UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Jovana Adžić

IMPLEMENTACIJA RASPOREĐIVAČA ZADATAKA U DISTRIBUIRANOM SISTEMU NA OSNOVU USMERENOG ACIKLIČNOG GRAFA ZAVISNOSTI

master rad

Beograd, 2024.

Mentor:

dr Mirko Spasić, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Milena Vujošević Janičić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Vesna Marinković, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Porodici

Naslov master rada: Implementacija raspoređivača zadataka u distribuiranom sistemu na osnovu usmerenog acikličnog grafa zavisnosti

Rezime: Distribuirani sistem je računarski sistem koji se sastoji od više nezavisnih računarskih entiteta koji su međusobni povezani mrežom i komuniciraju radi postizanja zajedničkih ciljeva, i kao takvi imaju bitnu ulogu u savremenom računarstvu zbog sposobnosti da omoguće skalabilnost i efikasnost u složenim okruženjima za koje obrada na jednom računaru postaje nedovoljna. U radu se daje prikaz jednog od rešenja implementacije raspoređivača zadataka za primenu u distribuiranom sistemu koje je zasnovano na acikličnom grafu zavisnosti. Rešenje se zasniva na dekomponovanju početnih problema veće kompleksnosti na više jednostanih problema koji u distribuiranom sistemu mogu biti raspoređeni u cilju paralelne obrade, odnosno postizanja ubrzanja.

U radu je analiziran jedan od pristupa dekompozicije kompleksnih problema primenom metode *podeli pa vladaj*, a u cilju dekompozicije osnovnog problema na manje celine pogodne za paralelnu obradu na više računarskih entiteta distribuiranog sistema. Novonastale manje celine međusobno mogu, ili ne, da zavise jedne od drugih. Na osnovu dobijenih zavisnosti zadataka formira se usmereni aciklični graf zavisnosti, koji predstavlja osnovu za formiranje redosleda zadataka za izvršavanje. Na osnovu formiranog grafa zavisnosti implementira se raspoređivač zadataka na procesne elemente distribuiranog sistema.

Predloženo rešenje je teorijski analizirano i eksperimentalno verifikovano pomoću tri praktična problema koji omogućuju sveobuhvatnu analizu više grupa acikličnih grafova. Eksperimentalni rezultati potvrđuju da paralelno distribuirano izvršavanje zadataka na više procesnih elemenata smanjuje vreme obrade u odnosu na njegovo izvršavanje na jednom računaru, što jeste u skladu očekivanom teorijskom analizom.

Ključne reči: graf zavisnosti, stablo, problem, zadatak, red, paralelizam, raspoređivač, distribuirani sistem

Sadržaj

Glava 1

Uvod

Kompanije često imaju velik obim posla koji se mora obaviti — problemi sadrže veliku količinu podataka koju je neophodno što efikasnije obraditi uz što manji utrošak vremena. Ti problemi se mogu izvršavati na jednom računaru. Ali često kompleksnost problema prevazilazi mogućnosti hardvera. Kako sistem raste i zahtevi prema sistemu se povećavaju, potrebno je unaprediti hardver kako bi mogao da pruži dovoljno dobru podršku za obradu. To se postiže vertikalnim skaliranjem.

Vertikalno skaliranje je poboljšavanje karakteristika hardvera jedne mašine. Ono je dobro dok je moguće, ali nakon određene tačke lako se uočava da i najbolji hardver nije dovoljan za obradu srednje/velike količine saobraćaja/komunikacije/izračunavanja [4]. Vertikalno skaliranje može da poboljša performanse sve do najnovijih mogućnosti hardvera. Ove mogućnosti se često pokazuju kao nedovoljne za kompanije sa srednjim ili većim obimom posla.

Nasuprot tome, postoji horizontalno skaliranje koje podrazumeva dodavanje više računara umesto poboljšavanja hardvera pojedinačnog računara. Važna osobina horizontalnog skaliranja je što ne postoji gornji limit skaliranja: kada se performanse degradiraju povećanim zahtevima, količinom podataka ili brojem korisnika potrebno je dodati novu mašinu [4]. U teoriji, dodavanje po potrebi je uvek moguće.

U distribuiranim sistemima horizontalno skaliranje je dominantan način skaliranja i ima brojnih prednosti u savremenom svetu gde se sve više javlja potreba za efikasnom i brzom obradom posla velikog obima. To su sistemi koji se sastoje od više komponenti lociranih na različitim mašinama i koje komuniciraju i koordiniraju sa ciljem da izgledaju kao jedan koherentni sistem krajnjem korisniku. Komponente koje mogu biti sastavni deo distribuiranog sistema su sve mašine koje imaju mogućnost povezivanja na mrežu, imaju svoju lokalnu memoriju i mogu da komuniciraju

putem slanja poruka [4]. Sve komponente sistema rade konkurentno (paralelno), ne postoji globalni časovnik (tj. ne postoji mogućnost globalne sinhronizacije putem časovnika, već to mora da se softverski obezbedi) i sve komponente mogu da prestanu sa radom neočekivano i nezavisno jedna od druge.

Kako se u distribuiranom sistemu izračunavanja često vrše nezavisno na svakom elementu (čvoru) sistema, sam princip i obrada je jednostavna i u opštem slučaju nije skupo dodavanje novih čvorova u sistem i funkcionalnosti kada je to potrebno zarad bolje efikasnosti. Dodatno, distribuirani sistemi mogu da reše neke zadatke i izazove koje nije moguće rešiti na jednoj mašini i zato su danas često korišćeni u praksi [4].

Poželjna osobina distribuiranih sistema je da ne postoje posledice ukoliko jedna mašina iz bilo kog razloga otkaže.[1] S druge strane, ukoliko postoji samo jedna mašina koja u potpunosti obavlja problem, onda se sa njenim padom gubi funkcionalnost u potpunosti. Komunikacija između čvorova sistema može da predstavlja izazov za performanse jer je vreme koje je potrebno da paket stigne od jednog do drugog čvora nekada nezanemarivo, imajući u vidu udaljenost između čvorova i brzinu signala kojim je paket predstavljen.

Pored svih prednosti, postoje i brojni izazovi prilikom dizajniranja ovakvog sistema kako bi bio efikasan. Distribuirani sistem treba da odluči koji posao treba da se izvršava, kada treba da se izvršava i gde treba da se izvršava. Raspoređivanje svakako ima ograničenja koja u nekim situacijama mogu da vode do nedovoljne iskorišćnosti hardvera ili do nepredvidljivog vremena izvšavanja. Takođe, što je šire sistem distribuiran, to je veće kašnjenje uslovljeno komunikacijom.

Ovim radom pokazaćemo jedan od efikasnijih načina razlaganja problema većeg obima na potprobleme u cilju njihove paralelne obrade unutar distribuirane mreže. Takođe, istaći ćemo kako efikasno raspoređivanje elemenata na čvorove sistema može da doprinese značajnom ubrzanju obrade početnog problema u odnosu na njegovo izvršavanje na jednom računaru. U glavi 2 najpre pokazujemo kako se kompleksni problemi mogu razložiti na manje celine koje međusobno mogu ili ne da zavise jedne od drugih, a koje je moguće izvršavati paralelno. Na osnovu dobijenih zavisnosti može se formirati usmereni povezan aciklični graf zavisnosti. Ova glava navodi osnovne pojmove o ovom tipu grafa koji će biti bitni za dalje razumevanje razvoja rešenja. Dalje, u glavi 3, iznete su osnovne ideje implementacije raspoređivača. Dato

[1] Većina distribuiranih sistema se razvijaju da budu tolerantni na padove, što je neophodno jer sistemi mogu biti sazdani od stotina čvorova koji rade zajedno

je proučavanje serijske obrade problema koja je bitna radi boljeg razumevanja samih prednosti ostvarenih paralelnom obradom u distribuiranom sistemu. Izneti su osnovni pojmovi i teorijske osnove, kao i kratak pregled dve klase problema koje će biti proučavane u radu kroz tri opisana predstavnika. Nakon toga prikazan je opis paralelne obrade: dati su osnovni pojmovi i prikazan je dizajn (idejna implementacija) samog raspoređivača zadataka dobijenih dekomponovanjem posmatranih problema metodom *podeli pa vladaj* [2] na osnovu datog usmerenog acikličnog grafa zavisnosti. Prikazani su osnovni koncepti implementacije nad predstavnicima dve klase problema kroz tri konkretna primera ranije pomenuta u delu o serijskoj obradi. U glavi 4 dat je pregled programske implementacije rešenja i osnovnih koraka koje je čine. Na kraju, u glavi 5, kao i zaključku, prikazani su i sami praktični rezultati radom opisanog rešenja, data je analiza ponašanja za različite probleme i izneti su zaključci o prednostima ovako napisanog raspoređivača, kao i nedostaci koji utiču na njegove performanse.

[2]Podeli pa vladaj je strategija dizajniranja algoritama zasnovana na rekurziji sa višestrukim grananjem. Ovakvi algoritmi se zasnivaju na rekurzivnom razlaganju problema na dva ili više podproblema istog (ili sličnog) tipa (podeli), sve dok problem ne postane dovoljno jednostavan da se može direktno rešiti (vladaj).

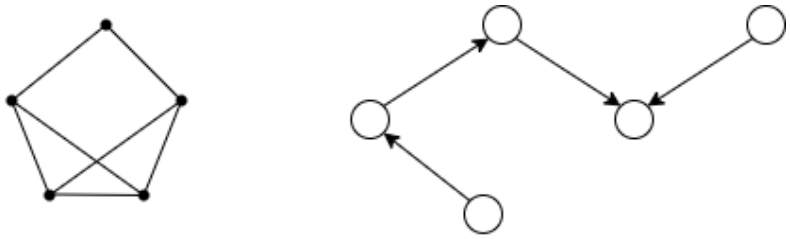
Glava 2

Grafovi i zadaci

2.1 Grafovi - osnovni pojmovi

Graf G je uređeni par skupova (V, E), gde je V skup čvorova, a E skup grana. Grane predstavljaju skup neuređenih parova elemenata iz skupa čvorova V [13].

Za par čvorova (v, w) kažemo da je **ivica grafa** ako postoji grana koja ih povezuje. Čvorovi koji određuju jednu ivicu se nazivaju krajevima ivice. Ukoliko je svaka ivica uređen par čvorova, za odgovarajući graf kažemo da je **usmeren** (digraf) [7]. U tom slučaju, grana koja povezuje čvorove ima svoj smer. Formalno, uređen par (v, w) je tada ivica usmerena od v ka w, odnosno ivica sa početkom u v i krajem u w. Vizuelno, graf je skup čvorova (tačaka, krugova) povezan granama (linijama ili strelicama), kao što je prikazano na slici 2.1.



Slika 2.1: Primeri grafa

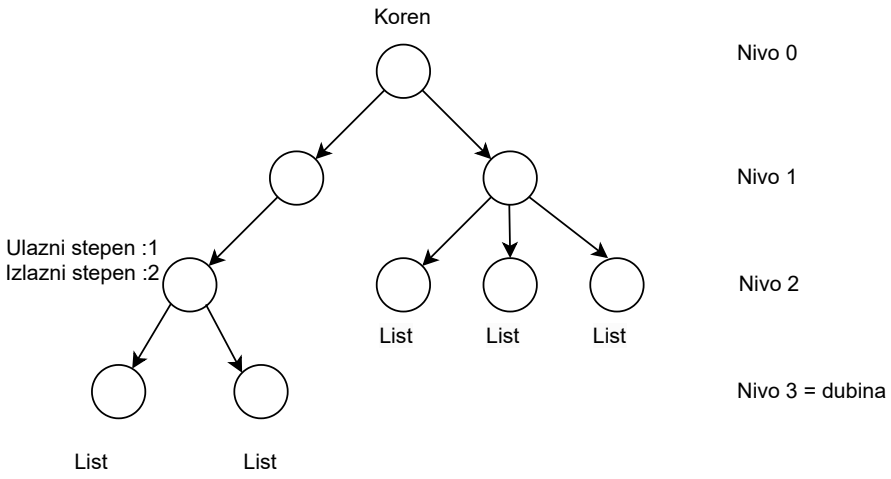
Put (*eng. path*) u grafu G je niz čvorova u kome je svaki par uzastopnih čvorova međusobno povezan. Put predstavlja putanju kojom se može kretati kroz graf prateći njegove grane i posećujući čvorove u određenom redosledu [12]. Graf G je **povezan** ako se svaka dva njegova čvora mogu povezati putem.

Ciklus (*eng. cycle*) u grafu G je put u G koji počinje i završava se istim čvorom. Dakle, ciklusi su zatvoreni putevi u grafovima. Za graf koji nema ciklusa u sebi

kažemo da je **acikličan** (*eng. acyclic*) [7]. Dakle, **usmeren aciklični graf** (*eng. directed acyclic graph — DAG*) je graf bez ciklusa u kojem je svaka grana usmerena od jednog čvora ka drugom (od roditelja ka detetu). Jedan od primera ovakvog grafa je **stablo**. Stablo se u programiranju ostvaruje korišćenjem pokazivača kako bi se jedan čvor usmerio ka drugome. Naime, svaki čvor se konstruiše tako da poseduje mogućnost čuvanja jedne ili više adresa drugih čvorova, što omogućava „prelaženje“ iz jednog čvora u drugi, tj. kretanje po stablu i daje njihovu međusobnu vezu.

Stepen čvora u stablu je broj podstabala datog čvora. Ulazni (izlazni) stepen čvora predstavlja broj grana koje ulaze u čvor (izlaze iz čvora) [8]. Koren stabla je čvor sa ulaznim stepenom nula, dok su listovi čvorovi čiji je izlazni stepen nula. Stablo čiji je svaki čvor (sem korena) stepena najviše dva se naziva binarno stablo. Moguće je konstruisati stabla različitih stepena. Primer stabla dat je na slici 2.2.

Nivo čvora predstavlja njegovo rastojanje (broj grana) od korena [8]. **Dubina stabla** je maksimalna vrednost nivoa nekog čvora u stablu, dok **visina stabla** predstavlja dužinu najdužeg mogućeg puta od korena do lista. Iako drugačije definisani, dati pojmovi predstavljaju istu vrednost.



Slika 2.2: Primer stabla

2.2 Paralelna obrada zadataka

Često nailazimo na probleme za čije je izvršavanje potrebno mnogo vremena. To mogu biti kompleksni problemi sa zahtevnijom obradom podataka, ali i jednostavni-

ji zadaci koji sadrže veliku količinu podataka za obradu. Za takve probleme serijsko izvršavanje jednog po jednog koraka je neefikasno. Nekada je, u takvim slučajevima, neophodno naći rešenje koje ubrzava obradu, tj. skraćuje vreme izvršavanja. Jedan od mogućih pristupa je paralelna obrada, tačnije distribuirano izračunavanje. Distribuirano izračunavanje (*eng. distributed computing*) je tehnika podele velikog problema (npr. agregacija 100 milijardi podataka) koji nijedan pojedinačni računar ne može praktično da izvrši, u puno manjih zadataka, tako da svaki manji može da se izvrši na pojedinačnoj mašini [4]. Mašine (procesni elementi) su entiteti distribuiranog sistema koji međusobno komuniciraju putem mreže koristeći klijent-server model komunikacije. One paralelno vrše obrade potproblema. Dakle, rešavanje početnog problema se svodi na podelu velikog zadatka na manje zadatke, izvršavanje dobijenih zadataka paralelno na pojedinačnim procesnim elementima distribuiranog sistema i agregiranje izračunatih podataka na odgovarajući način, u zavisnosti od prirode problema. Cilj je da se problem deli dok se ne formiraju jednostavniji elementi koji se nezavisno od ostalih mogu izvršavati. Prethodno opisana istovremena obrada tako dobijenih manjih jedinica dovodi do značajnog ubrzanja izvršenja primarnog problema.

