KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Dovydas Loda

NUOSEKLIŲ IR LYGIAGREČIŲ ALGORITMŲ TYRIMAS CUDA IR OPENMP TECHNOLOGIJOSE

Baigiamasis magistro darbas

Vadovas
doc. Romas Marcinkevičius

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

NUOSEKLIŲ IR LYGIAGREČIŲ ALGORITMŲ TYRIMAS CUDA IR OPENMP TECHNOLOGIJOSE

Baigiamasis magistro darbas
Programų sistemų inžinerija (621E16001)

Vadovas
(parašas) doc. Romas Marcinkevičius
(data)

Recenzentas
(parašas) doc. dr. Stasys Maciulevičius
(data)

Projektą atliko
(parašas) Dovydas Loda
(data)

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos fakultetas

(Fakultetas)

Dovydas Loda

(Studento vardas, pavardė)

Programų sistemų inžinerija (621E16001)

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Nuoseklių ir lygiagrečių algoritmų tyrimas CUDA ir OpenMP technologijose“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 17 m. gegužės 19 d.

Kaunas

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

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

(vardą ir pavardę įrašyti ranka) *(parašas)*

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Loda, D. Nuoseklių ir lygiagrečių algoritmų tyrimas CUDA ir OpenMP technologijose. Programų sistemų inžinerijos magistro baigiamasis projektas. Vadovas doc. R. Marcinkevičius; Kauno technologijos universitetas, Informatikos fakultetas.

Kaunas, 2017. 64 p.

SUMMARY

The project is called "Serial and parallel algorithms analysis in CUDA and OpenMP technologies“. The attention is paid to the aim of finding out OpenMP and CUDA technologies differences, capabilities and efficient configurations for most used algorithms.

In fact, the study examines algorithms which can be operated in parallel and sequential order using OpenMP and CUDA technologies. They are tested with various configuration, data samples. It is expected that after the study of algorithms, their speed and other problems, connected with OpenMP and CUDA technologies, will be easier to determine how to further improve the previous half of the year generated code conversion system for these technologies.

This project is also important because of computing resource utilization efficiency, the benefits of parallelism usage. Examination of sequential and parallel algorithms, visualized acceleration graphs will help to attract more emphasis on parallel programming by industry and educational institutions.

1. ĮVADAS

Šiuolaikiniame pasaulyje žmogus negali apsieiti be kompiuterių. Tiek pramonėje, tiek asmeniniame gyvenime, jie žmogui padeda atlikti įvairius darbas, spręsti problemas, palengvinti gamybą, informacijos rinkimą, apdorojimą [1]. Kompiuterinė bei programinė įranga yra nuolat tobulinama, siekiant greitesnio darbo, didesnio duomenų pralaidumo [2]. Procesoriai su keletu branduolių, „Nvidia“ CUDA technologija suteikia galimybes vykdyti įvairias programas daug sparčiau ir efektyviau, naudojant lygiagrečius procesus, panaudojant OpenMP ar CUDA technologijas [3][4]. Tačiau programinė įranga bei įvairios informacinės sistemos vis dar nėra kuriamos taip, kad būtų pritaikytos maksimaliai panaudoti lygiagretumo teikiamus privalomus. Šiame tyrime bus bandoma ištirti OpenMP bei CUDA technologijų galimybes bei nustatyti programų vietas, kurias galima labiausiai optimizuoti naudojant šias technologijas.

1.1. Dokumento paskirtis

Šio dokumento paskirtis yra pateikti visą informaciją, susijusią su kurtos sistemos realizacija, atliktais tyrimais bei jų rezultatais. Dokumentas sudarytas iš skyrių, kuriuose pateikiama atlikta mokslinių tyrimų, algoritmų ir įrankių analizė. Projektinėje dalyje pateikiama informacija apie sistemos realizaciją: naudotus įrankius, realizuotus algoritmus. Eksperimentinėje dalyje aprašomas vykdytas tyrimas, eksperimentai ir kitos įžvalgos.

1.2. Santrauka

Projektas vadinasi „Nuoseklių ir lygiagrečių algoritmų tyrimas CUDA ir OpenMP technologijose“. Didžiausias dėmesys yra skiriamas siekiui išsiaiškinti OpenMP ir CUDA technologijų galimybes, skirtumus, konfigūracijų skirtumus.

Atliekamas tyrimas su algoritmais, kuriuos galima vykdyti lygiagrečiai bei nuosekliai, naudojant OpenMP bei CUDA technologijas. Jie yra išbandomi su įvairiomis gijų paleidimo konfigūracijomis, duomenų imtimis. Tikimasi, kad ištirtų algoritmų greitaveika bei problemos, leis nustatyti kaip toliau tobulinti praeitame pusmetyje sukurtą kodo konvertavimo sistemą šioms technologijoms.

Tai pat projektas yra perspektyvus kompiuterinių išteklių panaudojimo efektyvumu, lygiagretumo teikiamų privalumų pagrindimu. Ištyrus nuoseklius bei lygiagrečius algoritmus, pademonstravus jų pagreitėjimą, tikimasi, kad ateityje bus daugiau dėmesio skiriama lygiagrečiam programavimui tiek pramonėje, tiek mokymo įstaigose.

2. ANALIZĖ

2.1. Darbo tikslas

Šio skyriaus darbo tikslas yra susipažinti su OpenMP [5] ir CUDA [6] technologijomis bei išanalizuoti egzistuojančias programas, kurios siūlo galimybę pritaikyti OpenMP ar CUDA technologiją naudojančios programos kodą priešingai technologijai. Taip pat aptarti kuriamos sistemos galimybes bei įgyvendinimo problemas.

2.2. Egzistuojantys sprendimai

2.2.1. GPSME programinis sprendimas

Viena iš sistemų, kurios gali padėti programuotojams paversti linijinį kodą lygiagrečiu, yra GPSME sistema [7][8]. Projekto aprašyme sakoma, kad šios sistemos pradiniai duomenys gali būti C/C++ kalbos kodo failai, kurie yra nuskaitomi ir transformuojami į abstrakčiosios sintaksės medį (angl. Abstract syntax tree), naudojant ROSE atviro kodo kompiliatoriaus platformą. Pačią sistemą sudaro kompiliatoriaus direktyvos, kurios suteikia galimybę programuotojui nustatyti, kaip GPSME sistemai kodą paversti į kuo efektyvesnį. Todėl, naudojant informaciją, gautą per panaudotas kompiliatoriaus direktyvas, sistema gali įgyvendinti skirtingas transformacijas, pasinaudodama abstrakčiosios sintaksės medžiu.

Šios sistemos rezultatas yra CUDA ir OpenCL karkasui pritaikytas kodas, kuris yra gaunamas gramatiškai išnagrinėjus abstrakčios sintaksės medį, panaudojant ROSE platformą. Tokiu būdu GPSME sistema dirbdama sukuria grafiniams CUDA procesoriams pritaikytus išlygiagretinamus ciklus. Be to, ši sistema remiasi *Mint* programavimo modeliu [5].

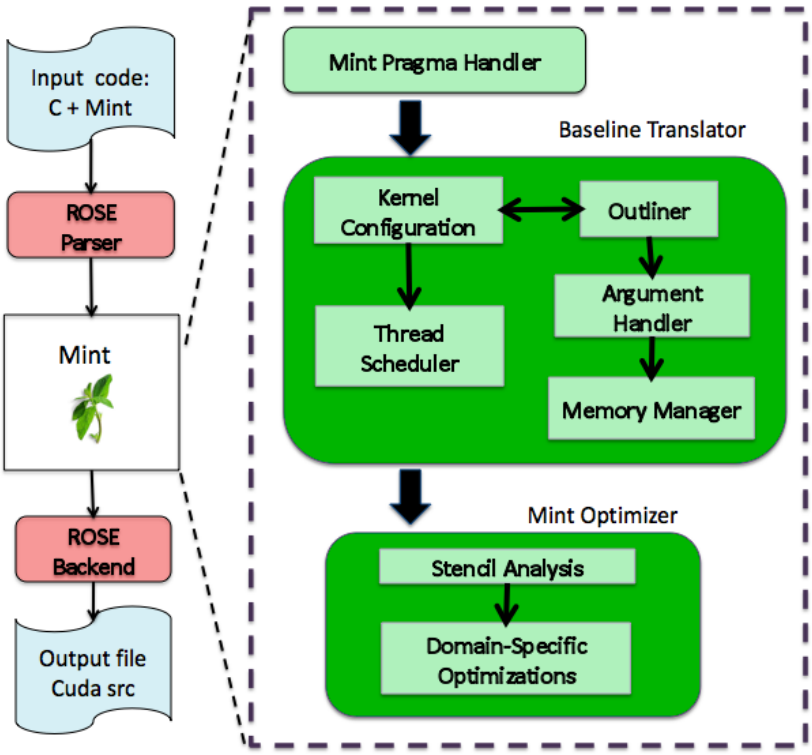
2.2.2. OpenMPC programinis sprendimas

Kita panaši sistema yra OpenMPC [9]. Ji praplečia standartinę OpenMP biblioteką naujomis kompiliatoriaus ir aplinkos direktyvomis, kurios programuotojams suteikia sudėtingo CUDA programavimo modelio abstrakciją ir aukšto lygio kontrolę bei optimizavimo galimybes. Taip pat šis projektas siūlo išbandyti OpenMP konvertavimą į CUDA OpenMPC interneto svetainėje, tačiau ši galimybė šiuo metu nėra pasiekiama. Ši sistema teikia CUDA technologijai pritaikytą vykdomąją programą, todėl jos kodo negalima redaguoti.

2.2.3. *Mint* programinis sprenimas

Mint [10] yra programavimo modelis ir vertyklė (angl. translator), kuris generuoja labai gerai optimizuotą kodą, pritaiką CUDA platformai, panaudojant tam tikromis *Mint* anotacijomis pažymėtą C programavimo kalbos kodą. Projekto aprašyme teigiama, kad programuotojas gali sėkmingai panaudoti grafinį procesorių, naudodamas tik penkias *Mint* kalbos kompiliatoriaus

direktyvas. Ši sistema sugeneruoja kodą, pritaikytą pagrindiniam ir grafiniams procesoriams, taip pat pasirūpina atminties išskyrimu bei atlaisvinimu. Toliau pateikiama *Mint* sprendimą vaizduojanti diagrama [11].



1 pav. Mint sprendimas

2.3. Kuriamos sistemos galimybės

Kuriama sistema sugebės išversti OpenMP kodą į CUDA technologijai pritaikytą kodą arba atvirkščiai. Numatoma, kad programuotojas galės sistemai pateikti C/C++ kalbos failus arba suarchyvuotus projektus, kuriuos sistema pati išskleis bei analizuos.

Sistemą numatoma kurti dviem etapais – sukurti vertyklę, kaip terminale veikiančią programą, bei įgyvendinti ją naudoti mokančią grafinę vartotojo sąsają. Grafinėje vartoto sąsajoje vartotojams būtų paprasčiau bendrauti su vertyklė nei per komandinę eilutę – būtų galima lengviau įkelti projekto failus, keisti konvertavimo parametrus bei parsisiųsti sukonvertuotą kodą. Taip pat sistema turės galimybę peržiūrėti vertėjo palaikomą OpenMP bibliotekos ir CUDA technologijos konvertavimo funkcionalumą – bus galima visas palaikomas funkcijas peržiūrėti atskirtai, arba įkelti savo projektą ir patikrinti ar vertėjas palaiko projekte naudojamas kompiliatoriaus direktyvas ar naudojamas funkcijas.

2.4. Įgyvendinimo problemos

Kuriant sistemą, pagrindinis uždavinys yra pritaikyti vienos programos kodą kitos kalbos kompiliatoriui. Siekiant tai įgyvendinti yra susiduriama su tokiomis problemomis kaip

funkcijų veikimo atpažinimas vienoje ir kitoje kalboje, programos kodo analizė, atminties išskyrimas bei atlaisvinimas ir kita.

2.4.1. Kodo įkėlimo problema

Naudojat grafinę sąsają kodui įkelti yra susiduriama su problema, kad įkeliamas kodas dažniausiai nebus vienas failas, todėl dažniausiai bus keliami projektai suarchyvuoti tam tikra biblioteka, pavyzdžiui zip, rar, 7zip ar tar.gz. Įkeltus projektus sistema turės gebėti išarchyvuoti bei suarchyvuoti jau sugeneruotą kodą.

2.4.2. Kodo atpažinimo problema

Tai pat labai aktuali problema yra kaip atpažinti kodą tam tikrai kalbai [9]. Skirtingoms technologijoms yra naudojamas kitoks atminties valdymas, kitoks programavimo stilius. Pavyzdžiui, nagrinėjant programą, kuri atskirose gijose sudeda dviejų masyvų (a ir b sveikųjų skaičių masyvai) skaičius ir atsakymą priskiria trečiajam c masyvui. Tačiau programos savo struktūra yra labai skirtingos (žiūrėti 1 lentelę).

1 lentelė. OpenMP ir CUDA kodo palyginimas

Kodas naudojantis OpenMP

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 5 //gijų skaičius

int main(){
   int a[N], b[N], c[N];
   //Masyvų užpildymas pradiniais duomenimis
   for (int i = 0; i < N; i++){
     a[i] = i,
     b[i] = 1;
     c[i] = 0;
   }

   #pragma omp parallel
   for (int i = 0; i < N; i++){
   //skaičiuoja kiekviena gija lygiagrečiai
     int sum = a[i] + b[i];
     #pragma omp critical
     c[i] = sum;
   }
   //rezultatų spausdinimas
   for (int i = 0; i < N; i++)
   printf("%d + %d = %d\n", a[i], b[i], c[i]);

   return 0;
}
```

Kodas naudojantis CUDA

```
   #include <cuda.h>
   #include <stdio.h>
   #include <stdlib.h>

   #define N 5 //gijų skaičius
   //branduolio funkcija
   __global__ void add(int *a, int *b, int *c){
               int tID = blockIdx.x;
   //skaičiuoja kiekviena gija lygiagrečiai
               if (tID < N)
               c[tID] = a[tID] + b[tID];
   }

