KLAIPĖDOS UNIVERSITETAS
GAMTOS IR MATEMATIKOS MOKSLŲ FAKULTETAS
INFORMATIKOS KATEDRA

REGIMANTAS KONTRIMAS
InfMag04 gr. studentas

MECHATRONIKOS UŽDAVINIŲ SPRENDIMAS, NAUDOJANT IŠSKIRSTYTĄ, REALIZUOJANČIĄ LYGIAGREČIUS SKAIČIAVIMUS, SISTEMĄ

Baigiamasis magistro darbas

Darbo vadovas doc. dr. Arūnas Andziulis

Darbo konsultantai:
moksl. darb. Pranas Mažeika
Kompiuterių centro direktorius Rolandas Garška
Kompiuterių centro tinklų inžinierius Valdas Jankauskas

KLAIPĖDA, 2006

ANOTACIJA

Šiame magistriniame darbe sprendžiama išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos netaikymo mechatronikos uždavinių sprendimui Klaipėdos universiteto Mechatronikos mokslo institute problema. Suformuluotas tikslas ir hipotezė, kurie leis įtvirtinti tokios sistemos taikymą, remiantis iškeltais uždaviniais. Darbe detaliai analizuojama išskirstytų sistemų veikimo teoriniai ir praktiniai principai. Atliekamas teorinis ir eksperimentinis tyrimas tik labiau leis paspartinti tokių metodų naudojimą.
PAGRINDINIAI ŽODŽIAI: *išskirstymas, padalinimas, sistema, lygiagretus vykdymas, ANSYS, mechatronika.*

ANNOTATION

In this Master Job solving problem of don‘t use distributed, realized parallel computing, system for mechatronics tasks in Mechatronic Institute of Klaipeda university. Stated purpose and hypothesis that’s let to embed using of this system, with reference of raised tasks. In job detail analyzing distributed systems action and practical principles. The performing theoretical and experiential research let to accelerate using of these methods.
KEYWORDS: *distribute, share, system, parallel computing, ANSYS, mechatronic.*

TURINYS

ĮVADAS

Informacinės technologijos yra viena iš pagrindinių teorinių metodų ir instrumentų dedamųjų šiuolaikinėje mechatronikoje, nes mechatronika – tai mokslas, apjungiantis mechaninės inžinerijos, valdymo teorijos, kompiuterių mokslo ir elektronikos sritis, nukreiptas į sudėtingų, neapibrėžtų ir sąveikaujančių inžinerinių sistemų valdymą [1]. Ji, apjungdama daugelį mokslo šakų, sudaro mechatronikos sistemą, kuri yra labai sudėtinga. Terminas *mechatronikos sistemos* savyje integruoja tokias keliamas sudėtines problemas, kaip laisvės laipsniai, kinematiškumas, pavaros ir transmisijos dispozicija, techninė ir programinė įranga [2]. Tikslinga sutapatinti mechatronikos sistemą su mechatronikos uždaviniu, kadangi mechatronikos sistema apima konkretų projektavimą. Šiuo metu mechatronikos sistemos yra populiari ir ypatingai gvildenama tema įvairių mokslininkų darbuose. Tai parodo mokslo publikacijų, esančių duomenų bazėse, skaičius.

Mokslininkų darbuose keliamos problemos [1, 2] ir pateikiami sprendimo variantai leidžia teigti, kad mechatronikos sistemos ir su jomis susijos problemos yra labai aktualios šiandieniame pasaulyje. Todėl mechatronikos uždavinio sprendimą būtina apžvelgti šiais aspektais-problemomis, kurios, mano nuomone, yra pagrindinės:

- n-laisvės laipsniai,
- techninė įranga,
- programinė įranga.

Toks aspektų-problemų atskyrimas leidžia uždavinius nagrinėti pasitelkiant informacines technologijas.

Klaipėdos universitete įsteigus Mechatronikos mokslo institutą (jis įkurtas 2005 m. sausio 28 d. Klaipėdos universiteto Senato nutarimu Nr. 11-22 [3]), buvo susidurta su rimtais mechatronikos uždaviniais. Vienas iš tokių uždavinių yra mechatronikos sistema – „Krumplinės pavaros darbo patikimumo analizė ir defektų diagnostika“.

Mechatronikos institute, nagrinėjant vis sudėtingesnes mechatronikos sistemas, susidurta su klasikinėmis, pasaulyje gerai žinomomis, problemomis [1, 2] tokiomis, kaip techninių ir programinių resursų trūkumas arba pritaikymas, globaliems uždaviniams su n-laisvės laipsniais [4]. Todėl matematiniam formulavimui sprendimui reikalingi milžiniški resursai.

Sprendžiant sudėtingas mechatronikos sistemas reikia taikyti išskirstytą sistemą – Klaipėdos universiteto Mechatronikos mokslo instituto atveju - *ANSYS* programinę įrangą. Tokios išskirstytos sistemos, realizuojančios lygiagrečius skaičiavimus (lygiagrečių skaičiavimų naudojimas susijęs su laisvės laipsniais), taikymas, pirmiausia leistų gauti bei rasti optimaliai

tikslų sprendimą, nagrinėjamu atveju, kai matematiniais modeliais siekiama aprašyti n-mates judesio sistemas.

Problemos aktualumas ir mokslo problema

Sprendžiant sudėtingus mechatronikos uždavinius neišvengiamai tenka taikyti išskirstytas sistemas [1, 2]. Su tokiomis problemomis, kai neįmanoma gauti atsakymo dėl tam tikrų techninių arba programinių resursų trūkumo, būtina ieškoti kito sprendimo būdo. Mechatronikos instituto sprendžiamų uždavinių sudėtingumas leidžia ir kartu iššaukia būtinumą naudoti išskirstytą, lygiagrečius skaičiavimus realizuojančią sistemą [3]. Darbo aktualumas grindžiamas sprendimo paieškos sudėtingų sistemų-uždavinių sprendimui institute reikalingumu.

Lygiagrečių skaičiavimų naudojimas padarė didelę įtaką perėjimui nuo mokslo ir inžinerijos skaičiavimų modeliavimo prie komercinės realizacijos [7]. Todėl galima sakyti, kad tokios perspektyvos atsisakymas užkirstų kelią resursų ir laiko optimizavimui.

Tyrimo problema – išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos netaikymas mechatronikos uždavinių sprendimui Mechatronikos mokslo institute. Problemos pagrindas – techninių ir laiko resursų trūkumas.

Tyrimo hipotezė

Ar tinka *ANSYS* mechatronikos uždavinių sprendimui, kaip išskirstyta, realizuojanti lygiagrečius skaičiavimus, sistema

Tikslas

Išplėtoti mechatronikos uždavinių sprendimo procesą Klaipėdos universiteto Mechatronikos mokslo institute, konkrečiai „Krumplinės pavaros darbo patikimumo analizės ir defektų diagnostikos“ sistemai, taip optimizuojant techninius ir laiko resursus.

Tyrimo uždaviniai

1. Atlikti problemos mokslinę analizę: išnagrinėti išskirstytų, realizuojančių lygiagrečius skaičiavimus, sistemų architektūros modelius ir jų taikymą;
2. Parinkti ir detalizuoti atminties architektūrą, tinkamiausią lygiagretiems skaičiavimams mechatronikos uždavinių sprendimuose (konkrečiai „Krumplinės pavaros darbo patikimumo analizės ir defektų diagnostikos“ sistemos sprendime);
3. Atlikti *ANSYS* kaip išskirstytos sistemos veikimo analizę, pagal techninius ir programinius parametrus;
4. Parinkti ir detalizuoti tinkamiausią *ANSYS* kaip išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos tinklo architektūrą;
5. Atlikti bandymus, taikant skirtingas atminties architektūras ir kt. bei pateikti rezultatus, susijusius su mechatronikos uždavinių sprendimu taikant ANSYS kaip išskirstytą sistemą;

6. Pateikti bandymo rezultatų techninių ir vaizdinių duomenų analizę.

Problemos naujumas ir praktinė reikšmė

Mechatronika, būdama pakankamai jauna ir perspektyvi sritis, formuoja problemas, kurioms spręsti pasitelkiami kiti mokslai. Todėl ne išimtis išskirstytų sistemų taikymas mechatronikos uždavinių sprendimui. Žvelgiant į šią sąsaja lokaliu žvilgsniu, Klaipėdos universiteto Mechatronikos institute sprendžiamų uždavinių optimizavimui ir sprendimo proceso pagerinimui reikalingas išskirstytos sistemos taikymas, nes tai naujas bei perspektyvus požiūris į uždavinio sprendimą [3]. Tai iki šiol nebuvo taikyta ir konkrečiais bandymais neįrodyta. Žvelgiant globaliai, mechatronikos uždavinių sprendimas, naudojant išskirstytas, realizuojančias lygiagrečius skaičiavimus, sistemas yra plačiai vystomas [5, 6].

Tyrimo objektas **–** išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos naudojimas mechatronikos uždavinio - „Krumplinės pavaros darbo patikimumo analizės ir defektų diagnostikos“ sprendimui Klaipėdos universiteto Mechatronikos mokslo institute.

Eksperimentinio tyrimo medžiaga ir vieta

Tyrimo metu buvo naudojami šie metodai:

1. žinomų mokslinių faktų analizė;
2. mokslinės literatūros apibendrinimas;
3. eksperimentas.

Tyrimą sudaro trys pagrindines dalis:

- mokslinės literatūros analizė,
- problemos tyrimas-eksperimentas,
- tyrimo rezultatų aptarimas.

Tyrimu siekiama nustatyti išskirstytų sistemų taikymą ir jo naudingumą mechatronikos sistemose, taip plėtojant mechatronikos mokslą, konkrečiai Mechatronikos mokslo instituto veiklą.

Mechatronikos institutas yra strategiškai svarbus Klaipėdai ir visai Vakarų Lietuvai – tai parodo jo gaunamų užsakymų gausa.

1. MOKSLINĖS LITERATŪROS ANALIZĖ

Problemai spręsti buvo atlikta mokslinės literatūros (naudotos duomenų bazės *ScienceDirecr, IEEE* ir kt.) apžvalga, siekiant išsiaiškinti, kaip modernios informacinės technologijos detalizuoja išskirstytą, realizuojančią lygiagrečius skaičiavimus, sistemą. Dėmesys kreipiamas į tai, kaip šiuolaikinėje mokslinėje literatūroje akcentuojami bendri teoriniai pasirinkimo motyvai, kurie konkretizuojami ir detalizuojami eksperimentinio tyrimo aprašymo dalyje.

Ieškoma sąšaukų su magistriniame darbe nagrinėjama sistema – „Krumplinės pavaros darbo patikimumo analizė ir defektų diagnostika“ [9], kur uždavinys aprašomas netiesinėmis lygtimis, kurių kiekviena detalizuoja krumplinės pavaros paviršiaus taškų n-laisvės laipsnius. Buvo svarbu rasti analogą, sudarant netiesinių lygčių sistemą, kuria aprašomas uždavinio imitacinis modelis, reikia naudoti lygiagrečius skaičiavimus. Todėl svarbu išsiaiškinti, kokia mokslinė patirtis šios problemos taikyme.

1.1. Lygiagretumo ir išskirstymo apjungimas

Šiuo metu išskirstytos sistemos sąvoka ypač paplitusi. 1970 metais buvo pradėta naudoti kompiuterių sistemą skirtingoms operacijos atlikti vienu metu. Tokių kompiuterių sistemų panaudojimas buvo pirmasis žingsnis link išskirstytų sistemų, t.y. vartotojai galėjo dalintis resursais ir gauti priėjimą prie informacijos. Šiandien išskirstytų sistemų sąvoka tampa dar platesnė ir aktualesnė. Aktualus yra tinklo architektūros parinkimas, siekiant optimizuoti išskirstytos sistemos veikimą. Pereinama prie smulkesnių detalių, tokių svarbių kaip laikas, procesorius, atmintis ir kitų algoritmų formavimas [8].

Išskirstytų sistemų naudojimas mechatronikos uždavinių sprendimui yra plačiai taikomas pasaulyje, tačiau vis dar susiduriama su problemomis, siekiant rasti tinkamiausią išskirstytos sistemos veikimą. Tai atsispindi mokslininkų publikacijų gausoje [5, 6, 7]. Siekiant optimizuoti mechatronikos sistemos veikimą būtina taikyti išskirstytą sistemą [8].

Išskirstytos sistemos taikymas siejamas su techninių resursų optimizavimu. Norint išspręsti sudėtingą mechatronikos sistemą yra būtini dideli techniniai bei programiniai resursai. Tokių resursų trūkumas neleidžia spręsti uždavinių, o vienas superkompiuteris – tai didelės išlaidos, dažniausiai neduodančios teigiamų rezultatų. Daugelis mokslininkų teigia, kad tokių superkompiuterių problemą gali išspręsti tik išskirstyta sistema, suprojektuota pagal visus programinės įrangos reikalavimus. Todėl siekiama rasti vieningą išskirstytos sistemos architektūrą mechatronikos uždavinių sprendimui.

Lygiagretumas nurodo vienalaikį tos pačios programos skirtingų komandų vykdymą [10]. Kitaip tariant, lygiagretumas yra technika, skirta kompiuterinių tinklų imitavimui. Išskirstymas nurodo programos duomenų arba programinio kodo (arba abiejų) skirtingiems kompiuteriams padalinimą. Norint naudoti išdalinimą būtinas lygiagretumas. Lygiagretumas ir išskirstymas gali būti naudojami kartu ir atskirai. Labiau detalizuojant, išdalinimas (*scalability*) priklauso nuo procesorių skaičiaus/mazgų, kurie yra susieti. Kai išskirstymas nėra naudojamas, lygiagretumas paprastai yra taikomas padalijamos atminties atveju [11].

1.2. Išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos pagrindiniai veiksniai

Didelio našumo sistemų paplitimas ir greitų (našių) tinklų pasirodymas (tera-bitiniai tinklai) leido dar labiau išplėtoti lygiagrečius ir išskirstytus skaičiavimus. Pagrindiniai veiksniai būtų tokie, kaip

- vykdymo technologijų pažanga,
- greitaeigių tinklų atsiradimas ir naudojimas,
- augantys tyrinėjimų skaičius, skirtas programinės įrangos vystymo palaikymo ir programavimo išskirstytų skaičiavimų sąlygomis [12].

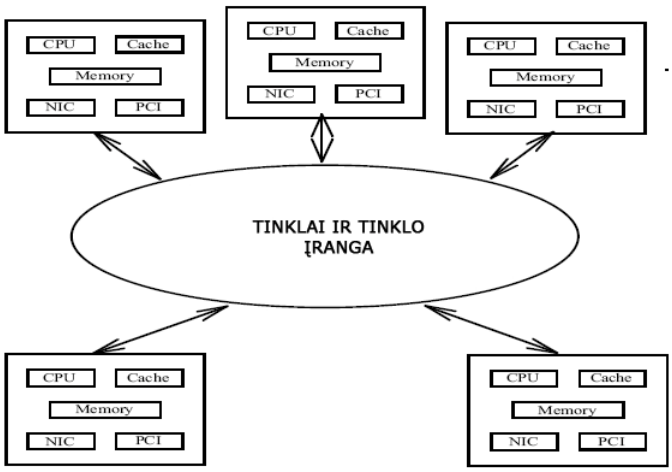
Šie trys pasiūlyti veiksniai ir nulemia išskirstytų sistemų, realizuojančių lygiagrečius skaičiavimus reikalingumą.

Mūsų atveju, kalbant apie veikimo technologiją, išskirstytos sistemos veikimo galia priklauso nuo kiekvieno tinkle esančio mazgo-taško. Daugelis autorių vienareikšmiškai teigia, kad išskirstytos sistemos veikimas priklauso nuo kiekvieno tinklo mazgo-taško techninių ir programinių resursų [12, 13]. Paieškos, susijusios su pagrindinės atminties technologijomis, didelio našumo diskų masyvais ir greitaeigėmis įvedimo ir išvedimo sistemomis ir tai yra labai svarbu veikimo technologijos pažangai ir rentabiliam didelio našumo išskirstytų sistemų vystymui.

Siekiant užsibrėžto tikslo, svarbu žinoti, kad išskirstytų algoritmų vykdymas priklauso nuo kuo didesnio kanalo pralaidumo ir nuo ryšio būsenos (sąveikos) tarp tinklo mazgų. Geresnį pralaidumas ir ryšio būsena priklauso ne tik nuo įrangos greitumo (spartumo), bet ir nuo efektyvaus protokolų ryšio (sąsajos). Tai sumenkina programinės įrangos sudėtingumas (apkrovimas) (*overhead*). Didelio pralaidumo tinklų vystymas leidžia naudoti gigabaitinius pralaidumo kanalus ir taip išauga didesnės galimybės didelio našumo išskirstytų sistemų panaudojime [12].

Išskirstyta sistema – tai sistema, susidedanti iš kompiuterių, kurie neturi padalintos bendros atminties ar sutampančių laikų (vienodos sinchronizacijos). Kompiuteriai gali naudotis

nutolusiais resursais taip kaip vietiniais (lokaliais) išskirstytoje sistemoje [14]. Paprastai yra blogiau naudotis nutolusiais resursais, nei vietiniais (lokaliais), nes gali atsirasti ryšio trikdžiai ir procesoriaus apkrovos per ryšio protokolą. 1 pav. vaizduojama išskirstytos sistemos architektūra.



1 pav. ***Išskirstytos sistemos architektūra*** **[14]**

Išskirstytų sistemos yra naudingos dėl savo nedidelės kainos, taip pat didelio našumo kompiuteriais ir tinklo įrenginiais. Kelių kompiuterių, sujungtų į vieną tinklą pilna skaičiavimo galia gali būti milžiniška. Visa išskirstyta sistema pasiekia didesnį našumą, negu keletas superkompiuterių.

Išskirstytos sistemos pasižymi šiomis savybėmis:

1. Konkurencija (*Concurency*) – išskirstytų skaičiavimų komponentės startuoja vienu metu.
2. Nepriklausomos nesėkmės būsenos (*Independent failure modes*) – išskirstytų skaičiavimų komponentės ir tinklo palaikymas gali žlugti nepriklausomai viena nuo kitos.
3. Nėra globalaus laiko (*No global time*) – kiekviena sistemos komponentė turi savo lokalų laikrodį, bet laikrodžiai gali būti, kad nefiksuoja to paties laiko. Techninė įranga, kurios pagrindu veikia laikrodis negali garantuoti bendro visos sistemos tikslumo.
4. Ryšio užlaikymas (*Communications delay*) – tai yra toks laiko efektas, kai įvykis iš vieno išskirstytos sistemos taško paplinta po visą sistemą.
5. Nesuderinamumo būsena (*Inconsistent state*) – konkurencija, nesėkmės (žlugimai), ryšio užlaikymas vienu metu vykdant išskirstytus skaičiavimus reiškia, kad nėra nuoseklumo visos sistemos atžvilgiu [15].

Iš pateiktų išskirstytų sistemų pagrindinių savybių galime daryti išvadą, kad išskirstytose sistemose nesama tam tikro išskirstytų skaičiavimų nuoseklumo vykdymo, tačiau atminties ir procesoriaus apkrovos turi būti valdomos tiksliai [16].

1.3. Išskirstytos sistemos projektavimo karkasas

Išskirstytos sistemos projektavimo karkase (*distributed system design framework*) labiausiai akcentuojami architektūriniai objektai, aptarnavimas ir kandidatinės technologijos, padedančios realizuoti pagrindines komponentes išskirstytose skaičiavimo sistemose [17]. Detaliau susipažinus, matome, kad išskirstytos sistemos projektavimo procesas apima pagrindines tris veiklas:

- ryšio sistemos (tinklo, komunikacijos) projektavimas leidžia išskirstytos sistemos resursais ir objektais, apsikeičiant informaciją;
- paaiškina sistemos struktūrą (architektūrą) ir sistemos servisus, kurie leidžia keletui kompiuterių veikti kaip vienai sistemai;
- o taip pat paaiškina išskirstytos sistemos, realizuojančios lygiagrečius skaičiavimus, programavimo technikas.

Išskirstytos sistemos projektavimo karkasas pateiktas 1 lentelėje.

1 lentelė. ***Išskirstytos sistemos projektavimo karkasas*** **[17]**

<table>
<tr><td colspan="4">IŠSKIRSTYTOS SISTEMOS PARADIGMOS</td></tr>
<tr><td colspan="2">Skaičiavimo modeliai</td><td colspan="2">Ryšio modeliai</td></tr>
<tr><td>Funkcinis lygiagretumas</td><td>Duomenų lygiagretumas</td><td>Pranešimo perdavimas</td><td>Padalinta atmintis</td></tr>
<tr><td colspan="4">SISTEMOS ARCHITEKTŪRA IR VEIKIMO PRINCIPAI</td></tr>
<tr><td colspan="2">Architektūros modeliai</td><td colspan="2">Veikimas sisteminiame lygyje</td></tr>
<tr><td colspan="4">KOMPIUTERINIS TINKLAS IR PROTOKOLAI</td></tr>
<tr><td colspan="2">Tinklo tinklai</td><td colspan="2">Ryšio protokolai</td></tr>
</table>

Išskirstytos sistemos veikimas grindžiamas pasiūlytu karkasu [17]. Išskiriami trys pagrindiniai sluoksniai, kurie ir charakterizuoja sistemos veikimą bei panaudojimą lygiagretiems skaičiavimams atlikti. Naudodami šį karkasą, atliksime tolesnę analizę.

1.4. Išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos paradigmų sluoksnis

Išskirstytos sistemos paradigmos charakterizuoja pagrindinius skaičiavimo ir ryšio modelius. Išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos yra apibrėžiamos pagal dvi paradigmas: funkcinį ir duomenų (informacijos) lygiagretumą.

Kalbant apie ryšio modelius, išskirstyta sistema gali būti vaizduojama (traktuojama) kaip grafas. Būtent sistema yra vaizduojama kaip grafas, kur įrenginius atitinka viršūnės, o ryšius – briaunos. Tarkime G = (V,E) bus grafas, kur V – viršūnės, o E – briaunų aibė [18].