2.3 Zadaci i grafovi zavisnosti

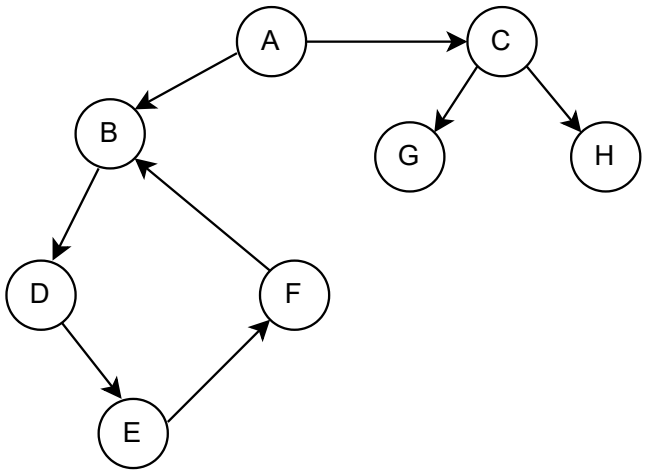
Zadatak (*eng. task*) je jedinica programa, slična potprogramu, koja može da se izvrši konkurentno sa drugim jedinicama istog programa [4]. Ona predstavlja manju nedeljivu jedinicu problema koja se sekvencijalno kratko izvršava. Zadaci međusobno moraju da komuniciraju. Komunikacija se odnosi na svaki mehanizam koji omogućava jednom zadatku da dobije informacije od drugog. Komunikacija se može ostvariti preko zajedničke memorije (ukoliko zadaci dele memoriju) ili razmenom poruka [4]. Problem treba biti podeljen tako da se svi dobijeni zadaci mogu što nezavisnije izvršavati.

Čest je slučaj da su zadaci međusobno zavisni i da je potrebno izvršiti ih u tačno definisanom redosledu kako bi se početni problem rešio [2]. Takva dekompozicija problema se može ilustrovati u vidu usmerenog grafa gde bi čvorovi odgovarali zadacima, a zavisnosti među njima bile predstavljene usmerenim granama [11]. Kako se vrši dekompozicija većih na manje zadatke između kojih se u svakom koraku stvara veza (grana u grafu), graf će biti povezan jer je svaki čvor dobijen razlaganjem zadatka sa kojim je direktno povezan, a svi čvorovi su nastali razlaganjem početnog

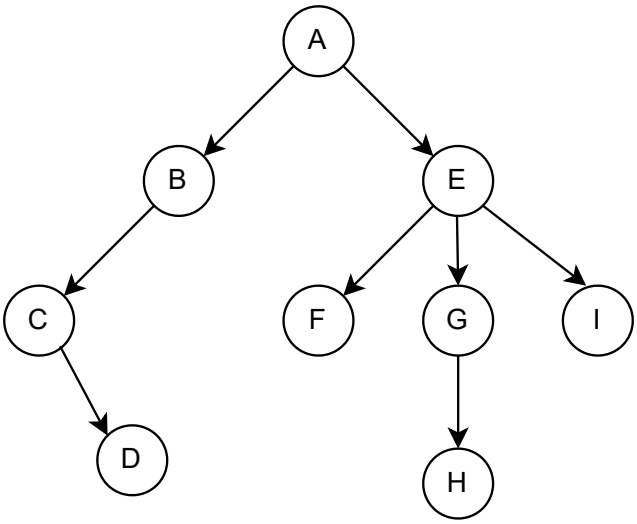
problema koji je koren grafa.

Grafovi nastali na taj način mogu biti različiti. Ako, na primer, problem razložimo tako da ne obratimo precizno pažnju na međusobne zavisnosti i zadatke projektujemo (dizajniramo) tako da u nekom trenutku jedan zadatak zavisi od izvršenja drugog, koji je u hijerarhiji niži od njega, dobićemo ciklus. Ovakav ciklični graf zavisnosti nije dobar jer može da prouzrokuje mrtvu petlju (*eng. deadlock*): dva zadatka ne mogu biti završena pošto čekaju na završetak izvršavanja onog drugog. Primer cikličnog grafa dat je na slici 2.3. Vidimo da je ciklus na slici označen nizom čvorova $B - D - E - F$.

Poželjno je da usmereni povezani graf zavisnosti bude bez ciklusa. Stablo je jedan od takvih primera grafa (slika 2.4). Koren predstavlja početni problem (čvor A na slici), dok su čvorovi dekomponovani zadaci. Listovi opisuju zadatke koji se mogu nezavisno izvršiti. Ovakvo stablo predstavlja osnovu za raspoređivanje elementa na različite jedinice distribuiranog sistema u cilju njihove paralelne obrade.



Slika 2.3: Primer cikličnog grafa



Slika 2.4: Primer acikličnog grafa — stablo

Glava 3

Idejna implementacija rešenja

U ovom poglavlju biće predstavljeni osnovna ideja implementacije i koncept rešenja raspoređivača. Kako bi u potpunosti razumeli prednosti paralelne obrade, kroz tri konkretna problema analizirani su serijska obrada i njena ograničenja. Koristeći iste probleme, potom je opisana njihova paralena izvedba zajedno sa konceptom rešenja, odnosno dizajnom raspoređivača.

3.1 Sekvencijalna obrada problema

Sekvencijalna (serijska) obrada problema se odnosi na rešavanje problema tako što se operacije obavljaju redom, jedna za drugom, u unapred određenom redosledu bez preklapanja. Svaka operacija zavisi od završetka prethodne, što dovodi do toga da su rezultati sekvencijalnih algoritama često predvidivi i reproducibilni [6].

Sekvencijalni algoritmi se često analiziraju u terminima vremena koje je potrebno da se završe i prostora koji zauzimaju koristeći O-veličinu. Neretko ovi algoritmi mogu biti sporiji za probleme sa kompleksnijom obradom ili probleme sa velikom količinom podataka za obradu [10]. Takođe, ograničeni su sposobnošću jednog procesora i ne mogu iskoristiti višeprocesorski sistem. Posmatraćemo tri problema i njihovo sekvencijalno izvršavanje.

3.1.1 Problem izračunavanja sume elemenata niza

Prvi problem koji ćemo posmatrati je problem izračunavanja sume elemenata niza. Sekvencijalno izračunavanje ovde podrazumeva iterativni postupak, prolazak

kroz svaki element niza jedan po jedan i njegovo dodavanje na globalnu sumu. Ovaj pristup je jednostavan, direktan i lako razumljiv. Pseudokod dat je algoritmom 1.

Algoritam 1 Izračunavanje sume elemenata niza

```
1: Ulaz: Niz A sa n elemenata
2: Izlaz: Suma elemenata niza sum
3:
4: sum ← 0
5: for i = 0 to n − 1 do
6:     sum ← sum + A[i]
7: end for
8: return sum
```

Vremenska složenost ovakvog algoritma je $O(n)$, gde je n broj elemenata u nizu. Jednostavan je i efikasan za manje nizove, pa predstavlja često korišćeno rešenje u tim slučajevima. Za jako velike nizove sporiji je u poređenju sa paralelnim pristupom sabiranja, tako da ovakvo izračunavanje nije optimalno za velike nizove podataka.

3.1.2 Problem nalaženja minimalnog elementa niza

Drugi problem jeste problem nalaženja minimalnog elementa niza. Iterativni postupak podrazumeva, najpre, definisanje najmanjeg elementa niza (minimuma), za koji se na početku može odabrati prvi element. Zatim, dodatno prolazimo od drugog do poslednjeg elementa niza i proveravamo za svakog od njih da li je manji od trenutnog minimuma. Ako jeste, onda se minimum menja vrednošću tog elementa niza. Pseudokod je dat algoritmom 2. Vremenska složenost ovakvog algoritma je $O(n)$. Izvršavanje problema za velike nizove se može ubrzati paralelnom obradom u odnosu na sekvencijalnu.

3.1.3 Problem računanja histograma

Treći problem koji ćemo proučavati je problem računanja histograma. Problem podrazumeva računavanje frekvencije (broj pojavljivanja) svakog elementa niza. Posmatrajmo algoritam dat pseudokodom 3.

Dati algoritam koristi mapu *histogram* za čuvanje frekvencija (pojavljivanja) elemenata niza A. Promenljiva *element* uzima vrednost $A[j]$ i vrednost tog elementa u histogramu (broj pojavljivanja elementa $A[j]$) uvećava svakim njegovim pojavlji-

Algoritam 2 Pronalaženje najmanjeg elementa niza

```
1: Ulaz: Niz A sa n elemenata
2: Izlaz: Najmanji element niza min_val
3:
4: min_val ← A[0]
5: for i = 1 to n − 1 do
6:     if A[i] < min_val then
7:         min_val ← A[i]
8:     end if
9: end for
10: return min_val
```

Algoritam 3 Izračunavanje histograma

```
1: Ulaz: niz A, veličina niza n
2: Izlaz: histogram
3: for j ← 0 to n − 1 do
4:     element ← A[j]
5:     histogram[element] ← histogram[element] + 1
6: end for
7: return histogram
```

vanjem za jedan. Ako ne postoji ključ sa vrednošću *element* u mapi, onda se on dodaje u mapu i daje mu se vrednost nula.

Složenost date implementacije zavisi od struktura podataka koje se koriste prilikom implementacije algoritma u određenom programskom jeziku. Ako koristimo standardnu `C++` biblioteku `std::map` za predstavljanje histograma, za korak ažuriranja frekvencije elementa je neophodno proći kroz mapu i proveriti da li element već tu postoji. Sama operacija u tom slučaju (provera i umetanje ili ažuriranje vrednosti) ima složenost $O(\log n)$. Kako imamo ukupno n elemenata za koje je potrebno ovu operaciju izvršiti, to ukupna složenost u tom slučaju iznosi $O(n \log n)$. Dakle, sam algoritam prema opisanoj složenosti, nije efikasan za velike nizove podataka [1].

[1] Napominjemo da je ovo samo jedna od mogućih vremenskih složenosti izračunavanja datog algoritma u slučaju upotrebe ovakve strukture podataka. Za različite korišćene strukture u različitim programskim jezicima, sama složenost algoritma će varirati. Ako je mapa implementirana pomoću heš tabele, složenost algoritma će biti $O(n)$ i biće bolja nego u slučaju korišćenja balansiranog stabla kakva je implementacija `std::map` u *C++*

3.2 Paralelna obrada problema i koncept rešenja

Sa rastućim potrebama za obradom velike količina podataka i izvođenjem složenih računarskih operacija u kraćem vremenskom roku, serijsko izvršavanje ne pruža neophodnu efikasnost. Paralelno procesiranje postaje ključno u savremenim računarima i distribuiranim sistemima. Paralelna obrada omogućava podelu zadataka na manje delove, koji se istovremeno izvršavaju na više procesora ili jezgara, što znatno ubrzava rad i povećava efikasnost sistema. Međutim, kako bi paralelna obrada bila efikasna, ključno je upravljati rasporedom zadataka i resursima na optimalan način.

Raspoređivač zadataka upravlja izvršavanjem zadataka u paralelnim i distribuiranim sistemima. Njegov osnovni cilj je da optimizuje vreme izvršavanja zadataka, ravnomerno raspodeli opterećenje i obezbedi pravilno korišćenje sistemskih resursa. Raspoređivač mora voditi računa, između ostalog, o prioritetima zadataka, raspoloživosti resursa i zavisnostima među zadacima. Efikasnost paralelne obrade će zavisiti od načina na koji raspoređivač, vodeći računa o prethodno navedenim stavkama, izvrši raspodelu zadataka na optimalan način.

3.2.1 Osnovni pojmovi paralelne obrade

Kao što je ranije istaknuto, jedan od načina efikasnije obrade složenih problema, u smislu kompleksnosti same obrade ili velike količine podataka, predstavlja paralelno izvršavanje. Podelom početnog problema na manje celine, takozvane zadatke, omogućava se primena paralelizma zarad dobijanja rezultata primarnog problema.

Rečeno je ranije da je od zadataka moguće napraviti graf zavisnosti među njima [9]. Naime, kako se početni problem deli na manje celine i zavisan je od njihovih rezultata obrade, on predstavlja koren stabla kao osnovu celog grafa. Povezan je usmerenim granama sa direktnim potomcima koji su dobijeni njegovom podelom. Postupak dobijanja ostalih čvorova grafa se rekurzivno nastavlja na potomcima dok se ne dostigne neki unapred postavljen uslov ili zadaci postanu u toj meri jednostavni da se ne mogu dalje razlagati. U oba slučaja, završnom podelom se dobijaju nezavisni zadaci, tj. zadaci koji ne zavise od obrade drugih elemenata.

Listovi ovako nastalog grafa će predstavljati delove koji se mogu izvršavati paralelno, nezavisno od ostalih elemenata jer ne poseduju potomke, kao ni direktne veze među sobom. Ovakav način podele problema dovodi na nastanka usmerenog acikličnog grafa zavisnosti. Graf je usmeren jer je smer grana jasno određen tj. tačno je utvrđena zavisnost izvršavanja među čvorovima. Graf nema ciklusa jer ne

postoji putanja koja bi povezala neki čvor sa samim sobom — svaki čvor se razlaže na zadatke koji se rekurzivno razlažu na još manje celine od kojih ne zavisi nijedan prethodan u hijerarhiji. Usmeren aciklični graf formiran na ovakav način jeste stablo.

Kritična putanja ovakvo dobijenog grafa predstavlja visinu stabla tj. najveće rastojanje između korena i listova [11]. Ako znamo da su listovi grafa najmanje celine koje ne zavise ni od jednog drugog elementa i mogu se izvršavati paralelno, a svi ostali čvorovi grafa zavise od njihove obrade, najefikasnije vreme za izvršenje primarnog problema će biti srazmerno vrednosti kritične putanje.

Brojač referenci (*eng. reference counter*) je tehnika koja prati koliko puta je određeni čvor u grafu referenciran od strane drugih čvorova. U grafu zavisnosti zadataka, brojač referenci za svaki čvor će predstavljati broj njegove dece (direktnih potomaka) od kojih direktno zavisi njegovo izvršavanje. Dakle, listovi su jedini čvorovi grafa čiji je brojač referenci jednak nuli.

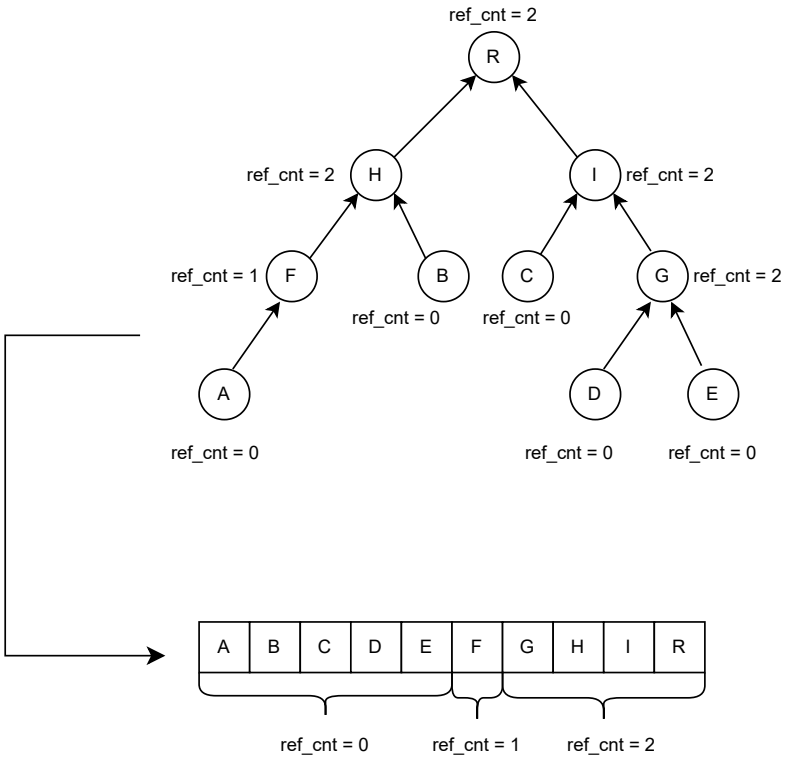
3.2.2 Dizajn raspoređivača

Osnovna ideja rada je napraviti raspoređivač zadataka u distribuiranom sistemu zarad paralelne obrade na osnovu grafa zavisnosti. Naime, posmatraćemo dve klase problema dobijene metodom *podeli pa vladaj*. Jedna će predstavljati probleme koji se mogu razložiti tako da su svi dobijeni zadaci međusobno nezavisni i kao takvi, mogu se nezavisno, paralelno izvršavati [11]. Druga klasa, pak, predstavlja probleme koji se razlažu na više zavisnih zadataka. Tačnije, primarni zadatak se deli na manje potprobleme koji su međusobno nezavisni, a oni dalje na svoje potprobleme od kojih direktno zavise. Obe klase formiraju usmeren aciklični graf, preciznije stablo, ali visine stabala, kao i stepeni čvorova će biti različiti. Ako posmatramo brojač referenci tako dobijenih stabala, unutrašnji čvorovi će imati vrednost različitu od nule jer poseduju direktne potomke čiji broj daje i samu vrednost brojača. Listovi su jedini čvorovi čiji je brojač jednak nuli. Dobijanje grafa na prethodno opisan način i određivanje vrednosti brojača referenci za svaki element grafa predstavlja prvi deo implementacije raspoređivača.

