Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti: Popović Luka, SV04/2021
Popović Matija, SV05/2021

Predmet: Paralelno programiranje

Tema projektnog zadatka: Paralelizam genetskog algoritama, problem pravljenja rasporeda časova

Analiza problema

Problem pravljenja rasporeda časova predstavlja problem koji pravi raspored časova za fakultet od različitih predmeta sa određenim parametrima problema. Kako fakultet ima određeni kapacitet učionica i s obzirom da ima radno vreme, očigledno se mora raspored napraviti pod tim uslovima. Neki od parametara su određen broj učionica, radnih dana, radno vreme (koje počinje minimalno u 7:00, a treba da je završeno do 19:00). Između svaka 2 predavanja mora da bude pauza od minimalno 15 minuta. Kriterijum optimalnosti po kome se formira problem je takav da prosečno vreme početka nastave što kasnije, a prosečno vreme završetna nastave bude što ranije (a da pritom budu zadovoljeni i prethodni uslovi). Matematički formalno treba da maksimizujemo f(p,k)= $\sum_1^{n*m} p_i * k_i$, gde p_i i k_i , predstavljalju redom, vreme proteklo od 7:00 do početka prvog časa i vreme proteklo od kraja poslednjeg časa do 19:00.

Koncept rešenja

Za rešavanje ovog problema koristićemo genetski algoritam, koji predstavlja simulaciju biološke evoluciju (proces prirodne selekcije). Genetski algoritam se sastoji od formiranja početne populacije, selekcije, ukrštanja i mutacija.

Implementaciju smo započeli učitavanjem svih stvari datih iz fajla „data_tabletime.txt“ iz kojeg smo izvukli podatke o broju učionica i sve moguće predmete koji treba da se održe na fakultetu.

Populacija je predstavljena nizom jedinki. Jedinka predstavlja 1 moguć raspored. Pošto se problem maksimizuje, onda je logično da nastava bude vezana u kontinuitetu (jer ako su pauze veće od 15 minuta) onda se smanjuje p_i, odnosno k_i. Jedinka se popunjava nabacanim nastavama po učionicama i vremenu (pri čemu je pamćeno samo vreme kretanja te nastave, podrazumevajući da je pauza između svake 2 nastave 15 minuta iz prethodnog razloga), vodeći računa o radnom vremenu.

Ocena predstavlja brojnu vrednost, koliko je jedno stanje bolje od drugog za određeni problem. U ovom slučaju je maksimizacija problema ocena kvaliteta rešenja.

Generacija predstavlja broj iteracija datog problema.

Selekcija predstavlja čuvanje i prenošenje dobrih osobina populacije (dobrog genetskog materijala) na sledeću generaciju. Ima više vrsta selekcije, među kojima se najčešće koriste: prirodna, ruletska, ruletska sa rangiranjem i turnirska selekcija.

Mi smo odabrali ruletsku selekciju.

Ruletska selekcija predstavlja biranje 2 najbolja rangirana roditelja koristeći proizvod (a*b), gde je a ocena (fitness, prilagođenost), koja je prethodna izračunata, a b random broj iz intervala (0,1).

Mutacija predstavlja minimalnu promenu jedinke na određeni način. U našem slučaju smo nekad u nekom rasporedu (u 1 učionici za 1 dan) menjali vreme početka nastave, proveravajući da li ispunjavaju uslove prethodnih uslova.

Elitizam predstavlja proces, pri kome se pri kreiranju nove generacije, prenese određeni broj “najboljih” (onih sa najvećom ocenom (fitness-om, prilagođenosti)) jedinki na sledeću generaciju.

Ona nam osigurava da se kvalitet rešenja neće smanjivati iz generacije u generaciju. Mi smo za elitizam uzeli broj 4 (elitizam.size), jer za velike vrednosti brzo dođe do lokalnog maksimuma, što nama nije poenta (nego globalni maksimum).

Programsko Rešenje

Za implementaciju programa je potreban jedan txt fajl gde će prvi red predstavljati učionice, a ostali redovi predmeti koji treba da se održe i koliko traju časovi.

