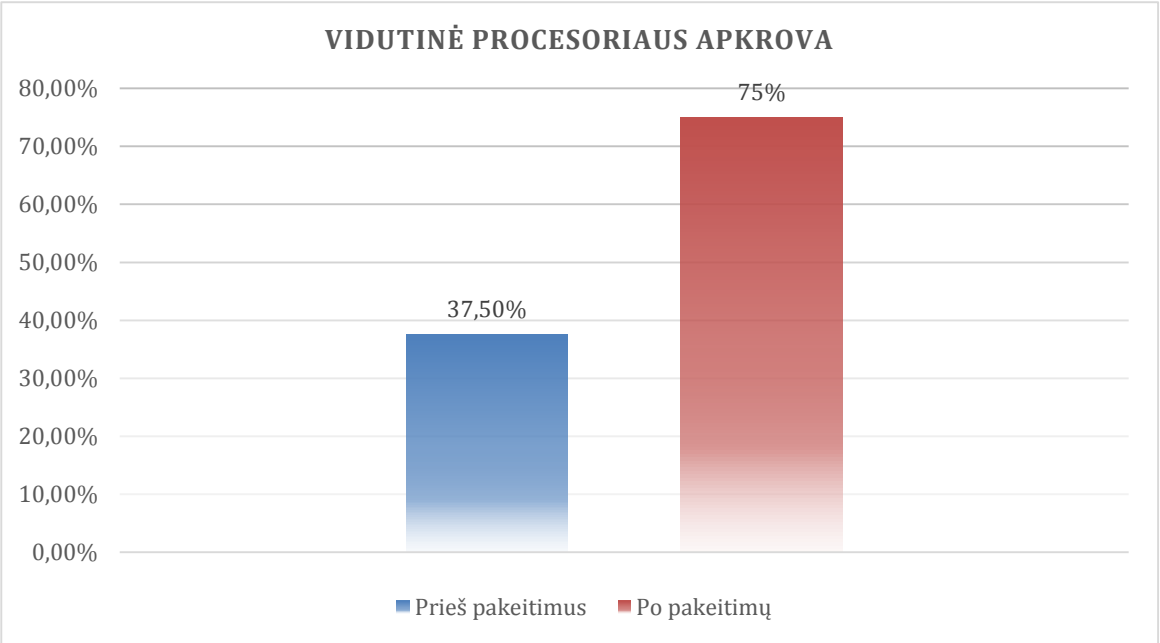
Užfiksavus gautus rezultatus, kurie matyti pateiktoje lentelėje (žr. 11 lentelę), buvo padaryta išvada, kad atlikti pakeitimai lygčių skaičiavime žymiai pagerino kodo vykdymo laiką (žr. 42 pav.).




***42 pav.*** *Vidutinis pjezoroboto trajektorijos formavimo laikas*

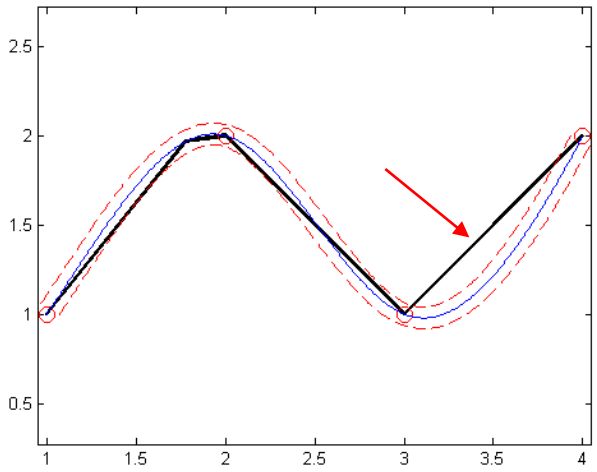
Iš pateiktų rezultatų galima spręsti, kad trajektorijos formavimo laikas drastiškai sumažėjo, bet analizuojant ir tikrinant apskaičiuotas reikšmes, pastebėta, kad ne visos reikšmės apskaičiuojamos teisingai (žr. 44, 45 pav.).

Procesoriaus apkrova formuojant pjezoroboto trajektorijas nepakito (žr. 43 pav.).