Drugi deo podrazumeva prolazak kroz dobijeno stablo i raspoređivanje datih čvorova tj. zadataka unutar neke strukture podataka. U našoj implementaciji biće korišćena struktura podataka red sa prioritetom. Red sa prioritetom podrazumeva strukturu u kojoj je svakom elementu pridružen određen prioritet koji određuje poredak elemenata. Element sa većim prioritetom se uzima iz reda pre onog sa

nižim. Prioritet koji se dodeljuje elementima može biti različit. Ovde je prioritet dodeljen prema vrednosti brojača referenci — oni sa manjom vrednošću brojača imaju viši prioritet.

Red sa prioritetom se formira na osnovu napravljenog grafa, tako da se u njemu najpre nalaze zadaci čiji je brojač referenci jednak nuli. To su oni zadaci koji ne zavise od drugih zadataka, nema ni međusobnih zavisnosti među njima samima, tako da se svi mogu istovremeno, paralelno izvršavati. Nakon njih, nalaze se zadaci sa narednom najnižom vrednošću brojača i tako redom sve do onog zadatka sa najvećom vrednošću datog brojača. Poslednji će biti onaj element čije izvršavanje zavisi od najvećeg broja direktnih potomaka tj. onaj koji ima najviše dece u grafu zavisnosti. U ovom redu prioritet će imati oni elementi sa nižom vrednošću brojača referenci. Primer je dat na slici 3.1. Sa *ref_ cnt* je označena vrednost brojača referenci.



Slika 3.1: Nastanak reda od grafa

Sledeći korak podrazumeva raspoređivanje zadataka za paralelnu obradu na procesne elemente datog distribuiranog sistema pomoću dobijenog reda. Prvi na izvršavanje će biti poslati oni elementi reda čiji je brojač referenci jednak nuli jer za

njihovu obradu nisu potrebni rezultati nijednog drugog zadatka. U zavisnosti od broja dostupnih procesnih elementa, biće moguće izvršavanje svih takvih zadataka paralelno, ili će neki, u slučaju manjeg broj procesnih elemenata u odnosu na broj onih čiji je brojač referenci jednak nuli, morati da sačekaju završetak izvršavanja već raspoređenog zadatka kako bi započeli svoju obradu. Kad se izračunavanje zadatka završi i dobije rezultat njegove obrade, tada njegov roditelj, tj. direktni predak koji zavisi od izvršenog zadatka, stiče potencijalni uslov za svoje izvršenje. Jedan njegov potomak je izvršen, pa se vrednost njegovog brojača može smanjiti za jedan. Ovaj postupak se ponavlja sve dok na raspored za izvršenje ne dođe koren tj. dok brojač referenci datog početnog problema ne postane jednak nuli. To znači da su se svi zadaci paralelno izvršili i rezultat njihove obrade je potrebno iskoristiti u obradi početnog problema.

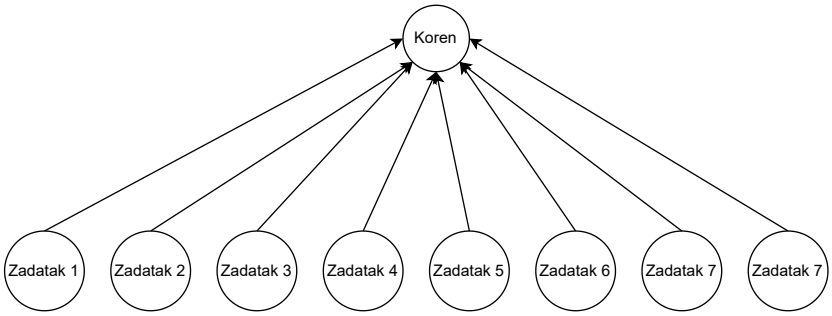
Osnovne ideje implementacije ćemo predstaviti pomoću, već pomenute, dve klase problema, a čiji su predstavnici i opisani u poglavlju 3.1 o serijskoj obradi. Ove dve klase će dati tri različita stabla koja će uticati na način raspoređivanja zadataka za paralelnu obradu.

3.2.3 Problem izračunavanja sume elemenata niza

Problem izračunavanja sume elemenata niza pripada prvoj klasi problema. To je klasa onih problema koji se mogu razložiti na nezavisne, manje zadatke tako da se svi mogu paralelno izvršavati, bez ikakvih međusobnih zavisnosti. Kako bismo razbili početni niz na zadatke, uvodimo pojam **stepena granulacije** (*eng. grain size*). Stepen granulacije predstavlja veličinu (dužinu) svakog podniza originalnog niza, tj. definiše broj elemenata u svakom podnizu (zadatku). Sam broj zadataka se onda dobija kao količnik ukupne dužine originalnog niza i stepena granulacije. Graf za ovakvu klasu problema će imati koren koji predstavlja početni niz (problem) i svi podnizovi, kako su međusobno nezavisni, biće listovi čija je vrednost brojača referenci jednaka nuli, a njihova dužina jednaka datom stepenu granulacije. Prikaz datog grafa dat je na slici 3.2.

Dalje, red zadataka ovako dobijenog grafa će sadržati sve elemente sa vrednošću brojača referenci jednakom nuli, osim poslednjeg. Brojač referenci poslednjeg elementa reda je jednak broju direktnih potomaka od kojih direktno zavisi njegovo izvršavanje, a to su svi prethodni elementi reda. Dakle, to je koren grafa, tj. početni problem. Obrada se onda može izvršavati paralelno na svim dostupnim procesnim elementima distribuiranog sistema za sve zadatke sem korena. Tražena suma će se

uvećavati rezultatom svakog obrađenog zadatka u redu. Nakon izvršenja svih listova, konačna suma će biti dobijena i vraćena kao rezultat obrade korena.



Slika 3.2: Graf nezavisnih zadataka — prva klasa problema

Kako je kritična putanja ovakvog grafa jednaka 1, vreme potrebno za izvršenje početnog problema, u najboljem slučaju, će biti srazmerno obradi jednog zadatka, tj. niza veličine datog stepena granulacije. To se ostvaruje kada je broj dostupnih procesnih elemenata jednak ili veći od broja zadataka. Ukoliko je taj broj manji, onda je neophodno sačekati na izvršenje prvobitno raspoređenih zadataka. Time se u zavisnosti od broja procesnih elemenata, povećava vreme obrade.

Serijska obrada podrazumeva postojanje samo jednog procesnog elementa na kom izračunavanje traje neko vreme t. Ukoliko imamo dva procesna elementa, očekivano vreme će se smanjiti na okvirno $t/2$. Analogno, za svaki broj m procesnih elemenata, vreme obrade se umanjuje na okvirno t/m, ali ne može biti brže od vremena neophodnog za obradu jednog zadatka jer je kritična putanja upravo jednaka 1.

Radi boljeg razumevanja, posmatrajmo uopšten primer 1 grafa koji pripada posmatranoj klasi problema datog na slici 3.3 i raspoređivanje njegovih čvorova i listova na procesne elemente sistema prikazano na slikama 3.4, 3.5 i 3.6.

Primer 1: *Neka problem, bez umanjenja opštosti, u ovom primeru budu podeljen na 8 podnizova (8 zadataka). Neka vreme izvršenja svakog zadatka bude* $1s$*, dok vreme izvršenja korena, koje podrazumeva samo korišćenje rezultata obrade listova (zadataka), uzima vrednost* $0.2s$*. Serijsko izvršenje bi podrazumevalo obradu svih zadataka redom, što znači da bi, u ovom slučaju bilo srazmerno broju zadataka —* $8s$*.*

Prvi slučaj na slici prikazuje obradu na dva procesna elementa. Kako se zadaci nezavisno mogu izvršavati, možemo ih redom slati na obradu, naizmenično, na oba

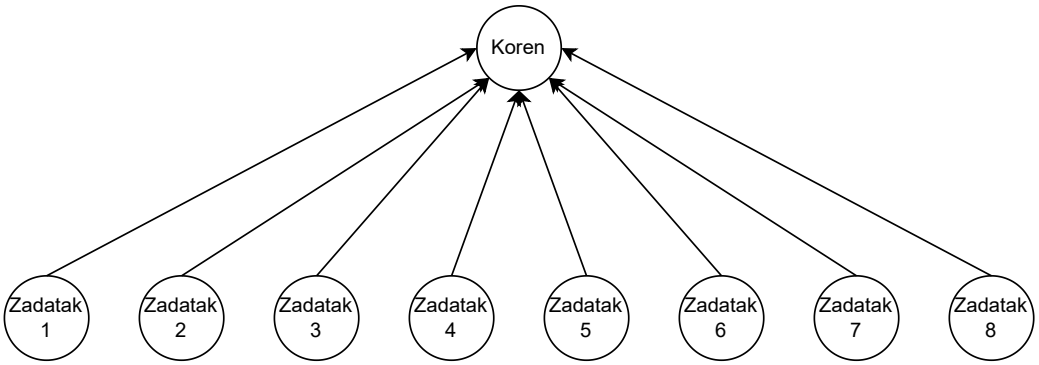
računara. Takva logika raspoređivanja će se koristiti i u ostala tri slučaja sa slike. Kako su dostupna dva računara, a broj zadataka je 8, optimalno raspoređivanje daje izvršenja 4 zadatka po procesnom elementu. Na kraju, neophodno je uraditi obradu korena koja traje $0.2s$*. Dakle, vreme obrade bi u tom slučaju bilo približno* $4s + 0.2s = 4.2s$*.*

Sledeći slučaj podrazumeva postojanje 4 procesna elementa. Osam zadataka na četiri elementa raspoređujemo tako da se na svakom izvršavaju po dva. Na kraju, izvršava se koren. Dakle, optimalno vreme izvršavanja u ovom slučaju bi okvirno bilo $2s + 0.2s = 2.2s$*.*

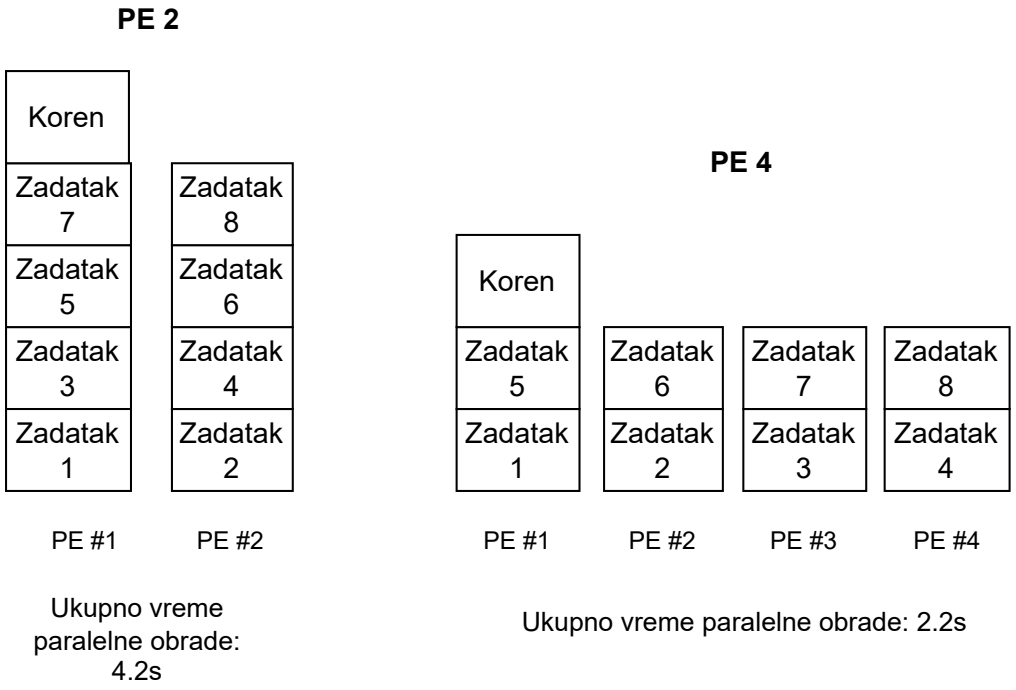
Ako imamo 8 dostupnih procesnih elemenata u mreži, možemo sve zadatke odjednom poslati na paralelnu obradu i njihovim istovremenim izvršanjem dobiti rezultat koji se koristi za obradu korena nakon obrade svih listova. Dakle, obrada će u ovom slučaju biti srazmerna $1s + 0.2s = 1.2s$*.*

Poslednji slučaj prikazuje situaciju kada je broj dostupnih računara veći od broja zadataka koje želimo da obradimo. U tom slučaju, koristimo onoliko elemenata koliko imamo zadataka, ostatak se ne koristi. Dakle, vreme obrade će i u ovom slučaju biti najmanje $1.2s$*.*

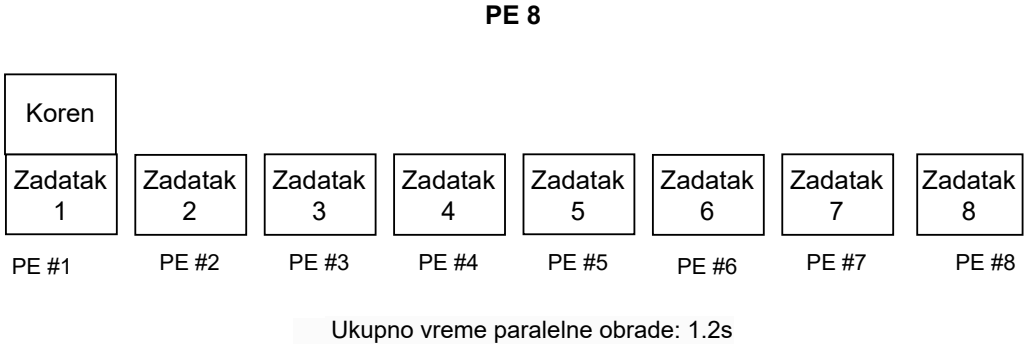
Podsetimo se priče o kritičnoj putanji. Sa ovog grafa vidimo da je njena vrednost jednaka 1. Rečeno je da je vreme paralelne obrade srazmerno vrednosti kritične putanje, ne može biti manje od visine stabla zbog zavisnosti u izvršavanju čvorova. Dakle, ovaj primer to i pokazuje.



