

**Univerzitet u Beogradu
Elektrotehnièki fakultet**

Dragan Miliæev

**Optimizacija koda programskih petlji
sa uslovnim grananjima**

Magistarski rad

Mentor: prof. dr Zoran Jovanoviæ

Beograd, 1995.

Kratak sadržaj

1. Uvod	1
2. Definicija problema	3
3. Pregled postojećih rešenja	8
4. Analiza problema	23
5. Ideja predloženog rešenja	39
6. Teorijski model i njegova analiza	52
7. Model za realni slučaj i heuristike	74
8. Eksperimentalna analiza	83
9. Pravci budućeg rada	91
10. Zaključak	94

Detaljan sadržaj

Kratak sadržaj	ii
Detaljan sadržaj	iii
1. Uvod	1
1.1. Okvir rada	1
1.2. Struktura rada	2
2. Definicija problema	3
2.1. Optimizacija koda programskih petlji	3
2.2. Softverska protočnost	6
2.3. Značaj problema	7
3. Pregled postojećih rešenja	8
3.1. Teorijski rezultati	8
3.1.1. Rezultati za petlje bez uslovnih grananja	8
3.1.2. Rezultati za petlje sa uslovnim grananjima	10
3.2. Raspoređivanja koja daju konstantan interval iniciranja	15
3.2.1. Raspoređivanje po modulu za petlje bez uslovnih grananja	15
3.2.2. Raspoređivanje po modulu za petlje sa uslovnim grananjima	18
3.3. Raspoređivanje koje daje promenljiv interval iniciranja	19
4. Analiza problema	23
4.1. Model programskog koda koji se optimizuje	23
4.1.1. Model softverske protočnosti	23
4.1.2. Model operacija i zavisnosti po podacima	24
4.1.3. Model kontrolnih zavisnosti	25
4.1.4. Model iteracionih distanci	25
4.2. Izvori paralelizma	26
4.2.1. Softverska protočnost komponenti jake povezanosti	26
4.2.2. Spekulativno izvršavanje	28
4.2.3. Efekat kontrolnih zavisnosti i promenljivih iteracionih distanci	28
4.2.4. Interferencija uslova	31
4.2.5. Pomeranje operacija	32
4.2.6. Efekat ka njenja uslova zbog pomeranja operacija	35
4.2.7. Efekti transformacija na rezultujućem kodu	37
4.2.8. Zaključak analize	38
5. Ideja predložnog rešenja	39
5.1. Model predikatske softverske protočnosti (PSP)	39
5.2. Pomeranje operacija	42
5.3. Generisanje rezultujućeg koda	45
5.4. Intuitivna svojstva modela	48
5.4.1. Svojstvo te nje ka optimalnom	48
5.4.2. Mogućnost eliminacije proizvoljne predikcije	50
6. Teorijski model i njegova analiza	52
6.1. Pretpostavke i ograničenja	52

6.2. Formalna definicija PSP modela	52
6.2.1. Formalne definicije osnovnih elemenata	52
6.2.2. Definicije predikatske matrice i stanja	54
6.2.3. Definicije grafa prelaza i klastera	54
6.2.4. Definicija pomeranja operacija	55
6.2.5. Definicija PSP modela	56
6.2.6. Definicija pro irenja PSP modela	56
6.2.7. Neka osnovna svojstva PSP modela	57
6.2.8. Svojstva pro irenja PSP modela	60
6.3. Definicija i dokaz svojstva te nje ka optimalnom	61
6.3.1. Intuitivna postavka pojma i dokaza svojstva te nje ka optimalnom	61
6.3.2. Definicija svojstva te nje ka optimalnom	65
6.3.3. Definicija svojstva te nje PSP modela ka optimalnom	65
6.3.4. Dokazi pomoćnih svojstava	66
6.3.5. Dokaz svojstva te nje ka optimalnom	68
6.3.6. Alternativna postavka svojstva te nje ka optimalnom	69
6.4. Ostala svojstva PSP modela	70
6.4.1. Neka svojstva pro irenja modela	70
6.4.2. Eliminacija proizvoljne predikcije	71
6.4.3. Petlje bez uslovnih grananja kao specijalni sluĀaj PSP modela	72
7. Model za realni sluĀaj i heuristike	74
7.1. Modifikacije PSP modela za realni sluĀaj	74
7.1.1. Operacije proizvoljnog trajanja	74
7.1.2. Ugne Āne IF strukture	75
7.1.3. OgraniĀenja u resursima	76
7.1.4. Eliminacija proizvoljne predikcije	76
7.1.5. Generisanje koda	76
7.2. Heuristike za rasporeĀivanje	77
7.2.1. Globalni postupak	77
7.2.2. Pro irenje predikatske matrice	78
7.2.3. Kriterijumi pomeranja operacija	80
7.2.4. PSP model kao vrsta Markovljevog lanca	81
7.2.5. RaĀunska svojstva modela	82
8. Eksperimentalna analiza	83
8.1. Uslovi eksperimenta	83
8.1.1. Analiza PSP modela	83
8.1.2. UporeĀenje sa tehnikom EPS	84
8.2. Rezultati eksperimenta	85
8.2.1. Analiza PSP modela	85
8.2.2. UporeĀenje sa tehnikom EPS	88
8.3. ZakljuĀak eksperimenta	89
8.3.1. Analiza PSP modela	89
8.3.2. UporeĀenje sa tehnikom EPS	89
9. Pravci budućeg rada	91
9.1. Potpuna teorijska analiza modela	91
9.2. Definisanje svih elemenata rasporeĀivanja	91
9.3. Definisanje svih detalja generisanja koda	92

9.4. Hardverska podpora	92
9.4.1. Arhitekture prilagojene PSP modelu	92
9.4.2. Veza sa prediktorima grananja	93
10. Zaključak	94
Reference	96
Zahvalnice	99

1. Uvod

Predmet ovog rada predstavlja analiza problema optimizacije koda programskih petlji sa uslovnim grananjima i predlog njegovog rešavanja. U radu su najpre izloženi razlozi razmatranja ovog problema i postojeći načini njegovog rešavanja, zatim su iznesena uočena svojstva ovog problema, u smislu izvora paralelizma koji se može detektovati u petljama sa uslovnim grananjima, i formiran je teorijski model softverske protočnosti ovih petlji. Predloženi teorijski model je analiziran, data je definicija pojma težnje ka optimalnom i dokazano je ovo svojstvo predloženog modela. Predložene su, takođe, i heuristike koje se mogu primeniti u optimizaciji koda u realnim slučajevima. Neke predložene heuristike su eksperimentalno ispitane i prikazani su dobijeni rezultati.

Ključne reči: Instrukcijski nivo paralelizma, programske petlje sa uslovnim grananjima, optimizacija koda, softverska protočnost.

1.1. Okvir rada

Moderni računari visokih performansi, zasnovani na procesorima tipa RISC sa protočnom obradom (engl. *pipeline*) [15, 22], superskalara [15] ili VLIW [15, 10], svoju računsku snagu zasnivaju na mogućnosti paralelnog izvršavanja više operacija. Za razliku od druge krajnosti koristi se paralelnog rada računarskih resursa, koji se eksplicitno ili implicitno specifikuje u izvornom kodu programa, ovi računari koriste paralelizam na nivou *instrukcija* (engl. *instruction level parallelism*, ILP)¹. Dobitak zbog korišćenja bilo koje vrste paralelizma je očigledan: ukoliko više resursa može izvršavati više zadataka istovremeno, ukupno vreme izvršavanja će biti kraće nego vreme izvršavanja istih zadataka sekvencijalno. Uslov je, naravno, da pojedini zadaci mogu da se izvršavaju uporedo, tj. da nisu međusobno zavisni.

Suštinska razlika između u tri navedene kategorije procesora je u načinu korišćenja ovog paralelizma. RISC procesori sa protočnom obradom izvršavanje svake operacije (instrukcije) dele na više faza. Svaka faza koristi različite hardverske resurse, a paralelizam se postiže preklapanjem različitih faza susednih operacija (instrukcija). Kod VLIW (engl. *Very Large Instruction Word*) arhitektura, svaka instrukcija u sebi sadrži nekoliko operacija. Svaka od operacija koristi različiti hardverski resurs. Operacije unutar jedne instrukcije se, dakle, izvršavaju paralelno. Na taj način, VLIW procesori koriste paralelizam koji je detektovan *statički*: prevodilac je taj koji međusobno nezavisne operacije grupiše u instrukcije tako da se mogu paralelno izvršavati. Superskalarni procesori, naprotiv, koriste *dinamički* detektovan paralelizam: procesor dohvata više susednih instrukcija (dve, četiri ili više) koje su na redu za izvršavanje, proverava da li su međusobno zavisne i, ako nisu, izvršava ih paralelno na različitim resursima. Naravno, i superskalarni i VLIW procesori mogu koristiti istovremeno i protočnu obradu, čime se kombinuju navedeni efekti. U svakom slučaju, korist prevodioca koji pronalazi međusobno nezavisne instrukcije i grupiše ih tako da budu susedne ima veliki značaj i za superskalarne procesore: dinamički paralelizam nikada ne može da "dosegne" onoliko daleko koliko to može prevodilac analizirajući izvorni program. Prevodilac, dakle, ima zadatak da pronađe međusobno nezavisne operacije i grupiše ih zajedno, kako bi procesor bio u

¹Za razliku od izvornog naziva paralelizma u kome se koristi reč *instrukcija*, mi ćemo u radu koristiti reč *operacija*, zato što kod VLIW arhitektura ova dva termina imaju različito značenje. Kod RISC procesora sa protočnom obradom i superskalarnih procesora, ova dva termina se mogu koristiti u istom značenju.

stanju da ih paralelno izvršava. Ovakav postupak naziva se *optimizacijom* koda za paralelne računare.

Međuzavisnost koja je do sada pominjana predstavlja zapravo pojavu da jedna operacija u izvornom programu koristi rezultat neke prethodne operacije. Ovakva zavisnost predstavlja tok podataka definisan samim algoritmom koji je realizovan programom. Ova zavisnost naziva se *prava zavisnost po podacima* [15]. Slučaj kada neka operacija upisuje u istu lokaciju koja je predstavljala izvori ni operand neke prethodne operacije naziva se *antizavisnost* i može se eliminisati preimenovanjem [15]. Takođe, ukoliko dve operacije upisuju svoje rezultate u istu lokaciju, zavisnost se naziva *izlaznom* i opet se može eliminisati preimenovanjem [15]. Svaki od navedenih oblika zavisnosti između dve operacije može da spreži zamenu mesta ove dve operacije u optimizovanom programu ili njihovo paralelno izvršavanje.

Iz prethodno rečenog sledi da performanse izvršavanja nekog programa na paralelnom računaru zavise primarno od tri faktora: 1) postojanja međusobno nezavisnih operacija koje se mogu izvršavati paralelno, 2) sposobnosti prevodioca da ove operacije detektuje i grupiše ih zajedno, zadržavajući pri tom semantičku ispravnost programa, i 3) raspoloživosti resursa procesora da se ovakve operacije izvršavaju paralelno. Prvi faktor je isključivo svojstvo samog algoritma koji program realizuje i o njemu se ovde nećemo diskutovati. Treći faktor postaje sve manje bitan, kako procesori postaju hardverski bogatiji, ali nikada ne može biti zavisna kao parametar ograničenja paralelizma. Drugi faktor je primaran predmet ovog rada.

Programi koji zahtevaju izvršavanje na računaru visokih performansi najvećim delom vremena provode izvršavajući programske petlje. Programske petlje pružaju kvalitativno novu mogućnost pronalaznja paralelizma u odnosu na kod van petlji, jer se može pretpostaviti da se iteracije petlje ponavljaju mnogo puta, a time se i povećava verovatnoća da se mogu pronaći operacije iz različitih iteracija koje su međusobno nezavisne. Programske petlje koje se dominantno pronalaze u numeričkim programima imaju visok stepen paralelizma i do sada su dobro izučene. Takođe, programske petlje u nenumeričkim (tzv. simboličkim) programima koje ne sadrže uslovna grananja (uslovne skokove tj. IF konstrukte) teorijski i praktično su dobro obrađene. Sa druge strane, programske petlje koje sadrže uslovna grananja do sada nisu detaljno teorijski obrađene, a praktična rešenja za optimizaciju ovih petlji su još uvek malobrojna i nedovoljno ispitana.

U ovom radu se razmatraju samo krajnje ugneđene (engl. *innermost*) petlje. Razlog za ovo ograničenje je dvostruk. Prvo, programi koji provode najviše vremena izvršavajući petlje, zapravo provode najviše vremena izvršavajući krajnje ugneđene petlje, jer se u svakoj iteraciji okružuju petlje izvršavajući iznova sve iteracije ugneđene petlje. Drugo, postoje tehnike [24] koje optimizuju okružuju petlje nakon optimizacije ugneđene petlje.

Zbog svega to je rečeno, predmet ovog rada predstavlja analiza i rešavanje problema optimizacije koda programskih petlji sa uslovnim grananjima na instrukcijskom nivou paralelizma.

1.2. Struktura rada

Rad je podeljen na deset glava. Glava 2 sadrži definiciju problema i diskusiju o njegovom značaju. Glava 3 daje kratak pregled postojećih rešenja ovog problema. U glavi 4 iznose se neka uočena svojstva petlji sa uslovnim grananjima, u pogledu mogućnosti pronalaznja paralelizma. U glavi 5 se iznosi ideja predložene rešenja i skicira celokupan tok daljeg razvoja teorijskog modela. Sam teorijski model je formalno opisan i analiziran u glavi 6. U ovoj glavi je data i definicija svojstva te nje ka optimalnom rešenju i dokazano da ovo svojstvo predloženi model poseduje. U glavi 7 se diskutuju prilagođanja predložene teorijskog modela za realne slučajeve, kada operacije traju proizvoljno dugo i kada postoje ograničenja u resursima. Takođe se predlažu i neke heuristike za raspoređivanje operacija. Glava 8 opisuje neke eksperimentalne provere predložene rešenja i daje rezultate ovih provera. Glava 9 daje smernice daljeg rada na ovom problemu. Najzad, glava 10 predstavlja zaključak.

2. Definicija problema

U ovoj glavi definisan je problem optimizacije koda programskih petlji sa uslovnim grananjima koji se razmatra u ovom radu, u kontekstu instrukcijskog nivoa paralelizma. Takođe je definisan i jedan od okvira za rešavanje ovog problema - softverska protočnost (engl. *software pipelining*). Na kraju se ukratko diskutuje značaj rešavanja ovog problema.

2.1. Optimizacija koda programskih petlji

Kao što je ranije naglašeno, programi koji zahtevaju efikasno izvršavanje najviše vremena provode izvršavanje petlje. Zbog toga je problem paralelizacije programskih petlji značajan. Programske petlje pružaju posebnu mogućnost za paralelizaciju, zato što postoji verovatnoća da se dve operacije koje se mogu izvršavati paralelno pronađu u različitim iteracijama iste petlje, umesto u jednoj iteraciji - telu petlje posmatranom kao bazični blok [15].

Posmatrajmo sledeći primer programske petlje [14]:

```
DO I=3,100
O1:   A[I]=C[I-1]+E[I-2]
O2:   B[I]=A[I]*C[I-3]
O3:   C[I]=B[I]*K1
O4:   D[I]=D[I-1]+K2
O5:   E[I]=C[I]+D[I]
O6:   F[I]=C[I-1]+E[I]
END
```

Primer 1: Izvorni kod jedne petlje bez uslovnih grananja

Uz pretpostavku da operacija sabiranja (+) traje jedan ciklus takta, a operacija množenja (*) dva, i da procesor poseduje dovoljno jednakih resursa koji mogu izvršavati sve operacije, optimizacija tela date petlje posmatranog kao bazični blok, bez selidbe operacija van granica bazičnog bloka [15], omogućava sledeću paralelizaciju:

Ciklus takta:	Operacija:
1	O1, O4
2	O2
3	
4	O3
5	
6	O5
7	O6

Dakle, umesto osam ciklusa takta, koliko traje inicijalno telo petlje, dobija se telo petlje trajanja sedam ciklusa². Međutim, ovakvo ograničenje optimizacije na telo petlje kao bazični blok, bez selidbe operacija van granica iteracije, smanjuje mogućnost paralelizacije. Kada se omogućava pomeranje operacija preko granice iteracije, uz iste pretpostavke i za isti primer, može se dobiti sledeći raspored:

²Trajanje rečijskih operacija petlje (inkrementiranje indeksne promenljive, ispitivanje uslova izlaska iz petlje i uslovni skok na novu iteraciju) se u ovom trenutku ne razmatra.

Ciklus takta:	Operacija:
1	O1, O4 , O6
2	O2
3	
4	O3
5	
6	O5

Rezultujuće transformisane (optimizovane) petlje iz Primera 1 izgleda ovako:

```

O4:      D[3]=D[2]+K2
O4:      D[4]=D[3]+K2
O4:      D[5]=D[4]+K2
O1:      A[3]=C[2]+E[1]
O2:      B[3]=A[3]*C[0]
O3:      C[3]=B[3]*K1
O5:      E[3]=C[3]+D[3]
DO I=3,97
O1:      A[I+1]=C[I]+E[I-1]
O4:      D[I+3]=D[I+2]+K2
O6:      F[I]=C[I-1]+E[I]
O2:      B[I+1]=A[I+1]*C[I-2]
O3:      C[I+1]=B[I+1]*K1
O5:      E[I+1]=C[I+1]+D[I+1]
END
O1:      A[99]=C[98]+E[97]
O2:      B[99]=A[99]*C[96]
O3:      C[99]=B[99]*K1
O5:      E[99]=C[99]+D[99]
O6:      F[98]=C[97]+E[98]
O1:      A[100]=C[99]+E[98]
O2:      B[100]=A[100]*C[97]
O3:      C[100]=B[100]*K1
O5:      E[100]=C[100]+D[100]
O6:      F[99]=C[98]+E[98]
O6:      F[100]=C[99]+E[100]

```

Dakle, telo petlje je sada trajanja est ciklusa takta, to znatno utiĹe na ukupno trajanje izvr avanja cele petlje.

Programske petlje sa uslovnim grananjima pru aju jo jedan nivo moguosti paralelizacije: kontrola toka izvr avanja razliĹitih iteracija petlje uzima razliĹite smerove, pa postoje razliĹite staze optimizacije. Međutim, ova slo enost kontrole toka unosi i znatno veće koœu optimizaciju koda.

Posmatrajmo sledećiprimer petlje sa uslovnim grananjima:

```

DO I=1,N
O1:      D=A+B
O2:      E=D*C
O3:      F=E*A
O4:      IF (D>0) THEN
O5:          G=F+5
O6:      ELSE
O7:          J=A-2
O8:      ENDIF
O9:      A=G*J
END

```

Primer 2: Izvorni kôd jedne petlje sa uslovnim grananjaima

Du ina izvr avanja inicijalne petlje je osam ciklusa takta za svaku granu izvr avanja ³. Uz iste pretpostavke o trajanju operacija i resursima procesora kao u prethodnom primeru, ukoliko se telo optimizuje bez selidbe operacija preko granice iteracije, tehnikom *List Scheduling* [15, 10] uz spekulativno izvr avanje i preimenovanje [15], mo e se dobiti telo petlje koje se za jednu granu izvr ava osam, a za drugu pet ciklusa takta (u istom redu su navedene operacije koje se mogu izvr avati paralelno):

```

DO I=1,N
01, 05':      D=A+B;  JP=A-2
              IF (D>0) THEN
02:          E=D*C
03:          F=E*A
04:          G=F+5
06:          A=G*J
              ELSE
02, 06:      E=D*C;  A=G*JP
03, 05":      F=E*A;  J=JP
              ENDIF
END

```

Međutim, kada se dozvoli selidba operacija preko granice iteracije, mo e se dobiti izvr avanje koje za jednu granu traje osam, a za drugu samo tri ciklusa takta. U glavi 4 eebiti pokazan naLin na koji se mo e dobiti ovakav optimizovan kôd ⁴:

```

Pretpetlja
DO I=2,N-1
  IF (D<=0) THEN
01,05':      D=A+B;  JP=A-2
              IF (D>0) THEN
02,03:      E=D*C;  F=E*A
03:          F=E*A
04:          G=F+5
06:          A=G*J
              ELSE
02,06,03,05":  E=D*C;  A=G*JP;  F=E*A;  J=JP
              ENDIF
  ELSE
01,05':      D=A+B;  JP=A-2
              IF (D>0) THEN
02:          E=D*C
03:          F=E*A
04:          G=F+5
06:          A=G*J
              ELSE
02,06,05":      E=D*C;  A=G*JP;  J=JP
              ENDIF
  ENDIF
END
Postpetlja

```

Zaključak je, dakle, da se veće stepen paralelizma u kodu petlje sa uslovnim grananjima mo e dobiti analizom međuzavisnosti operacija iz različitih grana izvr avanja različitih iteracija. Problem postojanja i definisanja postupka za pronala enje ovog paralelizma upravo je predmet ovog rada.

