KAUNO TECHNOLOGIJOS UNIVERSITETAS
Informatikos fakultetas

P170B328 Lygiagretusis programavimas

Individualus Projektas

Data: 2019-12-04

Dėstytojai:
lekt. Barisas Dominykas
lekt. Vasiljevas Mindaugas
Studentas:
Justas Milišiūnas
Grupė: IFF-7/2

KAUNAS, 2019

Turinys

Įvadas

Šio projekto tikslas išmokti naudoti lygiagretų programavimą praktikoje. Palyginti kaip programos vykdymo laikas skiriasi naudojant lygiagretinimo principus.

Užduotis

Plokštumoje ($-10 \leq x \leq 10$, $-10 \leq y \leq 10$) išsidėstę n taškų ($3 \leq n$), vienas jų fiksuotas koordinačių pradžioje (0; 0). Kiekvienas taškas su visais kitais yra sujungtas tiesiomis linijomis (stygomis). Raskite tokias taškų koordinates, kad atstumas tarp taškų būtų kuo artimesnis vidutiniam atstumui, o stygų ilgių suma kuo geriau atitiktų nurodytą reikšmę S ($10 \leq S$).

Sprendimo metodas

Ši uždavinį sprendimas:

1. Pradinių taškų susigeneravimas
2. Gradiento paskaičiavimas kiekvienam taškui kiekvienoje iteracijoje
3. Stygų ilgių sumos paskaičiavimas naudojant pradinius taškus
4. Stygų ilgių sumos skaičiavimas naudojant pakeistus taškus pagal gradientą
5. Jeigu suma pagerėja pradiniams taškams priskiriamia pakeistus taškus

Lygiagretinimas buvo padarytas gradiento paskaičiavimo vietoje. Dėl to, kad kiekvienoje iteracijoje reikia paskaičiuoti gradientą kiekvienam taškui. Su dideliu taškų skaičiu ši operacija užtrunka ilgai.

Šio optimizavimo uždavinio lygiagretinimui naudojama Python kalba su procesų paleidimu.

1. Programos aprašymas

1.1. Metodai

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

Šis metodas sugeneruoja duota kieki taškų koordinačių sistėmoje. 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 dvimaty masyvą.

- **generate_points()** - sugeneruoja taškus priklausomai nuo globaus kintamojo n
- **distance(point1, point2)** – paskaičiuoja atstumą tarp dviejų taškų
- **distance_sum(points)** - paskaičiuoja visų stygų ilgių vidurkį
- **optimize_points(points)** – pagrindinis metodas, kuris kviečia visus kitus metodus. Turi ciklą kur kiekvienoje iteracijoje paskaičuojamas taškų gradientas. Taškai perstumiami pagal gradientą, lyginamas stygų ilgių vidurkių palyginimas. Vyksta tol kol pasieka nustatytą iteracijų limitą arba pasiekia nurodyta tikslumą
- **move_by_gradient(gradient_vector, points)** – perstumia taškus pagal gradiento vektorių
- **points_gradient_vector(points, current_sum)** – paskaičiuoja kiekvieno taško gradientą iš kurių susidaro vektorius. Šitame metode padarytas lygiagretinimas
- **point_gradient(i, points, current_sum)** – paskaičiuoja taško pagl duotą indeksą iš taškų masyvo
- **connect_each_point(points)** – sudaro masyvą, kuriame sujungiami visi taškai. Naudojas rezultato grafike norint atvaizduoti stygas jungiančias taškus

1.2. Programos pagrindiniai tekstai

```
def optimize_points(points):
    global alpha
    points = copy.deepcopy(points)
    max_iterations = 1000
    current_sum = distance_sum(points)
    counter = 0
    while counter < max_iterations and alpha >= eps:
        counter += 1
        points_gradient = points_gradient_vector(points, current_sum)
        gradient_norm = [item / np.linalg.norm(points_gradient) for item in
                         points_gradient]
        moved_points = move_by_gradient(gradient_norm, points)
        next_sum = distance_sum(moved_points)
        if next_sum < current_sum:
            points = moved_points
            current_sum = next_sum
        else:
            alpha /= 2
    return points, current_sum, counter + 1
```

