KAUNO TECHNOLOGIJOS UNIVERSITETAS
Informatikos fakultetas

P170B328 Lygiagretusis programavimas

Individualus Projektas

Data: 2019.11.27

Dėstytojai:
lekt. Barisas Dominykas
lekt. Vasiljevas Mindaugas
Studentas:
Zubowicz Edwin
Grupė: IFF-7/2

KAUNAS, 2019

Turinys

Įvadas

Lygiagretaus programavimo modulio paskirtis yra suprasti kas yra lygiagretusis programavimas, išmokti jį naudoti praktikoje, tam, kad paspartinti programų veikimą. Šio individualiaus projekto tikslas yra dar geriau prisisavinti lygiagretaus programavimo praktinius įgudžius.

Užduotis

Užduotis šiam projektui buvo pasirinkta pagal Skaitinių metodų ir algoritmų modulio antrojo laboratorinio darbo optimizavimo užduotį. Šios užduoties turinys (pagal 23 variantą):

„Duotos n ($3 \leq n$) taškų koordinatės ($-10 \leq x \leq 10$, $-10 \leq y \leq 10$). (Koordinatės gali būti generuojamos atsitiktiniu būdu). Srityje ($-10 \leq x \leq 10$, $-10 \leq y \leq 10$) reikia padėti papildous m ($3 \leq m$) taškus taip, kad jų atstumai nuo visų kitų taškų (įskaitant ir papildomus) būtų kuo artimesni vidutiniam atstumui, o atstumas nuo koordinačių pradžios būtų kuo artimesnis nurodytai reikšmei S ($1 \leq S$).“

Sprendimo metodas

Šios užduoties išsprendimui reikalingas duotas eiliškumas: sugeneravimas atsitiktinių taškų koordinačių sistėmoje, apskaičiavimas ilgių tarp duotų taškų, suskaičiavimas stygų kainų pagal prieš tai suskaičiuotus ilgius, suskaičiavimas gradiento, priskyrimas naujų koordinačių taškams.

Lygiagretumą galima panaudoti tik vienoje vietoje t. y. gradiento skaičiavime. Gradientas rodo, kurioje vietoje funkcija auga sparčiausiai. Šią reikšmę reikia skaičiuot kekvieno taško x ir y koordinatei. Jeigu taškų turime n, tada per vieną iteraciją reikia suskaičiuot 2n tokiu pačiu skaičiavimu, bet nepriklausančiu vienas nuo kito, dėl to šioje vietoje panaudojamas lygiagretumas.

Ši užduotis atliekama go kalba, naudojant gijas bei vieną kanalą.

1. Programos aprašymas

1.1. Metodai

fillWithRandomPoints(min, max, count int) [][]float64

Šis metodas sugeneruoja duotą kiekį taškų koordinačių sistemoje. Priima argumentus min, max ir coun, kurie yra sveikieji skaičiai. Min ir max yra intervalas kuriame taškai turi būti sugeneruoti, count reiškia taškų skaičių. Metodas gražina dvimatį masyvą.

countLength(x1 []float64, x2[] float64) float64

Metodas countLength apskaičiuoja ilgį tarp dviejų taškų. X1 ir x2 yra taškų koordinatės. Metodas gražina realų skaičių – ilgį.

costFunction(points[][]float64) float64

Šis medotas apskaičiuoja kainą visų stygų tarp taškų t. y. ilgių. Points tai dvimatis masyvas, kuriame yra visi taškai ir jų koordinatės. Metodas gražina realų skaičių - kainą.

makeCopy(points [][]float64) [][]float64

Metodas makeCopy sukurtas tam, kad sudaryti kopija dvimačio masyvo. Priima taškų dvimatį masyvą ir gražina jo kopija.

countVectorLength(vector []float64) float64

Šis metodas suskaičiuoja skaliarinį ilgį viso vektoriaus. Šis metodas reikalingas normalizuoti duotą vektorį. Metodas priima argumentą vektoriui ir grąžina realų skaičių jo ilgį.