Funkcije koje su korišćene su sledeće:

```
void optimization(vector<vector<string>>& bestInduvidual, vector<vector<int>>& bestStartsClass);
pair<vector<vector<vector<string>>>,vector<vector<vector<int>>>> createPopulation(vector<vector<vector<string>>>population,
    vector<vector<vector<int>>>startsClassAll, int size);
pair<vector<vector<vector<string>>>, vector<vector<vector<int>>>> parallel_createPopulation(vector<vector<vector<string>>>population,
    vector<vector<vector<int>>>startsClassAll, int parentsSize);
void rate(vector<vector<vector<string>>>& population,vector<vector<vector<int>>>& startClassAll, vector<int>& rates);
void rulleteSelection(vector<vector<vector<string>>>& population, vector<vector<vector<int>>>& startClassAll, vector<int>& rates,
    vector<vector<vector<string>>>& selection);
void mutation(vector<vector<vector<string>>>& population);
```

1. Funkcija optimization služi za celokupno spajanje svih funkcija
2. Funkcija createPopulation kreira populaciju pocetnog algoritma i vraća dužine vremena kada je najbolje početi taj dan, dok parentsSize služi zbog deljenja u paralelizaciji
3. Funkcija parallel_createPopulation predstavlja paralelnu verziju funkcije createPopulation koja će biti opisana kasnije
4. Funkcija rate služi za izračunavanje ocena(fitnessa) populacije
5. Funkcija rulleteSelection služi za biranje druge(bolje) populacije
6. Funkcija mutation služi za male promene pojedinjenih jedinki, čisto da se ne bi često ponavljale jedne te iste jedinke

Sada ćemo predstaviti razliku izmedju paralelnog i serijskog rešenja.

Paralelizam je uočen u funkciji createPopulation. Prilikom kreiranja populacije, moguće je vektor populacije podeliti na manje celine (vektore), nad kojima se vrši obrada. Ovo predstavlja “Podeli pa zavladaj” algoritam, jer se deli na manje potprobleme (smanjivanjem dimenzije). Kada problem postane dovoljno mali (CUTOFF (prekidna tačka)) se poziva serijska implementacija te funkcije, jer tada postaje više isplativo serijsko rešenje od paralelnog rešenja.

```cpp
std::pair<std::vector<std::vector<std::vector<std::string>>>, std::vector<std::vector<std::vector<int>>>> GeneticAlgorithm::createPopulation
(std::vector<std::vector<std::vector<std::string>>>population, std::vector<std::vector<std::vector<int>>>startsClassAll,int size) {
    for (int i = 0; i < size; i++) {
        vector < vector<string>> vec;
        vector < vector<int>> vec1;
        population.push_back(vec);
        startsClassAll.push_back(vec1);
    }
    int duzina = minimumNumber(population.size(), startsClassAll.size(), 100000);
    for (int k = 0; k < popularizationSize; k++) {
        int i = 0;
        while (i < classes.size()) {
            i++;
            std::random_device rd;
            std::mt19937 generator(rd());
            std::uniform_int_distribution<int> distribution(0, duzina-1);
            int randomBroj = distribution(generator);
            if (population[randomBroj].size() >= 0) {
                vector <string> vec;
                vector <int> vec1;
                vec.push_back(classes[i - 1]);
                vec1.push_back(duration[i - 1]);
                population[randomBroj].push_back(vec);
                startsClassAll[randomBroj].push_back(vec1);
            }
        }
    }
    return make_pair(population, startsClassAll);
}
```

Serijski deo

```cpp
std::pair<std::vector<std::vector<std::vector<std::string>>>, std::vector<std::vector<std::vector<int>>>> GeneticAlgorithm::parallel_createPopulation
(std::vector<std::vector<std::vector<std::string>>>population, std::vector<std::vector<std::vector<int>>>startsClassAll,int parentsSize) {
    task_group g;
    std::pair<std::vector<std::vector<std::vector<std::string>>>, std::vector<std::vector<std::vector<int>>>> children;
    std::pair<std::vector<std::vector<std::vector<std::string>>>, std::vector<std::vector<std::vector<int>>>> children1;
    std::pair<std::vector<std::vector<std::vector<std::string>>>, std::vector<std::vector<std::vector<int>>>> children2;
    std::pair<std::vector<std::vector<std::vector<std::string>>>, std::vector<std::vector<std::vector<int>>>> children3;
    std::pair<std::vector<std::vector<std::vector<std::string>>>, std::vector<std::vector<std::vector<int>>>> children4;

    if (parentsSize <= GeneticAlgorithm::CUTOFF)
    {
        children = createPopulation(population,startsClassAll, parentsSize);
    }
    else
    {
        int size = parentsSize / 4;

        g.run([&] {children1 = parallel_createPopulation(population,startsClassAll, size); });
        g.run([&] {children2 = parallel_createPopulation(population,startsClassAll, size); });
        g.run([&] {children3 = parallel_createPopulation(population,startsClassAll, size); });
        g.run([&] {children4 = parallel_createPopulation(population,startsClassAll, size); });

        g.wait();
    }
    children.first.insert(children.first.end(), children1.first.begin(), children1.first.end());
    children.second.insert(children.second.end(), children1.second.begin(), children1.second.end());
    children.first.insert(children.first.end(), children2.first.begin(), children2.first.end());
    children.second.insert(children.second.end(), children2.second.begin(), children2.second.end());
    children.first.insert(children.first.end(), children3.first.begin(), children3.first.end());
    children.second.insert(children.second.end(), children3.second.begin(), children3.second.end());
    children.first.insert(children.first.end(), children4.first.begin(), children4.first.end());
    children.second.insert(children.second.end(), children4.second.begin(), children4.second.end());

    return children;
}
```

Paralelni deo

Rezultati testiranja

Merenja su vršena na procesoru AMD Ryzen 7 4800H.