³Trajanje operacije uslovnog skoka (IF-THEN-ELSE) se za sada zanemaruje.

⁴Ovde je dat strukturirani kôd na izvornom nivou. Na asemblerskom nivou ili na izvornom nivou uz nestrukturirane GOTO naredbe mo e se postići ~~isto~~ ekvivalentan datom, ali bez spoljne uslovne strukture IF (D<=0).

2.2. Softverska protočnost

Uopšteno gledano, postoje dva glavna pravca za optimizaciju koda programskih petlji: razmotavanje (engl. *unrolling*) i softverska protočnost (engl. *software pipelining*) [15, 10, 1].

Razmotavanje petlje predstavlja sledeći postupak. Kada tela petlje se replicira više puta, tako da više susednih iteracija predstavlja sada novi izvorni kod za optimizaciju. Naravno, pri tome je potrebno izvršiti preimenovanja promenljivih i modifikaciju uslova za izlazak iz petlje. Ovako dobijen kod se zatim optimizuje nekom od tehnika globalne optimizacije koda.

Tehnika razmotavanja ima svoje prednosti i mane. Prednost je u relativno jednostavnom postupku, koji se svodi na mnogobrojne tehnike optimizacije koda van programskih petlji. Druga prednost je ta što se za izvesne petlje može pronaći stepen razmotavanja (broj repliciranja tela petlje) za koji se dobija bolji rezultat od tehnike softverske protočnosti. Prvi nedostatak je eksplozija koda: sve operacije tela petlje se u optimizovanom kodu pojavljuju onoliko puta, koliko je puta petlja razmotana. Drugo, pokazuje se da razmotavanje ima izvesnih nedostataka u odnosu na softversku protočnost koji se ogledaju u tome da se na početku i kraju tela optimizovane petlje stepen paralelizma smanjuje, što znači da postoji izvestan režijski trošak da se telo petlje "zahukta" pre nego što dostigne puni "zalet" na sredini, i "zaustavi" pre nego što polne sledeće iteracija. Ovaj nedostatak ne postoji kod softverske protočnosti [15].

Tehnika razmotavanja petlje se može primeniti, praktično pravolinijski, i na petlje sa uslovnim grananjima. Razlika je u tome što se za optimizaciju koda programskih petlji moraju koristiti tehnike optimizacije koda koje obuhvataju pomeranja operacija preko granice baziranih blokova [15, 10, 1]. Verovatno je da se mogu definisati i posebne heuristike koje za specifičan problem optimizacije petlji sa uslovnim grananjima mogu da daju bolje rezultate u odnosu na globalne tehnike. Ovim problemom se u radu neće baviti.

Druga tehnika, softverska protočnost [15], zasniva se na preklapanju različitih iteracija petlje u cilju postizanja paralelizma. Tehnika je analogna hardverskoj protočnosti, s tim da se u ovom slučaju preklapaju operacije iz različitih iteracija petlje. Rezultujuć kod proizlazi iz inicijalne petlje sastoji se iz tri dela: tzv. *pretpetlje* (engl. *preloop* ili *prologue*), jezgra (engl. *kernel*) i tzv. *postpetlje* (engl. *postloop* ili *epilogue*). Pretpetlja inicira prvih p iteracija. Posle prvih $p \cdot II$ ciklusa, gde je II tzv. *interval iniciranja* (engl. *initiation interval*), postiže se stacionarno stanje izvršavanja. U ovom stacionarnom stanju, svakih II ciklusa takta inicira se nova iteracija. Kada se završi izvršavanje jezgra, izvršava se postpetlja, koja okončava izvršavanje poslednjih p iteracija inicijalne petlje. Za petlje sa velikim brojem iteracija, najveće vreme se provodi u izvršavanju jezgra, pa je cilj da se ono optimizuje, odnosno da se postigne minimalan II [39].

Tehnika softverske protočnosti ima manu u odnosu na razmotavanje u tome što se za petlje koje nemaju celobrojan maksimalan količnik ukupne dužine i ukupne iteracione distance kritičnog zatvorenog puta u cikličkom grafu zavisnosti [15] ne može postići teorijski optimum za neograničene resurse procesora. Ipak, čini se da je ovaj nedostatak daleko manje bitan u odnosu na prednosti koje ova tehnika poseduje: eksplozija koda je ograničena, jer telo optimizovane petlje sadrži samo po jednu kopiju svake operacije iz inicijalne iteracije, a "višak" koda se nalazi samo u pretpetlji i postpetlji; ovaj kod pretpetlje i postpetlje, ukoliko je petlja ugnječena, može da se "meša" sa operacijama okružujućih petlji, čime se okružujuća petlja optimizuje ili, ako je petlja okružena sekvencijalnim kodom, kod pretpetlje i postpetlje ulazi u globalnu optimizaciju koda van petlji. Drugo, ne postoji režijski trošak na početku i kraju svake iteracije kao kod razmotavanja, već se u toku izvršavanja jezgra postiže stacionarno stanje.

Petlje sa uslovnim grananjima predstavljaju posebnu teškoću za primenu softverske protočnosti, jer kod sadrži više staza kontrole toka. Ipak, zbog nabrojanih prednosti ove tehnike, u ovom radu neće definisani problem razmatrati isključivo u kontekstu softverske protočnosti. Sigurno je da kombinacija dve navedene tehnike može da ujedini pozitivna svojstva obe, ali se u ovom radu neće razmatrati ova mogućnost.

Svi primeri u prethodnom odeljku optimizovani su tehnikom softverske protoĹnosti, pa se Ĺitalac upućuje na njih.

2.3. Značaj problema

Kao što je već reĹeno, programi koji zahtevaju izvršavanje na računarima visokih performansi veliki deo vremena provode u izvršavanju petlji. IstraĹivanje u [19] pokazuje ne samo ovu Ĺinjenicu, već i niz drugih svojstava paralelizma na nivou petlji. Osnovni zakljuĹak ovog istraĹivanja je da postoji velika razlika izmeĹ numeričkih programa koji su orijentisani na obradu nizova (vektora) i nenumeričkih (simboličkih) programa. Prvo, simbolički programi imaju manje petlje koje se izvršavaju manji broj puta. Drugo, zavisnosti po podacima izmeĹ operacija iz razliĹitih iteracija predstavljaju bitno ograniĹenje paralelizma u simboličkim programima. Već postojeće optimizacionih tehnika orijentisana je ka petljama koje manipulišu u nizovima. Ove tehnike nemaju dobre rezultate pri optimizaciji petlji u simboličkim programima, pa je potrebno razvijati nove, primerenije tehnike.

Definisani problem se ovde tretirati na naĹin koji ne zavisi od vrste programa. Preliminarni zakljuĹci ukazuju na to da se ovde predloĹena tehnika može podjednako uspešno da se sprovodi i na numeričkim programima, Ĺije petlje poseduju visok stepen paralelizma, i na simboličkim programima, koji poseduju izraĹene zavisnosti izmeĹ razliĹitih iteracija, pri Ĺemu u oba sluĹaja postoje uslovna grananja. U tome je znaĹaj ovoga rada.

3. Pregled postojećih rešenja

U ovoj glavi prikazani su najpre postojeći teorijski rezultati vezani za petlje bez uslovnih grananja, a zatim i za petlje sa uslovnim grananjima. Potom su ukratko opisane postojeće značajnije tehnike raspoređivanja koda za petlje sa uslovnim grananjima. Svi rezultati odnose se na softversku protočnost, osim ako je eksplicitno rečeno drugačije.

Efektivni postupci koji vrše raspoređivanje operacija (engl. *scheduling*) za petlje sa uslovnim grananjima mogu se razvrstati po više kriterijuma. Prvi kriterijum je način na koji se operacije raspoređuju i razrešavaju konflikti na resursima [39]. Prvi pristup se bazira na globalnoj optimizaciji koda, pri čemu se operacija raspoređuje što je ranije moguće (prema zavisnostima po podacima), a konflikti na resursima rešavaju kada se postigne stacionarno stanje rasporeda. U ovu grupu spadaju sledeće značajnije tehnike: *Perfect Pipelining* [3], *Enhanced Pipeline Scheduling* [24] i GURPR* [35]. Drugi pristup, *Modulo Scheduling* u raznim varijantama [29, 18, 39], najpre pokušava da odredi donju granicu za interval iniciranja II koristeći ograničenja postavljena zavisnostima po podacima i resursima, a zatim raspoređuje operacije razrešavajući konflikte na resursima u toku raspoređivanja svake operacije.

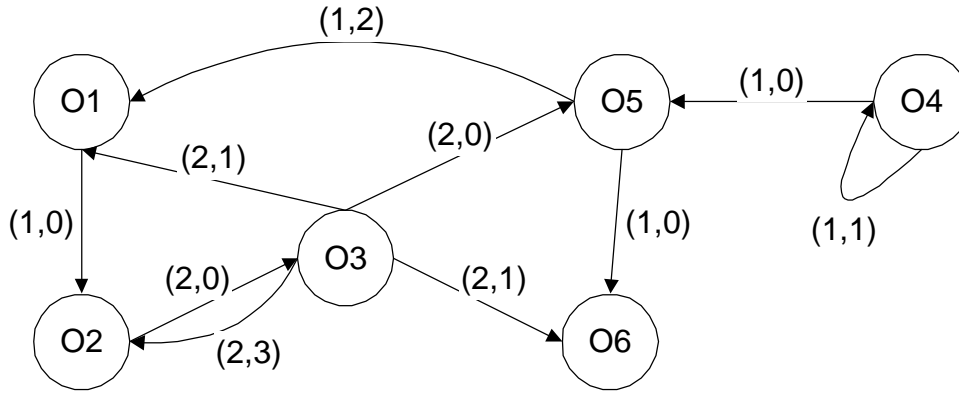
Drugi kriterijum za razvrstavanje tehnika optimizacije je prema tome da li se za petlje sa uslovnim grananjima dobija optimizovan kod sa konstantnim ili promenljivim II . U grupu tehnika koje daju konstantni II spadaju *Modulo scheduling* [29, 18, 39], GURPR* [35] i GPMB [36]. Za sada jedina značajna tehnika koja daje promenljiv II je *Enhanced Pipeline Scheduling* [24]. Ovde se tehnike mogu kategorizovati po ovom drugom kriterijumu.

3.1. Teorijski rezultati

Petlje bez uslovnih grananja su teorijski dobro izučene. Pokazano je da optimalno rešenje postoji i da se za neograničene resurse ono može dobiti (za softversku protočnost), a da se za slučaj ograničenja u resursima može dobiti optimalno rešenje za neke slučajeve petlji i ograničenja. Za petlje sa uslovnim grananjima, međim, dokazano je samo da se optimalno rešenje u opštem slučaju ne može dobiti, ali nijedan bliži rezultat ne postoji.

3.1.1. Rezultati za petlje bez uslovnih grananja

Zavisnosti po podacima između operacija tela petlje predstavljaju se usmerenim cikličkim grafom petlje [15]. Operacije predstavljaju čvorove grafa, a zavisnosti - usmerene grane grafa. Od čvoraa prema čvoru b polazi grana akko operacija b zavisi od operacije a . Svakoj grani (a,b) pridružen je uređeni par (l,d) , gde je l broj ciklusa takta koji moraju da proteknu od početka izvršavanja operacije a do početka izvršavanja operacije b , a d je tzv. *iteraciona distanca* - operacija b zavisi od operacije a koja se nalazi d iteracija pre operacije b . Veličinu l se može nazivati *dužinom* grane. U ovom radu se može podrazumevati da se za svaku granu može znati d u vreme prevođenja, kao i da je d konstantno za petlje bez uslovnih grananja. Za Primer 1 iz prethodne glave, usmereni ciklički graf dat je na Slici 1.



Slika 1: Usmereni ciklički graf petlje iz Primera 1

Grane koje imaju $d > 0$ predstavljaju zavisnosti između operacija različitih iteracija, tzv. *međuiteracione zavisnosti* (engl. *cross-iteration dependences* ili *loop-carried dependences*). Zatvoreni putevi u grafu⁵ predstavljaju *rekurencu* (engl. *recurrence*) - pojavu korišćenja rezultata ranijih iteracija kao operanada operacija iz narednih iteracija. Za slučaj neograničenih resursa, pokazano je da upravo ovi zatvoreni putevi predstavljaju ograničavajući faktor paralelizma. Naime, neka je:

$$CII = \max_{c \in C} \left[\frac{\sum_{e \in E_c} l_e}{\sum_{e \in E_c} d_e} \right],$$

gde je C skup svih usmerenih zatvorenih puteva u cikličkom grafu petlje, E_c skup svih grana koje čine zatvoreni put c , a (l_e, d_e) karakteristika grane e (dužina i iteraciona distanca grane). Pokazano je (dokaz se može naći u [15]) da je najmanji broj ciklusa za koji se jedna iteracija jezgra optimizovane petlje može izvršiti, u slučaju neograničenih resursa, upravo CII . To je zapravo donja granica za interval iniciranja II u opštem slučaju.

Takođe, postoje efektivni polinomijalni postupci kojima se postiže navedeno optimalno rešenje za slučaj neograničenih resursa. To su tzv. *raspoređivanje po modulu* (engl. *Modulo Scheduling*) [18, 15] i tzv. *perfektna protočnost* (engl. *Perfect Pipelining*) [3]. Prva tehnika biće ukratko opisana u 3.2.1.

Kada postoji ograničenje u resursima, može se desiti da se optimum od CII ciklusa po iteraciji ne može postići. Naime, neka procesor poseduje skup resursa R , i neka postoji n_r resursa kategorije r . Neka se u telu petlje resurs r koristi c_r ciklusa takta. Tada je interval iniciranja najmanje:

$$RII = \max_{r \in R} \left[\frac{c_r}{n_r} \right]$$

Dakle, interval iniciranja II je najpre odozdo ograničen veličinom CII koju određuju zavisnosti po podacima, a zatim i veličinom RII koju diktiraju raspoloživost resursa procesora i način korišćenja resursa od strane operacija. II je, dakle, najmanje jednak većoj od ove dve vrednosti:

$$II \geq \max(CII, RII)$$

⁵Podrazumevaju se da postoje samo prave zavisnosti po podacima.

Za sluĹaj petlji bez meĹiteracionih zavisnosti dokazano je da rasporeĹvanje po modulu daje optimalan raspored operacija za odreĹena ograniĹenja u resursima [29]. TakoĹ, dokazano je da optimalno rasporeĹvanje uz ograniĹenje u resursima predstavlja NP-kompletna problem [17].

3.1.2. Rezultati za petlje sa uslovnim grananjima

U [33] je pokazano da se u općem sluĹaju za petlje sa uslovnim grananjima ne moĹe postići rešenje koji se optimalno izvršava za svaki sluĹaj ishoda uslovnih grananja, Ĺak i ne uzimajući u obzir samo tehniku softverske protoĹnosti. Ovde æbiti ukratko izneseni osnovni delovi ovoga dokaza.

Dokaz pretpostavlja sinhronu paralelnu mašinu sa konaĹnom, ali neograniĹenom koliĹinom resursa. To znaĹi da ovakva mašina moĹe da izvrši svaki program koji koristi broj resursa ograniĹen nekim proizvoljno velikim celim brojem. Dalje, bez gubitka općosti, pretpostavlja se da izvršavanje svake operacije traje jedan ciklus takta. Najzad, pretpostavlja se proizvoljno spekulativno izvršavanje operacija [15]. Spekulativno izvršavanje predstavlja sledeće: ako je neka operacija *op* kontrolno zavisna [15] od nekog uslova *t* u sekvencijalnom programu *P*, onda se kaĹe da se *op* izvršava spekulativno u transformisanom programu *P'* akko je *op* rasporeĹna pre ili uporedo sa operacijom koja razrešava uslov *t*. Dokaz ne razmatra rešiske operacije petlje. U [33] nije eksplicitno reĹeno, ali se pretpostavlja da broj iteracija petlje moĹe da bude promenljiv i konaĹan, ali neograniĹen; drugim reĹima, broj iteracija petlje moĹe da bude vešod proizvoljnog celog broja; u svakom sluĹaju, ovaj broj nije poznat u vreme prevoĹnja.

U ovom razmatranju se najpre definiše pojam vremenske optimalnosti koji se zapravo razmatra. Postavlja se sledeća definicija.

Definicija 1: (*Vremenska optimalnost*) Program *P* je vremenski optimalan akko (po def.) za svaku instrukciju *I* iz *P*, koja se izvršava u ciklusu *c*, postoji bar jedna operacija *op* u *I* i bar jedan niz zavisnosti po podacima duĹine *c* koji se završava u *op*. □

Ako se svaka staza izvršavanja u *P* završava, ova definicija je ekvivalentna iskazu da se svaka staza izvršavanja u *P* završava u najkraćem mogućem vremenu. Drugim reĹima, dužina izvršavanja *P* jednaka je dužini kritiĹnog puta u grafu zavisnosti po podacima za svaku stazu izvršavanja *P*.

Postoje petlje koje nemaju uslovna grananja, a za koje ne postoji optimalno rešenje za konaĹnu koliĹinu resursa. To su petlje kod kojih ne postoje meĹiteracione zavisnosti, pa se proizvoljan broj iteracija moĹe izvršavati uporedo (tzv. DOALL petlje, [15]). MeĹtim, Ĺak i ako postoje meĹiteracione zavisnosti sa distancom 1, i ako petlja sadrži uslovno grananje, postoje petlje za koje ne postoji vremenski optimalan program. Dokaz se upravo zasniva na navoĹnju takvih petlji i neposredno je preuzet iz rada [33].

Teorema 1: (*O nepostojanju vremenski optimalnog rešenja*) Neka je *P* petlja prikazana na Slici 2. Tada ne postoji semantiĹki ekvivalentan, vremenski optimalan program *P'* koji moĹe da izvrši petlju *P*.

Dokaz: OĹigledno ne postoji semantiĹki ekvivalentan, vremenski optimalan necikliĹan program za petlju na Slici 2, ako je broj iteracija promenljiv i nije ograniĹen. U nastavku, *op(i)* oznaĹava izvršavanje operacije *op* iz iteracije *i*.

Bez gubitka općosti, pretpostavimo da *P* iterira $2k$ iteracija, za neki ceo broj *k*, i da je svaka staza izvršavanja moguæ. Pretpostavimo da *P* izvršava granu *False* uslova *t1* prvih *k* iteracija, a granu *True* preostalih *k* iteracija. U tom sluĹaju, bilo koje vremenski optimalno izvršavanje *P* zahteva taĹno $2k+1+k$ ciklusa takta zbog niza zavisnosti po podacima prikazanim na Slici 3.