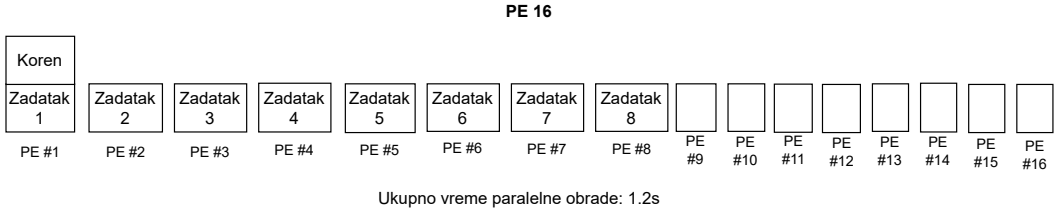
Slika 3.3: Graf nezavisnih zadataka — prva klasa problema



Slika 3.4: Raspoređivanje zadataka na dva i četiri procesna elementa



Slika 3.5: Raspoređivanje zadataka na osam procesnih elementa

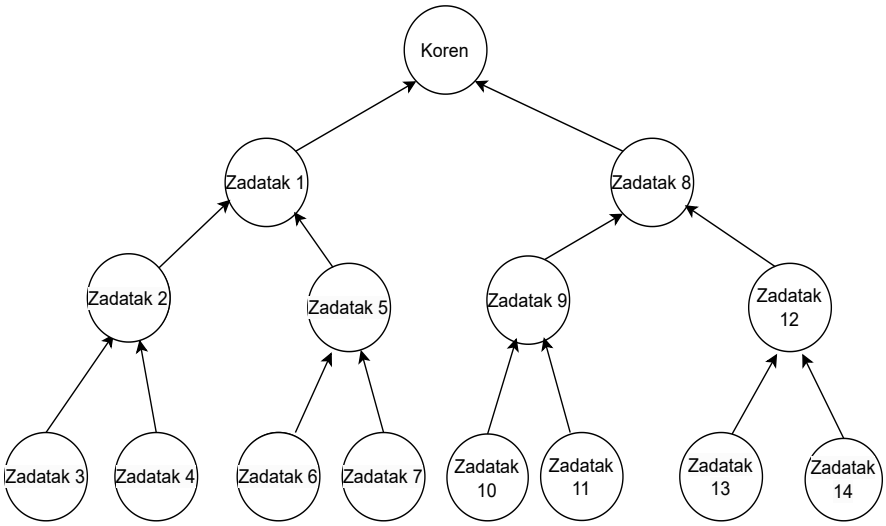


Slika 3.6: Raspoređivanje zadataka na šesnaest procesnih elementa

3.2.4 Problem nalaženja minimalnog elementa niza

Problem nalaženja minimalog elementa niza pripada drugoj klasi problema. To je klasa onih problema koji se razlažu na zadatke koji su međusobno zavisni tj. postoji zavisnost u izvršavanju koja se ogleda u postojanju određene hijerarhije u grafu. Naime, početni niz razlažemo na podnizove strategijom *podeli pa vladaj* dok se ne dostigne unapred postavljen uslov zaustavljanja (npr. veličina dobijenog podniza bude jednaka stepenu granulacije). Delimo problem najpre na dva osnovna podniza — dva nova zadatka od kojih će njegovo izvršavanje zavisiti, a pritom su novodobijeni zadaci međusobno nezavisni. Dalje, na isti način, svaki od dobijenih zadataka delimo na dva podzadatka od kojih njihovo izvršavanje zavisi. Postupak se nastavlja dok se ne dostigne postavljen uslov. Tada dobijamo zadatke koji su svi međusobno nezavisni i nalaze se na kraju hijerarhije, od njih zavisi izvršavanje svih ostalih zadataka sistema. Graf problema ove klase dat je na slici 3.7.

Jedini nezavisni zadaci su listovi. Njihov brojač referenci ima vrednost nula, tako da se oni mogu izvršavati paralelno na procesnim elementima. Njihova obrada podrazumeva iterativnu proveru nalaženja najmanjeg elementa podniza koji obrađuju. Kako je stablo dobijeno ovakvom podelom binarno, vrednost brojača referenci ostalih čvorova je najviše dva, tako da oni moraju sačekati izvršenje zadataka od kojih zavise. Kada se listovi izvrše, brojač referenci roditelja se umanjuje za jedan. Ako je neki dostigao nulu, njegovi potomci su se izvršili, a on se stavlja u stanje za izvršavanje. Pošto se oslanja na rezultate obrade zadataka od kojih direktno zavisi, njegova obrada se svodi na jednostavno poređenje vrednosti dobijenih od dece radi utvrđivanja manjeg elementa. Nakon obrade svih zadataka u redu (listova i čvorova grafa zavisnosti) koren će jednostavim poređenjem vrednosti rezultata obrade direktnih potomaka dati najmanji element celog niza.



Slika 3.7: Graf zavisnih zadataka — druga klasa problema

U ovom slučaju kritična putanja će imati vrednost veću od jedan jer postoji makar jedan nivo zavisnosti među zadacima tako da će ona biti srazmerna visini stabla. Zbog postojanja zavisnosti ne mogu se svi zadaci bezuslovno paralelno obrađivati već samo oni sa brojačem referenci jednakim nuli, dok ostali moraju čekati na izvršenje svoje dece. Samim tim, moguće je postojanje određenog čekanja na izvršavanje, tako da se vreme obrade u ovom slučaju povećava. Kako listovi rade glavnu obradu problema jer unutrašnji čvorovi, kao podnizovi od kojih su nastali listovi koriste rezultate obrade listova i rade jednostavnija izračunavanja, to je vreme obrade listova najveće, a samih unutrašnjih čvorova će trajati kraće, zavisno od problema koji se posmatra.

Radi boljeg razumevanja koncepta, neka je dat na slikama 3.8, 3.9, 3.10 i 3.11 jedan uopšten prikaz grafa opisanog primerom 2 koji pripada drugoj klasi problema u kojoj postoji zavisnost u izvršavanju zadataka i optimalno raspoređivanje njegovih čvorova i listova za obradu na procesne elemente distribuiranog sistema.

Primer 2: *Neka problem, bez umanjenja opštosti, u ovom primeru budu razložen tako da daje 8 podnizova listova, a da do korena budu svi zadaci koji od listova zavise i neka aciklični graf bude dat u vidu binarnog stabla. Neka vreme izvršenja svakog zadatka koji je list grafa lista bude 1s. Obrada svakog unutrašnjeg čvora koristi rezultate izračunavanja potomaka i neka njihovo vreme izvršavanja bude manje —* $0.2s$*. Serijsko izvršenje bi podrazumevalo obradu svih zadataka listova redom jer je*

ceo niz sadržan upravo u njima, što znači da bi, u ovom slučaju, bilo srazmerno broju zadataka — $8s$.

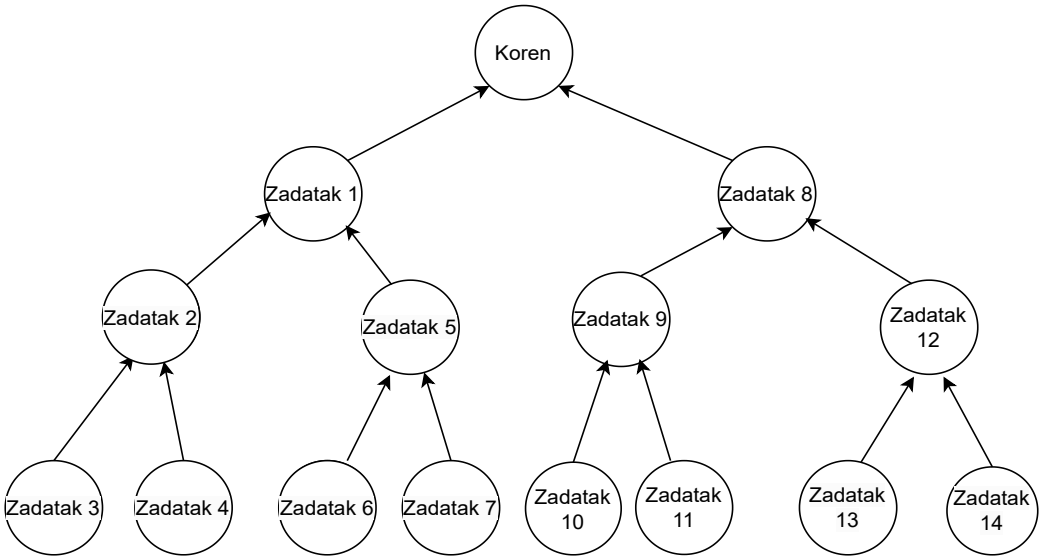
Prvi slučaj na slici prikazuje obradu na dva procesna elementa. Kako se zadaci koji su listovi nezavisno mogu izvršavati, možemo ih redom slati na obradu, naizmenično, na oba računara. Kako su dostupna dva računara, a broj zadataka je 8, optimalno raspoređivanje daje izvršenja 4 zadatka po procesnom elementu. Obrada preostalih zadataka zavisi od obrade listova i mora se čekati na njihovo izvršenje. Kada se steknu uslovi za izvršavanje nekog unutrašnjeg čvora, on se šalje na obradu na prvi slobodan element, pri čemu se obrada dva nezavisna iz grafa može istovremeno vršiti. Tako je u svim slučajevima do korena. Dodatno, $0.2s$ je potrebno za obradu korena. Dakle, optimalno vreme obrade bi bilo u tom slučaju $4s+0.2s*3+0.2s = 4.8s$.

Sledeći slučaj podrazumeva postojanje 4 procesna elementa. Osam zadataka koji su listovi na četiri elementa raspoređujemo tako da se na svakom izvršavaju po dva. Dakle, optimalno vreme izvršavanja u ovom slučaju bi bilo dve sekunde. Ostale elemente raspoređujemo analogno prethodnom slučaju — svaka dva zadatka sa istog nivo mogu se istovremeno izvršavati jer nema zavisnosti među njima. Kako sada imamo 4 elementa, nakon obrade listova, zadaci sa drugog nivoa mogu se svi istovremeno izvršavati na svim procesnim elementima. Nakon završetka njihove obrade, sledi obrada korena. Dakle obrada u ovom slučaju traje $2s+0.2s+0.2s+0.2s = 2.6s$.

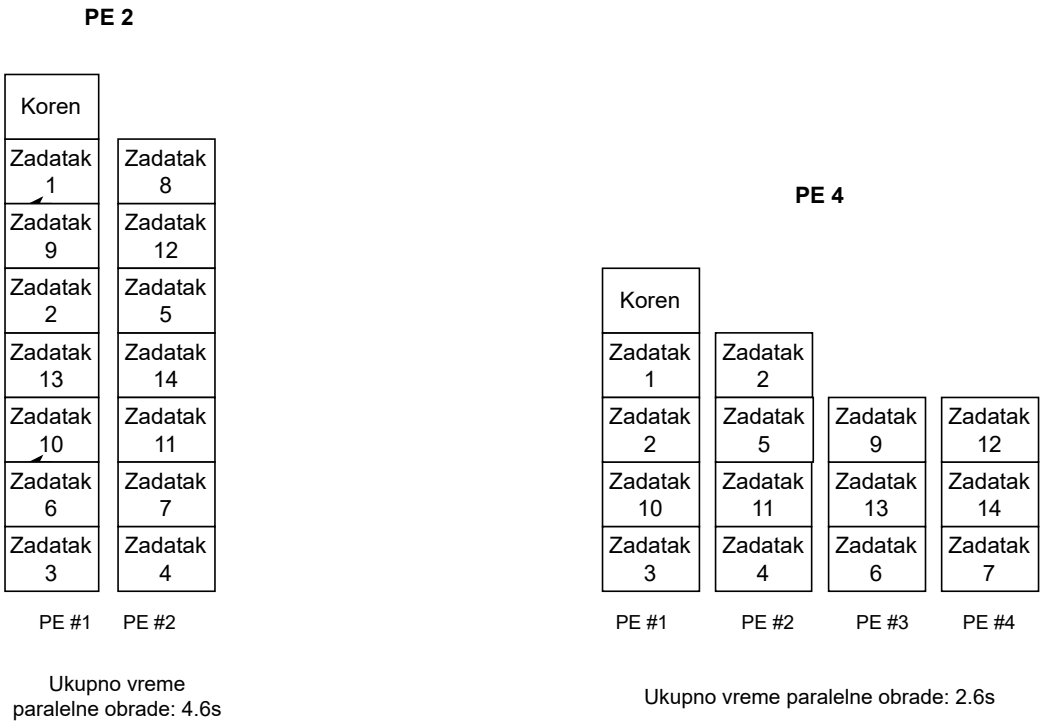
Ako imamo 8 dostupnih procesnih elemenata u mreži, možemo sve zadatke iz listova odjednom poslati na paralelnu obradu. Dakle, njihova obrada će u ovom slučaju biti srazmerna obradi jednog zadatka — $1s$. Dalje, ostale čvorove drugog nivoa možemo istovremeno izvršavati — $0.2s$ traje njihova paralelna obrada. Ali, iako ima dostupnih procesnih elemenata, obrada drugog nivoa mora čekati da se onaj nivo ispod njega završi, pa da krene sa svojim izvršavanjem — dakle u ovom slučaju je situacija kao u prethodnom $1s + 0.2s + 0.2s + 0.2s = 1.6s$.

Postojanje 16 procesnih elemenata zbog zavisnosti ne doprinosi značajnom ubrzanju, jer je 8 procesnih elemenata dovoljno za postizanje optimalnog paralelizma. Dakle, i ovde će obrada biti $1.6s$. U slučaju većeg broja zadataka, svakako bi postojanje 16 procesnih elemenata dovelo do značajnog ubrzanja.

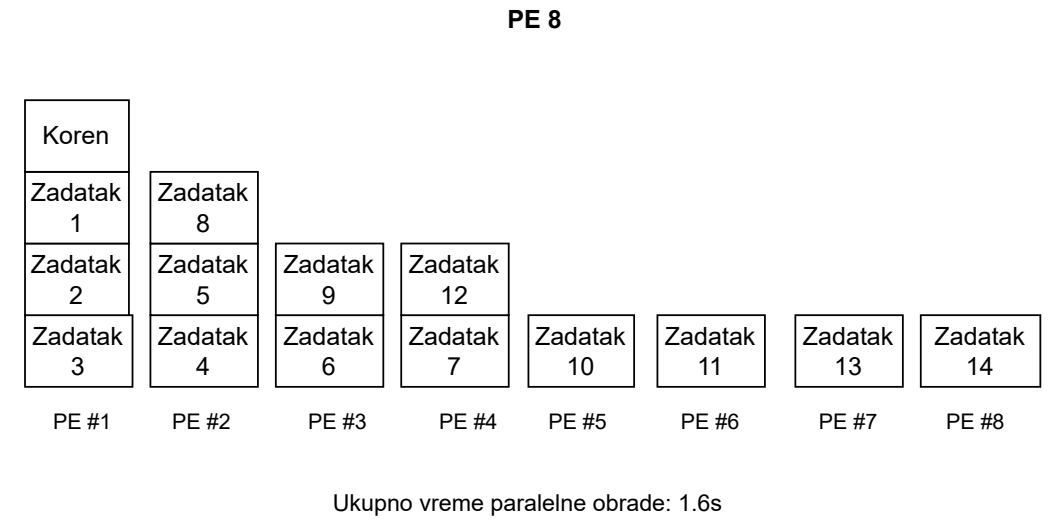
Podsetimo se opet priče o kritičnoj putanji. Sa ovog grafa vidimo da je njena vrednost jednaka 3. Rečeno je da je vreme paralelne obrade srazmerno vrednosti kritične putanje, ne može biti manje od visine stabla zbog zavisnosti u izvršavanju čvorova. Ako znamo da obrada listova traje $1s$, a čvorova $0.2s$, maksimalna brzina obrade bi bila $1.6s$, što je uzrokovano vrednošću 3 kritične putanje.



