KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Vytenis Šeinauskas

Lygiagrečių programų efektyvumo tyrimas

Magistro darbas

Darbo vadovas
Doc. Dr. Romas Marcinkevičius

Kaunas, 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Vytenis Šeinauskas

Lygiagrečių programų efektyvumo tyrimas

Magistro darbas

Recenzentas
dr. K.Paulikas
2008.05.26

Vadovas
doc. dr. R.Marcinkevičius
2008.05.26

Atliko
IFM-2/2 gr. stud.
Vytenis Šeinauskas
2008.05.26

Darbo vadovas
Doc. Dr. Romas Marcinkevičius

Kaunas, 2008

Turinys

Efficiency Analysis of Parallel Programs

Summary

Parallel program execution is often used to overcome the constraints of processing speed and memory size when executing complex and time-consuming algorithms. The downside to this approach is the increased overall complexity of programs and their implementations. Parallel execution introduces a new class of software bugs and performance shortcomings, that are usually difficult to trace using traditional methods and tools. Hence, new tools and methods need to be introduced, which deal specifically with problems encountered in parallel programs.

The goal of this project is the development of MPI-based parallel program performance monitoring tool and research into the ways this tool can be used for measuring, comparing and improving the performance of target programs.

Pratarmė

Šis magistrinis darbas skirtas lygiagrečių programų efektyvumo analizei atlikti, pasinaudojant sukurta lygiagrečių programų efektyvumo tyrimo programine įranga. Pagrindinis darbo tikslas – sukurti, ištirti bei pritaikyti mokymo programinę įrangą, skirtą lygiagrečių programų analizei. Tam tikslui buvo atliekamas sukurtos programos galimybių tyrimas bei suplanuoti ir vykdomi programinės įrangos tobulinimo darbai. Taip pat buvo atliekami pavyzdinių lygiagrečių programų tyrimai, naudojant sukurtą programinę įrangą, norint parodyti lygiagrečių programų efektyvumo tyrimo būdus bei sukurtos lygiagrečių programų efektyvumo tyrimo programinės įrangos galimybes.

1.1. Įvadas

Pradėjus kurti lygiagrečias programas buvo susidurta su sunkumais vertinant jų veikimo efektyvumą. Įrankiai ir parametrai, naudoti nuoseklių programų efektyvumui tirti pasirodė besą netinkami įvertinti lygiagrečiųjų programų veikimą. Lygiagrečiosios programos veikimą yra žymiai sudėtingiau įvertinti negu paprastų nuosekliųjų programų. Atsiranda nauji faktoriai, tokie kaip ryšiai tarp procesorių, sinchronizacija, duomenų dalinimasis, kritinės sekcijos ir kiti. Todėl yra kuriama specializuota programinė įranga lygiagrečiųjų programų veikimo efektyvumui vertinti ir atvaizduoti. Tokia įranga dažnai yra sudėtingesnė negu nuoseklių programų efektyvumo tyrimo įranga dėl didesnio efektyvumo parametrų skaičiaus, problemų kylančių vertinant, matuojant ir skaičiuojant šiuos parametrus. Surinktų duomenų atvaizdavimas vartotojui patogia ir suprantama forma taip pat dažnai tampa problema. [1]

1.2. Lygiagrečių programų efektyvumo parametrai

Lygiagrečių programų efektyvumas, bei metodai jam nustatyti dažnai skiriasi nuo metodų naudojamų nuoseklioms programoms. Reikia vertinti ne tik skaičiavimų atlikimo laikus, bet ir tarp procesų vykstančią komunikaciją bei su tuo susijusias laiko sąnaudas. Patys paprasčiausi lygiagrečios programos efektyvumo parametrai - tai statinės ją apibūdinančios savybės, tokios kaip skaičiavime dalyvaujančių procesų kiekis, bendras vykdymo laikas ir pan. Siūlomi tokie programos efektyvumo įvertinimo parametrai [2].

Np – Procesų kiekis

T – Vykdymo laikas

Tw – Laukimo laikas

Tcpu – procesoriaus laikas. Procesoriaus laikas parodo kiek laiko procesorius vykdė skaičiavimus.

Twait_cpu – procesoriaus laukimo laikas

R = T/Tcpu. Parodo santykį tarp programos vykdymo laiko ir išnaudoto procesoriaus laiko. Lygiagrečių programų atveju šis dydis < 1, nes bendras programos vykdymo laikas

būna mažesnis už sudėtą visų procesorių veikimo laiką, kai programa vykdoma lygiagrečiai ir procesorių skaičius > 1. Atvirkščias dydis paralelizmui.
P = Tcpu/T – lygiagretumo laipsnis
Mb – pranešimų siuntimo greitis (baitais/s)
Mm – pranešimų siuntimo greitis (pranešimai/s)
Nm – Kompiuterių kiekis
L = (Tcpu + Twait_cpu) / Tcpu - parodo kiek programos vykdymas pailgėja dėl jos išlygiagretinimo.

Thomas Fahringer [3] siūlo metodą programos efektyvumo numatymui, nagrinėjant tam tikrus programos parametrus jau kompiliavimo metu. Siūloma nagrinėti tokius parametrus:
Darbo pasiskirstymas – numato, kaip tolygiai užduotys yra paskirstomos tarp procesorių.
Perdavimų kiekis – numatomas duomenų perdavimų kiekis programoje. Perdavimas atitinka vieną pranešimą siunčiamą iš vieno procesoriaus į kitą.
Perduotų duomenų kiekis – numatomas perduodamų tarp procesorių duomenų kiekis (baitais)
Perdavimo laikas – numatomas laikas perduoti pranešimus tarp procesorių.
Skaičiavimo laikas – numatomas laikas, kurio prireiks visiems nuosekliems skaičiavimams programoje. Į šį laiką neįtraukiamas perdavimo laikas.

M. Crovella ir T. LeBlanc siūlo programos efektyvumą nustatyti įvertinant laiką praleistą nenaudingai, t.y neatliekant skaičiavimų [4] . Siūlomas būdas remiasi teisingai parinktais nenaudingo darbo matavimo parametrais:

Apkrovos nesubalansavimas – tušti procesoriaus ciklai, atsirandantys dėl to, kad yra nebaigtų lygiagrečių užduočių.
Nepakankamas lygiagretumas – tušti procesoriaus ciklai, atsirandantys kuomet nėra neužbaigtų lygiagrečių užduočių.

Sinchronizacijos vėlinimas – laikas, praleistas sinchronizuojant skirtingus procesus (lock, barrier).
Ryšių vėlinimas – laikas praleistas laukiant, kol duomenys keliauja sistemoje.
Resursų užimtumas – laikas praleistas laukiant priėjimo prie bendrų aparatinių resursų.

Akivaizdu, kad vien bendra informacija apie programos trukmę, ar naudingai praleistą laiko dalį negali suteikti daug naudingos informacijos apie naudojamo algoritmo bei programos lygiagretinimo efektyvumą. Norint nustatyti programos lygiagretinimo efektyvumą skaičiuojamas programos pagreitėjimas kuomet naudojama n procesorių, lyginant su 1 procesoriumi.

Pagreitėjimas
T(p) – laikas, per kurį programa įvykdoma naudojant p procesorių
p – procesorių kiekis

$$S_p = \frac{T(1)}{T(p)}$$

Idealiu atveju $S_p = p$. Tokiu atveju, padidinus procesorių skaičių 2 kartus, programos vykdymo laikas sumažėja 2 kartus. Praktikoje toks rezultatas beveik neįmanomas.

Programos lygiagretinimo efektyvumas apskaičiuojamas taip

$$E_p = \frac{S_p}{p}$$

Tai yra skaičius intervale [0;1]. Šis parametras, nepriklausomai nuo procesorių skaičiaus, parodo kaip efektyviai didėja uždavinio sprendimo greitis didinant procesorių kiekį.

Amdahl dėsnis [5]

Užduočių visiškai išlygiagretinti, kad visa programa galėtų būti atliekama lygiagrečiai praktiškai neįmanoma. Dalis užduoties turi būti atliekama nuosekliai. Be to, dalis laiko išnaudojama pačiam lygiagretinimui palaikyti (pranešimų perdavimo laikas, sinchronizacija). Paprastumo dėlei, kiekvieną programą galima dalinti į dvi dalis – lygiagrečią ir nuoseklią.

T_s – nuoseklios programos dalies vykdymo laikas

T_p – lygiagrečios programos dalies vykdymo laikas

Tokiu atveju programos pagreitėjimas gali būti užrašytas :

$$S_p(p) = \frac{T_s(1) + T_p(1)}{T_s + \frac{T_p}{p}}$$

Kur **p** yra procesorių kiekis. Iš šios išraiškos matome, kad nesvarbu, kiek didinsime procesorių kiekį, programos pagreitėjimas turi ribą, kuri priklauso nuo to, kokią programos vykdymo laiko dalį sudaro T_s

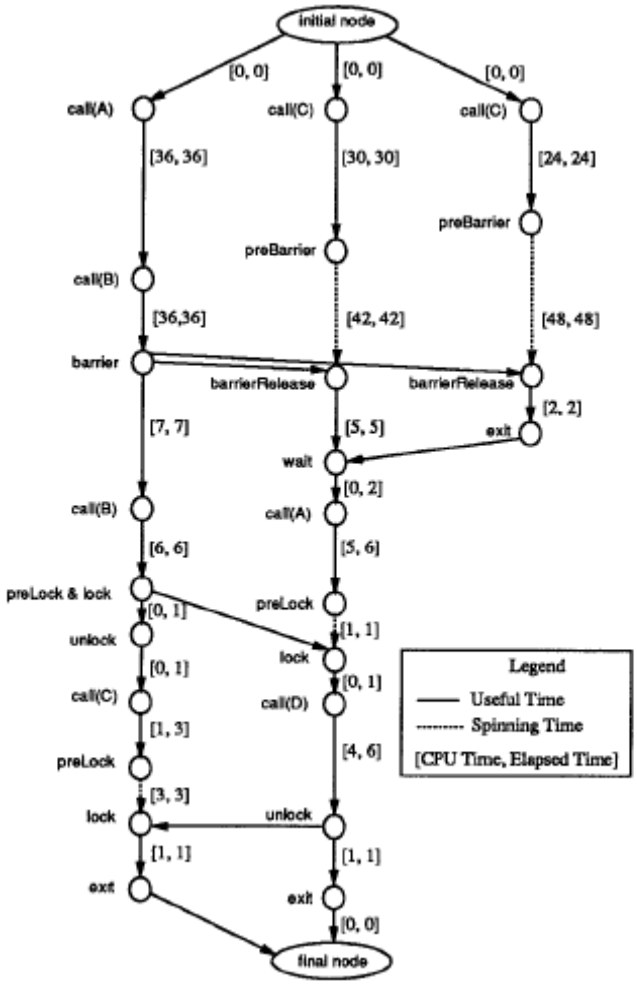
Aukščiau nurodyti programos veikimo parametrai parodo bendrą programos veikimo greitį, bet nenurodo, kur ieškoti būdų programą pagreitinti. Norint pagreitinti programos veikimą reikia žinoti, kurioje vietoje atlikti optimizacijas. Yra keletas būdų tai nustatyti. Vienas iš jų yra kritinio kelio radimas programos veiklos grafe. Programos veiklos grafas, kurio pavyzdys [6] parodomas žemiau, sudaromas iš viršūnių, kurios atitinka programos įvykius (procedūrų vykdymas, pranešimų siuntimas, gavimas ir pan). Grafo lankai parodo laiką, praleistą tarp įvykių, arba laiką praleistą vykdant sinchronizaciją (pranešimų siuntimas). Lygiagrečios programos kritinis kelias parodo, kurias programos vietas reikia tobulinti, o kurių pakeitimas neatneš bendro programos pagreitėjimo.