Šis metodas bando optimizuoti taškus tol kol pasiekia iteracijų limitą(šiuo atveju 1000) arba taško pakeitimo žingsnis(alpha) tampa mažesniu nei nurodytas limitas(eps=1e-6). Kiekvienoje iteracijoje paskaičiuoja taškų gradientą, perstumia taškus pagal gradientą, paskaičiuoja stygų ilgių vidurkį su pakeistais taškais. Galiausiai palygina tą vidurkį su prieš tai buvusių taškų vidurkiu. Jeigu naujas vidurkis mažesnis tai jis tampa dabartiniu, pakeisti taškai tampa pradiniais taškais. Jeigu vidurkis didesnis tada mažinamas žingsnius(alpha) ir skaičiuojama iš naujo.

```
def points_gradient_vector(points, current_sum):
    gradients = [0.0, 0.0]
    # Gives work to workers pool
    args = partial(point_gradient, points=points, current_sum=current_sum)
    result = pool.map(args, range(1, len(points)))
    # Converts 2d array to 1d
    [gradients.extend(point) for point in result]
    return gradients
```

Šis metodas paskaičiuoja gradientą kiekvienam taškui. Šioje vietoja panaudotas lygiagretumas. Naudojamas iš anksto sukurtas darbuotojų(procesų) pool. Su metodu pool.map darbas paskirstomas kiekvienam darbuotojui tolygiai. Rezultatai iš kiekvieno darbuotojai sustatomi į result masyvą eilės tvarka. Galiausiai tie rezultatai pridedami į gradients masyvą ir grąžinami.

```
def point_gradient(i, points, current_sum):
    # Point's x
    changed_points_x = copy.deepcopy(points)
    changed_points_x[i][0] += h
    # Point's y
    changed_points_y = copy.deepcopy(points)
    changed_points_y[i][1] += h
    return [(distance_sum(changed_points_x) - current_sum) / h,
(distance_sum(changed_points_y) - current_sum) / h]
```

Šį metodą vykdo darbuotojai. Grąžina paskaičiuota gradiento reiškmę taške.

```
pool = Pool(processes=processes)
```

Šis kodas sukuria procesų pool. Kiekis priklauso nuo globalaus kintamojo processses.

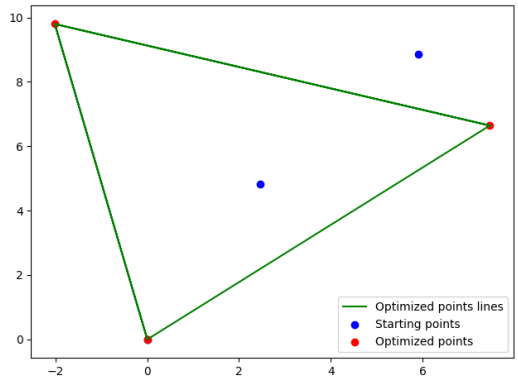
2. Testavimas ir programos instaliavimo bei vykdymo instrukcija

2.1. Testavimas

Testavimas bus atliekamas keičiant globalų taškų skaičiaus kintamąjį n. Galimas taškų optimizavimas labai priklauso nuo pasirinkto taškų generavimo seed.

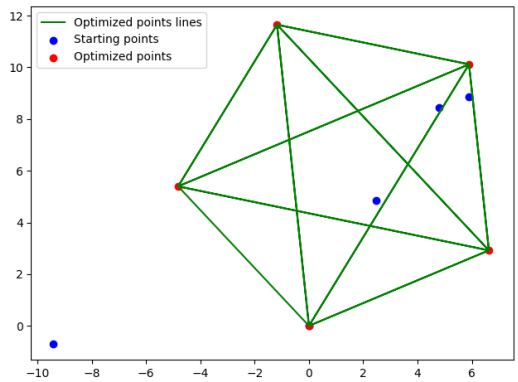
Skaičiavimams naudojamas 1 procesas

n = **2**
Primary strings lengths average: **14.509528245867159**
Optimization price: **1.2396533142351218e-05**
Iterations: **97**
Calculated in: **0.06653594970703125s**



