SADRŽAJ

Analiza izvršne specifikacije

Host platforma izvršne specifikacije je laptop ASUS (ASUSTeK COMPUTER INC. VivoBook_ASUSLaptop X515EA_X515EA) sa procesorom 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz × 8, sa 8 GB RAM memorije i 512.1 GB Disk prostora. Njegove mogućnosti ne mogu da se mere sa mogućnostima target platforme Zybo Z7 10 i ZYNQ 7000 sistema na čipu. Zbog toga trenutne rezultate analize treba tumačiti kao okvirne linije vodilje ka procesu particionisanja [1].

Izvršna specifikacija je softverski model koji pomoću tri jednostruko spregnute liste modeluje Dajkstrin algorintam na način opisan u Specifikaciji. Nad binarnom datotekom `main` pokreće se `valgrind`-ov alat `callgrind` [2]. `Callgrind` daje uvid u ukupan broj izvršenih instrukcija, kao i funkcije u modelu koje zazimaju najveći udeo u broju izvršenih instrukcija (engl. hotspots).

Funkcija	Procenat u ukupnom broju instrukcija [%]
`relaxationProcess`	63.15
`std::vector<unsigned int, std::allocator<unsigned int>>:: operator[] (unsigned long)`	30.01
`popVertexFromTable`	6.57

Tabela 1 `Callgrind` rezutati

Ukupno je izvršeno 6,432,213,469 instrukcija. Od ukupnog broja izvršenih instrukcija 63.15% zauzima proces relaksacije ivica grafa, koji se nalazi uokviru funkcije `relaxationProcess`. Indeksiranje elemenata u vektorima koji skladište informacije o mreži grada: `graph_latitudes`, `graph_longitudes`, `head`, `tail` i `edge_weight`, `std::vector::operator[]`, zauzima 30.01% instrukcija, dok 6.57% predstavalja proces izbacivanja temena iz tabele neposećenih temena, `popVertexFromTable`. Kako `std::vector::operator[]` zauzima skoro trećinu ukupnog broja instrukcija, zaključuje se da značajan deo izvršavanja modela predstavlja pristup elementima vektora. Prelazak na statičke nizove može eliminisati cenu pristupa elementima vektora. Statička alokacija je takođe i brža od dinamičke.

`Callgrind` pokazuje da je glavni kandidat za hardversku implementaciju funkcija `relaxationProcess` (tabela 1). Problem je što se trenutna implementacija procesa relaksacija ivica bazira na korišćenju jednostruko spregnute liste koja reprezentuje tabelu neposećenih temena grafa. Pri radu sa jednostruko spregnutim listama često se pristupa se nesusednim memorijskim lokacijama. Ovo doprinosi lošem kešingu, s obzirom na to da ovakav način obilaska memorijskih lokacija loše koristi osobinu prostorne lokalnosti. Svaki element liste sadrži pokazivač na sledeći, što nepotrebno troši memorijske resurse. Kao i kod vektora, prelazak na statički niz je opravdan.

Softverski model pred particionisanje

Prepravljen softverski model tabelu neposećenih temena i vektore podataka bitnih za reprezentaciju grafa modeluje kao statičke nizove.

```
Mreza Novog Sada se sastoji iz 9122 temena i 20043 ivice

...DONE!
-------------
Srecan put :)
-------------
Table deleted
Relaxation history deleted
Path deleted
```

Konzolni ispis 1 Konzolni izlaz izvršne specifikacije modifikovan da ukazuje na broj temena i ivica mreže Novog Sada

Pošto je mreža grada nepromenjiva, izvorna datoteka `ocitavanje_grafa.cpp` se svodi samo na podatke o grafu. Podaci se smeštaju u datoteke `graph_latitudes.dat`, `graph_longitudes.dat`, `head.dat`, `tail.dat` i `weight.dat`. `main.cpp` očitava ove podatke putem `load_data` funkcije u nizove `float graph_latitiudes[9122]`, `float graph_longitudes[9122]`, `unsigned head[20043]`, `unsigned tail[20043]` i `unsigned edge_weights[20043]`. Za navigacione uređaje se u praksi često koriste već ekstraktovani podaci koji se na odgovarajući način upišu u memoriju uređaja. Na primer, u auto industriji, radio-navigacioni sistem RNS-315, prilikom instaliranja podataka o grafu koristi SD karticu. Na SD kartici se nalaze svi potrebni podaci o saobraćajnicama određene oblasti. Zbog toga je opravdano preći na softverski model koji u ovom trenutku učitava podatke o mapi u nizove i dalje ih koristi u upitima o optimalnoj putanji.