int main(){
   int a[N], b[N], c[N];
   int *dev_a, *dev_b, *dev_c;
   //Atminties išskyrimas CUDA įrenginyje
   cudaMalloc((void **)&dev_a, N*sizeof(int));
   cudaMalloc((void **) &dev_b, N*sizeof(int));
   cudaMalloc((void **)&dev_c, N*sizeof(int));
   //Masyvų užpildymas pradiniais duomenimis
   for (int i = 0; i < N; i++){
               a[i] = i;
               b[i] = 1;
        c[i] = 0;
   }
           //duomenų nukopijaviams į GPU
   cudaMemcpy(dev_a, a, N*sizeof(int),
cudaMemcpyHostToDevice);
   cudaMemcpy(dev_b, b, N*sizeof(int),
cudaMemcpyHostToDevice);
//gijų paleidimas
   add << <N, 1 >> >(dev_a, dev_b, dev_c);
   //duomenų kopijavimas į CPU
  cudaMemcpy(c, dev_c, N*sizeof(int),
cudaMemcpyDeviceToHost);
   //rezultatų spausdinimas
   for (int i = 0; i < N; i++){
               printf("%d + %d = %d\n", a[i], b[i],
c[i]);
   }
   return 0;
}
```

Kaip matome iš programų kodo pavyzdžių, CUDA programa yra gerokai ilgesnė nei OpenMP programa. CUDA programą sudaro šie pagrindiniai veiksmai:

- Atminties išskyrimas CUDA įrenginiui (GPU);
- Atminties nukopijavimas į GPU;
- Gijų paleidimas, naudojant __global__ (brandulio) funkciją;
- Atminties parsikopijavimas iš GPU į pagrindinę atmintį;
- GPU atminties atlaisvinimas;
- Rezultatų atspausdinimas.

Atlikus šiuos veiksmus programa baigia darbą.

Nors OpenMP programa yra trumpesnė, nes nereikia atskirai išskirti atminties, tačiau ji yra sudėtingesnė dėl naudojamų kompiliatoriaus direktyvų, kurios prasideda mėlynai paryškintu žodžiu „#pragma“. Kiekviena jų turi tam tikrą prasmę. Visos kompiliatoriaus direktyvos yra labai aiškiai ir išsamiai aprašytos OpenMP dokumentacijoje. Šiuo atveju OpenMP „#pragma omp parallel“ reiškia, kad kodas bus vykdomas lygiagrečiai, o „#pragma omp parallel critical“ pažymi, kad kodą po šia direktyva gijos negali vykdyti visos iš karto, turi vykdyti kiekviena atskirai, nes c masyvas negali būti bendrai naudojamas.

Apibendrinus šiuos kodo pavyzdžius, aiškiai galima suprasti pagrindinius OpenMP ir CUDA skirtumus. Juos įvertinus matyti, kad kuriant sistemą reikės:

1. Išanalizuoti OpenMP direktyvas, funkcijas.
2. Išanalizuoti CUDA funkcijas.
3. Išanalizuoti CUDA atminties valdymą.
4. Sukurti algoritmą, kuris atpažintų kode naudojamas funkcijas ar direktyvas ir sugeneruotų kodą tai technologijai, į kurią yra verčiama pradinė programa.

2.5. Išvados

1. Naudojant programinę įrangą, sugebančią konvertuoti jau sukurtą produktą į kitai technologijai pritaikytą projektą būtų galima sutaupyti daug laiko bei pinigų, siekiant pasamdyti tam tikrą technologiją išmanančius programuotojus.
2. Nors yra panašių egzistuojančių sistemų, tačiau kuriama sistema su galimybėmis, tokiomis kaip konvertavimas iš OpenMP į CUDA, grafine vartotojo sąsaja, įkelto projekto patikrinimu, išsiskiria iš savo konkurenčių.

3. PROJEKTAVIMAS

3.1. Apribojimai sprendimui

Sistema remsis OpenMP ir CUDA SDK bibliotekomis bei vykdomosiomis programomis. Kuriama sistema sąveikaus su įvairiomis techninėmis bei programinėmis sistemomis.

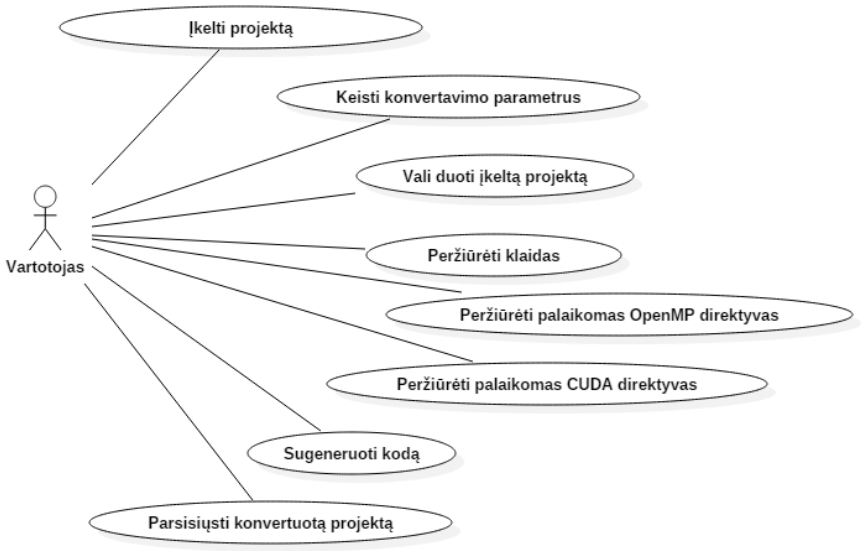
Techninės:

- Procesorius – Intel Celeron 1,3 Ghz arba galingesnis;
- Operatyvinė atmintis – 200 MB DDR3;
- Disko vieta sistemai – 100 MB;
- Nvidia vaizdo plokštė palaikanti CUDA technologiją.

Programinės:

- Interneto naršyklė, palaikanti HTML5 ir CSS3 standartus;
- Tomcat 7 serveris;
- Linux operacinė sistema;
- Java 7 arba Java 8 JDK.

3.2. Panaudos atvejai



2 pav. Panaudos atvejų diagrama

2 lentelė. Projekto įkėlimo panaudos atvejis

PANAUDOJIMO ATVEJIS:	Įkelti projektą
Tikslas:	Įkelti projektą konvertavimui .
Aktoriai:	Vartotojas
Prieš sąlygos:	Vartotojas turi projektą ir nori jį pakeisti.
Po sąlygos:	Sėkmingai įkeliamas projektas.
Sužadinimo sąlygos:	Vartotojas nori įkelti projektą konvertavimui.
Pagrindinis scenarijus:	Vartotojas bando įkelti projektą, spaudžia mygtuką įkelti.
Alternatyvūs scenarijai:	Esant klaidai, parodomas informuojantis pranešimas.

3 lentelė. Konvertavimo parametrų keitimo panaudos atvejis

PANAUDOJIMO ATVEJIS:	Keisti konvertavimo parametrus
Tikslas:	Nustatyti parametrus konvertavimui.
Aktoriai:	Vartotojas
Prieš sąlygos:	Vartotojas žino į kokią technologiją nori konvertuoti projektą.
Po sąlygos:	Pakeičiami parametrai.
Sužadinimo sąlygos:	Vartotojas nori pakeisti konvertavimo parametrus.
Pagrindinis scenarijus:	Vartotojas sužymi jam aktualius parametrus sistemoje.
Alternatyvūs scenarijai:	Sistema informuoja vartotoją apie klaidingus pasirinktus parametrus, kurie netinka tai technologijai. Esant klaidai, parodomas informuojantis pranešimas.

4 lentelė. Įkeliamo projekto vali davimo panaudos atvejis

PANAUDOJIMO ATVEJIS:	Validuoti įkeltą projektą
Tikslas:	Patikrinti ar įkeltame projekte nėra klaidų
Aktoriai:	Vartotojas
Prieš sąlygos:	Vartotojas nori patikrinti ar sistema supras jo projektą.
Po sąlygos:	Patikrintas projektas.
Sužadinimo sąlygos:	Vartotojas nori patikrinti ar jo projekte yra konvertavimui trukdančių klaidų.
Pagrindinis scenarijus:	1. Vartotojas sėkmingai įkelia projektą. 2. Sistema tikrina įkeltą projektą – bando jį sukompiliuoti tam tikrai technologijai, aptinka klaidas ir jeigu jų yra parodo vartotojui.
Alternatyvūs scenarijai:	Informuojama apie klaidas arba parodo pranešimą, kad įkėlimas sėkmingas. Esant klaidai, parodomas informuojantis pranešimas.

5 lentelė. Klaidų peržiūrėjimo panaudos atvejis

PANAUDOJIMO ATVEJIS:	Peržiūrėti klaidas
Tikslas:	Peržiūrėti klaidas
Aktoriai:	Vartotojas
Prieš sąlygos:	Turi būti įvykusi kokia nors klaida.
Po sąlygos:	Vartotojas peržiūri klaidų žurnalą.
Sužadinimo sąlygos:	Pradiniame projekte arba konvertavimo metu gaunama klaidų arba įspėjamųjų pranešimų.
Pagrindinis scenarijus:	1. Vartotojas paspaudžia mygtuką peržiūrėti žurnalą. 2. Vartotojas žiūri neseniai įvykusias klaidas žurnale.
Alternatyvūs scenarijai:	Esant klaidai, parodomas informuojantis pranešimas.

6 lentelė. OpenMP direktyvų peržiūros panaudos atvejis

PANAUDOJIMO ATVEJIS:	Peržiūrėti palaikomas OpenMP direktyvas
Tikslas:	Peržiūrėti OpenMP direktyvas
Aktoriai:	Vartotojas
Prieš sąlygos:	Vartotojas nori peržiūrėti ką sistema supranta apie OpenMP.
Po sąlygos:	Vartotojas išsiaiškina ką gali naudoti savo projekte.
Sužadinimo sąlygos:	Vartotojas nori peržiūrėti palaikomas OpenMP direktyvas.
Pagrindinis scenarijus:	1. Vartotojas spaudžia mygtuką peržiūrėti OpenMP palaikymą. 2. Sistema iš duomenų bazės paima įgyvendintas OpenMP direktyvas bei galimus parametrus.
Alternatyvūs scenarijai:	Esant klaidai, parodomas informuojantis pranešimas.

7 lentelė. CUDA direktyvų peržiūros panaudos atvejis

PANAUDOJIMO ATVEJIS:	Peržiūrėti palaikomas CUDA funkcijas
Tikslas:	Peržiūrėti CUDA direktyvas
Aktoriai:	Vartotojas
Prieš sąlygos:	Vartotojas nori peržiūrėti ką sistema palaiko apie CUDA technologijos funkcijas.
Po sąlygos:	Vartotojas išsiaiškina ką gali naudoti savo projekte.
Sužadinimo sąlygos:	Vartotojas nori peržiūrėti palaikomas CUDA funkcijas.
Pagrindinis scenarijus:	1. Vartotojas spaudžia mygtuką peržiūrėti CUDA palaikymui. 2. Sistema iš duomenų bazės paima įgyvendintas CUDA direktyvas bei galimus parametrus.
Alternatyvūs scenarijai:	Esant klaidai, parodomas informuojantis pranešimas.

8 lentelė. Kodo generavimo panaudos atvejis

PANAUDOJIMO ATVEJIS:	Sugeneruoti kodą
Tikslas:	Sugeneruoti kodą, pritaikytą CUDA arba OpenMP technologijai.
Aktoriai:	Vartotojas
Prieš sąlygos:	Turi būti įkeltas pradinis projektas. Įkelto projekto patikrinimas dėl klaidų sėkmingas.
Po sąlygos:	Sistema sukuria naują projektą, sugeneruoja kodą priešingai technologijai.
Sužadinimo sąlygos:	Vartotojas nori sugeneruoti kodą priešingai technologijai.
Pagrindinis scenarijus	1. Vartotojas spaudžia mygtuką generuoti. 2. Sistema sėkmingai sugeneruoja kodą.
Alternatyvūs scenarijai:	Esant blogai nurodytiems parametrams, sistema siūlo patikrinti duomenis ir bandyti dar kartą. Jeigu sistemoje neužtenka vietos sukurti naują projektą, sistema praneša apie klaidą. Esant kitoms klaidoms, parodomas informuojantis pranešimas.

9 lentelė. Projekto parsisiuntimo panaudos atvejis

PANAUDOJIMO ATVEJIS:	Parsisiųsti konvertuotą projektą
Tikslas:	Parsisiųsti konvertuotą projektą
Aktoriai:	Vartotojas
Prieš sąlygos:	Kodo generavimas buvo sėkmingas.
Po sąlygos:	Vartotojas turi parsisiųstą konvertuotą projektą.
Sužadinimo sąlygos:	Vartotojas nori parsisiųsti sugeneruotą projektą.
Pagrindinis scenarijus	1. Vartotojas paspaudžia mygtuką parsisiųsti. 2. Atsisiunčiamas failas.
Alternatyvūs scenarijai:	Esant blogai nurodytam failo vardui, sistema informuojama apie klaidą.

3.3. Funkciniai reikalavimai

10 lentelė. Konvertavimo progreso rodymas

Reikalavimas #:	1	Reikalavimo tipas:	9	Įvykis/PA #:	7
Aprašymas:	Vartotojas dirbdamas su sistema turi turėti galimybę matyti progresą.				
Pagrindimas:	Reikalinga, kad vartotojas galėtų suprasti, kiek laiko užtruks programos konvertavimas.				
Šaltinis:	Užsakovas				
Tikimo kriterijus:	Sistema rodo kiek užtruks konvertavimas.				
Užsakovo patenkinimas:	4	Užsakovo nepatenkinimas:	4		
Priklausomybės:	Nėra	Konfliktai:	Nėra		
Papildoma medžiaga:	Nėra				
Istorija:	Užregistruotas 2014-12-11				

11 lentelė. Parametrų nustatymas

Reikalavimas #:	2	Reikalavimo tipas:	9	Įvykis/PA #:	2
Aprašymas:	Vartotojas, labiau išmanantis CUDA ar OpenMP technologijas turi galėti nusistatyti papildomus parametrus.				
Pagrindimas:	Reikalinga, kad vartotojui nereiktų rankiniu būtu nustatinėti nuo technologijos priklausančius parametrus.				
Šaltinis:	Užsakovas				
Tikimo kriterijus:	Sistema leidžia nustatyti specialius parametrus.				
Užsakovo patenkinimas:	4	Užsakovo nepatenkinimas:	4		
Priklausomybės:	Nėra	Konfliktai:	Nėra		
Papildoma medžiaga:	Nėra				
Istorija:	Užregistruotas 2014-12-11				

3.4. Nefunkciniai reikalavimai

3.4.1. Reikalavimai sistemos išvaizdai

3.4.1.1. Išvaizda

Vartotojo sąsaja turi būti:

1. lengvai skaitoma – tam, kad nevaržytų naujų sistemos vartotojų;

2. neįkyri, nereikalaujanti daugkartinių patvirtinimų – tam, kad nevargintų daug ir ilgesnį laiką sistema besinaudojančių vartotojų;

3.4.1.2. Stilius

Visoje vartotojo sąsajoje turi būti vientisa: naudojamas vienas stilius, t.y. vienodi šriftai, spalvos, elementų apipavidalinimas. Tai pat turi būti patrauklios ir profesionalios išvaizdos – tam, kad dažnai sistema besinaudojantiems vartotojams būtų malonu dirbti.

3.4.2. Reikalavimai panaudojamumui

3.4.2.1. Naudojimosi paprastumas

Naudojimasis sistema turi būti greitai perprantamas, be apsimokymo (90% sėkmingas panaudojimas pirmu bandymu).

3.4.2.2. Vartotojui skirtos savybių ir kalbos konfigūravimo priemonės

Sistema turi būti tobulinama, vis palaikant daugiau CUDA ir OpenMP konvertuojamų funkcijų. Vartotojas turi galėti keisti konvertavimo parametrus.

3.4.2.3. Mokymosi reikalavimai

Nėra.

3.4.2.4. Suprantamumas ir mandagumas

Sistemos sąsaja turi būti pritaikyta anglų kalbą suprantamiems vartotojams. Turi būti naudojami OpenMP ir CUDA technologijose naudojami terminai.

3.4.2.5. Prieinamumas neįgaliesiems

Turi būti galimybė padidinti vartotojo sąsajos elementus ir teksto šrifto dydį.

3.4.3. Reikalavimai vykdymo savybėms

3.4.3.1. Užduočių vykdymo greitis

Sistema turi turėti greitą atsaką į vartotojų veiksmus. Vidutinis operacijos atlikimo laikas iki 3s.

3.4.3.2. Darbo saugos reikalavimai

Nėra.

3.4.3.3. Reikalavimai tikslumui

Nėra.

3.4.3.4. Patikimumas ir pasiekiamumas

Nėra.

3.4.3.5. Atsparumas trukdžiams

Nėra.

3.4.3.6. Reikalavimai apdorojamų duomenų apimtims

Sistema turi turėti bent 5 GB laisvos vietos, laikiniems failams ir .zip archyvams laikyti.

3.4.3.7. Reikalavimai išplečiamumui

Sistema turi būti kuriama iš atskirų modulių, kad būtų galima lengvai praplėsti jos galimybes.

3.4.3.8. Reikalavimai produkto ilgaamžiškumui

Tikimasi, kad sistema bus naudojamasi bent 4 metus.

3.4.4. Reikalavimai veikimo sąlygoms

3.4.4.1. Numatoma fizinė aplinka

Nėra.

3.4.4.2. Reikalavimai darbui su gretimomis sistemomis

Nėra.

3.4.4.3. Reikalavimai sist. platinimo/gamybos formatui

Sistema turėtų būti platinama kaip .deb ar .tar.gz paketas, siekiant užtikrinti sklandų sistemos įdiegimą Linux aplinkoje.

3.4.4.4. Reikalavimai leidybos procesui

Sistemos įdiegimo pakete turi būti įtrauktas ir sistemos kodas. Kartą per du metus bus išleidžiama nauja sistemos versija.

3.4.5. Reikalavimai sistemos priežiūrai

3.4.5.1. Sistemos aptarnavimas

Sistemos įdiegimo procesas turi vykti automatiškai ir užtrukti ne ilgiau kaip 2 minutes.

3.4.5.2. Sistemos palaikymas

Sistema turės vartotojo gidą, kuriame bus aprašytas sistemos veikimas.