pav. 1: Optimizavimo rezultatas kai n=2

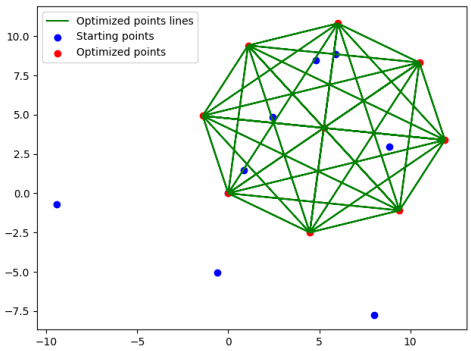
n=**4**
Primary strings lengths average: **55.24274483022175**
Optimization price: **10.557287898603931**
Iterations: **157**
Calculated in: **0.20158982276916504s**



pav. 2: Optimizavimo rezultatas kai n=4

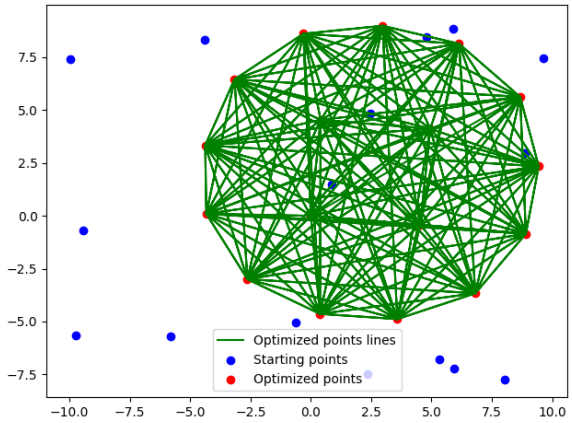
n=**8**
Primary strings lengths average: **86.28127016939679**
Optimization price: **41.19682745682422**
Iterations: **275**

Calculated in: **0.9847619533538818s**



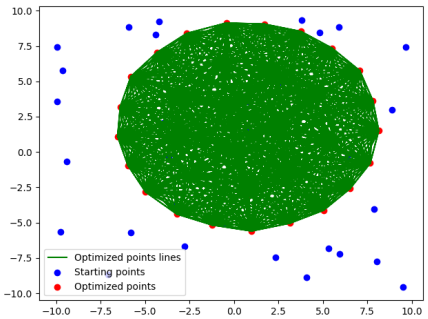
pav. 3: Optimizavimo rezultatas kai n=8

n=**16**
Primary strings lengths average: **249.25448123470233**
Optimization price: **108.88306848576966**
Iterations: **1001**
Calculated in: **18.299113273620605s**



pav. 4: Optimizavimo rezultatas kai n=16

n=**32**
Primary strings lengths average: **494.31034555920826**
Optimization price: **248.50087536719266**
Iterations: **3125**
Calculated in: **318.3662362098694s**



pav. 5: Optimizavimo rezultatas kai n=32

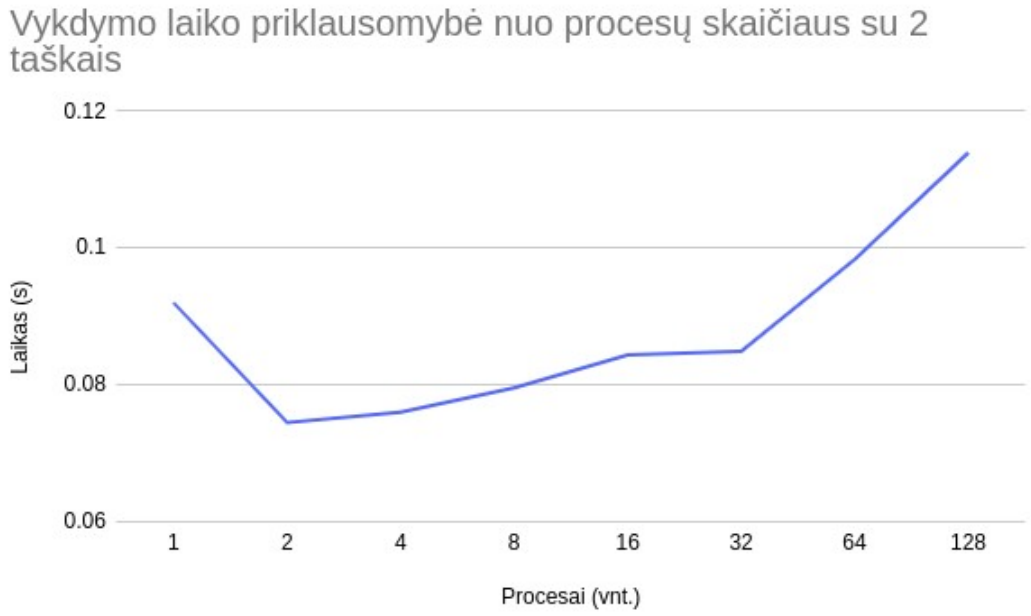
Iš šių testų matome, kad pavyko optimizuoti kiekvienu atveju. Tai parodo gauti grafikai bei gautos optimizavimo kainos.