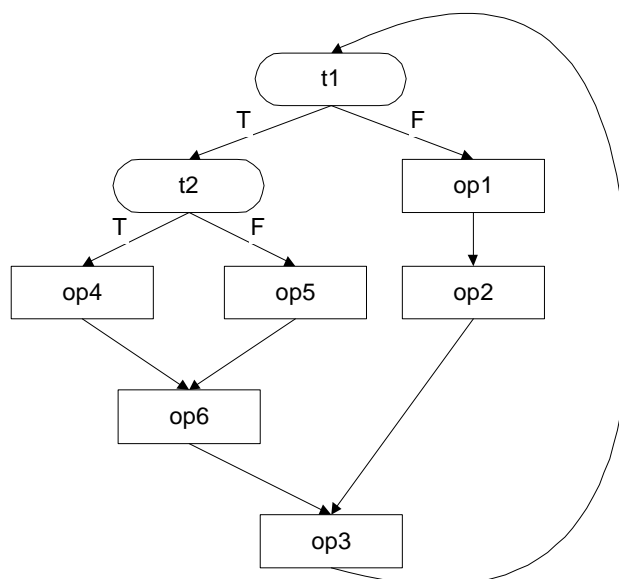
Posle ciklusa $2k+1$, bilo koja kopija operacije $op(3(k+j))$, $1 \leq j \leq k$, mora da se izvrši u ciklusu $2k+1+j$ da bi izvršavanje bilo optimalno. Prema zavisnostima na Slici 3b, sledi da kopija operacije $op(6(k+i))$, $1 \leq i \leq k$, mora da se izvrši najkasnije u ciklusu $2k+1+k-2(k-i)-1=k+2i$. Zbog

zavisnosti po podacima od $op3(k+i-1)$ do $t2(k+i)$, uslov $t2(k+i)$ ne može biti izvršen (pa zbog toga ni razrešen) pre ciklusa $2k+1+i$. Time se kreira rascep od najmanje $k+1-i$ ciklusa između trenutka kada $op6(k+i)$ treba da bude izvršen (da bi se obezbedilo vremenski optimalno izvršavanje) i trenutka kada se stvarni argumenti za izvršavanje $op6(k+i)$ zaista odrede⁶. Zato se mora čuvati trag od najmanje $2^{\lfloor (k+1-i)/2 \rfloor}$ kopija operacije $op6(k+i)$ iz isto toliko staza izvršavanja⁷. Kako je na raspolaganju samo $O(k)$ instrukcija⁸, broj operacija koje su raspoređene u jednu instrukciju, ili broj potrebnih procesora, konstantno raste sa porastom k . Kako je u bilo kom programu broj resursa po instrukciji ograničen, ne može postojati semantički ekvivalentan, vremenski optimalan program za datu petlju. \square

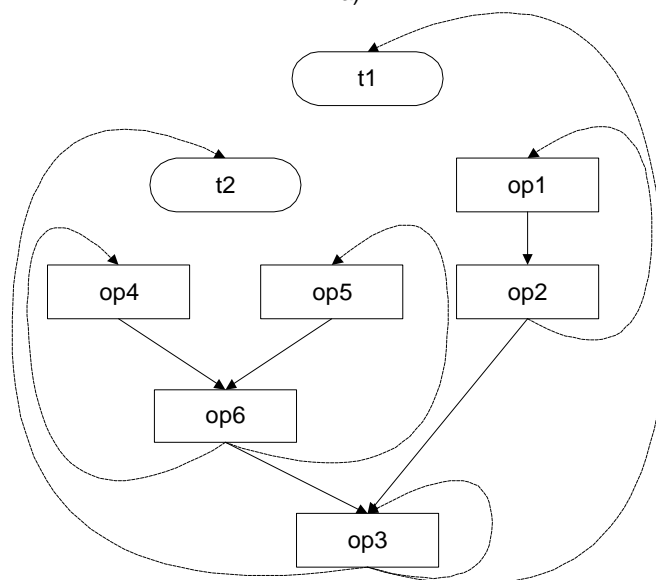
⁶Operacija $op6(k+i)$ se, tako, izvršava spekulativno sa promenljivim nivoom spekulativnosti koji zavisi od iteracije iz koje potiče operacija (prim. autora).

⁷Deljenje sa 2 u eksponentu potiče od toga što se operacija $op6$ pojavljuje u svakom drugom ciklusu. Broj staza izvršavanja zbog toga zavisi od eksponenta koji je dvostruko manji od broja ciklusa procepa koji se pominje (prim. autora).

⁸Kako je jedino rešenje ciklično, broj instrukcija u telu petlje je konačan, pa je ukupan broj instrukcija u izvršavanju cele petlje $2k$ puta proporcionalan broju k , dakle iznosi $O(k)$ (prim. autora).

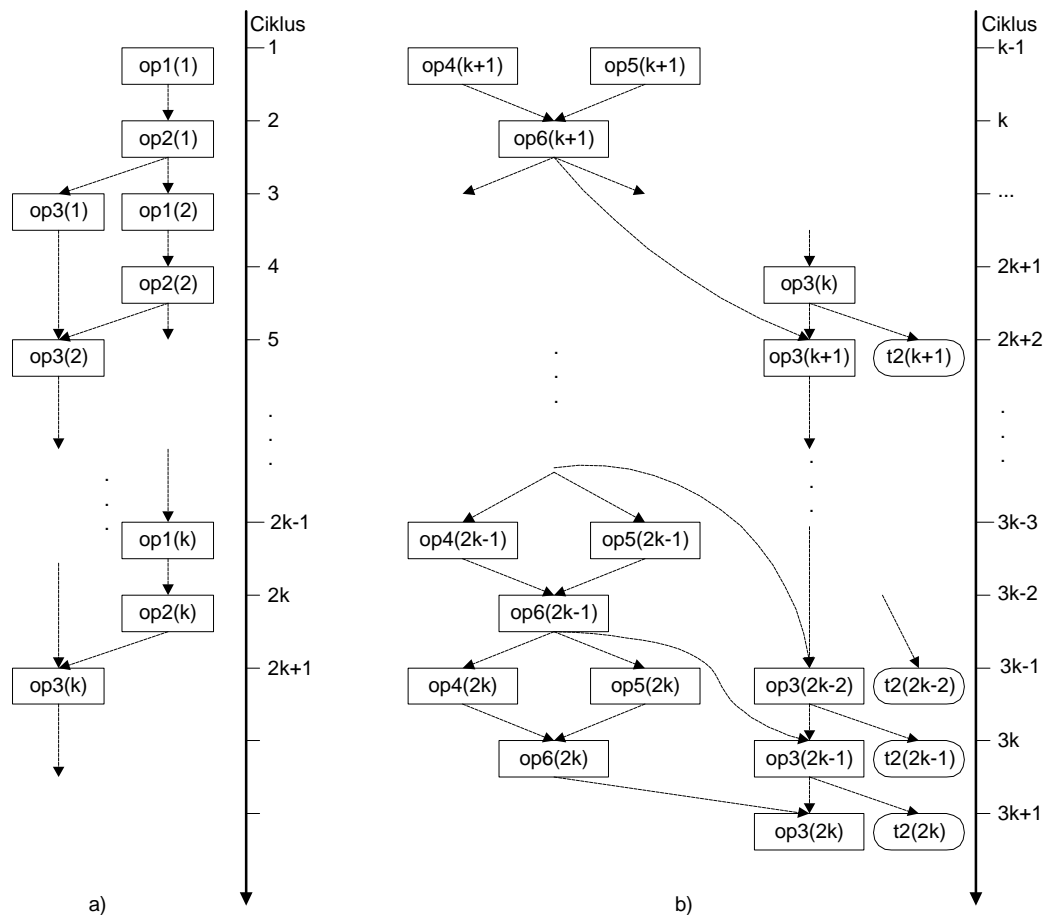


a)



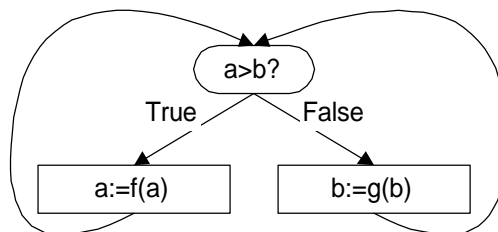
b)

Slika 2: (a) Graf kontrole toka jedne petlje. (b) Usmereni ciklički graf pravih zavisnosti po podacima za datu petlju. Pune grane označavaju zavisnosti unutar iste iteracije, a isprekidane - zavisnosti između operacija susednih iteracija.

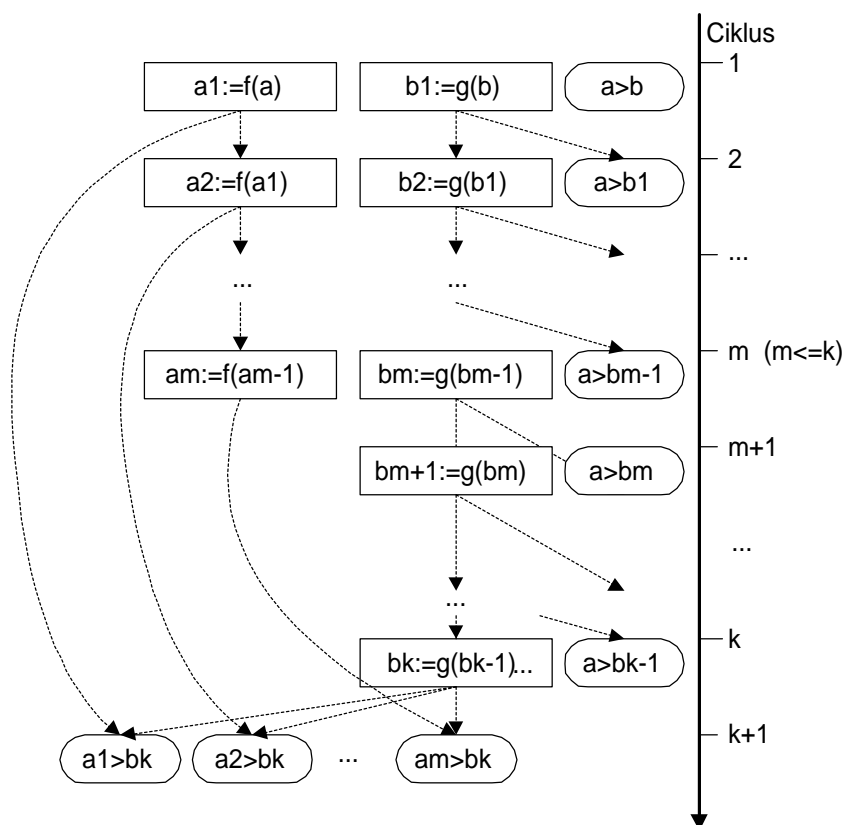


Slika 3: Nizovi zavisnosti po podacima za jedno izvršavanje petlje sa Slike 2. (a) Za prvih k operacija $op3$. (b) Za poslednjih k operacija $op3$.

Postoje λ ak i petlje koje sadre samo jedan uslov, a za koje ne postoji semantički ekvivalentan, vremenski optimalan program. Jednostavna takva petlja prikazana je na Slici 4. U slučaju da u prvih k iteracija ishod uslova bude *False*, a u preostalim m iteracija ($m \leq k$) *True*, vremenski optimalno izvršavanje traje $k+1$ ciklus (Slika 5). Da bi se ovo postiglo, m kopija operacije uslova mora da se izvrši uporedo u ciklusu $k+1$. Treba primetiti da je takođe neophodno preimenovanje promenljivih a i b . To znači da vremenski optimalno izvršavanje zahteva m resursa u ciklusu $k+1$. Kako k , pa time i m nisu ograničeni, ne postoji semantički ekvivalentan, vremenski optimalan program za datu petlju.



Slika 4: Jednostavna petlja sa samo jednim uslovnim grananjem



Slika 5: Graf zavisnosti po podacima za jedno izvršavanje petlje na Slici 4, sa preimenovanjem

Osvrnimo se ukratko na suštinu navedenog dokaza. Dokaz se jednostavno zasniva na pronalazenju primera petlje koja se ne može transformisati u vremenski optimalan kod koji koristi ograničen broj resursa, kada se pretpostavi da je broj iteracija petlje konačan, ali neograničen. Treba primetiti da za ostale slučajeve tvrdnje o nepostojanju optimalnog programa ne važi. Naime, kada bi se dozvolilo da program koristi neograničen broj resursa, optimalno rešenje bi se moglo postići. Isto tako, ako je broj iteracija ograničen (i poznata ta granica u vreme prevođenja), može se generisati optimalan program koji koristi ograničenu količinu resursa.

Treba takođe naglasiti da *optimalno izvršavanje svakog programa postoji*, to je upravo ono izvršavanje dato u Definiciji 1: definisano je kritičnim putem grafa zavisnosti datog izvršavanja. Teorema 1 samo tvrdi da se *ne može dobiti optimalan program* za uslove koji su upravo istaknuti i za sve staze izvršavanja.

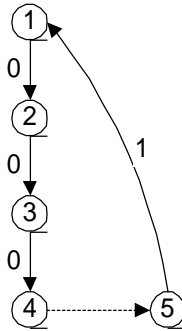
Dalje, uslov da optimalan program koristi ograničen broj resursa može se ekvivalentno postaviti i kao uslov ograničenja broja instrukcija (to je u dokazu i navedeno, broj instrukcija je $O(k)$), ili opetije, kao uslov ograničenja veličine rezultujućeg koda. Najzad, data analiza se ne ograničava na tehniku softverske protočnosti, to se u ovom radu čini.

Za razliku od navedenog, u [38] se ne razmatraju uslovi vremenski apsolutno optimalnog izvršavanja petlji sa uslovnim grananjima, već optimalnog u kontekstu softverske protočnosti. To znači da se svaka iteracija optimizovane petlje izvršava u najmanje jednom ciklusu takta⁹. Ovde će ukratko biti opisani osnovni rezultati ovog istraživanja.

Da bi se postiglo optimalno izvršavanje, potrebno je da se obezbedi spekulativno izvršavanje određenih operacija u općem slučaju. To znači da se u toku izvršavanja mora pratiti više staza izvršavanja pre nego što se sazna rezultat neke uslovne operacije. Na primer, pre nego

⁹Drugačije rečeno, to znači da se u svakom ciklusu inicira najviše jedna iteracija. Ovo je suštinsko ograničenje tehnike softverske protočnosti, to je razlikuje od DOALL izvršavanja. U ovom radu postoji uslov biti postavljen na još jedan ekvivalentan način.

to se sazna rezultat nekog uslova u petlji, potrebno je pokrenuti po dve niti izvravanja koje odgovaraju granama uslovnog skoka. Zbog udvostruĉavanja ovih staza za svaki uslovni skok u svakoj iteraciji, broj ovih staza je eksponencijalan u opremu sluĉaju. Pre nego to je u [38] pokazano drugaĉije, verovalo se da je pomenuti eksponent srazmeran broju iteracija, pa je time i broj resursa potrebnih za optimalno izvravanje petlje teorijski neograniĉen. U [38] je pokazano da je gornja granica broja potrebnih resursa znatno manja i ograniĉena.



Slika 6: Graf zavisnosti po podacima jedne petlje sa jednim uslovnim grananjem koga predstavlja operacija 4. Kontrolna zavisnost je izmeĉ operacija 4 i 5, a pored grana su date samo iteracione distance. Sve operacije traju jedan ciklus.

Dokaz se ograniĉava na petlje sa jednim uslovnim grananjem koje zavisi od neke operacije iz iteracije petlje, pri emu su zavisnosti po podacima najveće iteracione distance 1 i operacije trajanja 1 ciklus. U dokazu se posmatraju dva krajnja sluĉaja. Prvi je sluĉaj kada zavisnosti po podacima omoguavaju maksimalnu dubinu spekulativnog izvravanja. Posmatrajmo petlju ĉiji je graf zavisnosti po podacima prikazan na Slici 6. Operacija 4 predstavlja uslov od koga zavisi izvravanje operacije 5. Optimalno izvravanje u kontekstu softverske protoĉnosti je sledeće u ciklusu 1 inicira se operacija 1 iteracije 1 i operacija 5 iste iteracije se spekulativno izvrava. U ciklusu 2 iniciraju se dve verzije iteracije 2: jedna odgovara jednoj grani uslova, kada operacija 1 koristi rezultat operacije 5, a druga odgovara drugoj grani, kada operacija 1 ne koristi ovaj rezultat. Prema tome, u ciklusu 2 se generi u dve verzije iteracije 2 za jednu verziju iteracije 1. U narednom ciklusu 3 iniciraju se po dve verzije iteracije 3 za svaku verziju iteracije 2. Konaĉno, u ciklusu 4, razreava se uslov, koji dovodi do toga da se odbace sve one verzije iteracija 2 i 3 koje su odgovarale grani uslova koja se nije ostvarila. Osim toga, iniciraju se ponovo po dve verzije iteracije 4 za sve preostale iteracije.

Na ovaj naĉin se posti e stacionarno stanje: u svakom sledećem ciklusu se broj verzija iteracija dvostruko smanji zbog razrejenja jednog uslova, a istovremeno i dvostruko poveća zbog spekulativnog iniciranja novih iteracija. Formira se praktiĉno jedno binarno stablo dubine $L-1$, gde je L duina kritiĉnog puta do operacije uslova (u primeru je $L=4$). Broj verzija iteracija je tako:

$$N = 2^L - 1$$

Kako je broj operacija, a time i koliĉina resursa potrebnih za njihovo izvravanje srazmerna ovom broju, potrebe jesu eksponencijalne, ali je eksponent nezavisan od broja iteracija, već zavisi od duine kritiĉnog puta do uslova.

Ovo je jedan krajnji sluĉaj koji daje maksimalnu gornju granicu za broj resursa. Drugi graniĉni sluĉaj je kada neka od operacija iz neke grane uslova zavisi po podacima od neke operacije pre uslova. Bez dokaza ovde navodimo rezultat iz [38] koji govori da je gornja granica za broj verzija iniciranih iteracija, pa time i koliĉina resursa, polinomijalno zavisna od L , gde je sada L duina zatvorenog puta u cikliĉkom grafu zavisnosti petlje:

$$N = \frac{L(L+1)}{2}$$

U [38] se takođ diskutuju i sluĹajevi izmeđ ova dva graniĹna i daje precizan rezultat za ograniĹenja resursa kod njih. Takođ se diskutuju i ostali sluĹajevi vi e uslovnih grananja u iteraciji i operacija Ĺije je trajanje vi e od jednog ciklusa. KonaĹan zakljuĹak je da koliĹina resursa mo e da bude eksponencijalno zavisna, ali ne od broja iteracija, veđ du ine kritiĹnog puta. U mnogim sluĹajevima, ova gornja granica je polinomijalna ili Ĺak linearna. U ovom radu e se navedeni rezultati iskoristiti kao podr ka efektivnom postupku koji dovodi do navedenog optimalnog izvr avanja, u smislu ograniĹenog kori eanja resursa.

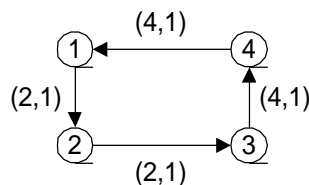
Treba na kraju naglasiti da Ĺak ni pojam *teŹnje rešenja ka optimalnom* (engl. *near-optimal*) nije u otvorenoj literaturi do sada definisan. Koliko je nama poznato, nikakvi drugi teorijski rezultati ne postoje vezani za optimizaciju petlji sa uslovnim grananjima u kontekstu softverske protoĹnosti.

3.2. Raspoređivanja koja daju konstantan interval iniciranja

Od tehnika koje daju konstantan II znaĹajnije su *Modulo Scheduling* [29, 18, 39], GURPR* [35] i GPMB [36]. Ovde eukratko biti prikazan samo *Modulo Scheduling* za petlje bez i sa uslovnim grananjima.

3.2.1. Raspoređivanje po modulu za petlje bez uslovnih grananja

Raspoređivanje po modulu (engl. *Modulo Scheduling*) [29, 18, 39] je tehnika koja raspoređuje operacije petlje po principu softverske protoĹnosti u cikluse takta u kojima eone biti inicirane, razre avajuđ konflikte na resursima u toku raspoređivanja svake pojedinaĹne operacije. Dakle, ovo je tehnika koja uzima u obzir ograniĹenja na resursima (engl. *resource-constrained technique*). Ovde e najpre biti opisan postupak raspoređivanja po modulu uop te, za petlje bez uslovnih grananja, a u narednom odeljku bi eobja njene neke tehnike koje ovo raspoređivanje prilagođavaju petljama sa uslovnim grananjima.



Slika 7: Ciklički graf zavisnosti jedne petlje

Postupak e mo pratiti na primeru petlje Ĺiji je ciklički graf dat na Slici 7. Postupak se sastoji u sledećem. Najpre se određuje teorijska donja granica za interval iniciranja II prema postupku opisanom u 3.1.1.:

$$II \geq \max(CII, RII)$$

Za dati primer, CII iznosi $12/4=3$, dok se RII mora odrediti prema raspolo ivim resursima. Pretpostavimo da procesor poseduje dve magistrale za operande, Src1 i Src2, jednu magistralu za rezultat R, i dve potpuno jednake ALU A1 i A2, pri Ĺemu je registarski fajl povezan preko magistrala sa obe ALU. Pretpostavimo da je registarski fajl dovoljno veliki da ne mo e doći do nedostatka registara za realizaciju date petlje.