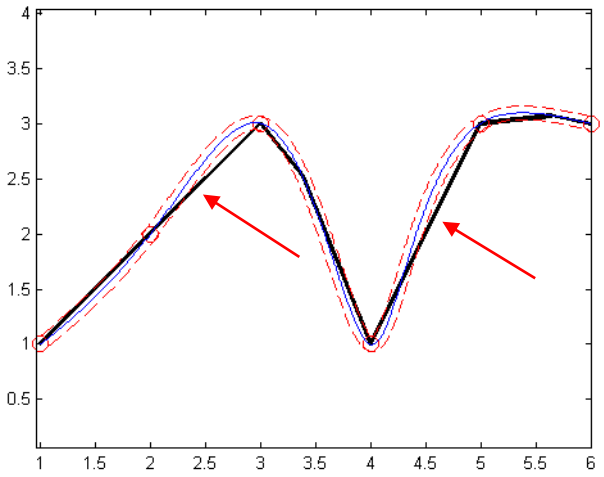



***43 pav.*** *Vidutinė procesoriaus apkrova*

Atlikus tyrimą galima teigti, kad šis optimizavimo būdas nėra tinkamas, nes prarandamas tikslumas, kuris yra būtinas korektiškam pjezoroboto judėjimui. Nekorektiškai apskaičiuotos atkarpos pažymėtos rodyklėmis (žr. 44, 45 pav.).




***44 pav.*** *Nekorektiška 3 atkarpų pjezoroboto judėjimo trajektorija*




***45 pav.*** *Nekorektiška 5 atkarpų pjezoroboto judėjimo trajektorija*

### 4. 3 Siūlomas algoritmo optimizavimas

Pasirinktas lygiagretinimo atvejis Nr. 11 yra tinkamiausias iš galimų optimizavimui variantų, nes įvertinus pjezoroboto trajektorijos formavimo liestinių metodo pakeitimus, matyti, kad programinio kodo optimizavimo bendras vykdymo laikas sumažėjo 36,29 proc., o procesoriaus apkrova padidėjo 100 proc. Gauti duomenys patvirtina, kad pasirinktas optimizavimo būdas yra efektyvus, o apskaičiuojamos reikšmės, reikalingos teisingam pjezoroboto judėjimui – teisingos bei pagrįstos. Darbe sukurtas algoritmas yra brutalios jėgos algoritmas (brute force).

# IŠVADOS

Darbe buvo įgyvendinti šie uždaviniai ir padarytos sekančios išvados:

1. Susipažinus su pjezorobotu, jo judesio trajektorijų formavimo būdais ir raidos ypatybėmis, mechatroniniais įrenginiais, pastebėta, kad šiuo metu yra sukurta nemažai įrenginių, bet jų judėjimo metodai gali būti tobulinami;
2. Išanalizavus MATLAB R2013b paketo, bei kitų programavimo kalbų lygiagretaus programavimo galimybės, galime daryti išvadą, kad lygiagrečių algoritmų realizavimo būdų yra pakankamai, kurie gali būti taikomi įvairioms problemoms spręsti;
3. Atlikus pjezoroboto judesio trajektorijos formavimo liestinių metodu analizę išsiaiškinta, kaip veikia analizuotas algoritmas, todėl galima daryti išvadą, kad metodas nėra efektyvus ir gali būti tobulinamas;
4. Identifikavus programinio kodo probleminės vietas, įvardyti algoritmo optimizavimo būdai, todėl galime daryti išvadą, kad priimti sprendimo būdai paspartins vykdymo laiką;
5. Sukūrus optimizuotą pjezoroboto judesio trajektorijos formavimo metodą atlikta išsami optimizuoto metodo analizė bei pastebėta, kad pasiūlytas naujas metodas, pagreitino trajektorijos formavimą 36,29 proc. laiko ir padidino procesoriaus išnaudojimą 100 proc.

Darbe taip pat buvo naudojama literatūros apžvalga, lyginamoji, sisteminė, skaitinė ir eksperimentinė analizės.