2.2. Programos instaliavimo bei vykdymo instrukcija

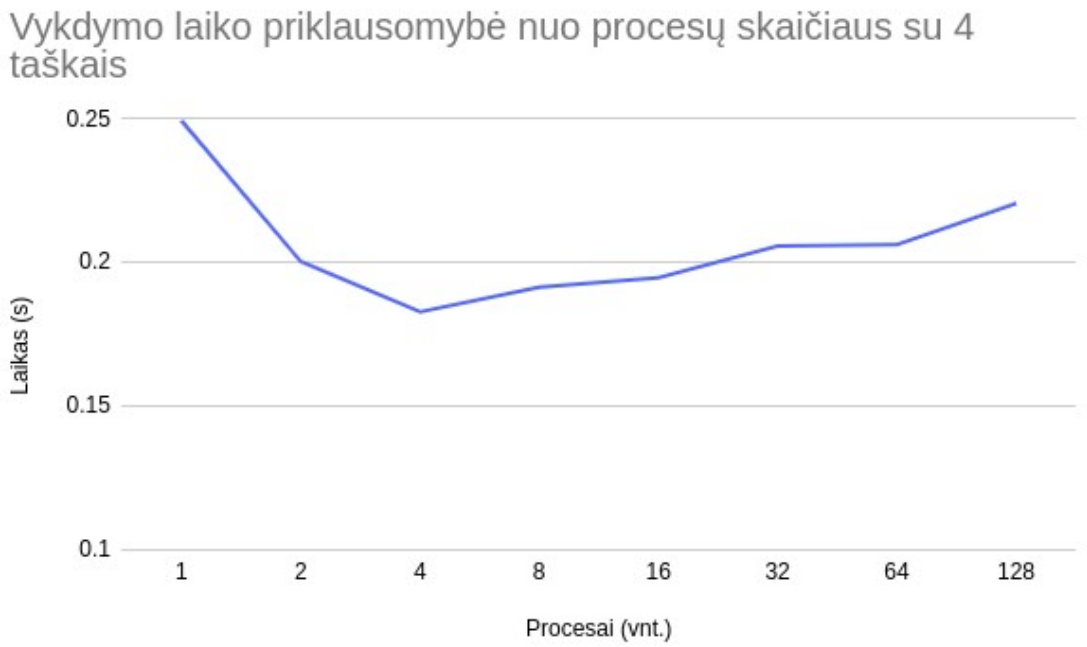
Norint pasileisti šią programą reikia parsisiųsti optimization-task.py failą iš moodle arba mano github repositorijos (https://github.com/Justas-Milisiunas/Lygiagretus-Programavimas/tree/develop/Projektas). Jeigu naudojama Linux OS, reikia susiinstaliuoti python3 komanda **sudo apt install python3**. Tada reikia susiinstaliuoti matplotlib biblioteką su komanda: **pip3 install matplotlib**. Tada optimization-task.py katalogo vietoje atsidarius terminalą galima paleisti su komanda **python3 optimization-task.py**.