Za svaku operaciju formira se njena osnovna tabela zauzeaa resursa. Ova tabela ima d vrsta, gde je d du ina trajanja operacije u ciklusima, i r kolona, gde je r ukupan broj svih resursa procesora. U polju (i,j) ove tabele stoji 1 ako ova operacija koristi resurs j u ciklusu i svog izvr avanja, a inaĹe stoji 0. Neka tabele zauzeaa resursa za operacije 1 i 2 izgledaju kao na Slici 8a, a za operacije 3 i 4 kao na Slici 8b.

Src1	Src2	R	ALU
X	X		
		X	X

(a)

Src1	Src2	R	ALU
X	X		
			X
			X
		X	

(b)

Slika 8: Tabele zauzeća resursa za operacije sa Slike 7. (a) za operacije 1 i 2; (b) za operacije 3 i 4. Oznaka X (ili 1) predstavlja zauzeće resursa u datom ciklusu, a prazno polje (ili 0) označava da se resurs u datom ciklusu ne koristi.

Prema datim tabelama RII iznosi:

$$RII = \max(4, 4, 4, 3) = 4,$$

jer je zauzeće resursa Src1, Src2 i R ukupno (za sve četiri operacije) po 4 ciklusa, a ALU ukupno $(1+1+2+2)/2=3$, jer postoje dve jednake ALU. Dakle, minimalni II iznosi 4.

Zatim se pokušava raspoređivanje operacija tako da se zadovolji ovaj minimalni II . Ako se ne pronađe raspored za ovaj minimalni II , pokušava se raspoređivanje ispočetka za prvi veći II itd. Dakle, u svakom koraku algoritma, jedna vrednost je tekuća II , i ona se povećava sve dok se ne pronađe raspored.

Za tekuću vrednost II raspoređuju se operacije jedna po jedna u cikluse označene brojevima počinju od 0 zaključno sa $II-1$. Za svaku operaciju se određuje ciklus u kome se ona najranije¹⁰ može izvršiti prema zavisnostima po podacima iz cikličkog grafa petlje, za operacije od kojih data operacija zavisi, a već su raspoređene. Pri tome se za dužine grana zavisnosti uzimaju ostaci originalnih vrednosti d iz grafa zavisnosti pri deljenju sa II (trajanje operacije se, tako, uzima "po modulu II ").

Kada se na ovaj način odredi ciklus u intervalu $[0, II-1]$ za neku operaciju, proverava se da li postoji konflikt na resursima te operacije i prethodno već raspoređenih operacija. Ova provera se vrši upoređivanjem tzv. *rezervacionih tabela po modulu* (engl. *modulo reservation tables*, MRT). Rezervaciona tabela po modulu za operaciju ima II vrsta i r kolona i dobija se iz osnovne tabele zauzeća resursa date operacije tako što se ova osnovna tabela "zamota" po modulu II : vrsta i tabele po modulu dobija se kao unija (logičko ILI) svih vrsta koje daju isti ostatak pri deljenju sa II . Pri tom, ako se dogodi da se u istoj koloni, a u redovima $i1$ i $i2$ osnovne tabele zauzeće nalazi 1, gde je $i1 \equiv i2 \pmod{II}$, onda je raspored za dati II nemoguć pa se II povećava za 1 i raspoređivanje ponavlja ispočetka. Za dati primer i tekuću II , tabele po modulu su iste kao i osnovne tabele operacija, jer je II jednako dužini najduže operacije.

U toku raspoređivanja formira se globalna rezervaciona tabela po modulu, tako što se pri raspoređivanju nove operacije njena rezervaciona tabela po modulu poredi sa trenutnim stanjem globalne rezervacione tabele po modulu. Tabela date operacije se "preklapa" sa globalnom tabelom počinju od onog reda globalne tabele koji predstavlja ciklus u koji se data operacija raspoređuje. Ako je ovaj ciklus veći od 0, "ostatak" rezervacione tabele operacije se "zamotava" na početak globalne tabele (opet po modulu). Ako sada postoji polje (i, j) sa vrednošću 1 u obe tabele, data operacija se ne može rasporediti u dati ciklus jer postoji konflikt na resursu, pa se pokušava raspoređivanje te operacije u naredni ciklus. Na taj način se data operacija sve više kasni, sve dok zavisnosti po podacima prema već raspoređenim operacijama to dozvoljavaju. Ako je raspoređivanje u datom ciklusu moguće, nova globalna tabela dobija se kao unija (logičko ILI) stare globalne tabele i rezervacione tabele date operacije.

¹⁰Postoji i modifikacija u kojoj se pronalazi najkasniji ciklus za operaciju.

Za dati primer, uzmimo da se najpre raspoređuje operacija 1. Ona se sme ta u ciklus 0 i u globalnu rezervacionu tabelu (koja je inicijalno prazna) upisuje zauzeće ove operacije. Zatim se raspoređuje operacija 2 u ciklus $(0+2)\bmod 4=2$ (jer je operacija 1 sme tena u ciklus 0 i traje 2 ciklusa), jer ne postoji konflikt na resursima, i u globalnu rezervacionu tabelu upisuje naLin njenog zauzeće. Operacija 3 se ne mo e rasporediti u ciklus $(2+2)\bmod 4=0$, jer dolazi do konflikta na resursima Src1 i Src2. Zato se ova operacija sme ta u ciklus 1. Najzad, operacija 4 se sme ta u ciklus 3, jer bi njeno sme tanje u cikluse 1 i 2 dovelo do konflikta. KonaLn raspored i izgled popunjene rezervacione tabele dat je na Slici 9. Treba primetiti da su zavisnosti po podacima zadovoljene; da to nije sluLaj, raspored za dati II ne bi bio moguć, pa bi se tra io raspored za veće II .

Ciklus	Operacija
0	1[0]
1	3[1]
2	2[1]
3	4[1]

(a)

Src1	Src2	R	A1	A2
1	1	3		4
3	3	1	1	4
2	2	4		3
4	4	2	2	3

(b)

Slika 9: KonaLn raspored i izgled globalne rezervacione tabele po modulu (MRT) za primer sa Slike 7. (a) Raspored operacija. Pored oznake operacije, u uglastim zagradama je navedeno iz koje originalne iteracije potile ([0]-ista iteracija, [1]-prva naredna iteracija). (b) KonaLn rezervaciona tabela. Oznaka u odgovarajućem polju predstavlja operaciju koja zauzima resurs u datom ciklusu.

Ako se nikako ne mo e pronaći raspored svih operacija za dati II , II se poveća za 1 i poku ava raspoređivanje ispoLetka po istom postupku. NajLe će se II ograničava nekom heuristikom i odozgo, tako da ako se ne postigne raspored ni za ovu gornju granicu II , od optimizacije se odustaje, jer se petlja smatra nepogodnom za softversku protoLnost.

Razne konkretne realizacije osnovnog principa raspoređivanja po modulu razlikuju se u heuristikama koje se primenjuju tokom samog raspoređivanja. Razlikuje se najpre redosled izbora operacija za raspoređivanje. Osnovna metoda je da se operacijama pridru uje prioritet i da se najpre raspoređuju operacije sa najvi im prioritetom. Pri tome se operacijama na najkritičnijim¹¹ zatvorenim putevima u cikličkom grafu zavisnosti po podacima pridru uje najvi i prioritet. Sledeće po prioritetu su operacije koje se nalaze na ostalim zatvorenim putevima. Iza njih po prioritetu su operacije koje su "odozgo" ili "odozdo" ograničene operacijama na zatvorenim putevima. Najni eg prioriteta su operacije koje su potpuno slobodne u grafu [39].

Dalje razlike su u postupku razre avanja konflikta na resursu. Jednostavniji postupak je da se pronađ naredni ciklus u kome se mo e izvr iti operacija, kao to je to pokazano ovde; ako se ovakav ciklus ne nađ, ide se na veće II . Druga varijanta je da se poku ava sa izbacivanjem već raspoređenih operacija iz rasporeda, Lime se praktično vr i *backtracking*.

Najzad, treba napomenuti da ova tehnika vr i samo raspoređivanje operacija (engl. *scheduling*), a da je neophodno izvr iti jo neke transformacije da bi se do lo do konaLnog optimizovanog koda. Prvo, po to je moguće da se operacije "prote u" na vi e iteracija, to je moguće i da se ivotni vek neke promenljive koja je rezultat neke operacije prote e na vi e iteracija [39, 32, 4]. Ako se ta promenljiva smesti u jedan registar, do lo bi do prepisivanja vrednosti od strane iste operacije iz naredne iteracije. Problem se mo e re iti na dva načina. Prvi je da u hardveru postoji kru ni registarski fajl za svaki nominalni registar, tako da se sukcesivni upisi susednih iteracija u isti nominalni registar zapravo vr e u različite registre [30]. Drugo re enje se ne oslanja na postojanje hardverske podr ke, već dobijena petlja razmotava onoliko puta koliko je potrebno da se obuhvati najdu i ivotni vek promenljive [39]. Tada, u razmotanoj petlji, različite

¹¹To su oni zatvoreni putevi koji imaju najvećoličnik zbira du ina grana i zbira iteracionih distanci grana.

kopije iste operacije koriste različite registre. Pored alokacije registara, postoji i problem generisanja pretpetlje i postpetlje kojim se mi ovde nećemo baviti.

Efikasno tretiranje ograničenja na resursima, jednostavnost implementacije, polinomijalan algoritam i dobri praktični rezultati čine ovu tehniku pogodnom za petlje bez uslovnih grananja.

3.2.2. Raspoređivanje po modulu za petlje sa uslovnim grananjima

Efikasnost raspoređivanja po modulu i njegova principijelna jednostavnost potakla je mnoge istraživače da pokušaju da ovu tehniku prilagode petljama sa uslovnim grananjima [17, 30, 39]. Tri značajnije tehnike će ovde biti ukratko opisane: *hijerarhijska redukcija* (engl. *Hierarchical Reduction* [17]), *if-konverzija sa predikatskim izvršavanjem* (engl. *If-conversion with Predicated Execution* [30]) i *prošireno raspoređivanje po modulu* (engl. *Enhanced Modulo Scheduling*, EMS [39]).

Hijerarhijska redukcija najpre raspoređuje operacije iz obe grane svakog IF-THEN-ELSE konstrukta jedne iteracije nezavisno, nekom globalnom tehnikom raspoređivanja (npr. *List Scheduling*, [10]). Zatim spaja tako dobijene dve grane u jednu, uzimajući u obzir uniju korišćenja resursa obe grane. Uzima se unija, a ne zbir, pa se dozvoljava da dve operacije iz različitih grana koriste isti resurs u istom ciklusu, jer se one inače nikada ne izvršavaju istovremeno. Tako dobijen spoj se proglašava za pseudooperaciju koja ima sloenu tabelu korišćenja resursa dobijenu na opisani način unijom pojedinih tabela. Ovakva pseudooperacija zatim ulazi u postupak raspoređivanja po modulu, ravnopravno sa ostalim operacijama i pseudooperacijama iste petlje. Nakon raspoređivanja po modulu, kada se regenerira razvoj pseudooperacija u IF-THEN-ELSE konstrukte i kopiranjem operacija raspoređenih u istom ciklusu u obe grane konstrukta. *RII* se određuje nakon formiranja pseudooperacija prema najkorišćenijem resursu po svim stazama izvršavanja.

Prednost hijerarhijske redukcije je to ona upravo uzima uniju korišćenja resursa, pa se ne može dogoditi konflikt na resursu između dve operacije iz različitih grana istog uslova. Druga prednost je to ova tehnika ne zahteva posebnu hardversku podršku. Nedostatak je to se konflikti na resursima daleko će se javljati nego kod drugih tehnika, jer pseudooperacije imaju sloenu na način korišćenja resursa, a zapravo se sastoje od jednostavnijih operacija koje druge tehnike posmatraju nezavisno. Zato se minimalni teorijski *RII* često ne može postići [39].

If-konverzija sa predikatskim izvršavanjem oslanja se na hardversku podršku, tzv. *predikatsko izvršavanje* [30]. Procesor poseduje poseban *predikatski* registarski fajl koji predstavlja zapravo skup jednobitnih adresibilnih registara. Postoje operacije koje postavljaju ili brišu ove registre u odnosu na ishod nekog uslova¹². Zapravo se binaran rezultat uslovne operacije (testa) upisuje u specifikovani predikatski registar. Dalje, svakoj operaciji može se pridružiti *predikat* - uređen par kojim se specifikuje predikatski registar i njegova vrednost. Procesor izvršava datu operaciju samo ako specifikovani predikatski registar ima specifikovanu vrednost. Tačnije rečeno, operacija se svakako izvršava, ali poseban hardver sprečava upis njenog rezultata ukoliko dati predikatski registar nema navedenu vrednost. Na ovaj način se *svaka* pojedina operacija može *usloviti* proizvoljnim uslovom.

Pre nego što se vrši raspoređivanje, kada sa uslovnim grananjima se transformira u pravolinijski kod, uz predikatsko izvršavanje. Operacija ispitivanja uslova (IF) se pretvara u operaciju postavljanja predikata (upis u predikatski registar). Operacijama iz grana uslova se pridružuju predikati, tako da praktično svaka uslovna operacija, osim svojih osnovnih operanada, koristi (čita) i vrednost predikata. Na ovaj način se, zapravo, kontrolna zavisnost [5] pretvara u zavisnost po podacima. Ovakva konverzija naziva se *if-konverzijom*.

¹²Ovo je zapravo generalizacija pojma flega u tradicionalnim CISC procesorima. Umesto da postoje specifični flegovi koji se postavljaju u odnosu na rezultat nekih aritmetičko-logičkih operacija, predikatski registri su potpuno ravnopravni i bilo koji uslov može postaviti bilo koji predikat - RISC princip.

Kada se na ovaj način dobije pravolinijski kod, sprovodi se standardan postupak raspoređivanja po modulu.

Prednost ove tehnike je jednostavnost koja se ogleda u principu "svođenja na prethodno". Nedostaci su mnogobrojni. Prvo, ova tehnika zahteva postojanje opisane hardverske podrške. Drugo, tokom raspoređivanja se ne razlikuju operacije koje potiču iz različitih grana istog uslova, pa se može dogoditi da one generiraju u konflikt na resursu koji realno ne postoji. Za razliku od hijerarhijske redukcije koja uzima uniju, ova tehnika uzima zbir korišćenja resursa od strane operacija iz različitih grana uslova. Osim toga, i sam hardver se ne ponaša drugačije: operacija se svakako izvršava, zauzimajući resurs, bez obzira da li će se njen rezultat upisati ili ne. Najzad, iako da i najvažnije, ova tehnika se oslanja na pretvaranje kontrolne zavisnosti u zavisnost po podacima, čime se eliminiše mogućnost spekulativnog izvršavanja, pa se potencijalno gubi na paralelizmu.

Prošireno raspoređivanje po modulu (EMS) pokušava da ujedini prednosti od obe navedene tehnike. Naime, kod ove tehnike se najpre vrši if-konverzija i označavanje operacija predikatima. *RII* se uzima kao maksimum korišćenja resursa po svim stazama izvršavanja p iz skupa P staza izvršavanja (procesor poseduje skup resursa R , postoji n_r resursa kategorije r , na stazi p resurs r se koristi c_r ciklusa takta):

$$RII = \max_{p \in P} \left(\max_{r \in R} \left\lfloor \frac{c_r}{n_r} \right\rfloor \right)$$

Ovde se uzima maksimum korišćenja resursa po stazama izvršavanja, a ne unija korišćenja resursa po svim stazama. Na taj način se dobija potencijalno manji *RII* nego kod hijerarhijske redukcije. Kada se izvrši raspoređivanje po modulu kao kod predikatskog izvršavanja, dobijeni raspored se transformiše nazad u IF-THEN-ELSE konstrukte, tako da dobijeni kod ne zahteva hardversku podršku.

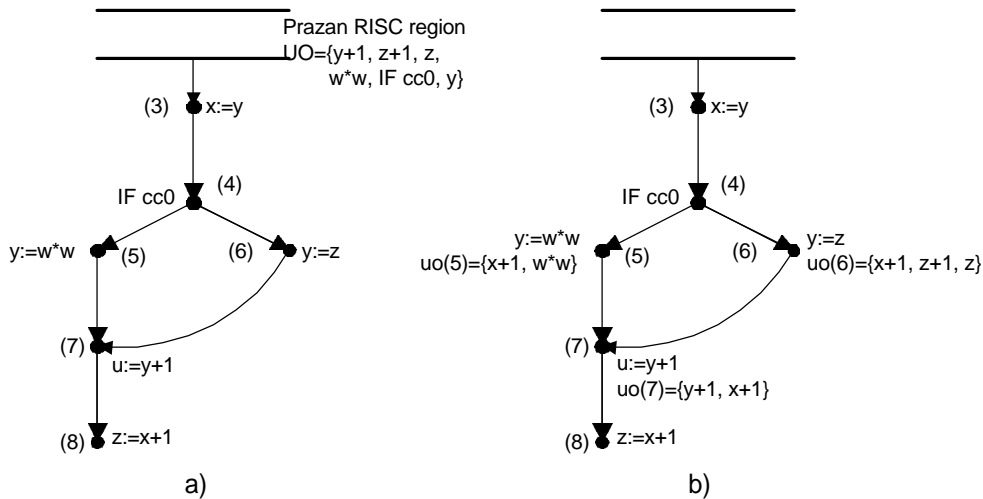
U [39] je empirijski pokazano da EMS tehnika postiže bolje rezultate od hijerarhijske redukcije, ali ne to slabije od predikatskog izvršavanja u slučaju da procesor podržava samo jedan uslovni skok po ciklusu. U slučaju da procesor može da izvrši više uslovnih skoka po ciklusu, EMS potencijalno postiže bolje rezultate od obe prethodne tehnike. Kako EMS ne zahteva posebnu hardversku podršku, u tom smislu ima prednost u odnosu na predikatsko izvršavanje. Međutim, i ova tehnika poseduje dve ključne mane. Prvo, i ovde se kontrolna zavisnost pretvara u zavisnost po podacima, pa se gubi na potencijalnom paralelizmu. Drugo, kao i ostale tehnike koje daju konstantan *II*, gubi se upravo na onome što uslovna grananja pružaju - različito vreme izvršavanja za različite ishode uslova.

3.3. Raspoređivanje koje daje promenljiv interval iniciranja

Jedina značajnija tehnika data u otvorenoj literaturi koja proizvodi promenljiv interval iniciranja je tehnika *proširenog protočnog raspoređivanja* (engl. *Enhanced Pipeline Scheduling*, EPS [24]). Ova tehnika polazi od ideje pomeranja operacija kao kod tehnike *Percolation Scheduling* [27, 15], ali uz mnoge modifikacije. Konačna varijanta predstavlja rezultat dugotrajnog istraživanja [7, 8, 24]. U prvim verzijama, tehnika se oslanjala na posebnu hardversku podršku i bila namenjena isključivo za VLIW mašine [7, 8]. Konačna verzija može se primeniti u optimizaciji koda i za RISC, superskalarne i VLIW procesore [24]. Tehnika daje rešenje za globalnu optimizaciju koda, i van petlji, i za optimizaciju ugnježanih petlji sa i bez uslovnih grananja tehnikom softverske protočnosti, uz poštovanje ograničenja na resursima.

Najpre definiše se opisan postupak globalne optimizacije koda, a zatim definiše se objektni način tretiranja petlji pomoću softverske protočnosti. Pri optimizaciji programskog koda bez petlji polazi se od acikličkog grafa kontrole toka (engl. *control flow graph*) dela programa koji se optimizuje.

Vorovi ovog grafa su operacije (izvorne instrukcije prevedenog, neoptimizovanog programa), a od svakog Ľvora polazi usmerena grana prema Ľvoru koji predstavlja narednu operaciju u izvr Ľvanju; od operacija uslovnog skoka polaze po dve grane. Prva operacija programa je koren grafa. Tehniku Ľno pratiti na primeru programa Ľiji je graf kontrole toka dat na Slici 10a.