3.4.5.3. Perkėlimo į kitas platformas reikalavimai

Sistema turi veikti Linux operacinėje sistemoje.

3.4.6. Reikalavimai saugumui

3.4.6.1. Prieigos reikalavimai (teisės)

Administratoriaus teisės nebus reikalingos sistemos paleidimui.

3.4.6.2. Vientisumo (integralumo) reikalavimai

Nėra.

3.4.6.3. Reikalavimai privatumui

Sistema baigus darbą neturėtų saugoti laikinų darbui reikalingų failų.

3.4.6.4. Audito reikalavimai

Nėra.

3.4.6.5. Reikalavimai savisaugai nuo išorinių grėsmių
Nėra.

3.4.7. Kultūriniai-politiniai reikalavimai

3.4.7.1. Kultūriniai reikalavimai
Nėra.

3.4.7.2. Politiniai reikalavimai
Nėra.

3.4.8. Teisiniai reikalavimai

3.4.8.1. Įstatyminiai reikalavimai
Autorinės teisės į galutinį produktą priklausys jos kūrėjams.

3.4.8.2. Reikalavimai standartams
Nėra.

3.4.9. Atviri klausimai

Ar reikėtų sukurti vartotojo dokumentaciją?

Ar reikėtų sukurti demonstracinį video filmuką?

3.4.10. Egzistuojantys sprendimai

3.4.10.1. Prieinamos sistemos
Sistemų leidžiančių konvertuoti kodą iš OpenMP į CUDA arba atvirkščiai nėra sukurta, tačiau yra keletas panašių sistemų (žiūrėti ankstesnius skyrius).

3.4.10.2. Prieinami komponentai
Sistema naudos laisvai prieinamus OpenMP ir CUDA SDK.

3.4.10.3. Kopijuotini sprendimai
Galima pasinaudoti jau egzistuojančiais kompiliatoriais, tokiais kaip GCC, MAKE failai, siekiant užtikrinti panašias valdymo galimybes.

3.4.11. Problemos

3.4.11.1. Poveikis diegimo aplinkai
Sistema bus integruota su kitomis Linux operacinėje sistemoje veikiančiomis sistemomis, todėl jokios įtakos joms neturės.

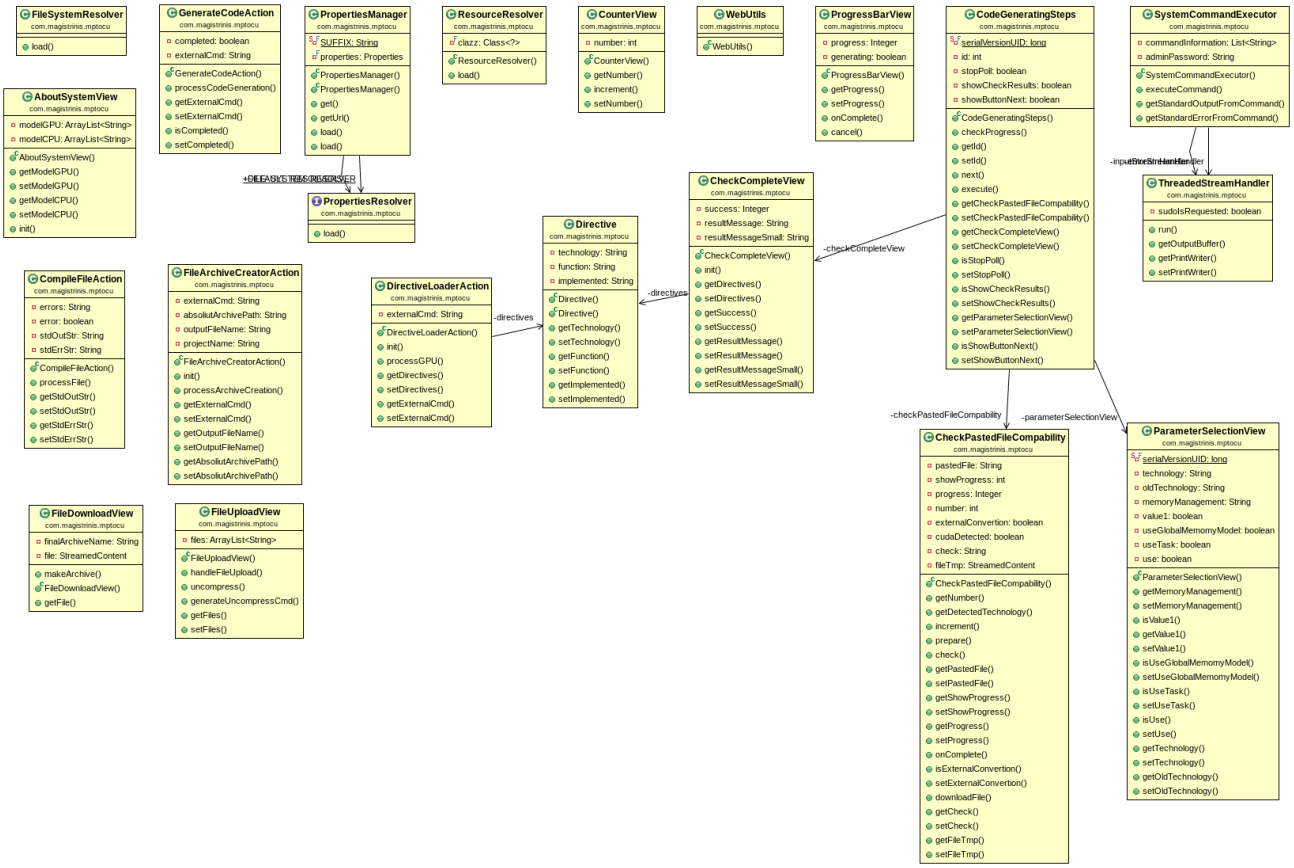
3.4.11.2. Poveikis esamoms sistemoms
Esamoms sistemos produktas įtakos neturės.

3.4.11.3. Probleminė vartotojų reakcija
Nėra.

3.4.11.4. Apribojimai diegimo aplinkoje
Nėra.

3.5. Sistemos architektūra

3.5.1. Grafinės sąsajos modulis



3 pav. Grafinės sąsajos modulio klasių diagrama

Grafinės sąsajos modulio pagrindą sudaro šios klasės:

- *Directive* - abstakčiai aprašo kalbos direktyvą kitamaisiais: technologija, funkcijos pavadinimas ir statusu ar galima pakeisti funkciją kita;
- *DirectiveLoaderAction* - klasė kuri užkrauna visas direktyvas sąrašo pavidalu;
- *SystemCommandExecutor* ir *ThreadedStreamHandler* suteikia galimybę vykdyti programas foniniam procese bei leidžia pasiimti stdout ir stderr srautus iš vykdomos programos;
- *FileDownloadView*, *FileUploadView*, *FileSysemResolver*, *ResourceResolver*, *PropertiesResolver*, *FileArchiveCreatorAction* aprašo bei vykdo veiksmus kurie yra reikalingi darbui su failų sistemą, įgalina failų parsisiuntimą, įkėlimą, archyvo sukūrimą iš keletos failų;
- *CodeGeneratingSteps*, *GenerateCodeAction*, *CompileFileAction*, *CheckPastedFileCompability* - atsakingos už logiką apdorojant kodo generavimo žingsnius bei kodo generavimo modulio iškvietimą.

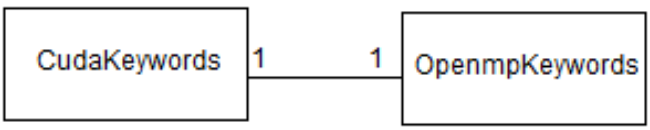
3.5.2. Kodo generavimo modulis

4 pav. Kodo generavimo modulio klasių diagrama

Kodo generavimo modulio pagrindą sudaro šios klasės:

- *LanguageKeyword* - abstrakčiai aprašo kalbos direktyvą kintamaisiais: technologija, funkcijos pavadinimas ir statusu ar galima pakeisti funkciją kita;
- *FileManager*, *LanguageKeywordLoader* - skirta atlikti veiksmams susijusiais su failų sistema - užkrauti visas LanguageKeyword, nuskaityti ar įrašyti informacijai į failus.
- *CheckCompability* - patikrina ar tam tikras perduotas failas gali būti konvertuotas pagal informaciją gautą iš LanguageKeyworLoader klasės.
- *GenerateCodeToOpenMP*, *GenerateCodeToCuda* bei *AbstarctCodeGenerator* - atlieka kodo generavimo procesą - analizuoja pradinį failą, kuria naujas konstrukcijas, saugoja sugeneruotą kodą į naujai sukurtą failą.

3.5.3. Duomenų vaizdas

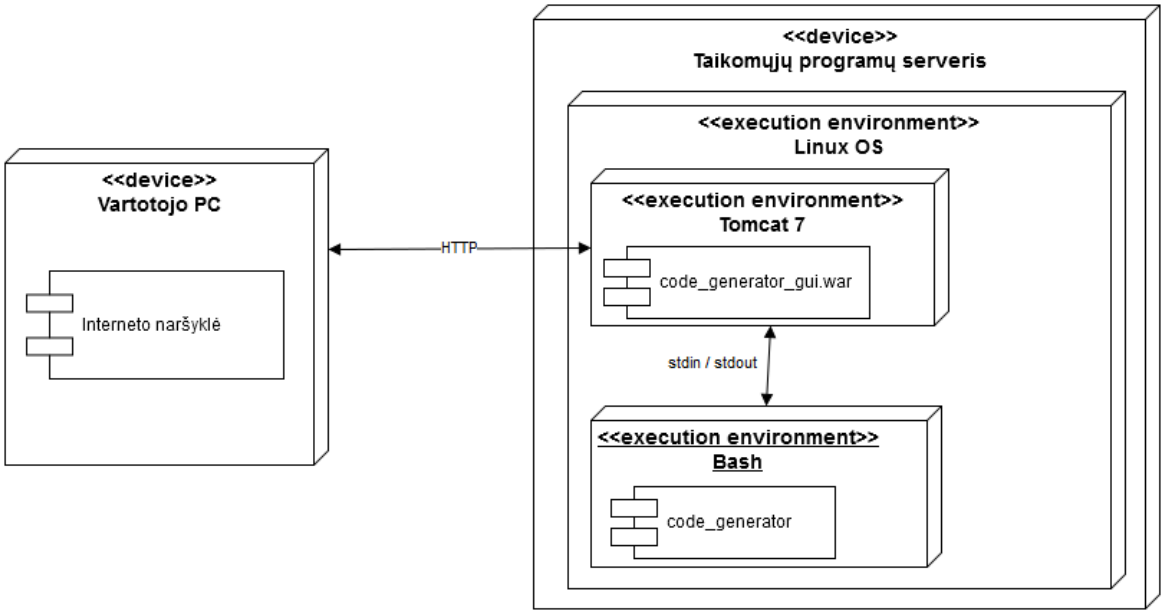


5 pav. Duomenų vaizdas

CudaKeywords – lentelė ar failas, apibūdinantis CUDA kalbos raktinius žodžius, jų aprašymą bei požymį ar realizuotas jo pakeitimas į OpenMP alternatyvą.

OpenmpKeywords – lentelė ar failas, apibūdinantis OpenMP kalbos raktinius žodžius, jų aprašymą bei požymį ar realizuotas jo pakeitimas į CUDA alternatyvą.

3.5.4. Išdėstymo vaizdas



6 pav. Sistemos išdėstymo vaizdas

Sistemos išdėstymą sudaro, kaip parodyta 6 paveikslėlyje, vartotojo įrenginys ir taikomųjų programų serveris. Vartotojo įrenginio (kompiuterio, planšetės ar išmaniojo telefono) operacinė sistema nėra svarbi, nes vartotojas kreipsis į serverį naudodamas interneto naršyklę. Bendravimas vyks HTTP protokolu, per grafinę sąsają. Tuo tarpu serveriui yra reikalinga Linux operacinė sistema, nes sistema yra priklausoma nuo keleto bibliotekų. Tomcat 7 vykdymo aplinka naudodama stdin ir stdout srautus keisis duomenimis su kodo generatoriaus posisteme ir atvaizduos rezultatus vartotojui.

3.6. Išvados

1. Projektavimo metu nustatyti sistemos apribojimai, funkciniai bei nefunkciniai reikalavai.
2. Realizuota sistemos architektūra, projektas išskaidytas į kodo generavimo bei grafinės sąsajos modulius.

4. NUOSEKLIŲ IR LYGIAGREČIŲ ALGORITMŲ TYRIMAS

4.1. Tyrimo tikslas

Atliekamo tyrimo tikslas yra nustatyti OpenMP ir CUDA technologijų teikiamus privalomus (funkcijas, struktūras, konfigūracijas, atminties tipą), kurie padėtų patobulinti sukurtą sistemą taip, kad pagal vartotojo įkeltą kodą būtų pasiekiamas geriausias išlygiagretinimo efektas [13] [14].

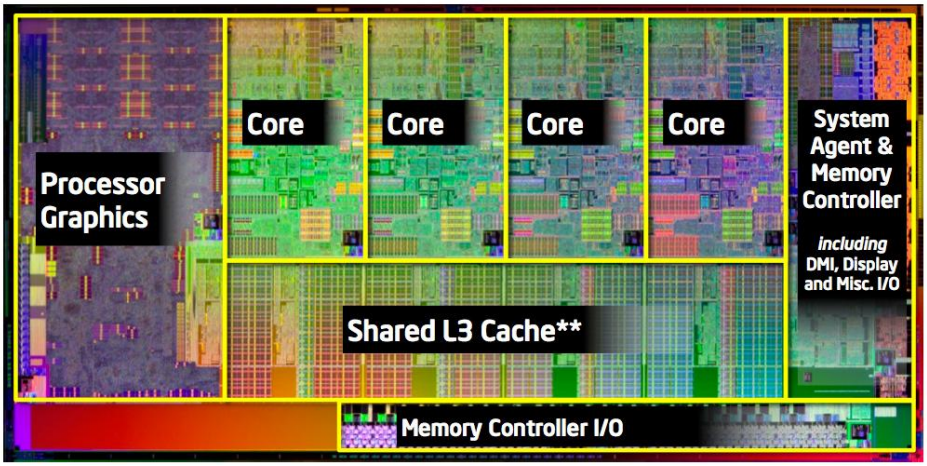
Tyrimui atlikti buvo pasirinkti algoritmai, kurie yra plačiai naudojami, reikalauja daugiau atminties bei parodo išlygiagretintų procesų privalumus. Toliau pateikiama lentelė, kuri trumpai apibūdina, kodėl buvo pasirinktas tam tikras algoritmas.

12 lentelė. Pasirinkti algoritmai tyrimui

Pavadinimas	**Tiriama technologijoje**	**Pasirinkimo priežastis**
Maksimumo radimas	OpenMP ir CUDA	Ieškant tam tikros reikšmės, dažniausiai duomenų tvarka neturi reikšmės.
Matricų daugyba	OpenMP ir CUDA	Matricos yra svarbios matematikoje, lygčių sprendimuose, vaizdų apdorojime, jų transformacijose. Išlygiagretinus šiuos veiksmus galima sutaupyti laiko bei resursų.
π apskaičiavimas	OpenMP ir CUDA	Skaičiuojant π ypač aktualus yra skaičiavimo tikslumas.
Rikiavimas	OpenMP ir CUDA	Rikiuojant duomenis, rikiavimo greitis dažnai priklauso nuo pradinės duomenų tvarkos.

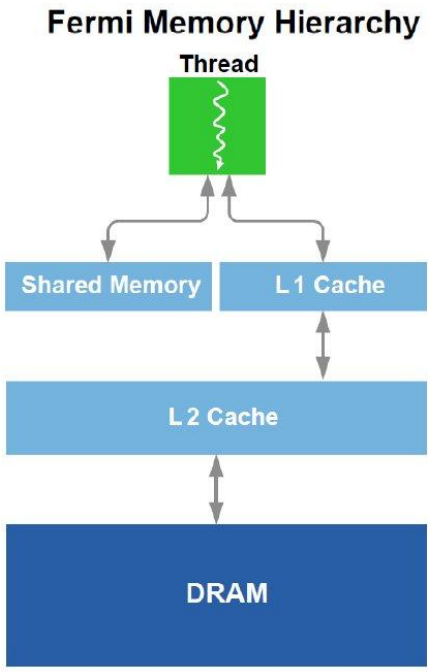
4.2. Naudojama techninė įranga

Tyrimui atlikti bus naudojamas vienas nešiojamas „ACER Aspire“ kompiuteris su antros kartos i7-2630QM procesoriumi, 8 GB RAM bei integruota „NVidia“ vaizdo plokšte „GT 540m“ 2GB, kuri pagaminta pagal „Fermi“ architektūrą.



7 pav. Intel i7 2nd gen. procesoriaus vidinė struktūra.

Kaip matome 7 paveikslėlyje, šis procesorius turi Intel kompanijos patentuotą „SmartCache“ technologiją, kuri įgalina visus 4 procesoriaus branduolius (dvi gijos per branduolį) dalintis L3 lygio spartinančiąja atmintimi (6 MB). Tai ypač yra aktualu naudojantis OpenMP technologija, nes ji geba panaudoti šią atmintį.



8 pav. Fermi architektūros atmintis

Kalbant apie „Nvidia“ kompanijos „Fermi“ architektūrą, kaip matome iš 8 paveikslėlio, ši vaizdo plokštė turi L1 bei L2 spartinančiąsias atmintines bei bendrą atmintį (angl. Shared Memory). Tai pat svarbu paminėti, kad kiekviena gija (angl. Thread) gali kreiptis į abi atmintis, tačiau kreipimasis į bendrą atmintį užima daug mažiau laiko, nei į L2 ar DRAM (globalią) atmintį. Tyrime naudojama vaizdo plokštė turi 128 KB L2 atminties, 48 KB bendros atminties (angl. Shared Memory), 16 KB L1 atminties bei 2 GB DRAM atminties.

4.3. Tyrimo kriterijai

Nuoseklių bei lygiagrečių algoritmų spartai palyginti bus naudojami spartinimo koeficientas ir algoritmo efektyvumo koeficientas.

4.3.1. Spartinimo koeficientas

Lygiagrečiojo algoritmo spartinimo koeficientu [15] vadinamas santykis $S_p = \frac{T_0}{T_p}$, įvertinantis pagreitėjimą, kuris pasiekiamas sprendžiant uždavinį lygiagrečiu algoritmu ir naudojant *p* procesorių. Čia T_p žymi laiką, per kurį duotas uždavinys išsprendžiamas lygiagrečiuoju algoritmu naudojant p procesorių. T_0 – laiką, per kurį tas pats uždavinys išsprendžiamas greičiausiu nuosekliu algoritmu. Idealiu atveju yra tikimasi, kad didėjant procesorių skaičiui, algoritmo spartinimo koeficientas didės tiesiškai.

4.3.2. Algoritmo efektyvumo koeficientas

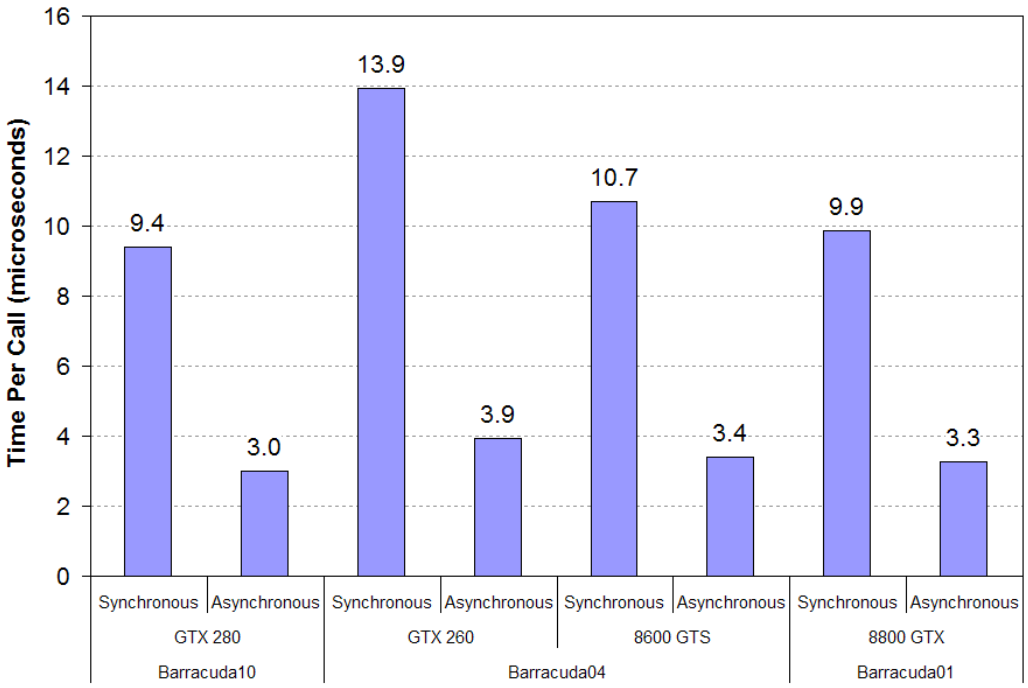
Algoritmo efektyvumo koeficientu E_p yra vadinamas algoritmo spartinimo koeficientas S_p ir naudojamų procesorių skaičiaus *p* santykis (1). Jis parodo, kokią dalį procesorių pajėgumo pasitelkiame spręsdami uždavinį duotuoju lygiagrečiuoju algoritmu.

$$E_p = \frac{S_p}{p} \qquad (1)$$

4.4. Išlygiagretinimo kaštai

Įvairių tyrimų teigimu, OpenMP ir CUDA technologijose pasiruošimas vykdyti išlygiagretintą kodą bei jo iškvietimas užima nemažai laiko ir šie kaštai yra skirtingi OpenMP ir CUDA technologijose.

Universiteto „Virginia“ atlikto tyrimo duomenimis [16], skiriasi sinchroninio ir asinchroninio CUDA branduolio vykdymo funkcijos kvietimo kaštai. Atlikto tyrimo metu buvo matuojamas vidutinis laikas, reikalingas paleisti tuščią branduolio funkciją ją daug kartų kviečiant. Tyrimas buvo atliekamas ant trijų skirtingų kompiuterių.



9 pav. „Virginia“ universiteto atlikto tyrimo duomenys. Laikas, reikalingas iškviesti tuščiai branduolio funkcijai.

Tyrimo rezultatuose teigiama, kad kvietimo laikas nepriklauso nuo kviečiamos branduolio funkcijos charakteristikų, tačiau priklauso nuo kvietimo būdo – ar reikalinga sinchronizacija ar ne, bei akcentuojama, kad asinchroninio kvietimo didžiąją dalį sudaro laikas, reikalingas pasiekti vaizdo plokštės tvarkyklę.

Tyrimuose, susisijusiame su OpenMP technologija [17][18] teigimu, įvairių OpenMP konstrukcijų panaudojimo kaštai priklauso nuo naudojamo kompiliatoriaus. Yra pastebima tendencija, kad naudojant GCC kompiliatorių yra sunaudojama daugiau procesoriaus ciklų nei naudojant Intel kompiliatorių. Taip pat nurodoma, kad didžiausią įtaką turi 4 faktoriai (žiūrėti 13 lentelę), kurie tyrime nėra išmatuoti, tik aprašomi. Tai pat nurodoma, kad įtakos turi ir neteisingai išlygiagretintas kodas, kaip kad labai mažos apimties ciklai, kurių išlygiagreinimas suteikia per mažai naudos, kad jie būtų vykdomi lygiagrečiai.

13 lentelė. OpenMP naudojimo kaštų faktoriai

Faktorius	**Aprašymas**
Gijų bibliotekos paleisties kaštai	Vienkartinis laiko tarpas, įskaičiuojamas kartu su programos paleisties laiku.
Pačių gijų paleidimo kaštai	Vienkartinis laiko tarpas, reikalingas sukurti gijoms.
Kiekvienos gijos kaštai	Laikas praleistas paskirstyti darbą kiekvienai gijai.
Gijų užrakinimo valdymo kaštai	Laikas praleistas valdant užraktus kritinėse sekcijose.

4.5. Klasikiniai nuoseklūs ir lygiagretūs skaičiavimo algoritmai

4.5.1. Matricų daugyba

Matricos [19] bei veiksmai su jomis yra labai dažnai naudojami įvairiuose skaičiavimo uždaviniuose. Tai viena iš sričių, kur ypač gali būti panaudojami lygiagretaus programavimo privalomai.

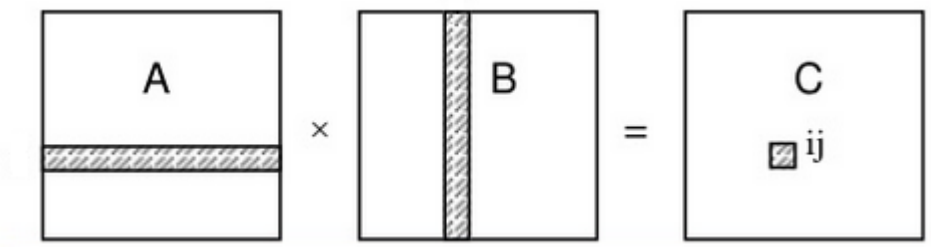
Matricą sudaro eilutės ir stulpeliai. Jeigu matricą sudaro *m* eilučių ir *n* stulpelių, ji vadinama [*m* × *n*] dydžio matrica. *i - osios* eilutės ir *j - otojo* stulpelio sankirtoje esantis elementas paprastai žymimas a_{ij}. Sakoma, kad matrica *A* yra 2×4 dydžio, jeigu ją sudaro dvi eilutės (*m* = 2) ir keturi stulpeliai (*n* = 4).

Matricas galima dauginti tarpusavyje, jeigu matricos yra suderintos. Tai reiškia, kad matricą $A = (a_{ik})$, kurios dydis – [*m* × *s*], galima dauginti iš tokios matricos $B=(b_{kj})$, jeigu eilučių skaičius sutampa su matricos A stulpelių skaičiumi – matricos B dydis turi būti [*s* × *n*]. Sudauginus *A* ir *B* matricas, gaunama [*m* × *n*] formato matrica $C = (c_{ij})$, t.y $A_{[m\times s]} \times B_{[s\times n]} = C_{[m\times n]}$. Kiekvienas matricos *C* elementas c_{ij} yra apskaičiuojamas pagal formulę:

$$c_{ij} = \sum_{k=1}^{s} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \ldots + a_{is} b_{sj}$$

čia $1 \le i \le m$ ir $1 \le j \le n$.

Vykdant matricų daugybą lygiagrečiai, kiekviena gija randa C_{ij} elementą daugindama matricos A_i eilutę su matricos B_j eilute, kaip parodyta 10 paveikslėlyje.



10 pav. Vienos gijos darbas matricų daugybos metu.

Nuoseklus ir lygiagretus matricų daugybos programų kodai pateikiami prieduose. Algoritmai parašyti remiantis literatūra [21][22][23].

4.5.2. π apskaičiavimo algoritmas

Vienas iš dažnai sprendžiamų uždavinių matematikoje yra π apskaičiavimas su kuo daugiau skaitmenų po kablelio. Šiam uždaviniui spręsti matematikai yra išvedę daug įvairių formulių [20]. Dažnai π apskaičiavimui yra naudojama integralo $\int_0^1 \frac{4}{1+x^2}\,dx$ skaičiavimas rėžiuose nuo 0 iki 1, kuo mažesniu žingsniu.