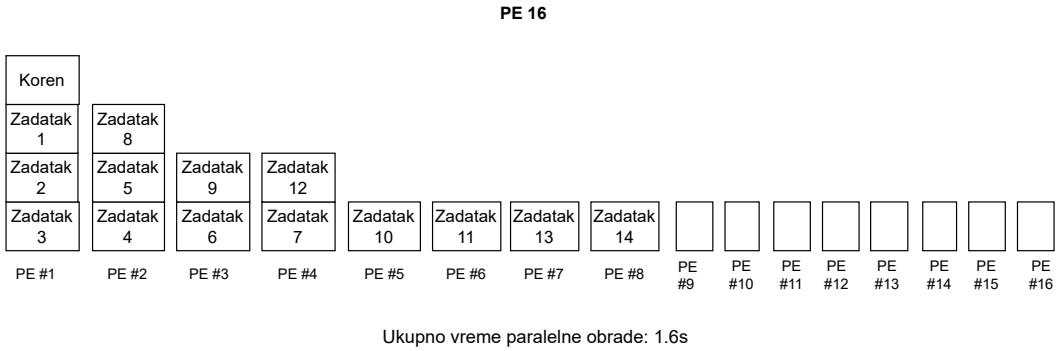
Slika 3.8: Graf zavisnih zadataka — druga klasa problema



Slika 3.9: Raspoređivanje zadataka na dva i četiri procesna elementa



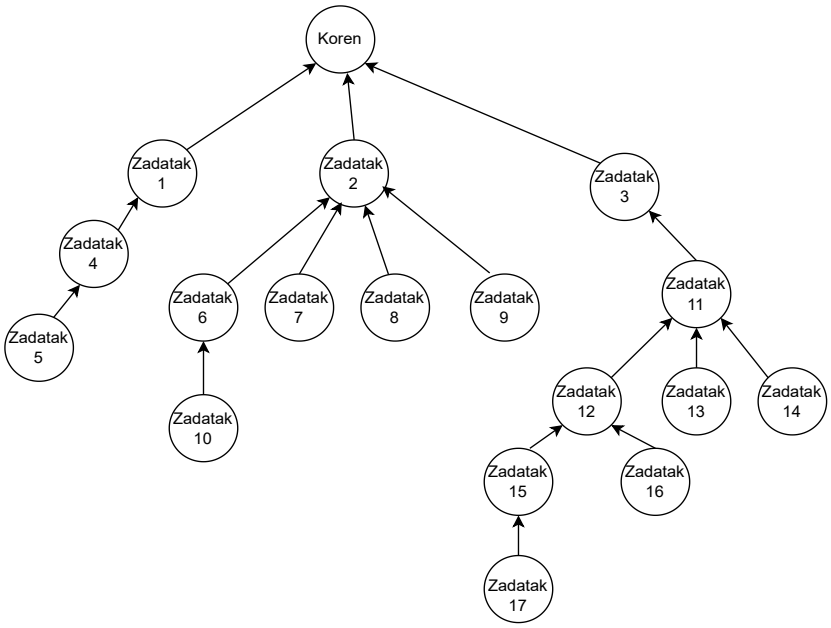
Slika 3.10: Raspoređivanje zadataka na osam procesnih elementa



Slika 3.11: Raspoređivanje zadataka na šesnaest procesnih elementa

3.2.5 Problem računanja histograma

Kao još jedan predstavnik druge klase problema može se uzeti primer nebalansiranog stabla za razliku od balansiranog binarnog koje je analizirano u prethodnom slučaju. Neka je sintetički graf koji će biti proučavan dat na slici 3.12.



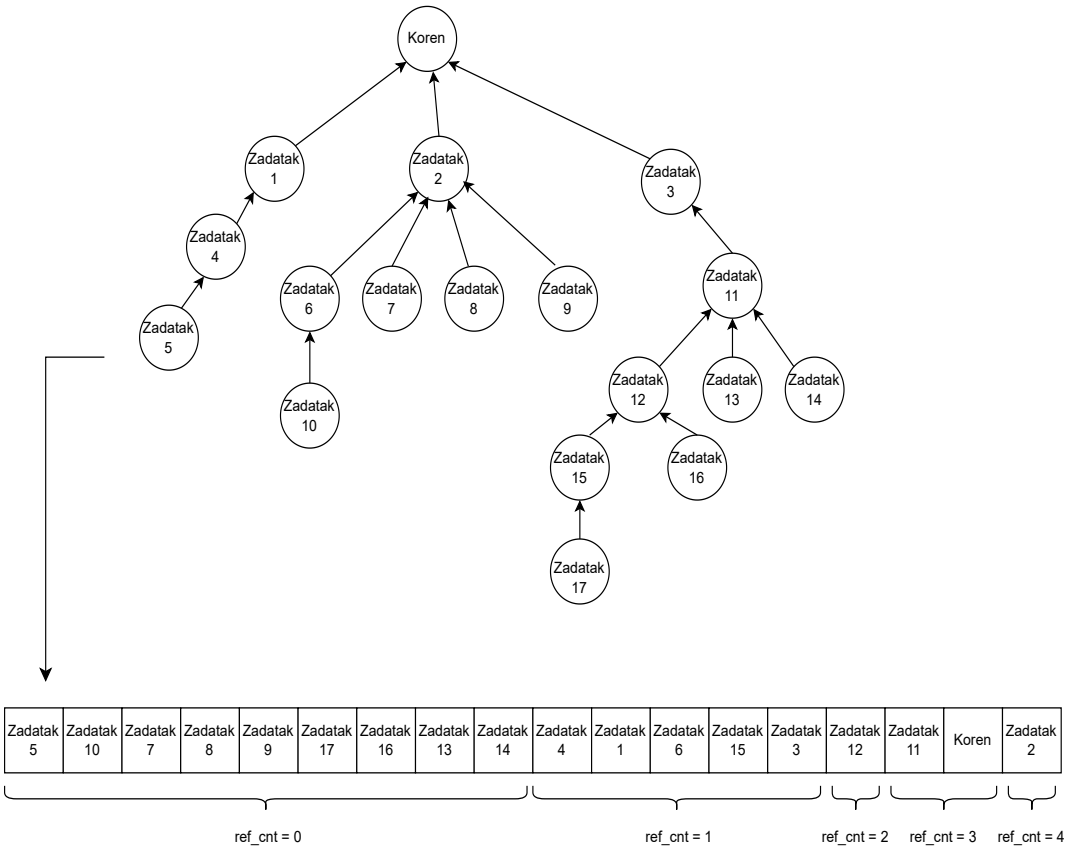
Slika 3.12: Sintetički graf zavisnih zadataka — druga klasa problema

Graf je veštački, ručno napravljen, kako bi na najbolji način predstavio što više mogućih različitih međusobnih zavisnosti među zadacima. Obrada koju zadaci obavljaju je izabrana tako da bude kompleksnija od one iz prethodnih primera. Obrada zadataka na elementima distribuirane mreže u ovom primeru podrazumeva računanje histograma na način prikazan ranije algoritmom 3. Sama obrada je proizvoljno odabrana s obzirom na samu prirodu grafa i način na koji je formiran. Zavisnost izvršavanja zadataka u predstavljenom grafu je određena na sledeći način: rezultati listova su vraćani unutrašnjim čvorovima koji su taj rezultat koristili u svojoj obradi pri računanju histograma: u opisanom algoritmu je vrednost histograma (vrednost ključa mape) za element trenutne obrade uvećavana dodatno i vrednošću histograma (vrednošću ključa) izračunatom u deci onog čvora koji se obrađuje. Podsetimo se da je sama priroda grafa takva da je cilj pokazati mogućnost paralelizacije ovakve vrste grafova zavisnosti i videti kakvo ubrzanje može da se ostvari paralelizacijom izvršavanja zadataka čija zavisnost odgovara prikazanom grafu koji nije u vidu balansiranog stabla.

Sam proces raspoređivanja zadataka iz datog grafa je isti kao i u prethodnim slučajevima. Svi elementi čija vrednost brojača referenci dostigne nulu mogu biti poslati na obradu na procesne elemente distribuiranog sistema. Oni elementi sa početnom vrednošću brojača referenci većom od nule moraju čekati završetak izvr-

šavanja obrade svih direktnih potomaka kako bi započeli svoju obradu tj. dok njihov brojač referenci ne postane nula. Takođe, oni koriste povratne vrednosti *histogram* svoje dece iz datog algoritma koje dodatno dodaju zbiru dobijenom tokom računanja frekvencije trenutnog elementa obrade. Kako je graf sintetički napravljen, ovo je samo jedan način da se prikaže zavisnost u izvršavanju među zadacima. S obzirom na veći broj međusobnih zavisnosti, čekanje na izvršenje za pojedine elemente grafa će biti veće nego u prethodnim primerima.

Prethodno opisano raspoređivanje čvorova konstruisanog grafa u red za izvršavanje na procesnim elemenatima prema vrednosti brojača referenci (*ref_ cnt*) prikazano je na slici 3.13.



Slika 3.13: Raspoređivanje čvorova grafa u red

Zbog međusobnih zavisnosti, čvorovi bliži korenu moraju čekati završetak obrade nižih nivoa u grafu od kojih zavise da bi započeli svoji obradu. To dovodi do povećanja vremena čekanja na izvršenje. Kako je već i istaknuto, visina stabla ima uticaja na vreme izvršavanja. Kako imamo 9 listova koji se mogu izvršavati para-

lelno, to za ovaj nivo obrade veći broj procesnih elemenata ima uticaja na ubrzanje vremena obrade paralelizacijom.

Na drugom nivou, sa početnom vrednošću od 1 kao vrednosti brojača referenci imamo 5 čvora. Nakon obrade listova, 3 takva čvora (Zadatak 4, Zadatak 6, Zadatak 15) mogu započeti svoju obradu. Ovde je maksimalan broj elemenata koji se mogu izvršiti paralelno jednak 3 bez obzira na broj dostupnih procesnih elemenata.

U sledećem koraku broj čvorova čija je vrednost brojača referenci postala nula je 3 (Zadatak 1, Zadatak 2, Zadatak 12) i oni se mogu paralelno izvršavati. Ovim je obrada prva dva podstabla korena završena, ali je potrebno čekati završetak obrade trećeg podstabla da bi se dobio rezultat početnog problema.

Treće podstablo je još dodatno potrebno izvršiti u dva sekvencijalna koraka. Na kraju, koren koristi rezultate obrade direktnih potomaka i vraća konačan rezultat obrade.

Glava 4

Programska implementacija rešenja

Na osnovu koncepta rešenja iznetog u prethodnom poglavlju, urađena je i programska implementacija rešenja. Implementacija rešenja je urađena u programskom jeziku *C++*, a kao razvojno okruženje je korišćen *Visual Studio 2019*. Implementacioni detalji dostupni su u [1]. Osnovni koraci implementacije obuhvataju sledeće:

1. Realizaciju klijent-server soket komunikacije unutar disribuirane mreže
2. Klase za predstavljanje opisanih problema i zadataka
3. Funkciju za formiranje usmerenog acikličnog grafa zavisnosti
4. Funkciju raspoređivanja zadataka na elemente distribuiranog sistema

4.1 Klijent-server komunikacija

Za implementaciju klijent-server komunikacije korišćena je *Windows Socket* biblioteka, tačnije, API koji omogućava implementaciju odgovarajućih interfejsa za klijentsku i serversku stranu unutar mreže. Korišćen je TCP/IP protokol zbog svoje sigurnosti za tačnu isporuku podataka u redosledu kojim su oni i poslati. Takođe, ovaj protokol upravlja i brzinom prenosa podataka kako bi se izbeglo zagušenje mreže i pruža sekvencijalnost u primanju podataka — podaci stižu u istom redosledu u kom su i poslati [3]. Najveći nedostaci su, pak, veći troškovi (*eng. overhead*) u odnosu na ostale protokole zbog dodatnih provera i potvrda, kao i više vremena za uspostavljanje veze i prenos podataka.

Server u implementaciji rešenja predstavlja jedan element distribuiranog sistema koji razrađuje logiku raspoređivača. Naime, u okviru server strane je implementirano

formiranje grafa zavisnosti zadataka dobijenih podelom datog problema, pravljenje reda na osnovu grafa, kao i samo raspoređivanje zadataka iz reda na procesne elemente.

Klijenti predstavljaju ostale procesne elemente mreže koji vrše obradu dobijenih zadataka — u našem slučaju svaki klijent je izvršavao serijsku obradu sabiranja elemenata dobijenog podniza, traženja najmanjeg elementa prosleđenog zadatka ili računanja vrednosti histograma dodeljenog podniza i vrednost dobijenu izračunavanjem vraćao kao rezultat serveru [5]. Takođe, zbog velikih troškova mreže prilikom prenosa velike količine podataka, u implementaciji svi podaci za obradu postoje na klijentskim mašinama. Prosleđuju se opsezi nizova koje treba obraditi na svakoj mašini iz originalnog niza, a rezultat obrade se vraća mrežom. Osnovi koraci za korišćenje TCP protokola u *Winsock* API-ju su:

- *Inicijalizacija Winsock-a* prikazana listingom 4.1
- *Kreiranje socket-a* prikazano listingom 4.2
- *Povezivanje klijenta na server* prikazano listingom 4.3
- *Server — slušanje i prihvatanje veza sa klijentima* prikazano listingom 4.4
- *Slanje i primanje podataka* prikazano listingom 4.5

Listing 4.1: Inicijalizacija *Winsock*-a

```
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult != 0) {
  std::cout << "WSAStartup failed with error: " <<
  iResult << std::endl;
  exit(1);
}
```

Listing 4.2: Kreiranje *socket*-a

```
ConnectSocket = socket(ipv4_addr, tcp_socket,
IPPROTO_TCP);
if (ConnectSocket == INVALID_SOCKET) {
    std::cout << "socket failed with error: " <<
    WSAGetLastError() << std::endl;
    WSACleanup();
    exit(1);
}
```

Listing 4.3: Povezivanje klijenta na server

```
iResult = connect(ConnectSocket, socket_addr,
sizeof(socket_addr));
if (iResult == SOCKET_ERROR) {
    closesocket(ConnectSocket);
    ConnectSocket = INVALID_SOCKET;
    return 1;
}
```

Listing 4.4: Server — slušanje i prihvatanje veza sa klijentima

```
SOCKET ListenSocket = socket(ipv4_addr, tcp_socket,
IPPROTO_TCP);
bind(ListenSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr));
listen(ListenSocket, SOMAXCONN);

SOCKET ClientSocket = accept(ListenSocket, NULL, NULL);
```

Listing 4.5: Slanje i primanje podataka

```
const char *sendbuf = "SEND BUFFER";
send(ConnectSocket, sendbuf, strlen(sendbuf), 0);

char recvbuf[n];
int recvbuflen = n;
int iResult = recv( ConnectSocket, recvbuf, recvbuflen, 0 );
```

4.2 Implementacija klasa za opisivanje problema

Klase za opisivanje datih problema napravljene su tako da odgovaraju elementima neophodnim za formiranje grafa zavisnosti, tj. zadacima od kojih je graf i sačinjen. Dakle, napisana je bazna klasa `Task` deklarisana kao *virtual* koju nasleđuju dve izvedene klase za opisivanje listova i čvorova (zajedno sa korenom) stabla — `LeafTask` i `RootTask`. Date klase nasleđuju interfejs i implementaciju od bazne, ali i implementiraju sve njene apstratkne metode. Takođe, implementirani su i određeni nabrojivi tj. *enum* tipovi za potrebe definisanja neophodnih polja klasa. Implementirana je i klasa za opisivanje procesnog elementa, tj. klijenta `ProcessElement` koja kao primarna polja sadrži indeks koji jedinstveno određuje svaki procesni element i red zadataka koji će se izvršavati na njemu. Osnovne javne metode i privatna polja datih klasa su predstavljeni listinzima 4.6 i 4.7.

Listing 4.6: Enumi i klasa za opisivanje zadataka

```
1  // Enum za opisivanje stanja zadatka u procesu raspoređivanja na
       klijente mreže
2  enum class TaskState : uint8_t
3  {
4      not_scheduled = 0,
5      scheduled = 1,
6      ready_for_schedule = 2,
7      processed = 3
8
9  };
10 // Enum za opisivanje tipa zadatka
11 enum class TaskType : uint8_t
12 {
13     leaf = 0,
14     root = 1
15 };
16 class Task_Class
17 {
18 public:
19     // Konstruktor klasa
20     Task_Constructor(int begin, int end, TaskState s, Task* parent,
           TaskType int ref_cnt);
21     // Destruktor klasa
22     ~Task_Destruktor(){};
23     // Metoda za izračunavanje datog zadatka
24     void execute() = 0;
25     // Metoda za vraćanje konačnog rezultata problema
26     auto get_result();
27     // Metoda za dodavanje dece svakom unutrašnjem čvoru grafa
28     void addChilden(Task *child);
29     // Metoda za uvećavanje brojača referenci
30     void incrementRefCnt();
31     // Set metode za privatna polja klasa
32     void setPrivateField(auto t);
33     // Get metode za privatna polja klasa
34     auto getPrivateField();
35 private:
36     auto result{0};
37     // Pozicija početka podniza zadatka u početnom nizu
38     int begin;
39     // Pozicija kraja podniza zadatka u početnom nizu
```

```
40      int end;
41      // Stanje zadatka
42      TaskState state;
43      // Tip zadatka
44      TaskType type;
45      // Vrednost brojača referenci
46      int ref_cnt;
47      // Referenca ka roditelju datog čvora
48      Task *parent;
49      // Vektor dece datog čvora
50      std::vector<Task *> children = {};
51       // Indeks procesnog elementa na kome se dati zadatak izvršava
52      int index;
53  }
```

Listing 4.7: Klasa za opisivanje procesnih elemenata

```
1   class ProcessElement
2   {
3   public:
4       // Konstruktor klasa
5       ProcessElement(int indeks);
6       // Destruktor klasa
7       ~ProcessElement(){};
8       // Set metode za privatna polja klasa
9       void setPrivateField(auto t);
10      // Get metode za privatna polja klasa
11      auto getPrivateField();
12  private:
13       // Indeks procesnog elementa
14      int index;
15      // Red zadataka koji se izvršavaju na datom procesnim elementu
16       std::queue<Task*> process_queue{};
17  }
```

4.3 Formiranje grafa zavisnosti

Programska realizacija grafa počiva na strategiji *podeli pa vladaj* i rekurzivnim pozivima. Ulazni niz podataka, koristeći ovu metodu, delimo na manje podnizove (zadatke) sve dok se ne zadovolji unapred definisan uslov (npr. dok podniz ne dostigne dužinu jednaku stepenu granulacije) koji predstavlja izlaz iz rekurzije. Tada

formiramo listove grafa (instance klase `LeafTask`), ažuriramo referencu prema odgovarajućem roditelju i uvećavamo njegov brojač referenci. Inače, pravimo unutrašnje čvorove grafa, ažuriramo reference ka roditeljima, uvećavamo njegov brojač, postavljamo novog roditelja za dalje čvorove. Dalje se rekurzivno pozivaju funkcije za obradu dveju polovina dobijenog podniza. Pseudokod predstavljen je algoritmom 4.