NPT metrika sudaroma nagrinėjant iš eilės kiekvieną programos veiklos grafo įvykį chronologine tvarka. Kiekvienam procesui, kuris vykdo naudingą darbą, jo trukmė duotu laiko intervalu yra dalinama iš skaičiaus procesorių, vykdančių naudingą darbą ir pridedama prie bendros trukmės. Tokiu būdu įvertinamas programos lygiagretumas

kiekvienu laiko momentu. Jei programos įvykiai rodo kreipinius į procedūras, šį metodą galima naudoti, norint nustatyti kiekvienos procedūros įtaką bendrai programos trukmei.

Leidžiamas vėlinimas - šis parametras parodo, kiek galima mažinti tam tikros procedūros, esančios kritiniame kelyje, trukmę, kol pasikeičia kritinis kelias. Kartais, nors procedūra yra kritiniame kelyje, mažinti jos trukmę neapsimoka, nes lygiagrečiai vykdoma kita procedūra, kuri trunka beveik tiek pat laiko, kaip ir pirmoji. Tokiu atveju, net ir nežymiai sumažinus procedūros trukmę, pasikeis programos kritinis kelias ir tolesnis mažinimas neduos laukiamo efekto.

Procedūros eliminavimas – Šis metodas parodo, kaip pasikeis programos trukmė, jei procedūrą pašalinsime iš programos, t.y jos trukmę prilygina 0. Šis parametras gali būti apskaičiuotas, programos veiklos grafe nustatant tam tikros procedūros lankų svorius į 0 ir perskaičiuojant kritinio kelio trukmę.



1 pav Programos veiklos grafo pavyzdys

1.3. Lygiagretus programavimas ir MPI

Egzistuoja du pagrindiniai lygiagretaus programavimo modeliai – bendros atminties ir pranešimų perdavimo. Pranešimų perdavimo modelis, kuris toliau nagrinėjamas šiame darbe, pagrįstas procesų su privačiais duomenimis aibe. Procesai tarpusavyje komunikuoja naudodami specialias siuntimo/gavimo operacijas. Šio modelio savybė yra tai, kad programos duomenys yra paskirstyti tarp procesų. Skaičiavimai taip pat turi būti

pritaikomi pranešimų perdavimo modeliui, atsižvelgiant į tai, kad procesoriai turi priėjimą tik prie dalies programos duomenų.
Bendros atminties modelis naudoja aibę gijų, kurios yra sukuriamos lygiagrečių užduočių vykdymui. Gijos dalijasi bendra adresų erdve, todėl kiekviena gija gali naudotis visais programos duomenimis. Atminties valdymas tokio tipo programose yra įgyvendinamas operacinės sistemos lygmenyje ir nereikalauja papildomo darbo, kaip pranešimų perdavimo atveju.

MPI buvo sukurtas siekiant standartizuoti pranešimų perdavimą lygiagrečiose programose. MPI standarte aprašytos funkcijos taškas-taškas, kolektyviniam, vienpusiam pranešimų perdavimui, taip pat lygiagrečiai duomenų įvesčiai/išvesčiai. Pagrindinės komunikavimo funkcijos buvo aprašytos 1994 metų pirmojoje MPI standarto versijoje, o kai kurios nuotolinio atminties valdymo bei lygiagrečios išvesties funkcijos buvo pridėtos tik naujesniame 2.0 MPI aprašyme [16].

1.3.1 Pagrindinės MPI funkcijos

MPI_Init – inicializuoja MPI biblioteką, turi būti kviečiama lygiagrečios programos pradžioje prieš bet kokią kitą MPI funkciją.
MPI_Finalize – atlaisvina MPI naudojamus resursus ir kviečiama programos pabaigoje.
MPI_Comm_size – skirta sužinoti kiek procesorių vykdo lygiagrečią programą.
MPI_Comm_rank – skirta sužinoti unikalų kviečiančio proceso identifikatorių.
MPI_Send – persiunčia pranešimą nurodytam procesui. Tai yra blokuojanti siuntimo operacija.
MPI_Recv – gauna pranešimą. Tai yra blokuojanti funkcija.

1.3.2 MPI Komunikatoriai

MPI funkcijos remiasi komunikatoriaus sąvoka. Komunikatorius susideda iš grupės procesų ir komunikavimo konteksto. Procesai komunikatoriuje turi nesikartojančius numerius, pagal kuriuos yra identifikuojami. Siunčiami pranešimai yra atskiriami pagal

siuntimo funkcijos perduodamą *tag* reikšmę, kuri padeda identifikuoti pranešimo paskirtį. Gavimo operacija atitinkamai naudoja tag reikšmę, kad filtruotų gaunamus pranešimus. Šios reikšmės galioja tik jeigu siuntimo/gavimo operacijos vyksta tame pačiame komunikavimo kontekste, kuris apsprendžiamas naudojamo komunikatoriaus.

1.3.3 Kolektyvinės operacijos

Svarbi MPI funkcijų grupė yra kolektyvinės operacijos, kurios yra atliekamos su visais tam tikro komunikatoriaus procesais. Tokiu atveju visi komunikatoriaus procesai vykdo tą pačią operaciją. Kolektyvinės operacijos pavyzdžiai galėtų būti:

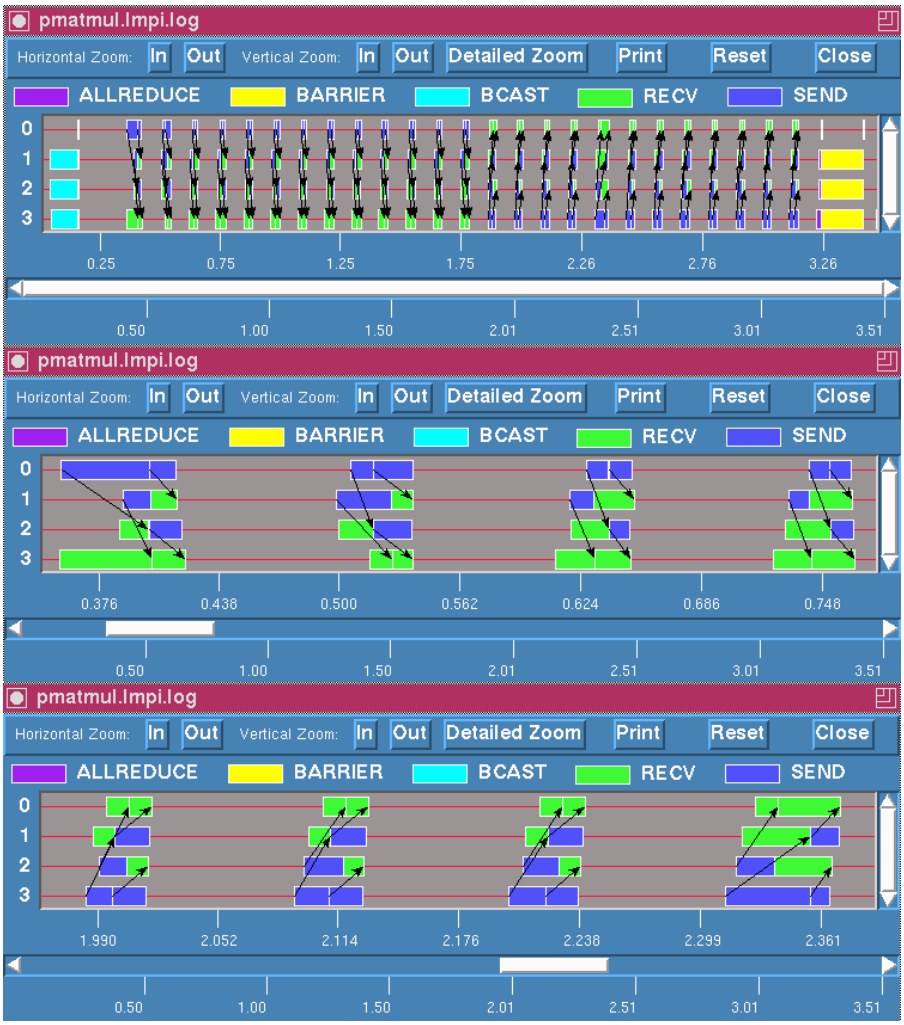
MPI_Reduce – atlieka globalią operaciją su kiekvieno komunikatoriaus proceso duomenimis (pvz., susumuoja kiekvieno proceso pateiktus skaičius).
MPI_Barrier – Sinchronizuoja visus komunikatoriaus procesus. Nė vienas procesas negali toliau vykdyti programos, kol visi procesai nepasiekė barjero.

1.4. Egzistuojantys sprendimai

Egzistuojantys sprendimai lygiagrečiųjų programų efektyvumo tyrimui dažniausiai skirti Unix šeimos operacinėms sistemoms ir skiriasi savo kaina bei siūlomomis galimybėmis.

LMPI - MPI programų profiliavimo biblioteka

Profiliavimo biblioteka. Naudojant šią biblioteką sukurtos programos veikdamos generuoja log failus, į kuriuos surašo savo veikimo informaciją. Šiuos failus galima atvaizduoti vizualiai, programai baigus darbą. Programos vizualizacijos lango pavyzdys:



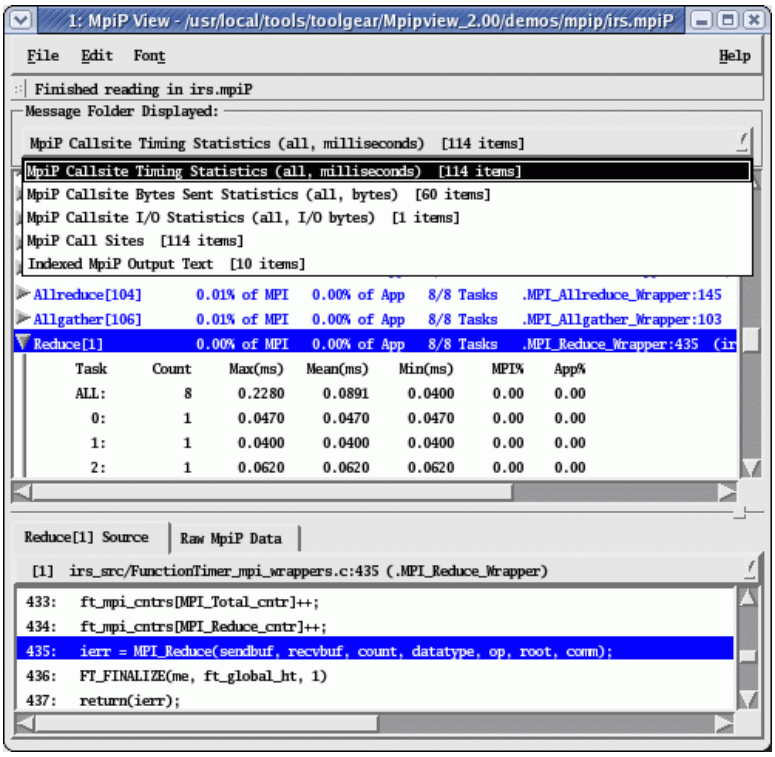
2 pav LMPI programos vaizdas

Galima sužinoti kiek laiko kiekviename procese užima MPI operacijos, taip pat matosi operacijų sukeltas vėlinimas. Galima vaizduoti norimas laiko atkarpas ir norimus procesus. Vaizdą galima didinti ir mažinti. Ši biblioteka nepalaiko realaus laiko programos veikimo analizės. [7]

mpiP – Atviro kodo MPI profiliavimo biblioteka

Atviro kodo MPI profiliavimo biblioteka. Ši biblioteka gali nurodyti kurioje vykdomosios programos vietoje yra MPI funkcijos ir kiek laiko jose programa užtrunka. Taip pat galima pažiūrėti, kiek kartų kviečiama kiekviena MPI funkcija, išsiunčiamų pranešimų dydžių statistiką. Biblioteka leidžia profiliuoti ne tik visą programą, bet ir

norimą jos dalį. mpiP neturi ataskaitų vizualizacijos funkcijos, bet yra sukurta programa mpiPView patogesniam ataskaitų peržiūrėjimui. [8]



3 pav mpiP programos vaizdas

MPICL – MPI komunikacijų biblioteka

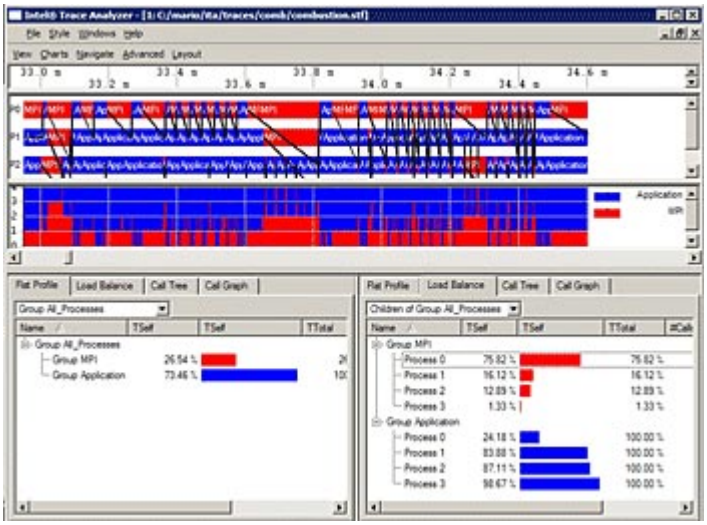
Biblioteka, naudojama MPI kodo profiliavimui. Gali surinkti bendrus programos veikimo parametrus (įvykių kiekis, bendras praleistas laikas MPI funkcijose, siunčiant pranešimus), arba dinaminius veikimo parametrus, kurie gali būti atvaizduoti naudojant ParaGraph vizualizacijos priemonę. Kaip ir dauguma analogiškų įrankių, MPICL generuoja įvykių žurnalą (log), kuris vėliau naudojamas programos veikimo analizei. ParaGraph vizualizacijos programa pateikia tris pagrindinius peržiūros režimus.