3. Vykdymo laiko kitimo tyrimas

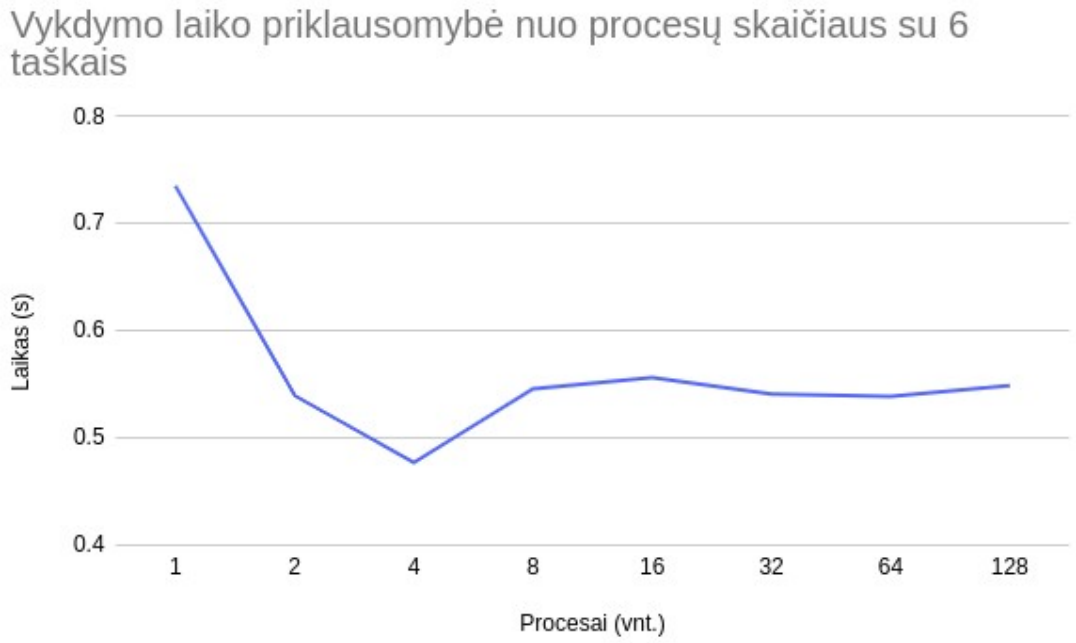
Vykdymo laiko kitimo tyrimui bus naudojami šie tašku skaičiai: 2, 4, 8, 10, 15, 20. Kiekvienam iš šių takų skaičių bus vykdomas optimizavimas su kiekvienu procesų skaičiumi. Procesai: 1, 2, 4, 8, 16, 32, 64, 128. Rezultatai bus vaizduojami vykdymo laiko priklausomybe nuo procesų skaičiaus. Grafiko x ašis rodys procesų skaičių, y ašis – vykdymo laiką.