Daugelis autorių siūlo įgyvendinti išskirstytas sistemas pagal tokias topologijas [14]:

1. pilnas n narinis medis;
2. žiedas.

Kiekvieną iš šių terminologijų yra charakterizuojama taip:

1. pilnas n narinis medis – tai medis, kur mazgai turi n vaikų mazgų ir visi žemiausi lygiai yra pilni, visi lygiai yra užpildomi iš kairės į dešinę [19];

2. žiedo atveju visi mazgai yra sujungti vienas su kitu uždaru ciklu, vienas mazgas yra sujungtas su dviem kitais mazgais ir vienas su visais [20].

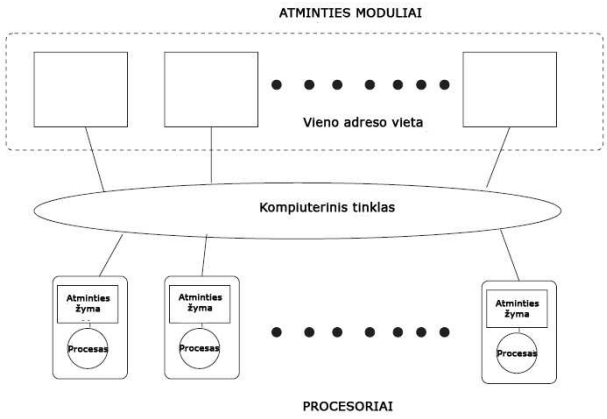
Kalbant apie išskirstytas sistemas būtina apibrėžti išskirstytas/padalintas atmintis. Išskirstytos atminties architektūra pavaizduota 2 paveiksle [17]. Išskirstytos atminties atveju egzistuoja sistema, kuri susideda iš p procesorių $P_0, P_1, P_2, ..., P_{p-1}$. Kiekvienas iš šių procesorių turi lokalią atmintį ir neturi jokios globalios padalintos atminties. Kaip vienas iš tokių procesorių tarpusavio komunikavimo būdu yra pranešimų siuntimas. Skaičiavimai ir ryšiai išskirstytoje sistemoje yra globaliai suskirstyti tam tikrais žingsniais. Skaičiavimo žingsnyje kiekvienas iš procesorių atlieka aritmetinius/loginius veiksmus arba yra neužimtas, o ryšio atveju – procesoriai siunčia ir priima pranešimus [21]. Ryšio žingsnis gali būti aprašytas tokia išraiška (žr. 1 formulę):

$$((\pi(0),v0),(\pi(1),v1),(\pi(2),v2),...,(\pi(p-1),vp-1)) \text{ [13]} \quad (1),$$

kur visiems $0 \leq j \leq p - 1$, procesorius P_j siunčia reikšmę v_j procesoriui $P_{\pi(j)}$ ir π yra atvaizduojamas į π: $\{0, 1, 2, ..., p - 1\} \rightarrow \{-1, 0, 1, 2, ..., p-1\}$.

Didžiausią susidomėjimą kelia vykdomi lygiagretūs skaičiavimai išskirstytos atminties sistemoje, kur T_1 skaičiavimo žingsniai ir T_2 – ryšio žingsniai, tuomet skaičiavimo laikas yra (žr. 2 formulę)

$$O\ (T_1 + T(p)T_2) \quad (2)$$



2 pav. ***Išskirstytos atminties architektūra*** **[17]**

Iš algoritminės pusės žiūrint, išskirstytos atminties sistema yra charakterizuojama funkcija *T(p),* kuri ir nustato ryšio pajėgumą (pralaidumą) tarpusavio ryšio tinkle.

1.5. Atminties architektūros ir valdymas

Yra skiriamos kelios atminties architektūros [14]:

- Padalintos atminties architektūra (Shared Memory) – pagal padalintos atminties sąlygą, vienos padalintos atminties adreso vieta yra prieinama visiems procesoriams. Kiekvienas procesorius dalinasi atmintimi su kitais. Gan įprastu pavyzdžiu gali būti IA32 Windows mašina su dviem procesoriais. Procesoriai dalinasi ta pačia pagrindine atmintimi per šynos architektūrą.
- Išskirstyta atmintis (Distributed Memory) – pagal išskirstytos atminties sąlygą, kiekvienas procesorius arba duomenų apdorojimo mazgas, turi vieną atminties adreso vietą, kuria nesidalina su kitais procesoriais ar duomenų apdorojimo mazgais. Tinkle ryšys tarp mašinų yra palaikomas Message Passing sąsajos (Message Passing Interface-MPI). Įprastu išskirstytos atminties sistemos pavyzdžiu gali būti rinkinys stalinių kompiuterių, sujungtų į viena tinklą. Kai sujungti procesoriai veikia kaip skaičiavimo mašina – tai jau vadinama klasteriu (blokiniu).
- Mišri atmintis (Mixed Memory) – parodo, kad klasteryje (blokinyje) naudojama tiek padalinta, tiek paskirstyta atmintis. Puikus mišrios atminties pavyzdys gali būti IA64 sistema su dviem procesoriais, kiekvienas iš jų atskirame korpuse, padalinta atmintis, bet kiekvienas atskirai pajungtas į tinklą [16, 23].

Literatūros, apie padalintą ir išskirstytą atmintis, analizės apibendrinimas pateiktas 2 lentelėje.

2 lentelė. ***Programos perėjimas nuo padalintos atminties modelio prie išskirstyto modelio***

Programos vykdymas	**Padalintos atminties architektūros lygiagretaus veikimo programa**	**Išskirstytos atminties architektūros lygiagretaus veikimo programa**	**Reikalingi veiksmai, siekiant perėjimo**
Nuoseklioji dalis	Pagrindinis mazgas vykdo šią dalį pagal nutylėjimą (nėra tikslaus aprašymo)	Pagrindinis mazgas vykdo tai, tik po to patikrinimo, ar jis yra pagrindinis mazgas (aiškus sąlygos tikrinimas)	Sąlygos sakinio įterpimas, ar jis yra pagrindinis. ar ne
Dviguba (visiška) kilpa	Bet kuris mazgas turi priėjimą prie bet kokių duomenų (nėra aiškaus išdalinimo)	Kiekvienas mazgas turi teisę prieiti tik prie jo vienintelio duomenų kopijos (aiškus išskirstymas)	Įterpimas MPI_Bcast arba MPI_Send ir MPI_Recv
	Doall funkcijos iškvietimas	Kiekvienas mazgas vykdomas lygiagrečiai pagal nutylėjimą (nėra visko vykdymo vienu metu)	Pilnas pašalinimas
Nuoseklioji dalis	Pagrindinis mazgas turi priėjimą prie visų rezultatų (nėra konkretaus parinkimo)	Kiekvienas mazgas siunčia apdorotus rezultatus pagrindiniam mazgui (aiškus išrinkimas)	Įterpimas MPI_Bcast arba MPI_Send ir MPI_Recv

2 lentelėje pateiktas programos perėjimas nuo padalintos prie išskirstytos architektūros modelio. Pastebėsime, kad įvertinus padalintos architektūros modelį, daugiau pranašumų turi išskirstytas modelis. Išskirstytos sistemos modelis yra saugesnis ir patikimesnis. Negalimas visų mazgų priėjimas prie visos adreso srities, o galimas – tik prie rezervuotos srities.

Bet kuri paskirstytos sistemos fizinė architektūra turi laikytis tam tikrų nustatytų projektavimo ir išskirstytos programinės įrangos reikalavimų [124]. Būtina išskirti tokias jos dedamąsias:

- modelis,
- architektūra,
- projektavimas.

Programinė įranga, kaip išskirstyta sistema – kuris sudaryta iš tarpusavyje „susikalbančių“ užduočių [25]. Užduotys išskirstytos skirtingiems sistemos procesoriams. Skiriamos dvi uždavinių išskirstymo technikos:

- statinė [26],
- dinaminė [27].

Statinės išskirstymo schemos priešingai nei dinaminės technikos, atsižvelgia į darbo krūvio charakteristikas. Atsižvelgiama į sistemą su sudėtinėmis užduotimis ir jų vykdymo laiką. Išskirstymas yra atliekamas, siekiant optimizuoti vieną ar daugiau tikslų:

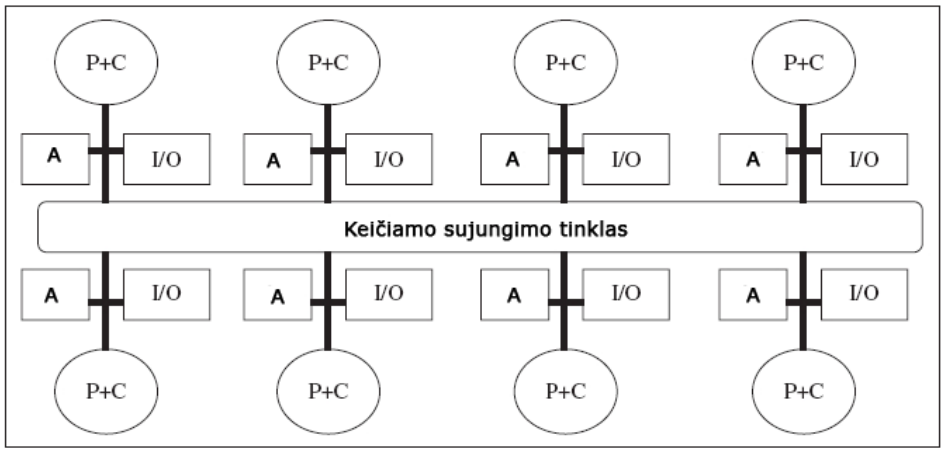
- pagerinti procesorių susisiekimą (bendravimą);
- išlaikyti pusiausvyrą procesoriuose;
- atminties panaudojimą.

Išskirstymo schemos gali būti klasifikuojamos į dvi kategorijas. Egzistuoja tam tikri tikslūs metodai, kurie leidžia, orientuojantis į konkretų tikslą, rasti optimalų išskirstymą. Egzistuoja grafų-teorinės technikos ir sveikųjų skaičių programavimas. Tačiau kaip problema išlieka *NP-complete* , be to šie metodai yra pakankamai brangūs. Yra dar vienas, pigesnis būdas išspręsti šią problemą t.y. sukonstruoti euristinį metodą [23].

Tipinė išskirstytos atminties multiprocesorinė sistema yra pavaizduota 3 pav. Ši architektūra leidžia keisti išskirstytą atmintį (skirstyti) visoje mašinoje, naudojant keičiamą susijungimą, tokiu būdu suteikiant procesoriams galimybę susijungti su atminties moduliais. Pagrindiniai tokių sujungimų mechanizmai gali būti:

- multikompiuterinė / pranešimo perdavimo architektūra;
- išskirstytos padalintos atminties modelis (Išskirstytos atminties modelis) (DSM).

Multikompiuteriai naudoja programinę įrangą (pranešimo perdavimo sluoksnyje), komunikavimui tarpusavyje ir nuo to laiko tai vadinama pranešimo perdavimo architektūra. Į šią sistemą paprastai jungiami daugialypiai skaičiavimo mazgai, naudojantys tik keičiamą susijungimą, tai – minėtieji multikompiuteriai. DSM mašinos (iš loginės pusės) yra vienos globalios atminties adreso vieta, nors atmintis fiziškai yra išskirstyta. Priėjimo prie atminties laikas priklauso nuo procesoriaus fizinės padėties ir padėtis nebekeičiama. Ši sistema yra vadinama nevienarūše atminties priėjimo (*NUMA – nonuniform memory access*) sistema.

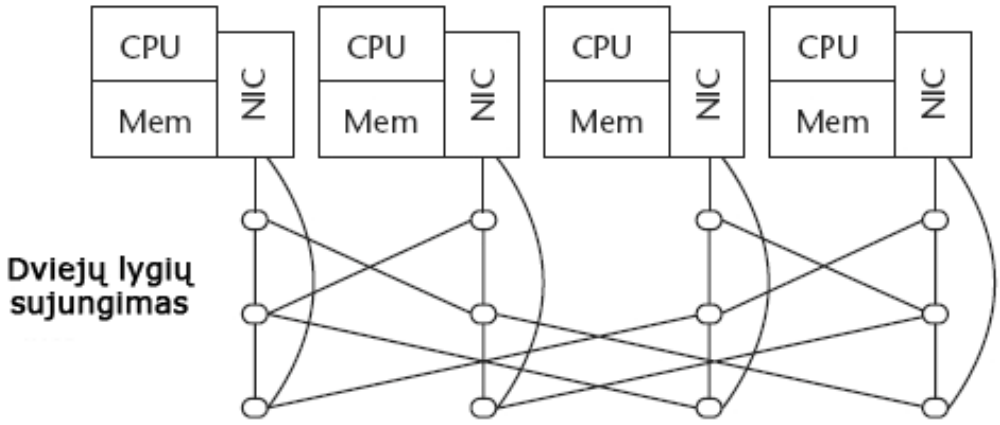


3 pav. ***Išskirstytos atminties multiprocesoriai (P+C-proc. ir spart. atmintis; A-atmintis)*** **[10]**

Pranešimo-perdavimo ir DSM sistemos turi tą patį organizavimo pagrindą. Pagrindinis skirtumas yra tas, kad DSM įgyvendina vieną padalinto adreso vietą, o pranešimo-perdavimo architektūra turi išskirstytą adreso vietą.

Numatant išskirstytos atminties sistemos funkcionalumą fizinėje išskirstytoje atmintyje (žr. 4 pav.), reikalinga įgyvendinti tris pagrindinius mechanizmus:

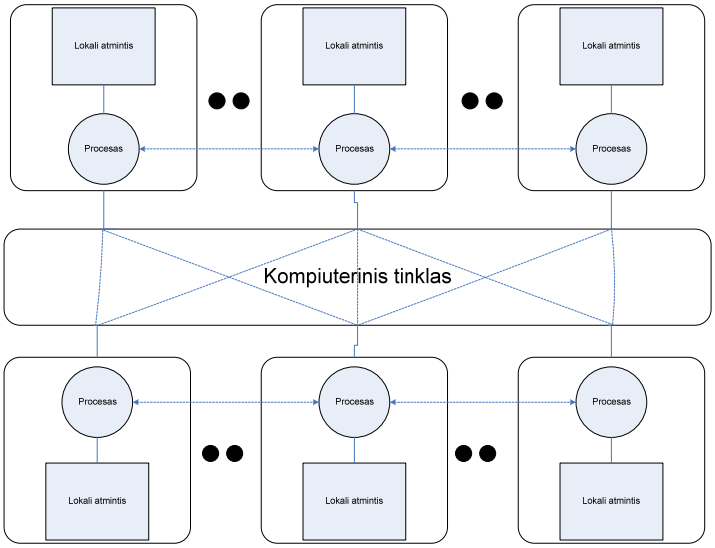
- Procesoriaus užklausimo/atsako tikrinimas (*Processor-side hit/miss check*) – operacija, iš procesoriaus pusės, yra naudojama, nustatyti ar yra ar ne reikalingų duomenų užklausimas, išskaidytas procesoriaus spartinančioje atmintyje. Užklausimas yra duomenų atitikimo spartinančioje atmintyje tikrinimas, o atsakymas reikalauja, kad duomenys būtų gauti iš pagrindinės atminties arba kito procesoriaus spartinančiosios atminties.
- Procesoriaus pusės užklausos siuntimas (*Processor-side request send*). Ši operacija naudojama procesoriaus dalyje, kai atsakymas prarastas, siunčiama užklausa kitam procesoriui arba pagrindinei atminčiai, ieškant paskutinių duomenų kopijos įrašo ir laukiama galutinio atsakymo.
- Atminties pusės operacijos. Šios operacijos leidžia atminčiai gauti užklausą iš procesoriaus, įvykdyti bet kuriuos būtinus sąsajų veiksmus ir siųsti atsakymą, paprastai duomenų užklausos forma.



4 pav. ***Išskirstytos atminties modelis pagal tinklo sąsajas*** **[10]**

Išskirstytos atminties sistemos atveju galima naudoti pranešimo siuntimo modelį (žr. 5 pav.), kuris padeda realizuoti lygiagretumo principą.

Dar 1990 metų viduryje, pranešimų siuntimas tapo dominuojančiu didelių lygiagrečių procesorių sistemų programavimo mechanizmu [28]. 1990 metų pabaigoje daugelis pranešimo siuntimo programų padarė įtaką MPI standarto atsiradimui [29]. Pranešimo siuntimo sąsaja yra ryšio (komunikacijos) biblioteka, skirta lygiagretiems skaičiavimams. Ji palaiko Single-Instruction-Multiple-Data (SIMD) lygiagrečių skaičiavimų modelį [30]. MPI standartas [31] užtikrina ryšio bazinių elementų įvairovę [30].



5 pav. ***Pranešimo siuntimo modelis (Message-Passing Model)*** **[17]**

MPI blokinio siuntimo ir gavimo baziniai elementai yra analogiški naudojamiems tinklų modelyje (*proccess networks*). Pavyzdžiui, tarkime du procesai gali siųsti tam pačiam gavimo procesą arba du procesai gali gauti iš to paties siuntimo proceso ir tai duoda į nedeterminizmą. Vieno gavėjo ir vieno siuntėjo suvaržymai eliminuoja nedeterminizmą, kai lygiagrečių skaičiavimų rezultatas nepriklauso nuo siuntimo ir gavimo tvarkos. Iš konceptualiosios pusės žiūrint, MPI realizuotas dviejų pranešimų eile – viena, kuri saugo neatliktas gavimo užklausas (tai vadinama gavimo eile) ir kita, kuri saugo ateinančių pranešimų sąrašą, bet nei vieno prieš tai buvusio užklausimo (nenumatytų, nenurodytų prašymų eilė) [28]. Ateinantys (įeinantys) pranešimai keliauja į gavimo eilę su galima pora, kur yra atitikimas ir pabaigoje į nenumatytų eilę, jei nepavyksta rasti poros (neegzistuoja gavėjas). Prieš tai, prašymas gali būti pridėtas į skelbiamų gavėjų eilę, nenumatytoji eilė privalo ieškoti galimos poros. MPI pranešimai suderinamumui naudoja tris parametrus:

- situacijos identifikatorių,
- šaltinio (gavėjo) padėtį,
- pranešimo žymę [28].

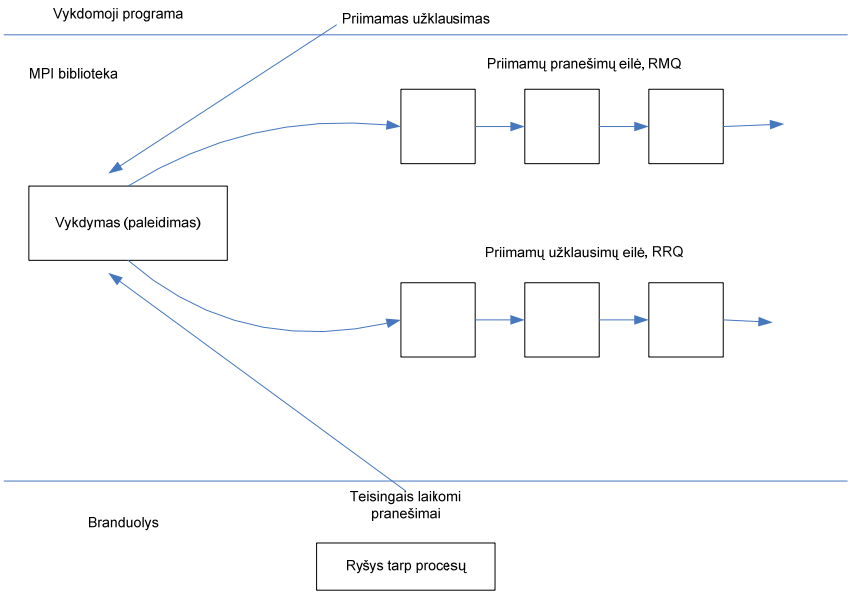
Situacijos identifikatorius reiškia MPI ryšio objektą, šaltinio padėtis reiškia siunčiamo proceso vidinę padėtį ryšio viduje (komunikatoriuje), pranešimo žymė – vartotojo priskirta reikšmė pranešimo išrinkimui.

Tipinis MPI įgyvendinimas, toks kaip *MPICH* [24] arba *LAM/MPI* [33], naudoja dvi eiles, kad būtų realizuota gavimo operacija. 6 paveiksle vaizduojama eilė: gaunamo pranešimo eilė ir gaunamo užklausimo eilė. Gaunamo pranešimo eile (RMQ) (kartais dar vadinama netikėtumo eile), laikomi gauti pranešimai, kai jie pereina į laukimo būseną, kur *MPI-Recv/MPI-Irecv* yra dar nepanaudotas. Visa informacija apie RMQ saugoma žymėse, šaltinyje ir pranešimo turinyje.

MPI biblioteka sukuria naujus įrašus ir įterpia juos į RMQ, kai pranešimas gautas. Gaunamo užklausimo eilė (RRQ) laikoma užklausimais, kurie yra pasiruošęs gauti pranešimą. Užklausimai yra laukimo būsenoje, kur atitinkamai *MPI-Recvi/MPI-Irecv* yra paskelbti, bet dar negauti atitinkami pranešimai. RRQ įraše pateikiama informacija apie žymę, šaltinį ir gaunamo buferio adresą gavimui. MPI biblioteka kuria naują įrašą ir įterpia į RRQ, kada *MPI-Recv/MPILIrecv* paprogramė yra iškviesta.

MPI pranešimų biblioteka [42] pateikia daug ryšio įvairovių ir palaiko daug programavimo stilių, taip pat akcentuotini skirtumai tarp MPI naudojimo stilių, tokių kaip sinchroninis arba asinchroninis programavimo stilius ir standartinės blokinio ir neblokinio ryšio naudojimas [43].