Skaičiuojant šiuo būdu, kiekviena gija suskaičiuoja tarpines integralo sumas bei perduoda jas pagrindinei (0 gijai), kuri sudeda visas tarpines sumas. Tokiu būdu yra gaunama apytikslė π reikšmė.

Nuoseklus ir lygiagretus šios sekos apskaičiavimo algoritmų programų kodai pateikiami prieduose. Algoritmai parašyti remiantis literatūra [24][25].

4.5.3. Maksimumo radimas

Kitas dažnai programavime sutinkamas uždavinys yra didžiausios reikšmės paieška masyve, vektoriuje ar sąraše, kai pradiniai duomenys yra nerikiuota skaičių seka. Šis uždavinys gali būti išspręstas keliais būdais. Pavyzdžiui, visų reikšmių išrikiavimas mažėjimo tvarka ir pirmo elemento reikšmės paėmimas ar pereinant per visus masyvo elementus lyginant dabartinę reikšmę su išsaugota pradine maksimumo reikšme.

Dažniausiai šis uždavinys yra sprendžiamas nuosekliai – nėra panaudojama nei OpenMP nei CUDA technologija, nes dažnai yra siekiama kuo greičiau atlikti užduotį, tačiau nėra pasirūpinama greitaveika.

Ieškant didžiausios reikšmės lygiagrečiai, kiekviena gija randa savo didžiausią reikšmę tam tikruose masyvo rėžiuose (1-gija rėžiuose [0..10], 2-gija – [11, 20] ir t.t.) bei perduoda gautą reikšmę pagrindinei 0 gijai, kuri palygina visas gautas reikšmes su didžiausia rasta bei atnaujina didžiausią rastą reikmę.

Nuoseklus ir lygiagretus maksimumo radimo algoritmų programų kodai, kai pereinama per visus elementus ieškant didžiausios masyvo reikšmės, pateikiami prieduose. Algoritmai parašyti remiantis literatūra [26][27].

4.5.4. Rikiavimas

Būna atvejų, kai programuojant reikia surikiuoti turimus duomenis didėjimo ar mažėjimo tvarka. Nors šiuolaikinės programavimo kalbos, tokios kaip Java 8 ar C++ 11 jau turi integruotas bibliotekas su aprašytais bei optimizuotais rikiavimo algoritmais, tačiau kartais gali reikėti naudoti savaip modifikuotą algoritmą, pavyzdžiui rikiuojant reikšmes disko atmintyje.

Kita rikiavimo algoritmų problema – jų įvairovė. Renkantis rikiavimo algoritmą svarbu atsižvelgti į keletą faktorių, tokių kaip:

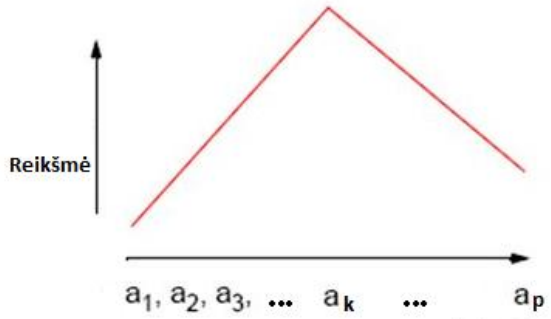
- skaičių tipas – sveikieji, realieji, simboliniai;
- duomenų imties dydis;
- turimų duomenų tvarka – atsitiktinai išsidėstę ar beveik išrikiuoti;
- ar visi norimi išrikiuoti duomenys telpa kompiuterio atmintyje;
- ar turimas kompiuteris turi „Nvidia“ vaizdo plokštę ar ne.

Tai pat svarbu paminėti, kad rikiavimo algoritmai gali būti paremti reikšmių palyginimu, kaip kad „Bubble Sort“, „Selection Sort“ bei paremti parametrais (vienodi raktai) – „Bucket Sort“, „Cout Sort“.

Apžvelgus šiuos faktorius galima numanyti, kad turint mažai sveikų skaičių galime naudoti – „Bubble Sort“ rikiavimo algoritmą, turint daug – „Quick Sort“, „Merge Sort. Turint labai daug duomenų galbūt geriau yra naudoti lygiagretų rikiavimo algoritmą. Įvertinus šiuos faktorius galima lengviau pasirinkti rikiavimo algoritmą, bei atsižvelgus į turimų duomenų imties dydį galima nuspręsti ar pasirinksime lygiagretų ar nuoseklų rikiavimo algoritmą.

Šiame tyrime bus bandomas „Bitonic Sort“ algoritmas, kuris priklauso rikiavimo tinklų grupei (angl. sorting networks). Duomenų sekos ir krypties palyginimai yra numatomi iš anksto ir nėra priklausomi nuo pačios sekos. Šis algoritmas remiasi bitonine seka a = (a_1, a_2, ..., a_p), susidedančia iš *p* skaičių. Ši seka yra vadinama bitonic seka tada ir tik tada, jeigu:

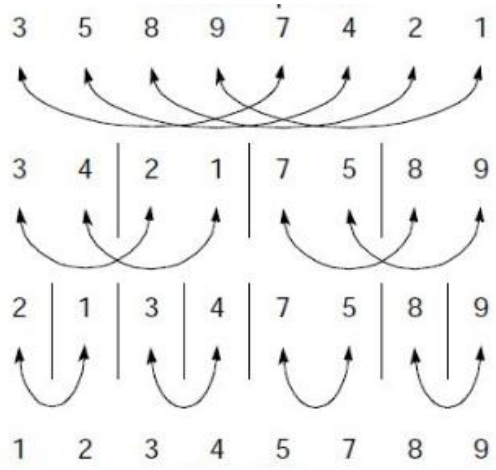
1. $a_1 <= a_2 <= ... a_k >= ... a_p$, kur *k* yra $1<k<p$;
2. $a_1 >= a_2 >= ... >= a_k <= ... <a_p$, kur *k* yra $1<k<p$;
3. a seka gali būti padalinta į dvi dalis taip, kad būtų galima sudaryti 1 arba 2 atvejus.



11 pav. Bitoninė seka

Algoritmo veikimo principas (žiūrėti 12 paveikslėlį):

1. Rikiuoti galime tik tokią seką, kurios ilgis n = 2^k (k yra teigiamas sveikas skaičius). Tokiu būdu, galime seką padalinti į daugiau nei vieną poaibį po 2 elementus.
2. Rūšiuojam kairę (mažesnę) pusę didėjimo tvarka ir kitą dešinę (didesnę) pusę mažėjančia tvarka.
3. Bitoninei sekai atliekame suliejimą taip, kad kairėje pusėje būtų mažesni elementai, o dešinėje didesni elementai. Atliekame palyginimo ir sukeitimo veiksmus.
4. Rekursyviai vykdome bitoninį suliejimą (angl. Bitonic merge) kiekvienai pusei, kol visi elementai yra išrikiuojami.



12 pav. „Bitonic sort“ algoritmo iliustracija.

Rikiavimo metu, kiekviena gija atlieka palyginimo ir sukeitimo vietomis veiksmus, kuriuos iliustruoja juodos rodyklės 12 paveikslėlyje. Nuoseklaus ir lygiagretaus „Bitonic Sort“ rikiavimo algoritmų programų kodai pateikiami prieduose. Algoritmai parašyti remiantis literatūra [28][29][30][31].

4.6. Eksperimentų rezultatai

Atliekant eksperimentus visuose algoritmuose buvo generuojami atsitiktiniai skaičių rinkiniai. Kiekvieną kartą vykdant programas, gijos turėjo vienodą duomenų rinkinį (skaičių padėtis sekoje buvo vienoda) bei keičiant gijų skaičių buvo naudojami tie patys duomenų rinkiniai.

4.6.1. Programos konvertavimo eksperimentas

Siekiant įvertinti praeitame pusmetyje sukurtą sistemą „OpenMP ir CUDA lygiagrečių programų tarpusio konvertavimo sistema“, buvo atliktas eksperimentas, kuriame buvo bandoma konvertuoti sukurtas didžiausios reikšmės radimo ir matricų daugybos programas, pritaikytas OpenMP technologijai, konvertuoti CUDA technologijai.

Bandant pakeisti didžiausios reikšmės radimo programą CUDA technologijai, susidurta su problema, kad pakeistas programos kodas neatlieka pageidautinos funkcijos. Konvertuotos CUDA programos atveju, kiekviena gija eita per masyvą, tačiau nepasidalinta darbo tarpusavyje, kaip kad yra OpenMP atveju.

14 lentelė. Maksimumo paieškos konvertavimo rezultatai

OpenMP programos fragmentas	**Konvertuota CUDA programa**

#pragma omp parallel
{
int my_value = 0;
#pragma omp for
for (int f = 0; f < N; f++) {
if (v[f] > my_value)
my_value = v.[f];
}
#pragma omp critical
{
if (my_value > maxa)
maxa = my_value;
}
}
	__global__ void doKernel(int *v, int N)
	{
	int my_value = 0;
	for (int f = 0; f < N; f++) {
	if (v[f] > my_value)
	my_value = v[f];
	}
	{
	if (my_value > maxa)
	maxa = my_value;
	}
	}

Keičiant matricų daugybos programą buvo nesklandumų – buvo netinkami masyvo indeksai. Juos pakeitus konvertuota programa veikė tinkamai.

15 lentelė. Matricų daugybos konvertavimo rezultatai

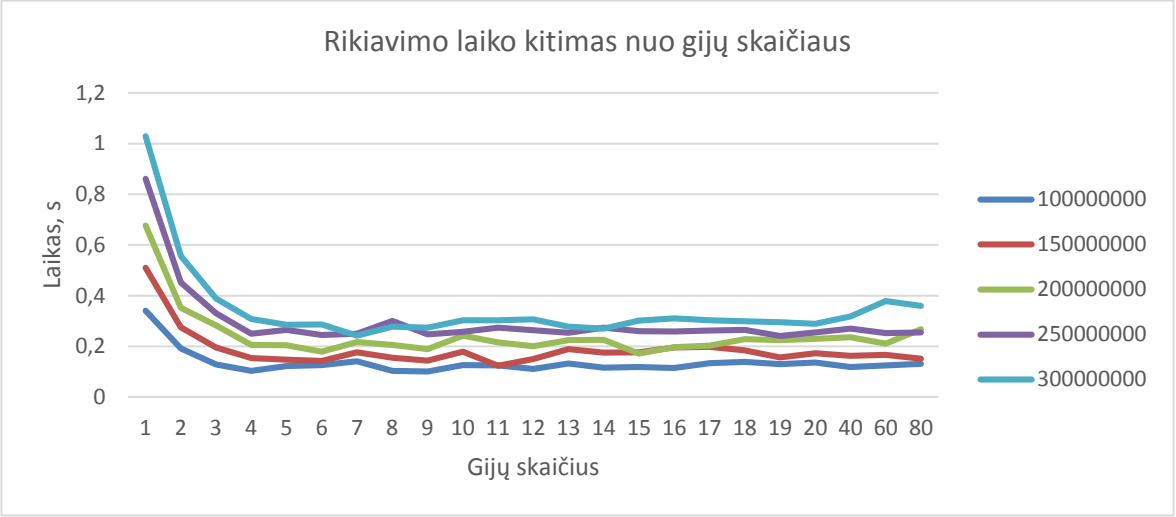
OpenMP programos fragmentas	**Konvertuota CUDA programa**

#pragma omp parallel shared(a,b,c)
private(i,j,k)
{
for (i = 0; i < size; i++){
#pragma omp for
for (j = 0; j < size; j++) {
for (k = 0; k < size; k++)
c[i][j] += a[i][k] * b[k][j];
}
}
}
	__global__ void
	doKernel (double *a , double *b , double *c , int N)
	{
	int row = blockIdx.y * blockDim.y + threadIdx.y;
	int col = blockIdx.x * blockDim.x + threadIdx.x;
	for (int k = 0 ; k<N ; k++)
	c[row * N+ col] += a[row * N+ k] * b[k * N + col];
	}

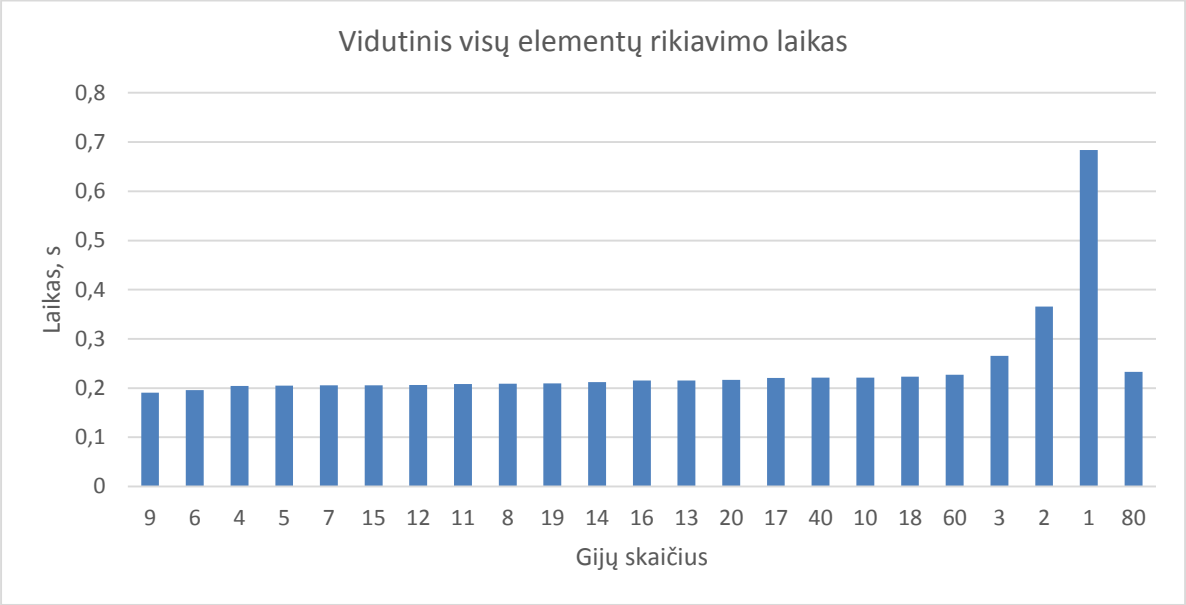
4.6.2. Nuoseklių ir lygiagrečių algoritmų tyrimas OpenMP technologijoje

4.6.2.1. Maksimumo radimas

Maksimumo paieškoje buvo naudojami sveikų skaičių duomenų rinkiniai. Algoritmo efektyvumas išbandytas naudojant masyvą bei vektorių.

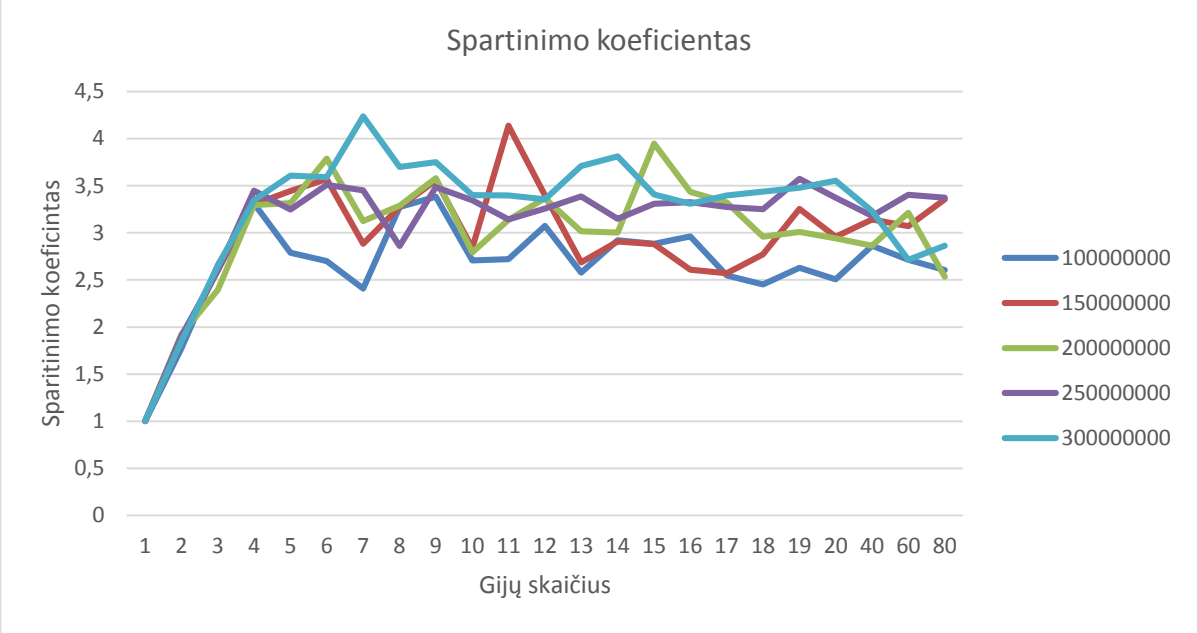


13 pav. Maksimumo paieškos masyve laiko kitimas nuo gijų skaičiaus

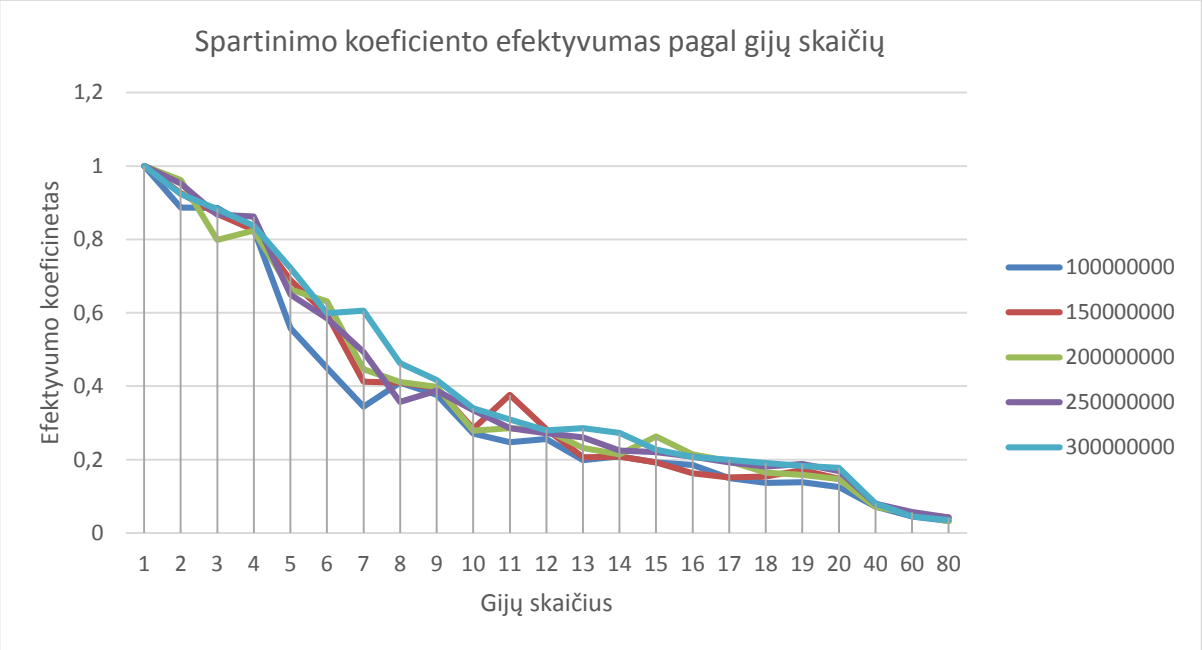


14 pav. Vidutinis maksimumo paieškos masyve laikas

Kaip matome 13 ir 14 paveikslėliuose, ilgiausias paieškos laikas buvo pasiektas ieškant per 300 mln. elementų dydžio masyvą, vykdant programą nuosekliai t.y. naudojant 1 giją. Greičiausiai paieška įvykdyta naudojant 9 gijas.

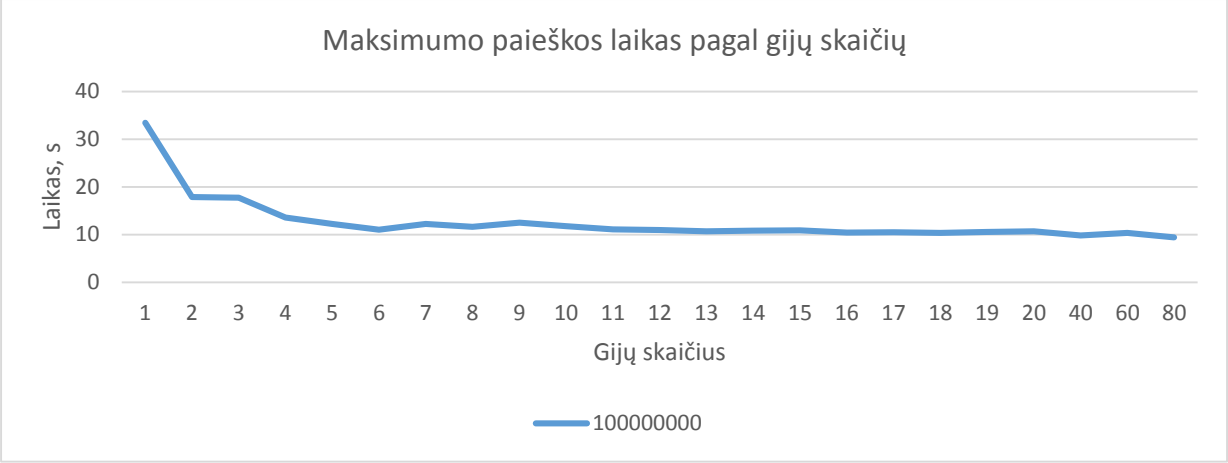


15 pav. Maksimumo paieškos masyve spartinimo koeficientas

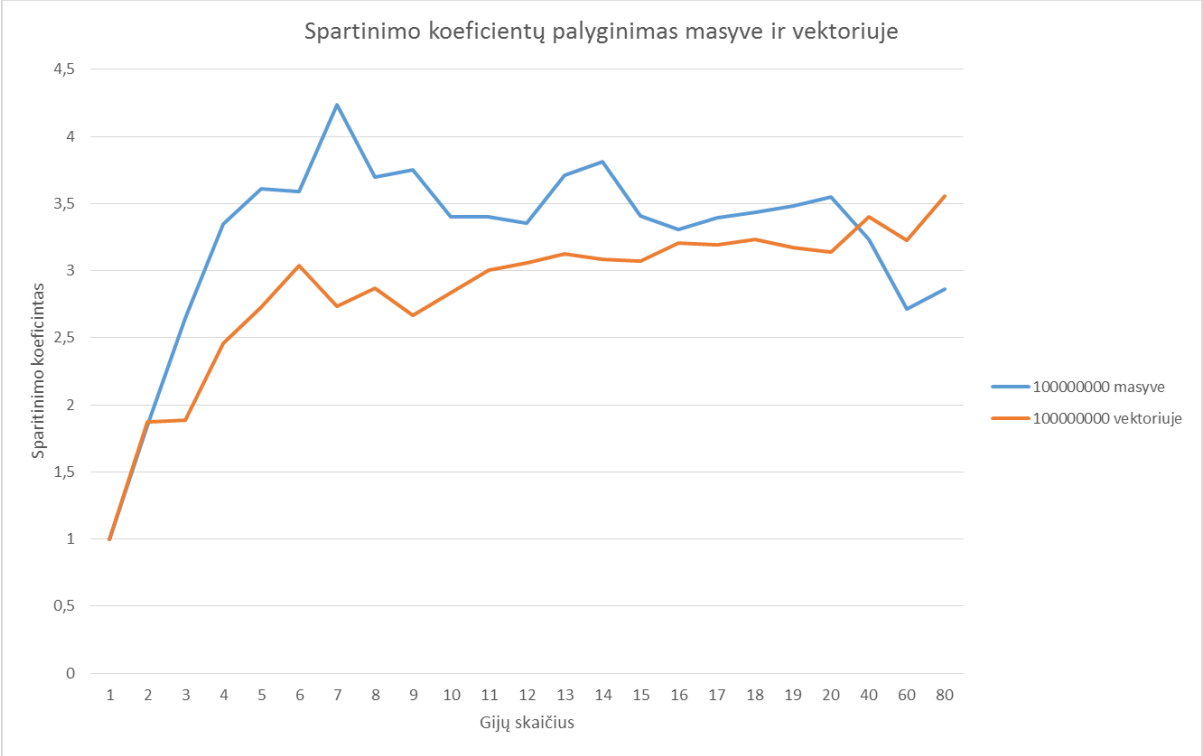


16 pav. Maksimumo paieškos masyve spartinimo koeficiento efektyvumas pagal gijų skaičių

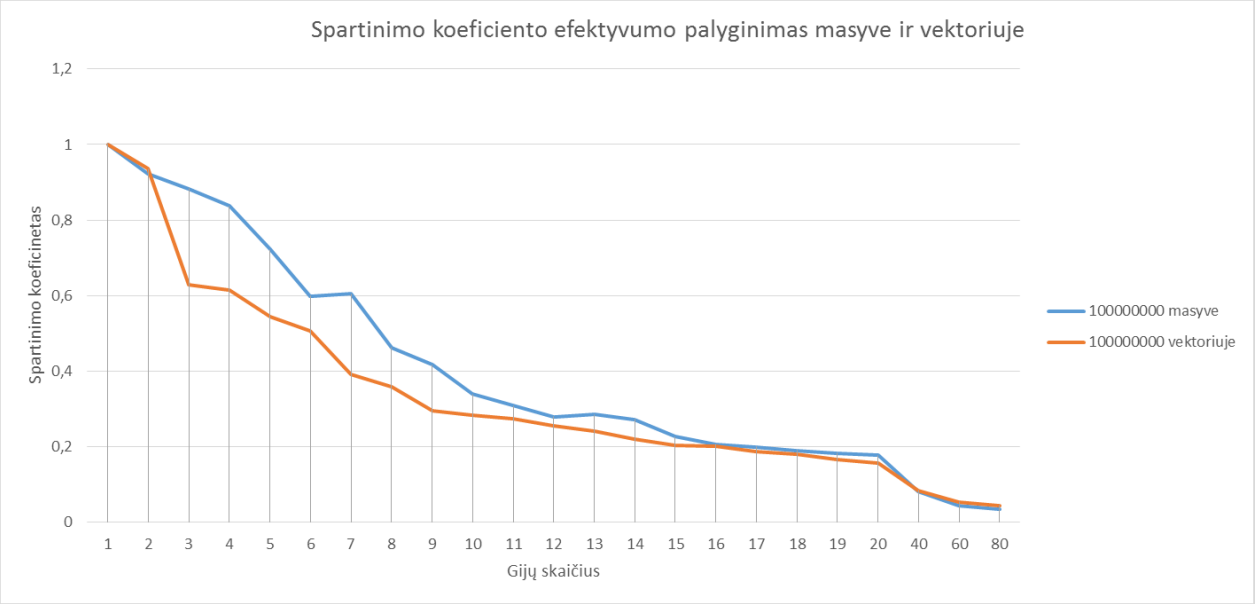
Atsižvelgus į spartinimo koeficiento grafikus, pavaizduotus 15 ir 16 paveikslėliuose, matome, kad paieška vykdoma lygiagrečiai (kai gijų skaičius >1) buvo pagreitėjus daugiau nei 4 kartus. Tai pat pastebimas dėsningumas, kad kuo daugiau naudojame gijų, tuo mažesnis pasiekiamas naudojamų gijų efektyvumas.



17 pav. Maksimumo paieškos vektoriuje paieškos laikas pagal gijų skaičių



18 pav. Spartinimo koeficientų palyginimas masyve ir vektoriuje



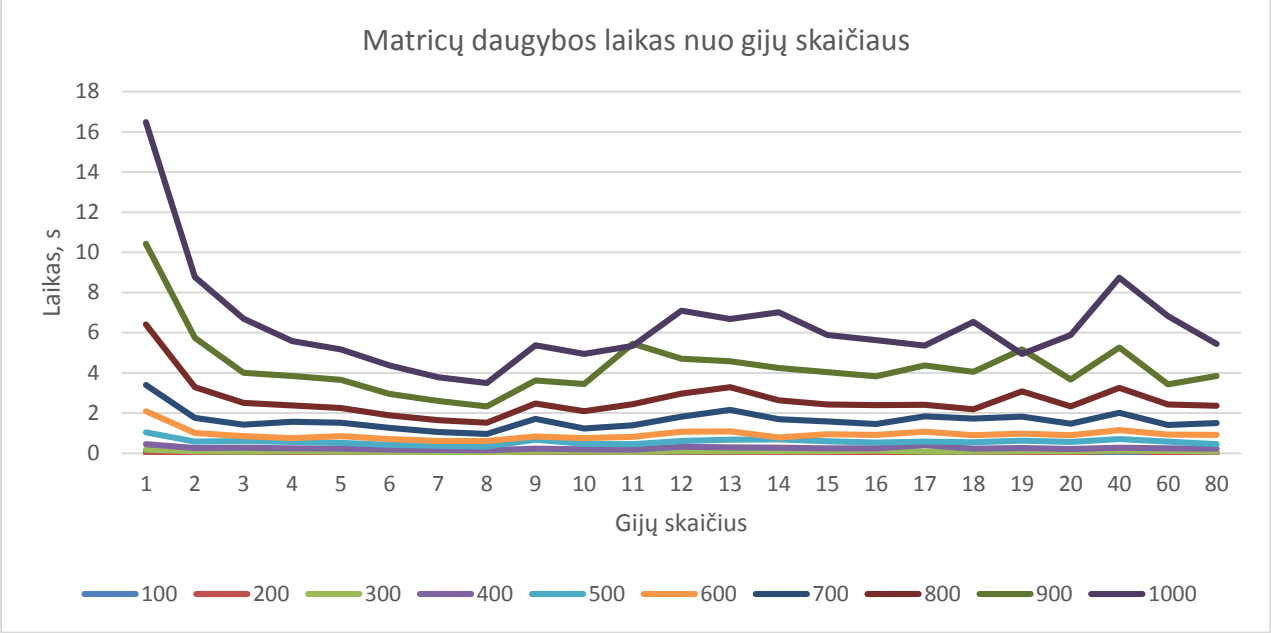
19 pav. Spartinimo koeficiento palyginimas masyve ir vektoriuje

Tai pat buvo atliktas tyrimas, kai vietoj sveikų skaičių masyvo duomenų struktūros buvo naudojamas sveikų skaičių vektorius. Kaip matome iš 17 paveikslėlio, maksimali reikšmė 100 mln. skaičių vektoriuje buvo rasta tik per daugiau nei 33s, kai tuo tarpu masyve 100mln. skaičių – 0,56 s. Dėl to, kad didžiausios reikšmės vektoriuje paieškos laikas su 100 mln. įrašų buvo labai ilgas, didesnės duomenų imtys nebuvo bandomos.

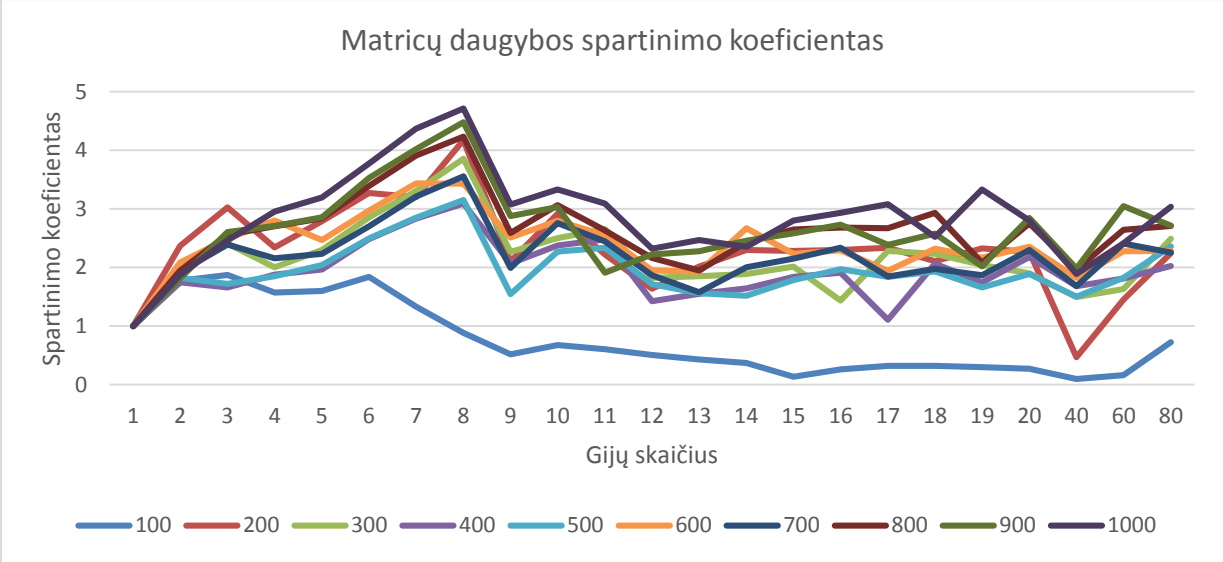
Palyginus masyvo ir vektoriaus spartinimo koeficientus (žiūrėti 18 ir 19 pav.) matome, kad masyvas efektyviau naudoja gijų resursus. Dėl šios priežasties, tolesniuose tyrimuose bus naudojamas tik paprastas dinaminis masyvas.

4.6.2.2. Matricų daugyba

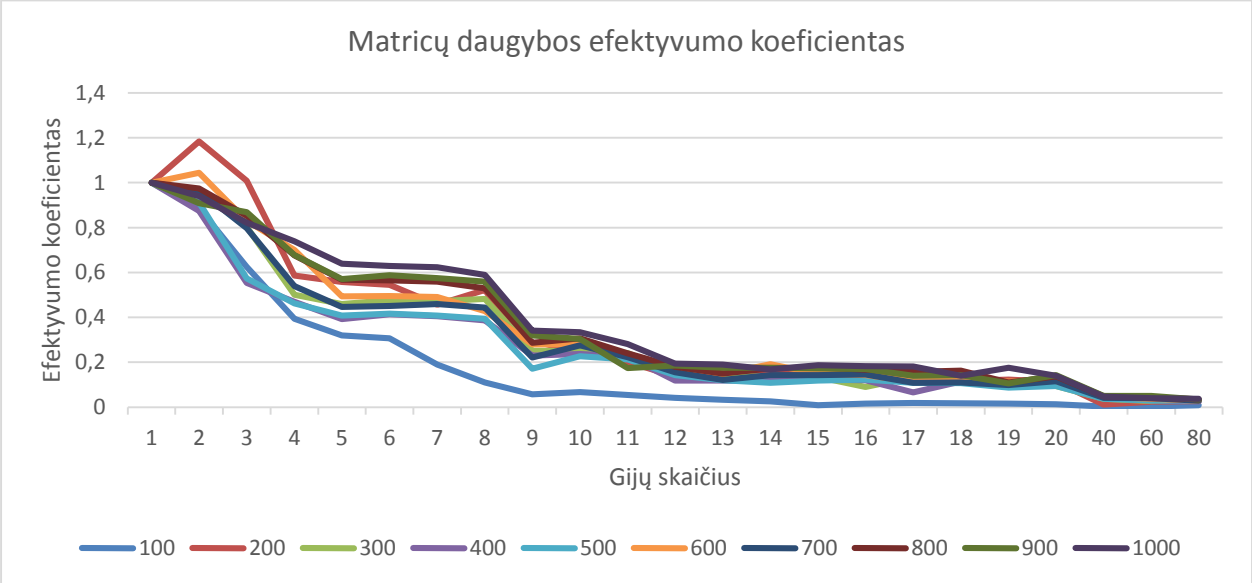
Matricų daugyboje buvo naudojamos kvadratinės matricos, sudarytos iš slankaus kablelio skaičių masyvų.



20 pav. Matricų daugybos laiko kitimas pagal naudojamų gijų skaičių



21 pav. Matricų daugybos spartinimo koeficientas pagal gijų skaičių.



22 pav. Matricų daugybos masyve efektyvumo koeficientas

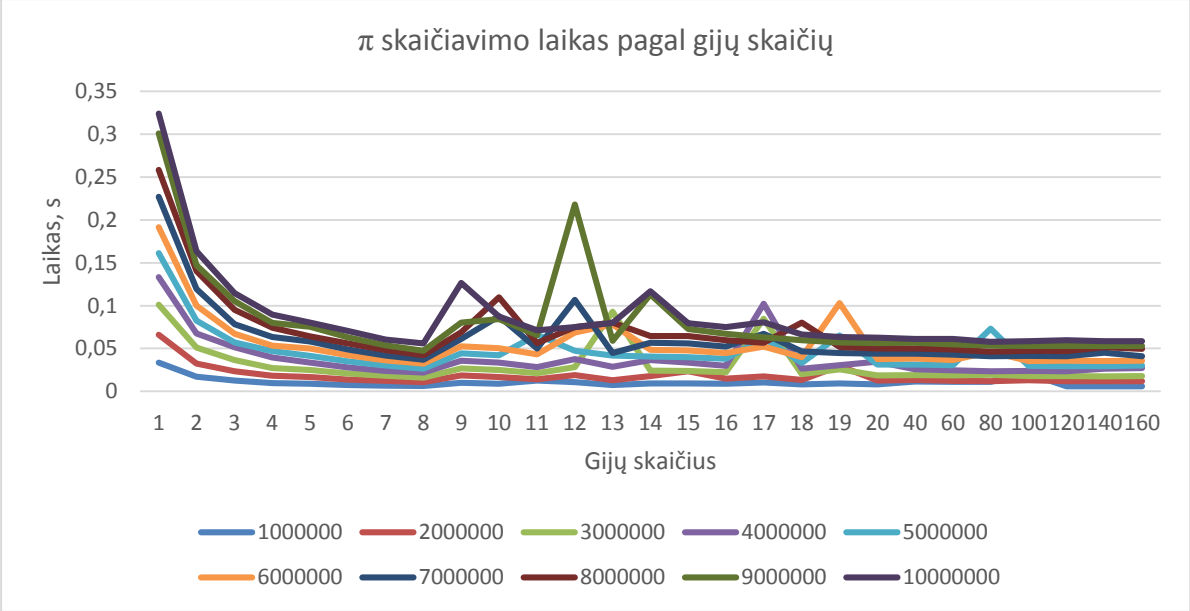
Kaip matome iš 20 paveikslėlio, matricų daugyba ilgiausiai truko beveik 16,48s. Tai naudojant nuo 2 iki 8 gijų, pastebimas ryškus pagreitėjimas. Tačiau nuo 9 iki 80 gijų naudojime, matomas daugybos laiko šuoliai. Šis kitimas yra susėjąs su gijų darbo pasidalinimu tarpusavyje – ne vienodai yra padalijamas darbas. Pavyzdžiui, kaip matome iš laiko skirtumų 8 ir 19 gijų kitime, imtis yra padalinama netolygiai, be to matomas pagreitėjimas nuo 19 iki 20 gijų.

16 lentelė. Gijų darbo pasidalinimas

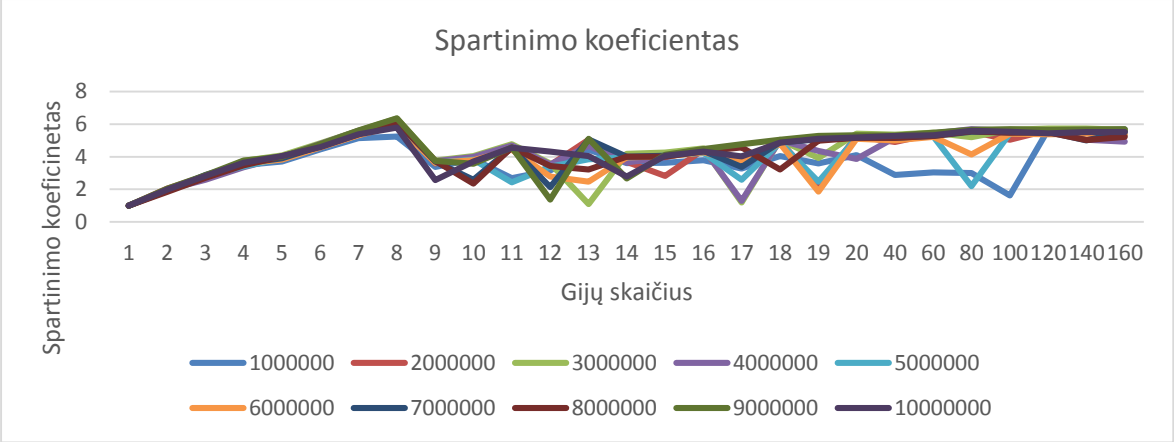
Gijų skaičius	Imties dydis	Vienos gijos elementų darbo imtis
8	1000	125
	400	50
19	400	21,05
	1000	52,63
20	400	20
	1000	50

4.6.2.3. π apskaičiavimas

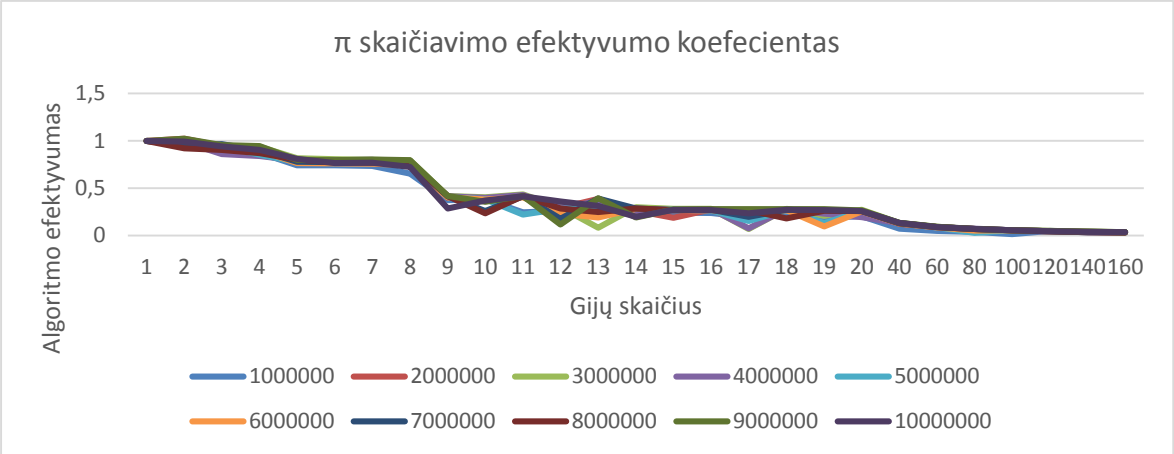
π skaičiavime buvo naudojamas skirtingas iteracijų bei gijų skaičius nuo 1 iki 160.



23 pav. π skaičiavimo laikas pagal gijų skaičių



24 pav. π skaičiavimo spartinimo koeficientas pagal gijų skaičių

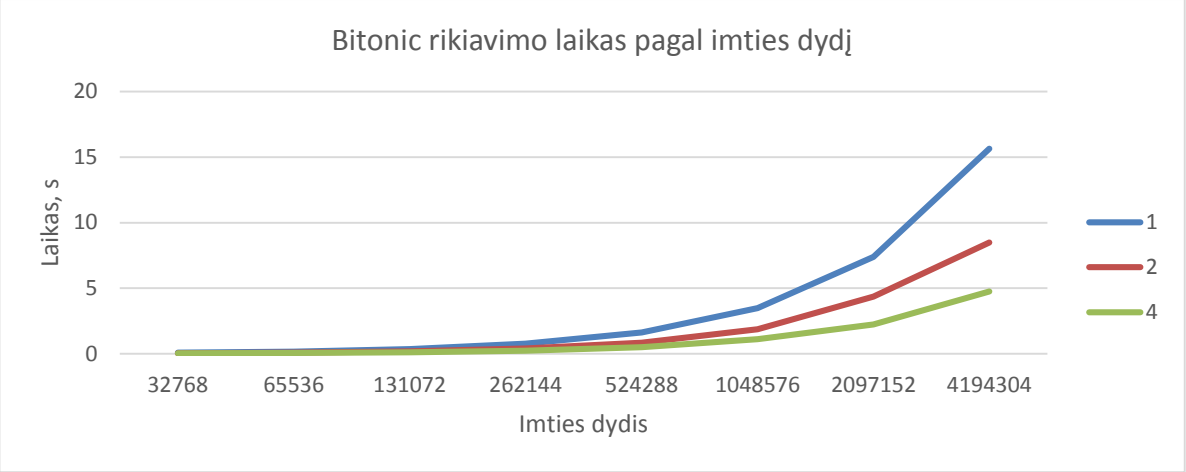


25 pav. π skaičiavimo spartinimo koeficientas pagal gijų skaičių

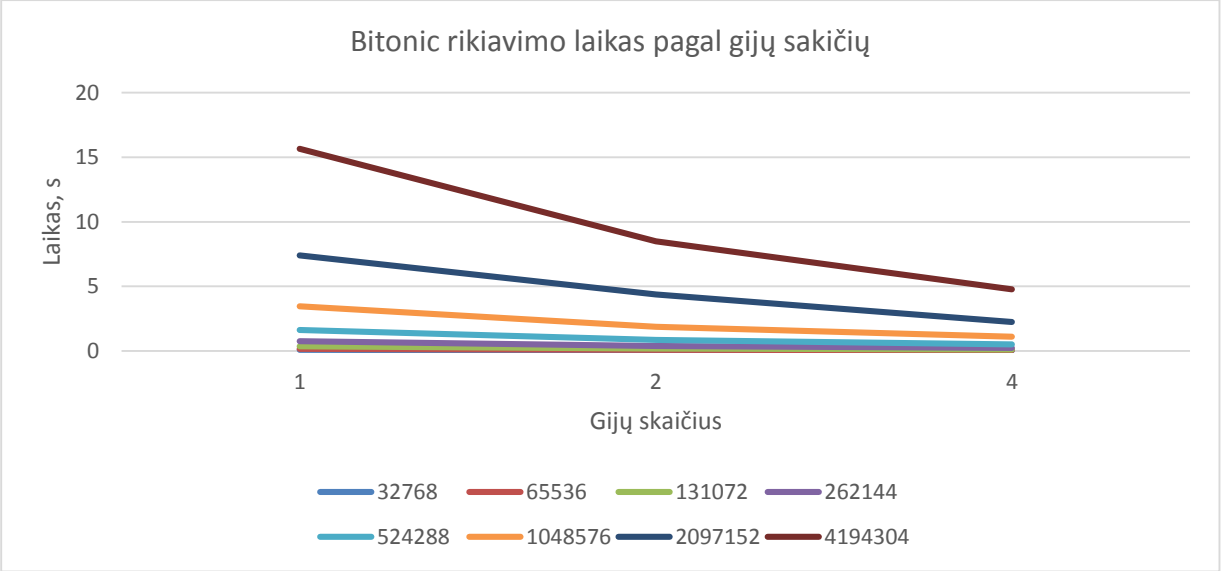
Atliekant π skaičiavimo algoritmo vykdymo tyrimą nustatyta, kad atliekant programos veiksmus lygiagrečiai, kurie yra nesusiję vienas su kitu. Šiuo atveju – formulės skaičiavimas. Paveikslėliuose 24, 25 matomas spartos kitimas, naudojant nuo 1 iki 8 gijų. Tačiau nuo 9 gijų efektyvumas pradeda staigiai mažėti bei naudojant nuo 9 iki 100 gijų – kisti. Tai galima paaiškinti OpenMP vykdoma kritine sekcija „#pragma omp critical“, kuri yra naudojama sudedant visų gijų apskaičiuotas tarpines sumas. Tai pat įtakos turi imties dydžio kitimas, kaip kad buvo matricų daugyboje (žiūrėti 14 lentelę).

4.6.2.4. Rikiavimas

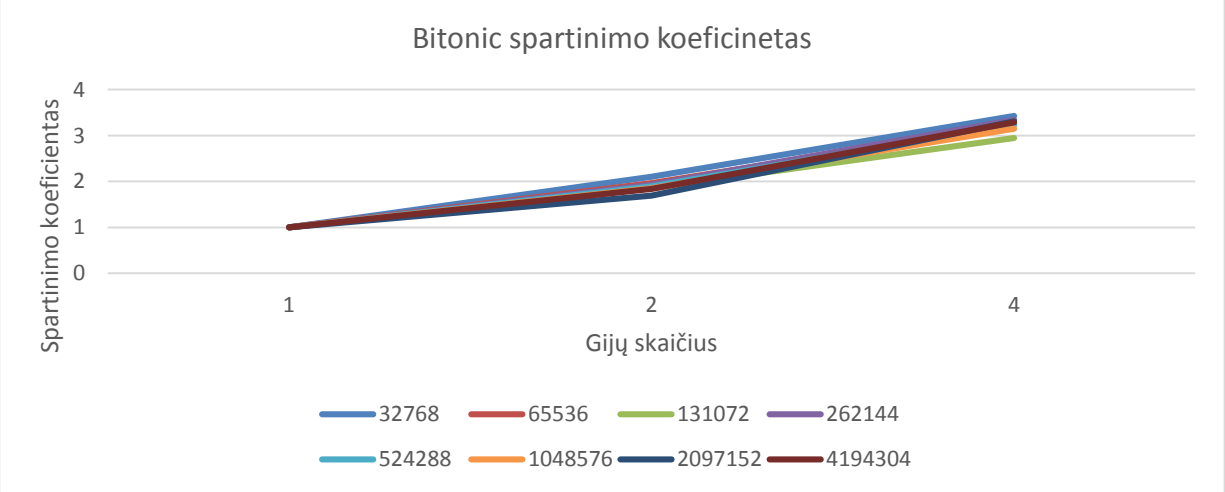
Rikiavimo tyrimui atlikti buvo panaudotas „Bitonic Sort“ algoritmas su duomenų imties dydžiais 2^n, kai n yra intervale [15; 22].



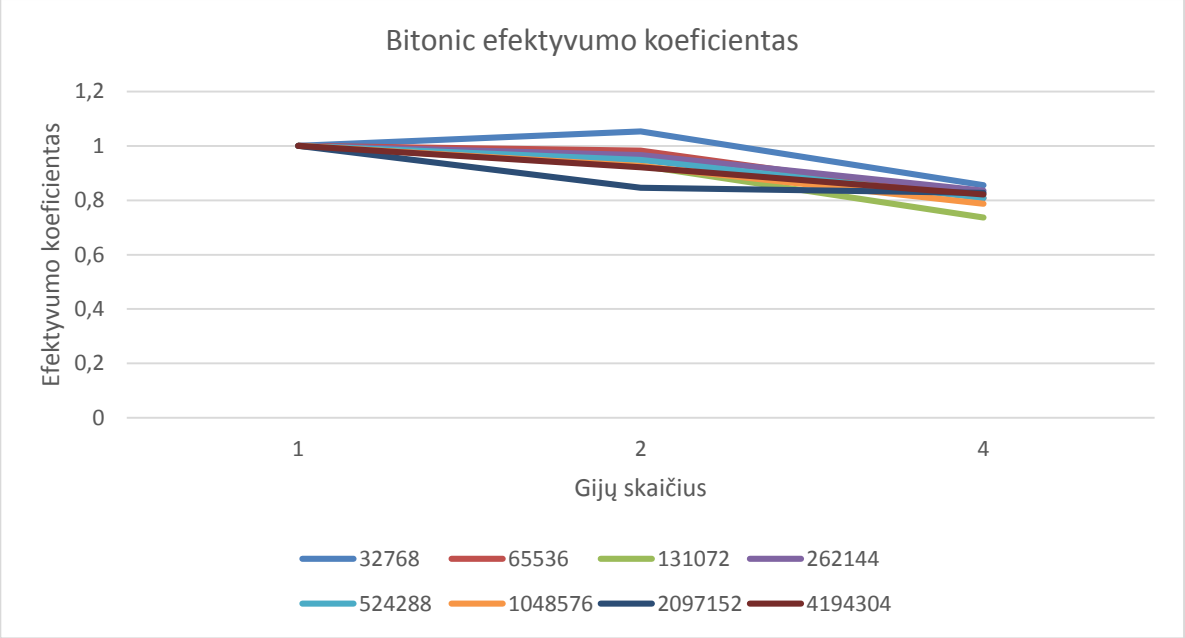
26 pav. Bitonic algoritmo rikiavimo laiko kitimas pagal imties dydį



27 pav. Bitonic algoritmo rikiavimo laiko kitimas pagal gijų skaičių



28 pav. Bitonic algoritmo spartinimo koeficientas pagal gijų skaičių



29 pav. Bitonic algoritmo efektyvumo koeficientas pagal gijų skaičių

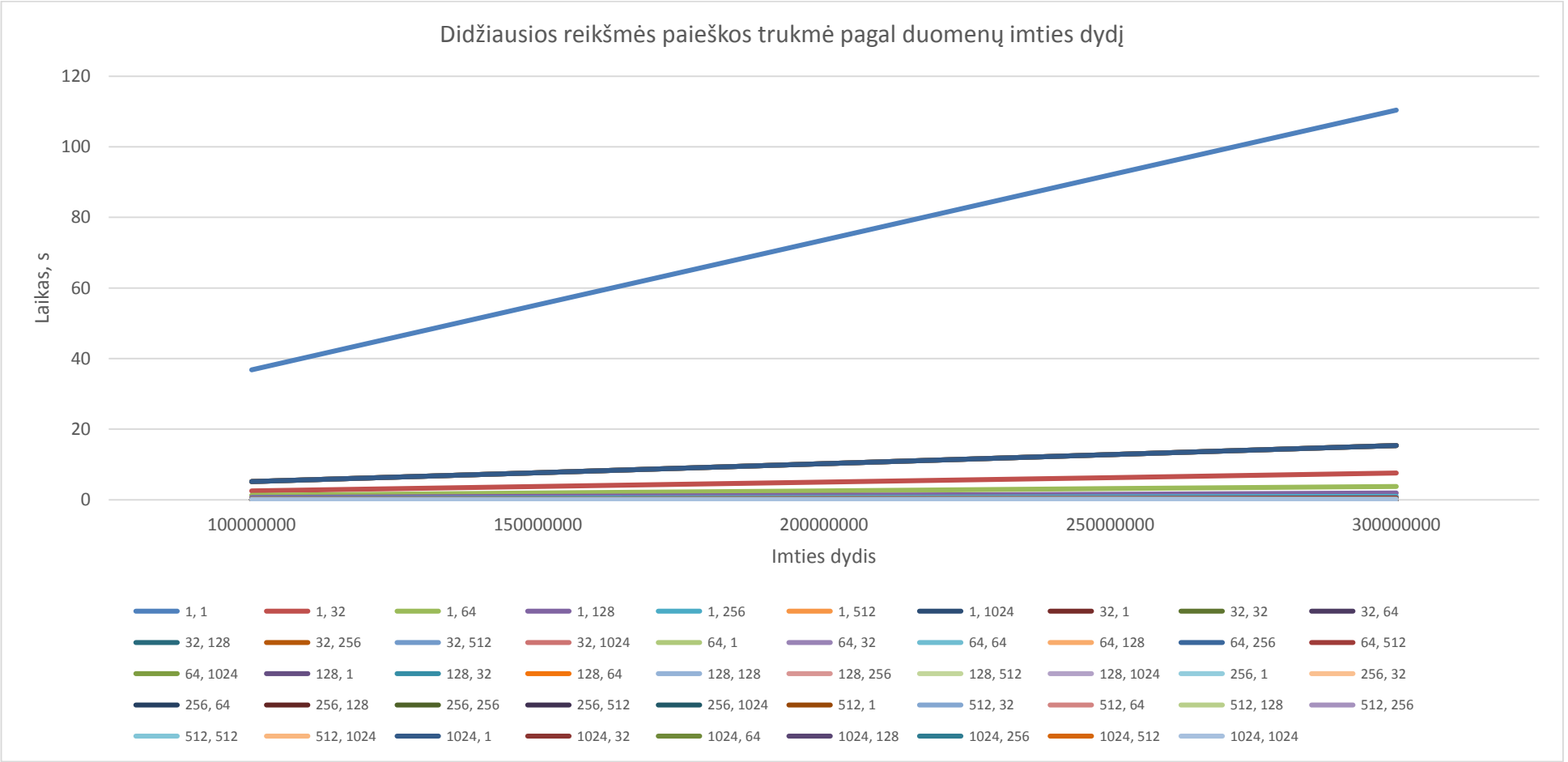
Atliekant rikiavimo algoritmo tyrimą susidurta su problema, kad ne su visom gijų ir skaičių imtimis jis tinkamai veikia. Atlikus išrikiuotų skaičių rinkinių patikrinimą ar visi skaičiai yra išrikiuoti didėjimo tvarka, rasta išsimėčiusių skaičių. Todėl diagramose yra rodomos tik tos gijos ir imtys, kuriose nebuvo rasta netikslumų. Taip yra todėl, nes šis algoritmas labai priklauso nuo naudojamų gijų skaičiaus bei duomenų imties dydžio.

Kaip matome iš 26, 27, 28 paveikslėlių, šis rikiavimo algoritmas vykdomas lygiagrečiai pagreitėja iki 3,5 karto. Jis yra labai efektyvus, kai rikiuojami maži duomenų kiekiai, bet tampa lėtesnis, kai rikiuojamos ilgos sekos.