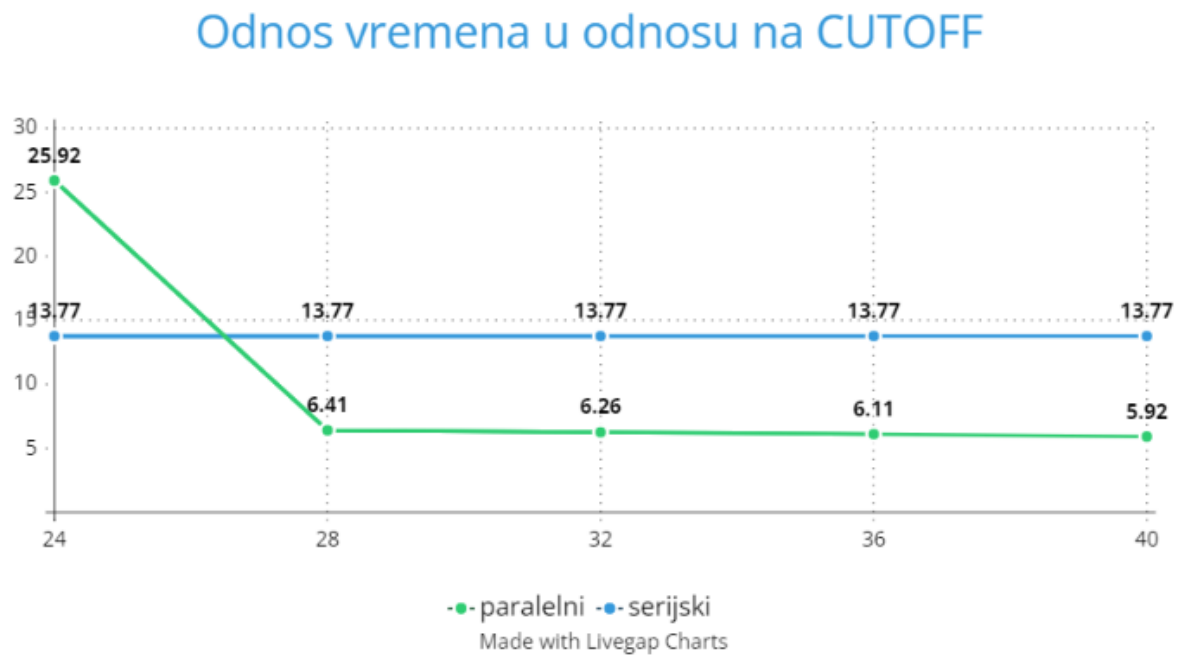
Napomena: Genetski algoritam nije ekzatan algoritam (za iste ulaze, skoro uvek daje različite rezultate). Matematički formalno, genetski algoritam nije injektivna funkcija. Genetski algoritam predstavlja stohastički sistem, te rezultati mogu varirati zbog nasumičnog izbora roditelja (u suštini se tu koristila ruletska selekcija, ali u pozadini i ona je zasnovana na random-izaciji) i mutacija.

Rezultati testiranja variraju od: data_tabletime.txt (zavisnost broj ucionica i broj casova) i parametara populationSize, elitism, generation, mutationRate kod serijskih, i dodatnog paramatra CUTOFF kod paralelnog dela.