pav. 6: Su 2 taškais

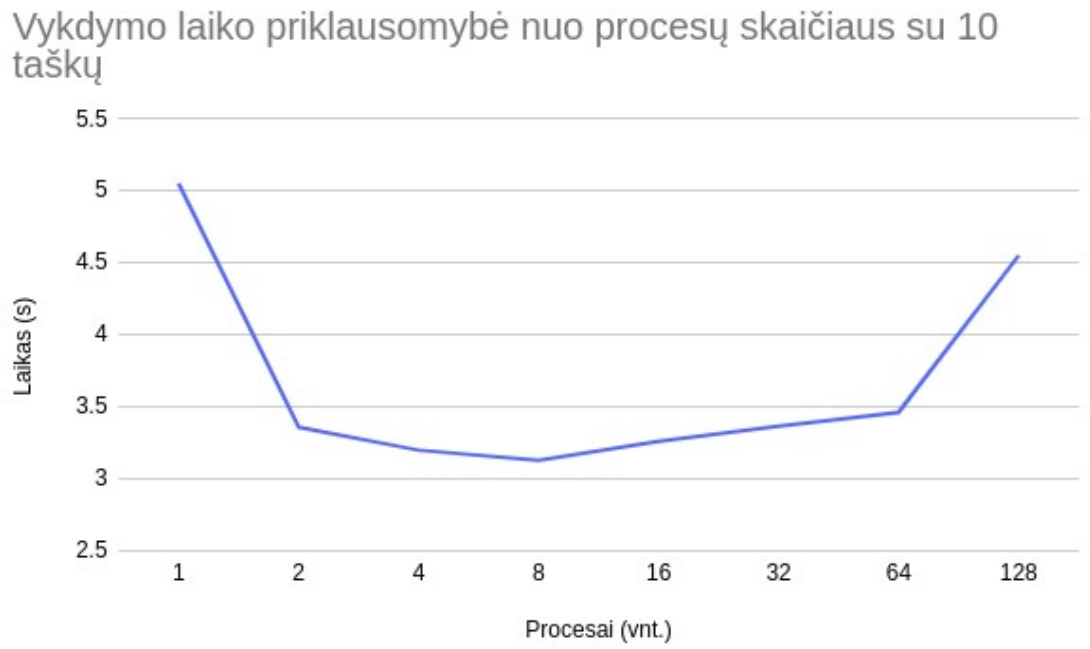


pav. 7: Su 4 taškais

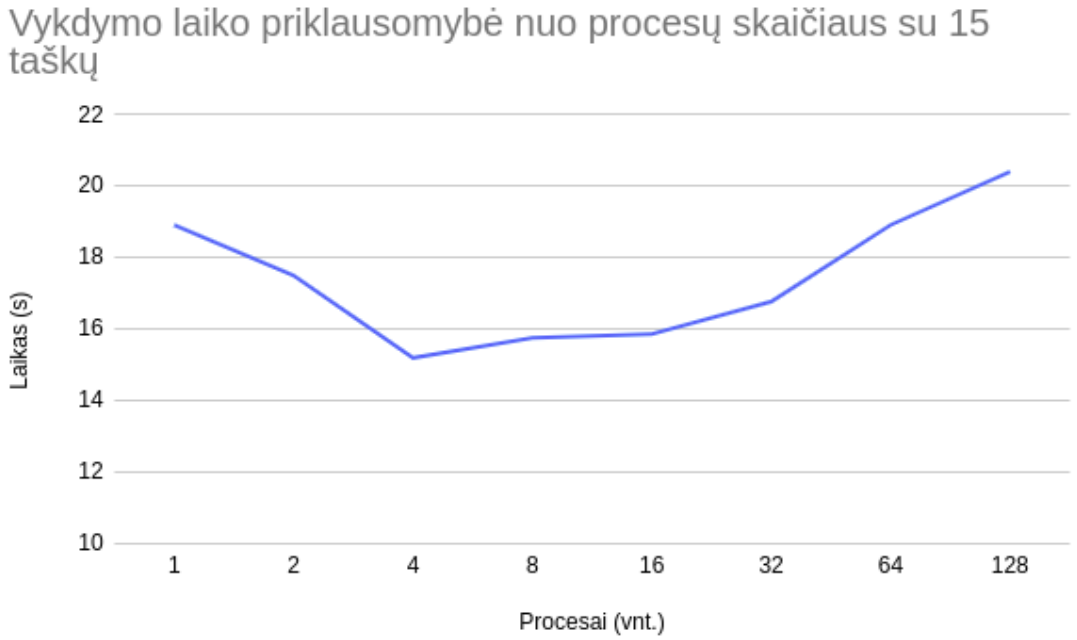


pav. 8: Su 6 taškais

pav. 9: Su 8 taškais



pav. 10: Su 10 taškų



pav. 11: Su 15 taškų

pav. 12: Su 20 taškų

Išvados

Individualaus projekto metu buvo lygiagretinama skaitinių algoritmų modulio 2 laboratorinio darbo optimizavimo uždavinio programa. Tikslas buvo pagreitinti programos darbą. Ši programa daugiausia laiko užtrunka skaičiuodami kiekvieno taško gradientą kiekvienoje iteracijoje. Ši vieta ir buvo sulygiagretinama.

Su lygiagretinus, programas pagreitėjo nuo 1.2 iki 2 kartų. Iš tyrimo grafikų matome:

- Labiausia skiriasi vykdymo laikas tarp 1 ir 2 procesų. Su 20 taškų skirtumas siekia net 75 sekundes
- Programa greičiausiai veikia su 4 arba 8 procesais.
- Naudojant daugiau nei 8 procesus programa pradeda letėti. Tai vyksta todėl, kad kompiuteris vienu metu gali vykdyti tik 8 procesus.

Literatūra

1. https://moodle.ktu.edu/pluginfile.php/45753/mod_resource/content/5/13-python.pdf
2. https://docs.python.org/3/