Slika 10: AcikliĽki graf kontrole toka jednog dela programa koji se optimizuje po metodi EPS.
(a) Polazni graf sa naznaĽenim operacijama i polaznim praznim RISC regionom. (b) Skupovi uo za operacije

Zatim se za svaki Ľvor (operaciju) n koji predstavlja poĽetak baziĽnog bloka defini Ľe tzv. uo skup (engl. *unifiable ops*). To je skup *desnih strana* (*rhs*, engl. *right-hand side*) svih operacija koje mogu da se pomere ispred (izvr Ľe pre) date operacije. PraktiĽno, to je skup onih operacija u programu koje se nalaze iza date operacije n u grafu kontrole toka, a nisu zavisne po podacima od date operacije. Uzimaju se samo desne strane operacija, jer se leve strane (odredi ni registri) odreĽuju tek nakon rasporeĽivanja operacija, Ľime se elimini Ľu izlazne i antizavisnosti. Osim ovih zavisnosti, mogu Ľe eliminisati i jedan broj pravih zavisnosti tzv. *kombinovanjem* (engl. *combining* [26]). Na primer, na Slici 10, operacija 7 se mo Ľe pomeriti ispred operacije 6 uz promenu njene desne strane iz $y+1$ u $z+1$, to predstavlja kombinovanje. Pri tome se uo skupovi izraĽunavaju postupno, od operacija na dnu grafa prema vrhu, tako da se uo skup za neku operaciju n dobija od uo skupa operacije koja joj neposredno sledi, iz koga su izbaĽene sve desne strane koje zavise po podacima od operacije n (a ne mo Ľe se ta zavisnost eliminisati kombinovanjem) i kome je dodata desna strana same operacije n ; za operacije uslovnih skokova, uo skup se dobija kao unija uo skupova operacija koje neposredno slede iza operacije uslova. Na Slici 10b prikazani su uo skupovi za operacije na poĽetku baziĽnih blokova.

Kada se izraĽunaju uo skupovi, ispred korena grafa formira se jedan prazan tzv. RISC region. RISC region predstavlja skup svih operacija koje se mogu izvr Ľavati paralelno. IzraĽunava se zatim uo skup ovog regiona kao unija uo skupova svih baziĽnih blokova koji slede u grafu iza baziĽnog bloka na Ľijem se poĽetku nalazi region (Slika 10a). U samom radu [24] se ne pominje eksplicitno, ali je oĽigledno da je uo skup zapravo skup svih operacija slobodnih na vrhu [15] u grafu zavisnosti po podacima datog programa, s tim da je eliminacija izlaznih i antizavisnosti izvedena dinamiĽkim preimenovanjem, a deo pravih zavisnosti eliminisan kombinovanjem. Razlika u odnosu na neke druge tehnike globalnog rasporeĽivanja je u tome to, kada se izraĽuna ovaj skup slobodnih operacija, ne rasporeĽuju se odmah sve te operacije, ve Ľe samo one kojima ograniĽanja na resursima to dozvoljavaju; ostale operacije ostaju za kasnije rasporeĽivanje.

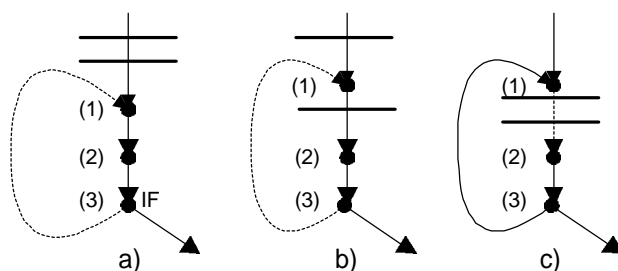
IzraĽunavanje uo skupa RISC regiona je prva faza algoritma. Kao to je reĽeno, uo skup regiona predstavlja skup operacija koje Ľe potencijalno biti rasporeĽene na poĽetak izvr Ľavanja.

Druga faza je izbor operacija koje ~~abit~~ zaista preme tene. Pri tome se primenjuju heuristike koje u [24] nisu precizno navedene. Rešeno je samo da je to poseban problem koji nije u vezi sa samom idejom algoritma. Date su ipak neke smernice: biraju se one operacije koje imaju manji nivo spekulativnosti, to se može pratiti pri izražavanju uo skupova bazičnih blokova. Ako se neka desna strana operacije pojavljuje samo u uo skupu jedne grane iza operacije uslova, njen nivo spekulativnosti se povećava za jedan ispred tog uslova; inače, ako se desna strana operacije nalazi u obe grane, njen novi nivo spekulativnosti ispred uslova postaje veći od dva nivoa spekulativnosti u te dve grane. Najzad, na ovom mestu se uzimaju u obzir i ograničenja na resursima, tako da se raspoređuju samo one operacije koje je paralelno izvršavanje moguće sa datim resursima.

Treća faza je samo pomeranje operacija, odnosno njihovo sme tanje u RISC region. U [24] je detaljno opisan ovaj postupak, a mi ga ovde nećemo obja njavati, većemo napomenuti da se pomeranja odvijaju po pravilima sličnim tehnici *Percolation Scheduling*. Rešeno samo to da se u ovom koraku a uriraju iterativno uo skupovi du onih staza du kojih se vr i pomeranje operacija. Na taj način se smanjuje količina računanja u toku raspoređivanja. Rezultat pomeranja je graf kontrole toka unutar RISC regiona koji eventualno ima više grana koje izlaze iz regiona: to se dešava u slučaju da je u region pomerena neka operacija uslovnog skoka.

Najzad, dobijeni popunjeni RISC region se proglašava za koren grafa, i na svakoj grani koja izlazi iz regiona, na granici regiona, formira se novi prazni RISC region koji se zatim popunjava na isti opisani način. Postupak se ponavlja dok se ne dođe do kraja grafa programa.

Sada ćemo objasniti kako se optimizuju petlje pomoću softverske protočnosti. Polazi se od cikličkog grafa kontrole toka petlje, pri čemu povratne grane petlje predstavljaju grane koje formiraju zatvorene puteve u grafu. Najpre se, uz privremenu eliminaciju povratnih grana petlje, aciklički graf topološki sortira i operacijama dodeljuju redni brojevi *seqno* koji označavaju njihovu poziciju u ovako sortiranom grafu. Na taj način sve grane u cikličkom grafu osim povratnih polaze od operacije sa manjim prema operaciji sa većim *seqno*. Zatim se ispred prve operacije tela petlje (korena grafa tela petlje) formira prazan RISC region koji se popunjava operacijama na opisani način, s tim da se ne dozvoljava pomeranja du grane koja polazi od operacije sa većim prema operaciji sa ni im *seqno* (to su u prvom koraku povratne grane petlje), kao što je pokazano na Slici 11a i b. Kada se na ovaj način prvi region popuni, svim operacijama iz ovog regiona dodeljuju se isti *seqno*, već od do tada najvećeg *seqno* u grafu. Zatim se iza svih izlaznih grana ovog regiona formiraju novi prazni regioni, i popunjavaju na isti način. Opet se ne dozvoljava pomeranje preko grana koje polaze od većeg prema manjem *seqno*, čime se posti e softverska protočnost (Slika 11c): sada operacije iz prethodno popunjenog regiona, koji sada pripada narednoj iteraciji, pomeraju u novi region koji pripada tekućoj iteraciji. Zbog odgovarajuće tehnike pomeranja operacija preko spoja kopiranjem u obe grane spoja, posti e se automatski i kreiranje pretpetlje i postpetlje. Treba primetiti da se na ovaj način operacije pomeraju samo nagore.



Slika 11: Primer optimizacije koda petlje po metodi EPS. (a) i (b) Primena tehnike globalnog raspoređivanja EPS. (c) Softverska protočnost.

Najzad, objasnimo kakva je strategija optimizacije celog programa. Najpre se pronalaze najdublje ugne čone petlje. One se optimizuju na opisani način. Zatim se operacije koje su "ispale" iz tela ovako optimizovane petlje (praktično predstavljaju pretpetlju i postpetlju) mešaju sa operacijama iz tela okružujuće petlje i sa njima podle u optimizaciji na potpuno isti način. Telo optimizovane ugne čone petlje se tretira posebno, kao superoperacija koja se ne razlaže. Zatim se optimizuje okružujuća petlja, itd. sve petlje do krajnje spoljašnje petlje. Pretpetlja i postpetlja ove spoljašnje petlje se mešaju sa kodom van petlje koji se optimizuje opisanom globalnom tehnikom.

Prednosti opisane tehnike EPS su višestruke. Najpre, ona proizvodi promenljiv interval iniciranja, što prirodno daje predispozicije za dobre rezultate. To je u mnogim radovima i potvrđeno. Dalje, ona daje veoma efikasan i krajnje specifikovan postupak za optimizaciju koda petlje sa uslovnim grananjima na nivou assemblerskog koda. Ovaj postupak se direktno može implementirati. Postupak kao sporedni efekat daje i pretpetlju i postpetlju, što ostale tehnike ne čine direktno, već se ovom problemu mora posvetiti posebna pažnja. Kontrola ograničenja na resursima i dubine spekulativnog izvršavanja je jednostavna. Izražavanje svih potrebnih podataka o zavisnostima i transformacijama koda nije glomazno. Najzad, ova tehnika dozvoljava proizvoljno pomeranje operacija duž samo nekih staza (proizvoljnu dubinu softverske protočnosti).

Zbog svega što je rečeno, opisani postupak teško da poseduje mane. Ipak, rečeno sledeće postoji izvesni stepen inhibicije pomeranja operacija zbog usputnog tretiranja ograničenja na resursima. Naime, u postupku softverske protočnosti se neka operacija smešta u region samo ako to resursi dozvoljavaju; u narednom koraku ovako popunjeni region postaje izvor operacija za selidbu iz naredne iteracije u tekuću. Na taj način, ako operacija nije smeštena u region zbog nedostatka resursa, neće joj biti omogućeno ni da pređe u prethodnu iteraciju, gde bi moglo da bilo uslova za njeno izvršavanje. Zbog toga se može desiti da operacije i resursi budu loše raspoređeni. Može se reći da je ovime plaćena dug krajnjoj praktičnosti i efikasnosti algoritma. Uzrok je baš u tome što se konflikti na resursima razrešavaju pre nego što se operacija raspoređi. Drugi pravac u raspoređivanju koji se bitno primenjen i u ovom radu je da se izvrši najpre raspoređivanje operacija uz tretiranje samo zavisnosti po podacima (praktično pretpostavljajući neograničene resurse), a da se konflikti na resursima rešavaju posle toga, nekom od postojećih tehnika globalnog raspoređivanja, npr. *List Scheduling*. Drugo, u originalnim radovima nije naglašeno, ali izbor operacije za raspoređivanje u region samo na osnovu raspolaganja resursa i nivoa spekulativnosti nije dovoljno dobro, jer se time ne favorizuje ranije izvršavanje operacija koje se nalaze na kritičnom putu zavisnosti po podacima. Uopšte, dužine puteva u grafu zavisnosti po podacima se nigde ne spominju. Ipak, ovaj nedostatak se može ispraviti poboljšanjem heuristika koje biraju operaciju za

raspoređivanje, ali se time op ti metod drastiĹno uslo njava, jer inaĹe uop te ne koristi graf zavisnosti.

U zakljuĹku ove glave mo emo reći da je pouka koja sledi iz celokupne analize postojeće re enja to da se mora ponavljati generalan postupak optimizacije koda petlji sa uslovnim grananjima koji ne imaju Ĺvrsto teorijsko opravdanje, davati promenljiv interval iniciranja i biti primenljiv u praksi. Ovaj rad ima za cilj da predlo i jedno takvo re enje.

4. Analiza problema

U ovoj glavi iznose se rezultati analize problema postavljenog u prethodnom izlaganju. Najpre se uvode pretpostavke koje se koriste u daljem izlaganju i objašnjava način modelovanja elemenata bitnih za ovaj problem: softverske protočnosti, operacija, zavisnosti po podacima i kontrolnih zavisnosti. Zatim se iznose zaključci koji ukazuju na izvore paralelizma u petljama sa uslovnim grananjima.

4.1. Model programskog koda koji se optimizuje

Za dalje izlaganje je bitno uvesti pretpostavke na koje se rad oslanjati. Ovo stoga što u raznim pristupima istom problemu postoje različiti modeli istih pojmova. Pojam softverske protočnosti se u literaturi različito definiše, a mi ćemo usvojiti jedan model koji je najviše prihvaćen. Dalje, operacije programa i zavisnosti po podacima se tretiraju na različitim nivoima: na nivou izvornog koda (programskih promenljivih) i na nivou asemblerskog koda (registara i memorijskih lokacija). Najzad, pojam kontrolne zavisnosti je ključan za optimizaciju koda sa uslovnim grananjima uopšte. Ovde će biti definisani svi pomenuti modeli.

4.1.1. Model softverske protočnosti

U [10] se pojam softverske protočnosti definiše krajnje apstraktno. Slobodno interpretirano, tamo se softverskom protočnošću smatra svaki raspored operacija petlje, pri kome se istovremeno izvršavaju neke operacije koje potiču iz različitih iteracija polazne petlje. Pod ovakvu definiciju spadaju time i DOALL petlje, kod kojih ne postoje međiteracione zavisnosti po podacima. Zbog toga se sve iteracije DOALL petlje mogu izvršavati uporedo, a stvaran broj iteracija koje se izvršavaju uporedo zavisi od raspoloživosti resursa. Na taj način se može postići prosečno izvršavanje više od jedne iteracije u jednom ciklusu takta.

Nasuprot pomenutom gledajući tu stoji opet prihvaćeno ograničenje koje predstavlja suptilnu koncepciju softverske protočnosti. Ovo ograničenje se sastoji u tome da se u svakom ciklusu izvršavanja optimizovane petlje inicira najviše jedna iteracija petlje. Ekvivalentno, prosečno izvršavanje jedne iteracije optimizovane petlje je najmanje jednako jedan ciklus takta. Reč prosečno stoji zbog toga što ukupno izvršavanje optimizovane petlje obuhvata i izvršavanje pretpetlje i postpetlje. Za izvršavanje koje podrazumeva iniciranje manje od p iteracija polazne petlje, gde je p broj iteracija koje inicijalizuje pretpetlja optimizovane petlje, jezgro optimizovane petlje se i ne izvršava.

Zbog postojanja pretpetlje i postpetlje, kako je upravo objašnjeno, broj iteracija transformisane petlje koje se izvršavaju u optimizovanom programu je manji od broja iteracija polazne petlje u ekvivalentnom izvršavanju. Zato se ovde uvodi definicija softverske protočnosti na sledeći način.

Definicija 2: (Softverska protočnost) Optimizovana petlja P' je dobijena od polazne petlje P tehnikom softverske protočnosti ako (po def.) za svako izvršavanje I polazne petlje P koje sadrži n iteracija i ekvivalentno izvršavanje I' petlje P' sa n' iteracija važi:

$$\lim_{n \rightarrow \infty} \frac{n}{n'} = 1 \quad \square$$

Ekvivalentno, za broj iteracija n polazne petlje P i broj iteracija n' optimizovane petlje P' , pri ekvivalentnim izvršavanjima, važi da je:

$$n = n' + k$$

za neki konstantan ceo broj k . Ovaj broj k predstavlja praktično broj iteracija polazne petlje koje se izvršavaju unutar pretpetlje i postpetlje.

Slobodnije rečeno, broj iteracija polazne petlje se ovljučava u optimizovanoj petlji. Očigledno je da zbog toga u svakom ciklusu takta mora da se inicira najviše jedna iteracija petlje, jer izvršavanje svake iteracije traje bar jedan ciklus. Ovo ograničenje se ogleda u tome da se i DOALL petlje moraju izvršavati u bar onoliko ciklusa takta, koliko iteracija sadrži izvršavanje polazne petlje. Drugačije tumačenje softverske protočnosti koje bi polazilo od toga da se u svakoj iteraciji optimizovane petlje nalazi samo po jedna instanca svake operacije iz polazne petlje, kao što je biti pokazano, ne bi bila korektna.

Iz navedene definicije sledi jedno intuitivno shvatanje koje ne možemo formalno iskazati:

Zaključak 1: *(O periodičnosti rasporeda operacija) Svako rešenje dobijeno softverskom protočnošću mora da poštuje periodičnost. Ova periodičnost se ogleda u tome da svaka iteracija optimizovane petlje može da bude proizvoljna po redu u izvršavanju petlje. Drugim rečima, svaka transformacija koja se izvrši nad kodom koji predstavlja neku specifičnu iteraciju u nekom specifičnom izvršavanju petlje, mora da ima kompenzacione transformacije koje održavaju semantiku programa i nad kodom koji predstavlja sve ostale iteracije svih ostalih izvršavanja.*

4.1.2. Model operacija i zavisnosti po podacima

Kao ulaz u analizu i optimizaciju smatraćemo kod u nekoj posrednoj formi koja u sebi nosi informacije o pristupanju indeksiranim promenljivama (vektorima). Operacijom se smatra funkcija uređene n -torke operanada čiji rezultat može biti takodje operand neke operacije. Na ovaj način se operacije nadovezuju po pravoj zavisnosti po podacima. Osim na mestima koja će biti prodiskutovana u 4.2.3., smatraćemo da postoje samo prave zavisnosti po podacima, a da su izlazne i antizavisnosti eliminisane preimenovanjem [15, 24]. Dužina trajanja operacije je prirodan broj pridružen operaciji. Zavisnosti po podacima predstavljajućemo grafom u kome su operacije predstavljene čvorovima, a zavisnosti usmerenim granama [15], pri čemu je dužina grane jednaka trajanju operacije od koje grana polazi.

Treba napomenuti da model opisan u ovom radu ne zahteva obavezno ukidanje izlaznih i antizavisnosti. U 4.2.3. je biti pokazano koje izlazne i antizavisnosti ne treba eliminisati. Nije neophodno eliminisati ni ostale, ali se time gubi na paralelizmu. Ukoliko postoji neka izlazna ili antizavisnost koja nije eliminisana, grana u grafu čini dužinu koja odgovara vremenskom intervalu (u ciklusima) koji mora da protekne od iniciranja operacije iz koje grana polazi, do iniciranja operacije u koju grana ulazi¹³.

Kada u kodu postoje uslovne operacije, grana u grafu (cikličkom ili acikličkom) postoji ako postoji zavisnost po podacima između te dve operacije i postoji staza izvršavanja polaznog (sekvencijalnog) programa u kojoj se obe te operacije izvršavaju u odgovarajućem redosledu. Dakle, u grafu zavisnosti tela petlje sa uslovnim grananjima postojeće sve grane za koje postoji mogućnost da u izvršavanju uzrokuju zavisnosti po podacima.

U svakom slučaju, smatra se da je zavisnosti po podacima moguće utvrditi u vreme prevođenja. Takođe, smatra se da su sve klasične metode optimizacije [1], kao što je eliminacija indukcionih promenljivih i neposredno ugrađivanje koda pozvane procedure (engl. *inlining*), izvršene pre optimizacije koja se ovde razmatra.

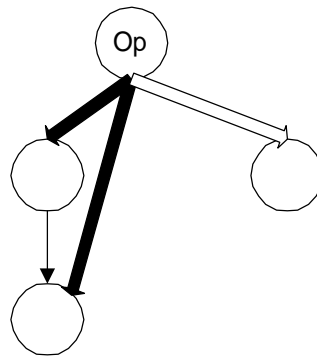
¹³Ova dužina može biti i manja od nule, npr. kod antizavisnosti. Bitno je da prva operacija proćita operand pre nego što druga operacije upiše rezultat, pri čemu prva operacija može biti znatno duže trajanja, pa ne mora da se završi pre nego što druga poćne.

4.1.3. Model kontrolnih zavisnosti

Uslovnom operacijom smatraemo operaciju koja ima proizvoljan broj operandata, ali čiji rezultat nije operand nijedne druge operacije. Drugim rečima, smatraemo da ne postoji prava zavisnost po podacima koja polazi od uslovne operacije. Sa druge strane, dozvoljeno je da od ove operacije polaze antizavisnosti. Uslovna operacija ima dva ishoda, T (*true*) i F (*false*).

Od uslovne operacije polaze *kontrolne zavisnosti* [15]. Neka operacija je kontrolno zavisna od operacije uslova, ako je njeno izvršavanje uslovljeno samo jednim ishodom uslovne operacije. Drugim rečima, kontrolno zavisnu operaciju treba izvršiti samo ukoliko je rezultat uslovne operacije jedan od T ili F .

Kontrolne zavisnosti možemo označavati u grafu granama dvostruke debljine: grana T je puna, a grana F je prazna, kao na Slici 12.