Algoritam 4 formirajGraf

```
1: Ulaz: početak, kraj, roditelj, uslov izlaska iz rekurzije M
2: if (kraj - početak) == 0 then
3:     return
4: end if
5: if ispunjen uslov M then
6:     Napravi podniz odgovarajuće dužine
7:     Postavi indekse početak i kraj za dati formirani podniz
8:     Kreiraj LeafTask objekat
9:     Postavi referencu na roditelja kreiranog objekta
10:    Uvećaj brojač referenci roditelja
11:    return
12: else
13:    m = (početak + kraj) / 2
14:    Kreiraj RootTask objekat
15:    Postavi referencu na roditelja kreiranog objekta
16:    Uvećaj brojač referenci roditelja
17:    Ažuriraj vrednost novog roditelja
18:    Call formirajGraf(početak, m, roditelj, M)
19:    Call formirajGraf(m, kraj, roditelj, M)
20:    return
21: end if
```

4.4 Raspoređivanje zadataka

Raspoređivanje zadataka podrazumeva postupak uzimanja zadataka iz reda svih zadataka sortiranih prema vrednosti brojača referenci i slanje na obradu na procesne elemente distribuirane mreže u tačno određenom redosledu prema međusobnoj zavisnosti. Procesni elementi su jedinstveno određeni odgovarajućim indeksima. Naime, neophodno je planiranje *(eng. scheduling)* redosleda slanja podataka na obradu i primanja istih nakon njihovog izvršenja.

Svi zadaci su na početku inicijalizovani na stanje `not_scheduled` tj. nisu planirani da se izvršavaju i dodeljen im je indeks procesnog elementa sistema na koji će biti izvršavani kad se steknu uslovi za njihovu obradu. Oni zadaci čija je vrednost brojača referenci jedanka nuli mogu biti poslati na izvršenje i prelaze u stanje `ready_for_scheduled`. Zadaci čija je vrednost brojača različita od nule zavisni su od drugih zadataka i moraju čekati na njihovo izvršenje. Kad se zadatak pošalje na obradu na procesni element, prelazi u stanje `scheduled`. U zavisnosti od broja dostupnih procesnih elemenata, svi zadaci sa brojačem referenci jednakim nuli mogu istovremeno biti (paralelno) poslati na obradu. Ako ima manje procesnih elemenata, šalje se onoliko zadataka koliko je dostupnih procesnih elemenata. Server, dakle, putem soket komunikacije šalje zadatke na obradu klijentima sistema koji paralelno rade.

Klijenti vrše serijsku obradu dobijenog zadatka (sabiranje elemenata, nalaženje najmanjeg elementa, računanje histograma) i vraćaju rezultat svoje obrade serveru. Server čeka odgovor od prvog klijenta koji se obradi i prihvata ga. Rezultat čuva i koristi za naredna izračunavanja. Tada i stanje zadatka postaje `processed`, on se skida iz početnog reda zadataka za raspoređivanje, vrednost brojača referenci njegovog roditelja se umanjuje za jedan. Naredni zadatak čije je stanje `ready_for_scheduled` server šalje na izvršenje na prvi slobodan procesni element. Proces se nastavlja sve dok u redu ne ostane samo jedan zadatak koji će biti koren grafa. Tada će biti poznati i rezultati među-obrade svih ostalih čvorova grafa i koren će iskoristiti rezultate obrade svoje dece i vratiti konačan rezultat. Pseudokod dat je algoritmom 5.

Algoritam 5 planiranjeIzvrsavanjaZadataka

```
1: Ulaz: koren, red_taskova red, broj_procesnih_elemenata n, pe_upravljač
2: Napravi vektor procesnih elemenata pe
3: for i ← 0 do n do
4:     Inicijalizuj pe[i]
5: end for
6: while red nije prazan do
7:     Uzmi element a sa početka red-a
8:     Dodaj a na poziciju pe[i]
9:     Uvećaj i za 1 po modulu n
10: end while
11: for svaki element u pe do
12:     if red element-a nije prazan then
13:         Uzmi čvor sa početka reda element-a
14:         if stanje čvora je ready_for_schedule then
15:             Pošalji element na izvršavanje na procesni element
16:             Postavi stanje čvor-a na scheduled
17:         end if
18:     end if
19: end for
20: do
21:     if dostupan odgovor sa bilo kog procesnog elementa then
22:         for svaki element u pe do
23:             if red element-a nije prazan then
24:                 if primljen odgovor za element sa procesnog elementa then
25:                     Primi podatke sa procesnog elementa u data
26:                     Sačuvaj data vrednost za dalju obradu
27:                     Uzmi čvor sa početka reda element-a
28:                     Postavi stanje čvor-a na processed
29:                     Umanji vrednost ref_counter roditelja čvor-a
30:                     Skini čvor iz reda element-a
31:                 end if
32:             end if
33:         end for
34:     end if
35:     for svaki element u pe do
36:         if red element-a nije prazan then
37:             Uzmi čvor sa početka reda element-a
38:             if stanje čvora je ready_for_schedule then
39:                 Pošalji element na izvršavanje na procesni element
40:                 Postavi stanje čvor-a na scheduled
41:             end if
42:         end if
43:     end for
44: while stanje koren-a nije processed
```

Glava 5

Evaluacija rezultata

U nastavku će biti predstavljeni rezultati merenja vremena izvršavanja serijske i paralelne obrade posmatranih problema. Eksperimentalna izračunavanja su urađena u distribuiranoj mreži sačinjenoj od najviše 16 povezanih računara (procesnih elemenata). Računari imaju sledeće specifikacije:

- Operativni sistem: Windows 10 Pro
- Procesor: 11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GH
- RAM memorija: 64.0 GB

Treba istakuti da su merenja za serijsku obradu davala približno iste rezultate za vremena izvršavanja problema, ma koliko puta bila pokrenuta.

Kod paralelne obrade ti rezultati su neretko varirali zbog uticaja same mreže, ali i procesa koji su istovremeno izvršavani na mašinama za obradu. Vreme obrade celokupnog problema u distribuiranom sistemu praktično obuhvata, pre svega, sama izračunavanja na procesnim elementima koji su klijentska strana komunikacije i dodatne obrade na serverskoj strani, ali i vreme slanja i primanja podataka mrežom.

Kada je u pitanju mreža računara, sama komunikacija ima neki trošak. To su indirektni troškovi koji nisu deo obrade, ali se tiču samog prenosa informacija unutar mreže. Sam TCP/IP protokol koji je korišćen u implementaciji ima značajan trošak zbog obezbeđivanja pouzdanosti slanja podataka.

Na vreme imaju uticaj i drugi procesi koji se odvijaju na računarima, a koji utiču da obrada bude sporija. Kako broj zadataka direktno utiče na to koliko puta će se podaci slati i primati u mreži, tada će veći broj zadataka dovesti do toga da uticaj mreže bude veći i samim tim određeni trošak će se javiti. Sa druge strane,

povećanjem broja elemenata zadataka (podnizova) povećava se vreme same obrade, što dovodi do toga da mreža ima manji udeo u vremenu celokupnog izvršenja problema u distribuiranom sistemu.

5.1 Problem izračunavanja sume elemenata niza

Prva klasa problema koja je posmatrana i za koju je raspoređivač zadataka u distribuiranom sistemu najpre implementiran je klasa međusobno nezavisnih zadataka koji se svi mogu izvršavati paralelno [2]. Dakle, stablo grafa zavisnosti sadrži koren i svi ostali čvorovi su listovi.

Posmatrana su dva celobrojna niza različite dužine. Broj elemenata nizova je 4294967296 i 8589934592 (skraćeno 4G i 8G). Stepen granulacije je odredio dužinu zadataka, tj. podnizova koji su se izvršavali paralelno na procesnim elementima. Ako k predstavlja stepen granulacije, tj. dužinu svakog podniza početnog niza dužine n, to je onda broj zadataka jednak n/k. Dakle, broj zadataka je obrnuto srazmeran stepenu granulacije. Paralelna obrada je vršena na 2, 4, 8 i 16 procesnih elemenata za sve veličine početnih nizova i njihovih podnizova.

Prvi posmatran slučaj je za niz od 4G elemenata. Stepen granulacije uzima tri različite vrednosti: 16777216, 8388608 i 4194304 (skraćeno 16M, 8M i 4M). Broj zadataka, je respektivno, 256, 512, 1024.

Najpre je izvršena serijska obrada problema za svaki od tri slučaja, tj. obrada na jednom procesnom elementu. U delu o sekvencijanoj obradi, istaknuto je da je vreme izvršavanja srazmerno veličini niza, i da za jako velike nizove podataka ovakvo izvršavanje nije efikasno jer uzima dosta vremena za izvršavanje. To je praktično i pokazano. Serijska merenja su vršena pet puta za svaki stepen granulacije. Odstupanja pri različitim merenjima nisu velika. Vreme izvršavanja serijske obrade je predstavljeno prvim redom tabele 5.1.

Kako su svi zadaci međusobno nezavisni, paralelna obrada se vršila na svim dostupnim procesnim elementima za dato izračunavanje i nije bilo međusobnih blokiranja i čekanja. Rezultati merenja vremena izvršenja za sva tri različita slučaja su prikazani u tabeli 5.1. Redovi tabele predstavljaju broj procesnih elemenata za konkretan slučaj merenja, dok su kolonama predstavljeni različiti stepeni granulacije za niz veličine 4G. Dato je prosečno vreme izvršenja u milisekundama dobijeno na osnovu osam merenja.

Drugi posmatran slučaj je niz od 8G elemenata. Kako bi broj zadataka ostao isti

Tabela 5.1: Tabela vremena izvršenja obrada niza dužine 4G sa odgovarajućim brojem procesnih elemenata i stepenom granulacije u milisekundama

	16M	**8M**	**4M**
PE 1	49200	49186,4	49176
PE 2	25382,3	26218,3	33200
PE 4	13263,2	13591,38	19217
PE 8	7299,5	7638,125	10097,43
PE 16	4303,5	4648,375	6193,817

Tabela 5.2: Tabela vremena izvršenja obrada niza dužine 8G sa odgovarajućim brojem procesnih elemenata i stepenom granulacije u milisekundama

	32M	**16M**	**8M**
PE 1	97500	97400	97432
PE 2	49200,7	49375,4	51594,56
PE 4	24900	26231	27577,1
PE 8	13663	14003,7	16288,13
PE 16	7371,7	8023,6	9632,7

zarad uporedive analize, stepeni granulacije su postavljeni na vrednosti: 335554432 (skraćeno 32M), 16M i 8M. Merenja su obavljena analogno kao i u prethodnom slučaju i rezultati su predstavljeni tabelom 5.2.

Na osnovu podataka iz tabele, uočavamo, najpre, da je serijsko izvršavanje niza od 8G elemenata duplo sporije od onog od 4G, što je i očekivano s obzirom na njihove dužine ako znamo čemu je, teorijski, srazmerno vreme izvršenja. Dalje, dobijeni rezultati prikazuju da se vreme izvršavanja na više procesnih elementa smanjilo u odnosu na serijsku obradu. Ta razlika je vidljiva već u slučaju dva procesna elementa. Dakle, kod velikih nizova (onih čija je dužina nekoliko desetina ili stotina hiljada elemenata) paralelna obrada na dva ili više elementa distribuiranog sistema je daleko efikasnija od njihovog serijskog izvršavanja. Takođe, dobijeni rezultati prikazuju da se i vreme izvršavanja paralelne obrade skoro linearno smanjuje porastom broja procesnih elemenata.

Setimo se primera 1 koji je dao teorijski prikaz optimalnog paralelnog izvršavanja zadataka na procesnim elementima. On pokazuje da bi za ovu klasu problema, vreme izvršavanja trebalo biti onoliko puta manje od serijskog na koliko se procesnih elemenata izvršava.

Prisetimo se, takođe, priče u uvodnom delu poglavlja 5 o tome šta utiče na

Tabela 5.3: Tabela ubrzanja paralelne obrade niza veličine 4G sa odgovarajućim brojem procesnih elemenata i stepenom granulacije

	16M	**8M**	**4M**
PE 2	1,93	1,87	1,48
PE 4	3,70	3,61	2,55
PE 8	6,74	6,43	4,87
PE 16	11,43	10,58	7,93

Tabela 5.4: Tabela ubrzanja paralelne obrade niza veličine 8G sa odgovarajućim brojem procesnih elemenata i stepenom granulacije

	32M	**16M**	**8M**
PE 2	1,98	1,97	1,88
PE 4	3,92	3,71	3,53
PE 8	7,14	6,95	5,98
PE 16	13,23	12,14	10,11

vreme obrade jednog problema u distribuiranom sistemu. Rečeno je tada kako broj zadataka može da povećava uticaj mreže na posmatrana izračunavanja, ali i kako povećanje broja elemenata u zadacima povećava uticaj same obrade na procesnim elementima u odnosu na mrežni uticaj.

U našem praktičnom merenju uticaj oba data faktora je proučavan — za istu veličinu podniza broj zadataka je bio različit, a, takođe, za isti broj zadataka, posmatrane su različite vrednosti veličine podnizova tj. samih zadataka. Uticaj ćemo videti kroz dobijeno ubrzanje. Naime, ubrzanje možemo predstaviti kao odnos vremena potrebnog za serijsko u odnosu na paralelno izvršavanje na procesnim elementima. Za svaki broj procesnih elemenata ubrzanje se menja. Teorijski, ubrzanje bi trebalo biti srazmerno broju procesnih elemenata na kojima se obrada vrši. U tabelama 5.3 i 5.4 je dato ubrzanje za dobijene rezultate obrade.