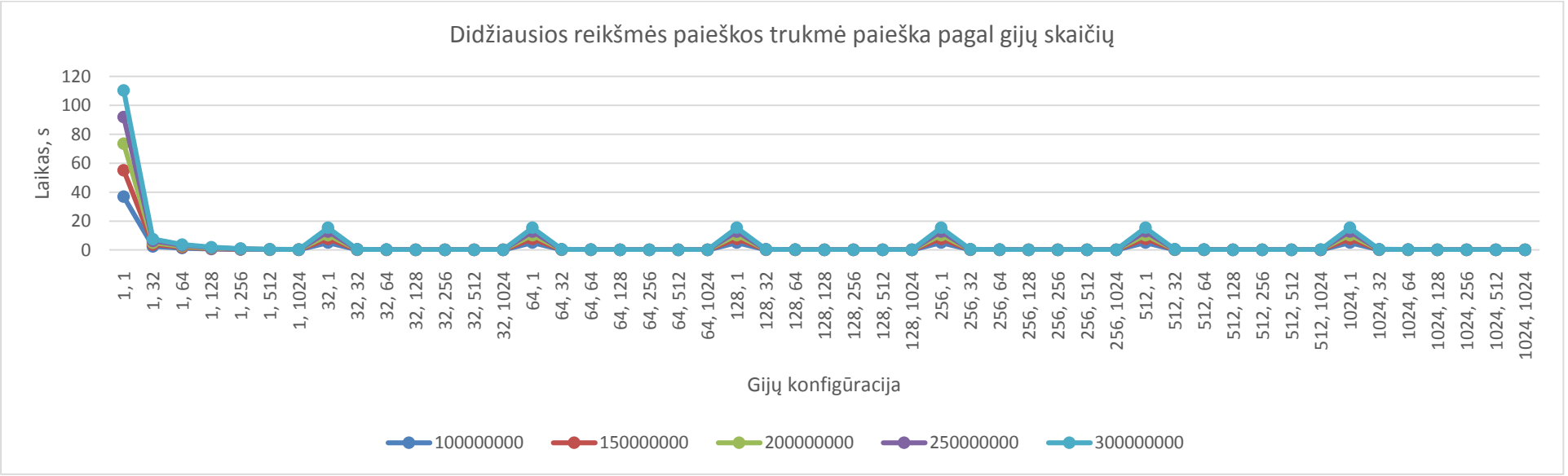
4.6.3. Nuoseklių ir lygiagrečių algoritmų tyrimas CUDA technologijoje

Atliekant CUDA tyrimą buvo naudojamos įvairios gijų konfigūracijos bei skaičių sekų dydžiai. Kaip ir ankstesniuose tyrimuose – skaičių sekos buvo vienodos. Gijų konfigūracija – tai 2 skaičiai, kurie aprašo kiek blokų ir kiek gijų bloke gali būti vykdoma. Kiekvieną šių skaičių galima sudaryti iš 1D, 2D ar 3D skaičių (tam aprašyti kode naudojama struktūra dim3). Kokią gijų konfigūraciją (1D, 2D ar 3D) naudoti, priklauso nuo algoritmo. Pavyzdžiui matricų skaičiavime buvo naudojama 2D konfigūracija, o kituose – 1D. Aprašant tyrimo duomenis blokų ir gijų per bloką skaičiai yra atskiriami kableliu, o 2D žymima skliausteliuose. Pavyzdžiui <128, 512> – blokų (128) ir gijų skaičius (512) 1D struktūroje. 2D struktūroje – <(100, 2), (1, 64)> t.y. 100*2 blokų ir 64*1 gijų. Tai pat svarbu paminėti, kad gijų skaičius negali viršyti 1024, nes priešingu atveju gaunama programoje klaida.

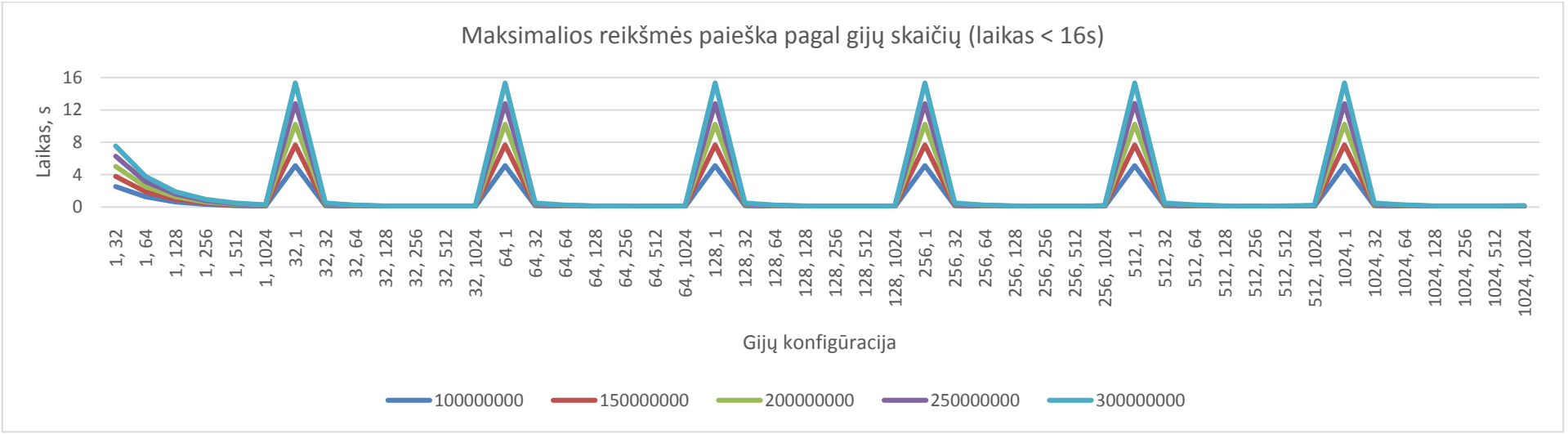
4.6.3.1. Maksimumo radimas



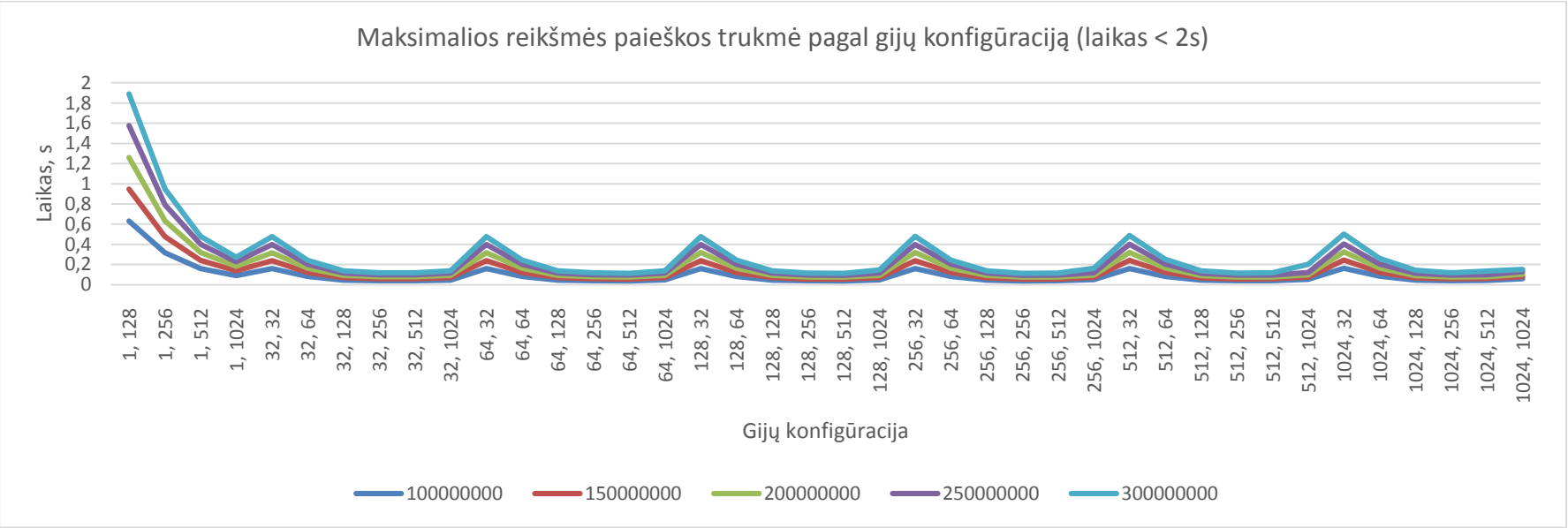
30 pav. Didžiausios reikšmės paieškos trukmė pagal imties dydį



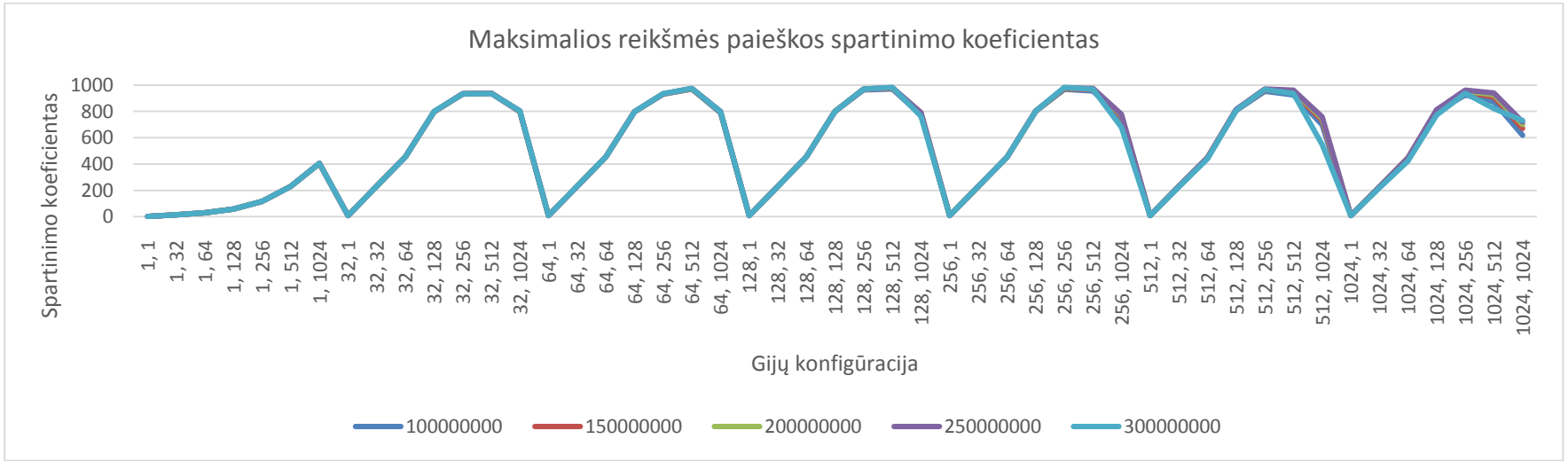
31 pav. Didžiausios reikšmės paieškos trukmė pagal gijų konfigūraciją



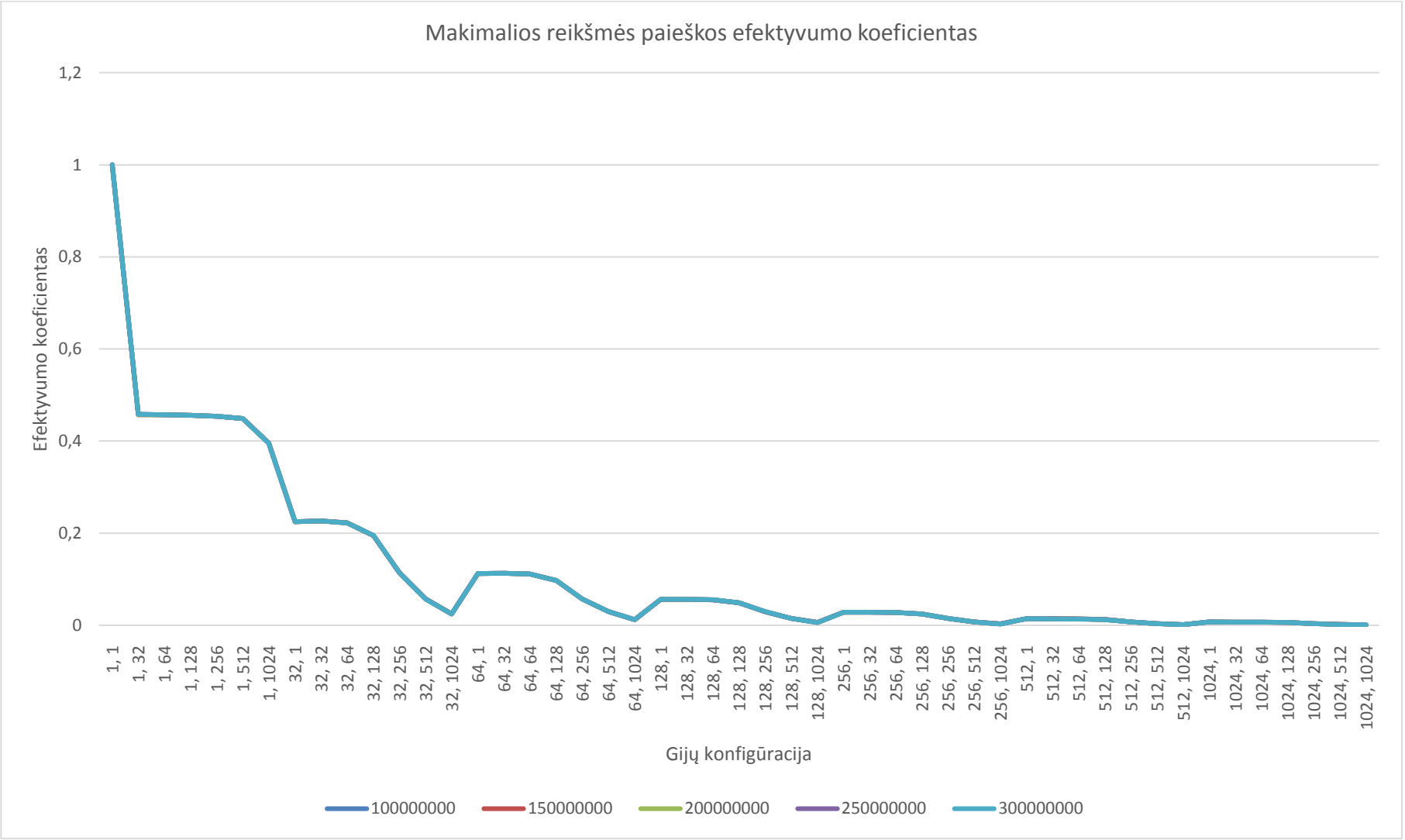
32 pav. Didžiausio reikšmės paieška pagal gijų konfigūraciją, kai laikas < 16s



33 pav. Maksimalios reikšmės paieškos trukmė pagal gijų konfigūraciją, kai laikas < 2s



34 pav. Maksimalios reikšmės paieškos spartinimo koeficientas



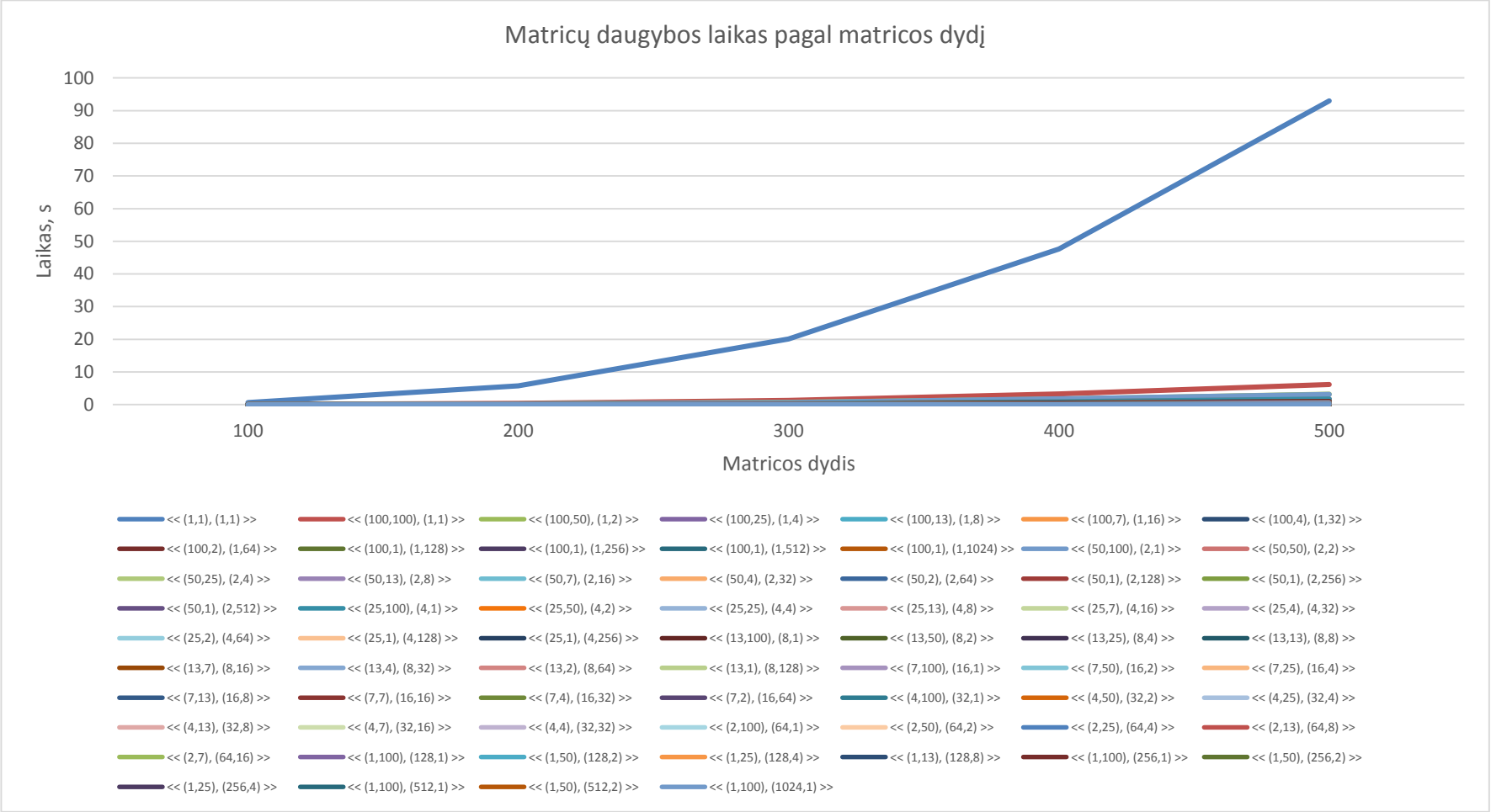
35 pav. Maksimalios reikšmės paieškos efektyvumo koeficientas

Atliekant maksimalios reikšmės paiešką naudojant CUDA technologiją, matomas didelis pagreitėjimas naudojant daugiau kaip 1 giją. Kaip matome iš 30 paveikslėlio – surikiuoti 300 mln. skaičių užtruko tik 0,112 sekundės, naudojant konfigūraciją <128, 512>, o naudojant konfigūraciją <1, 1> užtruko net 110,4 sekundės. Remiantis 33 paveikslėlio duomenimis, gautas didesnis nei 988 kartų pagreitėjimas. Tačiau atsižvelgus į efektyvumo koeficiento duomenis, matomus 35 paveikslėlyje, gijų resursai nėra pilnai išnaudojami.

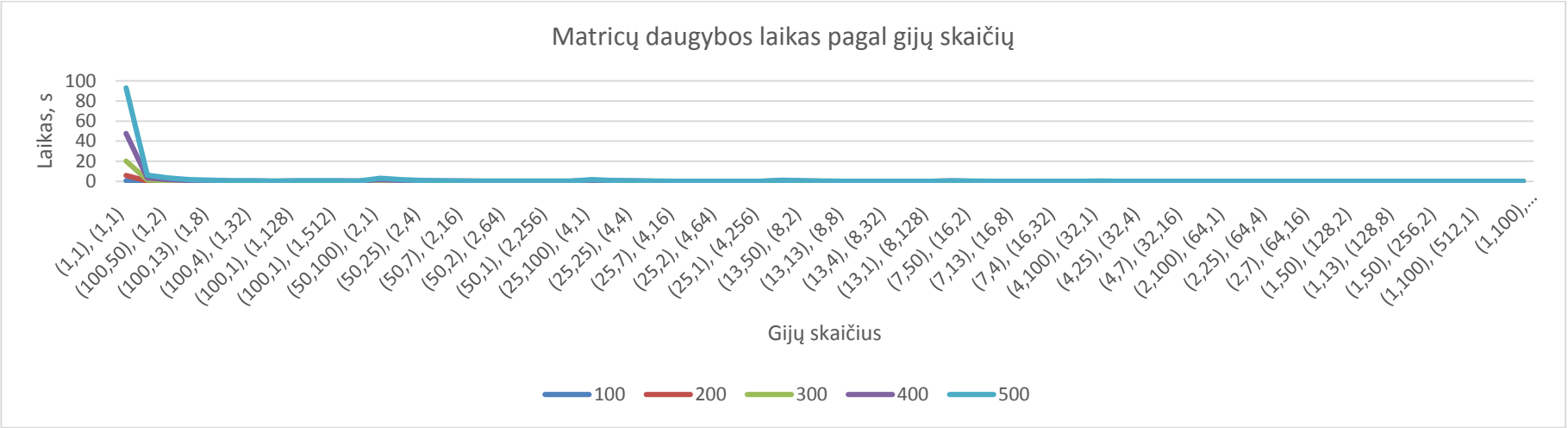
4.6.3.2. Matricų daugyba

Atliekant kvadratinių matricų daugybos tyrimą, susidurta su problema, kad vaizdo plokštės atmintyje netelpa duomenų matricos – A, B ir rezultatų matrica C – visos 500 x 500 dydžio. Todėl nebuvo atliekamas tyrimas su didesnėmis nei 500 x 500 kvadratinėmis matricomis.

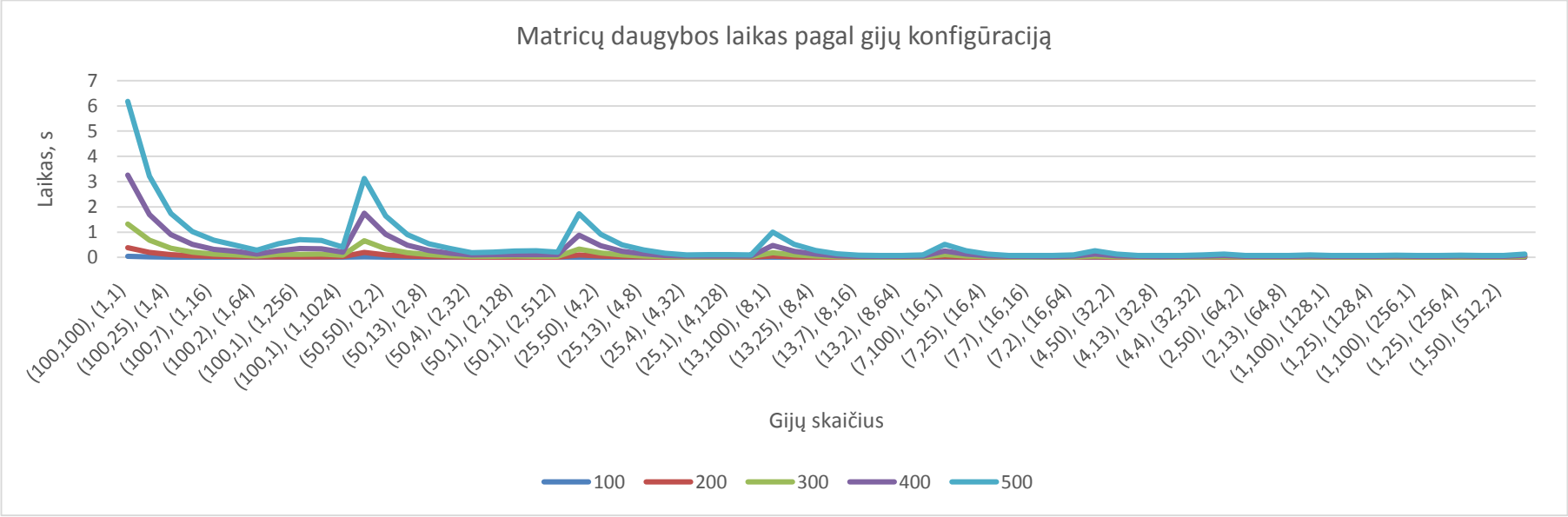
Matricų daugyboje yra pastebimas ypač didelis pagreitėjimas, naudojant daugiau kaip 1 giją. Kaip matome iš 36 - 38 paveikslėlių – sudauginti 500 x 500 dydžio matricas užtruko tik 0,071127 sekundės, naudojant konfigūraciją < (1,100), (256,1) > . Naudojant konfigūraciją <1, 1> užtruko net 93,018 sekundės. Remiantis 38 paveikslėlio duomenimis, gautas didesnis nei 1307,76 karto pagreitėjimas. Tačiau atsižvelgus į efektyvumo koeficiento duomenis, matomus 41 paveikslėlyje, gijų panaudojimo efektyvumas yra labai mažas. Taip yra todėl, kad kiekviena gija skaičiuoja tik po 1 matricos elementą.



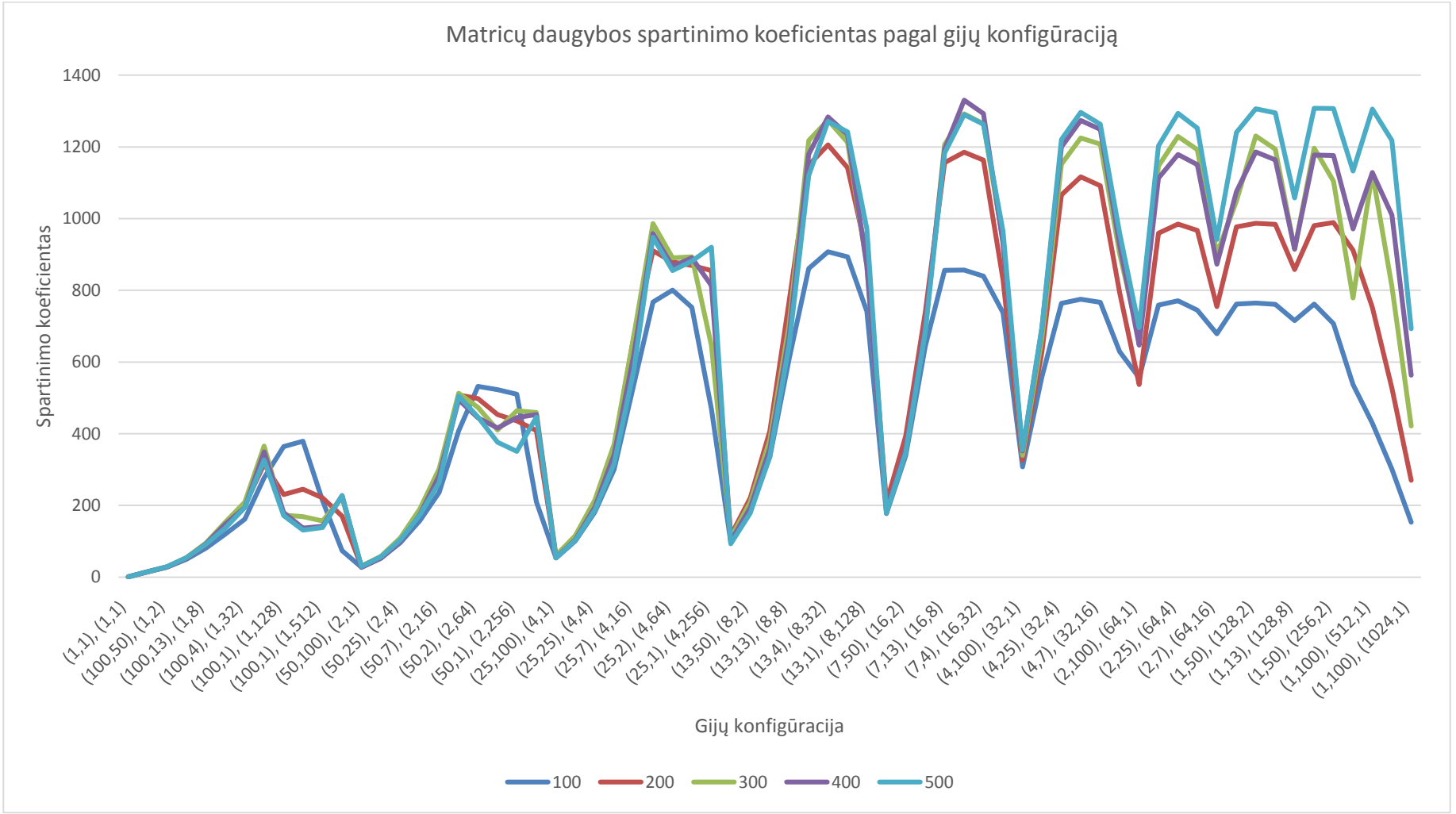
36 pav. Matricų daugybos laiko kitimas pagal matricų dydį



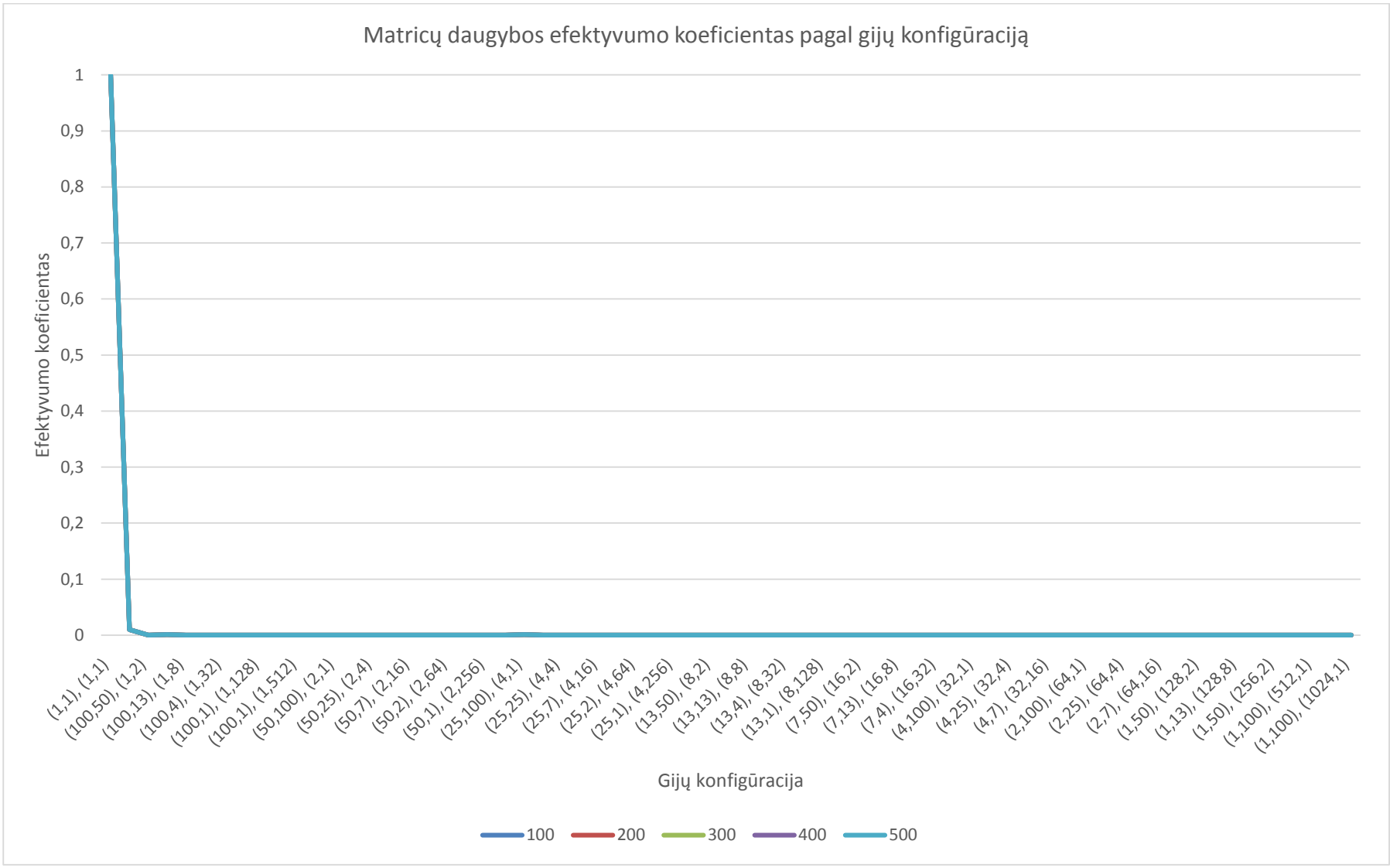
37 pav. Matricų daugybos laikas pagal gijų konfigūraciją



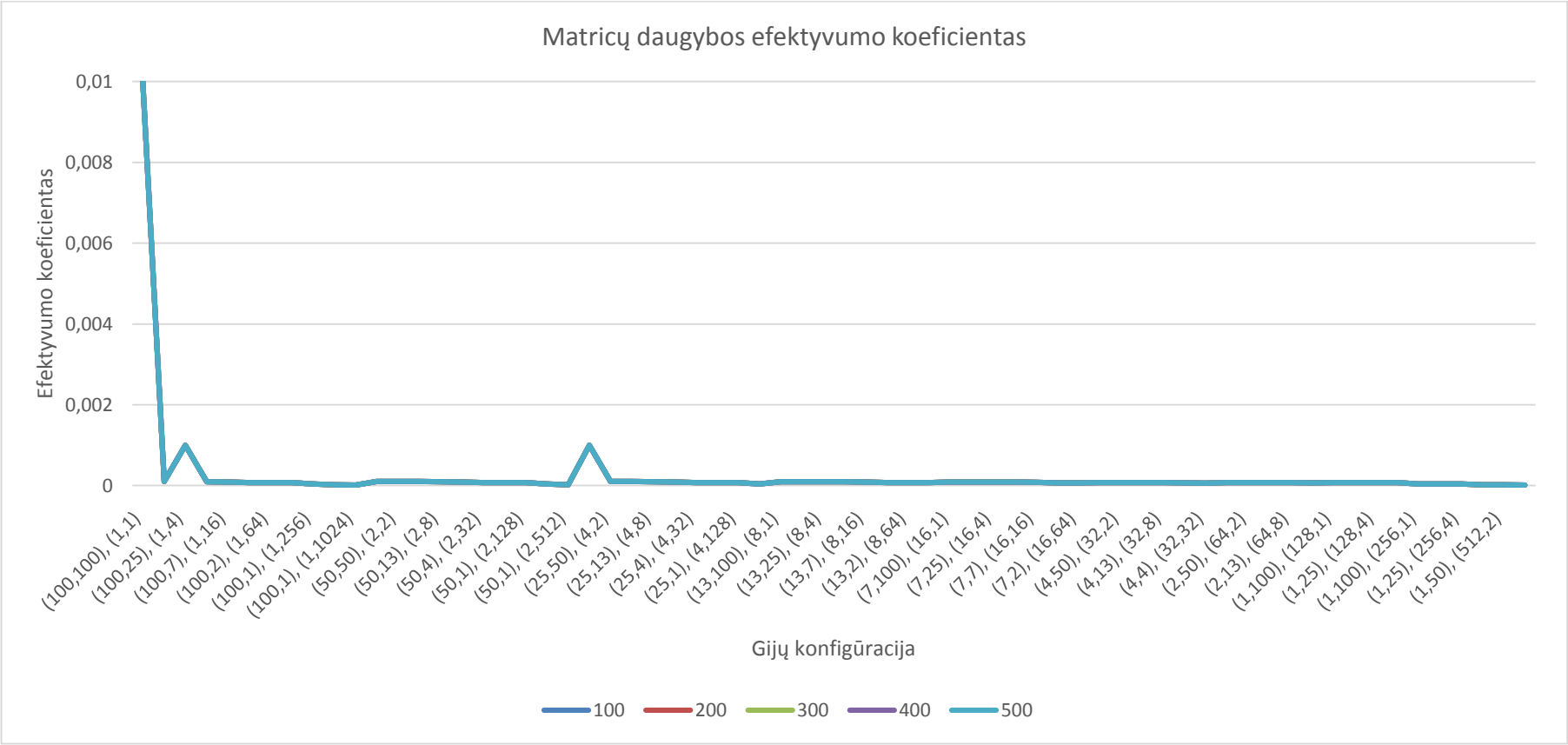
38 pav. Matricų daugybos laikas pagal gijų konfigūraciją



39 pav. Matricų daugybos spartinimo koeficientas pagal gijų konfigūraciją

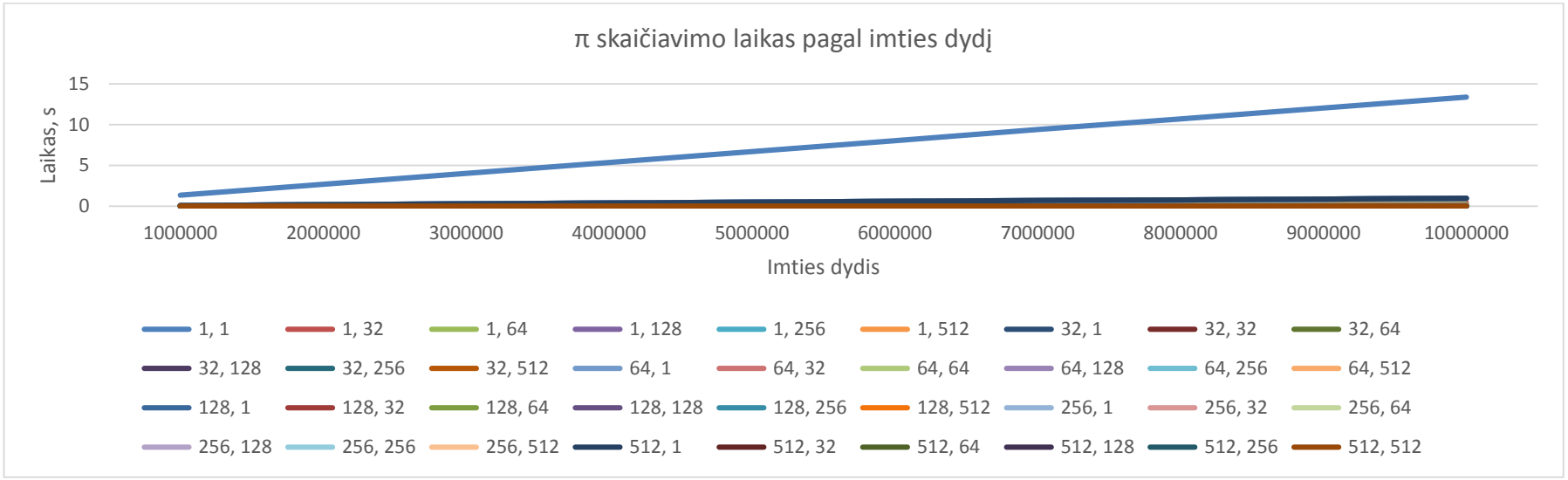


40 pav. Matricų daugybos efektyvumo koeficientas pagal gijų konfigūraciją

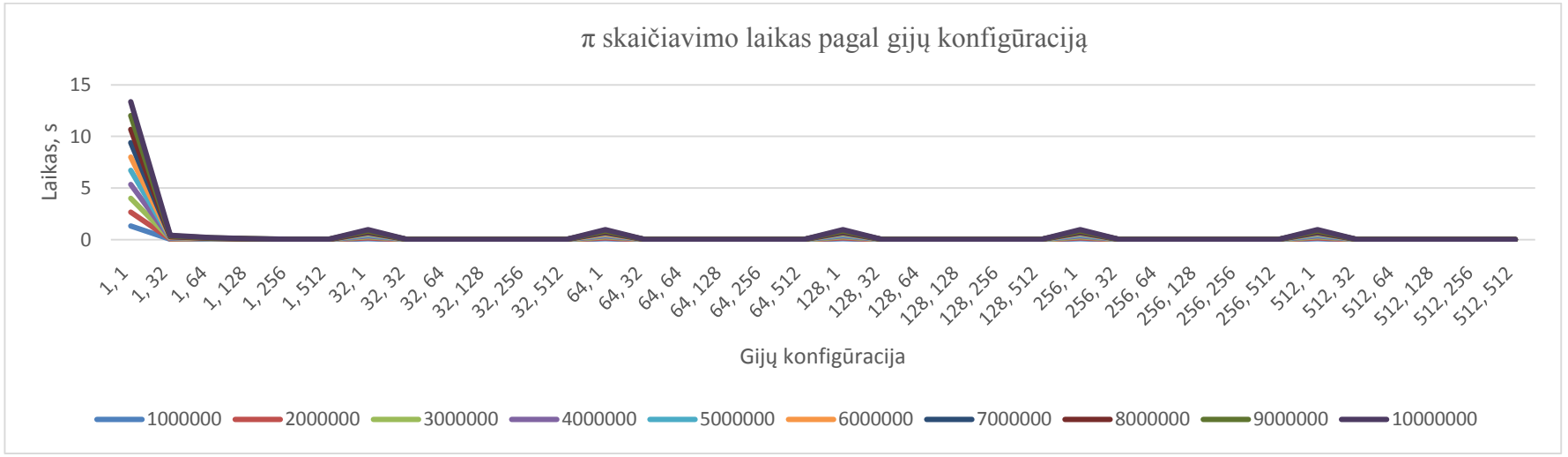


41 pav. Matricų daugybos efektyvumo koeficientas pagal gijų skaičių, kai efektyvumo koeficientas mažesnis nei 0,01

4.6.3.3. π apskaičiavimas

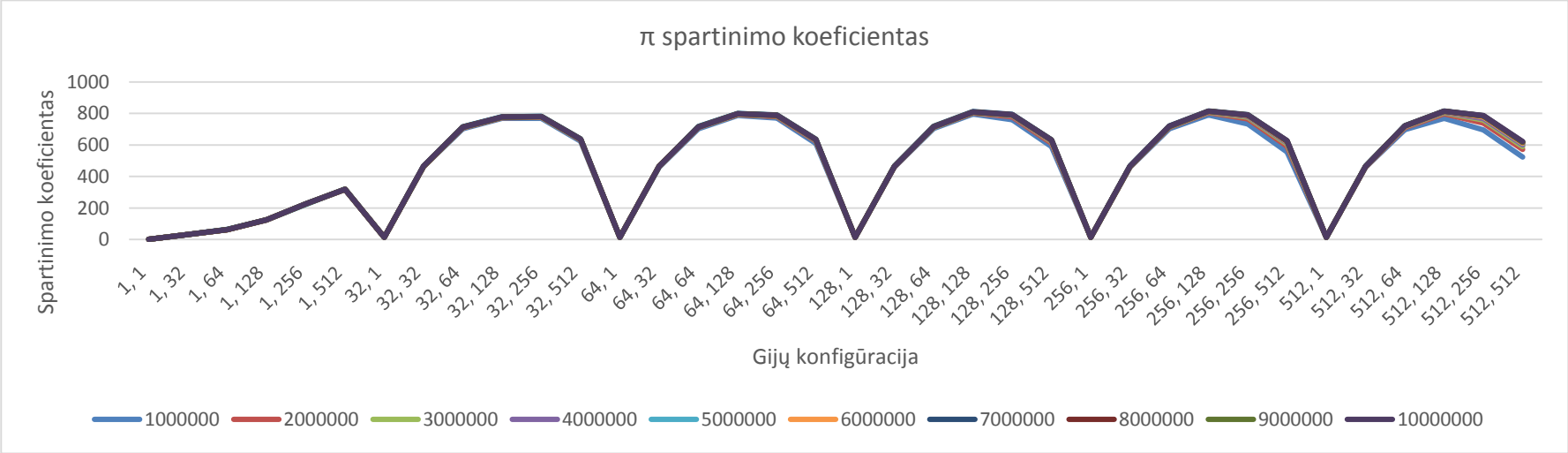


42 pav. π skaičiavimo laikas pagal imties dydį

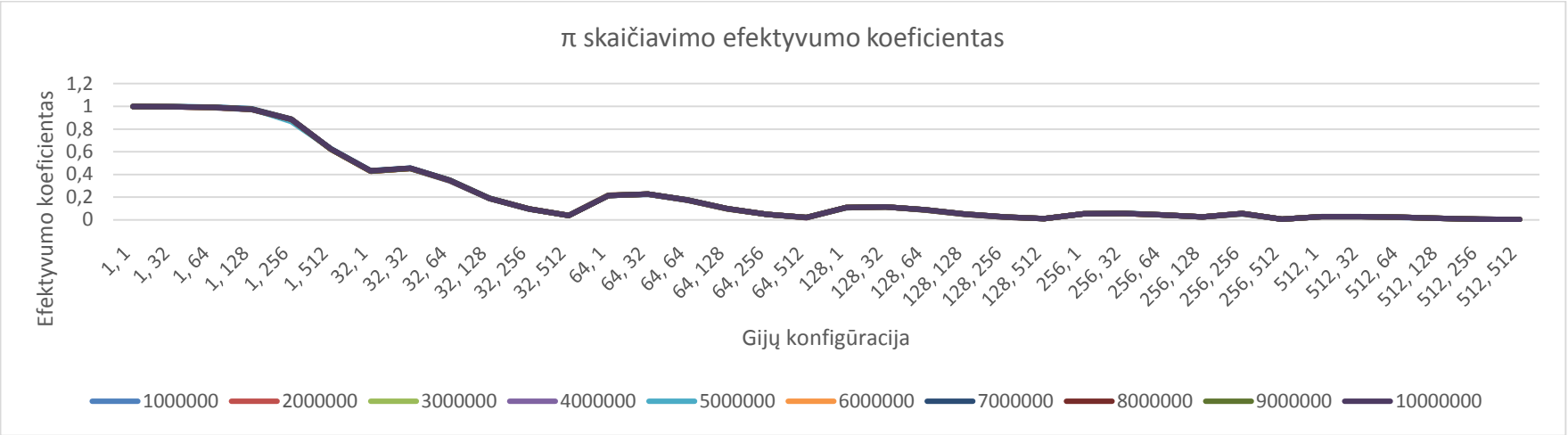


43 pav. π skaičiavimo laikas pagal gijų konfigūraciją

44 pav. π skaičiavimo laikas pagal gijų konfigūraciją, kai laikas mažesnis nei 1s



45 pav. π skaičiavimo spartinimo koeficientas pagal gijų konfigūraciją



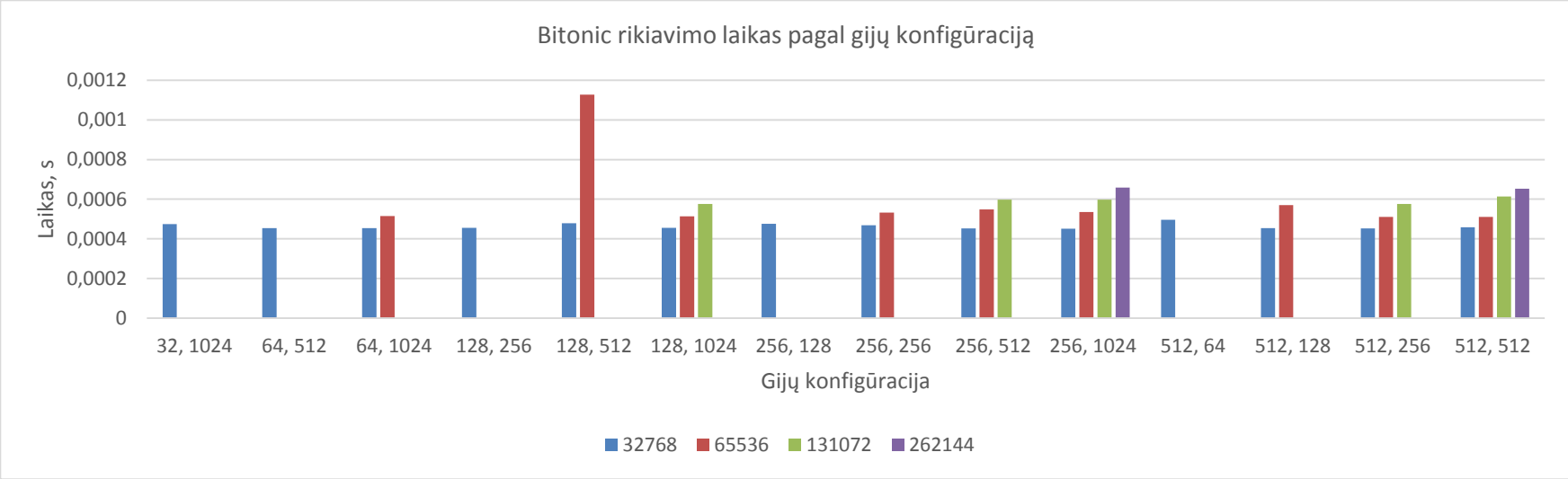
46 pav. π skaičiavimo efektyvumo koeficientas pagal gijų konfigūraciją

Kaip matome iš 42 – 44 paveikslėlių, skaičiuojant π reikšmę naudojant <1 1> konfigūraciją, 10 mln. tikslumu, truko 13,35 sekundės., o naudojant konfigūraciją < 512, 128> – 0,0164 sekundės. Pagal 45 paveikslėlį matome, kad spartinimo koeficientas – daugiau nei 800. Didėjant gijų skaičiui, π skaičiavimo greitis nuolat auga, nes kiekviena gija gali nepriklausomai viena nuo kitos skaičiuoti integralo reikšmes tam tikram intervale.

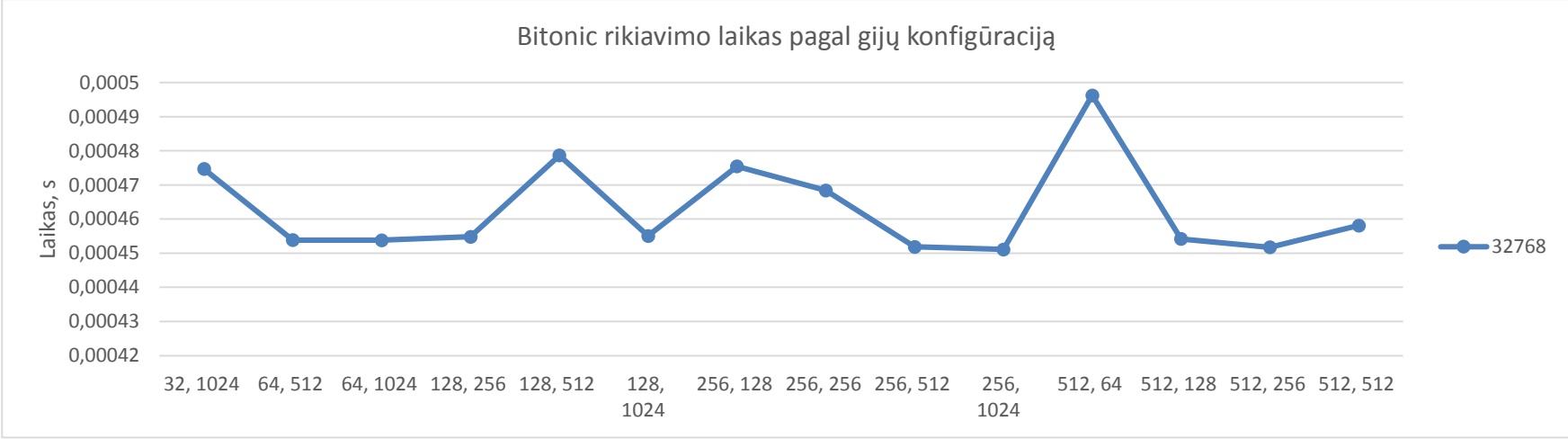
4.6.3.4. Rikiavimas

Atlikus rikiavimo tyrimą, naudojant „Bitonic“ rikiavimo algoritmą, susidurta su problema, kad nepavyko teisingai išrikiuoti skaičių naudojant daugumą gijų ir imties konfigūracijų (žiūrėti 4.6.2.4 paragrafą). Kadangi nepavyko atlikti šio algoritmo tyrimo ir su 1 gija, todėl spartinimo ir efektyvumo koeficientai taip pat nebuvo paskaičiuoti. Kaip matome iš 47 ir 48 paveikslėlių, teisingai išrikiuoti pavyko tik pagal tam tikras konfigūracijas.

Ypač be klaidų vyko rikiavimas, kai imties dydis yra 32768. Nors buvo išbandyti keli algoritmai šiai problemai spręsti, tačiau klaidų išvengti nepavyko – tik vienas ar keletas skaičių būdavo ne savo vietoje. Tai pat nepavyko CUDA technologijoje paleisti OpenMP dalyje naudoto algoritmo, nes naudojama vaizdo plokštė nepalaiko rekursyvių CUDA branduolio kreipinių.



47 pav. „Bitonic“ algoritmo rikiavimo laikas pagal gijų konfigūraciją



48 pav. „Bitonic“ algoritmo rikiavimo laikas, kai imties dydis 32768

4.7. Išvados

1. Atlikus sukurtos sistemos tyrimą su didžiausios reikšmės paieška, π reikšmės skaičiavimu, matricų daugyba bei rikiavimo algoritmais nustatyta, kad ne kiekvienas algoritmas gali būti vienareikšmiškai konvertuotas kitai technologijai. Gali reikėti perkoduoti visą algoritmo veikimo principą dėl naudojamų technologijų skirtumų.
2. Pastebėta, kad didžiausias efektyvumas naudojant OpenMP pasiekiamas tada, kai naudojamas gijų skaičius neviršija CPU gijų skaičiaus.
3. Visi tirti algoritmai pasižymėjo vykdymo laiko sumažėjimu vykdant juos lygiagrečiai OpenMP ir CUDA technologijose, tačiau didžiausias pagreitėjimas matomas naudojant CUDA technologiją. Tai pat CUDA technologija pasižymėjo keletu trūkumų – turi ribotą atmintį, sudėtinga konfigūracija. Reikia atlikti išsamius algoritmų konfigūracijų tyrimus, norit rasti geriausią konfigūraciją pasirinktai duomenų imčiai.

5. IŠVADOS

1. Tyrimas padėjo pasiekti užsibrėžtų tikslų – atskleidė praeitame pusmetyje kurtos sistemos trūkumus bei padėjo identifikuoti būsimiems vartotojams aktualius sistemos konfigūravimo parametrus, tokus kaip gijų skaičiaus nustatymas, aktualiausi jau paruošti ir optimizuoti darbui algoritmai bei galimybė leisti išmatuoti vartotojui įkeltos programos greitaveiką.
2. Išlygiagretintų algoritmų laiko kitimo, spartinimo bei efektyvumo koeficientų tyrimas parodė, kad galima pasiekti OpenMP technologijoje iki 4.5 karto pagreitėjimą, CUDA - iki 1300 kartų pagreitėjimą. Todėl galima teigti, kad pasinaudojus šiuolaikinių programavimo technologijų privalomais galima pagerinti sistemas, pasiekti spartesnį informacijos apdorojimą.
3. Ateityje būtų tikslinga išlygiagretinti jau egzistuojančią sistemą, dirbančią su duomenų baze, palyginti jų greitaveiką.

6. TERMINŲ IR SANTRUMPŲ ŽODYNAS

CUDA	„NVIDIA“ sukurta nauja duomenų apdorojimo architektūra išnaudojanti grafinio procesoriaus resursus.
PĮ	programinė įranga.
C/C++	programavimo kalba.
Java	programavimo kalba.
OpenMP	programavimo standartas, skirtas realizuoti lygiagretiesiems algoritmams bendros atminties kompiuteriuose.
GPU	grafinis procesorius (angl. Graphics processing unit).
UML	vieninga modeliavimo kalba (angl. Unified Modelling Language).

7. LITERATŪROS SĄRAŠAS

[1] „SUSIDŪRIMŲ PAIEŠKOS, NAUDOJANT LYGIAGREČIUS SKAIČIAVIMUS, METODŲ TYRIMAS“ [Žiūrėta 2015 11 02], prieiga internete http://vddb.laba.lt/fedora/get/LT-eLABa-0001:E.02~2013~D_20130826_104837-08217/DS.005.0.02.ETD

[2] „LYGIAGRETIEJI SKAIČIAVIMAI NAUDOJANT VAIZDO PLOKŠTES“ [Žiūrėta 2015 11 02], prieiga internete http://vddb.laba.lt/fedora/get/LT-eLABa-0001:E.02~2013~D_20130801_141911-32550/DS.005.0.01.ETD

[3] „NVIDIA Unveils CUDA™-The GPU Computing Revolution Begins“ [Žiūrėta 2015 11 14], prieiga internete http://www.nvidia.com/object/IO_37226.html

[4] „Introduction to OpenMP“ [Žiūrėta 2015 11 14], prieiga internete https://community.topcoder.com/tc?module=Static&d1=features&d2=091106

[5] „What is OpenMP“ [Žiūrėta 2015 11 14], prieiga internete http://openmp.org/openmp-faq.html#WhatIs

[6] „CUDA“ [Žiūrėta 2015 11 14], prieiga internete https://en.wikipedia.org/wiki/CUDA

[7] „GPSME toolkit“ [Žiūrėta 2015 11 14], prieiga internete http://ansmart.co.uk/?q=node/54

[8] „GPSME“ [Žiūrėta 2015 11 14], prieiga internete http://www.gp-sme.eu/

[9] „OpenMPC: Extended OpenMP Programming and Tuning for GPUs“ [Žiūrėta 2015 11 15], prieiga internete https://engineering.purdue.edu/paramnt/OpenMPC

[10] „MINT“ [Žiūrėta 2015 11 12], prieiga internete https://sites.google.com/site/mintmodel/

[11] „C to CUDA“ [Žiūrėta 2015 11 12], prieiga internete https://sites.google.com/site/mintmodel/home/c-to-cuda-translation

[12] „Parallel computing technology application to Curonian Lagoon ecological model“ [Žiūrėta 2015 11 12], prieiga internete http://vddb.library.lt/fedora/get/LT-eLABa-0001:E.02~2005~D_20050614_122654-32066/DS.005.0.02.ETD

[13] „High-performance particle simulation using CUDA“ [Žiūrėta 2015 11 10], prieiga internete http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-118776

[14] „Amdahl's law“ [Žiūrėta 2017 03 15], prieiga internete https://en.wikipedia.org/wiki/Amdahl%27s_law

[15] „Lygiagretieji skaičiavimai“ [Žiūrėta 2017 03 10], prieiga internete http://kopustas.elen.ktu.lt/studentai/lib/exe/fetch.php?media=lygiagretieji_skaiciavimai.ppt

[16] „CUDA Kernel Overhead“ [Žiūrėta 2017 03 10], prieiga internete https://www.cs.virginia.edu/~mwb7w/cuda_support/kernel_overhead.html

[17] „OpenMP micro benchmarking“ [Žiūrėta 2017 03 10], prieiga internete https://www.epcc.ed.ac.uk/sites/default/files/PDF/ewomp99paper.pdf

[18] „A Microbenchmark Suite for OpenMP 2.0“ [Žiūrėta 2017 03 10], prieiga internete https://www.epcc.ed.ac.uk/sites/default/files/PDF/OpenMPmicro2.pdf

[19] „Matrica (matematika)“ [Žiūrėta 2017 03 10], prieiga internete https://lt.wikipedia.org/wiki/Matrica_(matematika)

[20] „Pi formulas“ [Žiūrėta 2017 03 15], prieiga internete http://mathworld.wolfram.com/PiFormulas.html

[21] „OpenMP & MPI“ [Žiūrėta 2017 04 10], prieiga internete https://www.eecis.udel.edu/~cavazos/cisc879/Lecture-03.pdf

[22] „5KK73 GPU assignment website 2014/2015“ [Žiūrėta 2017 03 10], prieiga internete http://www.es.ele.tue.nl/~mwijtvliet/5KK73/?page=mmcuda

[23] „Running a parallel matrix multiplication program using CUDA on FutureGrid“ [Žiūrėta 2017 04 10], prieiga internete https://kb.iu.edu/d/bcgu

[24] „OpenMP PI“ [Žiūrėta 2017 03 15], prieiga internete http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/91-pthreads/openMP.html

[25] „Computation of pi in CUDA“ [Žiūrėta 2017 04 15], prieiga internete http://math.oregonstate.edu/~mpesz/teaching/654_F09/students/lab7_Computation_of_pi_in_CUDA_Patil.ppt

[26] „Implementing Max Reduce in Cuda“ [Žiūrėta 2017 04 15], prieiga internete http://stackoverflow.com/questions/17371275/implementing-max-reduce-in-cuda

[27] „Programming GPUs with CUDA“ [Žiūrėta 2017 04 15], priega internete https://wiki.scinet.utoronto.ca/wiki/images/9/93/CUDA_advanced2013.pdf

[28] „Bitonic Sort: CUDA“ [Žiūrėta 2017 04 16], prieiga internete https://www.cs.rutgers.edu/~venugopa/parallel_summer2012/cuda_bitonic.html

[29] „Bitonic Sort: CUDA“ [Žiūrėta 2017 04 16], prieiga internete https://gist.github.com/mre/1392067

[30] „Bitonic Sort on CUDA“ [Žiūrėta 2017 04 16], prieiga internete https://gist.github.com/mre/1392067

[31] „Bitonic Sort: OpenMP Implementation“ [Žiūrėta 2017 04 16] https://www.cs.rutgers.edu/~venugopa/parallel_summer2012/bitonic_openmp.html

[32] „Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms“, 1994. Autoriai: Vipin Kumar, Ananth Grama, Anshul Gupta, George Karpis. ISBN 0805331700.

8. PRIEDAI

1 Priedas. OpenMP maksimalios reikšmės paieška