Panaudojimo režimas parodo informaciją apie kiekvieną procesą, nurodant kiek laiko praleidžiama komunikuojant ir skaičiuojant. Yra daug būdų vizualizuoti duomenis, pvz. grafikas rodantis laiką praleistą kiekvienoje būklėje (skaičiavimas, komunikavimas) procentaliai. Kitas režimas parodo komunikacijas, kurios vyksta tarp skirtingų procesorių, t.y kiek ir kokių pranešimų yra siunčiama. Taip pat rodomos pranešimų eilės kiekviename procesoriuje. Užduočių peržiūros režimas naudojamas surasti kuri kodo

vieta atsakinga už vėlinimus. Norint, kad šis režimas teisingai veiktų, reikia sukurti vartotojo stebėjimo taškus norimose vietose.[9]

Intel Trace Analyzer and Collector 6.0

Programinių įrankių rinkinys MPI programoms. Intel Trace Analyzer renka, grafiškai atvaizduoja MPI programų veikimo parametrus. Rodo laikinį grafiką, kvietimų grafą. Duomenis galima peržiūrėti klasterio, proceso, gijos ir funkcijos detalizacijos lygiuose. Yra API vartotojo stebėjimo taškams kurti ir atvaizduoti. Galima stebėti ir lyginti programos veikimo pokyčius pakeitus algoritmą. Yra galimybė stebėti daugelio gijų MPI procesus. Stebėjimo rezultatai išsaugomi ir nenumatytos programos pabaigos atveju. Palaiko MPI programas parašytas C,C++ ir Fortran kalbomis.[10]



4 pav Intel Trace Analyzer and Collector programos vaizdas

MPI efektyvumo tyrimo programų palyginimas

Programos buvo lyginamos pagal jų savybes ir parametrus. Lygiagrečių programų efektyvumo tyrimo priemonės turi palaikyti programavimo kalbas, kuriomis dažniausiai rašomos lygiagrečios programos, tai yra – pagrinde C/C++/Fortran. Kitas svarbus aspektas – galimybė vizualizuoti gautus rezultatus, todėl buvo lyginamos ir programų vizualizacijos galimybės.

Kartais nėra reikalo tirti visos programos, užtenka tik tam tikros problematiškos dalies tyrimo. Tokiu atveju praverčia galimybė tirti atskirus programos fragmentus.

Programos trasavimas leidžia tiksliau įvertinti programos veikimo parametrus vieno proceso ribose. Tokiu būdu galima gauti išsamesnę informaciją ne tik apie pranešimų siuntimą, bet ir apie programos vykdymą kiekviename procesoriuje.

Norint sužinoti ar atlikti pakeitimai turėjo įtakos programos darbui, reikia lyginti programos veikimą prieš ir po pakeitimų. Tokiu atveju praverčia patogi priemonė skirtingiems programos veikimo bandymams lyginti.

Kartais reikia ištirti specifinius tiriamai programai, arba vartotojo nurodytus įvykius. Ši galimybė turi būti numatyta lygiagrečių programų efektyvumo tyrimo programinėje įrangoje.

	LMPI	**mpiP**	**MPICL**	**ITAC***
C programų palaikymas	+	+	+	+
C++ palaikymas	-	-	-	+
Fortran palaikymas	+	+	+	+
Vizualizacijos priemonė	-	-	+	+
Programos fragmento tyrimas	-	+	+	+
Programos trasavimas	-	-	-	+
Kelių bandymų palyginimas	-	-	-	+
Vartotojo nurodytų įvykių tyrimas	-	-	+	+

*Intel Trace Analyzer and Collector

1.5. Įgyvendinimo problemos

Kuriant programinės įrangos efektyvumo tyrimo programinę įrangą, kylančios problemos gali būti skirstomos į dvi dalis :

1)Programos veikimo duomenų surinkimas.

3) Programos efektyvumo parametrų ir programos veikimo atvaizdavimas.

Duomenų surinkimas turi kiek galima mažiau paveikti stebimą programą, kad pati duomenų surinkimo veikla neiškreiptų programos veikimo vaizdo. [11] Esama keleto būdų sukurti stebėjimo taškus. Vienas jų yra naudoti specialią biblioteką, kuri leistų matyti tarp procesorių perduodamus pranešimus. Šis būdas nereikalauja naujo stebimos programos kompiliavimo. Šio būdo trūkumas tas, kad neįmanoma matyti specifinės programos veikimo informacijos (vieno proceso lygyje). Kitas galimas būdas – tiesiogiai įterpti stebėjimo taškus vartotojo programoje. Šis būdas reikalauja kompiliatoriaus modifikacijos (ne visada įmanoma, nes ne visi kompiliatoriai atvirojo kodo). Taip pat, kiekvieną kompiliatorių reikia pritaikyti atskirai. Stebėjimo taškus galima įterpti tiesiai į jau sukompiliuotą programą. Toks būdas yra sudėtingas įgyvendinti bei priklauso nuo naudojamos operacinės sistemos. Šio būdo pranašumas tas, kad jį galima naudoti nepriklausomai nuo programavimo kalbos, naudotos kuriant programą bei programos, kurių išeities tekstai nėra žinomi.

Paprasčiausias stebėjimo būdas – įterpti stebėjimo taškus programos išeities tekste. Tokiu būdu vartotojas pats nusprendžia ką nori stebėti ir kur bus stebėjimo taškai.

Vaizduojant lygiagrečių programų veikimo efektyvumą kyla problema, kaip informaciją pateikti suprantama vartotojui forma. [12] Paprasčiausi statiniai programos veikimo parametrai lengviausiai atvaizduojami, bet ir suteikia mažiausiai informacijos apie lygiagrečią programą. Vienas iš būdų atvaizduoti programos veikimą laike – Ganto grafikas, kuriame vaizduojamas kiekvieno procesoriaus užimtumas laiko intervale. [13]

Šis grafikas gali būti papildytas, atvaizduojant tarp procesų vykstančius ryšius, bei identifikuojant procesoriaus laiko panaudojimą.

Programos veikimą galima pavaizduoti ir naudojant grafą. Tokiu būdu, kiekviename procesoriuje skaičiavimo periodo pradžioje ir pabaigoje sukuriama po grafo viršūnę,

kurios sujungiamos briauna, vadinama lokalia briauna. Šios briaunos svoris lygus vykdyto skaičiavimo trukmės įvertinimui. Kuomet lygiagreti programa vykdo pranešimų perdavimą, jie gali būti atvaizduojami briaunomis tame pačiame grafe, kurios vadinamos pranešimų briaunomis. [14] Sudarytas grafas gali būti naudojamas efektyvumo skaičiavimuose, arba gali būti vizualizuojamas. Toks grafas vartotojui suteikia intuityviai suprantamą informaciją apie lygiagrečios programos veikimą.

1.6. Išvados

1. Egzistuojanti lygiagrečių programų efektyvumo tyrimo programinė įranga dažniausiai skirta kelių programos efektyvumo parametrų tyrimui. Norint visapusiškai ištirti lygiagrečios programos efektyvumą tenka naudoti kelis programinės įrangos paketus.
2. Egzistuojantys integruoti efektyvumo tyrimo paketai su patogia vartotojo sąsaja ir grafiniu tyrimo atvaizdavimu yra komerciniai ir menkai prieinami dėl didelės kainos.
3. Tikslinga kurti programinę įrangą, kuri leistų lygiagrečias programas vykdyti, redaguoti bei stebėti jų rezultatus bendroje integruotoje grafinėje aplinkoje.

2 Reikalavimų specifikacija

2.1 Sistemos paskirtis

Lygiagrečių programų kūrimą apsunkina sudėtingas šių programų efektyvumo nustatymas ir įvertinimas. Priemonių, skirtų tokių programų efektyvumo vertinimui nėra daug, ir jos dažnai yra brangios arba neefektyvios. Kuriama sistema padės rinkti duomenis apie lygiagrečių programų vykdymo efektyvumą bei juos įvertinti, atvaizduoti grafiškai. Sukūrus šią efektyvumo vertinimo priemonę pagreitės lygiagrečių programų kūrimas, derinimas, klaidų aptikimas bei palengvės programų kūrėjų darbas.

2.2 Projekto kūrimo pagrindas

Kuriama sistema reikalinga dėl nemokamos ir efektyvios analogiškos sistemos trūkumo. Lygiagrečių programų efektyvumo problema gali būti sprendžiama nenaudojant specializuotų priemonių , bet tokiu atveju ilgėja programos kūrimo laikas, be to reikalinga papildoma programos kūrėjų kvalifikacija. Norint efektyviai kurti lygiagrečias programas šiuo metu reikalingos nemažos investicijos į mokamas programinės įrangos efektyvumo vertinimo sistemas.

2.3 Sistemos tikslai

Sistema turi surinkti duomenis apie lygiagrečios programos veikimą, apskaičiuoti jos efektyvumą pagal nurodytas metrikas ir atvaizduoti surinktą informaciją bei išvestinius efektyvumo parametrus grafiškai. Programa turi turėti patogią vartotojo sąsają lygiagrečių programų paleidimui.

2.4 Užsakovai, pirkėjai ir kiti sistema suinteresuoti asmenys

Užsakovas – doc. Dr. Romas Marcinkevičius.

2.5 Vartotojai

Vartotojo kategorija - studentas

Vartotojo sprendžiami uždaviniai - kuriama lygiagreti programinė įranga laboratoriniams darbams

Patirtis dalykinėje srityje - naujokas

Patirtis informacinėse technologijose – informatikas

Vartotojo kategorija – programuotojas

Vartotojo sprendžiami uždaviniai – komercinės arba mokslinės lygiagrečios programinės įrangos kūrimas.

Patirtis dalykinėje srityje – srities specialistas.

Patirtis informacinėse technologijose – informatikas.