countPointGradient(points [][]float64, pointsChanel <-chan []int, wg *sync.WaitGroup)

Metodas countPointGradient apskaičiuoja duodo taško x-o arba y-ko gradientą. Šis metodas reikalingas yra, kad sukurti gradiento vektorių. Argumentai kuriuos priima yra points- visų taškų dvimatis masyvas, wg- įrankis reikalingas sinchronizuoti gijas ir pointsChanel - kanalas per kurį perduodami duomenys skaičiavimui. Metodas, atlikus skaičiavimus, įdeda rezultatą į globalų gradiento masyvą.

executeGradient(threadsCount int, points [][]float64)

Metodas, kuris valdo gijas.

normalizeGradientVector(vector []float64)

Šis metodas normalizuoja gradiento vektorių. Į argumentus gauna vektorių- masyvą ir su duotu vektoriu atlieka atitinkamus veiksmus, kad vektorių sunormalizuot.

findLowerCost(points [][]float64, alpha float64)

Pagrindinis metodas, kuriame yra visi veiksmai, kad surasti geresnę stygų kainą. Šiame metode yra kviečiamos visos baigimo sąlygos ir visi reikalingi metodai. Metodas baigęs darbą išspausdina rezultatus į konsolę.

1.2. Programos pagrindiniai tekstai

Metodas findLowerCost, kuris atsako už visą geresnės kainos suradimą. Šio metodo užbaigimo sąlyga yra pasiekimas maksimalaus iteracijų skaičiaus arba sumažinimas žingsnio (alpha) iki 1e-5. Kai kažkurį iš šių sąlygu bus pasiekta, programa baigs savo darbą.

```
func findLowerCost(points [][]float64, alpha float64) {
    var iterationsCount = 0
    var result float64
    fmt.Println("\nPradinė kaina: \n", costFunction(points))

    // mazint zingsny
    for iterationsCount < MaxIterationCount && alpha > 1e-5 {
        iterationsCount++
        fValue := costFunction(points)
        count := 0
        concurencyGrandient(points)
        normalizeGradientVector(grad)
        pointsCopy := makeCopy(points)
        for i, point := range pointsCopy {
            for j, xy := range point {
                pointsCopy[i][j] = float64(xy) - (alpha * grad[count])
                count++
            }
        }
        fValueNext := costFunction(pointsCopy)
        if fValueNext < fValue {
            points = pointsCopy
            result = fValueNext
        } else {
            alpha = alpha / 2
        }
    }
    fmt.Println("\nGalutinė kaina: \n", result)
    fmt.Println("\nIteraciju skaicius: ", iterationsCount)
}
```

Metodas executeGradient atsakingas yra už gijų paleidimą ir laukimą, kol visos gijos baigs darbą. Šiame metode yra paleidžiamas atitinkamas gijų skaičius. Paleidus visas gijas, metodas perduoda duomenis į kanalą ir laukia kol visos gijos baigs darbą, tada metodas pats baigia darbą.

```
func executeGradient(threadsCount int, points [][]float64) {
    wg := sync.WaitGroup{}
    var index int = 0
    // create channels
    send := make(chan []int)
    wg.Add(threadsCount)
    for i := 0; i < threadsCount; i++ {
        go countPointGradient(points, send, &wg)
    }
    for i, point := range points {
        for j := range point {
            send <- []int{index, i, j}
            index++
        }
    }
    close(send)
    wg.Wait()
}
```

Metodas countPointGradient suskaičiuoja kekvienai taško koordinatei duoto taško gradientą. Šis metodas vykdomas lygiagrečiai, kadangi kekvienos koordinatės skaičiavimas gali būti vykdomas nepriklausomai vinena nuo kitos.

```
func countPointGradient(points [][]float64, pointsChanel <-chan []int, wg *sync.WaitGroup) {
    defer wg.Done()

    for xy := range pointsChanel {
        if xy != nil && xy[0] < 2 {
            grad[xy[0]] = 0
        } else if xy != nil {
            var pointsCopy = makeCopy(points)
            pointsCopy[xy[1]][xy[2]] += h
            grad[xy[0]] = (costFunction(pointsCopy) - costFunction(points)) / float64(h)
        }
    }
}
```

2. Testavimas ir programos instaliavimo bei vykdymo instrukcija

2.1. Testavimas

Testavimui galima naudot žemiau duotus duomenis. Jeigu gaunami rezultatai tokie patys, reiškiasi programa veikia taisyklingai. Laikas gali skirtis skirtinguose kompiuteriuose.