```
//vektoriuje
#include "stdafx.h"
#include <omp.h>
#include <chrono>
#include "Utils.h"
#include <random>
#include <algorithm>
#include <iterator>
#include <stdio.h>

int main()
{
    //int sets[] = { 1000000, 1048576, 1594323, 1679616, 1953125,
2097152, 4782969, 5764801, 9765625, 10000000, 10077696 };
    //int totalSet = 11;
    int sets[] = { 100000000, 150000000, 200000000, 250000000,
300000000 };
    int totalSet = 1;
    int currentSet = 0;
    int maxThreads = 80;
    int *v;

    Utils u;
    u.setFileName("MaxRadimas_vectorius.txt");
    u.writeHeader("");
    cout << u.getFileName() << endl;

    int threadNum = 0;
    while (threadNum <= maxThreads) {
        if (threadNum < 20)
            threadNum++;
        else
            threadNum += 20;
        //for (int threadNum = 60; threadNum <= maxThreads;
threadNum+= 20) {
        cout << "Thread num " << threadNum << " is " << maxThreads
<< endl;
        srand(0);
        omp_set_num_threads(threadNum);

        while (currentSet < totalSet) {
            int j = sets[currentSet];

            std::vector<int> v;
            for (int i = 0; i < j; i++) {
                int g = rand() % INT_MAX;
                v.push_back(g);
            }
            cout << "uzpildyta" << endl;

            int maxa = 0;
            u.start();