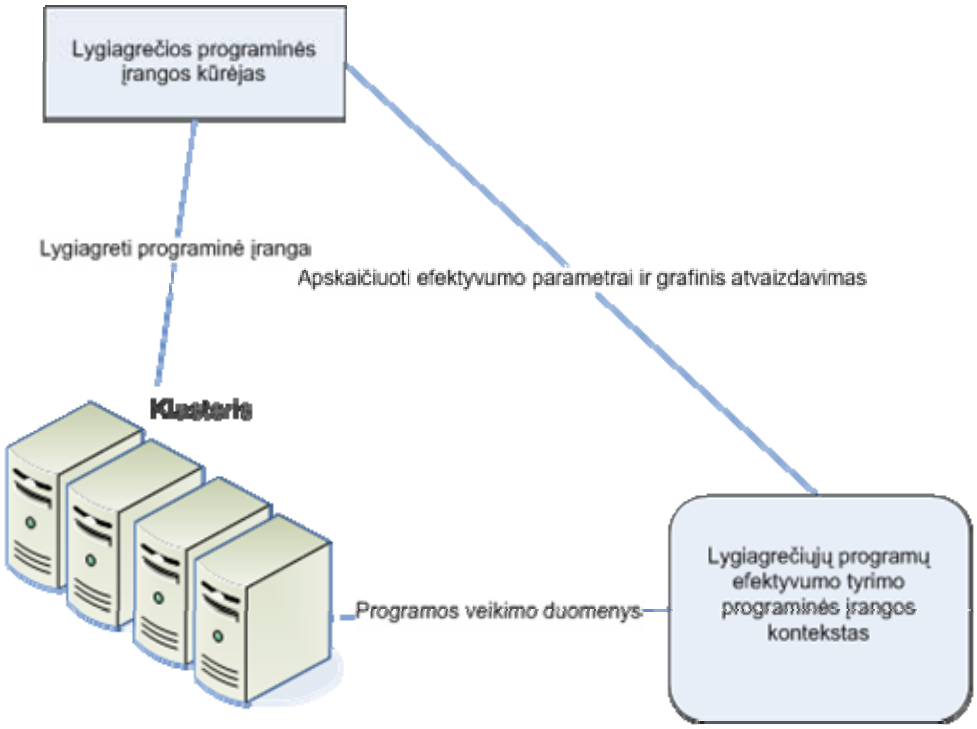
2.6 Projekto Apribojimai

Diegimo aplinka

Sistema veiks su MPI lygiagrečiomis programomis. Naudojama Linux operacinė sistema.

2.7 Funkciniai reikalavimai

2.7.1 Veiklos kontekstas



5 pav veiklos kontekstas

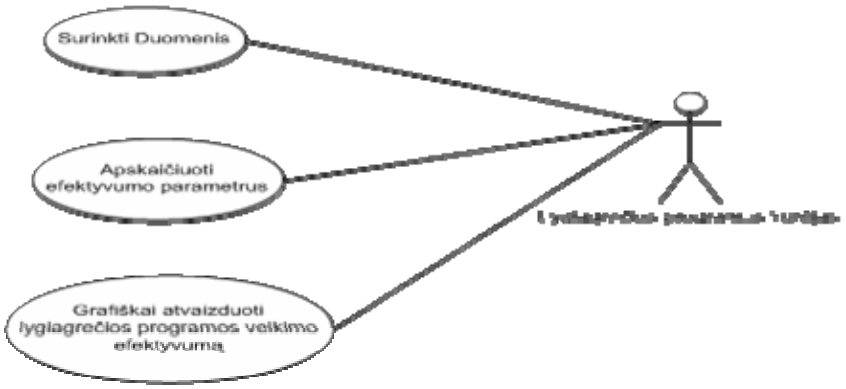
2.7.2 Veiklos padalinimas

Lentelė. Veiklos įvykių sąrašas

Eil. Nr.	*Įvykio pavadinimas*	*Įeinantys/Išeinantys informacijos srautai*
1	Vykdoma lygiagreti programinė įranga	Programos veikimo duomenys (in)
2	Baigiama vykdyti programinė įranga	
3	Suskaičiuojami efektyvumo parametrai	Programos efektyvumo parametrai (out), Grafinis parametrų atvaizdavimas (out)

2.8 Sistemos sudėtis

2.8.1 Sistemos ribos



6 pav Panaudos atvejai

2.9 Funkciniai reikalavimai

Reikalavimo nr.	Aprašymas	Pagrindimas
1	Sistemos vartotojo sąsaja turi būti pritaikyta lygiagrečios programinės įrangos paleidimui	Vartotojui patogiausia dirbti su integruota sąsaja, kurioje atliekami visi veiksmai nuo programos paleidimo iki rezultatų analizės
2	Sistema turi žurnale fiksuoti pranešimo šaltinį (procesorių) ir tikslą/tikslus.	Reikalinga efektyvumo parametrų skaičiavimui.
3	Sistema turi žurnale fiksuoti pranešimo išsiuntimo laiką, priėmimo laiką	Reikalinga efektyvumo parametrų skaičiavimui.
4	Sistema turi žurnale fiksuoti siunčiamo/gaunamo pranešimo dydį (baitais)	Reikalinga efektyvumo parametrų skaičiavimui.
5	Sistema turi žurnale fiksuoti statinius parametrus (procesų, procesorių kiekis, bendras vykdymo laikas)	Reikalinga efektyvumo parametrų skaičiavimui.
6	Sistema turi suskaičiuoti bendrus išvestinius lygiagrečios programos efektyvumo parametrus (bendras pranešimų srautas (b/s, ir praneš./s),	Šie parametrai leidžia spręsti apie analizuojamos programos efektyvumą.

	bendras laukimo laikas (% viso laiko), darbo pasiskirstymas, perduotų pranešimų kiekis, perduotų duomenų kiekis ir kt).	
7	Sistema turi sujungti atskirų procesorių žurnalinius failus į vieną bendra failą.	Norint analizuoti programos efektyvumą, reikalingas bendras žurnalinis failas
8	Sistema turi turėti sąsają su MPI biblioteka informacijos rinkimui.	Sistema turi dirbti su MPI biblioteka
9	Grafiškai atvaizduojamos komunikacijos tarp lygiagrečią programinę įrangą vykdančių procesorių	Atvaizdavus komunikacijas grafiškai lengva pastebėti komunikacijų vėlinimą, programos sulėtėjimo priežastis.
10	Grafiškai atvaizduojamos komunikacijos tarp lygiagrečią programinę įrangą vykdančių procesorių	Atvaizdavus komunikacijas grafiškai lengva pastebėti komunikacijų vėlinimą, programos sulėtėjimo priežastis.
11	Grafiškai atvaizduojamas vėlinimas dėl sinchronizacijos, resursų užimtumas, ryšių vėlinimas.	Pažiūrėjus į grafiką, lengva nustatyti vietą, kurioje galima taikyti papildomą optimizaciją dėl vėlinimo/ resursų užimtumo.

2.10 Nefunkciniai reikalavimai

2.10.1 Reikalavimai sistemos išvaizdai

Sistema turi turėti patogią vartotojo sąsają. Sąsają naudos programų kūrėjai, todėl numatoma vartotojo kvalifikacija yra aukšta ir vartotojo sąsajai nėra ypatingų paprastumo reikalavimų.

2.10.2 Reikalavimai panaudojamumui

Reikalingas sąsajos kalbų palaikymas. Panaudojimas turi būti suprantamas žmogui, susipažinusiam su lygiagrečių programų kūrimu.

2.10.3 Reikalavimai vykdymo charakteristikoms

Efektyvumo parametrų skaičiavimo laikas priklauso nuo lygiagrečios programos vykdymo laiko ir jos sudėtingumo, naudojamo procesorių kiekio. Paprastų programų efektyvumo tyrimas neturėtų užimti ilgiau negu 10 sekundžių.

2.10.4 Reikalavimai sistemos priežiūrai

Sistema turi būti sukurta taip, kad būtų galima pridėti papildomus programų veikimo parametrus, arba modifikuoti esamus.

2.11 Projekto išeiga

2.11.1 Atviri klausimai

Lygiagrečių programų veikimo parametrų parinkimas bei jų atvaizdavimas turi būti galutinai suderinti jau bandant sistemą, nes tik bandymu metu galima nustatyti tinkamiausius sistemos parametrus.

2.11.2 Uždaviniai

2.11.2.1 Sistemos pateikimo žingsniai

Sistema kuriama etapais:

1) Duomenų surinkimo modulis
2) Parametrų skaičiavimas
3) Parametrų atvaizdavimas, vartotojo sąsaja

2.11.3 Pritaikymas

2.11.3.1 Reikalavimai esamų duomenų perkėlimui

Nėra

2.11.3.2 Reikalingas duomenų transformavimas perkeliant į naują sistemą

Nėra

2.12 Rizikos

2.12.1 Galimos sistemos kūrimo rizikos

Rizikos faktorius	***Tikimybinis įvertinimas[1]***
Papildomų parametrų įtraukimas	2
Neteisingai parinkti efektyvumo parametrai	10

2.12.2 Atsitiktinumų planas

Rizikos faktoriai ir numatomi planai problemoms spręsti

Rizikos faktorius	***Problemos sprendimas***
Papildomų parametrų įtraukimas	Sistemoje turi būti numatyta galimybė greitam papildomų efektyvumo parametrų įtraukimui
Neteisingai parinkti efektyvumo parametrai	Efektyvumo parametrų keitimas turi užimti nedaug laiko.

2.13 Vartotojo dokumentacija ir apmokymas

Reikalinga vartotojo dokumentacija išsamiai aprašanti stebimus parametrus, grafikus ir kitus jų atvaizdavimo elementus. Taip pat dokumentacijoje turi būti pateikta informacija kaip interpretuoti tam tikras parametrų reikšmes bei grafinį jų atvaizdavimą.

3 Architektūros specifikacija

3.1 Apžvalga

Dokumentą sudaro bendras architektūros aprašymas, panaudojimo atvejų vaizdas, statinis sistemos vaizdas, dinaminis sistemos vaizdas. Dokumentas skirtas aprašyti techninį kuriamos programinės įrangos sprendimų įgyvendinimą ir pagrindimą.

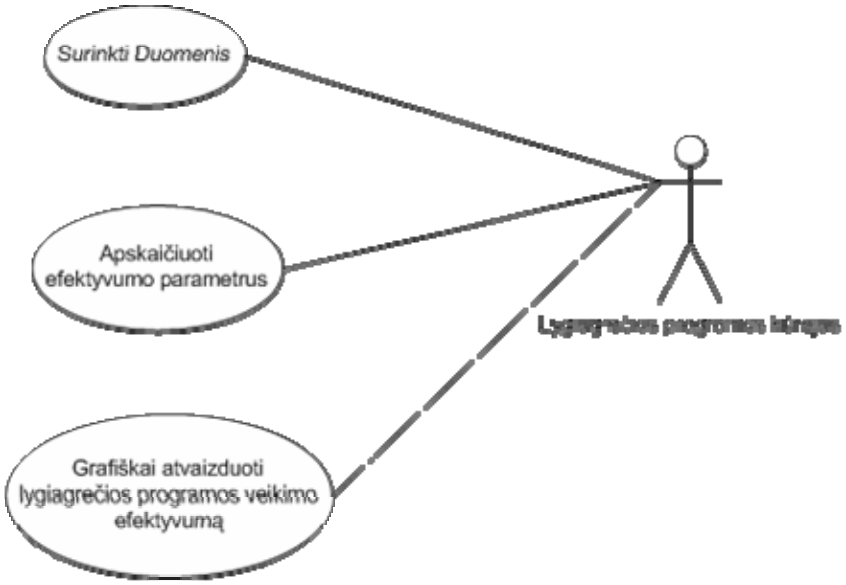
3.2 Architektūros pateikimas

Architektūra pateikiama panaudojimo atvejų, statiniu ir dinaminiu vaizdais. Panaudojimo atvejų vaizdą sudaro sistemos panaudos atvejai. Statinis sistemos vaizdas susideda iš paketų vaizdo, bei klasių diagramų.
Dinaminis sistemos vaizdas susideda iš sekų, bendradarbiavimo, būsenų ir veiklos diagramų.

3.3 Architektūros tikslai ir apribojimai

Sistema realizuojama naudojant objektinę programų kūrimo metodologiją. Sistema pradedama kurti nuo duomenų surinkimo modulio bei sąsajos su MPI biblioteka. Vėliau bus kuriama surinktos informacijos analizės dalis, ir galiausiai vartotojo sąsaja gautiems rezultatams pateikti, bei darbo procesui valdyti.