SIMD paradigma išskirstytam veikimui, naudojant MPI nustatas atlieka tris pagrindines ryšio operacijas – siuntimas, gavimas ir nustatymas (*Send, Receive, and Setup*). Žiūrint iš praktinės pusės, kiekviena MPI ryšio komanda turi pastovų inicializacijos laiką ir skirtingą (nepastovų) (priklauso nuo pranešimo dydžio) vykdymo laiką. Suprantama, kad susijusių *Send* ir *Receive* operacijų vykdymas gali būti lygiagretus (konkurencinis), su daliniu uždelsimu, o limituotas duomenų ryšio buferio dydis gali į blogąją pusę nulemti visą perdavimo greitį (spartą) [36].



6 pav. ***Tipinis MPI įgyvendinimas, naudojant dvi eiles*** **[37]**

Galima nustatyti vyraujančias tris pozicijas, kurios lemia pranešimo siuntimo sąsajos veikimą. 3 lentelė iliustruoja apžvelgtas pozicijas.

3 lentelė. ***Pozicijų-nuostatų MPI veikimui apžvalga***

	1 pozicija	**2 pozicija**
Trumpas aprašymas	SIMD paradigmos atveju išskirstymas veikimui naudojamos trys operacijos: siuntimas, gavimas ir nustatymas (*Send, Receive, and Setup*).	MPI realizuotas kaip dviejų pranešimų eilė: gavimo eilė ir nenumatytų eilė.
Išvados	Paviršutiniškas pozija į MPI, tiesiog skaldant visą veikimą į atskiras dalis – komandas.	Pozicija pakankamai detalizuota, bet sudėtinga. Pats veikimo principas kitoks.

1.6. *Vienas su vienu* ir bendras ryšio iškvietimas

MPI veikimo atveju vienas su vienu ryšio iškvietimas yra susijęs su siuntimais ir gavimais tarp dviejų procesorių. Yra dvi pradinės siuntimo ir gavimo kategorijos, kurios taip pat gali būti blokinės arba neblokinės. Blokinis iškvietimas yra vienas grąžinimas, kai siuntimas (arba gavimas) yra užbaigtas. Neblokinis iškvietimas grąžinamas nedelsiant ir tai priklauso nuo programuotojo, tikrinančio iškvietimo užbaigtumą.

Bendri ryšio (komunikacijos) iškvietimai leidžia programuotojui perduoti komandas procesorių pogrupiui virtualioje mašinoje. Bendro ryšio iškvietimo atveju yra lengviau įvykdyti užduotis tokias, kaip proceso sinchronizacija, globalus sumavimas, duomenų išskaidymas ir surinkimas [38].

Išskiriamos dvi svarbiausios programavimo paradigmos: pranešimo siuntimo (*Message Passing – MP*) modelis ir virtualios padalintos atminties modelis (*Virtual Shared Memory – VSM*). Pranešimo siuntimo modelis (*Message Passing Model*) buvo vienas pirmųjų vystomų ryšio ir duomenų perdavimo tarp procesų ir/arba procesorių išskirstytoje skaičiavimo sistemoje. MPI pagrindu laikoma nustatyta biblioteka, su aiškiai apibrėžtu pranešimo siuntimu ir skirta aukšto našumo (ypač sudėtingiems) skaičiavimams. Vis dėlto MPI kodų vystymas reikalauja nemažai pastangų ir finansinių išteklių [39].

Atlikus lyginamąją analizę nustatyta, kad MPM (*Message Passing Model*) yra pranašesnis už VSM (*Virtual Shared Memory*). 4 lentelėje parodyta, kad daugeliu atveju VSM turi ryškių pranašumų.

4 lentelė. ***Modelių privalumai ir trūkumai***

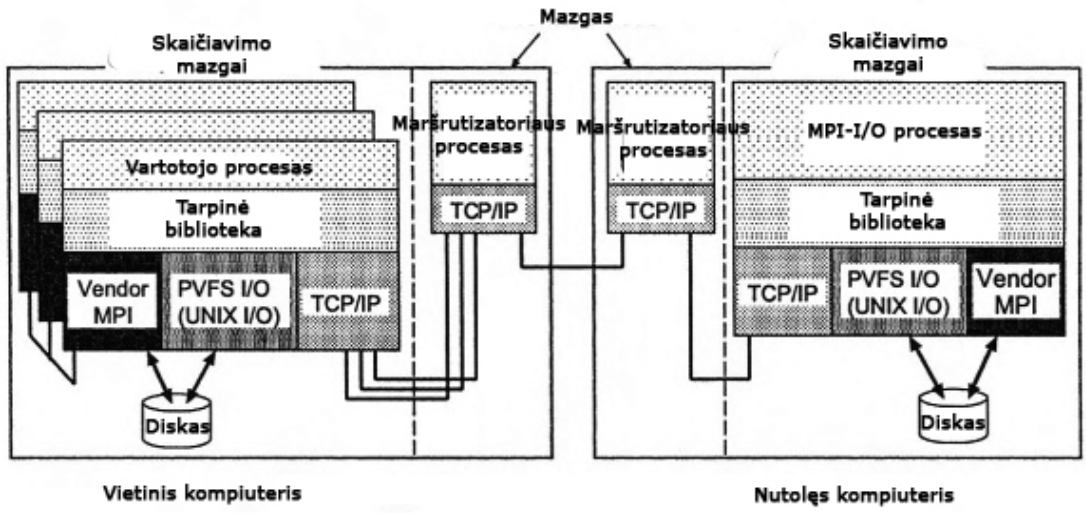
	MPM – *Message Passing Model*	VPM – *Virtual Shared Memory*
Privalumai	1. Išeities kodo greitas įvykdomumas.	1. Nesudėtingas įgyvendinimas ir pašalinimas reikiamos aukšto

	2. Fiksuotas procesų skaičius paleidimo metu.	lygio abstrakcijos; 2. Išeities kodo sumažinimas; 3. Labiau lanksti ir agreguotas kodo struktūra; 4. Atskirti procesai ir duomenys; 5. Aukštas kodo portatyvumas;
Trūkumai	1. Dideli finansiniai ištekliai. 2. Išeities kodo sudėtingumas ir apimtis.	1. Nesusijęs su klasikinėmis aukšto našumo skaičiavimų aplikacijomis. 2. Didelė kaina.

Matome, kad VPM turi daugiau privalumų, tačiau MPM išlieka greitesnis, nors kodas sudėtingesnis ir labiau tinkamas išskirstytuose sudėtinguose skaičiavimuose. Atlikti sudėtingi testai ir vertinimai [39] su šiais modeliais, parodė, kad MPM vystymas yra perspektyvus, bet brangus projektas.

1.7. MPI įvedimo/išvedimo sąsaja

Pastaruoju metu intensyvus sistemingų duomenų naudojimas pareikalavo lygiagrečios įvedimo/išvedimo sistemos. Lygiagreti MPI-įvedimo/išvedimo sąsaja buvo įtraukta į MPI-2 standartą [40]. Nors tai buvo įdiegta į keletą MPI bibliotekų, tačiau MPI-įvedimo/išvedimo operacijos su nutolusiais kompiuteriais, turi atskirą MPI biblioteką (remote MPI-I/O), kuri nebuvo palaikoma. Vartotojai gali vykdyti su nutolusiais objektais MPI-įvedimo/išvedimo operacijas naudodami MPI biblioteką [41]. MPI-įvedimo/išvedimo architektūra vaizduojama 7 pav.



7 pav. ***MPI-įvedimo/išvedimo mechanizmas*** **[41]**

Sąsajos sluoksnyje vartotojo procesai yra tarpinės sąsajos, kurios turi MPI API, kur egzistuoja pranešimų perdavimas tarp vartotojo procesų ir esančių ryšio ir įvedimo/išvedimo sistemų. Vartotojai gali vykdyti MPI ryšio funkcijas, įskaitant MPI-įvedimo/išvedimo funkcijas tarp kompiuterių be sudėtingų skirtumų, esančių ryšio ir įvedimo/išvedimo sistemose. Kad būtų galima realizuoti išskirstytą įvedimą/išvedimą PVFS bylų sistemoje su dideliu našumu, PVFS įvedimo/išvedimo funkcijos turi būti integruotos MPI-įvedimo/išvedimo mechanizme [41].

1.8. MPI ir PVM palyginimas

PVM (Lygiagreti Virtuali Mašina) programinė įranga aprūpina unifikuotą karkasą, kurio viduje lygiagrečios programos gali būti efektyviai vykdomos ir nesudėtingais metodais naudojama esanti įranga. PVM leidžia heterogeninei kompiuterių sistemai būti suprantamai kaip virtuali mašina. Tačiau tokį metodą naudojanti sistema turi ir daug trūkumų.

Lyginant su PVM, išskiriami tokie MPI privalumai (tuo grindžiamas MPI naudojimas):

- MPI turi daugiau negu vieną laisvai prieinamą realizaciją;
- MPI apibrėžia trijų dalių profiliuotą mechanizmą;
- MPI turi pilnai asinchroninį ryšį;
- MPI grupės yra vientisos, efektyvioss ir baigtinės;
- MPI efektyviai valdo pranešimų buferius;
- MPI efektyviai veikia su MPP ir blokiniais;
- MPI yra formaliai specifikuotas;
- MPI yra standartas.

Paminėtieji privalumai ir nulėmė šio metodo pasirinkimą tolesniuose darbuose.

1.9. Blokinio formavimas

Tinklų vystymasis, aukštos klasės kompiuteriai ir išaugusios galimybės peraugo į naują skaičiavimo infrastruktūrą, pavadinta kompiuterių tinklas (*Networks Of Computers – NOW*) arba personalinių kompiuterių blokinys (*Cluster*). Tokių skaičiavimų galimybės patraukė ir skaičiavimo industrijos dėmesį, kadangi ši technologija priklauso nuo konkrečių komponenčių. Be to, jie plačiai naudojami, siekiant pagerinti vykdymo našumą su intensyviais reikalavimais skaičiavimo galingumui [42].

NOW platformų populiarumas yra nulemtas jų galimybės augti, bei galėjimu aprūpinti pagal kaina efektyviam skaičiavimui, pasikliauti reikiama technologija ir efektyviai palaikyti procesorių interaktyvų veikimą ir didelę grupę lygiagretaus veikimo. Paprastai kompiuteriai yra

susieti didelio pralaidumo tinklu, tokiu, kaip Gigabit Ethernet, SCI arba Myrinet su operacinėmis sistemomis, kaip Microsoft Windows ir Linux [42].

Paprastai blokinės sistemos yra sudarytos iš Simetrinių multiprocesorinių mazgų (Symmetric Multi-Processor (SMP)), sujungtų didelio pralaidumo LAN (Local Area Networks) arba SAN (System Area Networks) [43].

Lygiagretumą realizuojančioje programoje yra dalis (grupė instrukcijų), kurios vykdomos nuosekliai. Dalis yra dažniausia vykdymo vienetas pagal lygiagretaus vykdymo sąlygą. Taip pat – tai planavimo vienetas, kai išskirstyta dalis apdirbama per elementus, siekiant pagerinti našumą. Būtina suformuluoti lygiagretaus vykdymo reikalavimus. Kada vykdoma lygiagreti programa, tuomet reikalingi papildomi elementai tokie, kaip kodo vykdymo laikas (lygiagretaus vykdymo), ryšio užlaikymas, sinchronizacijos užlaikymas, privalomos atsargos nutolusiame mazge ir t.t. Tarkime, kad apkrova bus O. Jeigu mes vykdome programą vieno mazgo sistemoje – tuomet nėra apkrovos. Grynąjį tinklo vykdymo laiką pavadinkime R. Supaprastinimui, būtina prisiminti padalintą vykdymą ir stebėjimą per 2 procesorius, parodytus 8 paveiksle. Tarkime, kad dalies 0 vykdymo laikas yra R0. Tai bus pabaigta anksčiau nei 1 ir 2 dalies vykdymas. Imami atitinkamai R1 ir R2 laikai. Po kiekvieno vykdymo, valdymas įliejamas į 3 dalį, kuri turi R3 vykdymo laiką [44].

Kada vykdoma ši programa 1 mazgo mašinoje, bendras vykdymo laikas (žr. 3 formulę)

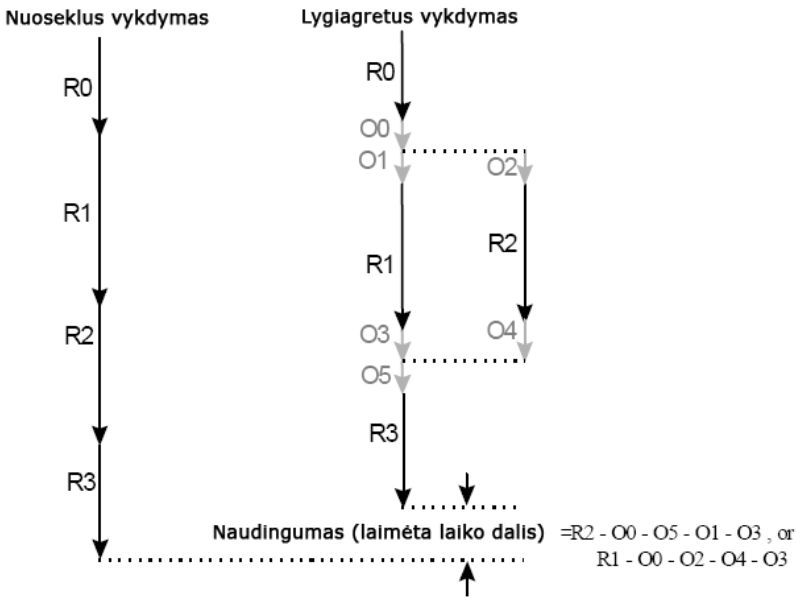
$$R = R0+R1+R2+R3 \quad (3)$$

Kalbant apie lygiagrečią sistemą, taigi neišvengiamai reikalingos sąnaudos lygiagrečiam vykdymui. Taip pat reikalingos išskirstytam kodui ir duomenims bei perdavimui, kad būtų galima realizuoti nutolusį vykdymą. Šie sudeda R0, kadangi 0 inicializuoja 1 ir 2 dalis. Tarkime, kad tos sąnaudos būtų O0 – sukūrimo (iškvietimo) sąnaudos. Po R0 ir O0, 1 ir 2 dalys yra vykdomos. Tai tokia situacija, kai pereinama nuo 0 dalies prie 1 pirmajame procesoriuje ir 2 dalies, galbūt prarandant atminties po užlaikymo. Ir tada 1 ir 2 dalys gali užklausti duomenų. Tarkime, kad tai O1 ir O2 atitinkamai. Tai vadinama paruoštomis sąnaudomis.

O3 ir O4 – tai sąnaudos skirtos siuntimui/saugojimui vykdymo rezultatų 3 daliai ir pranešimo siuntimui apie užbaigtumą. Tai vadiname išeinančiomis sąnaudomis. Gali būti resursų kaupimo laikas arba kitų procesorių vykdymo užbaigimas. Tai vadinama sugaištu laiku suirimo vykdymui, nes gali būti naudojama, bet šiuo atveju nėra naudojama. Po išeinančiųjų sąnaudų O3 ir O4, būna užlaikymas, kad būtų galima paruošti rezultatus ir inicializuotas R3. O5 ir nurodo šias sąnaudas. Dabar galima susumuoti lygiagretaus vykdymo laiką, kuris lygus (žr. 4 formulę) [44]

$$R0 + O0 + \max(O1+R1+O3, O2+R2+O4) + O5 + R3 \quad (4)$$

8 paveiksle pavaizduota dviejų procesorių sistema išskirstytai, lygiagretaus vykdymo, sistemai.



8 pav. ***Lygiagretaus vykdymo pranašumas prieš nuoseklų vykdymą*** **[44]**

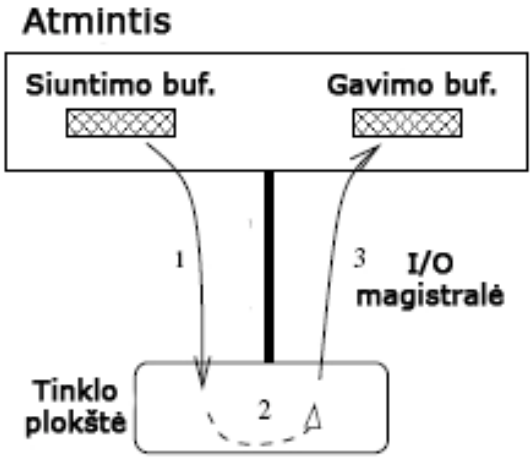
Kaip matome iš 8 pav., nuoseklaus vykdymo vieno procesoriaus sistema užduotis atlieka dalimi lėčiau nei lygiagretaus vykdymo dviejų procesorių sistema. Galime padaryti išvadą, kad didinant išskirstytos sistemos mazgą – reikalingas lygiagretus vykdymas.

1.10. Egzistuojantys mazgo veikimo mechanizmai

Išskiriami trys pagrindiniai mazgo veikimo mechanizmai:

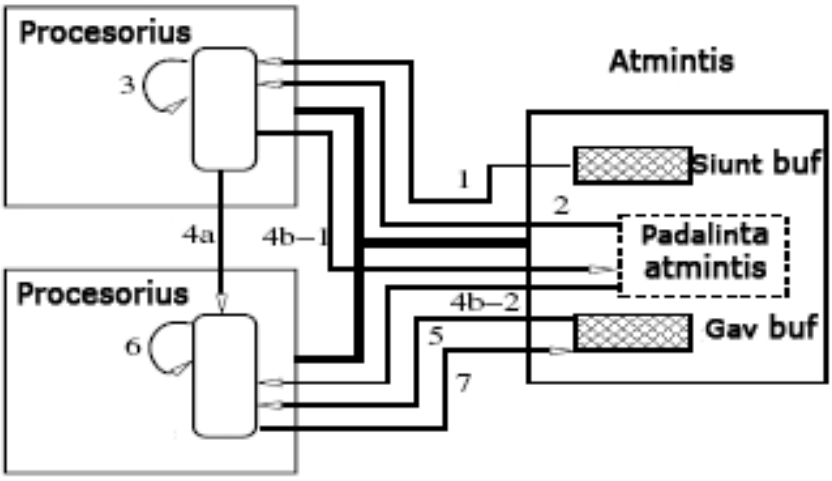
- tinklo plokštės duomenų pralaidumo lygis (*NIC-level Loopback*);
- vartotojo vietos padalinta atmintis (*User-space Shared Memory*);
- branduoliu grindžiamas sprendimas (*Kernel-Based Solution*) [43].

Gera tinklo plokštė gali palaikyti NIC lygio (*NIC-level*) pralaidumą. Kada pranešimo perdavimas yra pradėtas, NIC gali nustatyti, ar vieta yra tame pačiame mazge ar ne. Imant vietinį priėjimą prie atminties (*DMA – Direct Memory Access*) nuo NIC atminties atgal į kompiuterio atmintį (žr. 9 pav.), galime eliminuoti tinklo jungties sąnaudas, nes pranešimas neišleidžiamas į tinklą. Vis dėlto čia egzistuoja dvi operacijos. Net jei Įvedimo/Išvedimo magistralė yra greita, DMA sąnaudos gali būti dar didesnės. Toliau DMA operacijos negali panaudoti spartinančios atminties efekto [43].



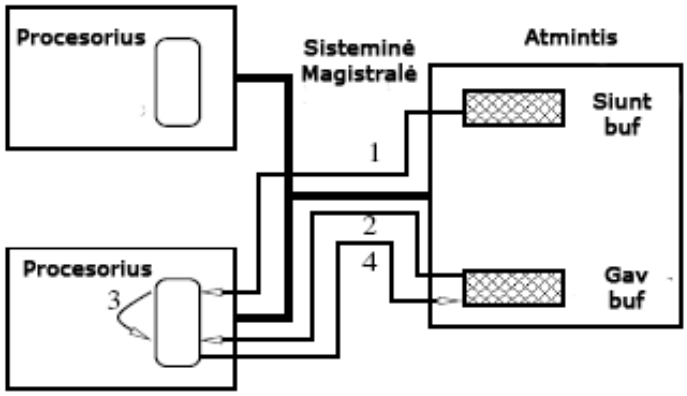
9 pav. ***Tinklo plokštės lygio pralaidumas (NIC-level loopback)*** **[35]**

Ši projektavimo alternatyva apima kiekvieną MPI procesą vietiniame mazge, priskiriant sau pačiam padalintos atminties dalį. Ši padalintos atminties dalis gali tada būti naudojama tarp vietinių procesų apsikeičiant pranešimais. Siuntimo procesas gali kopijuoti pranešimą į padalintos atminties sritį. Gavimo procesai gali kopijuoti pranešimą į savo buferį. Šis būdas reikalauja mažiausiai resursų kiekvieno pranešimo apsikeitimui (žr. 10 pav.).



10 pav. ***Vartotojo vietos padalinta atmintis (User-space Shared Memory)*** **[43]**

Branduoliu grindžiamas atminties priskyrimo būdas – operacinės sistemos branduolio pagalba kopijuojant papildomai iš vieno vartotojo proceso į kitą be papildomų kopijavimo operacijų. Siuntėjo arba gavėjo procesas siunčia užklausėjo deskriptoriaus pranešimą į pranešimų eilę, nurodant virtualų adresą, žymę ir kt. Ši atmintis priskiriama branduolio adresų vietai, kada kitas procesas ateina apsikeitimo principo pranešimu. Tuomet branduolys įvykdo tiesioginį kopijavimą iš siuntėjo buferio į gavėjo nurodytą buferį. Šis būdas realizuoja tik vieną kopiją (žr. 11 pav.).



11 pav. ***Branduoliu grindžiamas atminties priskyrimas (Kernel-based memory)*** **[43]**

Lyginamoji trijų mechanizmų analizė pateikta 5 lentelėje. Kaip matome iš šios lentelės, išryškėja kai kurie šių mechanizmų privalumai ir trūkumai. Toks jų sulyginimas būtinas, siekiant išaiškinti mechanizmo naudojimą tolesniuose darbuose. Atliekant eksperimentą bus taikomas „Tinklo plokštės lygio“ duomenų pralaidumo mechanizmas, kuris ir leis optimizuoti resursų panaudojimą, siekiant didelio našumo, sprendžiant sudėtingus uždavinius.