         #pragma omp parallel
            {
                //cout << "thread number = " << omp_get_thread_num()
<< endl;
                auto my_value = 0;
```

```
             #pragma omp for
                for (int f = 0; f < v.size(); f++) {
                    if (v.at(f) > my_value)
                        my_value = v.at(f);
                }

            #pragma omp critical
                {
                   if (my_value > maxa)
                     maxa = my_value;
                }
            }
            u.stop();
            cout << endl << "MAX = " << maxa << endl;
            u.write(threadNum, j, u.getSeconds());
            cout << u.getSeconds() << endl;
            currentSet++;
        }
        currentSet = 0;
    }
}
```

```
//masyve
#include "stdafx.h"
#include <omp.h>
 #include <chrono>
 #include "Utils.h"
 #include <random>
 #include <algorithm>
 #include <iterator>
 #include <stdio.h>

 int main()
 {
    int sets[] = { 1000000, 150000000, 200000000, 250000000, 300000000
};
    int totalSet = 5;
    int currentSet = 0;
    int maxThreads = 80;
    Utils u;
    u.setFileName("MaxRadimas_masyve.txt");
    u.writeHeader("");
    cout << u.getFileName() << endl;

    int threadNum = 1;
    while (threadNum <= maxThreads) {


        for (int threadNum = 60; threadNum <= maxThreads; threadNum+=
20) {
        cout << "Thread num " << threadNum << " is " << maxThreads <<
endl;
        srand(0);

        omp_set_num_threads(threadNum);
        while (currentSet < totalSet) {
            int j = sets[currentSet];
            int *v = new int[j];
            for (int i = 0; i < j; i++) {
                v[i] = rand() % INT_MAX;
                cout << " " << v[i];
            }
```

```
            cout << endl << "uzpildyta" << endl;
            int maxa = 0;
            u.start();
            #pragma omp parallel
            {
                 cout << "thread number = " << omp_get_thread_num() <<
endl;
                int my_value = 0;
                #pragma omp for
                for (int f = 0; f < j; f++) {
                     cout << "thread number = " <<
omp_get_thread_num() << endl;
                    if (v[f] > my_value)
                        my_value = v[f];
                }
                 #pragma omp critical
                {
                    if (my_value > maxa)
                        maxa = my_value;
                }
            }
            u.stop();
            delete v;
            cout << j << endl << "MAX = " << maxa << endl;
            u.write(threadNum, j, u.getSeconds());
            cout << u.getSeconds() << endl;
            currentSet++;
        }
        currentSet = 0;
        if (threadNum < 20)
            threadNum++;
        else
            threadNum += 20;

    }
 }
```

2 Priedas. OpenMP matricų daugyba

```
#include "stdafx.h"
#include <omp.h>
#include <iostream>
#include "Utils.h"
#include <iomanip>
using namespace std;

int main(int argc, char *argv[])
{
    int sets[] = { 100, 200, 300, 400, 500, 600, 700, 800, 900,
1000};
    int totalSet = 10;
    int currentSet = 0;

    int maxThreads = 40;

    Utils u;
    u.setFileName("MatrixMultiply_pararell.txt");
    u.writeHeader("");
    cout << u.getFileName() << endl;

    for (int threadNum = 40; threadNum <= maxThreads; threadNum+=20)
{
```

```
        cout << "Thread num " << threadNum << " is " << maxThreads
<< endl << endl;
        cout << (threadNum * 100 / maxThreads) << endl;
        omp_set_num_threads(threadNum);

        while (currentSet < totalSet) {
            int size = sets[currentSet];
            double **a = new double*[size];
            double **b = new double*[size];
            double **c = new double*[size];

            for (int i = 0; i < size; i++) {
                a[i] = new double[size];
                for (int j = 0; j < size; j++)
                    a[i][j] = i + j;
            }

            for (int i = 0; i < size; i++) {
                b[i] = new double[size];
                for (int j = 0; j < size; j++)
                    b[i][j] = i*j;
            }

            for (int i = 0; i < size; i++) {
                c[i] = new double[size];
                for (int j = 0; j < size; j++)
                    c[i][j] = 0;
            }

            int i, j, k;
            u.start();
            #pragma omp parallel shared(a,b,c) private(i,j,k)
            {
                //printf("Thread %d starting matrix multiply...\n",
omp_get_thread_num());
                for (i = 0; i < size; i++)
                {
                    //printf("Multiplying...  %d \n",
omp_get_thread_num());
                    #pragma omp for
                    for (j = 0; j < size; j++) {
                        for (k = 0; k < size; k++) {
                            c[i][j] += a[i][k] * b[k][j];
                        }
                    }
                }
            }

            u.stop();
            u.write(threadNum, size, u.getSeconds());
            cout << u.getSeconds() << endl;

            for (int i = 0; i < size; i++) {
                delete a[i];
                delete b[i];
                delete c[i];
            }

            delete a;
            delete b;
            delete c;
            currentSet++;
        }
        currentSet = 0;
    }
```

```
}
```

3 Priedas. OpenMP π skaičiavimas

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <omp.h>
#include "Utils.h"
#include <iomanip>
using namespace std;


int main(int argc, char *argv[])
{
    int sets[] = { 1000000, 2000000, 3000000, 4000000, 5000000,
6000000, 7000000, 8000000, 9000000, 10000000 };
    int totalSet = 1;
    int currentSet = 0;
    int maxThreads = 1;

    Utils u;
    u.setFileName("PiSkaiciavimas_pararell.txt");
    u.writeHeader("");

    for (int threadNum = 1; threadNum <= maxThreads; threadNum++) {
        cout << "Thread num " << threadNum << " is " << maxThreads
<< endl;
        omp_set_num_threads(threadNum);

        while (currentSet < totalSet) {
            int N = sets[currentSet];
            double pi;
            double step, x;

            step = 1.0 / N;
            pi = 0.0;
            u.start();
#pragma omp parallel
            {
                int i;
                double mypi, x;
                mypi = 0.0;

#pragma omp for
                for (i = 0; i < N; i++)
                {
                    x = step*(i + 0.5);
                    mypi = mypi + (4.0 / (1 + x*x));
                }
#pragma omp critical
                {
                    pi = pi + mypi;
                }
            }
            u.stop();
            pi *= step;
            cout << "Computed Pi = " << setprecision(10) << pi <<
endl << endl;
            u.write(threadNum, sets[currentSet], u.getSeconds());
            cout << u.getSeconds() << endl;
            currentSet++;
```

```
        }
        currentSet = 0;
    }
}
```

4 Priedas. OpenMP rikiavimas

```
#include "stdafx.h"
#include <iostream>
#include <omp.h>
#include "Utils.h"
using namespace std;

#define MAX(A, B) (((A) > (B)) ? (A) : (B))
#define MIN(A, B) (((A) > (B)) ? (B) : (A))
#define UP 0
#define DOWN 1

void bitonic_sort_seq(int start, int length, int *seq, int flag);
void bitonic_sort_par(int start, int length, int *seq, int flag);
void swap(int *a, int *b);

int m;

int main(int argc, char ** argv)
{
    int sets[] = { 32768, 65536, 131072, 262144 ,524288, 1048576,
2097152 ,4194304 };
    int totalSet = 8;
    int currentSet = 0;
    int maxThreads = 20;

    Utils u;
    u.setFileName("BitonicSort_pararell2.txt");
    u.writeHeader("");

    for (int threadNum = 1; threadNum <= maxThreads; threadNum *=2) {
        srand(0);
        cout << "Thread num " << threadNum << " is " << maxThreads <<
endl;
        omp_set_num_threads(threadNum);

        while (currentSet < totalSet) {
            int N = sets[currentSet];

            int * arr;

            arr = new int[N];
            for (int i = 0; i < N; i++)
                arr[i] = rand() % INT_MAX;

            cout << "Uzpildyta" << endl;

            int i, j;
            int flag;

            // start

            //numThreads = omp_get_max_threads();

            // making sure input is okay
            if (N < threadNum * 2)
```

```
            {
                printf("The size of the sequence is less than 2 * the
number of processes.\n");
                exit(0);
            }

            // the size of sub part
            m = N / threadNum;

            // make the sequence bitonic - part 1
            u.start();
            for (i = 2; i <= m; i = 2 * i)
            {
                #pragma omp parallel for shared(i, arr) private(j,
flag)
                for (j = 0; j < N; j += i)
                {
                    if ((j / i) % 2 == 0)
                        flag = UP;
                    else
                        flag = DOWN;
                    bitonic_sort_seq(j, i, arr, flag);
                }
            }

            // make the sequence bitonic - part 2
            for (i = 2; i <= threadNum; i = 2 * i)
            {
                for (j = 0; j < threadNum; j += i)
                {
                    if ((j / i) % 2 == 0)
                        flag = UP;
                    else
                        flag = DOWN;
                    bitonic_sort_par(j*m, i*m, arr, flag);
                }
                #pragma omp parallel for shared(j)
                for (j = 0; j < threadNum; j++)
                {
                    if (j < i)
                        flag = UP;
                    else
                        flag = DOWN;
                    bitonic_sort_seq(j*m, m, arr, flag);
                }
            }
            u.stop();

            // print a sequence
            /*for (i = 0; i < N; i++) {
                cout << arr[i] << "  ";
            }*/

            bool passed = true;
                for (int i = 1; i < N; i++) {
                    if (arr[i - 1] > arr[i]) {
                        passed = false;
                    }
                }
                if (passed){
                    cout << u.getSeconds() << "; ";
                    cout << threadNum << "    " << N << "    " <<
u.getSeconds() << endl;
                    u.write(threadNum, N, u.getSeconds());
                }
```

```
                else
                    cout << "; ";
            cout << endl;

            delete arr;


            //u.write(threadNum, N, u.getSeconds());

            currentSet++;
        }
        currentSet = 0;
    }




    return 0;
}



void swap(int *a, int *b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

void bitonic_sort_par(int start, int length, int *seq, int flag)
{
    int i;
    int split_length;

    if (length == 1)
        return;

    if (length % 2 != 0)
    {
        printf("The length of a (sub)sequence is not divided by
2.\n");
        exit(0);
    }

    split_length = length / 2;

    // bitonic split
    #pragma omp parallel for shared(seq, flag, start, split_length)
private(i)
    for (i = start; i < start + split_length; i++)
    {
        if (flag == UP)
        {
            if (seq[i] > seq[i + split_length])
                swap(&seq[i], &seq[i + split_length]);
        }
        else
        {
            if (seq[i] < seq[i + split_length])
                swap(&seq[i], &seq[i + split_length]);
        }
    }

    if (split_length > m)
```

```
    {
        // m is the size of sub part-> n/numThreads
        bitonic_sort_par(start, split_length, seq, flag);
        bitonic_sort_par(start + split_length, split_length, seq,
flag);
    }

    return;
}


void bitonic_sort_seq(int start, int length, int *seq, int flag)
{
    int i;
    int split_length;

    if (length == 1)
        return;

    if (length % 2 != 0)
    {
        printf("error\n");
        exit(0);
    }

    split_length = length / 2;

    // bitonic split
    for (i = start; i < start + split_length; i++)
    {
        if (flag == UP)
        {
            if (seq[i] > seq[i + split_length])
                swap(&seq[i], &seq[i + split_length]);
        }
        else
        {
            if (seq[i] < seq[i + split_length])
                swap(&seq[i], &seq[i + split_length]);
        }
    }

    bitonic_sort_seq(start, split_length, seq, flag);
    bitonic_sort_seq(start + split_length, split_length, seq, flag);
}
```

5 Priedas. CUDA maksimalios reikšmės paieška

```
#include <iostream>
#include <cuda.h>
#include <cuda_runtime.h>
#include <random>
#include <algorithm>
#include <iterator>
#include <stdio.h>
#include <vector>
#include <sstream>
#include "Utils.h"
using namespace std;


__global__ void find_maximum_kernel(int *arr, int *max, int *mutex,
```

```
unsigned int n) {
    unsigned int index = threadIdx.x + blockIdx.x * blockDim.x;
    unsigned int grid = gridDim.x * blockDim.x;
    unsigned int offset = 0;
    __shared__ int cache[1024];
    float temp = -1.0;
    while (index + offset < n) {
        temp = fmaxf(temp, arr[index + offset]);
        offset += grid;
    }

    cache[threadIdx.x] = temp;
    __syncthreads();

    unsigned int i = blockDim.x / 2;
    while (i != 0) {
        if (threadIdx.x < i) {
            cache[threadIdx.x] = fmaxf(cache[threadIdx.x],
cache[threadIdx.x + i]);
        }
        __syncthreads();
        i /= 2;
    }

    if (threadIdx.x == 0) {
        while (atomicCAS(mutex, 0, 1) != 0)
            ;
        *max = fmaxf(*max, cache[0]);
        atomicExch(mutex, 0);
    }
}

int main() {
    Utils u;
    u.setFileName("MaxRadimas_cuda_8.txt");
    u.writeHeader("");
    cout << u.getFileName() << endl;

    int sets[] = { 100000000, 150000000, 200000000, 250000000,
300000000 };
    int totalSet = 5;
    int currentSet = 0;

    int threadSet[] = { 1, 32, 64, 128, 256, 512, 1024 };
    int totalCudaThreads = 7;

    int blockSet[] = { 1024, 512, 256, 128, 64, 32, 1 };
    int totalBlokcs = 7;
    stringstream ss;

    cout << "; ";
    for (int p = 0; p < totalSet; p++) {
        cout << sets[p] << "; ";
    }
    cout << endl;
    for (int blockIndex = 0; blockIndex < totalBlokcs; blockIndex++)
{
        int blocks = blockSet[blockIndex];
        for (int threadIndex = 0; threadIndex < totalCudaThreads;
threadIndex++) {
            srand(0);
            int threads = threadSet[threadIndex];
            cudaError_t err;
            cout << "<< " << blocks << ", " << threads << " >> ; ";
            ss << "<< " << blocks << ", " << threads << " >> ; ";
```

```
            while (currentSet < totalSet) {
                int j = sets[currentSet];
                int* v = new int[j];
                for (int i = 0; i < j; i++) {
                    v[i] = rand() % INT_MAX;
                    //cout << " " << v[i];
                }
                cout << " ";

                int *dev_in;

                err = cudaMalloc((void **)&dev_in, j * sizeof(int));
                err = cudaMemcpy(dev_in, v, j * sizeof(int),
cudaMemcpyHostToDevice);

                int *h_max = new int(0);
                int *dev_maximumas;
                int *dev_mutex;

                cudaMalloc((void**)&dev_maximumas, sizeof(int));
                cudaMalloc((void**)&dev_mutex, sizeof(int));
                cudaMemset(dev_maximumas, 0, sizeof(int));
                cudaMemset(dev_mutex, 0, sizeof(int));

                u.start();
                find_maximum_kernel << <blocks, threads >> >(dev_in,
dev_maximumas, dev_mutex, j);
                err = cudaDeviceSynchronize();
                u.stop();
                err = cudaMemcpy(h_max, dev_maximumas, sizeof(int),
cudaMemcpyDeviceToHost);
                //cout << "Max = " << *h_max <<endl;

                //cout << u.getSeconds() << "; "; //"(TIME = " << j
<< " ----> " << u.getSeconds() << ") ";
                //ss << u.getSeconds() << "; ";

                if (err == cudaSuccess)
                    cout << u.getSeconds() << "; ";
                else
                    cout << "-" << u.getSeconds() << "; ";


                                    cudaError_t error =
cudaGetLastError();
                                    if (error != cudaSuccess) {
                                        cout << "-" << u.getSeconds() <<
"; ";
                                        // print the CUDA error message
and exit
                                        printf(" %d: CUDA error: %s
",j, cudaGetErrorString(error));
                                        //exit(-1);
                                    } else {
                                        cout << u.getSeconds() << "; ";
                                    }
                currentSet++;
                cudaFree(dev_in);
                cudaFree(dev_maximumas);
                cudaFree(dev_mutex);
                delete h_max;
                delete v;
                cudaDeviceReset();
            }
            currentSet = 0;
```

```
            cout << endl;
        }
    }
    cout << endl << "End." << endl;
    return 0;
}
```

6 Priedas. CUDA matricų daugyba

```
#include <sstream>
#include <stdio.h>
#include <iostream>
#include "Utils.h"
using namespace std;

__global__ void MatrixMultiply(float* A, float* B, float* C, int N)
{
    float cc = 0.0;
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    if (row > N || col > N)
        return;
    for (int e = 0; e < N; ++e)
        cc += (A[row * N + e]) * (B[e * N + col]);
    C[row * N + col] = cc;
}

int main(int argc, char* argv[]) {
    Utils u;
    u.setFileName("MatrixMultiply_CUDA.txt");
    u.writeHeader("");
    cout << u.getFileName() << endl;

    int sets[] = { 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
};
    int totalSet = 5;
    int currentSet = 0;

    dim3 blockSet[] = {
        dim3(1024, 2),
        dim3(2, 1024),
        dim3(2, 512),
        dim3(512, 2),
        dim3(4, 256),
        dim3(256,4),
        dim3(8, 128),
        dim3(128, 8),
        dim3(16, 64),
        dim3(64, 16),
        dim3(32, 32),
        dim3(16, 16),
        dim3(8, 8),
        dim3(4, 4),
        dim3(2, 2),
        dim3(1, 1) };
    int totalBlocks = 16;
    int currentThreads = 0;

    dim3 gridSet[] = {
        dim3(256, 256),
        dim3(256, 4),
        dim3(4, 256),
        dim3(64, 64),
```

```
        dim3(32, 32),
        dim3(16, 16),
        dim3(8, 8),
        dim3(1, 1)
    };

    int totalGrids = 8;
    stringstream ss;

    cout << "; ";
    for (int p = 0; p < totalSet; p++) {
        cout << sets[p] << "; ";
    }
    cout << endl;

    for (int blockIndex = 0; blockIndex < totalBlocks; blockIndex++)
{
        dim3 blocks = blockSet[blockIndex];
        for (int gridIndex = 0; gridIndex < totalGrids; gridIndex++)
{
            dim3 grids = gridSet[gridIndex];
            srand(0);
            //int threads = threadSet[threadIndex];
            cout << "<< (" << grids.x << "," << grids.y << "), " <<
"(" << blocks.x << "," << blocks.y << ")" << " >> ;";
            //ss << "<< " << blocks << ", " << blocks << " >> ; ";

            while (currentSet < totalSet) {
                int N = sets[currentSet];


                float* A = (float*)malloc(N * N * sizeof(float));
                float* B = (float*)malloc(N * N * sizeof(float));
                float* C = (float*)malloc(N * N * sizeof(float));


                for (int i = 0; i < N; i++)
                    for (int j = 0; j < N; j++) {
                        A[i * N + j] = i + j;
                        B[i * N + j] = i*j;
                        C[i * N + j] = 0;
                    }



                size_t size = N * N * sizeof(float);
                float* d_A;

                //cout << N << "<< (" << grids.x << "," <<grids.y <<
"), " << "(" << blocks.x << "," << blocks.y << ")" << " >>";

                cudaError_t err = cudaMalloc(&d_A, size);
                //printf("CUDA malloc A: %s\n",
cudaGetErrorString(err));
                err = cudaMemcpy(d_A, A, size,
cudaMemcpyHostToDevice);
                //printf("Copy A to device: %s\n",
cudaGetErrorString(err));
                float* d_B;
                err = cudaMalloc(&d_B, size);

                //printf("CUDA malloc B: %s\n",
cudaGetErrorString(err));
                err = cudaMemcpy(d_B, B, size,
cudaMemcpyHostToDevice);
```

```
                //printf("Copy B to device: %s\n",
cudaGetErrorString(err));
                // Allocate C in device memory
                float* d_C;
                err = cudaMalloc(&d_C, size);
                //printf("CUDA malloc C: %s\n",
cudaGetErrorString(err));


                //dim3 dimBlock(1024, 1);
                //dim3 dimGrid((N + dimBlock.x - 1) / dimBlock.x, (N
+ dimBlock.y - 1) / dimBlock.y, 1);
                //dim3 dimGrid(1, 1);
                u.start();
                MatrixMultiply << <grids, blocks >> >(d_A, d_B, d_C,
N);
                err = cudaThreadSynchronize();
                u.stop();


                err = cudaMemcpy(C, d_C, size,
cudaMemcpyDeviceToHost);
                //printf("Copy C off of device: %s\n",
cudaGetErrorString(err));

                cudaFree(d_A);
                cudaFree(d_B);
                cudaFree(d_C);

                //              for (int i = 0; i < min(10, N); i++)
{
                //                  for (int j = 0; j < min(10, N);
j++)
                //                      printf("%f ", A[i * N + j]);
                //                  printf("\n");
                //              }
                //              printf("\n");
                //              for (int i = 0; i < min(10, N); i++)
{
                //                  for (int j = 0; j < min(10, N);
j++)
                //                      printf("%f ", B[i * N + j]);
                //                  printf("\n");
                //              }
                //              printf("\n");
                //              for (int i = 0; i < min(5, N); i++)
{
                //                  for (int j = 0; j < min(5, N);
j++)
                //                      printf("%f ", C[i * N + j]);
                //                  printf("\n");
                //              }
                //              printf("\n");
                delete A;
                delete B;
                delete C;
                currentSet++;
                if (err == cudaSuccess)
                    cout << u.getSeconds() << "; ";
                else
                    cout << "-" << u.getSeconds() << "; ";

                //                              cudaError_t error =
cudaGetLastError();
                //                                  if (error !=
```

```
cudaSuccess) {
                //                                          cout << "-"
<< u.getSeconds() << "; ";
                //                                          // print the
CUDA error message and exit
                //                                          //printf("
%d: CUDA error: %s  ",N, cudaGetErrorString(error));
                //                                          //cout <<
"block index = " << blockIndex << "  grid index = " << gridIndex <<
endl;
                //                                          //cout <<
"<< (" << grids.x << "," <<grids.y << "), " << "(" << blocks.x <<
"," << blocks.y << ")" << " >>";
                //                                          //exit(-1);
                //                                      }
                //                                      else{
                //                                          cout <<
u.getSeconds() << "; ";
                //                                      }
                cudaDeviceReset();

            }
            currentSet = 0;
            cout << endl;
        }
    }
    cout << endl << "End." << endl;
    return 0;
}
```

7 Priedas. CUDA π skaičiavimas

```
#include <iostream>
#include <cuda.h>
#include <cuda_runtime.h>
#include <random>
#include <algorithm>
#include <iterator>
#include <stdio.h>
#include <sstream>
#include "Utils.h"
#include <iomanip>
using namespace std;

__global__ void cal_pi(double *sum, double nbin, double step, int
nthreads, int nblocks) {
    int i;
    double x;
    int idx = blockIdx.x * blockDim.x + threadIdx.x;  // Sequential
thread index across the blocks
    for (i = idx; i < nbin; i += nthreads * nblocks) {
        x = (i + 0.5) * step;
        sum[idx] += 4.0 / (1.0 + x * x);
    }
}

int main(void) {

    Utils u;
    u.setFileName("pi_cuda_.txt");
    u.writeHeader("");
    cout << u.getFileName() << endl;
```

```
    int sets[] = { 1000000, 2000000, 3000000, 4000000, 5000000,
6000000, 7000000, 8000000, 9000000, 10000000 };
    int totalSet = 10;
    int currentSet = 0;

    int threadSet[] = { 1, 32, 64, 128, 256, 512 };
    int totalCudaThreads = 6;
    int currentThreads = 0;

    int blockSet[] = { 1, 32, 64, 128, 256, 512 };
    int totalBlokcs = 6;
    int currentBlocks = 6;

    int maxThreads = 1;

    stringstream ss;

    cout << "; ";
    for (int p = 0; p < totalSet; p++) {
        cout << sets[p] << "; ";
    }
    cout << endl;

    for (int blockIndex = 0; blockIndex < totalBlokcs; blockIndex++)
{

        int blocks = blockSet[blockIndex];
        for (int threadIndex = 0; threadIndex < totalCudaThreads;
threadIndex++) {
            srand(0);
            int threads = threadSet[threadIndex];
            cout << "<< " << blocks << ", " << threads << " >> ; ";
            ss << "<< " << blocks << ", " << threads << " >> ; ";
            while (currentSet < totalSet) {
                int j = sets[currentSet];
                cudaError_t err;
                double *sumHost, *sumDev;

                double step = 1.0 / j;  // Step size
                size_t size = blocks * threads * sizeof(double);
                sumHost = (double *)malloc(size);
                cudaMalloc((void **)&sumDev, size);
                cudaMemset(sumDev, 0, size);
                u.start();
                cal_pi << <blocks, threads >> >(sumDev, j, step,
threads, blocks); // call CUDA kernel
                err = cudaThreadSynchronize();
                u.stop();
                double pi = 0.0;
                cudaMemcpy(sumHost, sumDev, size,
cudaMemcpyDeviceToHost);
                for (int tid = 0; tid < threads * blocks; tid++)
                    pi += sumHost[tid];
                pi *= step;

                if (err == cudaSuccess)
                    cout << u.getSeconds() << "; ";
                else {
                    cout << "-" << u.getSeconds() << "; ";
                    return 0;
                }
                //printf("PI = %f\n", pi);
                free(sumHost);
                cudaFree(sumDev);
```

```
                currentSet++;

                cudaError_t error = cudaGetLastError();
                if (error != cudaSuccess) {
                    // print the CUDA error message and exit
                    printf(" %d: CUDA error: %s  ", j,
cudaGetErrorString(error));
                    exit(-1);
                }
            }
            currentSet = 0;
            cout << endl;
        }
    }
    cout << endl << "End." << endl;
    return 0;
}
```

8 Priedas. CUDA rikiavimas

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <cuda.h>
#include <cuda_runtime.h>
#include <random>
#include <algorithm>
#include <iterator>
#include <stdio.h>
#include <sstream>
#include "Utils.h"
using namespace std;

__global__ void bitonic_sort(float *dev_values, int j, int k) {
    unsigned int i, ixj;
    i = threadIdx.x + blockDim.x * blockIdx.x;
    ixj = i ^ j;

    /* The threads with the lowest ids sort the array. */
    if ((ixj) > i) {
        if ((i & k) == 0) {
            /* Sort ascending */
            if (dev_values[i] > dev_values[ixj]) {
                float temp = dev_values[i];
                dev_values[i] = dev_values[ixj];
                dev_values[ixj] = temp;
            }
        }
        if ((i & k) != 0) {
            /* Sort descending */
            if (dev_values[i] < dev_values[ixj]) {
                float temp = dev_values[i];
                dev_values[i] = dev_values[ixj];
                dev_values[ixj] = temp;
            }
        }
    }
}

int main(void) {
    Utils u;
    u.setFileName("BitonicSort.txt");
    u.writeHeader("");
```

```
    cout << u.getFileName() << endl;

    int sets[] = { 32768, 65536, 131072, 262144 ,524288, 1048576,
2097152 ,4194304 };
    int totalSet = 8;
    int currentSet = 0;

    int threadSet[] = { 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024,
2048 ,4096 };
    int totalCudaThreads = 13;

    int blockSet[] = { 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024,
2048 ,4096 };
    int totalBlokcs = 13;

    stringstream ss;
    cout << "; ";
    for (int p = 0; p<totalSet; p++) {
        cout << sets[p] << "; ";
    }
    cout << endl;

    for (int blockIndex = 0; blockIndex < totalBlokcs; blockIndex++) {
        int blocks = blockSet[blockIndex];
        for (int threadIndex = 0; threadIndex < totalCudaThreads;
threadIndex++) {
            srand(0);
            int threads = threadSet[threadIndex];
            cudaError_t err;
            cout << "<< " << blocks << ", " << threads << " >> ; ";
            ss << "<< " << blocks << ", " << threads << " >> ; ";
            while (currentSet < totalSet) {
                int N = sets[currentSet];

                float *values = new float[N];
                for (int i = 0; i < N; i++) {
                    values[i] = rand() % INT_MAX;
                    //cout << " " << v[i];
                }

                float *dev_values;
                size_t size = N * sizeof(float);

                cudaMalloc((void**)&dev_values, size);
                cudaMemcpy(dev_values, values, size,
cudaMemcpyHostToDevice);

                dim3 block(blocks, 1);
                dim3 thread(threads, 1);

                u.start();
                int j, k;
                for (k = 2; k <= N; k <<= 1) {
                    for (j = k >> 1; j > 0; j = j >> 1) {
                        bitonic_sort << <block, thread >>
>(dev_values, j, k);
                    }
                }
                u.stop();
                cudaMemcpy(values, dev_values, size,
cudaMemcpyDeviceToHost);
                cudaFree(dev_values);

                bool passed = true;
                for (int i = 1; i < N; i++) {
```

```
                    if (values[i - 1] > values[i]) {
                        passed = false;
                    }
                }
                if (passed)
                    cout << u.getSeconds() << "; ";
                else
                    cout << "; ";
                currentSet++;