# LITERATŪRA


1. Barney B. Introduction to Paralle Computing. Lawrence Livermore National Laboratory. 2012 m. [žiūrėta: 2012-12-13]. Prieiga per internetą: < https://computing.llnl.gov/tutorials/parallel_comp/ />
2. Čiegis R. Lygiagretieji algoritmai ir tinklinės technologijos. Vilnius. 2005 m.
3. Fashanu T.A., Ale F., Agboola O.A., Ibidapo-Obe O. 2012. Performance Analysis of a Parallel Computing Algorithm Developed for Space Weather Simulation. International Journal of Advancements in Research & Technology, Vol. 1(7). 2012 m.
4. MathWorks. 2012. Parallel Computing Toolbox. Perform parallel computations on multicore computers, GPUs, and computer clusters. [žiūrėta: 2012-12-16]. Prieiga per internetą: < http://www.mathworks.com/products/parallel-computing />
5. Šilingas D. Lygiagretus daugialypis integravimas su Java kalba. VDU. [žiūrėta: 2012-12-17]. Preiga per internetą: < http://vaidila.vdu.lt/~i5dasi/parallel/javathreads.doc />
6. Thread Programming in Java | Java Programming Resources | Source Code Mania.COM. [žiūrėta: 2012-12-17]. Prieiga per internetą: < http://sourcecodemania.com/thread-programming-in-java/ />
7. Asta Drukteinienė. Daktaro disertacija. Nanometrų skyros judančių daugiamačių pjezorobotų trajektorijų formavimas. Vilnius, 2011. [žiūrėta: 2013-04-15]. Prieiga per internetą: < http://vddb.laba.lt/obj/LT-eLABa-0001:E.02~2011~D_20111207_132201-45128 />
8. Axel Bürkle, Sergej Fatikow. Computer Vision Based Control System of a Piezoelectric Microrobot. Institute for Process Control and Robotics, University of Karlsruhe.
9. T. Kanda 1, A. Makino 1, K. Suzumori 1, T. Morita 2 and M. K. Kurosawa 3. A Cylindrical Micro Ultrasonic Motor using a Micro-machined Bulk Piezoelectric Transducer. Department of Systems Engineering, Faculty of Engineering, Okayama University, Okayama, Japan. 2004.
10. Bruce R. Donald, Member, IEEE, Christopher G. Levey, Member, IEEE, Craig D. McGray, Member, IEEE, Igor Paprotny, and Daniela Rus. An Untethered, Electrostatic, Globally Controllable MEMS Micro-Robot. 2006.
11. Moss D. G., Efficient C/C++ Coding Techniques. 2001. [žiūrėta: 2014-03-15]. Prieiga per internetą: < http://www.open-std.org/jtc1/sc22/wg21/docs/ESC_Boston_01_304_paper.pdf />
12. Stephen Gilmore, Advances in Programming Languages: Efficiency. 2007. [žiūrėta: 2014-03-24]. Prieiga per internetą: < http://homepages.inf.ed.ac.uk/stg/teaching/apl/handouts/efficiency.pdf />
13. Tomas Šeirys, Simona Ramanauskaitė, Objektų ir masyvų naudojimo PHP kalboje našumo tyrimas. 2013. [žiūrėta: 2014-03-24]. Prieiga per internetą: < http://vddb.library.lt/fedora/get/LT-eLABa-0001:J.04~2013~ISSN_1648-8776.N_2_40.PG_134-137/DS.002.0.01.ARTIC/>
14. Techniques for Improving Performance. [žiūrėta: 2014-03-25]. Prieiga per internetą: < http://mathworks.se/help/matlab/matlab_prog/techniques-for-improving-performance.html />
15. Performance Optimization. [žiūrėta 2014-04-10]. Prieiga per internetą: < https://dev.day.com/docs/en/cq/current/deploying/performance.html />
16. R. Bansevicius, G. Kulvietis, D. Mazeika, A. Drukteiniene, A. Grigoravicius. Cylindrical piezoelectric mobile actuator based on travelling wave. ISSN 1392 - 1207. MECHANIKA. 2012 Volume 18(5): 554-560.

17. Ramutis Bansevicius, Asta Drukteiniene, Genadijus Kulvietis and Inga Tumasoniene. Design of a Mobile Microrobot Based on Standing and Travelling Waves. International Journal of Advanced Robotic Systems, 2013. [žiūrėta 2014-04-12]. Prieiga per internetą: < http://cdn.intechopen.com/pdfs-wm/44460.pdf />
18. R. Bansevicius, A. Drukteiniene, G. Kulvietis. Path-planning algorithms analysis of hemispheric mobile piezorobot. 2011. [žiūrėta 2014-04-12]. Prieiga per internetą: < http://isd.ktu.lt/it2011/material/Research/2_FM_1.pdf />