3.4 Panaudojimo atvejų vaizdas



7 pav. Sistemos panaudos atvejai

Pateikiamas bendras panaudos atvejų vaizdas, parodantis tris pagrindinius galimus sistemos panaudojimo būdus. Panaudos atvejai ir su jais susiję reikalavimai aprašyti [1].

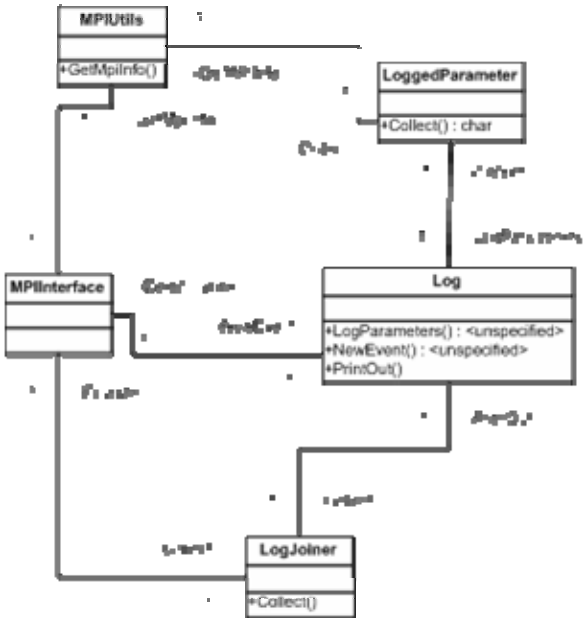
3.5 Sistemos statinis vaizdas

3.5.1 Apžvalga

8 Paketų diagrama

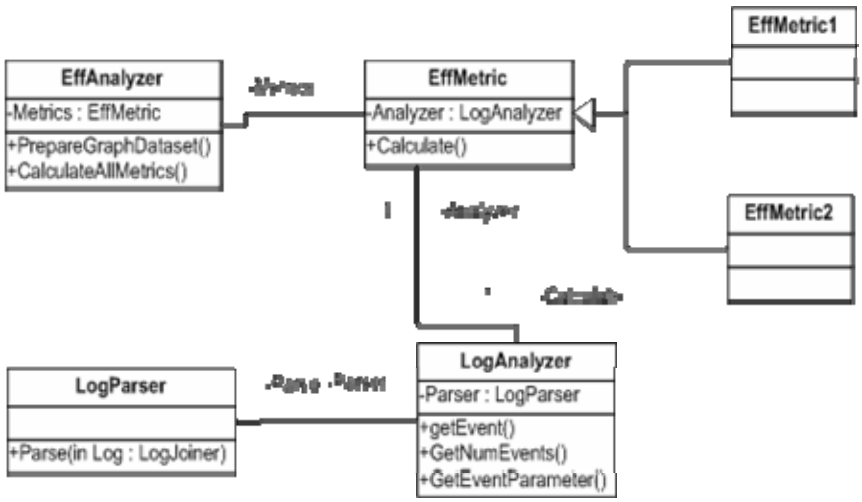
3.5.2 Paketų detalizavimas

Duomenų surinkimo modulis – Atsakingas už žurnalinius failus, jų apjungimą ir sąsają su MPI biblioteka. Turi būti galimybė lengvai papildyti sistemą nauja surenkama informacija(naujais parametrais).



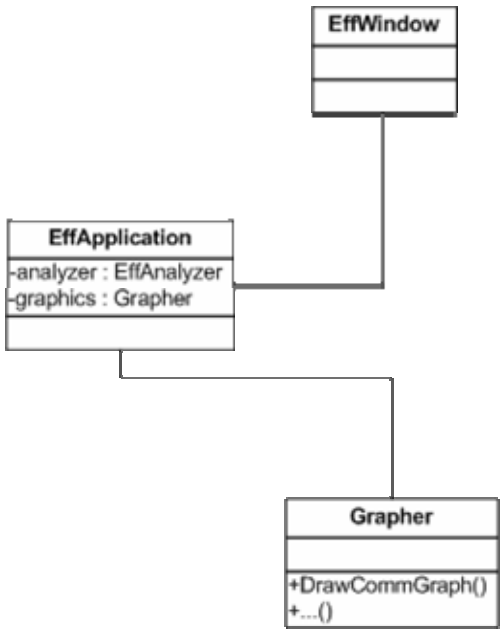
9 pav. Duomenų surinkimo modulio klasių diagrama

Skaičiavimų modulis – Atsakingas už programos efektyvumo parametrų(išvestinių) skaičiavimą, efektyvumo vertinimą. Pateikia duomenis grafinio atvaizdavimo moduliui.



10 pav. Skaičiavimų modulio klasių diagrama

Vartotojo sąsaja / grafinis atvaizdavimas – Modulyje realizuota programos vartotojo sąsaja, taip pat realizuota grafinio atvaizdavimo programos dalis.

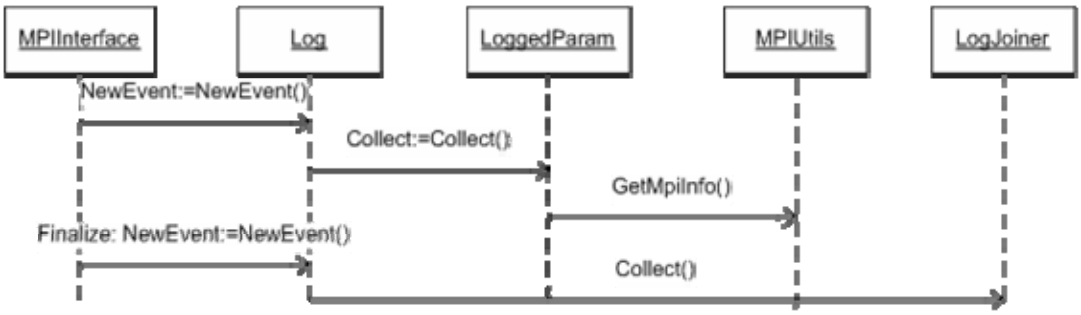


11 pav. Vartotojo sąsajos modulio klasių diagrama

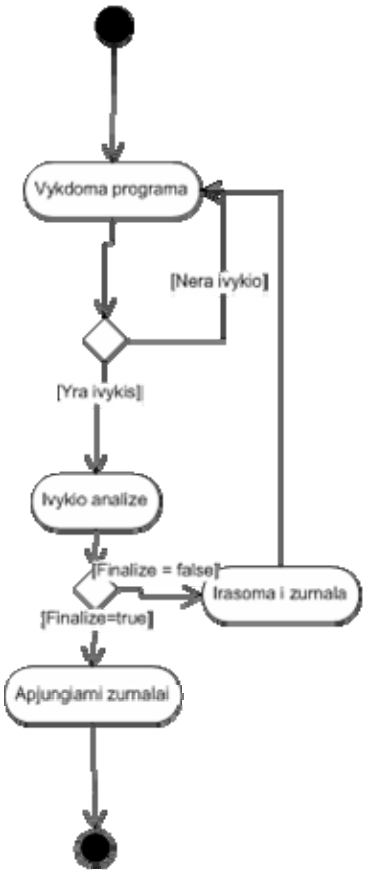
3.6 Sistemos dinaminis vaizdas

Pateikiamos sąveikos (interaction), būsenų (state) ir veiklos (activity) diagramos. Sąveikai atvaizduoti pakanka pasirinkti vieną iš dviejų diagramų – sekų (sequence) arba bendradarbiavimo (collaboration), tačiau esminiams sprendiniams pagrįsti galima pateikti

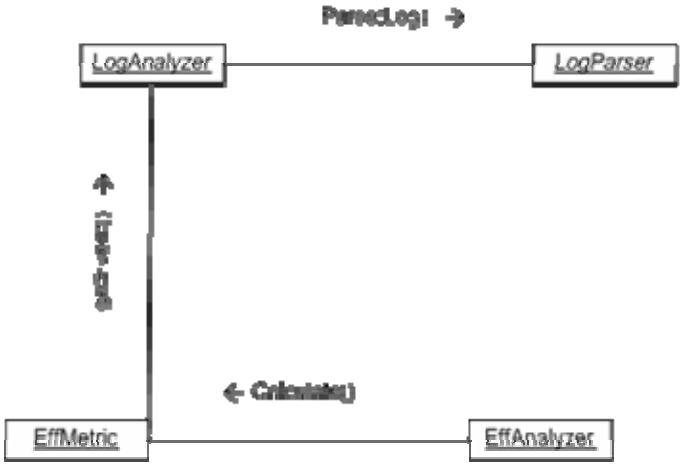
ir abi. Už pagrindą pasirinkus sekų (bendradarbiavimo) diagramas, dokumentacijoje privalo būti bent viena bendradarbiavimo (sekų) diagrama.



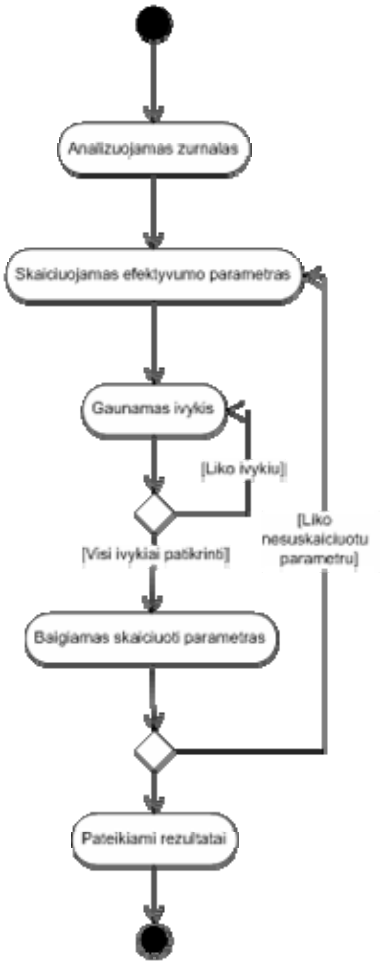
12 pav. Duomenų rinkimo modulio sekų diagrama



13 pav. Duomenų rinkimo modulio veiklos diagrama



14 pav. Skaičiavimų modulio bendradarbiavimo diagrama



15 pav. Skaičiavimų modulio veiklos diagrama

3.7 Išdėstymo vaizdas

Sistemos duomenų surinkimo dalis turi veikti kompiuterių klasteryje su įdiegta MPI programine įranga. Sistemos skaičiavimų modulis gali veikti nepriklausomai, svarbu kad duomenų surinkimo dalies sugeneruotas žurnalinis failas būtų pasiekiamas. Vartotojo sąsaja taip pat gali veikti nepriklausomai nuo duomenų surinkimo dalies, bet yra priklausoma nuo skaičiavimų modulio.

3.8 Kokybė

Sistema kuriama taip, kad būtų galima lengvai pridėti naujus surenkamus duomenis prie jau esamų, taip pat siekiama, kad būtų galima lengvai pridėti naujus efektyvumo parametrus. Sistema turėtų veikti bet kuriame MPI palaikančiame kompiuterių klasteryje. Duomenų surinkimo modulio atskyrimas leidžia sumažinti sistemos veikimo įtaką tiriamoms programoms. Naudojami žurnaliniai failai leidžia lygiagrečių programų analizę atlikti bet kada po programos įvykdymo ir lengvai saugoti senesnius žurnalinius failus palyginimui.

4 Sukurtos programinės įrangos tyrimas

4.1 Iškilusios problemos

Bandant bei eksploatuojant sukurtą lygiagrečių programų efektyvumo tyrimo sistemą buvo susidurta su tokiomis problemomis :

- Didelės laiko sąnaudos ir dideli pakeitimai programos kode, norint įtraukti naujų parametrų atvaizdavimą.
- Didelis žurnalinio failo dydis, kuomet tiriama programa persiunčia didelius pranešimų kiekius
- Nepakankamas laiko matavimo tikslumas
- Nepakankamas programos lankstumas atvaizduojant rezultatus

4.2 Keitimų įvertinimas

Buvo išanalizuoti galimi pakeitimai, suprojektuoti programos architektūros bei realizacijos pakeitimai siekiant sumažinti arba visai pašalinti pastebėtus trūkumus.