5 lentelė. ***Mazgo veikimo mechanizmų palyginimas***

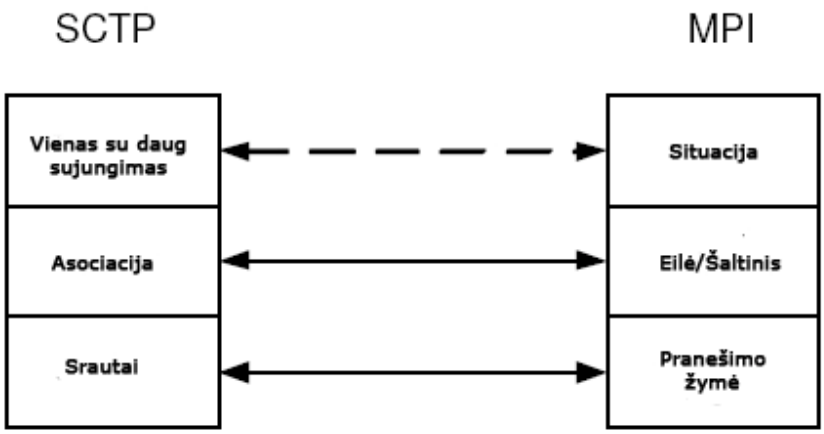
Vertinimo aspektai	**Tinklo plokštės lygio duomenų pralaidumui (*NIC-level Loopback*)**	**Vartotojo vietos padalinta atmintis (*User-space Shared Memory*)**	**Branduoliu grindžiamas atminties priskyrimas (*Kernel-based memory*)**
Privalumai	1. Eliminuojamos tinklo jungties sąnaudos. 2. I/O magistralės greitumas.	1. Reikalauja mažiausiai resursų pranešimo siuntimui. 2. Naudojama padalinta atmintis, taip išskiriant tik konkrečią sritį	1. Nereikalingos papildomos kopijavimo operacijos.
Trūkumai	1. DMA negali panaudoti spartinančios atminties.		1. Realizuoja tik vieną kopiją.

1.11. MPI ir SCTP naudojimo galimybių analizė

TCP ir UDP – tai du pagrindiniai IP protokolai plačiajuosčiam naudojimui IP tinkluose. Neseniai atsirado naujas transportinis protokolas – SCTP (Stream Control Transmission Protocol) ir kuris yra standartizuotas [45]. SCTP yra grindžiamas pranešimais kaip UDP, bet turi kaip TCP ryšio valdymą, perkrovų ir srautų valdymo mechanizmą [46].

SCTP yra tinkamas MPI, nes grindžiamas pranešimais ir srautų taisyklėmis. SCTP ir MPI panašumai pavaizduoti 12 paveiksle. MPI programa identifikuoja procesų aibę, kurioje

komunikuoja procesai vienas su kitu. Šitoks procesų grupavimas gali atstoti vienas su daug ryšį SCTP, kuris nustato asociacijas su procesais, esančiais aibėje. Kiekviena asociacija gali turėti daug srautų, kurie nepriklausomai surūšiuoti, ir ši savybė tiesiogiai atitinka pranešimo siuntimo semantiką MPI. MPI pranešimai siunčiami su tam skirtomis žymėmis/eile/situacija [46].



12 pav. ***SCTP ir MPI protokolų sugretinimas*** **[46]**

Šis panašumas tarp SCTP ir MPI yra matomas konceptualiajame modelyje, tačiau yra diskutuotinų vietų. Pirmiausia reikia išskirti perėjimą nuo asociacijos prie kategorijos ir nuo srauto prie pranešimo žymės. Situacijos sukūrimas MPI programos viduje gali būti dinaminė operacija ir jeigu mes priskiriame sujungimo mechanizmą prie situacijos, tai reikalauja sukurti dinaminį numerį, veikiant sujungimo mechanizmo programai [46].

Skirtingi tinklai reikalauja skirtingų kompiuterių skaičiavimų lygių, naudojant MPI pranešimus. Sakykime, vienas tinklas gali reikalauti kompiuterio nustatymo ir tinklo DMA aktyvumo sekimo, tol kol kitas tinklas galės pilnai nutraukti ryšį su tinklu ir vengti atsitiktinio bet kurio procesoriaus įsitraukimo į duomenų perdavimą. Nuo šių charakteristikų tipo priklauso tinklo pralaidumas, todėl galime paimti MPI procesoriaus bendresnę charakteristiką, analizuojant duomenų eilę [47].

Yra keletas MPI atminties naudojimo būdų. Tinklai paprastai turi baigtinį skaičių siunčiamų ir gaunamų užklausų, kurios gali būti lokalizuotos. Vienas iš atvejų, kai naudojamos kreditais grindžiamos schemos efektyviam tinklo užklausų perdavimo valdymui. Įgyvendinant reikia skirti atminties nenumatytų pranešimų talpinimui. Ši atmintis tikriausiai yra didžiausia problema bet kurio MPI įrankio. Daugelio įgyvendinimų atveju siunčiami trumpi pranešimai, siekiant optimizuoti gaišties laiką, tokia situacija kaip *N-1* sujungimas gali greitai užimti visą nenumatytų pranešimų buferio vietą [47].

1.12. Ryšių topologijos nustatymas ir įvertinimas

Pranešimų siuntimas išskirstytoje sistemoje yra vienintelė procesorių sujungimo priemonė. Artimiausiai veikimo elementai netiesiogiai komunikuoja vienas su kitu. Tokiu būdu, pranešimas gali keliauti per vieną ir daugiau mazgų tol, kol pasieks tikslą.

Ryšio laikas (gaišties) išskirstytoje sistemoje priklauso nuo keleto faktorių [48], tokių, kaip:

- Topologija. Tinklo topologija – tipinis modeliuojamas grafas, nusakantis kaip veikimo elementai yra sujungti.
- Maršruto nustatymas. Maršruto nustatymas parenka kelią, persiunčiant pranešimą į priskyrimo vietą.
- Srauto valdymas. Tinklas susideda iš kanalų ir buferių. Srauto valdymas nusprendžia resursų skirstymą, kaip pranešimas keliauja nurodytu keliu.
- Nukreipimas. Nukreipimas yra svarbus mechanizmas, kuris nusprendžia , kaip pranešimas keliauja nuo įėjimo kanalo į išėjimo kanalą.

Tinklo topologija, taip pat dar vadinamas vidinis (vidinio sujungimo) tinklas, gali būti klasifikuojamas pagal bendrą ir specialiąsias paskirtis. Norint pasiekti tikslą, būtina taikyti specialiosios paskirties tinklus. Specialiosios paskirties tinklai skirstomi į statinius ir dinaminius . Norint apžvelgti išskirstytą sistemą su N veikimo elementais, turime atlikti N faktorizaciją (išdalinimą į atskirus elementus) (žr. 5 formulę):

$$N = n_d \times n_{d-1} \times n_{d-2} ... \times n_1 \qquad (5)$$

Kiekvienas veikimo elementas yra iš mišrių skaičių sistemos imties su d-vienaženkliais skaičiais (0..9), tokiu būdu nurodant adresą: $(u_d, u_{d-1}, u_{d-2,} ... u_1)$. Į k-narį n-kubą su $k = n_i$, visiems i ir $n = d$, kiekvienas veikimo elementas yra sujungtas su kitu procesoriumi, kurio adresas skiriasi tiksliai vienaženkliu skaičiumi $\pm 1 \mod k$.

Detalizuosime galimas išskirstytų sistemos topologijas (žr. 6 lentelę), atlikdami vertinimus pagal šiuos kriterijus:

- valdomumas (valdymas) (***V***) – kompleksinės sistemos reikalauja: atnaujinimo, pataisymų ir registracijos;
- informacijos darnumas (***ID***) – išlaikomas neatmetimas, darnumas ir informacijos darnumo sekimas;
- klaidos išvengiamumas (***KI***) – klaidų išvengiamumas yra būtinybė didelėje išskirstytoje sistemoje;

- sistemos augimas (didumas) (***SA***) – augant sistemos apimčiai, auga ir jos apdorojimo sudėtingumas.

6 lentelė. ***Išskirstytų sistemų topologijų detalizacija***

Vertinimo kriterijus	**Centralizuota sistema**	**Žiedo sistema**	**Hierarchinė sistema**	**Decentralizuota sistema**	**Centralizuota + žiedas**	**Centralizuota + decentralizuota**
V	+	+	±	–	+	-
ID	+	+	±	–	+	?
KI	–	+	±	+	+	+
SA	?	+	+	?	+	+

„+“ – Taip; „-“ – Ne; „?“ – Neaišku, galbūt; „±“ – Dalinai.

Sekanti lentelė detalizuoja galimas išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos tinklo realizavimo galimybes (žr. 7 lentelę) [49]. Realizacija suprantama kaip grafas.

7 *lentelė.* ***Išskirstytų sistemų topologijų detalizacija***

Tinklo topologija	**Mazgų skaičius**	**Tinklo skersmuo**	**Bisection Width**
1D Mesh (linijinis masyvas)	k	k–1	1
1D Torus (žiedas, ciklas)	k	k/2	2
2D Mesh	k^2	2k–2	k
2D Torus (k-ary 2 cube)	k^2	k	2k
3D Mesh	k^3	3k-3	k^2
3D Torus (k-ary 3 cube)	k^3	3k/2	$2k^2$
Pyramidė	$(4k^2-1)/3$	$2\log_2 k$	2k
Dvejetainis medis	2l–1	2l–2	1
4-naris medis (4-ary hypertree)	$2^l(2^{l+1}-1)$	2l	$\underline{2^{l+1}}$
Butterfly	$2^l(l+1)$	2l	$\underline{2^l}$
Hyperkubas	$\underline{2^l}$	l	$\underline{2^{l-1}}$
Cube-connected cycles	$\underline{2^l l}$	2l	$\underline{2^{l-1}}$
Shuffle-exchange	2^l	2l–1	$\underline{\geq \frac{2^{l-1}}{l}}$
De Bruijn	2^l	l	$\underline{\frac{2^l}{l}}$

Šios tinklo topologijos turi savo privalumų ir trūkumų.

1.13. Lygiagretaus vykdymo ANSYS veikimo principai

Kadangi išskirstytos sistemos yra orientuotos į tam tikrų uždavinių sprendimą ir kartu to proceso optimizavimą, būtina pastebėti, kad ANSYS programinė įranga, skirta mechatronikos uždavinių sprendimui, gali būti naudojama kaip lygiagreti sistema. Sistemos lygiagretumui realizuoti galimi tokie sprendimai:

- Paskirstytas Prie-conditioned Jungtinis Gradientas (*Distributed Prie-conditioned Conjugate Gradient* (*DPCG*));
- Paskirstytas Jacobi Jungtinis Gradientas (*Distributed Jacobi Conjugate Gradient* (*DJCG*));
- Paskirstytos Srities Sprendimas (*Distributed Domain Solver* (*DDS*));
- Algebrinis Multigrid (*Algebraic Multigrid* (*AMG*)).

Kiekvieną iš šių sprendimų detalizuosime:

1. Paskirstytas Prie-conditioned Jungtinis Gradientas (DPCG) – yra pagrįstas nauja realizacija klasikinio PCG (Prie-conditioned Conjugate Gradient) sprendimo su išskirstytu lygiagrečiu apdorojimu. DPCG sprendimas išsaugo visas pagrindines savybes, naudojamas PCG sprendime ir gali būti paleista sistemoje su padalinta arba išskirstyta atmintimi. Lyginant su DDS sprendimu, DPCG sprendimas yra geresnis, nes naudojama mažiau atminties su mažesniu procesorių skaičiumi (mažiau kaip 16 procesorių).
2. Paskirstytas Jacobi Jungtinis Gradientas (DJCG) – yra pagrįstas nauja realizacija klasikinio JCG sprendimo su paskirstytu lygiagrečiu apdorojimu lauko tipo tokių kaip šilumos analizė. DJCG palaiko abi tiek padalintos, tiek išskirstytos atminties architektūras. Vienas iš šio sprendimo privalumų – reikia mažiau atminties.
3. Algebrinis Multigrid (AMG) – naudojamas kaip sudėtingo daugialygio supaprastinimas sprendžiant nesąlygines matricas, skirtas pritaikymui paskirstytoms atmintims daugiaprocesorinėse sistemose. AMG sprendimu atveju reikalinga daugiau atminties negu PCG ir tai skirta linijinei ir nelinijinei analizei atlikti.
4. Paskirstytos Srities Sprendimas (DDS) – vystomas sprendimas, skirtas naudoti pagal išskirstytos atminties sąlygą arba kaip derinys abiejų paskirstytos ir padalintos atminties sąlygomis. Tai skirta statinei arba nesudėtingai analizei su simetrinėmis matricomis. Pagal šį sprendimą modelis yra dalijamas į daugialypes sritis, t. y. grupuojama į baigtinius elementus, kurie ir sudaro pilną modelį.

Kiekviena individuali sritis yra paprastai nuo 1000 iki 10000 laisvės laipsnių dydžio.

1.14. ANSYS kaip išskirstytos, realizuojančios lygiagrečius skaičiavimus, sistemos taikymas mechatronikos uždavinių sprendimui

Specifinių mechatronikos uždavinių su n-laisvės laipsniais sprendimai reikalauja nemažai techninių ir programinių resursų. Atlikta programinių priemonių apžvalga rodo, kad galima taikyti ANSYS kaip lygiagrečią sistemą, kurią derėtų realizuoti pagal konkrečią atminties architektūrą ir techninę bei programinę platformą. Apžvelgti sprendimai realizuoja resursų paskirstymus tarp keleto sistemų. Daugiaprocesorinė sistema bus realizuota tokiais būdais [50]:

- serverių pagalba;
- kompiuterių, sujungtų į vieną potinklį;
- per 32 bitų Windows ir Linux klasterius.

Ieškoma optimaliausio sprendimo, siekiant rasti tinkamiausią atminties architektūros ar daugiaprocesorinės sistemos realizavimo pasirinkimą.

1.15. Mokslinės literatūros analizės apibendrinimas

Atlikta literatūros analizė parodo, kad išskirstytų sistemų projektavimas ir pritaikymas yra aktualus šiandieniniame moksliniame pasaulyje. Analizė atlikta vadovaujantis išskirstytų sistemų kūrimo karkasu. Papunkčiui apžvelgti pagrindiniai kūrimo aspektai. Literatūros analize siekta nustatyti ir išaiškinti galimus išskirstytų sistemų veikimo metodus, architektūras ir kitas glaudžiai susijusias detales: techninės ir programinės įrangos aspektus ir kt. Atlikta analizė bei naujausios literatūros gausa parodė, kad išskirstytos sistemos aktualios sudėtingų uždavinių sprendimui, kai akcentuojamas išsprendžiamumas, laikas ir resursai. Tokios sistemos yra operatyvesnės, tikslesnės (galimi tikslesni rezultatai). Daug kur vis dėlto akcentuojamas sudėtingas pritaikomumas ir išskirstytos sistemos vis dar pririšamos prie programinės įrangos palaikymo, nors sukurti tokie standartai, kaip MPI, leidžia lengviau spręsti problemas.

Temos aktualumas grindžiamas tokiais požymiais, kaip naujausios literatūros gausa ir globalių problemų analizė, siekiant optimizuoti resursus. Daugelis autorių vienareikšmiškai konstatuoja ir išskiria stipriąsias išskirstytų, realizuojančių lygiagrečius skaičiavimus, sistemų savybes. Tokių tyrimų, kai taikoma konkreti programinė realizacija yra dar nedaug, nes išskirstytos sistemos vis dar išlieka pakankamai brangios: reikalinga lygiagretaus vykdymo programinė įranga, techninė įranga bei tinklo įranga. Bet pagal efektyvumą lygiagretumas ir

išskirstytos sistemos yra reikalingos sprendimui uždavinių reikalaujančių didelio našumo. Atsiradę standartų atnaujinimai, patvirtina tokių sistemų reikalingumą.

2. TYRIMO METODIKA

Atliekant tyrimą bus reikalinga tokia programinė įranga:

- *ANSYS* programinė įranga – viso tyrimo pagrindas, nes šios sistemos pagrindu būtina bus realizuoti išskirstytą sistemą. Ši programa skirta mechatronikos uždavinio išeities kodo paruošimui.
- *MATLAB* programinė įranga – šios programinės įrangos pagalba bus realizuotas virtualus blokinys. Taip pat vykdomos kai kurios efektyvumo ir spartos tikrinimo bei grafikų generavimo programos. Virtualaus blokinio vykdymas ir grindžiamas šios programinės įrangos interpretacijomis.

Šios programinės įrangos panaudojimas leidžia spręsti iškeltą problemą bei pagrįsti hipotezę. Šios dvi metodikos formuoja darbo pagrindą.

3. TEORINĖ DALIS

Siekiant pagrįsti iškeltos hipotezės problemą, buvo būtina atlikti metodų pritaikymą, naudojant literatūros apžvalgoje sukonkretintus teorinius metodus. Kadangi *ANSYS* programinės įrangos veikimo principai buvo apžvelgti dar literatūros analizės dalyje, būtina detalizuoti tik tuos aspektus, kurie iš tikrųjų gali būti panaudoti realizuojant išskirstytą, realizuojančią lygiagrečius skaičiavimus, sistemą. Išskirstytos sistemos realizacija integruoja savyje atminties architektūrą, tinklo topologiją, procesorių apkrovas, procesorių skaičių, sujungimus (vidiniai ryšiai), laisvės laipsnių sąryšį su išskirstyta atmintimi, tinklo topologiją ir kt. (dėmesys kreipiamas į „krumplinės pavaros uždavinio“) [45].

3.1. Sistemos našumo požymiai

Pirmiausia, analizuojant sistemą, buvo būtina išsiaiškinti pagrindinius metodus, pagal kuriuos apsprendžiamas sistemos našumas:

- **Vykdymo greitis** (*Execution rate*) – matuojamas kompiuterio išvedimais (*output*) per laiko vienetą. Priklausomai nuo to kaip apibrėžti kompiuterio išvedimai, galimi keli vykdymo greičio matavimai. Komandų skaičiaus per sekundę koncepcija, įvertinama MIPS (milijonas komandų per sekundę) ir ji dažniausiai naudojama. Tačiau toks matavimas yra skirtas uniprocesorinėms bei multiprocesorinėms sistemoms ir netinka SIMD kompiuteriams, kuriuose viena komanda operuoja dideliais skaičiais. Nėra skirtumo tarp teigiamų rezultatų ir apkrovų komandų. Gera priemonė aritmetinėms operacijoms yra *Megaflops* (*Mflops*).
- **Spartumas** (*Speedup*) – lygiagrečių skaičiavimų veiksnys, naudojant **p** procesorius ir nusakomas tokia formule (žr. 6 formulę):

$$S_p = \frac{T_1}{T_p}, \tag{6}$$

kur T_1 yra laikas, reikalingas atlikti skaičiavimus su vienu procesoriumi ir T_p – laikas, reikalingas atlikti tuos pačius skaičiavimu su p procesoriais. Vadinasi S_p yra nuoseklaus ir lygiagretaus vykdymo santykis. Paprastai spartumo faktorius yra mažesnis negu procesorių skaičius dėl laiko praradimo sinchronizacijai, sujungimo (ryšio) laikui ir kitų apkrovų, kurios atsiranda vykdant lygiagrečius skaičiavimus. Pastebėsime, kad egzistuoja toks ryšys:

$$1 \le S_p \le p \tag{7}$$

Pastebėsime, kad esama tokių atvejų, kai lygiagretaus vykdymo sparta susilygina su nuoseklaus vykdymo sparta.

- **Efektyvumas** (*Efficiency*) – lygiagrečių skaičiavimų efektyvumas yra apibrėžiamas, kaip spartumo faktoriaus ir procesorių skaičiaus santykis (žr. 8 formulę):

$$E_p = \frac{S_p}{P} = \frac{T_1}{pT_p} \tag{8}$$

  Efektyvumas yra gaunamų skaičiavimų kokybės matas.

- **Sąnaudos** (**R_p**) – lygiagrečių skaičiavimų sąnaudos yra santykis tarp visų įvykdytų operacijų, atliekant lygiagrečius skaičiavimus, su **p** procesoriais ir operacijų skaičiumi. **O_1**, reikalingų atlikti skaičiavimus su vienu procesoriumi (žr. 9 formulę):

$$R_p = \frac{O_p}{O_1} \tag{9}$$

  R_p yra susijęs su laiko praradimu dėl apkrovų ir visuomet yra didesnis už 1.

- **Utilizacija (U_p)** – utilizacijos faktorius yra santykis tarp atliktų operacijų skaičiaus **O_p** ir operacijų, kurios galėjo būti atliktos su **p** procesoriais per laiko **T_p** vienetus, skaičiaus (žr. 10 formulę):

$$U_p = \frac{O_p}{pT_p} \tag{10}$$

- **Kokybė** (Q_p) – lygiagrečių skaičiavimų kokybės faktorius, naudojant p procesorius, yra aprašomas formule (žr. 11 formulę):

$$Q_p = \frac{T_p^3}{pT_p^2 O_p} \tag{11}$$

Pateikti lygiagrečius skaičiavimus realizuojančios sistemos našumo požymiai lemia blokinio veikimo operatyvumą. Siekiant realizuoti *ANSYS* kaip lygiagrečią sistemą, būtina įvertinti juos. Siekiant tiksliai suprojektuoti sistemą, kuri bus naudojama eksperimentinėje dalyje, būtina atlikti analizę šiais visais aspektais (grafinis vertinimas atliekamas *MATLAB* pagalba – vertinimo išeities kodai pateikti darbo prieduose). Dabar būtina įvertinti procesorių skaičių, kuris būtų pats optimaliausias, siekiant „Krumplinės pavaros uždavinio“ pritaikymui lygiagretaus vykdymo sprendimui [48].