Slika 12: Predstavljanje kontrolnih zavisnosti. Puna linija dvostruke debljine označava granu T , a prazna granu F uslovne operacije Op .

Pretpostavljamo da je pre optimizacije koda izvršena analiza i pronalazjenje kontrolno zavisnih operacija, za šta postoje efektivni postupci. To znači da su operacije koje se nalaze u jednoj grani strukture IF-THEN-ELSE od koje potiče uslovna operacija kontrolno zavisne od te uslovne operacije, a da operacije van ove strukture to nisu.

Uslov izlaska iz petlje nećemo tretirati u ovom radu, pa će primeri petlji u daljem tekstu biti predstavljeni simboličkim konstruktom LOOP-ENDLOOP.

4.1.4. Model iteracionih distanci

Pretpostavljamo da se iteraciona distanca zavisnosti po podacima može pronaći u vreme prevođenja nekim od postojećih postupaka [15]. Takođe možemo smatrati da premetanje iz jedne u drugu iteraciju operacije koja operiše elementom vektora ne utiče na trajanje te operacije, što bi bilo slučaj kada bi se u obzir uzimalo promenjeno izračunavanje indeksa. Na primer, premetanjem operacije:

```
...=a[i]
```

u prethodnu iteraciju dobija se operacija:

```
...=a[i+1]
```

Pretpostavljamo da se ovime ne unosi dodatna operacija izračunavanja indeksa $i+1$. Ovaj problem se može rešiti uvođenjem posebnih indeksnih promenljivih za svaki poseban pristup, praktično uvođenjem pokazivača, na račun povećanog korišćenja registara. A uriranje ovako dobijenih indeksnih promenljivih se takođe ne razmatra, jer su a uriranje nezavisna i ne unose dodatno vreme, osim što dodatno koriste resurse.

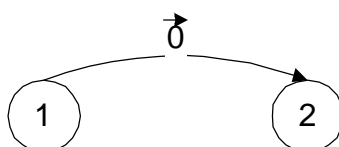
Postoji jedan poseban efekat koga proizvode uslovna grananja u petljama, a na koji u otvorenoj literaturi do sada nije eksplicitno ukazano. Posmatrajmo sledeći primer:

```

      LOOP
      ...
      IF (Test) THEN
1:      X=...
      ENDIF
2:      ...=f(X)
      ...
      ENDLOOP

```

U ovom primeru operacija 2 zavisi po podacima od operacije 1 iz iste iteracije, ukoliko je uslov Test zadovoljen. Ukoliko to nije slučaj, postoji prava zavisnost između operacije 1 iz prethodne iteracije i operacije 2 iz tekuće iteracije, ukoliko je uslov Test prethodne iteracije bio zadovoljen. Ako ni to nije slučaj, postoji zavisnost operacije 2 od operacije 1 iz dve iteracije ranije, ukoliko je njen uslov ispunjen, i tako dalje. Dakle, postoji *promenljiva* iteraciona distanca zavisnosti operacije 2 od operacije 1 koja može uzimati sve celobrojne vrednosti počev od 0 pa naviše. Ovakvu promenljivu distancu može u grafu označavati kao na Slici 13.



Slika 13: Označavanje promenljive iteracione distance zavisnosti po podacima (označena je samo distanca). Broj 0 označava da iteraciona distanca može biti najmanje 0, a može biti i 1, 2 itd.

Treba primetiti nekoliko stvari. Prvo, ovakav slučaj se može dogoditi samo kod operacija nad skalarnim promenljivim (X u prethodnom primeru). Drugo, upravo zbog ovoga, najmanja iteraciona distanca može biti samo 0 (kao u primeru) ili 1 (ako se primer izmeni tako da operacija 2 prethodi IF-THEN-ELSE strukturi); ne može se postići veća, jer bi to značilo operisanje nad vektorom. Treće, opet zbog dejstva nad skalarom, pored prave zavisnosti, postoji uvek i izlazna zavisnost operacije od same sebe sa promenljivom distancom počev od 1 (operacija 1 u primeru), kao i antizavisnost sa promenljivom distancom (operacije 1 od operacije 2 u primeru). U odeljku 4.2.3. će slučaj promenljive distance biti dalje diskutovan.

4.2. Izvori paralelizma

U ovom poglavlju detaljno se analiziraju aspekti koji su specifični za petlje sa uslovnim grananjima, a mogu da utiču na povećanje paralelizma u izvršavanju.

4.2.1. Softverska protočnost komponenti jake povezanosti

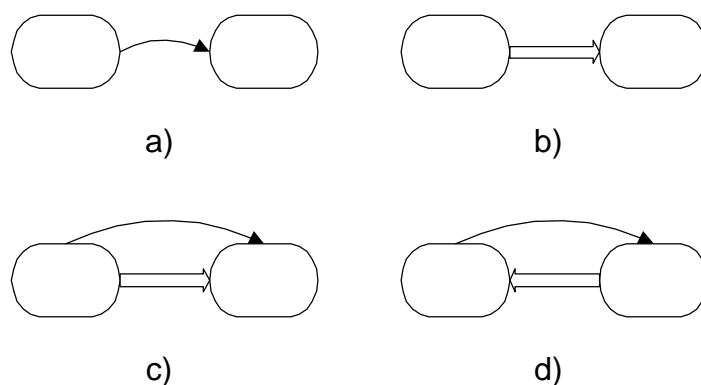
U 3.1.1. je navedeno da zatvoreni putevi u grafu zavisnosti po podacima petlje predstavljaju ograničenja paralelizma za petlje bez uslovnih grananja. Drugačije rečeno, operacije koje se nalaze u komponentama jake povezanosti cikličkog grafa zavisnosti su kritične za raspoređivanje. U [14] je pokazano kako se primenom tehnike softverske protočnosti na nivou komponenti jake povezanosti cikličkog grafa petlje može postići da grane koje spajaju komponente jake povezanosti uopšte ne utiču na raspoređivanje operacija, pod uslovom da su resursi neograničeni. Suština je u tome da se komponente jake povezanosti "razmiču" tako da se grane između njih "rastežu", to zapravo znači

dovoljno ranije izvršavanje operacija iz polazne komponente, tako da operacije iz zavisne komponente ne moraju čekati na njihovo izvršavanje. Ova pojava predstavlja prvi potencijalni izvor paralelizma kod petlji, pa je logično ispitati tu pojavu i kod petlji sa uslovnim grananjima.

Ako se pretpostavi da od uslovne operacije ne polaze grane zavisnosti po podacima, to je slučaj ako se eliminišu u izlazne i antizavisnosti, onda uslovne operacije predstavljaju zapravo pojedinačne komponente jake povezanosti grafa do kojih samo dolaze grane zavisnosti po podacima. Sa druge strane, neuslovne operacije imaju svojstvo da u njima mogu samo ulaziti grane kontrolnih zavisnosti. Neuslovne operacije se mogu razvrstati u komponente jake povezanosti, s tim da između komponenti jake povezanosti, osim grana zavisnosti po podacima, postoje i grane kontrolnih zavisnosti.

Razmotrimo najpre komponente jake povezanosti koje čine samo uslovne operacije. Prema [14], da bi se eliminisalo ograničenje koje grane koje ulaze u jednu ovakvu komponentu predstavljaju, potrebno je zakasniti izvršavanje svih operacija (zapravo samo jedne uslovne) u odnosu na operacije od kojih grane polaze. Na ovaj način se potencijalno dobija na paralelizmu, jer se uslovne operacije mogu izvršavati uporedo sa nekim drugim operacijama. Sa druge strane, međutim, kašnjenje uslovnih operacija dovodi do kasnijeg razrešavanja uslova, pa se time povećava stepen spekulativnosti u opštem slučaju.

Razmotrimo sada generalno komponente jake povezanosti, bilo da ih čine usamljene uslovne operacije ili proizvoljan broj neuslovnih operacija. Posmatrajmo različite slučajeve koji mogu da nastupe između dve komponente jake povezanosti između kojih mogu postojati zavisnosti po podacima i kontrolne zavisnosti istovremeno ili samo jedne od njih. Prvi slučaj (Slika 14a) je da između dve komponente postoje samo zavisnosti po podacima, to se svodi na slučaj petlji bez grananja. Drugi slučaj (Slika 14b) je da postoje samo kontrolne zavisnosti. U ovom slučaju "razmicanje" komponenti treba da se obavi u smeru u kome se zavisnosti "raste u", čime se postiže ranije razrešavanje uslova i smanjenje spekulativnosti. Treći slučaj (Slika 14c) je da između komponenti postoje i kontrolne i zavisnosti po podacima u istom smeru, to predstavlja jednostavnu kombinaciju prethodna dva slučaja i ne donosi nove probleme. Poslednji slučaj (Slika 14d) je kada su kontrolne i zavisnosti po podacima u suprotnom smeru. U ovom slučaju se "razmicanje" komponenti mora obaviti u smeru u kome se "razvlače" grane zavisnosti po podacima, jer se ove zavisnosti ne mogu eliminisati. Nasuprot njima, ovim "razvlačenjem" kontrolne zavisnosti postaju kritične, u smislu da se kasni razrešavanje uslova, čime se povećava spekulativnost.



Slika 14: Različiti slučajevi postojanja zavisnosti između dve komponente jake povezanosti. (a) Ne postoje kontrolne zavisnosti. (b) Ne postoje zavisnosti po podacima. (c) Zavisnosti po podacima i kontrolne zavisnosti su u istom smeru. (d) Zavisnosti po podacima i kontrolne zavisnosti su suprotnih smerova.

Najzad, posmatrani slučajevi ukazuju na efekte "razmicanja" samo dve komponente jake povezanosti. Ako graf petlje sadrži više komponenti, združeni efekti mogu biti u opštem slučaju

jako slo eni, tako da se ovde diskusija o softverskoj protoĹnosti komponenti jake povezanosti zaustavlja.

Treba samo primetiti da se veaada nazire sledeaeefekat. Ukoliko se eli to efikasnije paralelno izvr avanje, mo e doaeo toga da se uslovne operacije preme taju u iteracije koje se izvr avaju mnogo ranije nego iteracije u kojima se nalaze kontrolno zavisne operacije (sluĹajevi na Slici 14b i c). To znaĹi da eebiti potrebno pamtiti rezultate uslovnih operacija vi e prethodnih iteracija, jer se njihov ivotni vek prote e na kontrolno zavisne operacije rasporeĹne u vi e iteracija. U tim narednim iteracijama eeneke operacije zavisiti od rezultata uslova iz neke prethodne iteracije, a neke od istog uslova ali iz neke druge prethodne iteracije itd.

4.2.2. *Spekulativno izvršavanje*

Spekulativno izvr avanje [15] predstavlja izvr avanje neke kontrolno zavisne operacije pre uslovne operacije od koje ona zavisi. Drugim reĹima, spekulativno izvr avanje predstavlja naĹin eliminisanja kontrolnih zavisnosti. U [15] se pokazuje kako se efekat neke operacije koja je spekulativno izvr ena mo e poni titi kompenzacionom operacijom, ukoliko je rezultat uslovne operacije takav da data operacija nije trebalo da se izvr i. Ova kompenzaciona operacija je nezavisna po podacima od ostalih operacija, pa se mo e izvr iti u proizvoljnom trenutku. Osim ovakve softverske (statiĹke) spekulativnosti, postoji i harversko re enje istog problema - dinamiĹka spekulativnost.

Zbog diskusije navedene u prethodnom odeljku i ovoga to je upravo reĹeno, dalje eese smatrati da je dozvoljeno proizvoljno duboko spekulativno izvr avanje. TakoĹ, kompenzacione operacije za prvo vreme se neeeuzimati u razmatranje. Ipak, odreĹni mehanizam u konaĹnom algoritmu rasporeĹvanja kojim bi se kontrolisala spekulativnost mora da postoji. Naime, konaĹan algoritam treba da ukljuĹi rasporeĹvanje uz ograniĹenja na resursima. Zbog toga su potrebne heuristike kod izbora operacija pri rasporeĹvanju. Kada je veaako, u te heuristike se mo e ugraditi i kriterijum koji bi favorizovao operacije koje nisu spekulativne, ili imaju manji stepen spekulativnosti, kao to je opisano u [24].

Kao rezultat navedenog, u daljem razmatranju eegraf zavisnosti po podacima petlje (i cikliĹki i necikliĹki), taĹnije svaka njegoa komponenta jake povezanosti biti razmatrana tako da se sastoji samo od neuslovnih operacija, u koje eventualno spolja dolaze grane kontrolnih zavisnosti. Efekti paralelizacije eese najpre razmatrati samo na datoj komponenti sa neuslovnim operacijama. Kada se to razmatranje zavr i, posmatraeese efekat koji unose same uslovne operacije: ukoliko se uslovna operacija u konaĹnom rasporedu izvr ava pre kontrolno zavisnih operacija, spekulativnog izvr avanja nema; u suprotnom, potrebno je spekulativno izvr avanje.

4.2.3. *Efekat kontrolnih zavisnosti i promenljivih iteracionih distanci*

Razmotrimo sada efekat koji u graf petlje unose kontrolne zavisnosti. Kao to je u prethodnom odeljku reĹeno, posmatraeese samo jedna komponenta jake povezanosti u koju spolja ulaze grane kontrolnih zavisnosti. Uslovne operacije od kojih ove zavisnosti polaze predstavljaeeno simboliĹki kao usamljene operacije. Posmatrajmo sledeaeprimer petlje sa uslovnim grananjem:

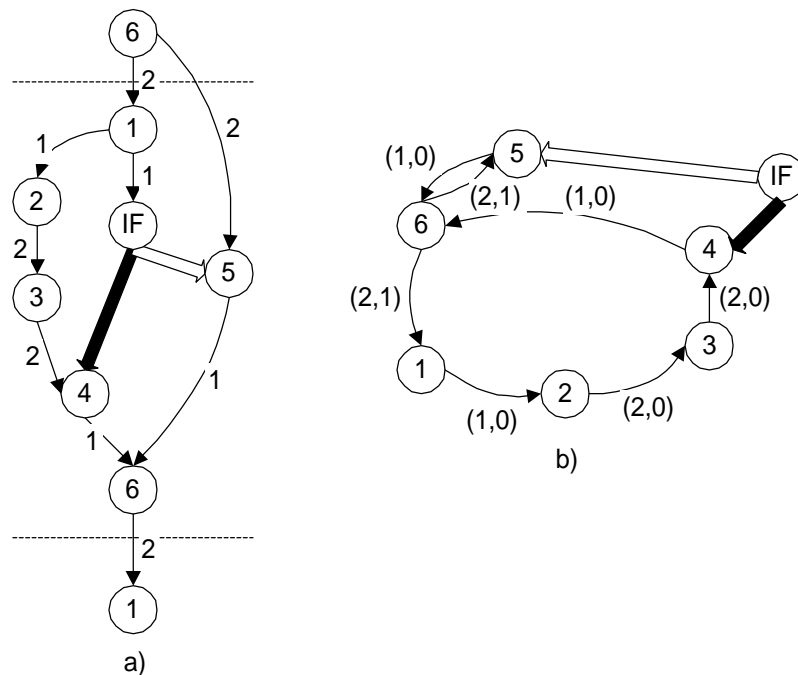
```

DO i=2,N
O1:    D[i]=A[i-1]+X
O2:    E[i]=D[i]*Y
O3:    F=E[i]*Z
IF:    IF (D[i]>0) THEN
O4:        G[i]=F[i]+5
        ELSE
O5:        J[i]=A[i]-2
        ENDIF
O6:    A[i]=G[i]+J[i]
END

```

Primer 3: Izvorni kod jedne petlje sa uslovnim grananjem

Pretpostavimo da operacija sabiranja traje 1, operacija množenja 2, a uslovna operacija 1 ciklus takta. Aciklički graf zavisnosti po podacima i kontrolnih zavisnosti jedne iteracije, zajedno sa delovima susednih iteracija prikazan je na Slici 15a. Kada se posmatra samo izvorni kod, vidi se da bez ikakve paralelizacije trajanje jedne iteracije iznosi 8 ciklusa takta za obe staze izvršavanja. Ako se posmatra aciklički graf jedne iteracije u kome se zanemaruju kontrolne zavisnosti (spekulativno izvršavanje operacije 5), onda trajanje jedne iteracije postaje 8 za jednu, a 5 ciklusa takta za drugu stazu izvršavanja.



Slika 15: Grafovi zavisnosti za petlju iz Primera 3. (a) Aciklički graf jedne iteracije (prikazane su samo dužine grana). (b) Ciklički graf komponente jake povezanosti u koju ulaze kontrolne zavisnosti.

Posmatranjem dejstva kontrolnih zavisnosti na ciklički graf zavisnosti date petlje može se uočiti da kontrolne zavisnosti uzrokuju da se u jednom ishodu uslova neka od operacija izvršava, odnosno postoji u grafu, a u drugom ne postoji, pa time i ne učestvuje u grafu sa zavisnostima koje u njega ulaze ili iz njega izlaze. Za dati primer, u slučaju da je ishod uslovne operacije IF jednak T , u grafu postoji operacija 4, a ne postoji operacija 5, pa se graf u tom slučaju sastoji samo iz jednog zatvorenog puta 1-2-3-4-6. Ograničenje paralelizma za ovaj zatvoreni put je, prema stavovima iz 3.1.1., 8 ciklusa (jer je zbir iteracionih distanci po ovom putu 1, a zbir dužina 8). U slučaju da je ishod F , u grafu ne postoji operacija 4, pa se graf sastoji samo od zatvorenog puta 5-6-1-2-3-4-6. U ovom slučaju je dužina 3, a ukupna distanca 1, i operacija 1, 2 i 3 koje svaka za sebe čine komponente jake povezanosti i zato ne utiču na ograničenje trajanja iteracije [5]. Ovo ukazuje da se za slučaj ishoda F može postići izvršavanje dužine 3 ciklusa po iteraciji! Ovo razmatranje nam dozvoljava da formiramo sledeći zaključak:

Zaključak 2: (O "nestajanju" operacija pod dejstvom kontrolnih zavisnosti) Jedan od efekata kontrolnih zavisnosti na kod je taj da neke operacije u jednom ishodu uslova uopšte ne postoje u grafu, zajedno sa zavisnostima koje do njih dolaze ili od njih polaze. Ovo je prvi potencijalni izvor paralelizma.

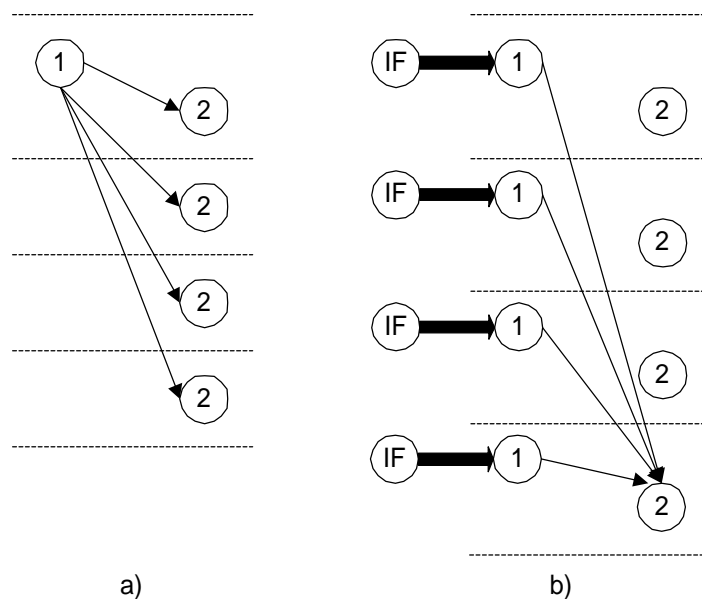
Posmatraćemo sada drugi efekat kontrolnih zavisnosti, a to je povećanje iteracionih distanci opisano u 4.1.4. Posmatrajmo opet isti primer:

```

      LOOP
      ...
      IF (Test) THEN
1:      X=...
      ENDIF
2:      ...=f(X)
      ...
      ENDLOOP

```

Na Slici 16a je prikazan aciklički graf zavisnosti četiri susedne iteracije za slučaj da je u prvoj iteraciji ishod uslova bio T , a u tri naredne iteracije bio F . Vidi se da od operacije 1 koja je izvršena polazi "lepeza" pravih zavisnosti prema instancama operacije 2 u sledećim iteracijama. U svakoj narednoj iteraciji u kojoj je ishod uslova F , povećava se iteraciona distanca od operacije 1 prema operaciji 2. To omogućava da se instance operacije 2 pomeraju naviše u ranije iteracije koliko to ostale zavisnosti dozvoljavaju (pored ostalih, izlazne između samih ovih instanci operacije 2), čime se povećavaju iteracione distance grana koje izlaze iz operacije 2 [15]. Time se potencijalno dobija na paralelizmu jer se lanci zavisnosti koji izlaze iz operacije 2 "raste u" na višim iteracijama i time vreme izvršavanja narednih iteracija smanjuje. Na Slici 16b je ista pojava prikazana drugačije, tako da u jednu instancu operacije 2 ulazi "lepeza" ovoga puta potencijalnih zavisnosti, pri čemu u konkretnom izvršavanju postoji samo ona zavisnost koja polazi od poslednje izvršene operacije 1 pre date instance operacije 2.