- Duomenų surinkimo modulis buvo perrašytas ir pritaikytas lengvesniam naujų parametrų pridėjimui.

- Sukurtas naujas duomenų failo dvejetainis formatas leido sumažinti žurnalinio failo dydį

- Laiko matavimo korekcijos atliekamos analizuojant žurnaliniame faile įrašytus sinchronizuojančius įvykius, bei koreguojant laiko matavimo paklaidas(drift)

Rezultatų grafinį atvaizdavimą galima padaryti lankstesnį tokiomis priemonėmis:

- Pridėti procesų filtravimo galimybę (filtruojama kuriuos procesus atvaizduoti)
- Pridėti pranešimų filtravimą pagal tipą.
- Numatyti procesų agregavimą (galima sudaryti iš kelių procesų vieną grupę ir ją atvaizduoti)
- Sudaryti galimybę keisti atvaizdavimo parametrus (linijos storio, tipo, spalvos ir etc. keitimas)
- Įvesti papildomus grafiko atvaizdavimo stilius
- Komunikacijas tarp procesų atvaizduoti lentele
- Numatyti pranešimų skirstymą pagal iškvietimo vietą programos kode
- Įvesti papildomas pranešimų savybių analizė priemones(trukmė, dydis etc.)
- Pridėti rezultatų failų palyginimo galimybę

Taip pat buvo atlikti pakeitimai siekiant pagerinti programos našumą apdorojant didelius žurnalinius failus.

4.2.1 Duomenų surinkimo modulio pakeitimai

Tobulinant duomenų surinkimą buvo atlikta:

- Sukurtos klasės apjungiančios bendras įvykių savybes bei metodus leido sumažinti kodo pasikartojimą plečiant apdorojamų įvykių sąrašą.
- Numatyta galimybė nustatyti norimą rezultatų failą bei jo formatą

Duomenų surinkimas vyksta programai kviečiant MPI funkcijas. Didelė dalis duomenų surinkimo funkcionalumo yra bendra, nepriklausanti nuo konkrečios kviečiamos MPI funkcijos(trukmės matavimas, įvykio įrašymas). Todėl yra tikslinga šį funkcionalumą aprašyti atskirai nuo MPI funkcijų. Buvo pridėta 10 apibendrintų funkcijų, kurios leido daugumą tiriamų funkcijų aprašų sumažinti nuo ~50 iki ~20 eilučių (vidutiniškai). Taip pat pagerėjo naujų papildomų funkcijų įvedimo galimybė, kadangi kiekvienai pridedamai funkcijai aprašyti ir surinkti apie ją reikiamus duomenis, reikia mažiau papildomų kodo eilučių. Pakeitimai įvykių surinkimo modulyje atliekami greičiau.

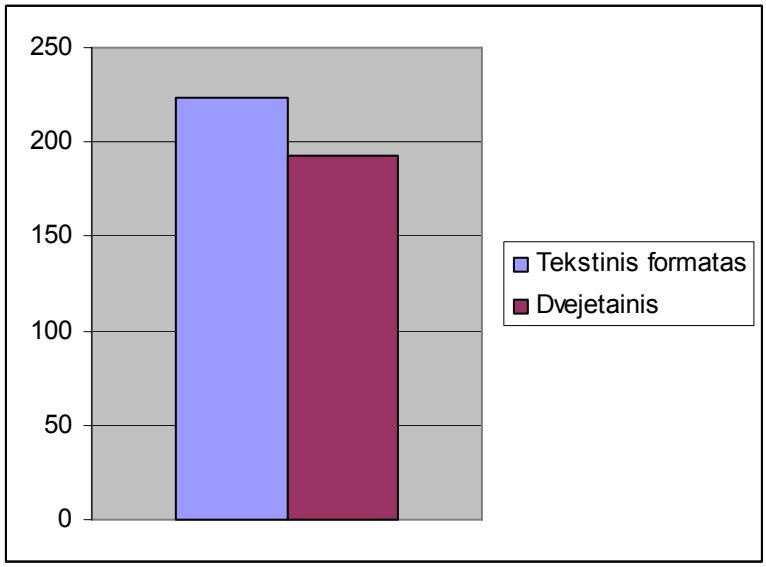
4.2.2 Naujas žurnalinio failo formatas

Naujas žurnalinio failo formatas leidžia gauti:

- Kompaktiškesnį įvykių užrašymą
- Greitesnį duomenų nuskaitymą bei įrašymą
- Mažesnę failo apimtį

Žurnalinis failas naudoja tekstinį duomenų formatą, kuris nėra optimalus užimamos vietos atžvilgiu. Kadangi failo struktūra yra pakankamai pastovi ir nesudėtinga, yra tikslinga duomenis įrašinėti dvejetainiu formatu, taip sumažinant duomenų apimtį ir padidinant failo nuskaitymo/įrašymo greitį. Failas nėra skirtas nagrinėti be atitinkamos atvaizdavimo programos, todėl jo suprantamumas nėra svarbus. Failo formato pakeitimas leidžia sumažinti jo apimtį 13%. Suderinamumo tikslais, yra paliktas ir tekstinis formatas, kuris praverčia programos derinimo metu. Yra galimybė pasirinkti norimą žurnalinio failo formatą.

Lygiagrečios matricų daugybos programos žurnalinis failas su 3020 įrašų užima apie 223 kB tekstiniame formate. Pervedus į dvejetainį formatą failo dydis sumažėjo iki 193 kB. Sumažėjimas pavaizduotas žemiau.



16 pav. žurnalinio failo formato dydis

4.2.3 Laiko matavimo tikslinimas

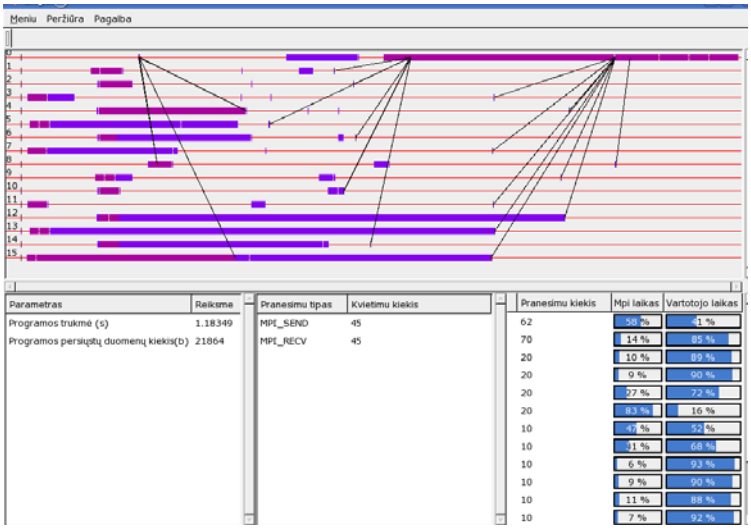
Iki šiol laikas būdavo sinchronizuojamas programos vykdymo pradžioje. Buvo įgyvendinti patobulinimai, kuomet laikas papildomai sinchronizuojamas kiekvieną kartą kai programa yra sinchronizuotoje būsenoje (barrier). Tokiu būdu buvo užtikrintas didesnis laiko matavimo tikslumas, ypač ilgiau veikiančiose programose.

Procesams veikiant skirtinguose kompiuteriuose gali išryškėti laiko matavimo netikslumai. Programai veikiant ilgą laiką buvo galima pastebėti laiko matavimo skirtumus sinchronizuotose programos vietose, kurie siekė iki 0.5 sek. Šiuos skirtumus galima kompensuoti, pasirenkant atskaitinį procesą ir atitinkamai pakoreguojant kituose procesuose užrašytus pranešimų siuntimo laikus.

4.2.4 Procesų filtravimas

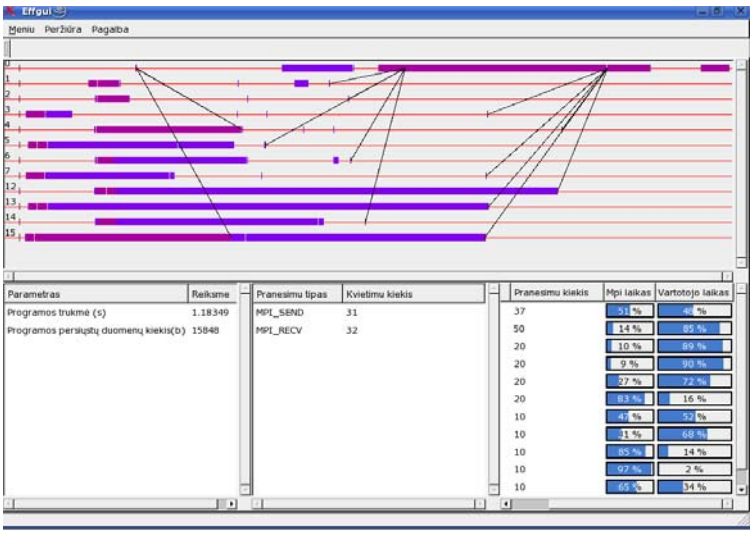
Procesų filtravimas suteikia programos kūrėjams galimybę peržiūrėti tik juos dominančius procesus bei paprasčiau atlikti norimus šių procesų palyginimus.

Pavyzdžiui taip atrodo programos su 16 procesų pranešimų grafikas :



17 pav. Nefiltruoti procesai

Panaudojus filtravimą išmetami 8-11 procesai, kaip matoma žemiau esančiame paveikslėlyje.



18 pav. Programos vaizdas po filtravimo

Kaip matome, procesų filtravimą galima panaudoti norint peržiūrėti tik dominančius procesus, išdėstant juos vienas šalia kito. Reikalui esant galima išskirti bet kuriuos du dominančius procesus ir peržiūrėti tarp jų siunčiamus pranešimus.

4.2.5 Pranešimų filtravimas

Pranešimų filtravimas leidžia susitelkti į dominančius pranešimų tipus. Yra supaprastinamas grafinis programos vaizdas, nereikalingi analizei pranešimai gali būti nevaizduojami.

4.2.6 Procesų agregavimas

Procesų agregavimas leidžia lanksčiau atlikti programos analizę, galima procesus agreguoti pagal atliekamas funkcijas ir tokiu būdu analizuoti komunikacijas tarp skirtingų programos dalių.

Procesų agregavimas iš esmės panašus savo funkcionalumu su procesų filtravimu, todėl įgyvendinimas buvo atidėtas vėlesniam laikui.

4.2.7 Grafinio atvaizdavimo stiliai

Galimybė pasirinkti kaip atvaizduoti programos veikimą ir komunikacijas. Ši galimybė nėra būtina programai, ir yra daugiau vartotojo patogumui skirta priemonė, kurios įgyvendinimas užtruktų gana ilgai, todėl ji gali būti įgyvendinta kitoje programos versijoje.

4.2.8 Komunikacijų tarp procesų atvaizdavimas lentele

Galimybė peržiūrėti tokius parametrus kaip iš vieno konkretaus proceso į kitą persiustų pranešimų kiekis ir pan. vienoje lentelėje. Pavyzdys kaip galėtų atrodyti lentelė pateiktas žemiau.

procesas	0	1	2	..	n
0	x	0.4131	0.4010	..	a

1	0.4129	x	0.3786	..	b
2	0.4057		x	..	c
..	..	..	..	x	
n	d	e	f	..	x

Langelių reikšmės parodo kokios nors metrikos reikšmę (pvz. vidutinė pranešimo siuntimo trukmė tarp duotų procesų).

Galimybė numatyta įgyvendinti ateityje.

4.2.9 Pranešimų skirstymas pagal vietą programos kode

Peržiūrėti pranešimų kiekį, dydžius ir kitus parametrus, grupuojant pagal kvietimo vietą programos kode. Tokiu būdu galima pažiūrėti kuriose programos vietose yra daugiausiai kviečiamos pranešimų perdavimo funkcijos.

Ši programos funkcija reikalauja tam tikrų tiriamos programos savybių bei reikalauja daug darbo įgyvendinant, be to nežinoma vietos nustatymo įtaka tiriamos programos veikimo greičiui.