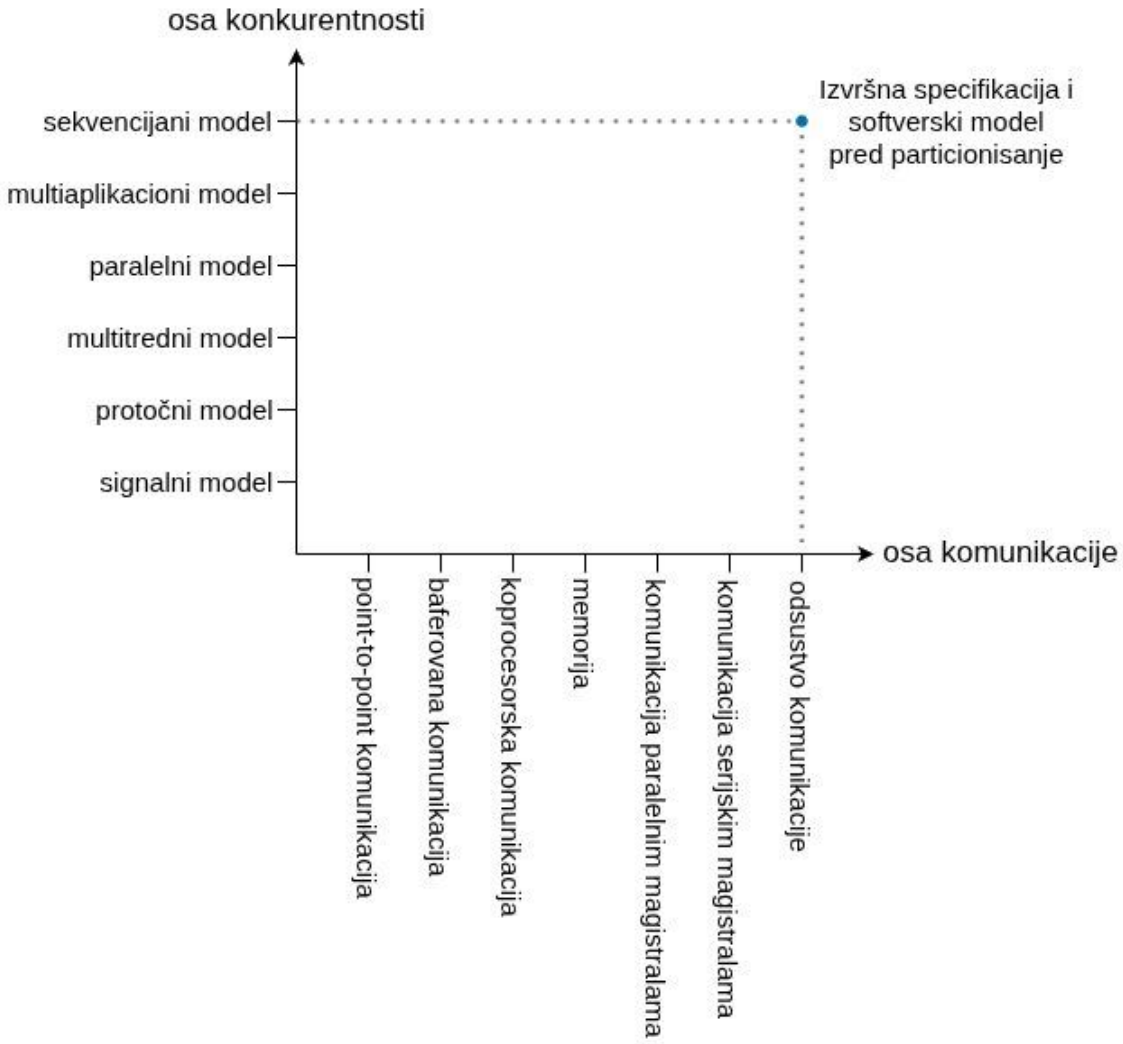
Slika 1 RNS-315 (levo) i odgovarajuća SD kartica sa podacima o saobraćajnicama istočne Evrope (desno)

Tabela neposećenih temena je predstavljena preko dva statička niza: jednog koji predstavlja udaljenost temena od početnog temena, `unsigned cost_from_start_vertex[9122]`, i niza `bool vertex_is_visited[9122]`. Vrednost `vertex_is_visited` elementa je `true` ukoliko je teme na poziciji tog elementa posećeno, a `false` u suprotnom. Jednostruko spregnuta lista koja je reprezentovala ovu tabelu, kao i sve njene pomoćne funkcije (`loadVertexInUnvisitedTable`, `popVertexFromTable`, `deleteTable`), više nisu potrebne.

```
main.cpp:

float graph_latitudes[VERTEX_NUM];
float graph_longitudes[VERTEX_NUM];
unsigned head[ARC_NUM];
unsigned tail[ARC_NUM];
unsigned edge_weights[ARC_NUM];

unsigned cost_from_start_vertex[VERTEX_NUM];
bool vertex_is_visited[VERTEX_NUM];
void load_data();

int main(){ ... }

dijkstra_algorithm.hpp:

#define VERTEX_NUM 9122
#define ARC_NUM 20043

int relaxationProcess(
    RELAXATION **prel_head,
    unsigned *cost_from_start_vertex,
    bool *vertex_is_visited,
    unsigned *head,
    unsigned *tail,
    unsigned *edge_weights
);
```

Kod listing 1 Izmene u softverskom modelu

Softverski model se po osama taksonimije ne razlikuje od izvršne specifikacije opisane u dokumentu Specifikacija.