                delete values;
                cudaDeviceReset();
            }
            currentSet = 0;
            cout << endl;
        }
    }
    cout << endl << "End." << endl;
    return 0;
}
```

9.Priedas. OpenMP maksimalios reikšmės paieškos diagramų duomenys

17 lentelė. Maksimalios reikšmės algoritmo efektyvumo koeficientas

	1E+08	1,5E+08	2E+08	2,5E+08	3E+08
1	1	1	1	1	1
2	0,886473	0,926087	0,961543	0,95238	0,923335
3	0,885459	0,868906	0,798326	0,867163	0,882633
4	0,827709	0,825386	0,82388	0,861887	0,836707
5	0,557757	0,688937	0,662925	0,649671	0,721689
6	0,44988	0,595026	0,631355	0,585205	0,59842
7	0,343823	0,412097	0,446441	0,492888	0,605405
8	0,409547	0,409389	0,411157	0,357569	0,46253
9	0,376036	0,395207	0,397795	0,387484	0,416801
10	0,270892	0,283922	0,279113	0,334681	0,340093
11	0,247493	0,376305	0,285257	0,285634	0,308965
12	0,256316	0,283009	0,280637	0,271556	0,279705
13	0,198232	0,206636	0,232018	0,260636	0,285312
14	0,208765	0,207737	0,214532	0,22498	0,272312
15	0,192269	0,192064	0,263205	0,220499	0,227141
16	0,185235	0,162952	0,21459	0,207935	0,206851
17	0,149849	0,151279	0,195475	0,192682	0,199765
18	0,136326	0,153947	0,164427	0,18064	0,190952
19	0,138414	0,171278	0,15849	0,188205	0,183098
20	0,125326	0,14793	0,147172	0,168716	0,177626
40	0,071592	0,078601	0,071553	0,079508	0,080799
60	0,045232	0,051195	0,053537	0,056751	0,045215
80	0,032568	0,04196	0,031668	0,042153	0,035789

18 lentelė. Maksimalios reikšmės spartinimo koeficientas

	1E+08	1,5E+08	2E+08	2,5E+08	3E+08

1	1	1	1	1	1
2	1,772946	1,852174	1,923087	1,904759	1,846669
3	2,656378	2,606717	2,394977	2,601488	2,647898
4	3,310835	3,301544	3,29552	3,447549	3,346828
5	2,788783	3,444687	3,314627	3,248354	3,608447
6	2,699279	3,570158	3,788132	3,511232	3,590521
7	2,406763	2,88468	3,125084	3,450215	4,237834
8	3,276377	3,275109	3,289259	2,86055	3,700243
9	3,384323	3,556863	3,580159	3,487356	3,751207
10	2,708918	2,839222	2,791126	3,346809	3,400929
11	2,722428	4,139355	3,137832	3,141975	3,398616
12	3,075788	3,39611	3,367645	3,258668	3,356466
13	2,577014	2,686272	3,016229	3,388266	3,709057
14	2,922715	2,908315	3,003448	3,149721	3,812372
15	2,884036	2,880965	3,948078	3,307484	3,407121
16	2,96376	2,607237	3,433445	3,326959	3,309613
17	2,54743	2,571744	3,323073	3,275589	3,396003
18	2,453872	2,771046	2,95969	3,251519	3,437137
19	2,629859	3,254276	3,011304	3,575886	3,478856
20	2,506527	2,958603	2,943448	3,374325	3,552523
40	2,863669	3,144051	2,862126	3,180316	3,231958
60	2,713903	3,071705	3,212242	3,405068	2,712919
80	2,605419	3,356775	2,533432	3,372223	2,863092

19 lentelė. Maksimalios reikšmės paieška pagal laiką

	1E+08	1,5E+08	2E+08	2,5E+08	3E+08
1	0,340622	0,510072	0,67689	0,861077	1,02923
2	0,192122	0,275391	0,351981	0,452066	0,557344
3	0,128228	0,195676	0,282629	0,330994	0,388697
4	0,102881	0,154495	0,205397	0,249765	0,307524
5	0,12214	0,148075	0,204213	0,265081	0,285228
6	0,12619	0,142871	0,178687	0,245235	0,286652
7	0,141527	0,176821	0,216599	0,249572	0,242867
8	0,103963	0,155742	0,205788	0,301018	0,278152
9	0,100647	0,143405	0,189067	0,246914	0,274373
10	0,125741	0,179652	0,242515	0,257283	0,302632
11	0,125117	0,123225	0,215719	0,274056	0,302838
12	0,110743	0,150193	0,200998	0,264242	0,306641
13	0,132177	0,189881	0,224416	0,254135	0,277491
14	0,116543	0,175384	0,225371	0,273382	0,269971
15	0,118106	0,177049	0,171448	0,260342	0,302082
16	0,114929	0,195637	0,197146	0,258818	0,310982
17	0,133712	0,198337	0,203694	0,262877	0,303071
18	0,13881	0,184072	0,228703	0,264823	0,299444
19	0,129521	0,156739	0,224783	0,240801	0,295853
20	0,135894	0,172403	0,229965	0,255185	0,289718
40	0,118946	0,162234	0,236499	0,270752	0,318454

60	0,12551	0,166055	0,210722	0,252881	0,379381
80	0,130736	0,151953	0,267183	0,255344	0,359482

10 Priedas. OpenMP matricų daugybos diagramų duomenys

20 lentelė. Matricų daugybos efektyvumo koeficientas

	100	200	300	400	500	600	700	800	900	1000
1	1	1	1	1	1	1	1	1	1	1
2	0,886725	1,183393	0,962087	0,873933	0,911686	1,043934	0,96413	0,974072	0,907994	0,939992
3	0,624793	1,008638	0,802375	0,553517	0,573624	0,830608	0,797009	0,852845	0,8682	0,821431
4	0,393393	0,586028	0,501224	0,4698	0,462852	0,699759	0,539539	0,675922	0,677044	0,738569
5	0,320083	0,557654	0,459157	0,39278	0,408091	0,493518	0,446075	0,567871	0,570746	0,638451
6	0,306824	0,545245	0,476372	0,413909	0,415953	0,495343	0,450309	0,565016	0,587794	0,628968
7	0,190046	0,456956	0,472196	0,405252	0,407421	0,491071	0,458816	0,559067	0,574339	0,623957
8	0,110666	0,521563	0,48223	0,386656	0,393816	0,428444	0,444136	0,528968	0,559675	0,589525
9	0,057379	0,233701	0,251808	0,229865	0,171653	0,279292	0,221531	0,28731	0,319771	0,34138
10	0,067657	0,291583	0,250105	0,23732	0,227283	0,279359	0,275676	0,306162	0,3033	0,333238
11	0,054918	0,201754	0,241239	0,225552	0,212516	0,231989	0,222334	0,239396	0,173902	0,280816
12	0,042274	0,136454	0,151746	0,118768	0,142871	0,162485	0,155714	0,180104	0,184982	0,19356
13	0,032886	0,155037	0,142353	0,119069	0,120048	0,148362	0,121128	0,149834	0,175208	0,190035
14	0,026201	0,164093	0,134713	0,117252	0,108406	0,190861	0,142917	0,173471	0,175347	0,168071
15	0,009087	0,152256	0,134228	0,122554	0,119131	0,148785	0,143259	0,176656	0,17219	0,186766
16	0,016253	0,143702	0,089629	0,119454	0,123016	0,143418	0,145978	0,16749	0,170493	0,183226
17	0,018773	0,137541	0,134311	0,065132	0,108277	0,114441	0,108655	0,156987	0,140483	0,18131
18	0,017876	0,11709	0,124005	0,114698	0,107363	0,128951	0,109785	0,16299	0,14312	0,140045
19	0,015716	0,122374	0,107437	0,092055	0,087273	0,11395	0,098122	0,109371	0,106299	0,175445
20	0,013625	0,113351	0,094857	0,108993	0,094404	0,117641	0,115013	0,137694	0,142258	0,140147
40	0,002375	0,011724	0,037422	0,042073	0,037516	0,045529	0,04204	0,049176	0,049556	0,047188
60	0,00272	0,024185	0,027212	0,030197	0,030373	0,037966	0,04009	0,044065	0,050763	0,040259
80	0,009076	0,028017	0,031129	0,025321	0,02951	0,028502	0,028203	0,033825	0,033942	0,037947

21 lentelė. Matricų daugybos spartinimo koeficientas

	100	200	300	400	500	600	700	800	900	1000
1	1	1	1	1	1	1	1	1	1	1
2	1,77345	2,366786	1,924174	1,747865	1,823371	2,087867	1,928261	1,948144	1,815988	1,879984
3	1,874378	3,025914	2,407126	1,660552	1,720872	2,491824	2,391026	2,558534	2,604661	2,464293
4	1,57357	2,344111	2,004897	1,879519	1,851406	2,799037	2,158158	2,703686	2,708177	2,954276
5	1,600415	2,788268	2,295785	1,963898	2,040455	2,467592	2,230376	2,839354	2,853728	3,192257
6	1,840941	3,271471	2,85823	2,483454	2,495718	2,972059	2,701853	3,390093	3,526762	3,77381
7	1,330325	3,19869	3,305369	2,836767	2,85195	3,4375	3,211709	3,913469	4,020375	4,367697
8	0,88533	4,172501	3,857841	3,093248	3,150533	3,427551	3,553086	4,231746	4,477404	4,716197
9	0,516407	2,103307	2,26627	2,068787	1,54488	2,513629	1,993781	2,58579	2,877943	3,072419
10	0,676574	2,915829	2,501054	2,373201	2,272834	2,793587	2,75676	3,061623	3,032997	3,332383
11	0,604096	2,219299	2,65363	2,481072	2,337677	2,551877	2,445679	2,633357	1,912926	3,08898
12	0,507293	1,637445	1,820958	1,425219	1,71445	1,949825	1,868568	2,161252	2,219787	2,322722
13	0,42752	2,015485	1,850586	1,547903	1,560629	1,928709	1,574662	1,947848	2,277699	2,470452
14	0,366814	2,297305	1,885987	1,641526	1,517686	2,672049	2,000833	2,428596	2,454854	2,352997
15	0,136311	2,283841	2,013425	1,838306	1,786971	2,231775	2,148879	2,649835	2,582857	2,801497
16	0,260047	2,299227	1,434065	1,911265	1,96826	2,294685	2,335652	2,679833	2,727882	2,93162
17	0,319142	2,338194	2,28328	1,107243	1,840701	1,945503	1,847137	2,668776	2,388203	3,082264
18	0,321763	2,107619	2,23209	2,064568	1,932535	2,321126	1,976131	2,933815	2,576167	2,520801
19	0,298604	2,325105	2,041302	1,749054	1,658192	2,165059	1,864322	2,078058	2,01968	3,333461
20	0,272505	2,267028	1,897132	2,179858	1,888086	2,352819	2,300263	2,753873	2,845151	2,802936
40	0,094992	0,468972	1,496878	1,682912	1,500654	1,821155	1,681592	1,967032	1,982229	1,887514
60	0,163207	1,451099	1,632716	1,811828	1,822395	2,277981	2,405385	2,643883	3,045751	2,41552
80	0,726079	2,241397	2,490281	2,025647	2,360834	2,280149	2,2562	2,706006	2,715331	3,035755

22 lentelė. Matricų daugybos laikas pagal matricos dydį

	100	200	300	400	500	600	700	800	900	1000
1	0,006355	0,05987	0,182185	0,447429	1,04194	2,08912	3,38861	6,40314	10,4225	16,4874
2	0,003583	0,0245	0,094682	0,255986	0,571436	1,0006	1,75734	3,28679	5,7393	8,76997
3	0,00339	0,019163	0,075686	0,269446	0,605472	0,83839	1,41722	2,50266	4,00148	6,69052
4	0,004039	0,024737	0,09087	0,238055	0,562783	0,746371	1,57014	2,3683	3,84853	5,58086
5	0,003971	0,020797	0,079356	0,227827	0,510641	0,846623	1,5193	2,25514	3,65224	5,16481
6	0,003452	0,017725	0,063741	0,180164	0,417491	0,70292	1,25418	1,88878	2,95526	4,3689
7	0,004777	0,018128	0,055118	0,157725	0,365343	0,607744	1,05508	1,63618	2,59242	3,77485
8	0,007178	0,013897	0,047225	0,144647	0,330719	0,609508	0,953709	1,51312	2,3278	3,49591
9	0,012306	0,027569	0,08039	0,216276	0,674447	0,831117	1,6995	2,47628	3,62151	5,36626
10	0,009393	0,019887	0,072843	0,188534	0,458432	0,747827	1,2292	2,09142	3,43637	4,94763
11	0,01052	0,026128	0,068655	0,180337	0,445716	0,81866	1,38555	2,43155	5,44846	5,33749
12	0,012527	0,035413	0,100049	0,313937	0,60774	1,07144	1,81348	2,9627	4,69527	7,09831
13	0,014865	0,028771	0,098447	0,289055	0,667641	1,08317	2,15196	3,28729	4,57589	6,67384
14	0,017325	0,025241	0,096599	0,272569	0,686532	0,781842	1,6936	2,63656	4,24567	7,00698
15	0,046621	0,02539	0,090485	0,243392	0,583076	0,93608	1,57692	2,41643	4,03526	5,88521
16	0,024438	0,02522	0,127041	0,234101	0,529371	0,910417	1,45082	2,38938	3,82073	5,62399
17	0,019913	0,0248	0,079791	0,404093	0,566056	1,07382	1,83452	2,39928	4,36416	5,34912
18	0,01975	0,027513	0,081621	0,216718	0,539157	0,900046	1,71477	2,18253	4,04574	6,54054
19	0,021282	0,024939	0,089249	0,255812	0,628359	0,964925	1,81761	3,08131	5,16047	4,94603
20	0,02332	0,025578	0,096032	0,205256	0,55185	0,887922	1,47314	2,32514	3,66325	5,88219
40	0,0669	0,123646	0,12171	0,265866	0,694324	1,14714	2,01512	3,25523	5,25797	8,73498
60	0,038938	0,03996	0,111584	0,246949	0,571742	0,917093	1,40876	2,42187	3,42198	6,82561
80	0,008752	0,025871	0,073158	0,220882	0,441344	0,916221	1,50191	2,36627	3,83839	5,43107

11 Priedas. OpenMP π skaičiavimo diagramų duomenys

23 lentelė. π skaičiavimo efektyvumo koeficientas

	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000

1	1	1	1	1	1	1	1	1	1	1
2	0,983613	1,022582	0,993222	0,988123	0,976388	0,959053	0,951374	0,921859	1,02147	0,991116
3	0,894128	0,943511	0,921034	0,861298	0,942474	0,949282	0,963964	0,905069	0,953381	0,942245
4	0,874196	0,900061	0,937569	0,839327	0,854756	0,896726	0,896538	0,870029	0,943125	0,905357
5	0,742907	0,791424	0,8165	0,792985	0,7757	0,764974	0,779936	0,80363	0,799213	0,808833
6	0,741286	0,79156	0,806431	0,797628	0,769983	0,761001	0,774725	0,771752	0,795913	0,76497
7	0,733933	0,77211	0,798196	0,798432	0,765145	0,759974	0,776062	0,768872	0,802184	0,768167
8	0,654769	0,765914	0,791949	0,758593	0,762714	0,742229	0,767899	0,769853	0,794464	0,723458
9	0,374639	0,398533	0,417985	0,414243	0,404136	0,404272	0,413043	0,415649	0,41583	0,284528
10	0,383932	0,395529	0,405083	0,398293	0,382396	0,379244	0,2588	0,235334	0,356411	0,371474
11	0,243703	0,423375	0,434491	0,426546	0,221881	0,403092	0,413414	0,416264	0,416654	0,413275
12	0,262455	0,290791	0,298878	0,294723	0,283048	0,232911	0,177472	0,286527	0,114951	0,359645
13	0,341527	0,388482	0,083962	0,359009	0,296283	0,189318	0,389471	0,247272	0,393298	0,311592
14	0,258224	0,263362	0,299182	0,260721	0,283594	0,282882	0,286438	0,286467	0,190034	0,198422
15	0,241786	0,188735	0,283544	0,264252	0,270281	0,26842	0,270253	0,267337	0,275257	0,271497
16	0,237277	0,276886	0,282786	0,279217	0,265327	0,267496	0,271908	0,27106	0,279888	0,270253
17	0,192547	0,222238	0,070238	0,076508	0,152592	0,215255	0,198544	0,26878	0,28013	0,236365
18	0,224496	0,276952	0,280629	0,278822	0,269942	0,267566	0,271429	0,179002	0,280227	0,270748
19	0,188858	0,116236	0,206698	0,229835	0,130156	0,097801	0,267024	0,262718	0,278123	0,269081
20	0,204863	0,263166	0,27065	0,192985	0,257745	0,255003	0,258905	0,257901	0,266257	0,259147
40	0,072148	0,122578	0,134184	0,129544	0,129056	0,12495	0,129318	0,129274	0,132646	0,132161
60	0,050526	0,090174	0,091164	0,090651	0,087513	0,08716	0,08817	0,088016	0,091324	0,088575
80	0,037511	0,06912	0,065066	0,071151	0,027529	0,051647	0,06919	0,069644	0,070717	0,070024
100	0,016175	0,050449	0,056531	0,055869	0,054401	0,054214	0,054999	0,054785	0,057004	0,055276
120	0,047308	0,04688	0,047756	0,047032	0,045725	0,044848	0,046041	0,045803	0,04717	0,045365
140	0,040138	0,039415	0,040982	0,035986	0,039274	0,038748	0,035817	0,035888	0,040583	0,039432
160	0,035447	0,034967	0,035317	0,030793	0,033386	0,033468	0,034565	0,032636	0,035569	0,034543

24 lentelė. π skaičiavimo spartinimo koeficientas

	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
1	1	1	1	1	1	1	1	1	1	1
2	1,967226	2,045164	1,986444	1,976246	1,952776	1,918105	1,902747	1,843719	2,042941	1,982231
3	2,682385	2,830534	2,763103	2,583895	2,827421	2,847847	2,891892	2,715207	2,860144	2,826734
4	3,496782	3,600243	3,750278	3,357306	3,419026	3,586904	3,586153	3,480114	3,772499	3,621427
5	3,714535	3,957119	4,082752	3,964925	3,878499	3,824871	3,899679	4,018151	3,996063	4,044166
6	4,447716	4,749362	4,838585	4,785765	4,619896	4,566007	4,648352	4,63051	4,775479	4,589819
7	5,137533	5,404768	5,587372	5,589027	5,356014	5,319815	5,432432	5,382104	5,615289	5,377167
8	5,238156	6,127315	6,335591	6,068746	6,101714	5,937836	6,143189	6,15882	6,355715	5,787667
9	3,371753	3,586795	3,761866	3,728187	3,637221	3,638447	3,717383	3,740841	3,742471	2,560756
10	3,839321	3,955295	4,050831	3,98293	3,82396	3,792439	2,587997	2,353345	3,564114	3,714737
11	2,680732	4,657124	4,779406	4,692005	2,440689	4,43401	4,547555	4,578906	4,583198	4,546021
12	3,149455	3,489492	3,58653	3,536674	3,396582	2,794927	2,129662	3,438326	1,37941	4,315744
13	4,43985	5,05027	1,091507	4,66712	3,851676	2,461134	5,063118	3,214538	5,11288	4,0507
14	3,61514	3,687065	4,188542	3,65009	3,970322	3,960345	4,010131	4,010538	2,660471	2,777903
15	3,626796	2,831031	4,25316	3,96378	4,054221	4,026307	4,053791	4,010059	4,12886	4,072454
16	3,796434	4,430175	4,524577	4,467474	4,245235	4,279943	4,350525	4,336961	4,478212	4,324045
17	3,273306	3,778049	1,19405	1,300633	2,594068	3,659337	3,375244	4,569266	4,762214	4,018211
18	4,040925	4,985142	5,051317	5,018798	4,858959	4,816197	4,885721	3,222044	5,044084	4,87346
19	3,588306	2,208482	3,92727	4,366861	2,472973	1,858216	5,073462	4,991638	5,284345	5,112546
20	4,097254	5,263317	5,412993	3,859708	5,154892	5,100059	5,178107	5,158024	5,325142	5,18293
40	2,885931	4,903116	5,367357	5,18177	5,162255	4,998018	5,172726	5,170943	5,305825	5,286456
60	3,031531	5,410434	5,469848	5,439051	5,250799	5,22961	5,290203	5,280941	5,47942	5,314474
80	3,00086	5,529626	5,205306	5,692087	2,202288	4,131782	5,535225	5,571537	5,657333	5,601887
100	1,617475	5,04491	5,65314	5,586871	5,440057	5,42144	5,499856	5,478461	5,700401	5,527632
120	5,676987	5,625589	5,730766	5,643838	5,487004	5,381751	5,524864	5,496331	5,660388	5,44375

140	5,619296	5,518033	5,737421	5,03808	5,498328	5,424663	5,014449	5,024307	5,68167	5,520431
160	5,671555	5,594765	5,650711	4,926889	5,341796	5,354895	5,53045	5,2217	5,691106	5,526803

25 lentelė. π skaičiavimo laikas

	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
1	0,033487	0,066064	0,101289	0,133229	0,161178	0,191649	0,226973	0,258482	0,30087	0,324184
2	0,017022	0,032303	0,05099	0,067415	0,082538	0,099916	0,119287	0,140196	0,147273	0,163545
3	0,012484	0,02334	0,036658	0,051561	0,057005	0,067296	0,078486	0,095198	0,105194	0,114685
4	0,009576	0,01835	0,027008	0,039683	0,047142	0,05343	0,063292	0,074274	0,079754	0,089518
5	0,009015	0,016695	0,024809	0,033602	0,041557	0,050106	0,058203	0,064329	0,075292	0,080161
6	0,007529	0,01391	0,020934	0,027839	0,034888	0,041973	0,048829	0,055822	0,063003	0,070631
7	0,006518	0,012223	0,018128	0,023838	0,030093	0,036026	0,041781	0,048026	0,053581	0,060289
8	0,006393	0,010782	0,015987	0,021953	0,026415	0,032276	0,036947	0,041969	0,047339	0,056013
9	0,009932	0,018419	0,026925	0,035736	0,044314	0,052673	0,061057	0,069097	0,080393	0,126597
10	0,008722	0,016703	0,025005	0,03345	0,04215	0,050535	0,087702	0,109836	0,084417	0,08727
11	0,012492	0,014186	0,021193	0,028395	0,066038	0,043223	0,049911	0,056451	0,065646	0,071312
12	0,010633	0,018932	0,028242	0,037671	0,047453	0,06857	0,106577	0,075177	0,218115	0,075117
13	0,007542	0,013081	0,092797	0,028546	0,041846	0,07787	0,044829	0,08041	0,058846	0,080032
14	0,009263	0,017918	0,024182	0,0365	0,040596	0,048392	0,0566	0,064451	0,113089	0,116701
15	0,009233	0,023336	0,023815	0,033612	0,039756	0,047599	0,05599	0,064458	0,07287	0,079604
16	0,008821	0,014912	0,022386	0,029822	0,037967	0,044778	0,052171	0,0596	0,067185	0,074972
17	0,01023	0,017486	0,084828	0,102434	0,062133	0,052373	0,067246	0,05657	0,063179	0,080679
18	0,008287	0,013252	0,020052	0,026546	0,033171	0,039793	0,046456	0,080223	0,059648	0,06652
19	0,009332	0,029914	0,025791	0,030509	0,065176	0,103136	0,044737	0,051783	0,056936	0,06341
20	0,008173	0,012552	0,018712	0,034518	0,031267	0,037578	0,043833	0,050113	0,0565	0,062548
40	0,011604	0,013474	0,018871	0,025711	0,031222	0,038345	0,043879	0,049987	0,056706	0,061324
60	0,011046	0,012211	0,018518	0,024495	0,030696	0,036647	0,042904	0,048946	0,054909	0,061

80	0,011159	0,011947	0,019459	0,023406	0,073187	0,046384	0,041005	0,046393	0,053182	0,057871
100	0,020703	0,013095	0,017917	0,023847	0,029628	0,03535	0,041269	0,047182	0,052781	0,058648
120	0,005899	0,011744	0,017675	0,023606	0,029375	0,035611	0,041082	0,047028	0,053154	0,059552
140	0,005959	0,011972	0,017654	0,026444	0,029314	0,035329	0,045264	0,051446	0,052955	0,058724
160	0,005904	0,011808	0,017925	0,027041	0,030173	0,03579	0,041041	0,049502	0,052867	0,058657

12 Priedas. OpenMP rikiavimo diagramų duomenys

26 lentelė. Bitonic efektyvumo koeficientas

	32768	65536	131072	262144	524288	1048576	2097152	4194304
1	1	1	1	1	1	1	1	1
2	1,053491844	0,982479783	0,92695441	0,96623437	0,9481967	0,92666959	0,846726886	0,92105697
4	0,856022643	0,808489037	0,73690354	0,83588656	0,8081603	0,78772474	0,826810855	0,82203415

27 lentelė. Bitonic rikiavimo spartinimo koeficientas

	32768	65536	131072	262144	524288	1048576	2097152	4194304
1	1	1	1	1	1	1	1	1
2	2,106983688	1,964959566	1,85390883	1,93246873	1,8963935	1,85333918	1,693453772	1,84211394
4	3,424090572	3,233956149	2,94761416	3,34354622	3,2326413	3,15089898	3,307243419	3,28813659

28 lentelė. Bitonic rikiavimo laikas

	32768	65536	131072	262144	524288	1048576	2097152	4194304
1	0,0774858	0,16564	0,347226	0,756805	1,6096	3,46643	7,3903	15,6513
2	0,0367757	0,0842969	0,187294	0,391626	0,848769	1,87037	4,36404	8,49638
4	0,0226296	0,051219	0,117799	0,226348	0,497921	1,10014	2,23458	4,75993

13 Priedas. CUDA maksimalios reikšmės paieškos diagramų duomenys

29 lentelė. Maksimalios reikšmės efektyvumo koeficientas

	1E+08	1,5E+08	2E+08	2,5E+08	3E+08
1, 1	1	1	1	1	1
1, 32	0,457991	0,457143	0,457178	0,4583	0,458015
1, 64	0,45709	0,457068	0,457078	0,457092	0,457082
1, 128	0,455877	0,455948	0,455945	0,455952	0,455988
1, 256	0,453919	0,453978	0,453955	0,45399	0,453985
1, 512	0,449084	0,449153	0,449128	0,44916	0,449172
1, 1024	0,395475	0,395416	0,395666	0,395464	0,395059
32, 1	0,225023	0,224742	0,224972	0,224754	0,225057
32, 32	0,226268	0,226287	0,226279	0,226307	0,226286
32, 64	0,222347	0,222471	0,22239	0,222408	0,222398
32, 128	0,19474	0,194894	0,194828	0,194868	0,195058

32, 256	0,113566	0,113919	0,113857	0,114107	0,113987
32, 512	0,057103	0,057176	0,057231	0,05721	0,057051
32, 1024	0,024357	0,024422	0,024472	0,024514	0,024502
64, 1	0,112422	0,112506	0,112437	0,112424	0,112475
64, 32	0,113071	0,112861	0,113079	0,112941	0,113062
64, 64	0,111053	0,111066	0,111087	0,111102	0,111106
64, 128	0,097321	0,097262	0,097434	0,097419	0,097418
64, 256	0,056839	0,056957	0,057019	0,057026	0,057056
64, 512	0,029575	0,029631	0,029694	0,029722	0,029757
64, 1024	0,012045	0,01213	0,012171	0,012197	0,012161
128, 1	0,056225	0,056231	0,05625	0,056228	0,056273
128, 32	0,056459	0,056471	0,05646	0,056411	0,056459
128, 64	0,05538	0,055412	0,055429	0,055437	0,055456
128, 128	0,048778	0,04881	0,048875	0,048859	0,048755
128, 256	0,029423	0,029542	0,029575	0,029583	0,029628
128, 512	0,01481	0,014895	0,014939	0,014974	0,014998
128, 1024	0,005902	0,005979	0,006015	0,006036	0,005819
256, 1	0,028105	0,028094	0,02811	0,028098	0,028098
256, 32	0,028147	0,02819	0,028202	0,028154	0,028048
256, 64	0,027587	0,027622	0,027591	0,027605	0,027599
256, 128	0,024406	0,024464	0,024499	0,024501	0,024474
256, 256	0,014756	0,014849	0,014896	0,014926	0,014985
256, 512	0,007292	0,007367	0,007415	0,007433	0,007379
256, 1024	0,002842	0,002913	0,002947	0,00297	0,002591
512, 1	0,01406	0,014059	0,01405	0,014038	0,014048
512, 32	0,014046	0,014046	0,01402	0,014018	0,013809
512, 64	0,013606	0,01365	0,013638	0,013659	0,013386
512, 128	0,012289	0,012365	0,012389	0,012422	0,012319
512, 256	0,007264	0,007341	0,007377	0,007404	0,007368
512, 512	0,003529	0,003602	0,003644	0,003663	0,003569
512, 1024	0,001331	0,001393	0,001424	0,001445	0,001043
1024, 1	0,007029	0,007029	0,007021	0,007026	0,007022
1024, 32	0,006916	0,006923	0,006929	0,006943	0,006727
1024, 64	0,006772	0,006798	0,006808	0,006815	0,006458
1024, 128	0,006069	0,006133	0,006172	0,006192	0,005888
1024, 256	0,003523	0,003606	0,003633	0,00367	0,003573
1024, 512	0,001667	0,001733	0,001769	0,001793	0,001567
1024, 1024	0,00059	0,00064	0,000668	0,000686	0,000697

30 lentelė. Maksimalios reikšmės spartinimo koeficientas

Giju konfig.			1E+08	1,5E+08	2E+08	2,5E+08	300000000
1	1	1, 1	1	1	1	1	1
1	32	1, 32	14,6557	14,62857	14,62968	14,66559	14,65649179
1	64	1, 64	29,25374	29,25233	29,25297	29,25388	29,25322361
1	128	1, 128	58,35229	58,36129	58,36098	58,36192	58,36650333
1	256	1, 256	116,2033	116,2184	116,2124	116,2215	116,2202729
1	512	1, 512	229,9311	229,9664	229,9537	229,9697	229,9758394

1	1024	1, 1024	404,9661	404,9058	405,1623	404,9547	404,5401587
32	1	32, 1	7,200729	7,191753	7,19909	7,192143	7,201810628
32	32	32, 32	231,6985	231,7183	231,7097	231,7379	231,7166583
32	64	32, 64	455,3657	455,6209	455,4554	455,4916	455,4720919
32	128	32, 128	797,6564	798,2848	798,0172	798,1801	798,9580318
32	256	32, 256	930,3336	933,2223	932,7173	934,7668	933,7821153
32	512	32, 512	935,5789	936,7646	937,6658	937,3217	934,7307113
32	1024	32, 1024	798,1147	800,274	801,9066	803,2667	802,8911528
64	1	64, 1	7,195013	7,200401	7,195994	7,195124	7,198383206
64	32	64, 32	231,5702	231,1401	231,5851	231,3034	231,5504713
64	64	64, 64	454,8711	454,9264	455,0106	455,0748	455,0891293
64	128	64, 128	797,2503	796,7687	798,1764	798,0555	798,0514177
64	256	64, 256	931,2446	933,1782	934,2064	934,3179	934,8019337
64	512	64, 512	969,1037	970,943	973,0044	973,9413	975,087648
64	1024	64, 1024	789,3949	794,9571	797,6687	799,3658	797,0087629
128	1	128, 1	7,196786	7,197538	7,200006	7,197206	7,202938164
128	32	128, 32	231,2573	231,3066	231,2598	231,06	231,2580374
128	64	128, 64	453,669	453,9351	454,0731	454,1427	454,2914861
128	128	128, 128	799,1753	799,7106	800,7674	800,4994	798,7961918
128	256	128, 256	964,1255	968,0437	969,1204	969,3897	970,8522742
128	512	128, 512	970,5809	976,1361	979,0245	981,3287	982,9346675
128	1024	128, 1024	773,5928	783,6473	788,3519	791,098	762,725797
256	1	256, 1	7,194788	7,192147	7,196205	7,193043	7,193177895
256	32	256, 32	230,5764	230,9293	231,0341	230,6401	229,7652738
256	64	256, 64	451,982	452,5509	452,0542	452,2765	452,1743404
256	128	256, 128	799,7432	801,6347	802,7952	802,8602	801,9755956
256	256	256, 256	967,0209	973,1182	976,2369	978,2218	982,0516925
256	512	256, 512	955,7162	965,5532	971,9637	974,2909	967,2208693
256	1024	256, 1024	744,9908	763,4953	772,5951	778,633	679,1905026
512	1	512, 1	7,198869	7,198167	7,193673	7,187255	7,192568756
512	32	512, 32	230,1237	230,1246	229,7097	229,6706	226,2437505
512	64	512, 64	445,8336	447,2823	446,8974	447,5904	438,618235
512	128	512, 128	805,3606	810,3798	811,9093	814,1194	807,3174477
512	256	512, 256	952,0524	962,1742	966,927	970,5	965,7657658
512	512	512, 512	925,0189	944,3678	955,3169	960,2684	935,4671999
512	1024	512, 1024	697,9002	730,3977	746,5051	757,736	546,7383662
1024	1	1024, 1	7,197659	7,19782	7,189739	7,194786	7,19092928
1024	32	1024, 32	226,6183	226,8659	227,0415	227,4918	220,4286156
1024	64	1024, 64	443,7892	445,5101	446,1552	446,641	423,2054702
1024	128	1024, 128	795,4308	803,8383	808,9675	811,5416	771,7890469
1024	256	1024, 256	923,6122	945,2377	952,4787	962,1562	936,5701393
1024	512	1024, 512	873,8657	908,8298	927,4584	939,9174	821,5659576
1024	1024	1024, 1024	618,3104	670,8365	700,1341	719,0906	731,1882073

31 lentelė. Maksimalios reikšmės paieška pagal laiką (laikas < 2s)

	1E+08	1,5E+08	2E+08	2,5E+08	3E+08
1, 1	36,8053	55,2085	73,6107	92,0134	110,416

1, 128	0,630743	0,945978	1,2613	1,5766	1,89177
1, 256	0,316732	0,475041	0,633415	0,791707	0,950058
1, 512	0,160071	0,240072	0,320111	0,400111	0,48012
1, 1024	0,090885	0,136349	0,181682	0,227219	0,272942
32, 32	0,15885	0,238257	0,317685	0,397058	0,476513
32, 64	0,080826	0,121172	0,16162	0,202009	0,242421
32, 128	0,046142	0,069159	0,092242	0,115279	0,1382
32, 256	0,039561	0,059159	0,078921	0,098435	0,118246
32, 512	0,03934	0,058935	0,078504	0,098166	0,118126
32, 1024	0,046115	0,068987	0,091795	0,114549	0,137523
64, 32	0,158938	0,238853	0,317856	0,397804	0,476855
64, 64	0,080914	0,121357	0,161778	0,202194	0,242625
64, 128	0,046165	0,069291	0,092224	0,115297	0,138357
64, 256	0,039523	0,059162	0,078795	0,098482	0,118117
64, 512	0,037979	0,056861	0,075653	0,094475	0,113237
64, 1024	0,046625	0,069448	0,092282	0,115108	0,138538
128, 32	0,159153	0,238681	0,318303	0,398223	0,477458
128, 64	0,081128	0,121622	0,162112	0,202609	0,243051
128, 128	0,046054	0,069036	0,091925	0,114945	0,138228
128, 256	0,038175	0,057031	0,075956	0,094919	0,113731
128, 512	0,037921	0,056558	0,075188	0,093764	0,112333
128, 1024	0,047577	0,070451	0,093373	0,116311	0,144765
256, 32	0,159623	0,239071	0,318614	0,398948	0,48056
256, 64	0,081431	0,121994	0,162836	0,203445	0,244189
256, 128	0,046021	0,06887	0,091693	0,114607	0,13768
256, 256	0,038061	0,056734	0,075403	0,094062	0,112434
256, 512	0,038511	0,057178	0,075734	0,094441	0,114158
256, 1024	0,049404	0,07231	0,095277	0,118173	0,16257
512, 32	0,159937	0,239907	0,320451	0,400632	0,48804
512, 64	0,082554	0,123431	0,164715	0,205575	0,251736
512, 128	0,0457	0,068127	0,090664	0,113022	0,136769
512, 256	0,038659	0,057379	0,076129	0,09481	0,11433
512, 512	0,039789	0,058461	0,077054	0,095821	0,118033
512, 1024	0,052737	0,075587	0,098607	0,121432	0,201954
1024, 32	0,162411	0,243353	0,324217	0,404469	0,500915
1024, 64	0,082934	0,123922	0,164989	0,206012	0,260904
1024, 128	0,046271	0,068681	0,090993	0,113381	0,143065
1024, 256	0,039849	0,058407	0,077283	0,095633	0,117894
1024, 512	0,042118	0,060747	0,079368	0,097895	0,134397
1024, 1024	0,059526	0,082298	0,105138	0,127958	0,151009

32 lentelė. Maksimalios reikšmės paieška pagal laiką

	1E+08	1,5E+08	2E+08	2,5E+08	3E+08
1, 1	36,8053	55,2085	73,6107	92,0134	110,416
1, 32	2,51133	3,77402	5,0316	6,2741	7,53359
1, 64	1,25814	1,88732	2,51635	3,14534	3,77449
1, 128	0,630743	0,945978	1,2613	1,5766	1,89177

1, 256	0,316732	0,475041	0,633415	0,791707	0,950058
1, 512	0,160071	0,240072	0,320111	0,400111	0,48012
1, 1024	0,090885	0,136349	0,181682	0,227219	0,272942
32, 1	5,11133	7,67664	10,225	12,7936	15,3317
32, 32	0,15885	0,238257	0,317685	0,397058	0,476513
32, 64	0,080826	0,121172	0,16162	0,202009	0,242421
32, 128	0,046142	0,069159	0,092242	0,115279	0,1382
32, 256	0,039561	0,059159	0,078921	0,098435	0,118246
32, 512	0,03934	0,058935	0,078504	0,098166	0,118126
32, 1024	0,046115	0,068987	0,091795	0,114549	0,137523
64, 1	5,11539	7,66742	10,2294	12,7883	15,339
64, 32	0,158938	0,238853	0,317856	0,397804	0,476855
64, 64	0,080914	0,121357	0,161778	0,202194	0,242625
64, 128	0,046165	0,069291	0,092224	0,115297	0,138357
64, 256	0,039523	0,059162	0,078795	0,098482	0,118117
64, 512	0,037979	0,056861	0,075653	0,094475	0,113237
64, 1024	0,046625	0,069448	0,092282	0,115108	0,138538
128, 1	5,11413	7,67047	10,2237	12,7846	15,3293
128, 32	0,159153	0,238681	0,318303	0,398223	0,477458
128, 64	0,081128	0,121622	0,162112	0,202609	0,243051
128, 128	0,046054	0,069036	0,091925	0,114945	0,138228
128, 256	0,038175	0,057031	0,075956	0,094919	0,113731
128, 512	0,037921	0,056558	0,075188	0,093764	0,112333
128, 1024	0,047577	0,070451	0,093373	0,116311	0,144765
256, 1	5,11555	7,67622	10,2291	12,792	15,3501
256, 32	0,159623	0,239071	0,318614	0,398948	0,48056
256, 64	0,081431	0,121994	0,162836	0,203445	0,244189
256, 128	0,046021	0,06887	0,091693	0,114607	0,13768
256, 256	0,038061	0,056734	0,075403	0,094062	0,112434
256, 512	0,038511	0,057178	0,075734	0,094441	0,114158
256, 1024	0,049404	0,07231	0,095277	0,118173	0,16257
512, 1	5,11265	7,6698	10,2327	12,8023	15,3514
512, 32	0,159937	0,239907	0,320451	0,400632	0,48804
512, 64	0,082554	0,123431	0,164715	0,205575	0,251736
512, 128	0,0457	0,068127	0,090664	0,113022	0,136769
512, 256	0,038659	0,057379	0,076129	0,09481	0,11433
512, 512	0,039789	0,058461	0,077054	0,095821	0,118033
512, 1024	0,052737	0,075587	0,098607	0,121432	0,201954
1024, 1	5,11351	7,67017	10,2383	12,7889	15,3549
1024, 32	0,162411	0,243353	0,324217	0,404469	0,500915
1024, 64	0,082934	0,123922	0,164989	0,206012	0,260904
1024, 128	0,046271	0,068681	0,090993	0,113381	0,143065
1024, 256	0,039849	0,058407	0,077283	0,095633	0,117894
1024, 512	0,042118	0,060747	0,079368	0,097895	0,134397
1024, 1024	0,059526	0,082298	0,105138	0,127958	0,151009

14 Priedas. CUDA matricų daugybos diagramų duomenys

33 lentelė. Matricų daugybos efektyvumo koeficientas

	100	200	300	400	500
(1,1), (1,1)	1	1	1	1	1
(100,100), (1,1)	0,01	0,01	0,01	0,01	0,01
(100,50), (1,2)	0,0001	0,0001	0,0001	0,0001	0,0001
(100,25), (1,4)	0,001	0,001	0,001	0,001	0,001
(100,13), (1,8)	9,62E-05	9,62E-05	9,62E-05	9,62E-05	9,62E-05
(100,7), (1,16)	8,93E-05	8,93E-05	8,93E-05	8,93E-05	8,93E-05
(100,4), (1,32)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(100,2), (1,64)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(100,1), (1,128)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(100,1), (1,256)	3,91E-05	3,91E-05	3,91E-05	3,91E-05	3,91E-05
(100,1), (1,512)	1,95E-05	1,95E-05	1,95E-05	1,95E-05	1,95E-05
(100,1), (1,1024)	9,77E-06	9,77E-06	9,77E-06	9,77E-06	9,77E-06
(50,100), (2,1)	0,0001	0,0001	0,0001	0,0001	0,0001
(50,50), (2,2)	0,0001	0,0001	0,0001	0,0001	0,0001
(50,25), (2,4)	0,0001	0,0001	0,0001	0,0001	0,0001
(50,13), (2,8)	9,62E-05	9,62E-05	9,62E-05	9,62E-05	9,62E-05
(50,7), (2,16)	8,93E-05	8,93E-05	8,93E-05	8,93E-05	8,93E-05
(50,4), (2,32)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(50,2), (2,64)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(50,1), (2,128)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(50,1), (2,256)	3,91E-05	3,91E-05	3,91E-05	3,91E-05	3,91E-05
(50,1), (2,512)	1,95E-05	1,95E-05	1,95E-05	1,95E-05	1,95E-05
(25,100), (4,1)	0,001	0,001	0,001	0,001	0,001
(25,50), (4,2)	0,0001	0,0001	0,0001	0,0001	0,0001
(25,25), (4,4)	0,0001	0,0001	0,0001	0,0001	0,0001
(25,13), (4,8)	9,62E-05	9,62E-05	9,62E-05	9,62E-05	9,62E-05
(25,7), (4,16)	8,93E-05	8,93E-05	8,93E-05	8,93E-05	8,93E-05
(25,4), (4,32)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(25,2), (4,64)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(25,1), (4,128)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(25,1), (4,256)	3,91E-05	3,91E-05	3,91E-05	3,91E-05	3,91E-05
(13,100), (8,1)	9,62E-05	9,62E-05	9,62E-05	9,62E-05	9,62E-05
(13,50), (8,2)	9,62E-05	9,62E-05	9,62E-05	9,62E-05	9,62E-05
(13,25), (8,4)	9,62E-05	9,62E-05	9,62E-05	9,62E-05	9,62E-05
(13,13), (8,8)	9,25E-05	9,25E-05	9,25E-05	9,25E-05	9,25E-05
(13,7), (8,16)	8,59E-05	8,59E-05	8,59E-05	8,59E-05	8,59E-05
(13,4), (8,32)	7,51E-05	7,51E-05	7,51E-05	7,51E-05	7,51E-05
(13,2), (8,64)	7,51E-05	7,51E-05	7,51E-05	7,51E-05	7,51E-05
(13,1), (8,128)	7,51E-05	7,51E-05	7,51E-05	7,51E-05	7,51E-05
(7,100), (16,1)	8,93E-05	8,93E-05	8,93E-05	8,93E-05	8,93E-05
(7,50), (16,2)	8,93E-05	8,93E-05	8,93E-05	8,93E-05	8,93E-05
(7,25), (16,4)	8,93E-05	8,93E-05	8,93E-05	8,93E-05	8,93E-05
(7,13), (16,8)	8,59E-05	8,59E-05	8,59E-05	8,59E-05	8,59E-05
(7,7), (16,16)	7,97E-05	7,97E-05	7,97E-05	7,97E-05	7,97E-05

(7,4), (16,32)	6,98E-05	6,98E-05	6,98E-05	6,98E-05	6,98E-05
(7,2), (16,64)	6,98E-05	6,98E-05	6,98E-05	6,98E-05	6,98E-05
(4,100), (32,1)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(4,50), (32,2)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(4,25), (32,4)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(4,13), (32,8)	7,51E-05	7,51E-05	7,51E-05	7,51E-05	7,51E-05
(4,7), (32,16)	6,98E-05	6,98E-05	6,98E-05	6,98E-05	6,98E-05
(4,4), (32,32)	6,1E-05	6,1E-05	6,1E-05	6,1E-05	6,1E-05
(2,100), (64,1)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(2,50), (64,2)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(2,25), (64,4)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(2,13), (64,8)	7,51E-05	7,51E-05	7,51E-05	7,51E-05	7,51E-05
(2,7), (64,16)	6,98E-05	6,98E-05	6,98E-05	6,98E-05	6,98E-05
(1,100), (128,1)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(1,50), (128,2)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(1,25), (128,4)	7,81E-05	7,81E-05	7,81E-05	7,81E-05	7,81E-05
(1,13), (128,8)	7,51E-05	7,51E-05	7,51E-05	7,51E-05	7,51E-05
(1,100), (256,1)	3,91E-05	3,91E-05	3,91E-05	3,91E-05	3,91E-05
(1,50), (256,2)	3,91E-05	3,91E-05	3,91E-05	3,91E-05	3,91E-05
(1,25), (256,4)	3,91E-05	3,91E-05	3,91E-05	3,91E-05	3,91E-05
(1,100), (512,1)	1,95E-05	1,95E-05	1,95E-05	1,95E-05	1,95E-05
(1,50), (512,2)	1,95E-05	1,95E-05	1,95E-05	1,95E-05	1,95E-05
(1,100), (1024,1)	9,77E-06	9,77E-06	9,77E-06	9,77E-06	9,77E-06

34 lentelė. Matricų daugybos spartinimo koeficientas

	100	200	300	400	500
(1,1), (1,1)	1	1	1	1	1
(100,100), (1,1)	14,54171	14,97145	15,17599	14,5901	15,07091
(100,50), (1,2)	27,57217	28,56589	29,22894	28,12794	28,9528
(100,25), (1,4)	49,82229	52,5742	54,5214	52,60169	53,55933
(100,13), (1,8)	80,80702	90,92604	95,22829	92,62488	91,36261
(100,7), (1,16)	119,4437	142,5946	153,0715	147,5805	136,6067
(100,4), (1,32)	161,8028	200,7241	209,019	194,3227	193,9853
(100,2), (1,64)	277,2919	315,3681	365,3395	348,9927	326,83
(100,1), (1,128)	364,3694	230,5466	173,0386	180,0134	171,7364
(100,1), (1,256)	379,3312	245,4765	168,4132	136,4801	131,4028
(100,1), (1,512)	213,2089	221,0675	156,3477	141,5114	138,0306
(100,1), (1,1024)	73,80724	170,2702	221,6165	227,6081	227,4253
(50,100), (2,1)	27,91359	29,56301	30,32239	27,16472	29,71587
(50,50), (2,2)	52,87869	56,55407	58,37862	52,41329	56,73877
(50,25), (2,4)	95,62685	104,4876	108,9049	98,22407	103,5703
(50,13), (2,8)	157,2315	181,6192	190,0588	174,2243	172,9897
(50,7), (2,16)	236,1546	283,0364	301,7107	282,6136	263,4253
(50,4), (2,32)	408,1414	508,0592	512,9499	494,3282	505,4095
(50,2), (2,64)	532,1856	498,3496	473,2075	444,5274	446,1728
(50,1), (2,128)	523,0831	453,3602	410,4229	415,5921	376,3005
(50,1), (2,256)	509,9316	435,9238	464,4433	445,0428	350,591

(50,1), (2,512)	209,6079	408,9811	459,2688	453,2506	446,4447
(25,100), (4,1)	53,14536	58,18352	60,21413	53,90214	53,72452
(25,50), (4,2)	100,4126	111,6866	115,6716	104,2781	102,2834
(25,25), (4,4)	181,3303	206,6127	214,9215	194,5991	186,3198
(25,13), (4,8)	300,517	359,3946	370,7428	341,6367	316,9346
(25,7), (4,16)	535,9569	669,7883	671,0497	618,7505	575,2831
(25,4), (4,32)	767,2735	910,4204	986,5149	958,892	948,131
(25,2), (4,64)	800,7705	879,509	890,3926	866,37	854,8751
(25,1), (4,128)	752,4714	869,1947	893,6043	890,8618	881,3039
(25,1), (4,256)	468,7685	855,4129	645,9449	813,6768	920,5156
(13,100), (8,1)	97,70654	115,506	109,3226	100,5057	92,53415
(13,50), (8,2)	185,3675	221,0615	209,1295	193,6058	178,6926
(13,25), (8,4)	334,9615	407,5755	382,7378	358,8746	337,5161
(13,13), (8,8)	608,7808	769,5928	704,3779	674,5088	648,5261
(13,7), (8,16)	860,9249	1149,25	1218,04	1179,777	1120,102
(13,4), (8,32)	907,9051	1205,59	1274,92	1284,148	1271,064
(13,2), (8,64)	893,5064	1142,467	1212,652	1234,121	1241,736
(13,1), (8,128)	741,2317	902,121	868,8081	867,295	971,6064
(7,100), (16,1)	177,0336	207,9863	183,6356	185,0222	179,4334
(7,50), (16,2)	338,4807	394,0954	354,1944	358,7097	347,2263
(7,25), (16,4)	644,2222	738,9779	687,6551	702,1115	685,6
(7,13), (16,8)	855,9578	1156,479	1206,44	1194,178	1182,949
(7,7), (16,16)	856,8423	1185,192	1292,218	1330,386	1290,194
(7,4), (16,32)	839,7019	1163,03	1264,675	1292,951	1263,278
(7,2), (16,64)	737,2833	829,3738	937,6741	930,8383	964,852
(4,100), (32,1)	307,4169	325,7937	339,2522	352,1866	354,8115
(4,50), (32,2)	558,5175	627,945	666,9082	693,6133	696,3371
(4,25), (32,4)	763,4678	1066,234	1152,035	1199,927	1220,611
(4,13), (32,8)	775,4174	1116,384	1225,314	1273,645	1296,447
(4,7), (32,16)	766,6821	1092,046	1207,994	1248,865	1263,532
(4,4), (32,32)	629,4196	794,0024	902,8841	926,6869	960,0422
(2,100), (64,1)	555,947	536,9733	665,7907	646,8647	695,6861
(2,50), (64,2)	758,8075	959,0534	1147,371	1113,195	1203,104
(2,25), (64,4)	770,7455	984,8039	1229,157	1178,757	1293,459
(2,13), (64,8)	744,6398	967,0168	1192,283	1150,722	1252,319
(2,7), (64,16)	678,775	754,843	901,7562	873,1939	941,5922
(1,100), (128,1)	761,3348	976,7331	1049,267	1075,362	1240,58
(1,50), (128,2)	764,6756	987,1851	1230,589	1185,757	1306,64
(1,25), (128,4)	760,9523	984,2409	1193,858	1164,382	1295,315
(1,13), (128,8)	715,8474	858,3416	914,1959	915,4598	1058,137
(1,100), (256,1)	761,9662	980,5786	1196,706	1177,495	1307,764
(1,50), (256,2)	706,9862	989,35	1104,855	1175,947	1307,014
(1,25), (256,4)	537,127	911,819	778,593	971,2287	1133,082
(1,100), (512,1)	429,3091	752,3541	1128,06	1128,453	1305,986
(1,50), (512,2)	301,1686	527,4303	810,5097	1010,085	1217,761
(1,100), (1024,1)	153,3791	270,1771	421,5699	563,5984	692,8413

35 lentelė. Matricų daugybos laikas pagal matricos dydį

	100	200	300	400	500
(1,1), (1,1)	0,60278	5,71592	20,0666	47,604	93,0181
(100,100), (1,1)	0,0414518	0,381788	1,32226	3,26276	6,17203
(100,50), (1,2)	0,0218619	0,200096	0,686532	1,69241	3,21275
(100,25), (1,4)	0,0120986	0,108721	0,36805	0,90499	1,73673
(100,13), (1,8)	0,0074595	0,0628634	0,210721	0,513944	1,01812
(100,7), (1,16)	0,00504656	0,0400851	0,131093	0,322563	0,680919
(100,4), (1,32)	0,0037254	0,0284765	0,0960037	0,244974	0,479511
(100,2), (1,64)	0,00217381	0,0181246	0,0549259	0,136404	0,284607
(100,1), (1,128)	0,00165431	0,0247929	0,115966	0,264447	0,541633
(100,1), (1,256)	0,00158906	0,023285	0,119151	0,348798	0,707885
(100,1), (1,512)	0,00282718	0,025856	0,128346	0,336397	0,673895
(100,1), (1,1024)	0,00816695	0,0335697	0,0905465	0,209149	0,409005
(50,100), (2,1)	0,0215945	0,193347	0,661775	1,75242	3,13025
(50,50), (2,2)	0,0113993	0,10107	0,343732	0,908243	1,63941
(50,25), (2,4)	0,00630346	0,0547043	0,184258	0,484647	0,898116
(50,13), (2,8)	0,00383371	0,031472	0,105581	0,273234	0,537709
(50,7), (2,16)	0,00255248	0,020195	0,0665094	0,168442	0,35311
(50,4), (2,32)	0,00147689	0,0112505	0,03912	0,0963004	0,184045
(50,2), (2,64)	0,00113265	0,0114697	0,0424055	0,107089	0,20848
(50,1), (2,128)	0,00115236	0,0126079	0,0488925	0,114545	0,247191
(50,1), (2,256)	0,00118208	0,0131122	0,0432057	0,106965	0,265318
(50,1), (2,512)	0,00287575	0,013976	0,0436925	0,105028	0,208353
(25,100), (4,1)	0,0113421	0,0982395	0,333254	0,883156	1,73139
(25,50), (4,2)	0,00600303	0,0511782	0,173479	0,45651	0,909415
(25,25), (4,4)	0,00332421	0,0276649	0,0933671	0,244626	0,499239
(25,13), (4,8)	0,00200581	0,0159043	0,0541254	0,139341	0,293493
(25,7), (4,16)	0,00112468	0,00853392	0,0299033	0,0769357	0,161691
(25,4), (4,32)	0,000785613	0,00627833	0,0203409	0,0496448	0,0981068
(25,2), (4,64)	0,00075275	0,00649899	0,0225368	0,0549465	0,108809
(25,1), (4,128)	0,000801067	0,00657611	0,0224558	0,0534359	0,105546
(25,1), (4,256)	0,00128588	0,00668206	0,0310655	0,0585048	0,10105
(13,100), (8,1)	0,00616929	0,0494859	0,183554	0,473645	1,00523
(13,50), (8,2)	0,00325181	0,0258567	0,095953	0,245881	0,520548
(13,25), (8,4)	0,00179955	0,0140242	0,0524291	0,132648	0,275596
(13,13), (8,8)	0,000990143	0,0074272	0,0284884	0,0705758	0,14343
(13,7), (8,16)	0,000700154	0,00497361	0,0164745	0,04035	0,0830443
(13,4), (8,32)	0,000663924	0,00474118	0,0157395	0,0370705	0,0731813
(13,2), (8,64)	0,000674623	0,00500314	0,0165477	0,0385732	0,0749097
(13,1), (8,128)	0,000813214	0,00633609	0,0230967	0,0548879	0,0957364
(7,100), (16,1)	0,00340489	0,0274822	0,109274	0,257288	0,518399
(7,50), (16,2)	0,00178084	0,0145039	0,0566542	0,132709	0,267889
(7,25), (16,4)	0,000935671	0,0077349	0,0291812	0,0678012	0,135674
(7,13), (16,8)	0,000704217	0,00494252	0,0166329	0,0398634	0,0786324
(7,7), (16,16)	0,00070349	0,00482278	0,0155288	0,0357821	0,0720962
(7,4), (16,32)	0,00071785	0,00491468	0,015867	0,0368181	0,0736323

(7,2), (16,64)	0,000817569	0,00689185	0,0214004	0,051141	0,0964066
(4,100), (32,1)	0,00196079	0,0175446	0,0591495	0,135167	0,262162
(4,50), (32,2)	0,00107925	0,00910258	0,030089	0,0686319	0,133582
(4,25), (32,4)	0,000789529	0,00536085	0,0174184	0,0396724	0,0762062
(4,13), (32,8)	0,000777362	0,00512003	0,0163767	0,0373762	0,0717485
(4,7), (32,16)	0,000786219	0,00523414	0,0166115	0,0381178	0,0736175
(4,4), (32,32)	0,000957676	0,00719887	0,022225	0,0513701	0,0968896
(2,100), (64,1)	0,00108424	0,0106447	0,0301395	0,0735919	0,133707
(2,50), (64,2)	0,000794378	0,00595996	0,0174892	0,0427634	0,0773151
(2,25), (64,4)	0,000782074	0,00580412	0,0163255	0,0403849	0,0719142
(2,13), (64,8)	0,000809492	0,00591088	0,0168304	0,0413688	0,0742767
(2,7), (64,16)	0,000888041	0,00757233	0,0222528	0,0545171	0,0987881
(1,100), (128,1)	0,000791741	0,00585208	0,0191244	0,0442679	0,0749795
(1,50), (128,2)	0,000788282	0,00579012	0,0163065	0,0401465	0,0711888
(1,25), (128,4)	0,000792139	0,00580744	0,0168082	0,0408835	0,0718112
(1,13), (128,8)	0,000842051	0,00665926	0,02195	0,0520001	0,0879074
(1,100), (256,1)	0,000791085	0,00582913	0,0167682	0,0404282	0,0711276
(1,50), (256,2)	0,000852605	0,00577745	0,0181622	0,0404814	0,0711684
(1,25), (256,4)	0,00112223	0,0062687	0,0257729	0,0490142	0,082093
(1,100), (512,1)	0,00140407	0,00759738	0,0177886	0,0421852	0,0712244
(1,50), (512,2)	0,00200147	0,0108373	0,024758	0,0471287	0,0763845
(1,100), (1024,1)	0,00393	0,0211562	0,0475997	0,0844644	0,134256

15 Priedas. CUDA π skaičiavimo diagramų duomenys

36 lentelė. π skaičiavimo spartinimo koeficientas

	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
1, 1	1	1	1	1	1	1	1	1	1	1
1, 32	0,995009	0,995291	0,995365	0,995414	0,999859	0,995456	0,99545	0,995485	0,995485	0,995497
1, 64	0,989068	0,989924	0,990067	0,99008	0,994622	0,990198	0,99465	0,990219	0,990268	0,990273
1, 128	0,971396	0,972806	0,973035	0,973284	0,977814	0,973537	0,97793	0,973643	0,973657	0,973692
1, 256	0,879658	0,884604	0,883402	0,884013	0,865039	0,88452	0,88866	0,884747	0,885075	0,884328
1, 512	0,62063	0,620379	0,622519	0,622238	0,625297	0,62238	0,623917	0,621589	0,622163	0,62286
32, 1	0,429124	0,427653	0,429978	0,427658	0,431838	0,427676	0,432674	0,430755	0,427678	0,429215
32, 32	0,450736	0,453823	0,452844	0,451783	0,454584	0,452965	0,455394	0,45354	0,454109	0,45461
32, 64	0,342897	0,345525	0,346373	0,347069	0,349081	0,347551	0,349425	0,347914	0,347181	0,347798
32, 128	0,187547	0,18858	0,189209	0,189076	0,190425	0,189454	0,190052	0,18963	0,189687	0,189734
32, 256	0,093835	0,094671	0,094821	0,094893	0,095475	0,094968	0,0955	0,095159	0,095081	0,095164
32, 512	0,03804	0,038534	0,03861	0,038755	0,038938	0,038838	0,038961	0,038821	0,038822	0,038834
64, 1	0,2148	0,2153	0,2145	0,2149	0,2159	0,2153	0,2151	0,2141	0,2145	0,2141

	96	54	66	54	72	75	5	95	79	97
64, 32	0,223837	0,226691	0,226819	0,226509	0,226784	0,226757	0,227796	0,227134	0,227067	0,226894
64, 64	0,171335	0,173613	0,173486	0,173302	0,174885	0,174277	0,175034	0,174248	0,174183	0,174196
64, 128	0,095988	0,096743	0,096914	0,097137	0,097573	0,097177	0,097618	0,097264	0,097309	0,097282
64, 256	0,047054	0,04761	0,047815	0,047899	0,048166	0,048034	0,048258	0,048055	0,048106	0,048098
64, 512	0,018613	0,019014	0,019138	0,019216	0,019375	0,019301	0,019394	0,019313	0,019354	0,019362
128, 1	0,107066	0,10737	0,107088	0,107092	0,107965	0,107095	0,107671	0,107193	0,107392	0,107489
128, 32	0,112521	0,112765	0,113383	0,112965	0,113644	0,113275	0,113934	0,113506	0,113407	0,113342
128, 64	0,085922	0,086948	0,086866	0,08713	0,087511	0,08736	0,087625	0,087365	0,087321	0,087317
128, 128	0,048556	0,04898	0,04913	0,049209	0,049485	0,049252	0,049563	0,049343	0,049355	0,049356
128, 256	0,023209	0,023717	0,023931	0,023978	0,024149	0,024091	0,024239	0,024133	0,024144	0,024156
128, 512	0,008998	0,009324	0,009464	0,009505	0,009617	0,009583	0,009644	0,009626	0,009613	0,009632
256, 1	0,053655	0,053564	0,053545	0,053594	0,053884	0,053596	0,053787	0,05362	0,05362	0,053573
256, 32	0,056038	0,056506	0,056599	0,056452	0,056904	0,056498	0,0569	0,056708	0,056699	0,056781
256, 64	0,042933	0,04345	0,043561	0,043644	0,043898	0,043787	0,04396	0,043781	0,043812	0,043822
256, 128	0,024142	0,024538	0,024645	0,024703	0,02484	0,02477	0,024901	0,024815	0,024814	0,024829
256, 256	0,051167	0,053359	0,054087	0,05449	0,054945	0,054889	0,055251	0,055033	0,05513	0,055132
256, 512	0,004237	0,004516	0,004617	0,004672	0,004725	0,004731	0,004766	0,004759	0,004767	0,004782
512, 1	0,026773	0,026794	0,026779	0,026785	0,026904	0,026831	0,026893	0,026807	0,02682	0,026795
512, 32	0,027964	0,028112	0,02825	0,028271	0,028432	0,028342	0,02847	0,028304	0,028365	0,02837
512, 64	0,021281	0,021652	0,021818	0,021866	0,022011	0,021939	0,022045	0,021969	0,021982	0,021974
512, 128	0,011736	0,012167	0,012284	0,012342	0,012418	0,012383	0,01245	0,012408	0,01242	0,012423
512, 256	0,005303	0,005629	0,005778	0,005859	0,005931	0,005934	0,005984	0,005973	0,005988	0,005997
512, 512	0,001999	0,002183	0,002251	0,002293	0,002326	0,002333	0,002356	0,002355	0,002361	0,00237

37 lentelė. π skaičiavimo spartinimo koeficientas

	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
1, 1	1	1	1	1	1	1	1	1	1	1
1, 32	31,84028	31,84093	31,85168	31,85325	31,99 55	31,85 46	31,85 439	31,85 553	31,85 552	31,855 91
1, 64	63,30034	63,35 517	63,36 43	63,36 515	63,65 582	63,37 269	63,65 761	63,37 402	63,37 718	63,377 47
1, 128	124,3	124,5	124,5	124,5	125,1	124,6	125,1	124,6	124,6	124,63

	386	192	485	803	602	127	767	264	281	26
1, 256	225,1925	226,4586	226,151	226,3072	221,4499	226,437	227,4969	226,4952	226,5791	226,3879
1, 512	317,7628	317,6342	318,7296	318,5857	320,1519	318,6587	319,4453	318,2536	318,5476	318,9043
32, 1	13,73196	13,68489	13,75931	13,68505	13,81881	13,68562	13,84556	13,78417	13,68569	13,73487
32, 32	461,5541	464,7147	463,712	462,6254	465,494	463,8364	466,3231	464,4249	465,0075	465,5203
32, 64	702,2533	707,636	709,3713	710,7983	714,9186	711,7854	715,623	712,5283	711,0277	712,2898
32, 128	768,1916	772,423	774,9997	774,4556	779,9826	776,0038	778,4549	776,723	776,957	777,1501
32, 256	768,6956	775,5489	776,7724	777,3649	782,1305	777,9774	782,3325	779,5448	778,9052	779,5812
32, 512	623,2538	631,3462	632,5808	634,9693	637,9626	636,3282	638,337	636,0374	636,0587	636,2502
64, 1	13,75334	13,78268	13,73219	13,75707	13,82222	13,78398	13,76962	13,70849	13,73307	13,7086
64, 32	458,4178	464,2632	464,5253	463,8907	464,4531	464,3981	466,5269	465,1709	465,0327	464,6798
64, 64	701,7884	711,117	710,5991	709,8436	716,3293	713,8395	716,9396	713,718	713,454	713,5074
64, 128	786,3345	792,5166	793,9209	795,7465	799,3185	796,0755	799,685	796,7848	797,1527	796,9346
64, 256	770,936	780,0416	783,4085	784,782	789,1467	786,9842	790,6572	787,3273	788,1635	788,0391
64, 512	609,9267	623,0578	627,1152	629,6735	634,8715	632,4616	635,4864	632,8619	634,1797	634,4581
128, 1	13,70444	13,7434	13,70725	13,70773	13,81946	13,70818	13,7819	13,72072	13,74619	13,75859
128, 32	460,8869	461,8873	464,4155	462,7056	465,4875	463,9733	466,673	464,92	464,5134	464,2486
128, 64	703,8704	712,2792	711,606	713,7685	716,8896	715,6498	717,8273	715,6922	715,3347	715,2993
128, 128	795,5416	802,4842	804,9422	806,2326	810,7697	806,9524	812,0335	808,443	808,6327	808,6486
128, 256	760,5158	777,1665	784,1858	785,7272	791,3231	789,4183	794,2612	790,7941	791,1668	791,5462
128, 512	589,6788	611,0303	620,2171	622,8942	630,2905	628,0604	632,048	630,8184	630,0021	631,2529
256, 1	13,73444	13,71243	13,70744	13,72009	13,79425	13,72051	13,76948	13,72673	13,72671	13,71461
256, 32	459,0669	462,8941	463,661	462,4573	466,1571	462,8292	466,1217	464,5481	464,4775	465,1507
256, 64	703,4219	711,8768	713,7048	715,0591	719,2261	717,4114	720,2383	717,3112	717,8077	717,9753
256, 128	791,0752	804,0686	807,5522	809,4755	813,9668	811,6547	815,9702	813,1368	813,1176	813,5987
256, 256	733,5365	764,9562	775,3851	781,1723	787,6911	786,8838	792,0842	788,9491	790,3398	790,3753
256, 512	555,3629	591,9234	605,1343	612,4056	619,2707	620,0766	624,7035	623,7487	624,8707	626,786

512, 1	13,70777	13,71841	13,71091	13,714	13,77508	13,73753	13,76937	13,72522	13,73191	13,71918
512, 32	458,1568	460,5925	462,8465	463,199	465,8269	464,3578	466,4481	463,7337	464,7379	464,8221
512, 64	697,3516	709,4853	714,9247	716,4974	721,2699	718,8979	722,3822	719,8917	720,2936	720,0304
512, 128	769,1161	797,3776	805,0296	808,8714	813,8118	811,5528	815,9418	813,1863	813,9599	814,1789
512, 256	695,033	737,7518	757,3085	767,9306	777,4534	777,7887	784,3252	782,9374	784,9215	786,0497
512, 512	523,9896	572,2281	590,1734	601,1977	609,7859	611,6904	617,678	617,3917	618,8777	621,1779

38 lentelė. π skaičiavimo laikas

	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
1, 1	1,33577	2,67151	4,00726	5,343	6,70856	8,01449	9,39198	10,686	12,0217	13,3575
1, 32	0,0419522	0,0838797	0,12581	0,167738	0,209672	0,251596	0,294841	0,335452	0,377382	0,41931
1, 64	0,0211021	0,0421672	0,0632416	0,0843208	0,10538	0,12646	0,147539	0,168618	0,189685	0,210761
1, 128	0,010743	0,0214546	0,0321743	0,042888	0,0535998	0,0643152	0,0750298	0,0857443	0,0964606	0,107175
1, 256	0,00593168	0,0117969	0,0177194	0,0236095	0,0302938	0,0353939	0,041284	0,0471798	0,0530574	0,0590027
1, 512	0,00420367	0,00841065	0,0125726	0,016771	0,0209543	0,0251507	0,0294009	0,033577	0,0377391	0,0418856
32, 1	0,0972745	0,195216	0,29124	0,390426	0,485466	0,585614	0,678339	0,775237	0,878414	0,972525
32, 32	0,00289407	0,00574871	0,0086417	0,0115493	0,0144117	0,0172787	0,0201405	0,0230091	0,0258527	0,0286937
32, 64	0,00190212	0,00377526	0,00564903	0,0075169	0,00938367	0,0112597	0,0131242	0,0149973	0,0169075	0,0187529
32, 128	0,00173885	0,00345861	0,00517066	0,00689904	0,00860091	0,0103279	0,0120649	0,0137578	0,0154728	0,0171878
32, 256	0,00173771	0,00344467	0,00515886	0,00687322	0,00857729	0,0103017	0,0120051	0,013708	0,0154341	0,0171342
32, 512	0,00214322	0,00423145	0,00633478	0,00841458	0,0105156	0,0125949	0,0147132	0,0168009	0,0189003	0,0209941
64, 1	0,0971233	0,193831	0,291815	0,388382	0,485346	0,581435	0,68208	0,779517	0,875383	0,974388
64, 32	0,00291387	0,00575543	0,00862657	0,0115178	0,01444	0,0172578	0,0201317	0,0229722	0,0258513	0,0287456
64, 64	0,00190338	0,00375678	0,00563927	0,00752701	0,00936519	0,0112273	0,0131001	0,0149723	0,016785	0,0187209
64, 128	0,00169873	0,00337092	0,00504743	0,00671445	0,00839285	0,0100675	0,0117446	0,0134114	0,0150808	0,0167611
64, 256	0,00173266	0,00342483	0,00511516	0,00680826	0,00850103	0,0101838	0,0118787	0,0135725	0,0152528	0,0169503
64, 512	0,00219005	0,00428774	0,00638999	0,00848535	0,0105668	0,0126719	0,0147792	0,0168852	0,0189563	0,0210534
128, 1	0,0974699	0,194385	0,292346	0,38978	0,485443	0,58465	0,681472	0,778822	0,874548	0,970848

128, 32	0,0028 9826	0,0057 839	0,0086 2861	0,0115 473	0,0144 119	0,0172 736	0,020 1254	0,022 9846	0,025 8802	0,028 7723
128, 64	0,0018 9775	0,0037 5065	0,0056 3129	0,0074 8562	0,0093 5787	0,0111 989	0,013 0839	0,014 931	0,016 8057	0,018 674
128, 128	0,0016 7907	0,0033 2905	0,0049 7832	0,0066 2712	0,0082 7431	0,0099 318	0,011 566	0,013 218	0,014 8667	0,016 5183
128, 256	0,0017 564	0,0034 375	0,0051 1009	0,0068 0007	0,0084 7765	0,0101 524	0,011 8248	0,013 513	0,015 1949	0,016 8752
128, 512	0,0022 6525	0,0043 7214	0,0064 6106	0,0085 777	0,0106 436	0,0127 607	0,014 8596	0,016 9399	0,019 082	0,021 1603
256, 1	0,0972 57	0,1948 24	0,2923 42	0,3894 29	0,4863 3	0,5841 25	0,682 087	0,778 481	0,875 789	0,973 961
256, 32	0,0029 0975	0,0057 7132	0,0086 4265	0,0115 535	0,0143 912	0,0173 163	0,020 1492	0,023 003	0,025 8822	0,028 7165
256, 64	0,0018 9896	0,0037 5277	0,0056 1473	0,0074 7211	0,0093 2747	0,0111 714	0,013 0401	0,014 8973	0,016 7478	0,018 6044
256, 128	0,0016 8855	0,0033 2249	0,0049 6223	0,0066 0057	0,0082 4181	0,0098 7426	0,011 5102	0,013 1417	0,014 7847	0,016 4178
256, 256	0,0018 21	0,0034 9237	0,0051 6809	0,0068 3972	0,0085 1674	0,0101 851	0,011 8573	0,013 5446	0,015 2108	0,016 9002
256, 512	0,0024 0522	0,0045 1327	0,0066 221	0,0087 2461	0,0108 33	0,0129 25	0,015 0343	0,017 1319	0,019 2387	0,021 3111
512, 1	0,0974 462	0,1947 39	0,2922 68	0,3896 02	0,4870 07	0,5834 01	0,682 092	0,778 567	0,875 457	0,973 637
512, 32	0,0029 1553	0,0058 0016	0,0086 5786	0,0115 35	0,0144 014	0,0172 593	0,020 1351	0,023 0434	0,025 8677	0,028 7368
512, 64	0,0019 1549	0,0037 6542	0,0056 0515	0,0074 5711	0,0093 0104	0,0111 483	0,013 0014	0,014 8439	0,016 69	0,018 5513
512, 128	0,0017 3676	0,0033 5037	0,0049 7778	0,0066 055	0,0082 4338	0,0098 755	0,011 5106	0,013 1409	0,014 7694	0,016 4061
512, 256	0,0019 2188	0,0036 2115	0,0052 9145	0,0069 5766	0,0086 2889	0,0103 042	0,011 9746	0,013 6486	0,015 3158	0,016 9932
512, 512	0,0025 4923	0,0046 6861	0,0067 8997	0,0088 8726	0,0110 015	0,0131 022	0,015 2053	0,017 3083	0,019 425	0,021 5035

16 Priedas. CUDA rikiavimo diagramų duomenys

39 lentelė. Bitonic rikiavimo laikas pagal gijų konfigūraciją

	32768	65536	131072	262144	524288	1048576	2097152	4194304
32, 1024	0,0004746 8							
64, 512	0,0004538 46							
64, 1024	0,0004538 25	0,000515 35						
128, 256	0,0004548 04							
128, 512	0,0004786 24	0,001127 7						
128, 1024	0,0004550 56	0,000514 04	0,00057 6					
256, 128	0,0004754							

	55							
256, 256	0,000468385	0,00053223						
256, 512	0,000451883	0,0005476	0,000598					
256, 1024	0,000451104	0,00053575	0,000597	0,000659				
512, 64	0,000496242							
512, 128	0,000454192	0,00056983						
512, 256	0,000451729	0,0005103	0,000576					
512, 512	0,000458117	0,00051065	0,000613	0,000653				