4.2.10 Pranešimų savybių analizė

Pateikiama statistika histogramos pavidalu apie pranešimų dydžių, persiuntimo laiko ir kitų parametrų pasiskirstymą.

Ši programos funkcija gali būti įgyvendinta kitoje programos versijoje.

4.2.11 Rezultatų palyginimo galimybė

Realizuota galimybė šalia atvaizduoti du rezultatų failo vaizdus. Tokiu būdu galima greitai surasti pokyčius programos veikime atlikus pakeitimus programos kode, po programos optimizavimo.

5 Lygiagrečių programų efektyvumo tyrimas

5.1 Pranešimų dydžio įtakos siuntimui analizė

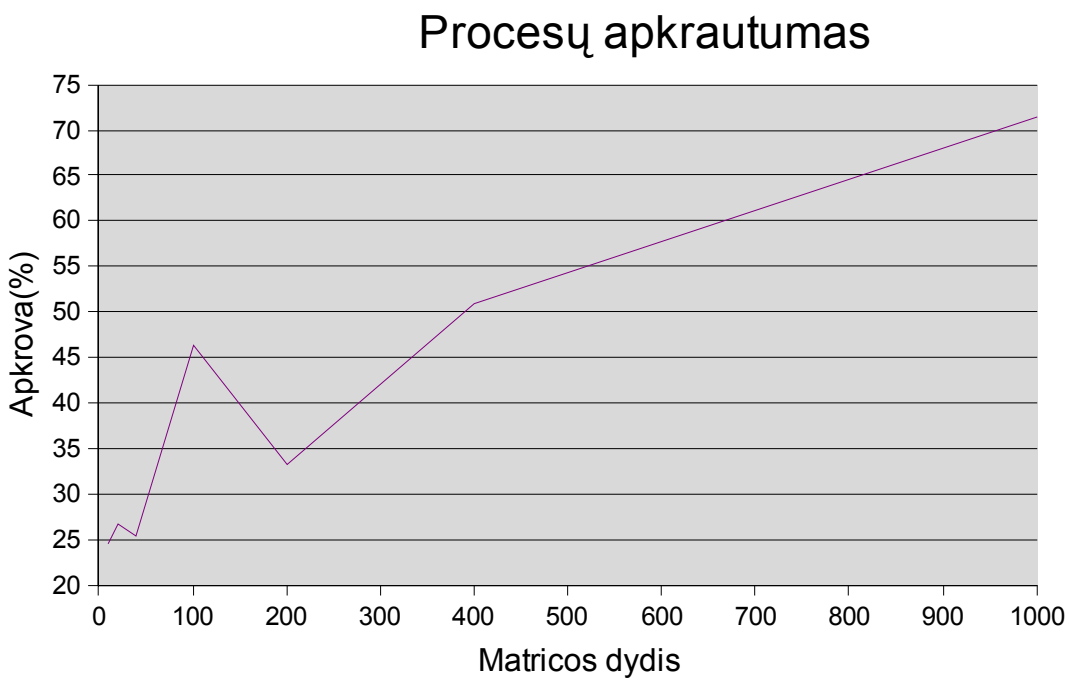
Tarp procesų persiunčiamų duomenų apimtis priklauso nuo uždavinio tipo, algoritmo, persiuntimui naudojamų funkcijų ir kitų faktorių. Kai kuriems uždaviniams persiunčiamų duomenų apimtį galima lanksčiai keisti. Tai įtakoja bendrą skaičiavimų trukmę. Analizei buvo pasirinkti du uždaviniai. Pirmas uždavinys – masyvo rikiavimas radix sort algoritmu. Antras uždavinys – matricų daugyba naudojant Fox algoritmą.

5.1.1 Matricų daugyba Fox algoritmu

Algoritmas skirtas dviejų matricų dauginimui naudojant kelis lygiagrečius procesus. Algoritmas gali būti naudojamas tik kvadratinių matricų dauginimui, su sąlyga, kad procesų skaičiaus kvadratinė šaknis yra sveikas skaičius ir matricos eilė dalijasi iš šio skaičiaus be liekanos. Programa vykdoma su atsitiktinai sugeneruotomis matricomis, kurių dydžiai kinta nuo 10x10 iki 1000x1000. Tiriama, kaip kinta procesų apkrovimas, priklausomai nuo dauginamų matricų dydžio. žemiau pateikiama lentelė, kurioje surašyti programos vykdymo laikai, perduotų duomenų kiekis ir procesų apkrovimas, priklausomai nuo dauginamų matricų dydžio. Programa leidžiama naudojant keturis lygiagrečius procesus.

matr_dydis	10	20	40	100	200	400	1000
trukm(s)	2.86	2.87	2.89	3.44	1.36	3.5	15
kiekis(B)	3416	13616	54416	340016	1360016	5440016	34000016
apkrova(%)	24.5	26.75	25.25	46.25	33.25	51	71.5

Žemiau pateiktas grafikas, kuriame vaizduojamas procesų apkrovimas kintant matricų dydžiui.



19 pav. Procesų apkrautumas kintant matricos dydžiui

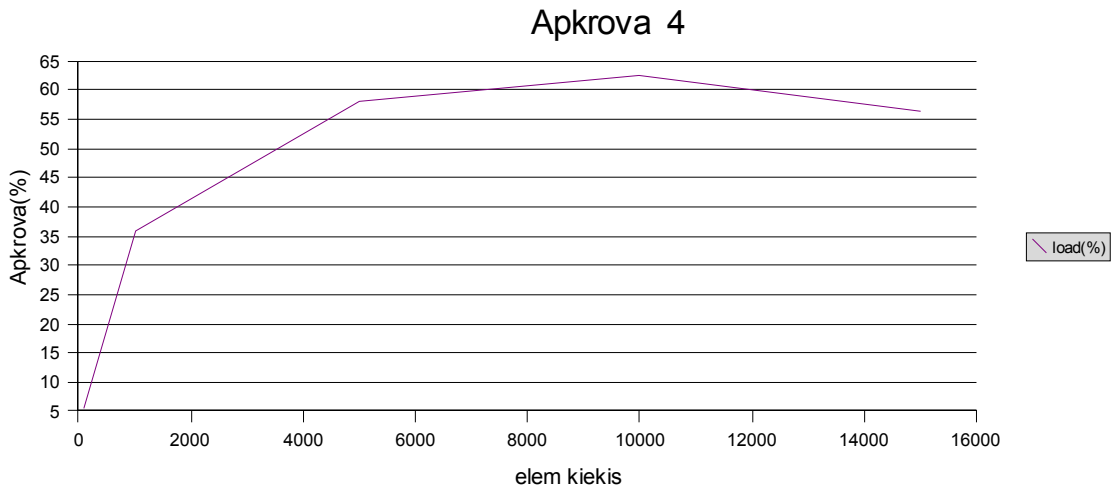
Iš grafiko matome, kad apkrova didėja didinant dauginamų matricų dydį. Algoritmo specifika sąlygoja sumažėjusį efektyvumą esant 200x200 matricos apimtims.

5.1.2 Masyvų rikiavimas lygiagrečiai naudojant „Radix sort“

Algoritmas rikiuoja masyvą radix sort algoritmu. Rikiavimas vykdomas lygiagrečiai keliuose procesuose. Komunikavimui naudojama MPI biblioteka. Skaičiavimų efektyvumas priklauso nuo naudojamų procesų kiekio bei siunčiamų duomenų kiekio bei siuntimų kiekio. Nagrinėjama siuntimų kiekio ir jų dydžio įtaka programos efektyvumui. Rikiavimui pasirinktas atsitiktinai sugeneruotas masyvas iš 100000 elementų. Programa sukurta taip, kad skirtingiems procesams apdoroti skirti masyvo fragmentai gali būti skirtingo dydžio. Keičiant šiuos nustatymus keičiasi ir programos veikimo laikas bei skaičiavimams praleisto laiko dalis. Žemiau pateikiama lentelė, kurioje surašyta priklausomybė tarp siunčiamų masyvo fragmentų dydžio bei programos veikimo trukmės ir skaičiavimų efektyvumo vykdant programą su 4 procesais.

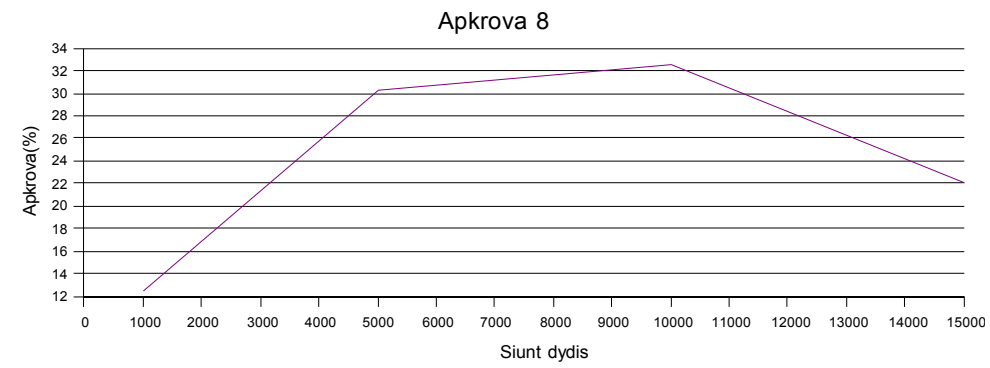
fragmento dydis	100	1000	5000	10000	15000
trukm(s)	13.01	2.03	1.68	1.49	1.49
kiekis(B)	601040	628064	599800	600224	398736
apkrova(proc)	5.6	36	58	62.6	56.3

Atitinkamas grafikas, parodantis efektyvaus darbo dalies priklausomybę nuo proceso apdorojamo masyvo fragmento dydžio(siuntimo dydžio)



20 pav. Procesorių apkrova kintant masyvo fragmento dydžiui. 4 procesai

Apkrovos grafikas su aštuoniais procesais atrodo taip



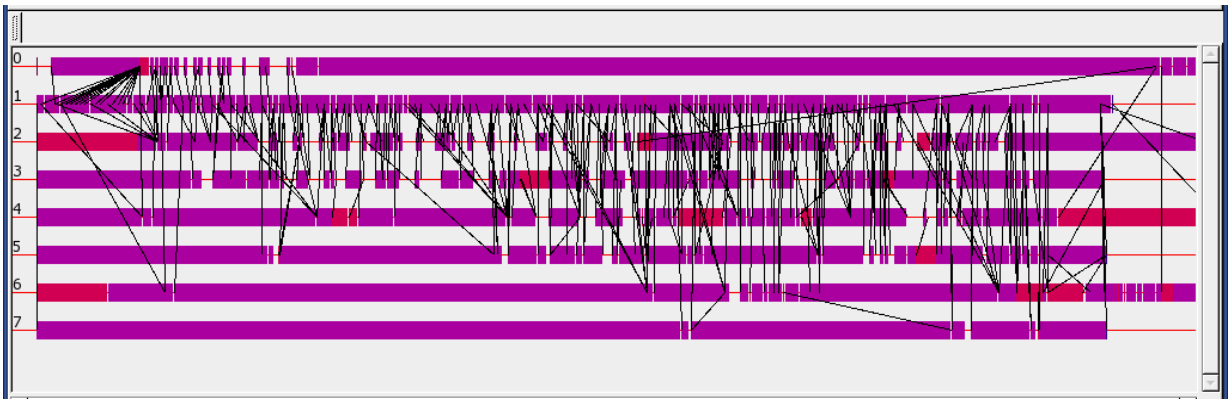
21 pav. Procesorių apkrova kintant masyvo fragmento dydžiui. 8 procesai.

Kaip matome, didėjant siunčiamų duomenų paketų dydžiui, programos efektyviai praleisto laiko dalis didėja iki tam tikro paketo dydžio. Vėliau tokiam masyvo dydžiui skaidymas dėl algoritmo ypatumų tampa neefektyvus, išnaudojami ne visi procesai, todėl naudingo darbo dalis pradeda mažėti.