Slika 16: Graf zavisnosti nekoliko susednih iteracija za slučaj promenljive iteracione distance. (a) Za slučaj kada je ishod uslova bio T u jednoj iteraciji, a zatim F u tri naredne iteracije. (b) Prikaz promenljive zavisnosti koja ulazi u operaciju 2 za različite ishode uslovne operacije. Izlazne i antizavisnosti nisu prikazane.

Može se pomisliti da ovakva pojava promenljivih iteracionih distanci samo unosi teškoće u proces optimizacije i da je potrebno eliminisati preimenovanjem, jer je ionako posledica ponovne upotrebe iste skalarne promenljive u narednim iteracijama. Takođe je da se ova pojava može eliminisati uvođenjem vektorske promenljive umesto skalarne, što se vidi iz sledećeg koda dobijenog od koda petlje iz prethodnog primera:

```

      LOOP
      ...
      IF (Test) THEN
1:      X[i]=...
2':     ...=f(X[i])
      ELSE
1':     X[i]=X[i-1]
2:      ...=f(X[i-1])
      ENDIF
      ...
      ENDLOOP

```

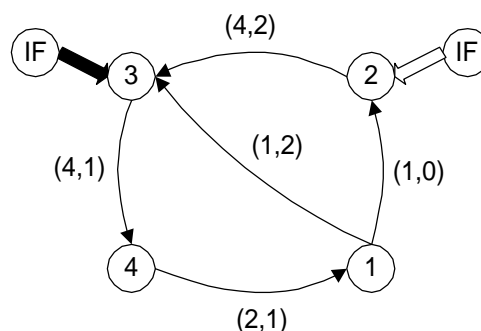
Treba primetiti da su operacije 1' i 2 sada nezavisne, pa se mogu izvršavati paralelno. Međutim, ovakvo rešenje uvodi jedan lanac zavisnosti po lev od poslednje izvršene instance operacije 1, preko niza instanci operacije 1' u narednim iteracijama, do neke instance operacije 2. Taj lanac ne dozvoljava proizvoljno pomeranje operacije 2 u prethodne iteracije, pa se efekat koji je malopre objašnjen gubi, a time gubi i potencijalni paralelizam. Sledi zaključak:

Zaključak 3: (O povećanju iteracionih distanci pod dejstvom kontrolnih zavisnosti) Jedan od efekata kontrolnih zavisnosti na kod je taj da se pojedine iteracione distance povećavaju u određenim kombinacijama ishoda uslova. Time se omogućava da se neke operacije premeštaju u ranije iteracije, čime se iteracione distance zavisnosti koje od njih polaze povećavaju. Ovo je drugi potencijalni izvor paralelizma.

4.2.4. Interferencija uslova

Iz pojava nestajanja operacija iz grafa i promenljive iteracione distance opisane u prethodnom odeljku naslućuje se da postoji efekat na kod koji proizvode uslovne operacije iz više različitih iteracija. Takođe, za slučaj postojanja više uslovnih operacija u telu petlje, može se pretpostaviti da postoji interferencija ovih različitih uslova. Prva pomisao navodi na to da interferencija koja daje potencijalni paralelizam postoji između uslova iz susednih iteracija. Ovde će biti pokazano da u opštem slučaju ne mora da bude tako.

Posmatrajmo ciklički graf zavisnosti prikazan na Slici 17. Petlja poseduje jednu uslovnu operaciju označenu sa IF, od koje kontrolno zavise operacije 2 i 3. Kako je iteraciona distanca grane 2-3 jednaka 2, zajednički efekat na izgled grafa imaju zapravo uslovi IF iz dve iteracije na rastojanju 2. Prema tome, izgled grafa, a njime i uslovljena pomeranja operacija pri optimizaciji, zavisi od interferencije uslova IF iz iteracije i i iteracije $i-2$.



Slika 17: Ciklički graf zavisnosti jedne petlje

Označimo ishod T uslova IF iz iteracije $i+k$ sa $p[k]$, gde je i neka posmatrana fiksna iteracija, a ishod F istog uslova sa $\bar{p}[i+k]$. Na slici na liniji 16 označavati i operaciju op iz iteracije $i+k$: $op[k]$. Postoje četiri moguća slučaja:

- a) $\bar{p}[0]\bar{p}[-2]$; u ovom sluĉaju graf se pretvara u niz zavisnih operacija 4-1-2, jer operacija 3[0] ne postoji;
- b) $\bar{p}[0]p[-2]$; u ovom sluĉaju graf se pretvara u niz zavisnih operacija 4-1, jer operacije 3[0] i 2[-2] ne postoje;
- c) $p[0]\bar{p}[-2]$; u ovom sluĉaju graf je isti kao to je na poĉetku, jer i 3[0] i 2[-2] postoje;
- d) $p[0]p[-2]$; u ovom sluĉaju graf se pretvara u zatvoreni put 4-1-3-4, jer operacija 2[-2] ne postoji.

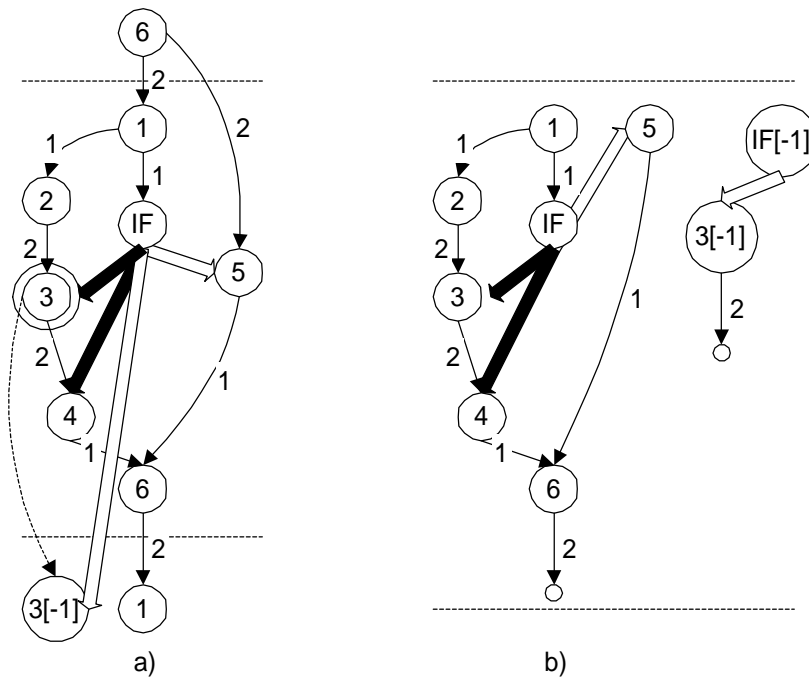
Iz pokazanog primera se jasno vidi da nema nikakvog uticaja na izgled grafa to to u petlji postoji samo jedan uslov, jer bi u su tni isti efekat proizvela dva uslova IF1 i IF2, od kojih IF1 uslovljava operaciju 3, a uslov IF2 operaciju 2. Nikakve bitne interferencije ne bi postojale izmeĊ ovakva dva uslova u istoj iteraciji. Dalje, vidi se da interferencija izmeĊ uslova iz susednih iteracija ne mora da bude bitna, veće bitna interferencija izmeĊ uslova iz iteracija udaljenih onoliko iteracija, koliko to diktiraju iteracione distance zavisnosti po podacima izmeĊ uslovljenih operacija. Zbog toga bi re enje problema zasnovano na razmotavanju petlji ili, op tije, ograniĉeno iskljuĉivo na tretiranje uslova iz susednih iteracija bilo inferiorno, bez obzira koliko iznosi stepen razmotavanja odnosno opseg tretiranja susednih iteracija. Sledi zakljuĉak:

Zakljuĉak 4: *(O interferenciji uslova) Bitne interferencije su izmeĊu onih uslova koji su udaljeni onoliko iteracija koliko to diktiraju iteracione distance zavisnosti izmeĊu uslovljenih operacija. Pri tome nije bitno da li interferiraju instance razliĉitih ili istih uslovnih operacija iz razliĉitih iteracija. Ovo je treći potencijalni izvor paralelizma.*

4.2.5. Pomeranje operacija

Razmotrićmo sada kako se zaista mo e dobiti na paralelizmu kori ećem svih do sada uoĉenih izvora. Drugim reĉima, kako se zapravo vr e pomeranja operacija u druge iteracije kao posledica razliĉitih ishoda uslova, u cilju postizanja softverske protoĉnosti.

Vratimo se ponovo na petlju iz Primera 3 Ĺiji je graf jedne iteracije zajedno sa susednim iteracijama (graf razmotane petlje [15]) prikazan na Slici 15a. Kao to je ranije uoĉeno, u sluĉaju da je ishod uslova T , ne postoji operacija 4, pa operacije 1, 2 i 3 postaju deo lanca zavisnosti koji je slobodan na dnu [15]. Sa druge strane, na Slici 15 se vidi da u tom sluĉaju preostaje lanac (zapravo zatvoreni put u cikliĉkom grafu) zavisnosti operacija 6-5-6 Ĺija du ina ne dozvoljava kraće izvr avanje date iteracije od 3 ciklusa. Prema tome, potrebno je i lanac 1-2-3 prilagoditi ovom minimalnom izvr avanju od 3 ciklusa, to se posti e pomeranjem operacije 3 u narednu iteraciju (Slika 18a). Kako i ova naredna iteracija traje najmanje 3 ciklusa, ovo pomeranje je dovoljno, jer se operacija 3 sigurno zavr ava u toj narednoj iteraciji. Izvr avanje operacija 1 i 2 traje 3 ciklusa, to je upravo jednako trajanju izvr avanja operacija 5 i 6, pa operacije 1 i 2 nije potrebno preme tati.

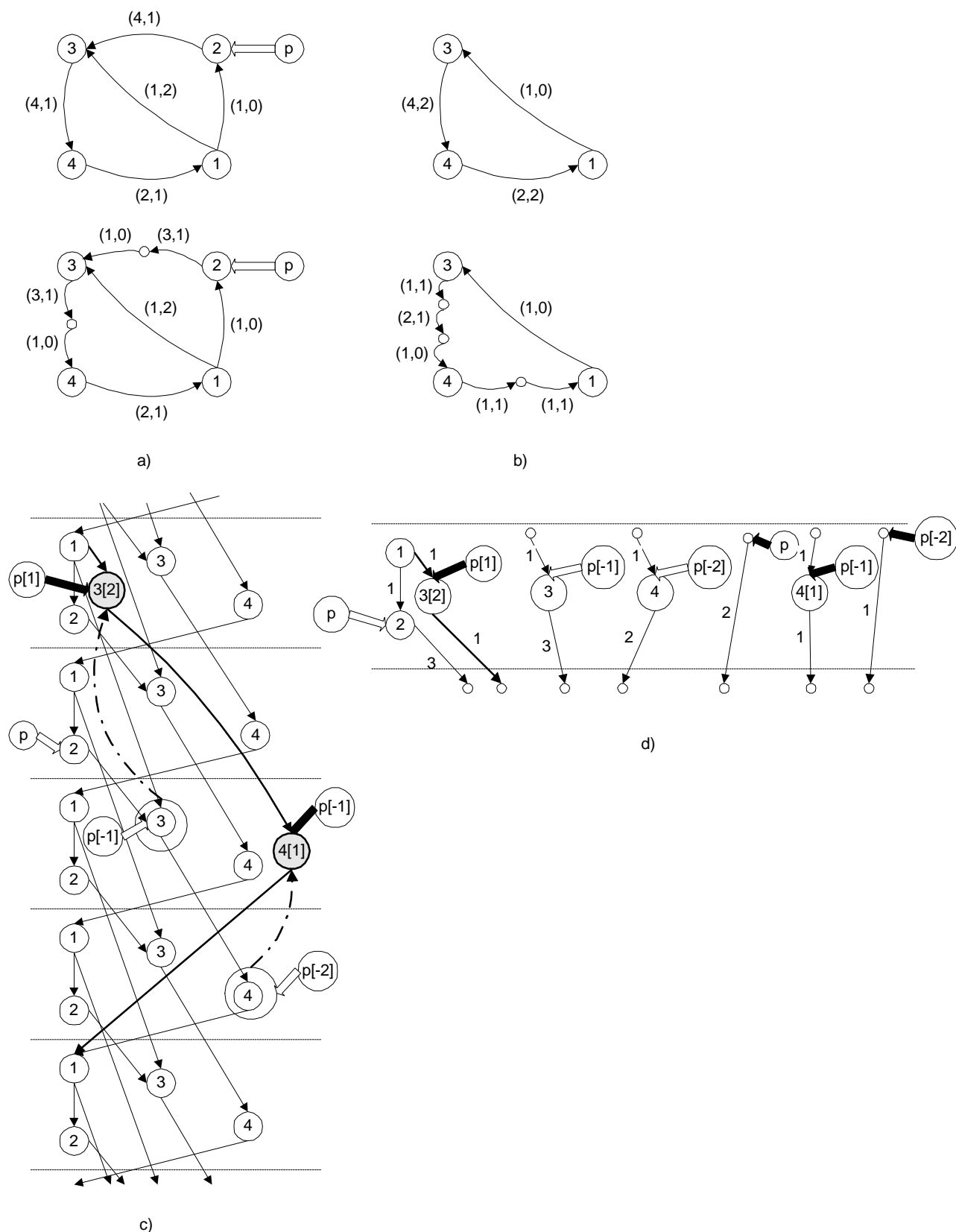


Slika 18: Jednostavan primer pomeranja operacija u cilju optimizacije. Operacija 3 se, u slučaju ishoda T uslova, pomera u narednu iteraciju. (a) Graf razmorane petlje sa naznačenim pomeranjem operacije 3. (b) Aciklički graf jedne iteracije optimizovane petlje. Naznačene su samo dužine grana.

Operacija 3 se premeće u narednu iteraciju pod uslovom da je ishod uslovne operacije T . Zbog toga se operacija 3 date iteracije izvršava u toj iteraciji samo pod uslovom F , čime i operacija 3 postaje kontrolno zavisna. U narednoj iteraciji postoji operacija 3 iz posmatrane (prethodne) iteracije pod uslovom T te prethodne iteracije. Iz razloga periodičnosti koja je objašnjena u Zaključku 1, u svakoj iteraciji se može naći operacija 3 iz prethodne iteracije (sa oznakom 3[-1]), pod uslovom da je ishod uslovne operacije iz prethodne iteracije bio T . Na taj način se dobija optimizovana iteracija čiji je aciklički graf dat na Slici 18b. Tako se i dobija izvršavanje od 3, odnosno 8 ciklusa.

Mogu se primetiti dve stvari. Prvo, u svakoj iteraciji mogu da postoje dve instance iste operacije (operacija 3 u primeru), koje su kontrolno zavisne od različitih instanci istog ili različitih uslova. To znači da se u jednoj iteraciji mogu pojaviti sve, samo neke, ili nijedna instanca date operacije, u zavisnosti od kombinacije ishoda uslova. Drugo, kao što je i ranije veoma objašnjeno, postoji mogućnost da neka operacija kontrolno zavisi od uslovne operacije iz neke ranije iteracije, što uzrokuje potrebu da se rezultati uslovnih operacija pamte u skalarnim (ukoliko je ivotni vek ograničen na sledeću iteraciju) ili vektorskim promenljivim (ukoliko je ivotni vek višestruko iteracija).

Upravo iznesene zaključke potvrđuje i još jedan primer. Na Slici 19a prikazan je ciklički graf jedne petlje. Ako je ishod uslova F jednak F , operacija 2 postoji, pa graf zavisnosti ima upravo oblik prikazan na Slici 19a. Da bi izvršavanje ovakve strukture operacija bilo optimalno, operacije treba da budu raspoređene po iteracijama kao što je to prikazano na drugom delu Slike 19a [15]. Izgled grafa razmotrane petlje prikazan je na Slici 19c. Za slučaj da je ishod T , operacija 2 ne postoji, pa je raspored operacija po iteracijama neoptimalan za taj slučaj, prema rezultatima iz [14]. Da bi izvršavanje ovakve strukture bilo optimalno, potrebno je ukupnu dužinu zatvorenog puta podeliti na jednake delove, kao što je to pokazano na Slici 19b. Prema tome, u odnosu na osnovni izgled grafa na Slici 19a, ovaj graf se razlikuje po tome što je operacija 3 bliža operaciji 1 (nalazi se u istoj iteraciji), što se može postići pomeranjem operacije 3 dve iteracije ranije. Slično je i za operaciju 4, koju treba pomeriti jednu iteraciju ranije.



Slika 19: Jo jedan primer pomeranja operacija u cilju optimizacije. (a) Ciklički graf zavisnosti petlje sa prikazanom raspodelom izvršavanja operacija po iteracijama. (b) Ciklički graf za slučaj ispunjenog uslova p i nađin raspodele izvršavanja operacija po iteracijama. (c) Graf razmotane petlje sa naznačenim pomeranjem. (d) Aciklički graf novodobijene iteracije.

Graf razmotane petlje zajedno sa naznačenim pomeranjima dat je na Slici 19c. Na istoj slici su navedeni i uslovi koji kontroli u pomerene operacije i njihove originale, prema istoj logici kao u prethodnom primeru. Pomeranje je naznačeno samo za jednu iteraciju, dok se primenom principa periodičnosti dobija konačan izgled jedne iteracije kao na Slici 19d. Posmatranjem Slike 19d može se dobiti sledeća tabela koja daje dužinu izvršavanja jedne iteracije, u zavisnosti od kombinacije uslova koji učestvuju u kontrolnim zavisnostima:

$p[-2]$	$p[-1]$	$p[0]$	$p[+1]$	Dužina
F	F	F	F	4
F	F	F	T	4
F	F	T	F	4
F	F	T	T	4
F	T	F	F	4
F	T	F	T	4
F	T	T	F	3
F	T	T	T	3
T	F	F	F	4
T	F	F	T	4
T	F	T	F	4
T	F	T	T	4
T	T	F	F	4
T	T	F	T	4
T	T	T	F	2
T	T	T	T	2

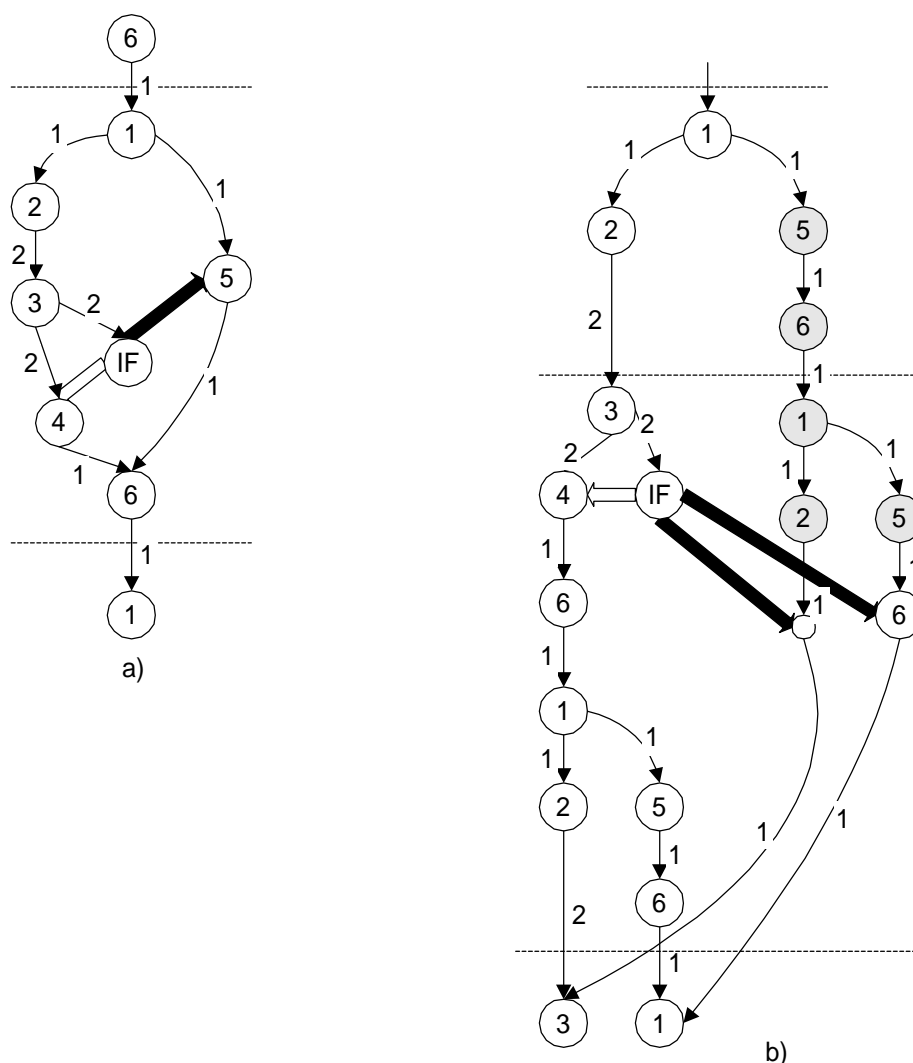
Prema tome, izvršavanje iteracije ove petlje traje između 2 i 4 ciklusa, što predstavlja promenljiv interval iniciranja. Sledi zaključak:

Zaključak 5: (O višestrukim instancama operacija) Da bi se dobilo na paralelizmu, potrebno je da u jednoj iteraciji postoji više instanci iste operacije iz različitih iteracija, s tim da su one kontrolno zavisne od različitih instanci istog ili različitih uslovnih operacija, pa zato mogu da se izvršavaju sve, samo neke, ili nijedna od njih u zavisnosti od kombinacije ishoda.