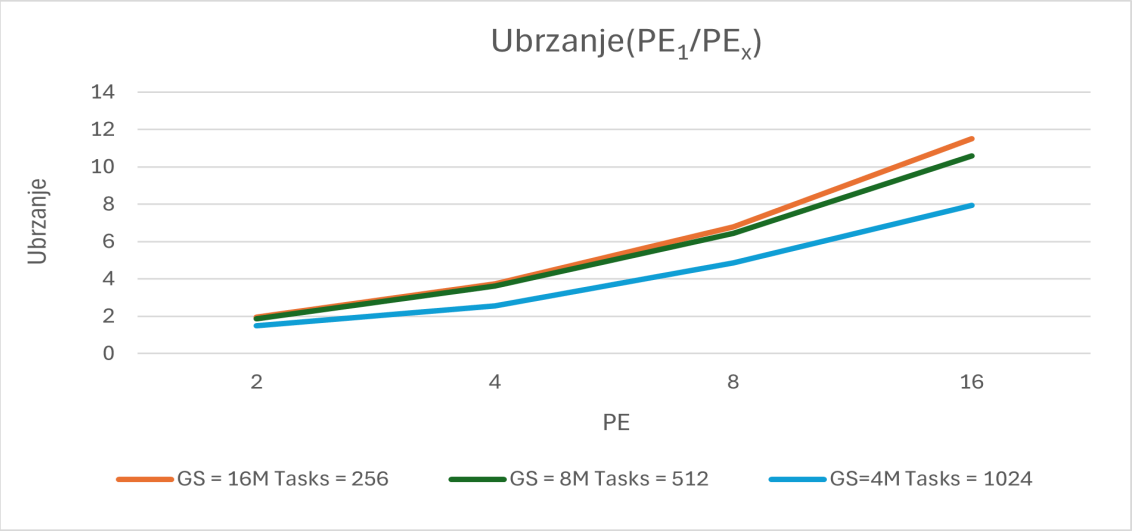
U prikazanim tabelama, broj zadataka za najveći stepen granulacije tj. veličine podniza, je najmanji što predstavlja prve kolone obe tabele — broj zadataka je jednak 256. Središnja kolona određuje sa datim stepenima granulacije postojanje 512 zadataka za posmatrane slučajeve, dok poslednjoj koloni odgovara 1024 zadataka jer je njihova dužina najmanja, pa se shodno tome i sam broj zadataka povećava.

Vidimo da u istim tabelama i u istim redovima ubrzanje opada povećanjem broja zadataka, a smanjenjem veličine niza. Dakle, u tom slučaju mreža ima značajan trošak i uticaj na rezultate. Takođe, ako posmatramo isti broj zadataka — iste

kolone u različitim tabelama gde je, pak, veličina podniza koji se obrađuje različita, uočavamo da je ubrzanje veće sa većim stepenom granulacije. Dakle, sama obrada je imala veći uticaj. Ono što je zajedničko jeste da je najveće ubrzanje ostvareno u slučaju najmanjeg broja zadataka, a najvećeg stepena granulacije, kao i analogno tome, ono najlošije.

Zbog svih uticaja u sistemu, praktična merenja odstupaju u određenim granicama od teorijskih, nisu direktno srazmerna broju procesnih elemeneta kako je očekivano. Ipak, jasno se vidi uvećanje brzine izvršavanja sa porastom broja procesnih elemenata. To jasno govori da je paralelna obrada, u svim prikazanim slučajevima, efikasnija od serijske.

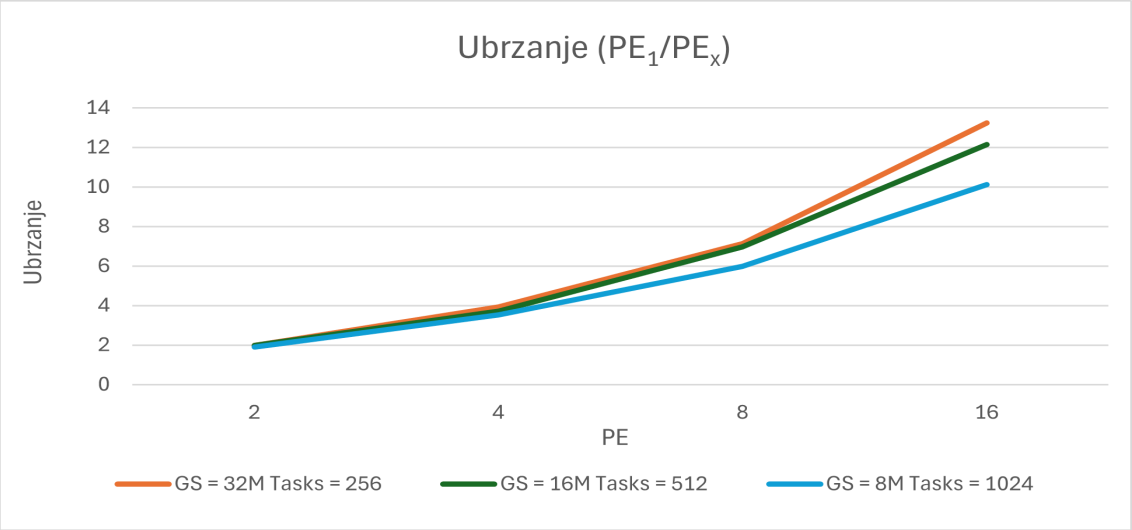
Na slikama 5.1 i 5.2 je grafički prikazano ubrzanje porastom broja procesnih elemnata. Na njima se vidi da ubrzanje raste porastom broja elemenata za obradu, tj. stepenom granulacije (GS na slikama).



Slika 5.1: Grafikon ubrzanja za niz dužine 4G

5.2 Problem nalaženja minimalnog elementa niza

Druga klasa problema koja je posmatrana i za koju je raspoređivač zadataka u distribuiranom sistemu implementiran je, kao što je već rečeno, klasa međusobno zavisnih zadataka, koji se za razliku od prethodne klase, ne mogu svi izvršavati istovremeno odjednom. Zadaci su međusobno zavisni i ta zavisnost utiče na vreme obrade celokupnog problema. Graf koji odgovara ovoj klasi je takođe stablo, ali za



Slika 5.2: Grafikon ubrzanja za niz dužine 8G

razliku od prethodne klase, visina stabla je veća zbog prirode samog problema i načina formiranja grafa koji je opisan u delu 3.2.4. U ovom slučaju, imamo binarno stablo.

Posmatrana su, kao i u prethodnom slučaju, dva celobrojna niza različite dužine. Broj elemenata nizova je isti — 4G i 8G.

Stepen granulacije tj. uslov zaustavljanja u ovom slučaju određuje dužinu zadataka u listovima. Naime, listovi imaju dužinu jednaku datom broju k, dok je dužina nizova njihovih direktnih predaka duplo veća od k. Analogno se dužina zadataka duplira prolaskom uz stablo dok se ne dođe do korena koji predstavlja početni niz. Dakle, listovi predstavljaju podnizove dužine k početnog problema dužine n, što znači da broj listova iznosi n/k.

Broj početnih zadataka koji se na početku obrade svi istovremeno mogu izvršavati na dostupnim procesnim elementima je upravo jednak broju listova — n/k. Broj zadataka se onda uz stablo, sa povećanjem dužine podnizova, smanjuje duplo, dok ne dođemo do jednog zadatka koji je koren i istovremeno početni problem. Dakle, u ovom slučaju je broj zadataka zbog zavisne obrade veći od broja zadataka za probleme prve klase kada je dužina početnog niza ista i dužina podnizova listova stabla.

Paralelna obrada je i u ovom slučaju vršena na 2, 4, 8 i 16 procesnih elemenata za sve veličine početnih nizova i njihovih podnizova.

Prvi posmatran slučaj je niz od 4G podataka. Stepen granulacije, kao kriterijum zaustavljanja za dužinu krajnjih zadataka, uzima iste tri različite vrednosti kao i

Tabela 5.5: Tabela vremena izvršenja obrada niza dužine 4G sa odgovarajućim brojem procesnih elemenata i stepenom granulacije u milisekundama

	16M	**8M**	**4M**
PE 1	47346	49219,6	49176
PE 2	26181	27634	35467
PE 4	13987,4	15400	20392,7
PE 8	7867	8345	10545
PE 16	4897	5079,45	6345

kod prve klase: 16M, 8M i 4M. Broj zadataka koji predstavljaju listove grafa je, respektivno, 256, 512, 1024. Na sledećem nivou hijerarhije taj broj se smanjuje na 128, 256, 512, i dalje na 64, 128, 256. Smanjivanjem do korena grafa ukupan broj zadataka za sva tri slučaja iznosi, respektivno, 512, 1024, 2048.

Izvršena je prvo serijska obrada problema za svaki od tri slučaja, tj. obrada na jednom procesnom elementu. U delu o sekvencijanoj obradi, istaknuto je da je vreme izvršavanja srazmerno veličini niza, i da, kao i u prvom problemu, zbog velikih nizova, takva obrada nije efikasna. To je praktično i pokazano. Serijska merenja su vršena pet puta za svaki stepen granulacije. Odstupanja pri različitim merenjima nisu velika. Takođe, razlike u vremenu obrade u odnosu na prvi problem nisu značajne jer sama obrada problema nije puno kompleksnija od one ranije. Vreme izvršavanja serijske obrade je predstavljeno prvim redom tabele 5.5. Ostala polja su popunjena merenjima na više procesnih elemenata. Sam prikaz je analogan onom za prvi problem.

Drugi posmatran slučaj je niz od 8G elemenata. Stepeni granulacije su ostavljeni na iste vrednosti kao za prvi problem kako bi broj zadataka koji su listovi grafa ostao isti: 32M, 16M i 8M. Ukupan broj zadataka je isti kao u slučaju obrade niza od 4G elemenata. Merenja su obavljena analogno kao i u prethodnom slučaju i rezultati su predstavljeni tabelom 5.6.

Tabele pokazuju, kao i u prethodnom slučaju, da je serijsko izvršavanje sporije od paralelnog već za dva procesna elementa, kao i da postoji razlika u vremenu samog serijskog izvršavanja za različite dužine početnog niza. Takođe, dobijeni rezultati prikazuju da se i vreme izvršavanja paralelne obrade skoro linearno smanjuje porastom broja procesnih elemenata.

Osnovna razlika u odnosu na prethodni problem jeste postojanje zavisnosti među zadacima. Kako je broj elemenata početnih nizova ostao isti, kao i sama obrada

Tabela 5.6: Tabela vremena izvršenja obrada niza dužine 8G sa odgovarajućim brojem procesnih elemenata i stepenom granulacije u milisekundama

	32M	**16M**	**8M**
PE 1	98453	98790	98679
PE 2	51733	54870	57675,89
PE 4	26345	28521,38	35927,44
PE 8	14534,56	15043	17334
PE 16	7645	9105	10826,25

problema, koja nije puno kompleksnija od prethodne, vidimo da postoje razlike u vremenu izvršavanja zbog većeg broja zadataka grafa uzrokovanim postojanjem zavisnosti. Kako određeni zadaci moraju da čekaju izvršenje drugih kako bi započeli svoju obradu, to se ukupno vreme izvršavanja povećava. U ovom konkretnom slučaju, listovi vrše iterativnu proveru nalaženja najmanjeg elementa datog podniza. Roditelji koriste rezultate obrade dece i vrše jednostavno poređenje dva elementa kako bi odredili koji je manji i svoj rezultat dalje propagiraju uz stablo. Dakle, njihova obrada je kraća. Međutim, sam broj takvih zadataka je nezanemarljiv i on će uticati na vreme izvršavanja.

Raspoređivanje na procesne elemente se za listove vrši kao u prethodnom slučaju — svi se mogu istovremeno izvršavati ako je dostupan dovoljan broj elemenata ili se rasporediti čim bude slobodnih za izvršavanje jer nema nikakvih preduslova za njihovu obradu. Ostali čvorovi se raspoređuju na procesne elemente kad se steknu uslovi za njihovo izvršavanje (kada se obrade zadaci koji su im direktni potomci) i šalju se u tom trenutku na prvi slobodan procesni element.

I u ovom slučaju uticaj mreže i dodatni troškovi su vidljivi, posebno jer je broj zadataka sad veći. Videli smo da se povećanjem broja zadataka uticaj mreže povećava, što dovodi do toga da je za ekvivalentne dužine nizova obrada i ubrzanje manje u ovom slučaju nego kod obrade ranije posmatranog problema prve klase. Takođe, za ovu klasu je potvrđeno, kao i ranije, da se samim smanjenjem broja zadataka za istu dužinu početnog niza vreme obrade smanjuje i da postoji razlika u vremenima obrade u zavisnosti od stepena granulacije i broja zadataka.

Ubrzanje se zbog svega pomenutog za iste dužine nizova smanjilo u odnosu na problem izračunavanja sume nizova. Takođe, uticaj povećanja broja zadataka se za različite dužine nizova za problem nalaženja najmanjeg elementa vidi i u smanjenu ubrzanja. U odnosu na teorijske iznete osnove, postojanje više zadataka je dovelo do

Tabela 5.7: Tabela ubrzanja paralelne obrade niza veličine 4G sa odgovarajućim brojem procesnih elemenata i stepenom granulacije

	16M	**8M**	**4M**
PE 2	1,84	1,78	1,35
PE 4	3,44	3,19	2,35
PE 8	6,13	5,89	4,55
PE 16	9,84	9,69	7,56

Tabela 5.8: Tabela ubrzanja paralelne obrade niza veličine 8G sa odgovarajućim brojem procesnih elemenata i stepenom granulacije

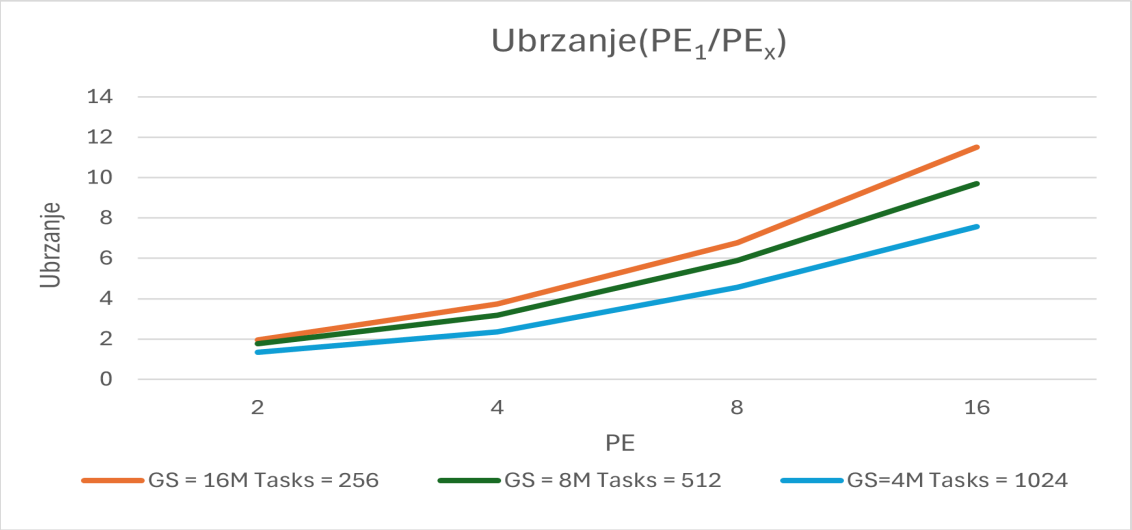
	32M	**16M**	**8M**
PE 2	1,87	1,80	1,71
PE 4	3,73	3,46	2,74
PE 8	6,77	6,56	5,69
PE 16	12,87	10,85	9,11

stvaranja razlike ubrzanja na 8 i 16 procesnih elementa. Takođe, obrada unutrašnjih čvorova je daleko brža od onog iznetog u teorijskom delu (oko 40 puta je izvršavanje unutrašnjih čvorova brže od listova u praktičnom primeru, u odnosu na korišćen odnos 5). Postignuta ubrzanja za dve dužine početnog niza date su tabelama 5.7 i 5.8.

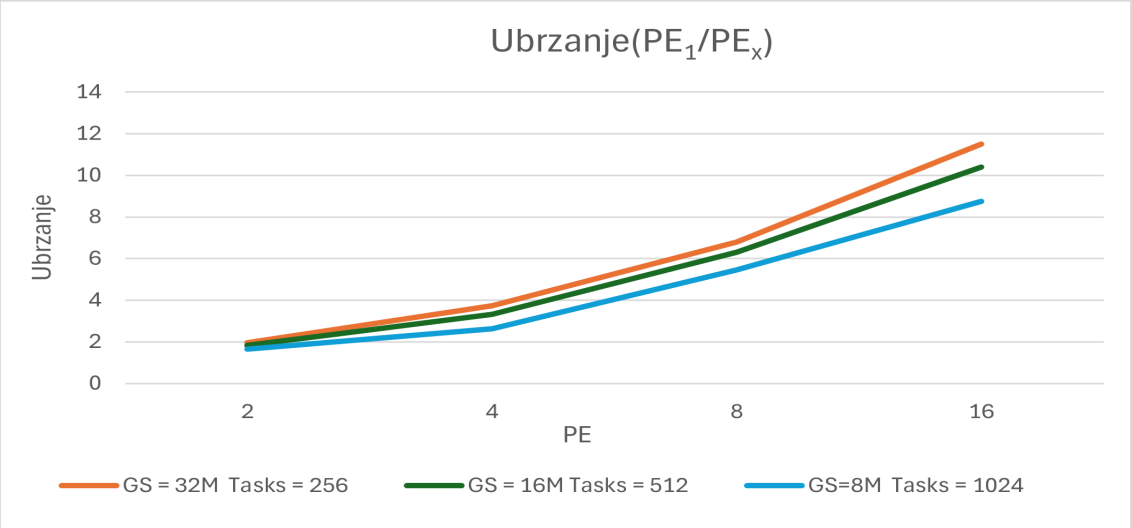
Tabele su ekvivalentne onima za prethodni slučaj, tj. kolone predstavljaju stepen granulacije za listove grafa. Na osnovu njega se može odrediti broj zadataka koji su nezavisni od ostalih i koji su jednaki onim vrednostima za ukupan broj zadataka u prethodnom primeru za problem prve klase.