Siekiant išsiaiškinti optimaliausią procesorių skaičių išskirstytai, lygiagrečius skaičiavimus realizuojančiai, sistemai, kai skaičiavimai atliekami, žinant, kad būtina atsižvelgti į krumplinės pavaros uždavinio n-laisvės laipsnius, nes visas *ANSYS* (kaip lygiagrečius skaičiavimus

realizuojančios) sistemos veikimas priklauso nuo uždavinio sudėtingumo pagal n-laisvės laipsnius ir procesorių skaičiaus bei atminties dydžio išskirstytoje sistemoje. Taigi, pirmiausia reikia įvertinti lygiagrečią sistemą spartumo principu. Naudosime 16 ir 32 uniprocesorių sistemą ir kiekvienu iš šių atvejų nustatysime galimas sistemų spartas:

1. 16 uniprocesorių sistema – turi atlikti 100 užduočių;
2. 32 uniprocesorių sistema – turi atlikti 100 užduočių.

Visa uniprocesorinė sistema sudaryta iš vienodų procesorių. Siekiama išsiaiškinti tendenciją dėl užduočių skaičiaus, po to bus galima nagrinėti n-laisvės laipsnių sąryšį su procesorių skaičiumi sistemoje arba posistemėje. Tarkime, kad vieną užduotį procesorius atlieka per 1 sekundę, tai gaunama:

1. $S_p = \frac{T_1}{T_p} = \frac{100}{6{,}25} = 16s;$
2. $S_p = \frac{T_1}{T_p} = \frac{100}{3{,}12} = 32s.$

Matome, kad egzistuoja priklausomybė tarp procesorių skaičiaus ir sistemos spartumo.

3.2. Matematiniai modeliai

Šiame poskyryje nustatysime lygiagretaus skaičiavimo ribas ir atliksime grafinius vertinimus, pagal tam tikrus metodus.

Idealus spartumas gali būti pasiektas lygiagretaus kompiuterio su n identiškais procesoriais vienu metu vykdančių vieno uždavinio sprendimą ir tai yra n kartų greičiau, negu su vienu procesoriumi. 2.1 dalyje jau buvo atliktas bandymas su identiškais procesoriais. Iš tikrųjų spartumas yra žymiai mažesnis, nes yra papildomų faktorių.

Nežymūs apribojimai **$\log_2$ n** yra žinomi kaip Minsky spėjimas, kad spartumas yra proporcingas dvejetainiu logaritmu nuo procesorių skaičiaus n. Šis spėjimas naudojamas ir priėjimo prie duomenų konfliktams atsitiktinai generuojant adresus. Šie konfliktai sulėtina visus procesus, tuomet keturi procesoriai pasiekia tik dviejų procesorių našumą. Vis dėlto prie duomenų priėjimas realiose aplikacijose yra atsitiktinis. Daugelis aplikacijų turi gerą reguliaraus priėjimo procentą ir vieta (apie tai bus kalbama vėliau), kurie ir padeda padidinti našumą.

Toks „pesimistiškas“ dėsnis arba spartos kritimą. Priklausomai nuo aplikacijos, reali sparta svyruoja nuo **$\log_2$ n** iki **n.** Vis dėlto, viršutinioji riba **n**, praktikoje būna žymiai tikslesnė.

Svarstytinas skaičiavimo uždavinys, kuris gali būti vykdomas vieno uniporcesoriaus:

- Laiko vienetas **T_1=1**.
- Tarkime **f_i** yra tos paties uždavinio priskyrimo **i** procesoriams tikimybė.

- Kiekvienas procesorius dirba vienodai su apkrovos vidurkiu $\mathbf{d_i}$ =1/**i** procesoriui.
- Tarkime, kad vienoda valdymo būdo, naudojant i procesorius, tikimybė, atitinkamai $\mathbf{f_i = 1/n,}$ su kiekvienu **n** valdymo rėžimu: I = 1, 2, 3,...n.

Laiko vidurkis, kuris reikalingas uždavinio sprendimui n procesorinėje sistemoje, yra pateiktas žemiau, kur suma reiškia n valdymo rėžimus (žr. 12 formulę):

$$T_n = \sum_{i=1}^{n} f_i \cdot d_i = \frac{\sum_{i=1}^{n} \frac{1}{i}}{n} \tag{12}$$

Spartos **S** vidurkis, kai koeficientas $\mathbf{T_1}$ = 1 iki T**n,** tuomet (žr. 13 formulę):

$$S = \frac{T_1}{T_n} = \frac{n}{\sum_{i=1}^{n} \frac{1}{i}} \le \frac{n}{\ln n} \tag{13}$$

Sunkaus harmoninio vidurkio principas (*Weighted Harmonic Mean Principle*) yra susijęs su vykdymo greičiu pagal sunkųjį harmoninį vidurkį. Tarkime $\mathbf{T_n}$ yra laikas, reikalingas atlikti **n** skirtingų **k** tipų užduočių. Kiekvienas tipas susideda iš $\mathbf{n_i}$ užduočių, reikalaujančių $\mathbf{t_i}$ sekundžių kiekvienam įvykdymui (žr. 14 formulę):

$$T_n = \sum_{i=1}^{k} n_i \cdot t_i \tag{14}$$

ir (žr. 15 formulę)

$$n = \sum_{i=1}^{k} n_i \tag{15}$$

Pagal apibrėžimą, vykdymo tempas **R** yra įvykių ar operacijų skaičius per laiko vienetą, taigi (žr. 16 formulę):

$$R_n = \frac{n}{T_n} = \frac{n}{\sum_{1}^{n} n_i \cdot t_i} \tag{16}$$

F_i apibrėžkime kaip dalis sugeneruotų rezultatų pagal tempą $\mathbf{R_i}$ ir $\mathbf{t_i = 1/Ir}$ – laikas reikalingas rezultatų gavimui. Tuomet (žr. 17 formulę):

$$R_n = \frac{1}{\sum_{1}^{k} f_i \cdot t_i} = \frac{1}{\sum_{1}^{k} \frac{f_i}{R_i}} \tag{17}$$

kur $f_i = n_i / n$, $\sum_i f_i = 1$, $R_i = \frac{1}{t_i}$.

Pastebėsime, kad 17 formulė išreiškia pagrindinį principą kompiuterio architektūroje per netikslaus perdavimo laiko, ir tai silpna vieta, arba sunkiojo harmoninio vidurkio principas. Šio principo svarba ir 17 formulė duoda:

- Jeigu vieno tempas už pusiausvyros ribos (pvz. žemiau nei kitų), tuomet šis daro įtaką visam sistemos našumui.

Toks požiūris yra skirtas universaliam *Beowulf* blokiniui (*cluster*) klientas-mazgas su homogenine įranga. Tai nėra skirta heterogeniniams mazgų sujungimams per įvairius skirtingus tinklus. Heterogeninis tinklas ir mazgas yra sistemų, kurios išeina už matematinio modeliavimo ribų arba nulemia našumą anksčiau realizacijos, pagrindas.

Svarbus **Amdahl dėsnio** taikymas yra savotiška išraiška sunkiojo harmoninio vidurkio principo. Apibrėžiami du parametrai:

1. Aukščiausias arba lygiagretaus vykdymo tempas $\mathbf{R_H}$**.**
2. Žemiausias arba skaliarinis vykdymo tempas $\mathbf{R_L}$**.**

Jeigu **f** išreiškia dalį gautų duomenų aukščiausiu tempu $\mathbf{R_H}$ ir 1-**f** yra dalis, sugeneruota žemiausiu tempu, tuomet iš 17 formulės gaunama (žr. 18 formulę):

$$R_f = \frac{1}{f / R_H + (1 - f) / R_L} \quad (18)$$

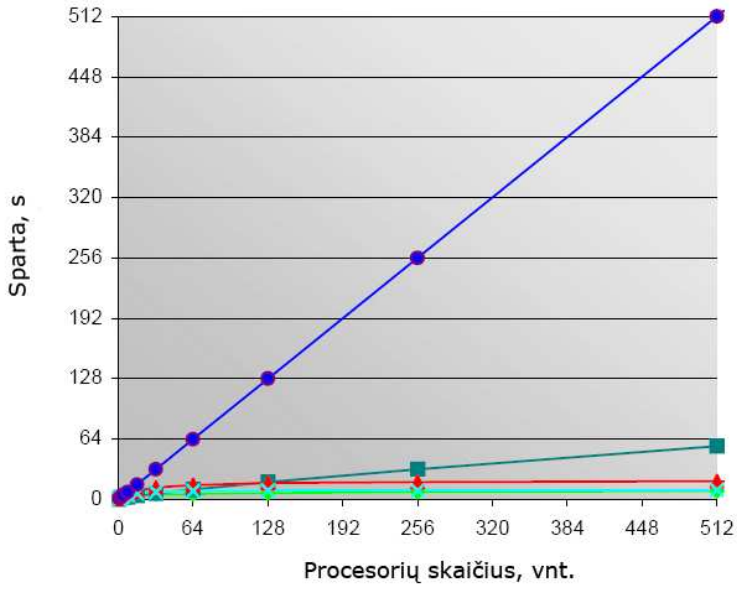
Ši formulė yra žinoma kaip Amdahl dėsnis. Naudojama analizuoti sistemos našumą, kuomet rezultatai gaunami iš dviejų individualių vykdymo rodiklių, tokių kaip vektorinės ar skaliarinės operacijos, ar lygiagrečios, ar paeiliui einančios operacijos. Tai yra naudinga pilnoms lygiagrečioms sistemoms, kuriose vienas rodiklis išeina iš pusiausvyros kartu su kitais. Pavyzdžiui, žemas rodiklis gali būti sukeltas Įvedimo / Išvedimo ryšio operacijų ir aukštas rodiklis – vektorių, atminties arba kitų operacijų. Amdahl dėsnis taip pat gali būti pritaikytas kitų programų vykdymui. Iš šio dėsnio matome, kad maža dalis **f** nuoseklaus arba nelygiagretaus skaičiavimo smarkiai riboja spartą, kuri gali būti pasiekta su **p** procesoriais. Atsižvelgiant į užduoties laiko vienetą, kuriai dalis **f** yra nelygiagreti (tai užima tiek pat laiko, kai f lygiagrečiame ir nuosekliame kompiuteriuose) ir lieka **1-f** yra pilnai lygiagretus (startuoja laiku **(1-f)/p p** –procesoriniame kompiuteryje).

Pastebėsime, kad Amdahl formulė, kai **f = 0** išreiškia idealias sąlygas su **n** laiko sparta.

Taigi, norint pateikti rezultatų kreivę, reikia nustatyti Beowulf blokinio našumą, pagal procesorių, kuris yra naudojamas, skaičių. Praktikoje tai vis dėlto nenaudojama. Praktikoje tai vis dėlto labiau vyrauja, jeigu geras tinklo ryšys, gaunama atitinkama sparta (13 pav.). Žemiau parodyti trys paveikslai su gautais duomenimis pagal anksčiau nurodytus principus (13, 14, 15 pav.). Visi trys paveikslai vaizduoja visas tendencijas ir nedidelio procesorių skaičiaus (grafikai sugeneruoti programinės įrangos *MATLAB* paruošto išeities kodo pagalba (Priedas Nr. 1)).

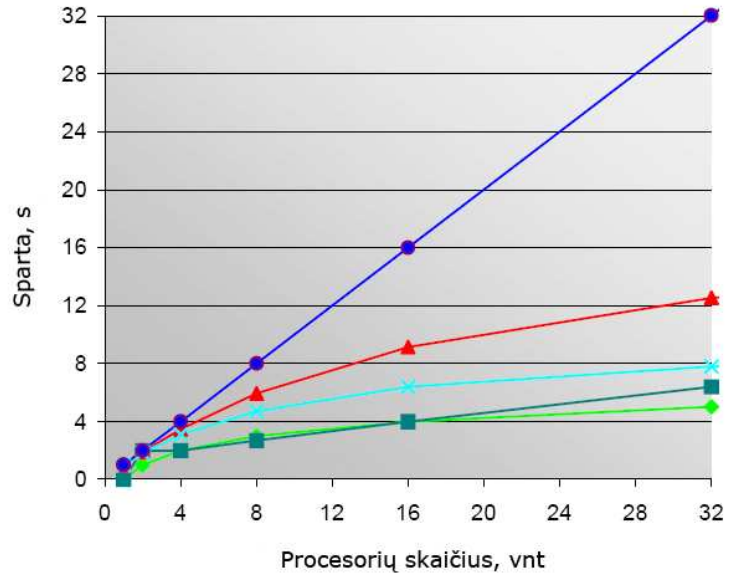
Iš analizės ir grafikų galima pastebėti, kad tipinės komercinės multiporcesorinės sistemos susideda iš dviejų arba keturių procesorių. Suprantama, tai atsiliepia atitinkamų sistemų našumui ir veikimo principams.

Pastebėsime, kad Amdahl dėsnis yra faktas, kai daugelis aplikacijų stokoja lygiagretumo, tokiu būdu mažindami spartą, kuri pasiekiama, kai veikia daug procesorių [1, 2, 3].



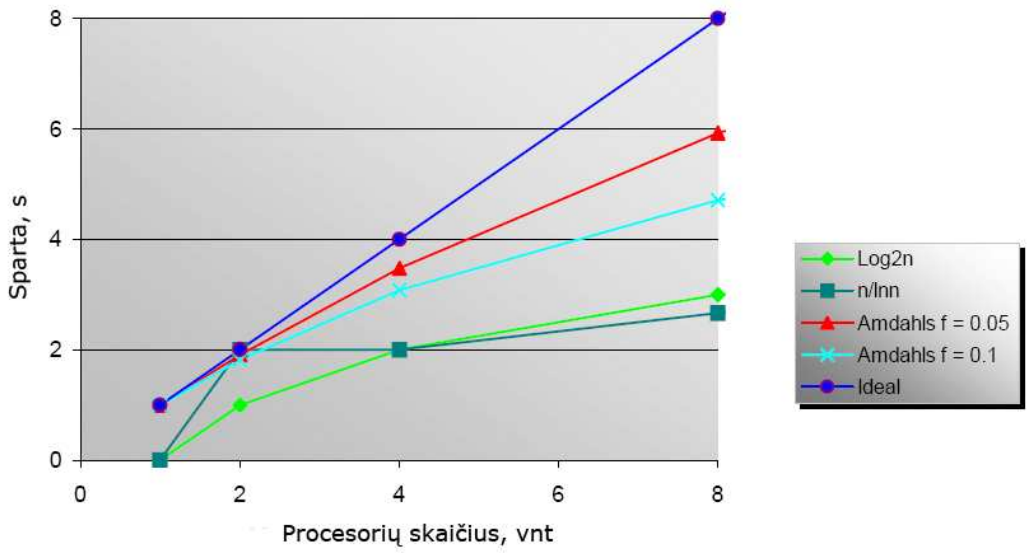
13 pav. **Procesorių skaičiaus ir spartos santykis (vaizduojami skirtingo galingumo procesoriai skirtingomis veikimo sąlygomis)**

Daugeliu atveju, siekiant nustatyti išorinį poveikį procesorių lygiagrečiam veikimui, įvedama tikimybės sąvoka. Tikimybė šiuo atveju filtruoja išorinius faktorius, kurie gali įtakoti lygiagrečius skaičiavimus (14 pav.). *ANSYS* realizuota kaip lygiagreti sistema gali veikti nepriekaištingai su uždaviniais iki 100 mln. laisvės laipsnių, prie tam tikrų idealių sąlygų.



14 pav. ***Procesorių skaičiaus ir spartos santykis (vaizduojami skirtingo galingumo procesoriai ir skirtingomis veikimo sąlygomis)***

Išoriniams veiksniams veikiant, tokios lygiagrečios programos sparta ir efektyvumas sumažėja (14 pav.).



15 pav. **Skirtingi n-procesoriaus spartos vertinimai**

Atlikus grafinį vertinimą buvo svarbu nustatyti lygiagrečios sistemos veikimo principus ir aplinkas (15 pav.). Formuojant blokinį svarbu atsižvelgi į anksčiau minėtus teiginius.

3.3. *ANSYS* lygiagretaus vykdymo sprendimai

Kaip paaiškėjo iš literatūros analizės, egzistuoja 4 *ANSYS* lygiagretaus veikimo sprendimai, bet svarbiausi ir optimaliausi bei efektyviausi metodai (vadovaujamasi [4] testais) yra du:

- DDS (Distributed Domain Solver);
- AMG (Algebraic Multigrid Solver).

3.3.1. *ANSYS* lygiagretaus vykdymas DDS metodu

Detalizuojant kiekvieną iš jų, galima pasakyti, kad šiuo metu naudojamas DDS metodas, nes pagal kai kuriuos testus, gaunama, kad šiuo metu sprendžiami sudėtingesni, reikalaujantys daugiau atminties uždaviniai.

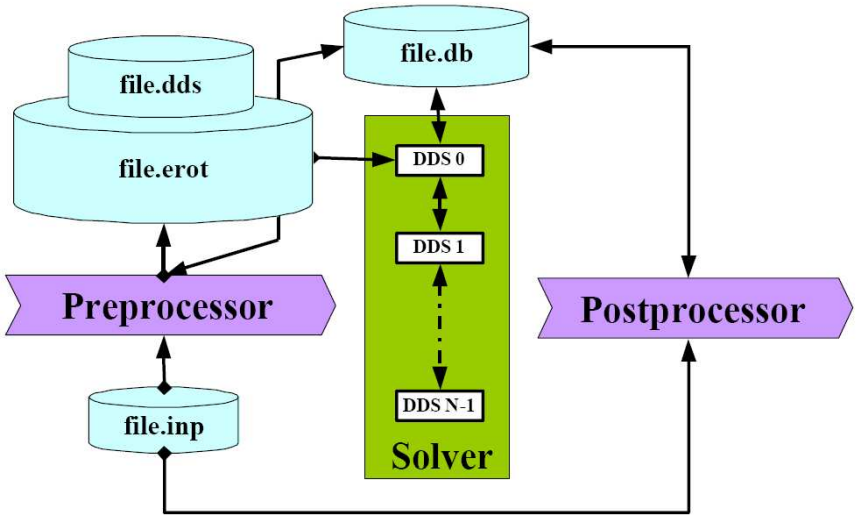
DDS metodas yra grindžiamas išdalinimu į sritis. DDS metodo koncepcija yra ta, kad modelis padalinamas į mažas sritis ir kiekviena iš tų sričių analizuojama atskirai. Kiekviena iš tų sričių turi nuo 1000 iki 10000 laisvės laipsnių. Išėjimu už srities ribų rūpinasi pakartotini

sprendimai. Kadangi kiekviena sritis yra analizuojama individualiai ir sprendimas yra nepriklausomas nuo kitų, todėl yra laikoma, kad šis metodas tinka lygiagrečiam vykdymui.

DDS *ANSYS* yra skirtas didelių modelių analizei SMP ir DMP architektūros kompiuteriuose, taip pat blokinių sistemose. Pusiausvyra bus pasiekta tik iteratyviais metodais, bet netinkamais modeliais.

DDS yra skirtas naudoti išskirstytos atminties terpėje arba išskirstytos arba apjungtoje išskirstytos ir padalintos atminties terpėje. Tai skirta didelei statinei ir pilnai analizėms su simetrinėmis matricomis.

16 pav. vaizduojama *ANSYS* lygiagretaus veikimo mechanizmas pagal DDS metodą. Tokiu būdu galima teigti, kad būtina naudoti lygiagretumą sudėtinguose modeliuose.



16 pav. ***ANSYS veikimo mechanizmas DDS metodu [50]***

Kaip matome iš 16 pav. – egzistuoja sričių eilė. Tai pagrindinis lygiagretaus veikimo aspektas su pirminiu ir antriniu procesoriumi. Iš 16 pav. pastebėsime, kad:

- Išdalinama sistema sprendimo funkcijomis į atskirus darbus;
- Sistema veikia tik su dideliu kiekiu atminties;
- DDS metodas veikia vienodų mazgų blokinyje.

3.3.2. *ANSYS* lygiagretaus vykdymas AMG metodu

Kitas lygiagretaus veikimo metodas yra Algebraic Multigrid Solver (AMG). AMG yra iteratyvus metodas. Tokie sprendimai atliekami labai sunkiai su nelinijiniais modeliais (atminties ir disko vietos reikalavimai yra dideli). Iteratyvūs sprendimai yra žymiai veiksmingesni,

sprendžiant didelius modelius, nes reikalaujama santykinai nedaug kieto disko vietos ir atminties.

Iteratyvaus sprendimo konvergencijos sparta yra nulemta „Sąlygos skaičiaus“. Sąlygos skaičius yra tikrinių verčių maksimumo ir minimumo santykis. Didžiausias sąlygos skaičius, kai kuo daugiau iteracijų konverguoja. 1 sąlygos sakinys yra idealus (konverguoja vienoje iteracijoje). Prieš sąlyga, C yra naudojamas, siekiant sumažinti [K] sąlygos skaičių, taigi gaunama greitesnė konvergencija. Iteratyviame procese $C^{-1}Ku = C^{-1}f$ yra išsprendžiama, kur C yra panašus į K. Jeigu C = K, tuomet labiausiai sąlygojama, pavyzdžiui, kai tikslus sprendimas u yra gaunamas per vieną iteraciją. Kuo C artimesnis K, tuo geresnis sąlygos skaičius. *ANSYS* AMG priešsąlygotumas yra grindžiamas Algebraic Multigrid metodu naudojant apytikrį K. ANSYS AMG lygiagretus veikimas yra įgyvendintas POSIX bibliotekos pagalba.

Tai apibendrinus šiuos du metodus pastebėsime, kad eksperimente naudingiau naudoti yra DDS lygiagretaus veikimo metodą, kuris reikalauja mažiau atminties bei procesorių tokio pat uždavinio sprendimui, lyginant su AMG metodu [50]. Aprašant krumplinės pavaros su n-laisvės laipsniais uždavinį, būtina taikyti šiuo atveju DDS metodą. ANSYS kompanija teigia, kad DDS metodas blokinyje yra veiksmingesnis būdas didelio sudėtingumo uždavinių sprendimui.