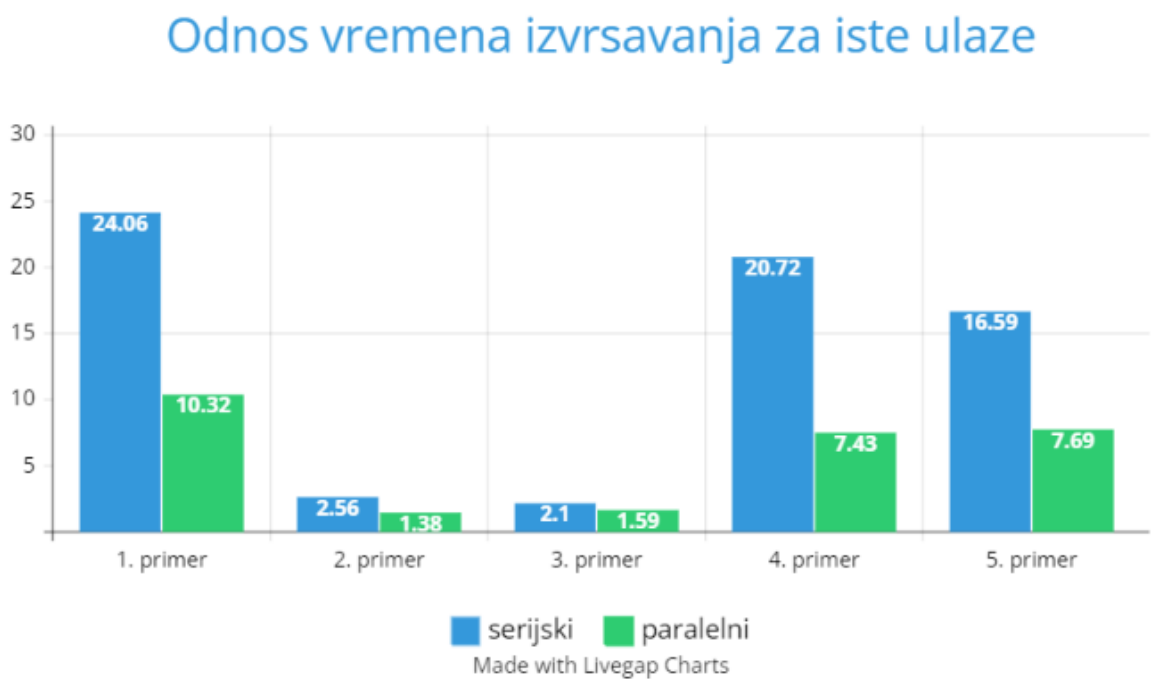
Rezultate genetskog algoritma nema potrebe prikazivati, jer kao što je već rečeno, svaki put daje drugačije validno rešenje, te ćemo se najviše u ovom delu pozabaviti odnosu između serijskih i paralelnih rezultata, te podešavanju parametara.

Odnos između serijskog i paralelnog (za isti data_tabletime.txt, fiksne parametre popularizationSize,elitism, generation, mutationRate) zavisi od parametra CUTOFF.

Sledeći grafik predstavlja odnos između serijskog i paralelnog za različite vrednosti CUTOFF-a.



Sledeći grafici će predstavljati odnos između serijskog i paralelnog za fiksnu vrednost CUTTOFF-a i različite vrednosti ostalih promenljivih.



1. Primer predstavljaju vremena serijskog i paralelnog algoritma za parametre: popularizationSize=1000, generation=5, elitism=4, CUTOFF=32
2. Primer predstavljaju vremena serijskog i paralelnog algoritma za parametre: popularizationSize=300, generation=3, elitism=10, CUTOFF=50
3. Primer predstavljaju vremena serijskog i paralelnog algoritma za parametre: popularizationSize=500, generation=2, elitism=8, CUTOFF=30
4. Primer predstavljaju vremena serijskog i paralelnog algoritma za parametre: popularizationSize=400, generation=6, elitism=5, CUTOFF=100
5. Primer predstavljaju vremena serijskog i paralelnog algoritma za parametre: popularizationSize=1200, generation=4, elitism=10, CUTOFF=36

Analiza rezultata

Na osnovu rezultata testiranja koje smo prikazali u odeljku Rezultati testiranja, primećujemo da rezultat genetskog algoritma značajno zavisi od ulaznih parametara. Iako je genetski algoritam po prirodi sporiji uopšteno, posebno kada se suočava sa velikim ulazima, imamo zadovoljstvo da vidimo značajno ubrzanje u našim eksperimentima, gotovo trostruko. Uz dalje povećanje ulaza, očekuje se da će to ubrzanje nastaviti da raste. Takođe, primetili smo da je parametar CUTOFF značajan i ima snažan uticaj na rezultate.

Paralelizacija je odigrala važnu ulogu, posebno kod većih ulaza, gde bi ubrzanje bilo još izraženije. Ovo ukazuje na uspešnost primene paralelizacije u našem algoritmu. Nadalje, moguće je dalje poboljšanje brzine genetskog algoritma kroz dublje proučavanje problema i identifikaciju drugih kritičnih tačaka.

Cilj nam je da izrazimo pozitivne aspekte i perspektive paralelizma programa i genetskog algoritma, naglašavajući istovremeno moguće oblasti za dalje istraživanje i optimizaciju.