Lygiagretaus programavimo laboratorinių darbų analizė

Pijus Petkevičius

October 23, 2022

Contents

1. 1 laboratorinis darbas

1.1. Aprašymas

1.1.1. Įžanga

Aibę A sudaro geografiniai taškai, nurodant platumos ir ilgumos koordinates. Iš šios aibės reikia parinkti taškų aibę X tokią, kad atstumų nuo kiekvieno aibės A taško iki jam artimiausio aibės X taško suma būtų minimali $X \subset A$.

Faile lab_data.dat pateikti 50000 geografinių taškų, kur viena eilutė aprašo vieno geografinio taško koordinates.

Faile lab_01_2_algorithm.cpp pateikta programa, kuri randa nurodyto n taškų aibę X, atitinkančią uždavinio sąlygą, naudojant paprastosios atsitiktinės paieškos (angl. Pure Random Search, PRS) algoritmą.
Pagrindiniai algoritmo parametrai (globalūs kintamieji):

- num_points: duomenų aibės A dydis (max 50000)
- num_variables: ieškomos taškų aibės X dydis
- num_iterations: sprendinio paieškai skirtų iteracijų skaičius (kuo daugiau, tuo didesnė tikimybė rasti geresnį sprendinį).

Algoritmų vykdymo pradžioje sudaroma atstumų matrica, kurioje saugomi atstumai kilometrais tarp taškų, suskaičiuoti pagal Haversino formulę. Atsižvelgiant į tai, kad atstumas nuo taško a iki taško b yra lygus atstumui nuo taško b iki taško a, yra užpildoma tik pusė matricos. Šioje matricoje saugomi atstumai yra naudojami vykdant aibės X taškų paiešką.

1.1.2. Užduotis

1. Pasirinkti duomenų aibės dydį ir algoritmo iteracijų skaičių, kad atstumų matricos skaičiavimas užtruktų ne mažiau 10 sekundžių, o sprendinio paieškos laikas būtų nemažesnis nei 20 sekundžių.
2. Duomenų įkėlimą ir atstumų matricos skaičiavimą laikyti nuosekliąja algoritmo dalimi, o sprendinio paiešką - lygiagretinama dalimi, įvertinti teorinius galimus algoritmo pagreitėjimus naudojant 2 ir 4 procesorius, bei didžiausią galimą pagreitėjimą.
3. Duomenų įkėlimą ir atstumų matricos skaičiavimą laikyti nuosekliąja algoritmo dalimi, sudarykite lygiagretųjį bendros atminties algoritmą ir eksperimentiniu būdu ištirkite jo pagreitėjimą naudodami 2 ir 4 procesorius.
4. Sudarykite lygiagretų bendros atminties algoritmą atstumų matricos skaičiavimui ir eksperimentiniu būdu ištirkite jo pagreitėjimą naudodami 2 ir 4 procesorius.
5. Ištirti algoritmo pagreitėjimo priklausomybes nuo procesorių skaičiaus, kai matricos skaičiavimas ir sprendinio paieška išlygiagretinti.

1.2. Kompiuterinės įrangos ir parametrų pasirinkimas

Algoritmo analizei buvo naudojama **Apple Mac Mini Desktop Computer, 3.2GHz 6-Core Intel Core i7** kompiuteris, kurio dėka, buvo galima paleisti ant 2, 4 ir 6 procesorių. Kad įgyvendinti **1** nurodymą, buvo pasirinkta:

- num_points = 12000
- num_iterations = 30000

Duomenų nuskaitymas (s)	Atstumų matricos skaičiavimas (s)	PRS skaičiavimas (s)
0.003	10.312	19.955
0.004	10.315	19.993
0.003	10.321	19.967

Lentelė 1: Algoritmo skaičiavimo dalių rezultatai, naudojant **Mac Mini** kompiuterį

1.3. Teoriniai įverčiai

Paleidus programą 3 kartus, gauti skaičiavimo dalių rezultatai:

Duomenų nuskaitymas (s)	Atstumų matricos skaičiavimas (s)	PRS skaičiavimas (s)
0.004	10.316	19.972

Lentelė 2: Nuoseklaus algoritmo skačiavimo dalių rezultatai

Pagal **2** nurodymą, nuosekliąja dalimi (α) laikoma duomenų nuskaitymas ir atstumų matricos skaičiavimas, o lygiagrečiąja (β)- PRS skaičiavimas.

$$\alpha = \frac{\text{nuoseklioji dalis}}{\text{visas laikas}}$$

$$\beta = \frac{\text{lygiagrečioji dalis}}{\text{visas laikas}}$$

Gauname kad:

$$\alpha = 0.341$$

$$\beta = 0.659$$

Teorinis pagreitėjimas naudojant p procesorių:

$$S_p = \frac{1}{\alpha + \frac{\beta}{p}}$$

$$S_2 = \frac{1}{0.341 + \frac{0.659}{2}} = 1.492$$

$$S_4 = \frac{1}{0.341 + \frac{0.659}{4}} = 1.978$$

Teorinis maksimumas pagal Andalo Dėsnį:

$$S_{max} = \lim_{p \to \infty} \frac{1}{\alpha + \frac{\beta}{p}} = \frac{1}{\alpha} = \frac{1}{0.341} = 2.935$$

1.4. Pure random search (PRS)

Bandymas išlygiagretinti PRS algoritmą, buvo atliktas 2 būdais.