Keičiamos tik konstantos programos pradžioje. Count- taškų kiekis, c- rand.Seed(c) reikšmė, pagal kurią parenkami atsitiktiniai taškai.

Pirmas rinkinys:
Count = 10, c = 1
Rezultatai:
Pradinė kaina: 3907.250103119917, Galutinė kaina: 4.944609075135628e-06,
Iteraciju skaicius: 2602, Laikas: 281.537981ms

Antras rinkinys:
Count 20, c = 1
Rezultatai:
Pradinė kaina: 20911.403658968426, Galutinė kaina: 2.699620587945922e-05
Iteraciju skaicius: 4336, Laikas: 3.683346561s

Trečias rinkinys:
Count = 10, c = 2
Rezultatai:
Pradinė kaina: 4308.198720175277, Galutinė kaina: 4.94462086931804e-06
Iteraciju skaicius: 2911, Laikas: 328.108813ms

Ketvirtas rinkinys:
Count = 20, c = 2
Rezultatai:
Pradinė kaina: 21848.310149675708, Galutinė kaina: 2.6992199059207145e-05
Iteraciju skaicius: 4668, Laikas: 3.545859992s

2.2. Programos instaliavimo bei vykdymo instrukcija

Norint įsirašyti programą savo kompiuteryje, reikia parsisiųsti duotą programa iš moodle bei iš kitos vietos, kurioje ši programa yra (pvz github: https://github.com/AbziBzi/KTU_LP.git).

Kai programa bus jau parsiųsta į kompiuterį, įejus į programos aplanką užtenka atidaryti terminalą ir suvesti komandą *go run main.go* . Suvedus šia komandą, programa vykdys savo darbą.

Norint įvesti pakeitimus programoje, reikia atidaryti failą main.go bet kokiame teksto editoriuje, sudaryti ir išsaugoti pakeitimus.

Prieš vykdant programą, reikia turėti įdiegtą go kalbą savo kompiuteryje.

3. Vykdymo laiko kitimo tyrimas

3.1. Duomenų rinkiniai

Parinkta 10 skirtingu dydžių duomenų rinkinių. Šiam tyrimui parinkti tokių kiekių duomenų rinkiniai: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50. Šie skaičiai reiškia taškų kiekį koordinačių sistemoje. Kekvienam taškų rinkiniui surandama mažiausia kaina naudojant gijų kiekius: 1, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200.

Apačioje matome grafikus kekvienam rinkiniui, kuriuose parodoma laiko priklausomybė nuo procesų skaičiaus. Juose matome: x ašyje procesų skaičiu o y ašyje gautus laikus.

Pagal duotus grafikus galima pasakyti, kad su viena gija programos veikimas yra pats lečiausias, tačiau taip pat matome, kad naudojant labai daug gijų, t. y. 100 ar 200 programos realizacijos sparta ne didėja, o net mažėja. Optimaliausias gijų skaičiu šiai užduočiai ir praktiškai visiems rinkiniams būtų 15 iki 100 gijų.