4. PROJEKTINĖ DALIS

4.1. Tinklo architektūros parinkimas ANSYS veikimui

Žinant, kad ANSYS gali veikti kaip išskirstyta sistema, būtina numatyti tinklo topologiją. Tokiu būdu analizuojamos tokios ANSYS realizacijos galimybės per interryšius:

- InfiniBand (rekomenduojama);
- Myrinet (rekomenduojama);
- Gigabit Ethernet (rekomenduojama);
- Ethernet (nerekomenduojama).

Pastebėsime, kad interryšiai turi labai daug įtakos atminties skirstymui ir jos greičiui. Todėl būtina nustatyti tokius aspektus:

- intersąryšio laipsnis, kuris reikalingas tarp kiekvieno mazgo, kad būtų galima paleisti aplikaciją;
- procesoriaus efektyvumas intersąryšio pralaidumui blokinyje (*cluster*).

Parodant kiekvieno mazgo vidaus procesorių ir atminties struktūras, išskirstytos atminties architektūra yra visų pirma charakterizuojama mazgų interryšių tinklu.

Toks tinklas gali būti identifikuotas kaip grafas su:

- viršūnėmis atitinkančiomis procesoriaus-atminties mazgus;
- briaunomis atitinkančiomis ryšį tarp jų.

Naudojant dvikryptį sujungimo ryšį – naudojamos dvikryptės briaunos. Dvikryptės briaunos reiškia dvikryptį ryšį, kuris gali būti ir *half duplex*, ir *full-duplex* ryšys.

Pastebėsime, kad yra keletas svarbių interryšinio tinklo parametrų:

- tinklo diametras;
- padalijamas pralaidumas;
- viršūnės arba mazgo laipsnis.

Projektuojant blokinio struktūrą būtina numatyti metodus, kurie galėtų padėti realizuoti blokinį ANSYS pagrindu. Būtina naudoti baigtinių elementų lygiagretumą. Baigtinių elementų lygiagretus veikimas, kai kiekvienas procesorius skaičiuoja subsrities išraiškas. Tų reikšmių vektorius yra mazgų sąsaja kiekvienoje subsrityje, turi būti keičiamas kiekviename procesoriuje per laiko žingsnį. Kad būtų padaryta tai, kiekvienas procesorius turi žinoti:

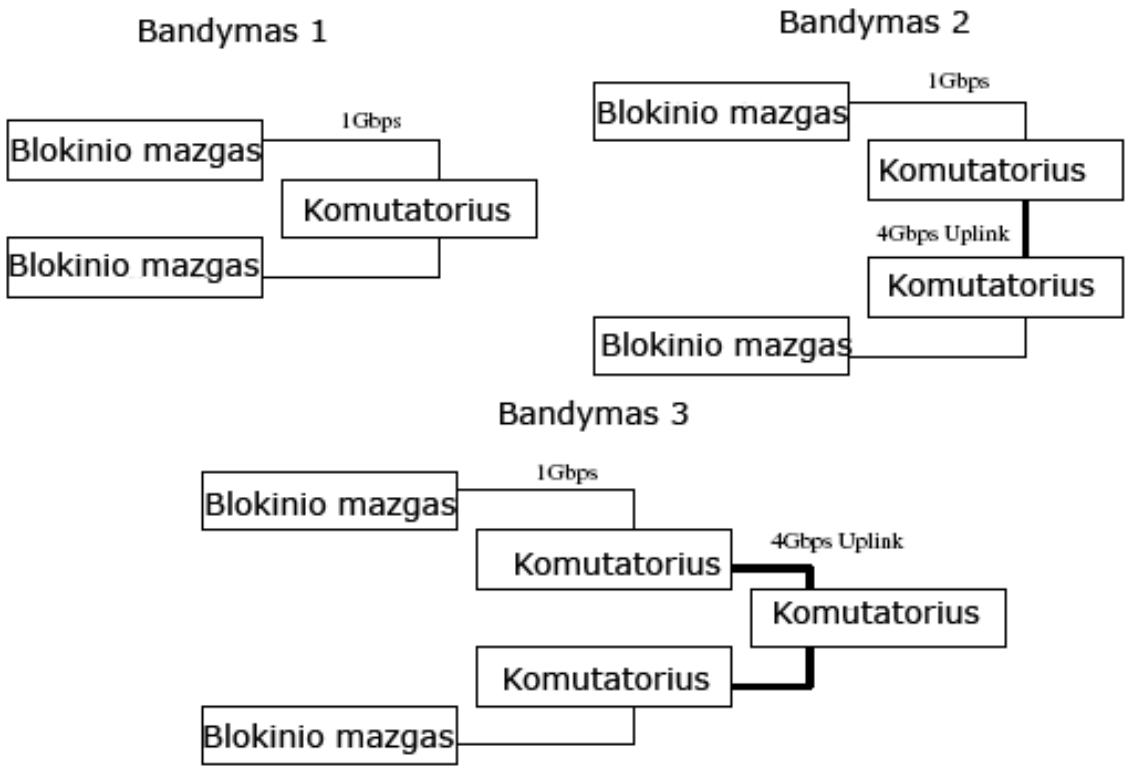
- procesorių skaičių ir sąrašą;

- mazgų skaičių kiekvienoje padalintoje sąsajoje ir tų mazgų sąrašą. Tokia informacija yra gaunama pirminio vykdymo metu ir nekeičia baigtinių elementų tinklo ryšio ir subsričių sąsajos.

Su šia informacija kiekvienas procesorius žino, kuriam mazgui reikia siųsti ir iš kurio gauti. Siūloma atlikti dekompozicija, tai yra atskirti pagrindinį mazgą ir sekančius mazgus.

4.2. Blokinio formavimas ANSYS pagrindu

Suformuosime tris pagrindinius blokinius, kurių pagrindu būtų galima realizuoti ANSYS, kaip išskirstytą sistemą. 17 pav. vaizduojama tokių blokinių galimos schemos.



17 pav. ***Galimi tinklo architektūros sprendimo variantai***

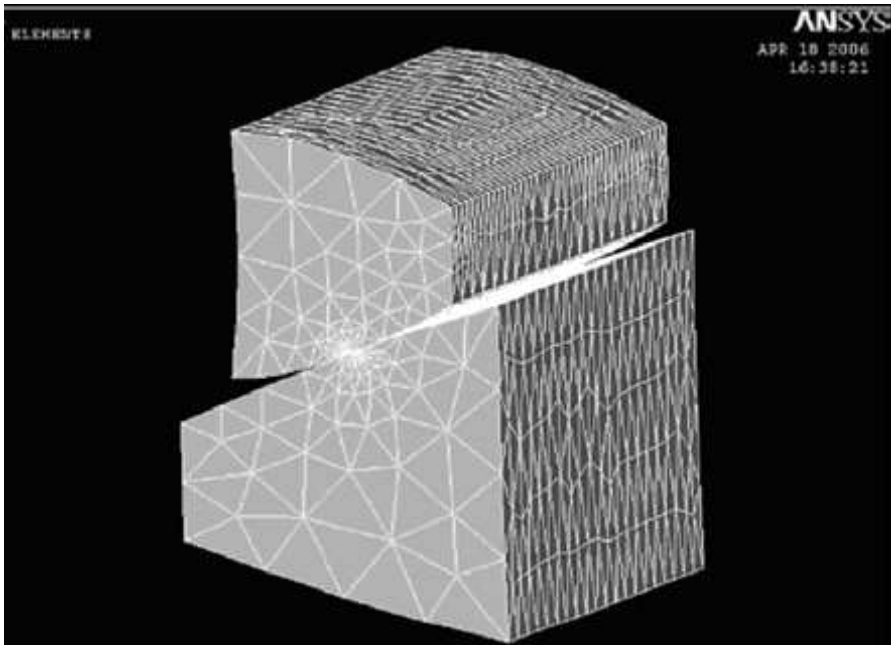
17 pav. pateikta galima tinklo sprendimo sąsaja, kurios pagalba gali būti realizuotas ANSYS išskirstymas. Tokiu būdu reikia pastebėti, kad trijų bandymų, pralaidumo testai iš tikrųjų skiriasi. Tai nulemia kai kurių aparatų pralaidumo lygis. Siekiant realizuoti kuo spartesnį blokinį, būtina pasirinkit blokinio mazgo dydį ir pagal tai reguliuoti tinklo įrangos kiekį bei jos pralaidumą.

5. EKSPERIMENTINĖ DALIS

Sukurtas blokinys leido realizuoti *ANSYS* išskirstytos sistemos pagrindu. Suprojektuotos galimos blokinio struktūros leido *ANSYS* testuoti įvairiais požiūriais. *MATLAB* programinės įrangos pagalba buvo sukurtas išeities kodas, kuris leido nustatyti blokinio efektyvumą ir spartą. Eksperimente buvo naudojamas ***4, 6, 8, 9, 16*** uniprocesorinių sistemų blokiniai (sujungti aukštą pralaidumo lygį palaikančiais tinklo įrenginiais), kurie ir leido nustatyti tam tikrus galimus lygiagretaus skaičiavimo greičius bei tikslumus.

Sprendžiamo mechatronikos uždavinio sudėtingumas nusakomas statinių lygčių gausa, kurios ir charakterizuoja sričių n-laisvės laipsnius n-matėje sistemoje.

Detalizuojant uždavinį, reikia pastebėti, kad geometrinis krumplinės perdavos modelis sudarytas naudojant grafinę programą *SOLID WORKS* ir *ANSYS* išeities kodus (2 priedas), varančiajam z_3 ir varomajam z_4 krumpliaračiams priskirtos identiškos realios medžiagų charakteristikos: tamprumo modulis $E = 2 \cdot 10^{11} \frac{N}{m^2} \cdot$ir Puasono koeficientas $\mu = 0{,}3$. Geometrinis krumpliaračių krumplių modelis suskaidytas tetraedriniais ir heksaeidriniais SOLID95 tūriniais elementais, kaip pavaizduota 17 pav.



17 pav. ***Varančiojo ir varomojo krumpliaračių krumplių, esančių poliniame susikabinime, suskaidytų baigtiniais elementais bendras vaizdas***

Realizuojamo uždavinio sprendimui reikalingi tam tikri dideli techniniai ir laiko sąnaudų kaštai, kurie ir apsprendžia tokio uždavinio sprendimo tikslumą ir išbaigtumą. Tai yra krumpliaračių matematinis modeliavimas atliktas taikant statikos lygtį [4] (žr. 18 formulę):

$$[K]\{u\} = \{F\} \quad (18)$$

Atliekant eksperimentą su vienu kompiuteriu, kurio parametrai buvo tokie (žr. 8 lentelę), gauta, kad atliekant kiekvieną iteraciją, kompiuteris išsprendė daugiau nei 1299000 statikos lygčių.

8 lentelė. ***Kompiuterio, naudoto bandymo metu, sprendžiant uždavinį, techninės savybės***

Procesoriaus dažnis	3,2 GHz
Atminties kiekis	2GB
Kietojo disko talpa	120GB

Konstruojant blokinį, naudotos tokios uniprocesorinės sistemos, kaip (žr. 9 lentelę):

9 lentelė. ***Konstruojamo blokinio techninės savybės***

Procesoriaus dažnis	3,2 GHz
Atminties kiekis	2GB
Kietojo disko talpa	120GB
Tinklo pralaidumas	100Mb/s
Tinklo plokštės	*Gigabit* technologiją palaikanti plokštė

Pradedant konstruoti blokinius, buvo atliktas galimų sistemų veikimo preliminarus įvertinimas (*MATLAB* išeities kodas pateiktas 1 priede).

Virtualaus blokinio modeliavimas pagal teorines ir praktinių testų prielaidas

Atliekant bandymus su virtualiu blokiniu (blokinys buvo sukonstruotas, vadovaujantis baigtinių elementų metodu) buvo būtina parinkti tokias atsitiktinių dažnių sekos (parodyta žemiau), kurios leido imituoti tikrų procesorių apkrovas blokinyje, spendžiant mechatronikos uždavinį:

y_60 = [11.256163 2.97289 2.085062 1.533152 1.26716 0.996021];

y_120 = [45.810661 11.491478 7.839005 5.383871 4.187208 3.232426];

y_180 = [111.721949 26.280184 17.838946 12.176307 9.413069 7.149453];

y_240 = [205.516207 48.729024 32.982407 22.048036 17.014949 13.091097];

y_300 = [324.565092 80.836117 52.27633 35.216773 27.207582 20.906174];

y_360 = [469.384824 121.780736 79.046601 52.920637 39.762668 30.649895];

y_420 = [640.612062 169.099137 111.880208 73.514546 55.915544 42.594644];

y_480 = [837.288399 222.474732 148.684467 98.82164 74.699427 56.220476];

y_540 = [1058.459214 282.420194 189.511962 127.652788 96.424833 72.806032];

y_600 = [1303.937413 349.278586 234.46219 159.520055 121.226566 91.406496];

y_660 = [1572.888626 425.870722 283.411603 194.181991 148.103974 112.493721];

y_720 = [1864.402242 503.134313 336.78972 231.952082 176.332697 135.101946];

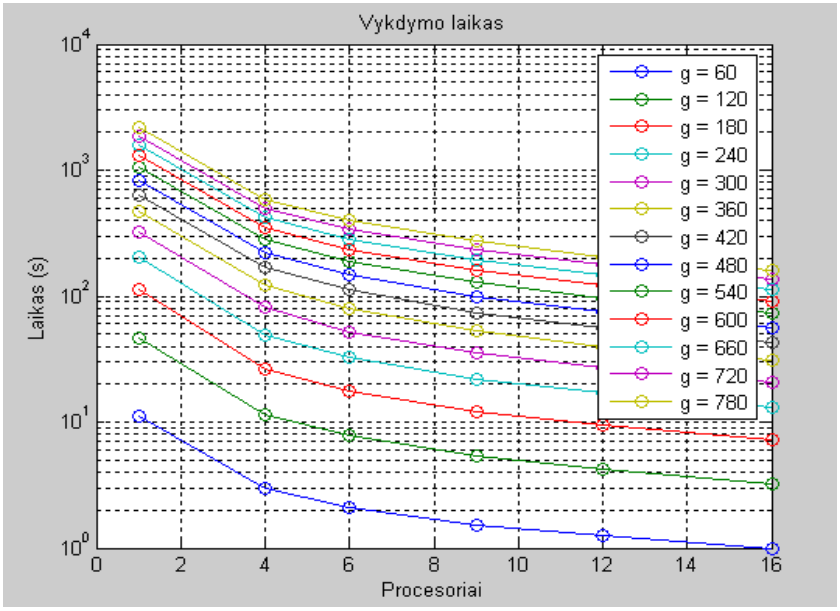
y_780 = [2175.61912 591.267074 394.989867 271.357013 206.680045 158.929052];

Atlikus šiuos bandymus paaiškėjo, kad konstruojamų blokinių našumas nebus labai didelis. Tačiau siekdami pagrįsti išsikeltą hipotezę buvo būtina atlikti realų eksperimentą su tam tikrais blokinių parametrais.

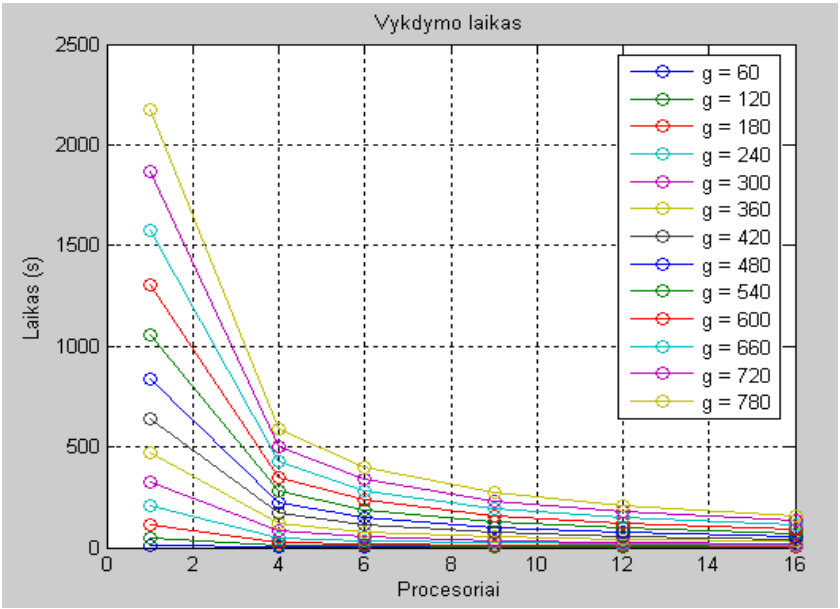
Pirmiausia būtina detalizuoti gautus virtualaus blokinio procesorių apkrovų modelius. Iš 18 pav. matome, kad vertinimas virtualiame blokinyje atliekamas naudojant tokius parametrus, kaip:

- procesorių skaičius (vnt.);
- laikas (s).

Kaip iš 18 pav. matome, netikro laiko skalė grafike yra labai didelė, nes virtualus blokinys gali imituoti pasirinktą laiko skalę. Šiuo atveju taip ir buvo padaryta. Procesorių skaičius grafike lygus 16. Kiekviena kreivė atpažįstama iš legendos. Kiekviena kreivė vaizduoja tam tikro dydžio tinklelį, kaip savotiškas apkrovas arba laisvės laipsnius. Tokio tinklelio taikymas buvo būtinas, norint imituoti laisvės laipsnius virtualiame blokinyje (tinklelis – tai sričių rinkinys). Esant mažiausiam tinkleliui ($g = 60$) pastebime, kad lygiagretaus vykdymo pradžioje inicializacijos ir pirmosios operacijų laikas pakyla iki 10 sekundžių, o po to, kaip buvo būtina detalizuoti teorinė šio darbo dalyje, lygiagretus vykdymas išryškėja. Laiko sąnaudos krenta. 16 procesorių lygiagretaus vykdymo sistema užtrunka vos 1 sekundę, kad būtų gauti rezultatai. Vykdant sudėtingesnį tinklelį, būtina pastebėti, kad laiko sąnaudos veikiant 16 procesorių tampa apylygės ir skiriasi tik sekundės dalimi. Grafikai sugeneruoti, vadovaujanti teorinėje dalyje pateiktomis formulėmis.

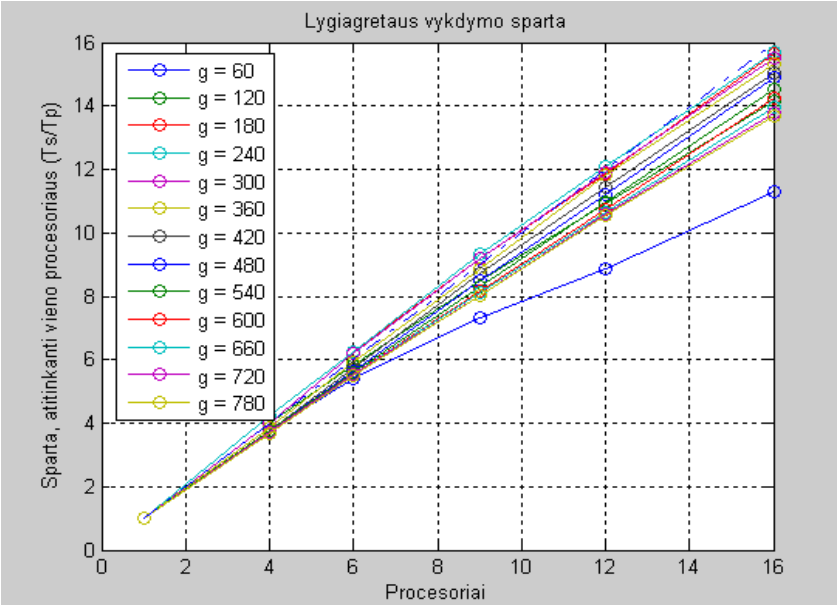


18 pav. ***Mechatronikos sistemos uždavinio sprendinio paieška virtualiame blokinyje, modeliuojant procesorių ir laiko sąnaudų santykį***



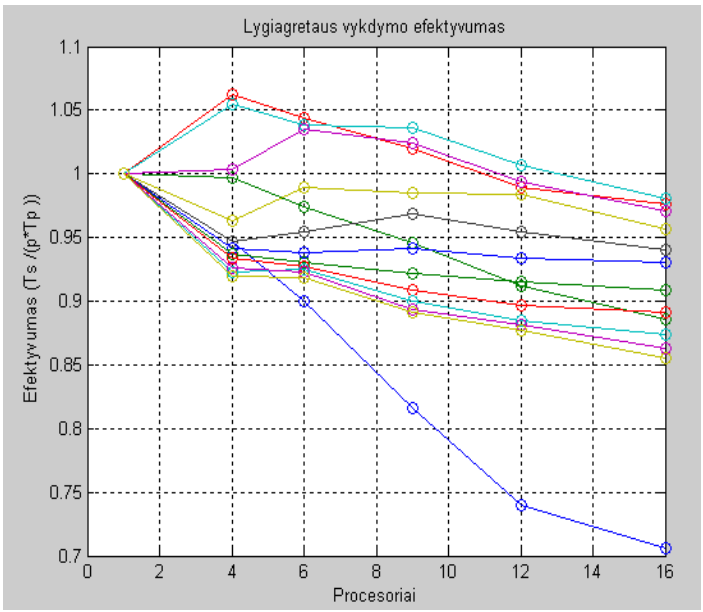
19 pav. ***Mechatronikos sistemos uždavinio sprendinio paieška virtualiame blokinyje, modeliuojant procesorių ir laiko sąnaudų santykį***

Pastebėsime, kad 19 pav. laiko skalė dar labiau padidinta, siekiant pavaizduoti vieno procesoriaus laiko sąnaudos lengviausio ir sunkiausio uždavinio sprendimo kontekste. 19 pav. paryškinamas lygiagrečių skaičiavimų privalumai laiko ir procesorių skaičiaus atžvilgiu.