1 algoritme buvo pasitelkta **Dynamic** scheduling strategija, ir reduction min: f_best_solution. Kievieną karta, kai randamas geresnė sprendinio reikšmė, ji priskiriama f_best_solution kintamajam(jis automatiškai pasiima tik mažesnę reikšmę). Vėliau viskas geriausio sprendinio reikšmės buvo išsaugojamos naudojant critical žymę:

```
int *best_solution;
double f_solution, f_best_solution;
#pragma omp parallel reduction (min: f_best_solution ) private (f_solution)
#pragma omp for schedule(dynamic)
for (int i=0; i<num_iterations; i++) {
    // random find and evaluate solution
    f_solution = evaluate_solution(solution);
    if (f_solution < f_best_solution) {
        f_best_solution = f_solution;
        if(f_best_solution == f_solution){
            #pragma omp critical (DataCollection)
            {
               // copy solution values to the best_solution array
            }
        }
    }
}
```

Pseudokodas 1: PRS pirmas algoritmas

2 antrajame algoritme veikimo principas gana panašus, tik ciklas paskirstomas keliems branduoliams, randamas lokali geriausia f_best_solution_tmp reikšmė ir po ciklo ji priskiriama f_best_solution ir išsaugojamos geriausio sprendinio reikšmės:

```
double f_best_solution;
int *best_solution;
#pragma omp parallel reduction(min: f_best_solution)
{
    int *best_solution_tmp;
    double f_solution, f_best_solution_tmp;
    #pragma omp for schedule(dynamic)
    for (int i=0; i<num_iterations; i++) {
        // random find and evaluate solution
        f_solution = evaluate_solution(solution);
        if (f_solution < f_best_solution_tmp) {
            f_best_solution_tmp = f_solution;
           // copy solution values to the best_solution_tmp array
        }
    }
    f_best_solution = f_best_solution_tmp;
    #pragma omp barrier
    if(f_best_solution == f_best_solution_tmp){
        // copy best_solution_tmp values to the best_solution array
    }
}
```

Pseudokodas 2: PRS antras algoritmas

Abu algoritmai buvo ištestuoti su 2, 4 ir 6 branduoliais ir rezultatai matomi **1** diagramoje:

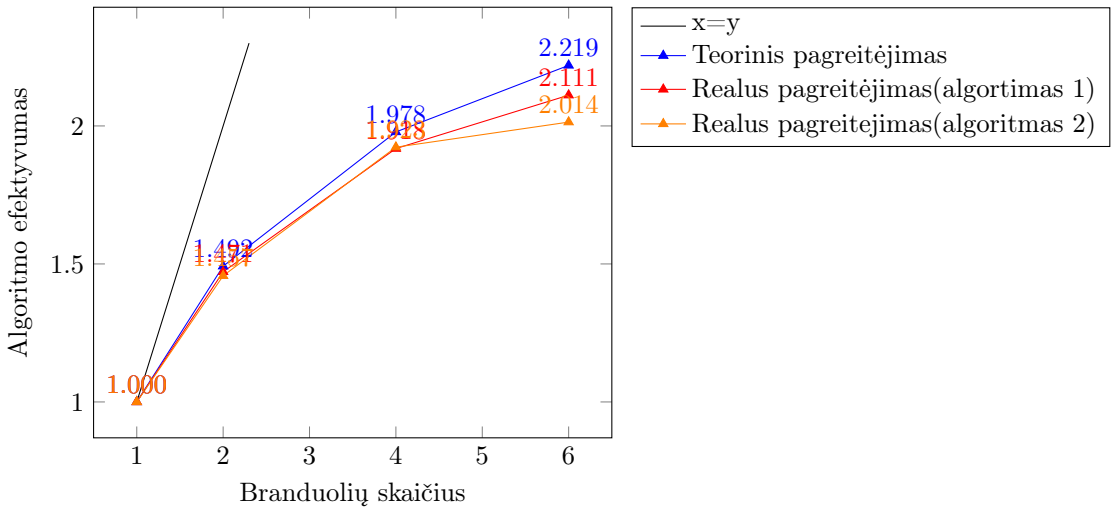

Diagrama 1: PRS algoritmo lygiagretinimo diagrama

Pastebime, kad **1** ir **2** algoritmai su 2 ir 4 branduoliais turėjo ganėtinai panašų efektyvumą:

Branduolių skaicius	1 algoritmas	2 algoritmas
2	1.471	1.457
4	1.918	1.922
6	2.111	2.014

Lentelė 3: Algoritmų efektuvumo priklausomybė nuo branduolių skaičiaus

Tačiau, kai branduolių skaičius pasiekia 6, 2-sis algoritmas nusileidžia efektyvumu 1-jam. **1-jį** algoritmą naudosime tolimesniuose 1 užduoties eksperimentuose.