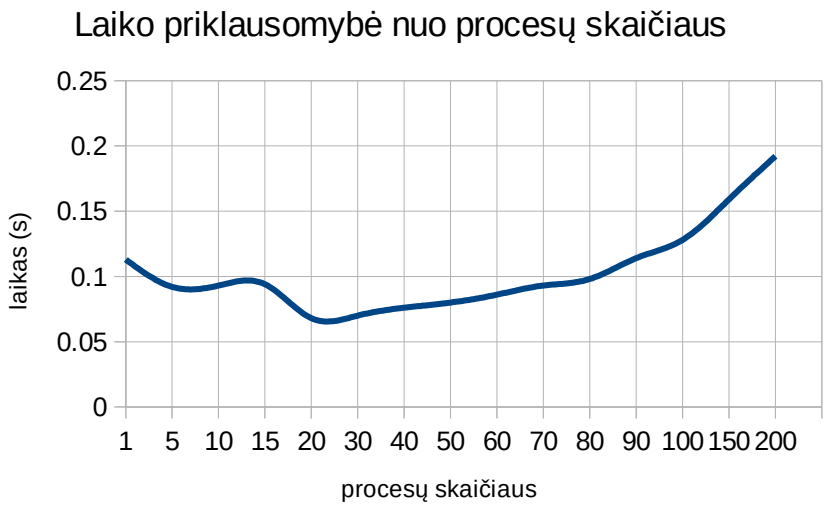

Diagrama 1: Laiko priklausomybė nuo procesų skaičiaus su 5 taškais

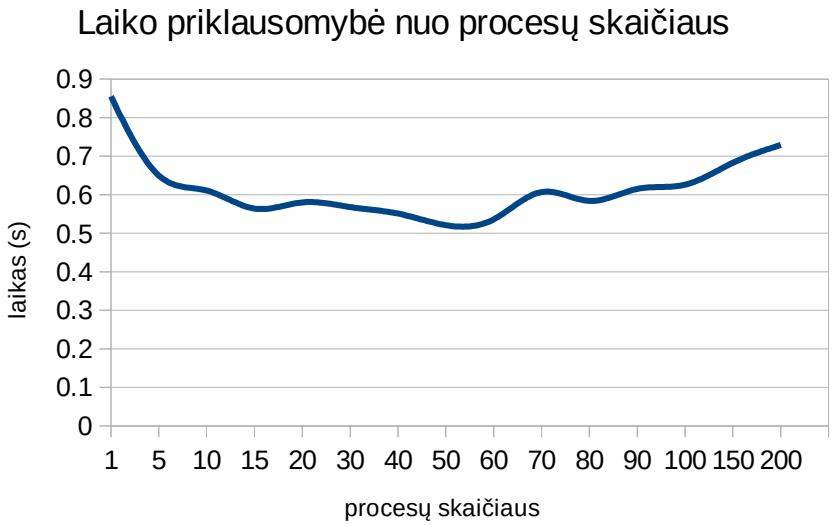

Diagrama 2: Laiko priklausomybė nuo procesų skaičiaus su 10 tašų

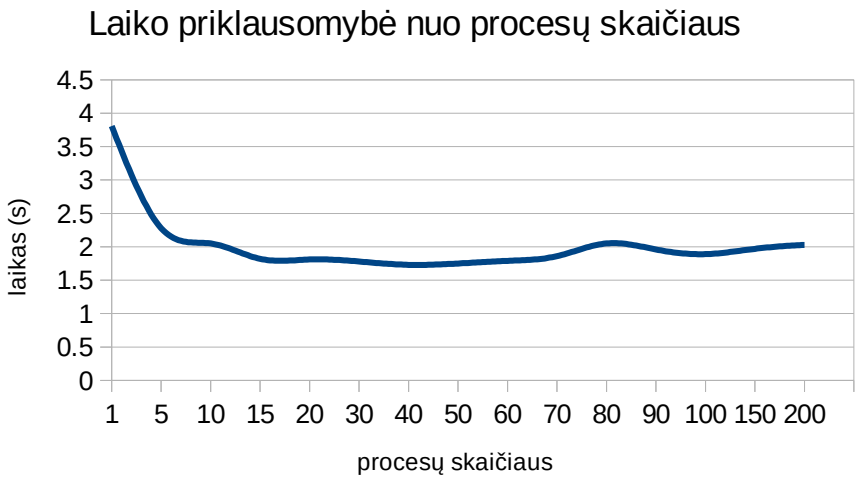

Diagrama 3: Laiko priklausomybė nuo procesų skaičiaus su 15 taškų

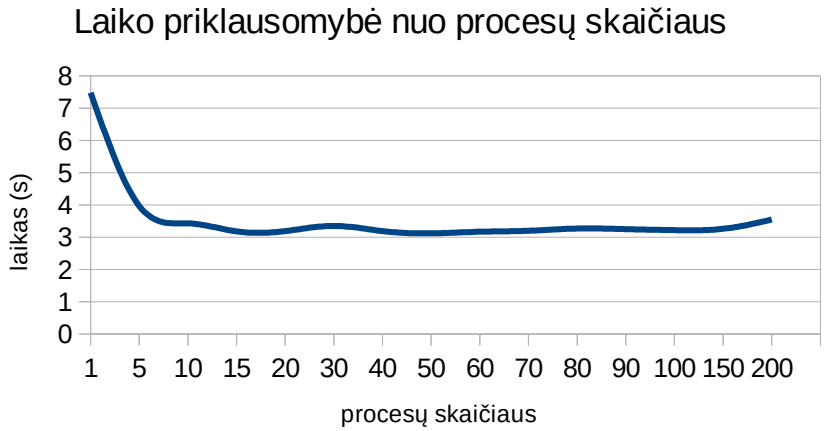

Diagrama 4: Laiko priklausomybė nuo procesų skaičiaus su 20 taškais

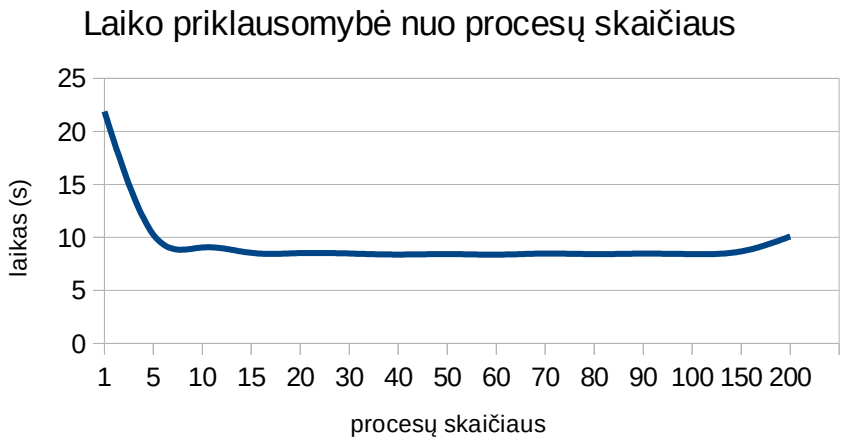

Diagrama 5: Laiko priklausomybė nuo procesų skaičiaus su 25 taškais

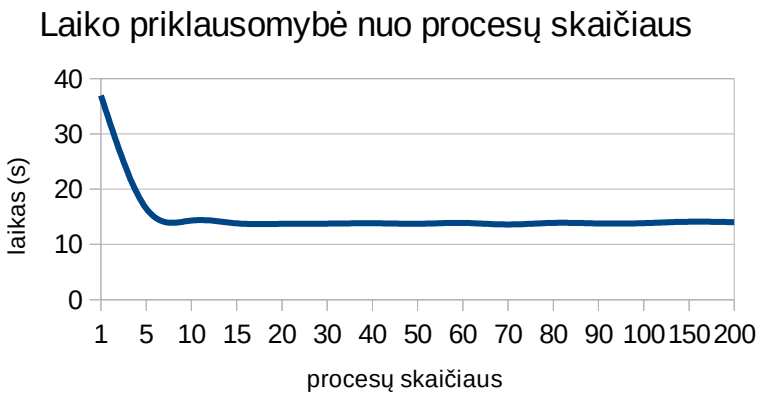

Diagrama 6: Laiko priklausomybė nuo procesų skaičiaus su 30 taškų

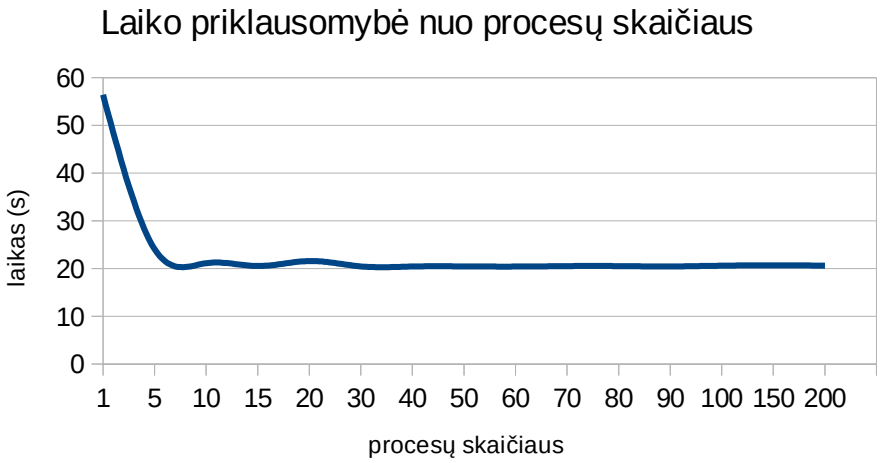