5.2 „Radix sort“ algoritmo parametrų parinkimas

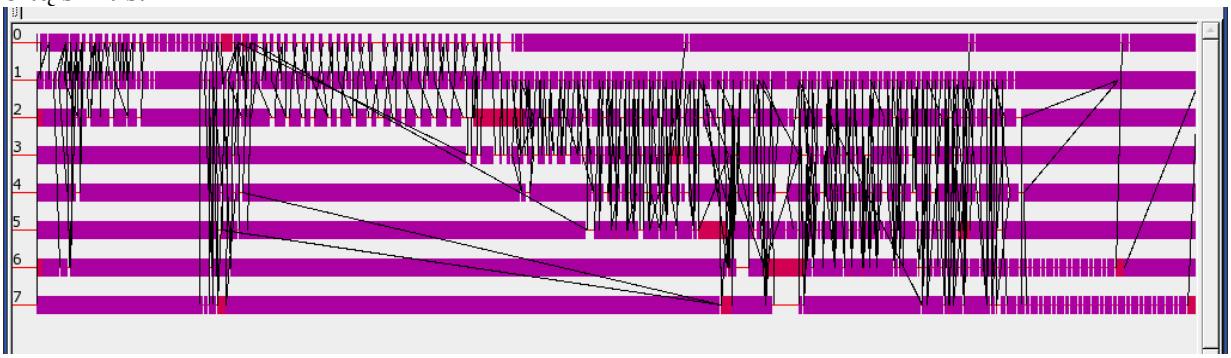
Šiame tyrime analizuojamas radix sort algoritmo efektyvumas keičiant vienu metu rikiuojamų skilčių skaičių. Vienu kartu rikiuojama nuo 1 bito iki 8 bitų duomenų. Gautame grafike matosi kaip keičiasi naudojamų procesorių apkrovimas priklausomai nuo skilčių dydžio. Tiriamas algoritmas naudojant 8 ir 16 procesorių.

Programos veikimo vaizdas su 8 procesoriais vienu metu rikiuojant po vieną bitą pateikiamas žemiau.



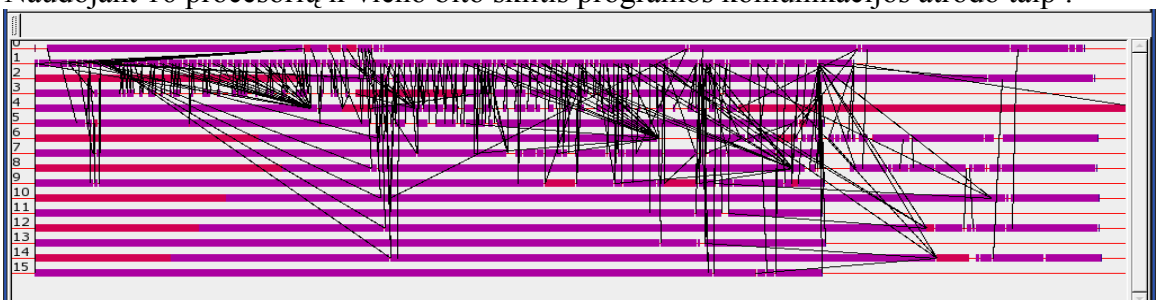
22 pav. rikiavimas po 1 bitą

Toliau galime pažiūrėti kaip keičiasi programos komunikacijos rikiuojant vienu metu 4 bitų skiltis.



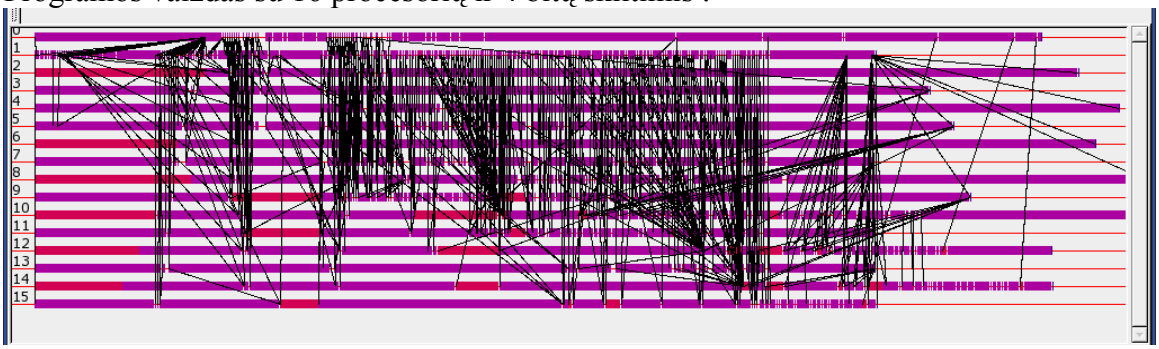
23 pav. Rikiavimas po 4 bitus

Naudojant 16 procesorių ir vieno bito skiltis programos komunikacijos atrodo taip :



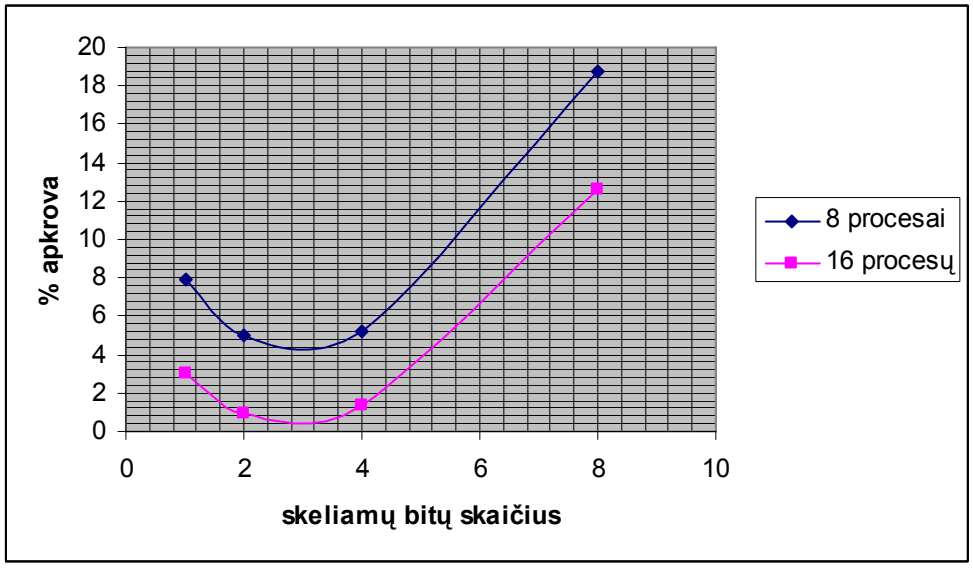
24 pav. Rikiavimas po 1 bitą, 16 procesų

Programos vaizdas su 16 procesorių ir 4 bitų skiltimis :



25 pav. Rikiavimas po 4 bitus, 16 procesų

Atlikus procesorių apkrovimo analizę gauname, kad vidutinis procesorių apkrautumas yra mažiausias kai naudojamos 2 ir 4 bitų ilgio skiltys ir padidėja naudojant 8 bitų skiltis. Šis tyrimas neparodo skilčių dydžio įtakos programos naudojamam atminties kiekiui.



26 pav. apkrovos priklausomybė nuo rikiuojamų bitų skaičiaus

Galima daryti išvadą, kad skaičiavimo efektyvumo požiūriu apsimoka naudoti didesnius (8 bitų) skilčių dydžius radix rūšiavimo algoritme. Galima pastebėti, kad naudojamų procesorių kiekis iš esmės nepakeičia efektyvumo kitimo tendencijų, nors pastebimas bendras mažesnis efektyvumas didinant procesorių skaičių. Programos naudojama algoritmo realizacija neleidžia naudoti didesnių negu 8 bitų skilčių.

6 Išvados

- Suprojektuota, sukurta bei išbandyta lygiagrečių MPI programų efektyvumo tyrimo mokomoji programinė įranga.

- Atsižvelgiant į sukurtos sistemos pastebėtus trūkumus, atlikti pakeitimai duomenų surinkimo bei analizės modeliuose, kurie palengvina lygiagrečių programų efektyvumo analizę. Suplanuoti tolimesni sistemos pakeitimai ir tobulinimo darbai, kurie leistų dar labiau padidinti programinės įrangos galimybes.

- Išanalizuota pranešimų dydžio įtaka jų siuntimo laikui. Buvo pasirinkti bandomieji uždaviniai ir ištirta, kaip kinta jų vykdymo trukmė keičiant siunčiamų pranešimų dydžius. Ištyrus buvo rastos efektyvumo kitimo tendencijos minėtiems uždaviniams, bei geriausi pranešimų dydžiai.

- Buvo ieškomi tinkamiausi radix sort algoritmo parametrai. Tam tikslui buvo tiriama bandomoji programa, kurios pagalba nustatytos geriausios algoritmo parametrų reikšmės.

7 Literatūra

[1] Introduction to parallel programming.[Žiūrėta 2006.11.20], prieiga internete (http://www.mhpcc.edu/training/workshop/parallel_intro/MAIN.html)

[2] Yang, C. and Miller, Performance Measurement for Parallel and Distributed Programs: a Structured and Automatic Approach. *IEEE Trans. Softw. Eng.* 15, 12, Dec. 1989.

[3] Thomas Fahringer, Estimating and Optimizing Performance for Parallel Programs, IEEE Computer, 28(11):47 -- 56, November 1995.

[4] M. Crovella and T. LeBlanc, Parallel Performance Prediction Using Lost Cycle Analysis, Proc. Supercomputing 94, IEEE, pp. 600-609, November 1994.

[5]http://www.phy.duke.edu/resources/computing/brahma/brahma_old/als/als/node3.html (2006 m. lapkričio 9 d.)

[6] J. K. Hollingsworth and B. P. Miller, Parallel Program Performance Metrics: A Comparison and Validation , Supercomputing 1992, Minneapolis, MN, November 1992, pp.

[7] LMPI: A Profiling Library for MPI Programs [Žiūrėta 2006.11.20], prieiga internete http://www.lrz-muenchen.de/services/software/parallel/lmpi/

[8] mpiP: Lightweight, Scalable MPI Profiling [Žiūrėta 2006.11.20], prieiga Internete http://mpip.sourceforge.net/

[9] MPICL (Instrumentation library for MPI) [Žiūrėta 2006.11.20], prieiga Internetą http://www.csm.ornl.gov/picl/

[10] Intel® Trace Analyzer and Collector 7.0 for Linux [Žiūrėta 2006.11.20], prieiga Internete (http://www.intel.com/cd/software/products/asmo-na/eng/cluster/tanalyzer/306321.htm#interface

[11] Jeffrey K. Hollingsworth, James Lumpp, Barton P. Miller, Techniques for Performance Measurement of Parallel Programs, Parallel Computers: Theory and Practice, IEEE Press, 1995

[12] Nutt, G. J., Griff, A. J., Mankovich, J. E., and McWhirter, J. D. Extensible Parallel Program Performance Visualization. In *Proceedings of the 3rd international Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (January 10 - 18, 1995). P. W. Dowd and E. Gelenbe, Eds. MASCOTS. IEEE Computer Society, Washington, DC, 205-211, 1995.

[13] T. J. Godin, Michael J. Quinn, C. M. Pancake, Parallel Performance Visualization using Moments of Utilization Data. In *Proceedings of the 12th. international Parallel Processing Symposium on international Parallel Processing Symposium* (March 30 - April 03, 1998). IPPS. IEEE Computer Society, Washington, DC, 777, 1995.

[14] M. Sottile, V. Chandu, and D.A. Bader, Performance analysis of parallel programs via message-passing graph traversal,' Proc. 20th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS), Rhodes Island, Greece, April 2006.

[15] Chassin de Kergommeaux, J., Maillet, É., and Vincent, J, Monitoring parallel programs for performance tuning in cluster environments. In *Parallel Program Development For Cluster Computing: Methodology, Tools and integrated Environments* Advances In Computation: Theory And Practice, vol. Volume 5. Nova Science Publishers, Commack, NY, 131-150, 2001

[16] MPI-2: Extensions to the Message-Passing Interface, [Žiūrėta 2006.11.20], prieiga internete http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html