Vidimo i ovde da u istim tabelama i njihovim redovima ubrzanje opada povećanjem broja zadataka, a smanjenjem veličine niza. Dakle, u tom slučaju mreža ima značajan trošak i uticaj na rezultate.

Na slikama 5.3 i 5.4 je grafički prikazano ubrzanje porastom broja procesnih elemenata. Na njima se vidi da ubrzanje raste porastom broja elemenata za obradu tj. stepenom granulacije listova (GS na slikama). Ako poredimo tabele ubrzanja za probleme obe klase, vidimo da je ubrzanje u ovom slučaju manje od prvog posmatranog problema. To je očekivano zbog postojanja zavisnosti u grafu koja su uslovila međusobno blokiranje u izvršavanju i, takođe, dala veći broj samih zadataka za obradu što sve ima uticaja na izvršavanje.



Slika 5.3: Grafikon ubrzanja za niz dužine 4G



Slika 5.4: Grafikon ubrzanja za niz dužine 8G

5.3 Problem računanja histograma

Drugi primer grafa iz druge klase problema je, kao što je ranije opisano, veštački (ručno) napravljeno nebalansirano stablo kako bi se opisao i jedan od ovakvih slučajeva upotrebe u kojima je zavisnost među zadacima raznolikija i veća nego u prethodnim slučajevima.

Dužine nizova koji su obrađivani u ovom primeru jesu u listovima grafa 2097152, 1048576 i 524288 (skraćeno 2M, 1M i 0.5M) elemenata. Ukupan broj zadataka za obradu, prema grafu, jeste 17, od kojih su 9 listovi koji nisu zavisni ni od jednog

Tabela 5.9: Tabela vremena izvršenja zadataka na osnovu sintetički napravljenog grafa sa odgovarajućim brojem procesnih elemenata i brojem elemenata nizova u milisekundama

	2M	**1M**	**0.5M**
PE 1	255467	121459	57567
PE 2	150411	72719	35429,85
PE 4	94859,8	45965,8	23305,14
PE 8	75629,4	36365,1	18092,25
PE 16	75224,2	36301,2	17828

drugog zadatka i mogu se paralelno izvršiti.

Prva grana koja polazi iz korena grafa daje podstablo gde postoji sekvencijalna zavisnost u izvršavanju čvorova jer nema mogućnosti za paralelizaciju obrade. Druga grana koja polazi iz korena daje podstablo gde je moguće izvršiti paralelizaciju i visina tog podgrafa je dva. Dakle, postoji zavisnost u izvršavanju pojednih zadataka uz one koji se i nezavisno mogu izvršiti. Treća grana koja polazi iz korena daje podstablo koje grananjem daje više podgrafova međusobno zavisnih zadataka čija je visina četiri. Ovde sekvencijalna zavisnost određenog dela podgrafa smanjuje mogućnost paralelizacije.

Merenja paralelnih izvršavanja su vršena na 2, 4, 8 i 16 procesnih elemenata u distribuiranom sistemu, kao i serijsko izvršavanje zadataka koje je podrazumevalo obradu zadataka sekvencijalno.

Setimo se raspoređivanja čvorova grafa na procesne elemente opisanog u poglavlju 3.2.5 slikom 3.13. Tamo je opisano kolika je maksimalna paralelizacija moguća u zavisnosti od broja zadataka koji se mogu paralelno izvršavati.

Broj procesnih elemenata, dakle, ima uticaja na izvršavanje listova — tu je paralelizacija efikasnija sa više procesnih elemenata. Ali na višim nivoima, ta mogućnost paralelizacije je ograničena zavisnošću zadataka i veći broj dostupnih elemenata za izvršavanje nema puno uticaja. Za izvršavanje pojedinih čvorova je potrebno čekati duže zbog većeg broja zavisnosti, tj. veće visine odgovarajućeg podstabla.

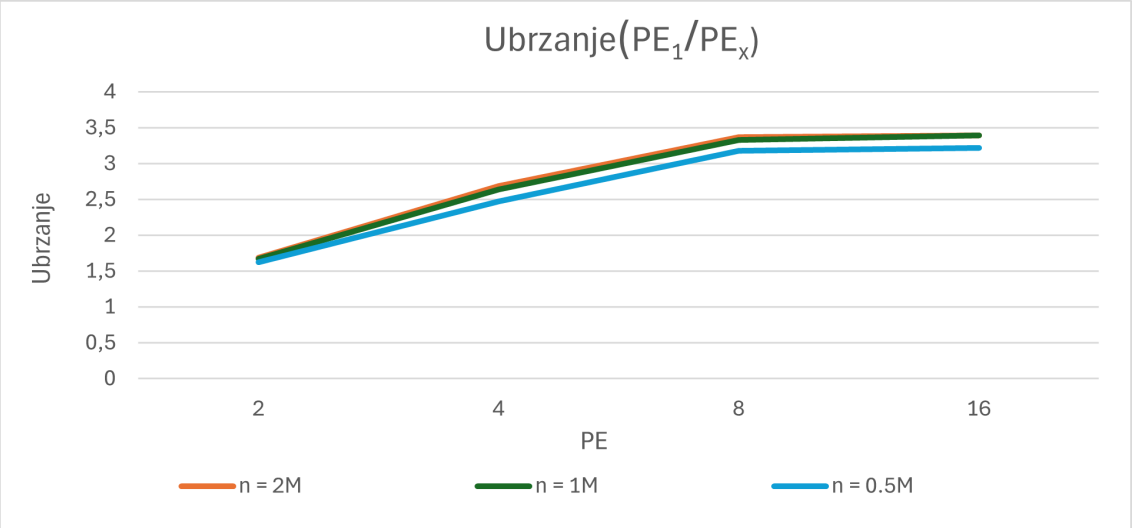
U tabeli 5.9 je prikazano vreme izvršavanja na svakom procesnom elementu za nizove pomenutih dužina.

Vrednosti ubrzanja, kao i grafikon, predstavljeni su tabelom 5.10 i slikom 5.5.

Posmatrajmo tabele vremena izvršavanja i ubrzanja datog grafa. Vidimo da postoji određeno ubrzanje paralelnim izvršavanjem u odnosu na sekvencijalno. Zbog postojanja velikog broja zavisnosti, to ubrzanje je manje u odnosu na prethodno

Tabela 5.10: Tabela ubrzanja zadataka na osnovu sintetički napravljenog grafa sa odgovarajućim brojem procesnih elemenata i brojem elemenata nizova

	2M	1M	0.5M
PE 2	1,69	1,67	1,62
PE 4	2,69	2,64	2,47
PE 8	3,37	3,33	3,18
PE 16	3,39	3,34	3,22



Slika 5.5: Grafikon ubrzanja zadataka na osnovu sintetički napravljenog grafa

posmatrane probleme. Sekvencijalna zavisnost određenih podstabala je otežavala mogućnost paralelizacije.

Samo paralelno izvršavanje na različitom broju procesnih elemenata, s druge strane, dovodi do povećanja ubrzanja porastom broja procesnih elemenata na kojima je obrada vršena. To ubrzanje je manje nego u prethodnim slučajevima grafa zavisnosti.

Prvi je slučaj, naime, predstavljao obradu svih zadataka paralelno, u drugom je zbog oblika grafa u vidu balansiranog, binarnog stabla zavisnost zadataka bila manja (jedan zadatak je direktno zavisio od obrade maksimalno dva potomka). Kako je zavisnost među zadacima u ovom primeru grafa veća (neki čvorovi zavise i od 4 direktna potomka), to je i manja mogućnost paralelizacije, ali i sama sekvencijalna zavisnost koja se javlja u pojedinim delovima grafa doprinosi tome. Više vremena se troši na čekanje za izvršavanje zadataka od kojih unutrašnji čvorovi zavise.

Ako posmatramo vreme izvršavanja, primećujemo da se ono na 8 i 16 elemen-

ta nije značajno promenilo, što znači da je sa 8 procesnih elemenata postignuta maksimalna paralelizacija koja je za ovakav graf moguća i da dalje povećanje broja elemenata u sistemu za paralelnu obradu neće uticati na smanjenje vremena izvršavanja. Teorijsko objašnjenje ovakvog ponašanja je opisano ranije u poglavlju 3.2.4 primerom 2.

Glava 6

Zaključak

Kako je u modernom računarstvu današnjice sve veća potreba za obradom kompleksnih problema sa velikom količinom podataka uz što manji utrošak vremena, to je i sve veća potreba za pronalaženjem rešenja koji bi takve zahteve ispunili. Takvi problemi se mogu izvršavati na jednom računaru, ali u brojnim prilikama kompleksnost samih problema prevazilazi mogućnosti hardvera. Zato je pristup paralelnoj obradi problema u distribuiranom sistemu jedan od čestih rešenja problema ovakve vrste u savremenom svetu. Načini za samu paralelizaciju kompleksnih problema su brojni, a u radu je izložen jedan od njih zasnovan na implementaciji raspoređivača zadataka na osnovu usmerenog acikličnog grafa zavisnosti.

U okviru rada je, najpre, izložen princip dekomponovanja kompleksnih problema na jednostavnije zadatke čije međusobne zavisnosti mogu biti predstavljene usmerenim acikličnim grafom zavisnosti. Metodom *podeli pa vladaj*, te zavisnosti mogu se predstaviti grafovima koji pripadaju dvema klasama — klasi svih međusobno nezavisnih zadataka koji se svi istovremeno mogu izvršavati ili onoj klasi gde među zadacima postoji zavisnost u izvršavanju.

Na osnovu formiranih grafova je moguće napraviti raspoređivač koji će izvršiti raspodelu zadataka, uzimajući u obzir njihove međusobne zavisnosti, na procesne elemente distribiranog sistema na takav način da ubrza obradu u odnosu na serijsko izračunavanje. Na osnovu različitih problema date dve klase i različitih usmerenih acikličnih grafova dobijenih njihovom dekompozicijom, predstavljen je način na koji raspoređivač zadatake šalje na obradu i takođe prima same rezultate izračunavanja sa različitih programskih entiteta sistema.

Bez umanjenja opštosti, a u svrhu validacije teorijskog koncepta rešenja predloženog za obe klase problema, odabrana su tri konkretna primera koja su programski

implementirana i prakično testirana.

Praktični rezultati pokazuju da je najefikasnija paralelna obrada bila za problem prve klase kod kojeg su svi zadaci međusobno nezavisni i svi se mogu, bez preduslova, izvršavati istovremeno. Ubrzanje obrade u odnosu na serijsko izvršavanje se povećavalo sa porastom broja elemenata u distribuiranom sistemu na kojima je moguće vršiti neophodna izračunavanja.

Za drugu klasu međusobno zavisnih problema raspoređivač je dao takvu raspodelu kojom je ubrzanje manje nego u prethodnom slučaju. Zbog prirode grafa, pojedini zadaci su morali da čekaju završetak obrade svojih potomaka da bi započeli svoje izvršavanje. To je uticalo na povećanje ukupnog vremena paralelnog izvršavanja i smanjenja ubrzanja u odnosu na serijsko izvršavanje. Takođe, u okviru ove klase, različite zavisnosti koje rezultiraju različitim oblicima grafova daju drugačije rezultate. Najveće ubrzanje se ostvaruje u onim slučajevima kada je graf u vidu balansiranog stabla, kao što je npr. binarno, gde izvršavanje svakog zadatka zavisi od maksimalno dva direktna potomka. Kroz dva različita primera to je u radu detaljno opisano i praktično demonstrirano.

Kako je raspodela vršena putem mrežnih protokola, to su se javljali i određeni troškovi koji su imali uticaja na praktične rezultate i vremena paralelnog izvršavanja na većem broju procesnih elemenata distribuiranog sistema. Mreža je imala najveći uticaj u onim slučajevima kada se veći broj zadataka slao na obradu na procesne elemente u odnosu na one slučajeve gde je broj zadataka bio manji, što je očekivano jer usled povećanog broja zadataka povećana je i mrežna komunikacija. Takođe, tokom eksperimentalnog testiranja, uočeno je da aktivnost mreže, koja nije direktno uzrokovana našim programom (bočni efekti), može uticati na preciznost rezultata, te je bilo potrebno vršiti nekoliko serija merenja za preciznije rezultate pošto nije bilo moguće izolovati komunikaciju sa mreže.

Pravci daljeg razvoja mogu ići u pogledu optimizacije tehničkog rešenja, u smeru korišćenja mrežnih protokola poput UDP koji će uticati na smanjenje mrežnih troškova, kao i identifikaciju potencijalnih novih problema za različite slučajeve korišćenja za koje ovaj koncept može biti primenjen.

Bibliografija

[1] Implementacija rešenja. `https://github.com/jovanaMATF/MasterRad`.

[2] Beverly A. Sanders Berna L. Massingill, Timothy G. Mattson. *Patterns for Parallel Application Programs*. ResearchGate, 2000.

[3] John Davidson. *An introduction to TCP/IP*. Springer-Verlang New York, 1988.

[4] Milena Vujošević Janičić. *Materijali sa kursa „Dizajn programskih jezika"*. Matematički fakultet Univerziteta u Beogradu. `https://www.programskijezici.matf.bg.ac.rs/dpj/2021/predavanja/konkurentno_programiranje.pdf`.

[5] Vitali Herrera Semenets Jose Kadir Febrer-Hernandez. *A Framework for Distributed Data Processing*. CENATAV 12200 Havana, Cuba.

[6] Gordon D. Logan. Parallel and serial processing. *ResearchGate*, 2002.

[7] Filip Marić. *Materijali sa kursa „Konstrukcija i analiza algoritama"*. Matematički fakultet Univerziteta u Beogradu.

[8] Siniša Nešković. *Strukture podataka i algoritmi - skripta*. Fakultet organizacionih nauka Univerziteta u Beogradu, 2019.

[9] James Reinders. *Intel Threading Building Blocks, 1st Edition*. O'Reilly Media Sebastopol, CA 9547, 2007.

[10] Seyed H. Roosta. *Parallel processing and parallel algorithms*. Springer Science New York, 2000.

[11] Vivek Sarkar. *Decomposition Techniques for Parallel Algorithms*. Department of Computer Science Rice University. `https://cs.rice.edu/~vs3/comp422/lecture-notes/comp422-lec4-s08-v1.pdf`.

[12] Ivan Stankovic. *Strukture i baze podataka - nelinearne strukture.* Prirodno-matematički fakultet u Nišu, 2007.

[13] W.T. Tutte. *Graph teory.* Cambridge University Press, 2001.

Biografija autora

Jovana Adžić je rođena 08. juna 1998. u Kruševcu. Gimnaziju opšteg smera u Aleksandrovcu je završila kao nosilac Vukove diplome i učenik generacije. Smer računarstvo i informatika na modulu matematika na Matematičkom fakultetu Univerziteta u Beogradu upisala je 2017. godine, a završila 2021. godine. Nakon toga upisuje master studije na istom smeru. Od februara 2022. godine do danas je zaposlena na poziciji Software engineer u kompaniji *TTTech Auto* u Beogradu. Projekti na kojima je radila su iz oblasti automobilske industrije i rad je zasnovan, pre svega, na programskim jezicima C++ i C.