Diagrama 7: Laiko priklausomybė nuo procesų skaičiaus su 35 taškais

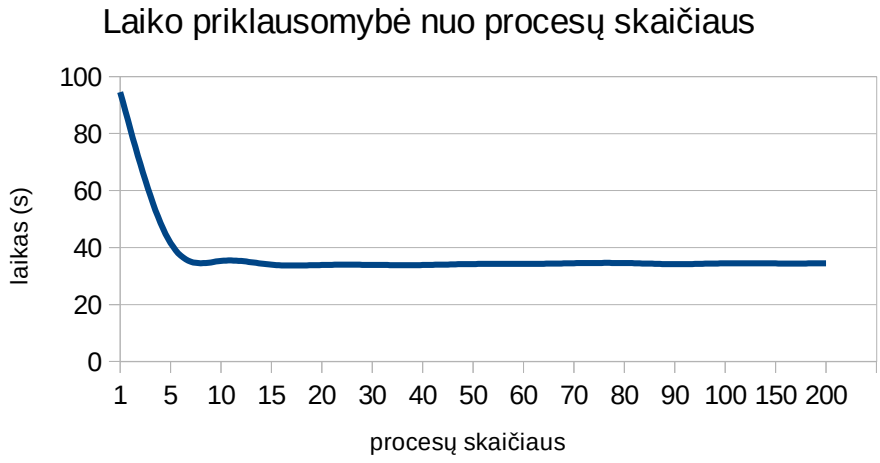

Diagrama 8: Laiko priklausomybė nuo procesų skaičiaus su 40 taškais

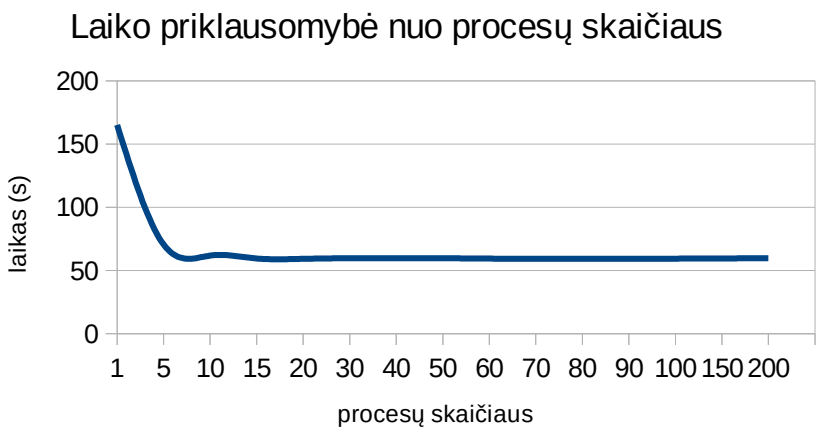

Diagrama 9: Laiko priklausomybė nuo procesų skaičiaus su 45 taškais

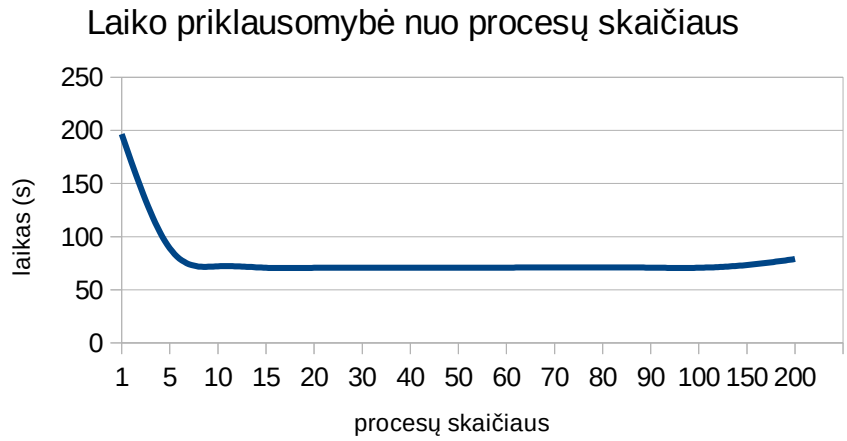

Diagrama 10: Laiko priklausomybė nuo procesų skaičiaus su 50 taškais

Išvados

Individualaus projekto metu, buvo lygiagretinama programa, kurią reikėjo sukurti skaitinių metodų ir algoritmų modulio antrame namų darbe. Šios užduoties esmė buvo optimizuoti duotą užduotį. Lygiagretaus programavimo individualiame projekte reikėjo šią programą sulygiagretint. Ši užduotis pavyko. Lygiagretumą