Slika 2 Softverski model pred partionisanje po osama taksonomije

Analiza softverskog modela pred particionisanje

`Callgrind` ukazuje na značajno smanjene broja instrukcija. Sa 6,432,213,469 prelazi se na 2,365,790,720 instrukcija. Usko grlo modela ostaje `relaxationProcess` funkcija, koja troši 99.36% instrukcija. Najzahtevnije linije funkcije predstavlja prolazak kroz nizove `cost_from_start_vertex[9122]` i `vertex_is_visited[9122]` pri određivanju trenutno najbližeg neposećenog temena (46.62% instrukcija), i proces dolaska do tog temena u nizu `tail[20043]` (47.33% instrukcija), kako bi se za njega ispitalo da li može da relaksira svoje susede.

Bitska analiza

Konzolni ispis 1 prikazuje parametre funkcije `relaxationProcess`. Parametri se u potpunosti mogu mapirati na BRAM memoriju (270 kB), čime im se ubrzava pristup. Jedna BRAM ćelija skladišti 32-bitnu vrednost.

Parametar	Broj bita za predstavljanje jedne vrednosti	Ukupno zauzeće memorije [kB]
`bool vertex_is_visited[9122]`	1	1.1
`unsigned head[20043]`	14	34.3
`unsigned tail[20043]`	14	34.3
`unsigned edge_weights[20043]`	20	48.9
`cost_from_start_vertex[9122]`	31	34.5

Tabela 2 Bitska analiza i ukupno zauzeće memorije za parametre `relaxationProcess` funkcije

Elementi `vertex_is_visited[9122]` niza sadrže informaciju o tome da li je teme grafa posećeno ili ne, stoga je dovoljan samo jedan bit za interpretaciju. `head[20043]` i `tail[20043]` sadrže identifikacione brojeve temena, celobrojne vrednosti od 0 do 9122, za koje je dovoljno 14 bita. `invalid_id`, ID vrednost neispravnog temena, može da se interpretira kao `0b11111111111111`. Maksimalna vrednost težine ivice je 58860, za šta je potrebno 20 bita.

Na početku algoritma, sva temena osim prvog imaju beskonačnu udaljenost od početnog temena, što je implementirano sa `inf_weight` vrednošću koja u softverskom modelu iznosi 2147483647 (pogledati RoutingKit-ovo zaglavlje `constants.h`) [3]. Umesto da se `cost_from_start_vertex[9122]` vrednosti i "beskonačnost" implementiraju pomoću 32 bita, implementiraće se pomoću 31 bita. Za 32-bitnu BRAM ćeliju se ovim dodatno ušteđuje na resursima, jer se u istu 32 bitnu BRAM ćeliju može smestiti i odgovarajuća jednobitna `vertex_is_visited[9122]` vrednost. Ovim je ukupno zauzeće memorije za parametre `relaxationProcess` funkcije 153.1 kB.

Vremenski interval izvršavanja procesa relaksacija može da se izmeri kao razlika dva vremenska trenutka dobijena pomoću funkcije `high_resolution_clock::now()` imenskog prostora `std::chrono`.

```
auto start = std::chrono::high_resolution_clock::now();
while(relaxation_process_is_not_finished)
    relaxation_process_is_not_finished = relaxationProcess(...);
auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
```

Kod listing 2 Merenje vremena trajanja procesa relaksacija

Ispisom vrednosti promenljive `duration` na terminalu dobija se vreme trajanja procesa relaksacija u sekundama.

```
...DONE!
Vreme izvršavanja procesa relaksacija je: 0.384973 s.
-------------
Srecan put :)
-------------
```

Konzolni ispis 2 Vreme trajanja procesa relaksacija u sekundama

Fixed point analiza

Funkcija `relaxationProcess` ne koristi realne brojeve (kod listing 1, parametri funkcije `relaxationProcess`). Tabela 3 prikazuje oblik svih korišćenih realnih brojeva u formatu sa fiksnom tačkom, ukoliko to bude potrebno za buduća unapređenja sistema. Za prikaz realnih brojeva u tabeli je korišćen minimalan broj bita pri kojem su svi proračuni još uvek tačni.

Promenljiva u modelu	Ukupan broj bita	Broj bita za prikaz celobrojnog dela
Nizovi `graph_latitudes` i `graph_longitudes`	54	6
Izlaz funkcije `geo_dist`	54	15
Kosinusne vrednosti unutar `geo_dist` funkcije	44	2
Zbirovi i razlike geografskih koordinata unutar funkcije `geo_dist`	25	8

Tabela 3 Minimalan potreban broj bita potreban za prikaz realnih brojeva u formatu sa fiksnom tačkom

Literatura

[1] Zybo Z7 10: https://digilent.com/reference/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf [decembar 2024]

[2] O Valgrind-u: https://valgrind.org/info/about.html [decembar 2024]

[3] Više o RoutingKit-u: https://github.com/RoutingKit/RoutingKit/tree/master/doc [decembar 2024]