20 pav. ***Mechatronikos sistemos uždavinio sprendinio paieška virtualiame blokinyje, modeliuojant blokinio ir pavienių procesorių spartas***

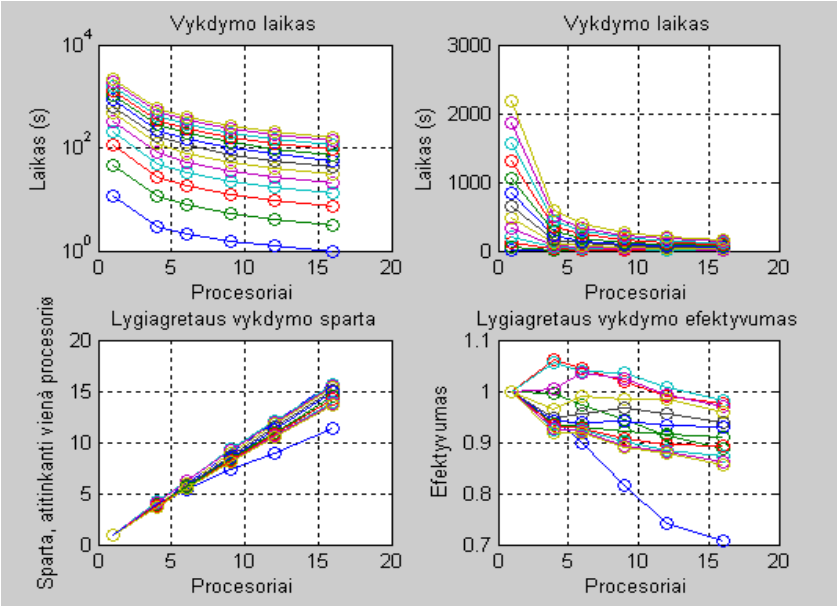
Kaip matome iš 20 pav. paryškinama lygiagretaus veikimo (uždavinio sprendimo principas). Sprendžiant nesudėtingiausią uždavinį, pastebėsime, kad 16 procesorių blokinio atliekamų skaičiavimų sparta yra didesnė, tiesiog prilyginant 8 pavienių procesorių galimybėms. Uždaviniui sudėtingėjant, matome, kad spartos priartėja viena prie kitos, tačiau 16 procesorių blokinio naudojimas yra perspektyvesnis.



21 pav. ***Mechatronikos sistemos uždavinio sprendinio paieška virtualiame blokinyje, modeliuojant blokinio ir pavienių procesorių efektyvumus***

Iš 21 pav. matome, kad sprendžiant nesudėtingus uždavinius 16 procesorių blokinio efektyvumas nėra aukštas. Uždaviniui sudėtingėjant artėjama prie konkretaus efektyvumo,

efektyvumu tokio blokinio naudojimas nepasižymi. Efektyvumas pasieks didžiausią reikšmę, kai g = 60 uždavinys.



22 pav. ***Uždavinio sprendinio paieška virtualiame blokinyje***

Sulyginus visus gautus virtualaus blokinio matavimų rezultatus, sprendžiant priartintą mechatronikos uždavinį prie realaus (tai atliekama, naudojant tinklelio principą). Pastebėsime, kad augant spartai, mažėja efektyvumas. Tai gali būti nulemta tam tikrų faktorių, kuriuos buvo būtina charakterizuoti teorinėje šio darbo dalyje. Panaudojus virtualų blokinį ir gavus virtualių bandymų rezultatus, susijusius su sprendžiama realia mechatronikos sistema, būtina padaryti tokias išvadas:

- virtualaus blokinio formavimas buvo priartintas prie realaus tinklelio principo;
- virtualų blokinį buvo būtina modeliuoti, siekiant išsiaiškinti kuriamo blokinio galimus efektyvumus ir spartas, jau atsižvelgiant į tam tikrus išorinius faktorius.

Realaus blokinio konstravimas

Konstruojamas realus blokinys, siekiant pagrįsti darbo hipotezę, kad *ANSYS* gali būti realizuotas ir pritaikytas lygiagrečių skaičiavimų atveju sprendžiant suformuluotą mechatronikos uždavinį (išeities kodas pateiktas 3 priede). 22 pav. vaizduojami gauti blokinių rezultatai, atspindint santykį tarp procesoriaus skaičiaus ir skaičiavimo laiko. Parodoma, kad su 16 procesorių galima gauti sprendimą per 11,6 karto greitesnis nei vieno procesoriaus atliekami vykdomi sprendimai. 10 lentelėje matome gautus skaičiavimo laiko rezultatus realaus bandymo metu, *ANSYS* veikiant kaip išskirstytai lygiagrečiai sistemai, lyginant tris atvejus su trejais skirtingais laivės laipsnių skaičiais, analizuojant 1 ir 16 procesorių blokinių sistemas.

10 lentelė. ***Skaičiavimo laikas lyginant 1 ir 16 procesorių***

Laisvės laipsniai (D.O.F) [milijonais]	1Proc. [min]	16Proc. [min]
0,5	41	3,7
1,5	170	14,7
2,0	250	23,6
3,0	400	45,3

Iš 10 lentelės matome, kad 16 procesorių sparta realaus bandymo metu sprendžiant mechatronikos sistemą yra 12 kartų didesnė nei uniprocesorinės sistemos. Todėl būtina nustatyti, kad sprendžiama mechatronikos sistemą bandymo metu buvo išspręsta tik iki 3 mln., siekiant išsiaiškinti tam tikrus lygiagretaus veikimo tempus. Todėl galima padaryti išvadą, kad didinant laisvės laipsnius per 1 milijoną, procesorių apkrovos išauga žymiai, nes mūsų sprendžiamos sistemos atžvilgiu statikos lygčių, aprašančių sritis, skaičius išauga žymiai.

Konstravimas DDS metodu

Pagal gautus eksperimento rezultatus galime modeliuoti beveik 8 mln. laisvės laipsnių sistemos parametrus, kurie buvo gauti atlikus analizę DDS metodu, aprašytu teorinėje dalyje (žr. 11 lentelę):

11 lentelė. ***Apie 8 mln. laisvės laipsnių modeliuojamos sistemos dedamosios***

Elementų skaičius	1,736,728
Mazgų skaičius	2,441,863
Laisvės laipsniai	7,325,589
Subsričių skaičius	6,117
Procesorių skaičius	64
Skaičiavimo laikas	42 [min]

Lyginant *ANSYS* kompanijos oficialius testus su gautais eksperimento rezultatais, išryškėja tam tikros tendencijos:

- koks bebūtų blokinio našumas, didėjant sistemos laisvės laipsnių skaičiui, didėja laiko sąnaudos ir galima teigti kardinaliai;
- matome, kad *ANSYS* sistema gali teisingais skaičiuosit sistema DDS metodu iki 100 mln. laisvės laipsnių. *ANSYS* šiuo atžvilgiu yra rinkos lyderis.

Taigi, hipotezė pagrįsta, remiantis gautais virtualaus ir realaus blokinio skaičiavimo rezultatais, nagrinėjama konkreti mechatronikos sistema įgauna didesnį sprendimo pagreitį didinant techninius ir ryšio parametrus.

IŠVADOS

Stiprėjant mokslo vystymosi tendencijoms – neišvengiamai yra būtina taikyti vienų mokslų metodus kituose. Todėl tokioms tendencijoms ryškėjant – buvo nuspręsta apjungti mechatronikos ir informatikos sričių reikalingas dalis.

Siekiant tikslo, suformuluotų problemų sprendimo ir hipotezės pagrindimo analizės ir konstravimo metu buvo gauti rezultatai, kurie ir iššaukia atitinkamas išvadas:

1. Atlikta mokslinės literatūros analizė rodo, kad šiuolaikinės mechatronikos moksle tikslinga taikyti naujausius informacinių technologijų metodus ir įrankius, atliekant matematinių modelių konstravimą, loginio lygmens architektūros parinkimą konstruojamai sistemai, sistemos struktūrizavimui, standartizavimui bei optimizavimui.
2. Sukurta išskirstyta sistema, pagrįsta teorinėmis prielaidomis (šio baigiamojo darbo hipoteze) ir dėsniais leidžia nustatyti svarbiausius teorinius tokios sistemos veikimo principus, parametrus, sąryšį su lygiagretumu, tinklo topologiją, sąryšį su n-laisvės laipsnių sritimis, mechatronikos sistemomis.
3. Vadovaujantis atliktu išskirstytų sistemų preliminariu modeliavimu, buvo nustatyta, kad egzistuoja tam tikros teorinės prielaidos, leidžiančios vertinti konkrečius sistemos parametrus ir tokiu būdu pasiekti optimalaus iškeltos hipotezės pagrindimo.
4. Sukurtas blokinys, kurio veikimo principai pagrįsti ir įrodyti preliminariais skaičiavimais, leido išspręsti ir kartu įrodyti, kad mechatronikos sistemų sprendimus būtina „rišti“ su tam tikrais išoriniais parametrais ir tai leis realizuoti iki tam tikro lygio konkrečią mechatronikos sistemą, kuri yra aprašoma sudėtingomis statikos lygtimis.

Rezultatų panaudojimo rekomendacijos

Mechatronikos mokslo ir informatikos apjungimas duoda gerus rezultatus. Pastebėta, kad apjungiant šias sritis galima gauti greitesnius ir tikslesnius sprendimus. Klaipėdos universiteto Mechatronikos mokslo instituto veikla leis ateityje bendradarbiauti, sprendžiant dar sudėtingesnius uždavinius.

Išskirstytos sistemos taikymas ANSYS atveju leidžia optimizuoti konkrečios mechatronikos sistemos sprendimus, todėl galima išskirti tokius pasiūlymus:

- ANSYS taikymas Klaipėdos universiteto Mechatronikos mokslo institute suteiks tikslumo ir operatyvumo.
- Reikalinga įsigyti aukštos kokybės įrangą tolesnių uždavinių (įskaitant kompiuterius ir tinklo įrangą) sprendimui.

- Atlikti detalią tokios sistemos realizacijos poreikio analizę, nes gauti eksperimento rezultatai parodė, kad tokios sistemos taikymas yra nebrangus, bet svarbiausia perspektyvus.
- Reikalinga labiau plėtoti lygiagretaus veikimo diegimą Lietuvoje, nes Pasaulyje dominuoja JAV, Lenkijos, Vokietijos ir Kinijos mokslininkai, kurie perprato tokių sistemų naudingumą.

LITERATŪRA

1. Kapila V. *Mechatronic Systems Fundamentals [Book Review]*. Digital Object Identifier 10.1109/MCS.2005.1499392. Volume 25, Issue 4, Aug. 2005. 73 – 77 p.
2. Rose Xiujuan Lu, Clarence W. de Silva, Marcelo H. Ang Jr., Jim A.N. Poo, and Henk Corporaal. *A New Approach for Mechatronic System Design: Mechatronic Design Quotient (MDQ)*, Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics Monterey, California, USA, 24-28 July, 2005.
3. Oficiali Klaipėdos universiteto Mechatronikos mokslo instituto interneto svetainė – www.ku.lt/mmi [žiūrėta 2006-03-20].
4. Oficiali ANSYS kompanijos interneto svetainė – http://www1.ansys.com/cgi-bin/HardwareSupport/recommended/recommended.html.
5. Kuan-ChingLi, Hsun-Chang Chang' Chao-Tung Yan. *On Construction of a Visualization Toolkit for MPI Parallel Programs in Cluster Environments.* Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05), 1550-445X/05 IEEE. 2005.
6. Angela C. Sodan. *Message-Passing and Shared-Data Programming Models—Wish vs. Reality*. Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05) 1550-5243/05 IEEE. 2005.
7. Grama A. and others. *Introduction to Parallel Computing*. Second Edition, Addison Wesley, 2003. ISBN: 0-201-64865-2.
8. Kontrimas R., Bulbenkienė V., Andziulis A.. Išskirstytos sistemos taikymas mechatronikos uždavinių sprendimui. Klaipėdos universitetas, JTF informatikos inžinerijos katedra. 2006.
9. Mažeika P., Didžiokas R., Vasylius M., Barzdaitis V. *Krumplinių perdavų su riedėjimo guoliais modeliavimas ir eksperimentinis tyrimas*. 2006.
10. Luc Hogie, Pascal Bouvry, Fr´ed´eric Guinand. *An Overview of MANETs Simulation.* Electronic Notes in Theoretical Computer Science 150, 2006. 81–101 p.
11. Kevin Walsh and Emin Sirer. *Staged simulation: A general technique for improving simulation scale and performance.* ACM Trans. Model. Comput. Simul., 14(2), 2004. 170–195 p.
12. John Wiley & Sons, Inc., Hoboken, New Jersey. *Tools and environments for parallel and distributed computing.* Published by Published simultaneously in Canada, 2004. 4-5 p.
13. Seung Hoon Paik a, Ji Joong Moon a, Seung Jo Kim b, M. Lee. *Parallel performance of large scale impact simulations on Linux cluster super computer*. Computers and Structures 84, 2006. 732–741 p.

14. Min Huang, Brett Bode. *A Performance Comparison of Tree and Ring Topologies in Distributed Systems.* Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) 1530-2075/05, 2005.
15. Jean Bacon, Tim Harris. *Operating Systems: Concurrent and Distributed Software Design*. IEEE Trans. Parallel and Distributed Systems, vol. 10, 2003. 234-241 p.
16. Kontrimas R., Bulbenkienė V., Andziulis A. *Išskirstytos sistemos taikymas.* Informacinės technologijos 2006: studijų plėtra Lietuvos ekonominių bei visuomenės pokyčių kontekste. V mokslinės praktinės konferencijos pranešimų medžiaga, Alytus, 2006. 33-36 p.
17. John Wiley & Sons, Inc., Hoboken, New Jersey. *Tools and environemnts for parallel and distributed computing.* John Wiley & Sons, Inc, 2004. 23-24 p.
18. Hiroshi Tamura, Futoshi Tasaki, Masakazu Sengoku and Shoji Shinoda. *Scheduling Problems for a Class of Parallel Distributed Systems.* IEEE, 2005.
19. Bruno R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*, Wiley, 1998. [žiūrėta 2006.05.20] http://www.brpreiss.com/books/opus4/html/page356.html.
20. Network Topologies, [žiūrėta 2006.05.21] http://www.webopedia.com/quick_ref/topologies.asp.
21. Keqin Li. *Fast and Scalable Parallel Matrix Computations on Distributed Memory Systems.* Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) 1530-2075/05 **IEEE, 2005.**
22. Jonathan M. Smith and David J. Farber. *Traffic Characteristics of a Distributed Memory System.* Parallel and Distributed Computing, vol. 15, 2006. 110-116 p.
23. Kontrimas R., Bulbenkienė V., Andziulis A. *Išskirstytos sistemos taikymas.* Informacinės technologijos 2006: studijų plėtra Lietuvos ekonominių bei visuomenės pokyčių kontekste. V mokslinės praktinės konferencijos pranešimų medžiaga, Alytus, 2006. 33-36 p.
24. Jean Bacon, Tim Harris Operating Systems: *Concurrent and Distributed Software Design.* IEEE Trans. Parallel and Distributed Systems, vol. 10, 2003. 234-241 p.
25. Niraj K. Jha. *Safety and Reliability Driven Task Allocation in Distributed Systems*. Parallel and Distributed Computing, vol. 12, 2003. 212-219 p.
26. G.C. Sih and E.A. Lee. *Declustering: A New Multiprocessor Scheduling Technique*. IEEE Trans. Parallel and Distributed Systems, vol. 4, no. 6, 1998. 625-637 p.
27. G. Cybenko. *Dynamic Load Balancing for Distributed Memory Multiprocessors*. Parallel and Distributed Computing, vol. 7, 1999. 279-301 p.
28. Keith D. Underwood, K. Scott Hemmert, Arun Rodrigues, Richard Murphy, and Ron Brightwell. *A Hardware Acceleration Unit for MPI Queue Processing.* Proceedings of the

19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) 1530-2075/05, IEEE, 2005.

29. Message Passing Interface Forum. *MPI: A message-passing interface standard.* The International Journal of Supercomputer Applications and High Performance Computing, 8, 1994.
30. Thomas M. Parks. *A Comparison of MPI and Process Networks.* Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) 1530-2075/05, IEEE, 2005.
31. The message passing interface. [žiūrėta 2006.05.02] http://www-unix.mcs.anl.gov/mpi/.
32. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. *A High-performance, Portable Implementation of the MPI Message Passing Interface Standard.* Parallel Computing, V01.22, No.6, 2005. 789-828 p.
33. ***G.*** Burns, R. Daoud, and J. Vaigl. *LAM: An Open Cluster Environment fur MPI.* Roc. of Supercomputing Symposium, 1994. pp.379-386 [žiūrėta 2006.05.26] http://www.lam-mpi.org.
34. Snir, M., Otto, S., Huss-Ledemann, S., Walker, D., and Dongarra, J. *MPI—The Complete Reference, Vol. 1: The MPI Core, 2nd edition*, Cambridge: MIT Press, 1998. [žiūrėta 2006.05.29] http://www.netlib.org/utk/papers/mpibook/mpi-book.html.
35. Angela C. Sodan. *Message-Passing and Shared-Data Programming Models—Wish vs. Reality***.** Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05) 1550-5243/05 IEEE, 2005.
36. Steve McFee, Qingying Wu, Mark Dorica, and Dennis Giannacopoulos. *Parallel and Distributed Processing for h-p Adaptive Finite-Element Analysis: A Comparison of Simulated and Empirical Studies.* IEEE, 2006, 45-53 p.
37. Motohiko Matsuda, Tomohiro Kudoh, Hiroshi Tazuka. *The Design and Implementation of an Asynchronous Communication Mechanism for the MPI Communication Model*. IEEE, 2004. 14-22 p.
38. Dr. Syed Jamal Hussain, Ghufran Ahmed. *A Comparative Study and Analysis of PVM and MPI for Parallel and Distributed Systems.* IEEE, 2005. 183-187 p.
39. Wilfried N. Gansterer and Joachim Zottl. *Message passing vs. Virtual shared memory. A perfomance compariso.* Distributed and parallel systems cluster and grid computing, 2005. 41-46 p.
40. Message Passing Interface Forum (2006). *MPI-2: Extensions to the message-passing interface Standard* [žiūrėta 2006.05.06].

41. Yuichi Tsujita. *MPI-I/O WITH A SHARED FILE POINTER USING A PARALLEL VIRTUAL FILE SYSTEM IN REMOTE I/O OPERATIONS.* DISTRIBUTED AND PARALLEL SYSTEMS CLUSTER AND GRID COMPUTING, 2005. 47-55 p.
42. Kuan-Ching Hsun-Chang Chang' Chao-Tung Yan Chung-Yuan Yin-Yi Mao-Yueh Pel Liria Matsumoto Hsiang. *On Construction of a Visualization Toolkit for MPI Parallel Programs in Cluster Environments.* Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05) 1550-445X/05, IEEE, 2005.
43. Hyun-Wook Jin Sayantan Sur Lei Chai Dhabaleswar K. Panda. *LiMIC: Support for High-Performance MPI Intra-Node Communication on Linux Cluster.* Proceedings of the 2005 International Conference on Parallel Processing (ICPP'05) 0190-3918/05, IEEE, 2005.
44. Daesuk Kwon and Sangyong Han, Heunghwan Kim. *MPI Backend for an Automatic Parallelizing Compiler.* Proceedings of the 2005 International Conference on Parallel Processing (ICPP'05) 0190-3918/05, IEEE, 2005. 425-436 p.
45. Stewart, R. R., and Xie, Q. *Stream control transmission protocol (SCTP): a reference guide.* Addison-Wesley Longman Publishing Co., Inc., 2002. 120-136 p.
46. Humaira Kamal, Brad Penoff, Alan Wagner. *SCTP versus TCP for MPI.* Proceedings of the 2005 ACM/IEEE SC|05 Conference (SC'05)1-59593-061-2/05, IEEE 2005. 23-34 p.
47. Ron Brightwell, Sue Goudy, Keith Underwood. *A Preliminary Analysis of the MPI Queue Characterisitics of Several Applications.* Proceedings of the 2005 International Conference on Parallel Processing (ICPP'05)0190-3918/05, IEEE, 2005. 208-216 p.
48. Ni, L. M. and P. K. McKinley. *A survey of routing techniques in wormhole networks*. *IEEE Computers*, 26, 2, Feb. 2003. 62-76 p.
49. Behrooz Parhami. *Introduction to Parallel Processing Algorithms and Architectures*. First Edition 1999 Plenum Press P15 – 1.4 Types of Parallelism: Taxonomy P78 – Table 4.2 Topological Parameters of Selected Interconnection Newtorks.
50. Oficiali ANSYS interneto svetainė – www.ansys.com [žiūrėta 2006.06.02].
51. McGraw Hill, Kai Hwang and Fayě A. Briggs. *Computer Architecture and Parallel Processing.* International Edition 2003.
52. Behrooz Parhami. *Introduction to Parallel Processing.* Algorithms and Architectures, First Edition 1999.
53. Dan I. Moldovan. *Parallel Processing.* From Applications to Systems. First Edition 1993.

TERMINŲ IR SANTRUMPŲ ŽODYNĖLIS

DDS – Distributed Domain Solver;

PVM – Private Virtual Machine;

DACS – Digital Access Cross-connect System ;

DSM – Distributed Shared Memory;

FE – Finite Element;

PRAM - Parallel random access machine;

NIC – Network Interface Card.