pavyko pritaikyti gradiento skaičiavime. Kekvienam taškui reikėjo suskaičiuoti gradientadu kartus. Du kartus todėl, nes taškas turi x ir y koordinate. Pagal tai išeina, kad programa vieno suradimo geresnės kainos metu, turėjo suskaičiuoti du kart taškų kiekio skaičiavimus, kurie nepriklauso vienas nuo kito.

Sulygiagretinus programą gavosi, jog programa veikia apytiksliai 2.39 karto sparčiau nei nenaudojant gijų. Taip pat pastebėta, kad prie mažesnio kiekio duomenų daugiau gijų ne spartina programos o ją lėtina, pvz pagal pirmą diagramą su duomenų kiekiu lygiam 5 matome, jog sparčiausiu metu programos vykdymas užtrunka apie 0.06 s su 20 gijų, o naudojant 200 gijų programos vykdymo laikas padidėja iki 0.19 sekundės, kas yra lečiau nei su 1 gija. Tačiau prie didelių kiekių, tas skirtumas nėra net toks didelis. Jis turbūt yra, bet prie dar didėsnio gijų skaičiaus, kuris čia nebuvo nagrinėtas.

Prie mažio duomenų kiekio skirtumas tarp naudojimo lygiagretumo o jo nenaudojimo nera net toks didelis. Su 5 taškais programa pagreitėja per 1.66 karto. Tačiau prie dideliu kiekių programos sparta didėja naudojant lygiagretumą, su 50 taškų programos sparta padidėjo per 2.77 karto.

Literatūra

1. https://moodle.ktu.edu/pluginfile.php/45742/mod_resource/content/8/02-Gijos-%C4%AFvadas-%C4%AF-Go.pdf
2. https://moodle.ktu.edu/pluginfile.php/45747/mod_resource/content/5/pranešimų_perdavimas.pdf
3. https://golang.org/pkg/sync/