1.5. Atstumų matricos skaičiavimas

```
#pragma omp parallel for schedule(dynamic)
for (int i=0; i<num_points; i++) {
    ...
    for (int j=0; j<=i; j++) {
        distance_matrix[i][j] = Haversine_distance(...);
    }
}
```

Pseudokodas 3: Atstumų matricos skaičiavimas

3 algoritmas, eksperimentiniu būdu buvo išbandytos įvairios lygiagretinimo strategijos (**Dynamic**,**Guided**, **Static**). Gauti rezultatai matomi **2** diagramoje:

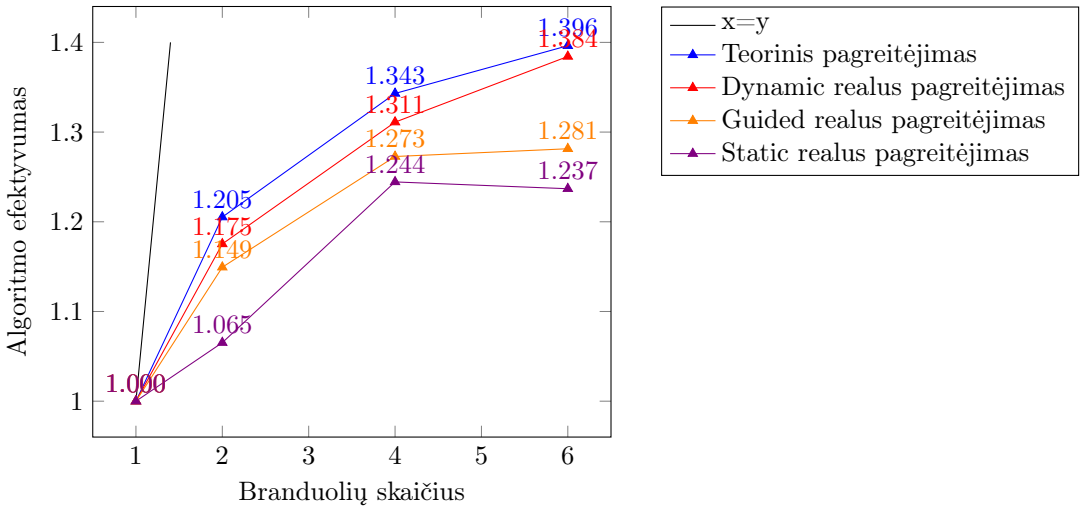

Diagrama 2: Įvairių strategijų efektyvumo diagrama

Pastebime, jog **Dynamic** yra kur kas efektyvesnis, lyginant su **Static** ir **Guided** lygiagretinimo strategijomis ir kur kas labiau priartėja prie teorinio pagreitėjimo.

Dynamic direktyvą naudosime tolimesniuose 1 užduoties eksperimentuose.

1.6. Atstumų matricos ir PRS dalių lygiagretinimo rezultatai

Iš ankstesnių eksperimentų radome, jog **1** PRS algoritmas ir **Dynamic** direktyva atstumų matricai skaičiuoti buvo efektyviausi sprendimo būdai. Eksperimentiniu būdu ištestavus progamą, buvo gauti tokie rezultatai:

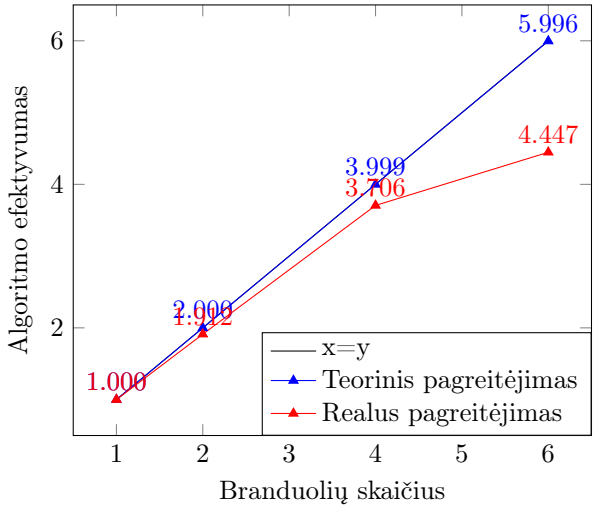

Diagrama 3: Efektyvumo diagrama, kai PRS ir matricos skaičiavimas lygiagretinami