PRIEDAI

PRIEDAS Nr. 1

Mechatronikos uždavinio sprendimo proceso techninių ir laiko komponenčių modeliavimas virtualiame blokinyje

```
function plots

y_60 =  [11.256163 2.97289 2.085062 1.533152 1.26716 0.996021 ];

y_120 = [45.810661 11.491478 7.839005 5.383871 4.187208 3.232426 ];

y_180 = [111.721949 26.280184 17.838946 12.176307 9.413069 7.149453 ];

y_240 = [205.516207 48.729024 32.982407 22.048036 17.014949 13.091097
];

y_300 = [324.565092 80.836117 52.27633 35.216773 27.207582 20.906174
];

y_360 = [469.384824 121.780736 79.046601 52.920637 39.762668 30.649895
];

y_420 = [640.612062 169.099137 111.880208 73.514546 55.915544
42.594644 ];

y_480 = [837.288399 222.474732 148.684467 98.82164 74.699427 56.220476
];

y_540 = [1058.459214 282.420194 189.511962 127.652788 96.424833
72.806032 ];

y_600 = [1303.937413 349.278586 234.46219 159.520055 121.226566
91.406496 ];

y_660 = [1572.888626 425.870722 283.411603 194.181991 148.103974
112.493721 ];

y_720 = [1864.402242 503.134313 336.78972 231.952082 176.332697
135.101946 ];

y_780 = [2175.61912 591.267074 394.989867 271.357013 206.680045
158.929052 ];

procs = [1 4 6 9 12 16];

ys = [ y_60 ; y_120 ; y_180 ; y_240 ; y_300 ; y_360 ; y_420 ; y_480 ;
y_540 ; y_600 ; y_660 ; y_720 ; y_780 ];
```

```
labels ={'g = 60', 'g = 120', 'g = 180', 'g = 240', 'g = 300', 'g =
360', 'g = 420', 'g = 480', 'g = 540', 'g = 600', 'g = 660', 'g =
720', 'g = 780'};

figure (1);

semilogy (procs, ys ,'o-');

box ('on');

grid ('on');

title ('Vykdymo laikas');

ylabel ('Laikas (s)');

xlabel ('Procesoriai');

legend ( labels );

plot_name = strcat (' execution_time_semilogy . eps'); % plot name

print (gcf ,'-depsc2', plot_name ); % print plot


figure (2);

plot (procs ,ys ,'o-');

box ('on');

grid ('on');

title ('Vykdymo laikas');

ylabel ('Laikas (s)');

xlabel ('Procesoriai');

legend ( labels );

plot_name = strcat (' execution_time_linear . eps'); % plot name

print (gcf ,'-depsc2', plot_name ); % print plot


y_60_s = speedup ( y_60 ); y_60_e = efficiency (y_60 , procs );

y_120_s = speedup ( y_120 ); y_120_e = efficiency (y_120 , procs );

y_180_s = speedup ( y_180 ); y_180_e = efficiency (y_180 , procs );

y_240_s = speedup ( y_240 ); y_240_e = efficiency (y_240 , procs );

y_300_s = speedup ( y_300 ); y_300_e = efficiency (y_300 , procs );
```

```
y_360_s = speedup ( y_360 ); y_360_e = efficiency (y_360 , procs );
y_420_s = speedup ( y_420 ); y_420_e = efficiency (y_420 , procs );
y_480_s = speedup ( y_480 ); y_480_e = efficiency (y_480 , procs );
y_540_s = speedup ( y_540 ); y_540_e = efficiency (y_540 , procs );
y_600_s = speedup ( y_600 ); y_600_e = efficiency (y_600 , procs );
y_660_s = speedup ( y_660 ); y_660_e = efficiency (y_660 , procs );
y_720_s = speedup ( y_720 ); y_720_e = efficiency (y_720 , procs );
y_780_s = speedup ( y_780 ); y_780_e = efficiency (y_780 , procs );

figure (3);
yss = [ y_60_s ; y_120_s ; y_180_s ; y_240_s ; y_300_s ; y_360_s ;
y_420_s ; y_480_s ; y_540_s ; y_600_s ; y_660_s ; y_720_s ; y_780_s ];
hold ('on')
plot (procs ,yss ,'o-');
plot (procs ,procs ,' -.');
box ('on');
grid ('on');
title ('Lygiagretaus vykdymo sparta');
ylabel ('Sparta, atitinkanti vieno procesoriaus (Ts/Tp)');
xlabel ('Procesoriai');
legend ( labels ,'Location','NorthWest');
plot_name = strcat ('speedup_4 . eps'); % plot name
print (gcf ,'-depsc2', plot_name ); % print plot

figure (4);
yss_e = [ y_60_e ; y_120_e ; y_180_e ; y_240_e ; y_300_e ; y_360_e ;
y_420_e ; y_480_e ; y_540_e ; y_600_e ; y_660_e ; y_720_e ; y_780_e ];
plot (procs ,yss_e ,'o-');
title ('Lygiagretaus vykdymo efektyvumas');
ylabel ('Efektyvumas (Ts /(p*Tp ))');
```

```
xlabel ('Procesoriai');
 % legend ( labels );
grid ;
plot_name = strcat (' parallel_efficiency . eps'); % plot name
print (gcf ,'-depsc2', plot_name ); % print plot
figure (5);

subplot (2 ,2 ,1);
semilogy (procs ,ys ,'o-');
title ('Vykdymo laikas');
ylabel ('Laikas (s)');
xlabel ('Procesoriai');
grid ;

subplot (2 ,2 ,2);
plot (procs ,ys ,'o-');
title ('Vykdymo laikas');
ylabel ('Laikas (s)');
xlabel ('Procesoriai');
grid ;

subplot (2 ,2 ,3);
hold ('on');
plot (procs ,yss ,'o-');
plot (procs ,procs ,' -.');
title ('Lygiagretaus vykdymo sparta');
ylabel ('Sparta, atitinkanti vieną procesorių');
xlabel ('Procesoriai');
hold ('off');
```

```
grid ('on');
box ('on');

subplot (2 ,2 ,4);
plot (procs ,yss_e ,'o-');
title ('Lygiagretaus vykdymo efektyvumas');
ylabel ('Efektyvumas');
xlabel ('Procesoriai');
grid ('on');
end

function e = efficiency (time , procs )
[x, y] = size ( time );
e = zeros (1, y);
for i =1: y
e(i) = time (1)/( procs (i) * time (i ));
end ;
end
function s = speedup ( time )
[x, y] = size ( time );
s = zeros (1, y);
for i =1: y
s(i) = time (1)/ time (i);
end ;
end
```

PRIEDAS Nr. 2.

Mechatronikos uždavinio sprendimui sukurtas ANSYS išeities kodas

```
function Data=Mast(C,N,MOD,ENC)
Data=[]; TAG=7; DEBUG=0;
                              MastFLG=0;
if nargin>1 & C<0, C=1;       MastFLG=1; end
if nargin<4, help(mfilename), return, end
flag=0;                                          CC=int2str(C);
flag=flag | fix(C)~=C | C<0;           NN=int2str(N);
flag=flag | fix(N)~=N | N<1;           MOD=lower(MOD);
flag=flag | isempty(findstr(MOD,'sr'));ENC=lower(ENC);
flag=flag | isempty(findstr(ENC,'dn'));
if flag, help(mfilename), return, end
if C
  [info flag]=MPI_Initialized;
  if info | ~flag
    RTFLAGS=RTF_HOMOG+RTF_MPIC2C*(ENC=='d');
    putenv(['TROLLIUSRTF=' int2str(RTFLAGS)]), MPI_Init;
  end
  RTFLAGS=str2num(getenv('TROLLIUSRTF'));
  if ~bitand(RTFLAGS,RTF_HOMOG)
          warning('use mpirun -O for homogeneous cluster!!!');
  end
  if ~bitand(RTFLAGS,RTF_MPIC2C)
    if ENC~='n', error('ENC parameter does not match'); end
  else
    if ENC~='d', error('ENC parameter does not match'); end
  end
  [info attr flag]=MPI_Attr_get(MPI_COMM_WORLD,MPI_UNIVERSE_SIZE);
  if  info  |  ~flag,  error('attribute  MPI_UNIVERSE_SIZE  does  not
exist?'), end
  if attr<=C, error(['need ' int2str(C+1) ' computers in LAM']), end
  if DEBUG
    DISP=getenv('DISPLAY');
    [info children errs]=MPI_Comm_spawn('/usr/X11R6/bin/xterm',...
                    {'-display' DISP '-e' 'matlab'},C,'NULL',0,'SELF');
  else
    [info children errs]=MPI_Comm_spawn('matlab',...
```

```
                                        {'-
nosplash'},C,'NULL',0,'SELF');
  end
  if info | any(errs),       error('cannot start MATLABs'), end
  [info numt] = MPI_Comm_remote_size(children);
  if info | numt~=C,         error('cannot start MATLABs'), end
  [info NEWORLD]=MPI_Intercomm_merge(children,0);
  if info,                   error('cannot start MATLABs'), end

  if DEBUG
    MPI_Errhandler_set(NEWORLD,MPI_ERRORS_RETURN);
  end
  if MastFLG                                  % Running Mast(0) in
1st slave
    FN=[ pwd '/tmp.mat'];
    cmd=['Data=Mast(0,' NN ',''' MOD ''','''' ENC ''''); save ' FN '
Data'];
    MPI_Bcast(cmd,0,NEWORLD);          % Pass command
    while ~exist(FN,'file'), pause(1), end
    load(FN), delete(FN)
  else
    cmd=['Work(' CC ',' NN ',''' MOD ''')'];
    MPI_Bcast(cmd,0,NEWORLD);          % Pass command
%   MPI_Barrier(NEWORLD);              % initial Synch
    NULL=[];
    for numt=1:C,  MPI_Recv(NULL,MPI_ANY_SOURCE,TAG,NEWORLD); end
    for numt=1:C,  MPI_Send(NULL,numt,          TAG,NEWORLD); end
    P2=0; Psum=0;
    switch(MOD)
    case 'r' %%%%%%%%%%%%
keyboard
          MPI_Reduce(P2,Psum,'SUM',0,NEWORLD);
    case 's'
    for     numt=1:C,          MPI_Recv(P2,MPI_ANY_SOURCE,TAG,NEWORLD);
Psum=Psum+P2; end
    end

    Psum     =Psum/N;
    Data.pi  =Psum;
    Data.err =Psum-pi;
  end
```

```
else
  width=1/N; Sum=0;
  i=0:N-1;                    % for i=0:N-1
  x=(i+0.5)*width;  %   x=(i+0.5)*width;
  Sum=sum(4./(1+x.^2));       %   Sum=Sum+4/(1+x^2);
  Sum      =Sum*width;
  Data.pi  =Sum;
  Data.err =Sum-pi;
end % if C
if abs(Data.err)>5E-10, warning('pretty nice pi value!'), end

FINISH
/CLEAR
/TITLE,pavadinimas
/PREP7
/FACET,NORML

BTOL,0.1e-10
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!Krumpliaratukas
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
~PARAIN,krumpliaratukas,x_t,,SOLIDS
VDELE,ALL
ADELE,ALL
LDELE,1,276,1
LDELE,413,414
KDELE,1,140
!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!
LDIV,296,0.5
LDIV,304,0.5
LDELE,1
LDELE,277,295
LDELE,304,412
KDELE,141,160
KDELE,169,276
LDELE,296
LDELE,2
```

```
LDELE,297,298
LDELE,302,303
KDELE,1,2
KDELE,161,162
KDELE,167,168
LGEN,1,ALL,,,,,0.110,,,1
K,1
K,2,0.2
K,3,,0.05
KWPLAN,11,1,2,3
CSYS,1
L,166,163
!-----------------------------
!----Cia nustatomas
!-------krumpliaratuko kontaktas
          LDIV,301,0.497
KDELE,2
KDELE,3
CSYS,0
KGEN,2,4,,,0.02
KGEN,2,4,,,,0.02
KWPLAN,12,4,2,3
CSWPLA,12,1
!-----------------------------
!----Kontakto plotis
          CYL4,,,0.00045
LSBL,2,5
LSBL,301,3
ADELE,ALL
LDELE,3,6,1,1
L,9,10
LDIV,3,0.08!!!!!!!!!!!!!!!!!
LDIV,4,0.92!!!!!!!!!!!!!!!!!
CSYS,0
L,5,6
LDELE,4
LDIV,6,0.5
L,4,7
!-----------------------------
```

```
!----Rutuliukas 1
          CYL4,,,0.00065
LSBL,8,13
LSBL,9,11
ADELE,ALL
LDELE,11,14,1,1
CSYS,12
L,14,15
LDIV,9,0.5
L,7,8
CSYS,12
LDIV,9,0.15
LDIV,11,0.85
CSYS,0
L,5,11
L,6,12
!-----------------------------
!----Rutuliukas 2
          CYL4,,,0.002
LSBL,16,22
LSBL,17,20
ADELE,ALL
LDELE,20,23,1,1
CSYS,12
L,19,20
LDIV,17,0.5
CSYS,0
L,8,13
!-----------------------------
!----Plotai
AL,3,7,10,6
AL,10,2,5,4
AL,15,3,18,9
AL,18,6,12,13
AL,12,4,19,11
AL,5,8,14,19
AL,24,9,13,21,17
AL,21,11,14,16,20
AL,17,20,26,300,299,1,25
```

```
!---------------------------
!----Turiai
VOFFST,1,0.220
VOFFST,2,0.220
VOFFST,3,0.220
VOFFST,4,0.220
VOFFST,5,0.220
VOFFST,6,-0.220
VOFFST,7,0.220
VOFFST,8,0.220
VOFFST,9,-0.220
NUMMRG,ALL,1e-10
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
~PARAIN,krumpliaratis,x_t,,SOLIDS
K,1030,1.320
K,1040,1.4
K,1050,1.320,0.1
CSYS,0
VGEN,1,10,,,1.320,,,,,1
CSKP,13,1,1030,1040,1050
!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!KEISTI
          VGEN,1,10,,,,1.56483,,,,1 !Kr.pasukimas
VSEL,S,VOLU,,10
ASLV,S,1
LSLA,S,1
KSLL,S,1
VDELE,ALL
ADELE,ALL
LDELE,415,419,1,1
LDELE,421
LDELE,423,438,1,1
!---------------------------
!----Cia nustatomas
!-------krumpliaracio kontaktas
          LDIV,422,0.5601
```

```
CSYS,0
KGEN,2,22,,,0.02
KGEN,2,22,,,,0.02
KWPLAN,14,22,23,26
CSWPLA,14,1
!-----------------------------
!----Kontakto plotis
          CYL4,,,0.00045
LSBL,422,44
LSBL,33,37
ADELE,ALL
LDELE,37,38,1,1
LDELE,44,45,1,1
L,34,35
LDIV,33,0.08!!!!!!!!!!!!!!!!
LDIV,37,0.92!!!!!!!!!!!!!!!!
LDELE,37
CSYS,0
L,29,30
LDIV,37,0.5
L,22,31
!-----------------------------
!----Rutuliukas 1
          CYL4,,,0.00065
LSBL,49,57
LSBL,53,54
ADELE,ALL
LDELE,54,60,6,1
LDELE,56,57,1,1
CSYS,14
L,41,42
LDIV,53,0.5
LDIV,53,0.15
LDIV,54,0.85
CSYS,0
L,37,29
L,30,38
L,31,33
!-----------------------------
```

```
!----Rutuliukas 2
          CYL4,,,0.002
LSBL,62,72
LSBL,64,69
ADELE,ALL
LDELE,69,70,1,1
LDELE,72,76,4,1
CSYS,14
L,48,50
LDIV,64,0.5
CSYS,0
L,33,39
CSYS,13
L,280,279
L,278,281
!----------------------------
!----Plotai
AL,33,37,45,48
AL,45,44,38,52
AL,33,61,53,60
AL,60,56,68,37
AL,68,54,65,44
AL,65,57,49,38
AL,53,56,70,64,77
AL,54,70,69,62,57
AL,69,64,78,72,420,76,81
VOFFST,16,-0.200
VOFFST,26,-0.200
VOFFST,29,0.200
VOFFST,31,0.200
VOFFST,34,-0.200
VOFFST,39,-0.200
VOFFST,44,0.200
VOFFST,47,0.200
VOFFST,51,0.200
NUMMRG,ALL,1e-10
ALLSEL
```

```
!/pnum,line,1
!/pnum,kp,1
lplot
WPSTYLE,,,,,,,,0
!!!!!!!!!!!!!!!!krumpliaratukas
MP,EX,1,2e11
MP,NUXY,1,0.3
MP,DENS,1,7800
!!!!!!!!!!!!!!!!krumpliaratis
MP,EX,2,2e11
MP,NUXY,2,0.3
MP,DENS,2,7800

ET,1,SOLID95
ET,2,TARGE170
ET,3,CONTA174
KEYOPT,3,5,3


TYPE,1
MAT,1
REAL,1
MSHAPE,0,3D
MSHKEY,1
LESIZE,7,,,4
LESIZE,2,,,4
LESIZE,10,,,1
LESIZE,32,,,1200
          VMESH,1,2

TYPE,1
MAT,1
REAL,1
MSHAPE,1,3D
MSHKEY,0
!---------------3
LESIZE,15,,,1
LESIZE,41,,,1
LESIZE,9,,,1
```

```
LESIZE,42,,,1
LESIZE,18,,,1
LESIZE,43,,,1
LESIZE,46,,,750
LESIZE,47,,,750
!---------------4
LESIZE,12,,,3
LESIZE,51,,,3
LESIZE,13,,,3
LESIZE,50,,,3
LESIZE,55,,,750
!---------------5
LESIZE,11,,,3
LESIZE,58,,,3
!---------------6
LESIZE,8,,,1
LESIZE,66,,,1
LESIZE,19,,,1
LESIZE,59,,,1
LESIZE,14,,,1
LESIZE,67,,,1
LESIZE,71,,,750
LESIZE,63,,,750
          VMESH,3,6
!---------------7
LESIZE,24,,,3
LESIZE,73,,,3
LESIZE,21,,,3
LESIZE,75,,,3
LESIZE,17,,,4
LESIZE,74,,,4
LESIZE,79,,,480
LESIZE,80,,,430
!---------------8
LESIZE,85,,,3
LESIZE,16,,,3
LESIZE,84,,,4
LESIZE,20,,,4
LESIZE,90,,,480
```

```
          VMESH,7,8
!---------------9
LESIZE,25,,,4
LESIZE,99,,,4
LESIZE,26,,,4
LESIZE,95,,,4
LESIZE,299,,,4
LESIZE,97,,,4
LESIZE,1,,,4
LESIZE,98,,,4
LESIZE,300,,,3
LESIZE,96,,,3
LESIZE,106,,,20
LESIZE,105,,,20
LESIZE,103,,,20
LESIZE,104,,,40
          VMESH,9
TYPE,1
MAT,2
REAL,1
MSHAPE,0,3D
MSHKEY,1
LESIZE,48,,,4
LESIZE,52,,,4
LESIZE,45,,,1
LESIZE,92,,,1018
          VMESH,10,11
TYPE,1
MAT,2
REAL,1
MSHAPE,1,3D
MSHKEY,0
!---------------12
LESIZE,61,,,1
LESIZE,113,,,1
LESIZE,60,,,1
LESIZE,111,,,1
LESIZE,53,,,1
LESIZE,112,,,1
```

```
LESIZE,116,,,700
LESIZE,117,,,700
!---------------13
LESIZE,121,,,3
LESIZE,56,,,3
LESIZE,68,,,4
LESIZE,120,,,4
LESIZE,125,,,700
!---------------14
LESIZE,54,,,3
LESIZE,127,,,3
!---------------15
LESIZE,49,,,1
LESIZE,57,,,1
LESIZE,65,,,1
LESIZE,136,,,1
LESIZE,128,,,1
LESIZE,135,,,1
LESIZE,140,,,700
LESIZE,132,,,700
          VMESH,12
          VMESH,15
          VMESH,13
          VMESH,14
!---------------16
LESIZE,64,,,4
LESIZE,145,,,4
LESIZE,70,,,3
LESIZE,144,,,3
LESIZE,77,,,3
LESIZE,146,,,3
LESIZE,151,,,490
LESIZE,150,,,410
!---------------17
LESIZE,62,,,3
LESIZE,154,,,3
LESIZE,69,,,4
LESIZE,155,,,4
LESIZE,160,,,490
```

```
          VMESH,16
          VMESH,17
!---------------18
LESIZE,78,,,4
LESIZE,167,,,4
LESIZE,81,,,4
LESIZE,163,,,4
LESIZE,76,,,4
LESIZE,164,,,4
LESIZE,420,,,4
LESIZE,165,,,4
LESIZE,72,,,2
LESIZE,166,,,2
LESIZE,174,,,20
LESIZE,173,,,20
LESIZE,172,,,20
LESIZE,171,,,20
          VMESH,18
/FACET,NORML
 /ZOOM,1,SCRN,0.333373,0.002756,0.356651,-0.043788
 /DIST,1,0.924021086472,1
 /REP,FAST
 /DIST,1,0.924021086472,1
 /REP,FAST
 /DIST,1,0.924021086472,1
 /REP,FAST
 /ZOOM,1,SCRN,0.429810,0.022703,0.622684,-0.246588
 /FOC,   1,  0.175089062395    , -0.137736959071E-01,  0.100000000000
 /REPLO
 /DIST,1,0.924021086472,1
 /REP,FAST
 /DIST,1,0.924021086472,1
 /REP,FAST
 /DIST,1,0.924021086472,1
 /REP,FAST
 /DIST,1,0.924021086472,1
 /REP,FAST
 /FOC,   1,  0.174993360196    , -0.138269911436E-01,  0.100000000000
 /REPLO
```

```
vplot
!!!!!!!!!!!!Kontaktai
!!!CONTA
TYPE,3
MAT,1
ASEL,S,AREA,,14
ASEL,A,AREA,,19
NSLA,S,1
ESURF
ALLSEL
!!!!!!!!!!!!Kontaktai
!!!TARGE
TYPE,2
MAT,2
ASEL,S,AREA,,62
ASEL,A,AREA,,67
NSLA,S,1
ESURF
ALLSEL
!!!!!!!!!!!!!ITVIRTINIMAI
ASEL,S,AREA,,58
NSLA,S,1
CSYS,1
NROTAT,ALL
D,ALL,UX,0
CSYS,0
D,ALL,UZ,0
ALLSEL
!!!!!!!!!!!!!!!!!!!Jegos
ASEL,S,AREA,,58
NSLA,S,1
CSYS,1
F,ALL,FY,(-9673/0.3194)/(20*14+5)
CSYS,0
ALLSEL
ASEL,S,AREA,,103
NSLA,S,1
D,ALL,ALL,0
ALLSEL
```

```
FINISH
/SOLU
ANTYPE,STATIC
NLGEOM,ON
SOLVE
FINISH
/POST1
SAVE,KR_sudetingas,db
```

PRIEDAS Nr. 3

Spausdintos publikacijos ir konferencijos medžiaga