4.2.6. Efekat kašnjenja uslova zbog pomeranja operacija

Razmotrimo sada efekat koji proizvodi pomeranje operacija na pomeranje uslovnih operacija koje su do sada posmatrane relativno nezavisno. Pomeranje neuslovnih operacija može da ima za posledicu i pomeranje uslovnih operacija koje su vezane zavisnostima po podacima. Pri tome, pomeranje uslovnih operacija u ranije iteracije dovodi do ranijeg razrešavanja uslova, pa se time spekulativno izvršavanje umanjuje. Problem može da predstavlja pomeranje uslovnih operacija prema kasnijim trenucima izvršavanja.

Posmatrajmo primer petlje čiji je aciklički graf zavisnosti prikazan na Slici 20a. U slučaju ishoda F , operacija 5 ne postoji, ali postoji lanac zavisnosti 1-2-3-4-6 dužine 7, što predstavlja minimalnu dužinu izvršavanja iteracije u tom slučaju. U slučaju ishoda T , u grafu postoji lanac zavisnosti 1-5-6 dužine 3, što predstavlja minimum trajanja iteracije za ovaj slučaj. U tom slučaju ne postoji operacija 4, pa operacija 3 postaje slobodna na dnu. Kako dužina lanca zavisnosti 1-2 iznosi 3, koliko predstavlja optimalna dužina iteracije u ovom slučaju, operaciju 3 treba premestiti u narednu iteraciju kao što je prikazano na Slici 20b.



Slika 20: Efekat pomeranja uslovnih operacija prema kasnijim trenutcima izvršavanja. (a) Graf zavisnosti jedne iteracije petlje. (b) Izgled dve susedne iteracije posle pomeranja operacije 3 u narednu iteraciju. Osenčene su operacije koje se izvršavaju spekulativno. Od uslovne operacije IF prema nizovima operacija iz leve grane izvršavanja (4-6-1-2,5-6) polaze takođe kontrolne zavisnosti koje nisu označene radi preglednosti. Naznačene su samo dužine grana.

Međutim, pomeranje operacije 3 prema kasnije uzrokuje i pomeranje uslovne operacije u narednu iteraciju, jer ona zavisi po podacima od operacije 3. Zbog toga se u posmatranoj (prvoj) iteraciji ne može znati da li njeno izvršavanje treba da traje 3 ili 7 ciklusa. Da se ne bi nepotrebno izvršavala operacija 3 u posmatranoj iteraciji, kako bi se saznao ishod uslova, i time narušavao potencijalni optimum od 3 ciklusa, potrebno je posmatranu iteraciju ograničiti na 3 ciklusa u svakom slučaju, i operaciju 3 premestiti u narednu iteraciju (Slika 20b). U ovoj narednoj iteraciji postoje dva toka izvršavanja: jedan (na desnoj strani slike) pretpostavlja da je u prethodnoj iteraciji ishod bio *T*, pa se započinje odmah nova iteracija izvršavanjem operacija 1, 2 i 5, i drugi (na slici levo), koji pretpostavlja da je ishod uslova bio *F*, pa nastavlja sa izvršavanjem operacije 4 i 6 kada se uslov razreši. Za ovaj drugi tok je potrebno izvršiti ponovo operacije koje započinju novu iteraciju (1, 2 i 5) kada se završi izvršavanje operacija 4 i 6, jer je započet tok pod pretpostavkom *T* pogrešan (poništavanje spekulativnih operacija). Treba primetiti da ukupno trajanje dve posmatrane iteracije iznosi 3+7 ciklusa, što je opet optimalno, s obzirom da je minimalno izvršavanje prve iteracije 7 za slučaj *F*, a druge još uvek nepoznato, ali ne manje od 3.

Prema tome, može se verovati da se efekat pomeranja uslova na kasnije može kompenzovati na opisani način praktično bez gubitka u vremenu izvršavanja. Princip se u opštem

služaju sastoji u sledećem. Primenjuje se najpre postupak optimizacije pod pretpostavkom da su izvori uslova sasvim nezavisni, kao što je to rađeno ranije. Uslovne operacije se tretiraju isto kao i ostale¹⁴. Za određenu iteraciju dobijaju se rasporedi operacija uslovljeni različitim kombinacijama uslova. Zatim se utvrđuje tačno vreme razrešavanja uslova. Ukoliko u ovako dobijenom rasporedu postoje dva lanca zavisnosti operacija različitih dužina, od kojih je jedan pod uslovom p , a drugi pod suprotnim uslovom \bar{p} , a uslovna operacija koja razrešava uslov p se izvršava kasnije, potrebno je duži lanac svesti na dužinu kraćeg lanca, premeštanjem preostalih operacija u narednu iteraciju. Ovaj princip bio u radu korišćen bez dokaza da se navedeni postupak u svakom slučaju okončava i da zadržava ukupno vreme izvršavanja. Ovo tvrdjenje može se ponoviti u obliku zaključka:

Zaključak 6: *(O eliminisanju spekulativnih izvršavanja različitih dužina) Ukoliko u rasporedu operacija dobijenom nezavisnim posmatranjem uslova postoje dva lanca zavisnosti operacija različitih dužina, od kojih je jedan pod uslovom p , a drugi pod suprotnim uslovom \bar{p} , a uslovna operacija koja razrešava uslov p se izvršava kasnije (razlika u izvršavanju je spekulativna), potrebno je duži lanac svesti na dužinu kraćeg lanca, premeštanjem preostalih operacija u narednu iteraciju.*

4.2.7. Efekti transformacija na rezultujuće kod

Ovde će biti formalno definisani efekti koje pomeranja operacija između iteracija proizvode u rezultujućem kodu. Instanca operacije op iz iteracije i biće označavana sa $op[i]$. Konjunkcija (logičko I) nekoliko ishoda različitih ili istih uslova iz različitih ili istih iteracija biće ovde nazivan *predikat* i označavan sa $p[i]$, gde se i odnosi na neku posebnu iteraciju. Na primer, ako se posmatraju dva uslova a i b i ako se kombinacija ishoda uslova $(a[i]=T \wedge b[i-2]=F)$ obeleži sa $p[i]$, onda oznaka $p[i+k]$ predstavlja kombinaciju $(a[i+k]=T \wedge b[i+k-2]=F)$, a oznaka $\bar{p}[i+k]$ predstavlja kombinaciju $(a[i+k]=F \vee b[i+k-2]=T)$. Pomeranje za $d>0$ iteracija neke operacije predstavlja pomeranje u naredne iteracije (prema kasnije), a za $d<0$ pomeranje u prethodne iteracije (prema ranije).

Na Slici 21 je prikazan slučaj pomeranja neke operacije $op[i+j]$ za d iteracija, pod uslovom $p[i]$. Vidi se da se u iteraciji $i+j+d$ pojavljuje operacija op iz d iteracija ranije, to, zbog razloga periodičnosti, znači da se u svakoj iteraciji i javlja operacija $op[i-d]$ pod uslovom $p[i-d-j]$. To zapravo znači da će optimizovanom kodu umesto npr. operacije

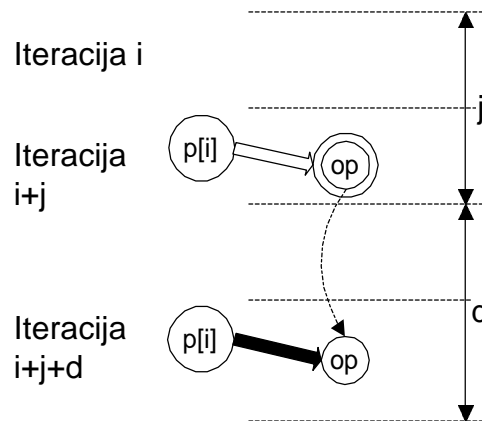
$$B[i+1]=A[i-2]+C[i]$$

stajati operacija:

$$B[i+1-d]=A[i-2-d]+C[i-d]$$

pod odgovarajućim uslovom. Originalna operacija $op[i]$ će ostati na svom mestu ukoliko ovaj uslov nije ispunjen.

¹⁴U nekom smislu, razdvajaju se uslovne operacije od uslova koji generišu kontrolne zavisnosti.



Slika 21: Efekat uslovnog pomeranja operacije $op[i+j]$ za d iteracija pod uslovom $p[i]$.

Evo sada formalno iskazanih efekata pomeranja operacije op za d iteracija:

1. Bezuslovno pomeranje operacije:

$move(op, d)$

Ako operaciju op treba pomeriti bezuslovno za d iteracija, onda se u rezultujućem kodu, iz razloga periodičnosti, umesto operacije $op[i]$ u i -toj iteraciji pojavljuje operacija $op[i-d]$.

2. Pomeranje operacije op iz iteracije $i+j$ za d iteracija pod uslovom $p[i]$:

$move(p[i], op[i+j], d)$

U i -toj iteraciji se pojavljuje operacija $op[i-d]$ pod uslovom $p[i-d-j]$. Takođe, u i -toj iteraciji se pojavljuje operacija $op[i]$ samo ako je zadovoljen uslov $\bar{p}[i-j]$. Takođe, originalna operacija $op[i]$ ostaje na svom mestu ako su ispunjeni svi uslovi $\bar{p}[i-j]$ (njihova konjunkcija) iz svih pomeranja prikazanog tipa. Treba primetiti da konjunkcija svih ovakvih uslova može da bude slovena disjunktivna normalna forma (DNF), što otežava postupak generisanja koda na ovaj način.

4.2.8. Zaključak analize

U zaključku ove glave sumirajmo rezultate analize problema. Potencijalni izvori paralelizma su nestajanje operacije iz grafa zavisnosti pod nekim uslovom i produžavanje iteracione distance između operacija. Oba ova izvora dozvoljavaju pomeranje operacija u druge iteracije. Ova pomeranja uzrokuju da se u jednoj iteraciji može javiti više instanci iste operacije iz različitih iteracija, uslovljenih različitim predikatima. Pri tome važnu ulogu ima interferencija istih ili različitih uslova iz različitih iteracija na rastojanju koje diktiraju zavisnosti po podacima. Najzad, pomeranje operacija može uzrokovati da se u jednoj iteraciji pojave dva lanca zavisnosti po podacima različitih dužina, ali uslovljenih suprotnim predikatima koji se ne mogu razrešiti na vreme. U tom slučaju ove lance treba izjednačiti premeštanjem operacija iz drugog lanca u kasnije iteracije.

Efekti pomeranja operacija mogu da budu veoma složeni. Prvo, izvršavanje operacije može da bude uslovljeno veoma složenim disjunktivnim izrazom, što otežava softversku ili hardversku kontrolu ovog izvršavanja. Drugo, problem leži u operacijama koje traju više ciklusa takta. Ove operacije mogu da se "protežu" na više susednih iteracija, pri čemu u se svakoj od tih iteracija izvršava jedan deo operacije. Ovi delovi povećavaju interferenciju uslova koji definišu trajanje jedne iteracije. Zbog toga optimalno ili približno optimalno trajanje jedne iteracije nije lako dobiti samo posmatranjem cikličkog grafa zavisnosti petlje.

Ovi zaključci predstavljaju smernice za pronalaženje rešenja problema. Prvo, uspešno rešenje mora da iskoristi sve pronađene izvore paralelizma. Drugo, ovo rešenje mora da se uspešno "izbori" sa potencijalno složenom interferencijom uslova iz različitih iteracija koje proizvode sve opisane efekte. Najzad, pomeranja operacija u cilju softverske protočnosti moraju da

biti takva da se lako može kontrolisati i da inače izvršavanja iteracije u svakom pojedinačnom slučaju kombinacije ishoda uslova.

5. Ideja predloženog rešenja

U prethodnoj glavi detaljno je analiziran problem optimizacije koda petlji sa uslovnim grananjima i postavljene su smernice za rešenje tog problema. U ovoj glavi bice predloženo jedno rešenje koje zadovoljava uslove otkrivene analizom. Rešenje predstavlja zapravo jedan potpuno novi model petlji sa uslovnim grananjima. Ovde je samo neformalno opisan model u cilju razumevanja njegove suštine. Formalno će model biti definisan u narednoj glavi. Model je nazvan *Modelom predikatske softverske protočnosti* (engl. *Predicated Software Pipelining*, predlog termina) ili skraćeno PSP model.

5.1. Model predikatske softverske protočnosti (PSP)

Kao što je u prethodnoj glavi pokazano, efekat pomeranja operacija pod različitim uslovima je da se u konačnom kodu jedne iteracije nalaze razne instance operacija iz različitih operacija, uslovljenih različitim kombinacijama predikata. Ako se ovo posmatra malo drugačije, u konačan skup uslova koji definišu izvršavanje jedne iteracije ulazi niz ishoda uslovnih operacija iz različitih iteracija.

Pretpostavimo da u petlji postoje 4 uslovne operacije, označimo ih sa p , q , r i s . Neka razna pomeranja opisana u prethodnoj glavi uzrokuju da se u i -toj iteraciji pojavljuju uslovi $p[i-1]$, $p[i]$, $p[i+1]$, $p[i+2]$, $q[i-1]$, $q[i+1]$, $q[i+2]$, $r[i+1]$, $r[i+2]$, $s[i]$ i $s[i+1]$ od kojih zavisi izvršavanje nekih operacija. Ovaj skup možemo predstaviti matricom koja ima sledeći oblik¹⁵, gde je $p[i+k]$ označeno kraćo $p[k]$:

$$\begin{bmatrix} p[-1] & p[0] & p[1] & p[2] \\ q[-1] & b & q[1] & q[2] \\ b & b & r[1] & r[2] \\ b & s[0] & s[1] & b \end{bmatrix}$$

Kao što se vidi, matrica ima $m=4$ vrste i $n=4$ kolone, gde je m broj uslovnih operacija u polaznoj petlji (broj koji zavisi samo od petlje koja se optimizuje), a n broj za 1 veći od maksimalne iteracione distance uslova koji učestvuju u konačnoj petlji. Po vrstama su navedene instance istog uslova iz različitih iteracija, a u istoj koloni se nalaze različiti uslovi iz iste iteracije. Simbol b označava da odgovarajuća instance uslova ne učestvuju na kôdu. Ova prikazana matrica predstavlja jednu po kojoj će biti generisane matrice konkretnih ishoda uslova i bice nazivana *predikatskom matricom* (engl. *predicate matrix*, predlog). Ove konkretne matrice ishoda nazivaćemo *matricama stanja* (engl. *state matrix*, predlog). U svakom polju ovih matrica stanja nalaziće se jedan od tri simbola: 1 označava da je ishod odgovarajuće instance uslova iz predikatske matrice T , 0 da je ishod F , a b da odgovarajuće uslov ne učestvuje u razmatranju.

Za prikazani primer matrice, postojaće 2¹¹ različitih matrica stanja, jer je 11 polja u prikazanoj predikatskoj matrici popunjeno ne- b simbolima. Za svaku od ovih konkretnih matrica stanja postojaće odgovarajuće raspored operacija koje treba izvršiti u datoj iteraciji, to je uslovljeno time da su odgovarajuće ishodi T ili F . Neki od ovih rasporeda mogu biti isti, a neki ili svi različiti.

Ideja se sada razvija na sledeći način. U svakom konkretnom izvršavanju petlje postoje sasvim konkretni ishodi uslova svih iteracija. Izvršavanje prelazi sa iteracije na iteraciju, pri čemu se svakoj iteraciji može pridružiti matrica stanja. Posmatrajmo sledeće izvršavanje:

¹⁵Matrica ne mora biti kvadratna.

$$\begin{array}{cccc}
 1 & 2 & 3 & 4 \\
 \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}
 \end{array}$$

pri čemu u ovoj tablici prvi red označava redni broj iteracije, a elementi u koloni označavaju ishode uslova p , q , r i s u odgovarajućoj iteraciji. Za dato izvršavanje, iteracijama 1, 2, 3 i 4 odgovaraju redom sledeće matrice stanja prema datoj predikatskoj matrici:

$$\begin{bmatrix} ? & 1 & 1 & 0 \\ ? & b & 0 & 1 \\ b & b & 1 & 0 \\ b & 1 & 1 & b \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & b & 1 & 0 \\ b & b & 0 & 1 \\ b & 1 & 0 & b \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & ? \\ 0 & b & 0 & ? \\ b & b & 1 & ? \\ b & 0 & 1 & b \end{bmatrix}, \begin{bmatrix} 0 & 0 & ? & ? \\ 1 & b & ? & ? \\ b & b & ? & ? \\ b & 1 & ? & b \end{bmatrix}$$

gde znakovi "?" predstavljaju nepoznat ishod uslova za datu ograničenu sekvencu izvršavanja; ove vrednosti zavise od ishoda uslova koji prethode, odnosno slede datu sekvencu. Uočava se da data matrica ishoda "plovi" kroz matricu stanja iz iteracije u iteraciju, tako što se u svakoj iteraciji pomera za jedno mesto ulevo.

Ovo posmatranje nameće sledeće ideje: predstavimo petlju pomoćnog automata koji ima onoliko stanja, koliko postoji matrica stanja, tako da se svakom stanju automata pridruži tačno jedna matrica stanja. Prelazi iz stanja u stanje modelovanjem prelaza izvršavanja petlje iz iteracije u iteraciju. Kako se svakoj iteraciji može pridružiti jedna matrica stanja, to se zapravo svakoj iteraciji pridruži jedno stanje automata. Iz jednog stanja automata može se preći u 2^m stanja. Ova sledbenička stanja određena su na taj način da se njihove matrice dobijaju pomeranjem matrice polaznog stanja ulevo, pri čemu se krajnje leve vrednosti koje "otpadaju" odbacuju, a sa desne strane na "upravljen" mesta koja nemaju b oznaku upisuju sve varijacije (njih 2^m) ishoda 0 i 1.

Na primer, iz stanja definisanog matricom

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & b & 1 & 0 \\ b & b & 0 & 1 \\ b & 1 & 0 & b \end{bmatrix}$$

može se preći u sva stanja definisana matricama

$$\begin{bmatrix} 0 & 1 & 0 & x \\ ? & b & 0 & x \\ b & b & 1 & x \\ b & 0 & x & b \end{bmatrix}$$

kada se umesto simbola x stave simboli 0 ili 1. Znak "?" u ovom slučaju ukazuje na problem što se ne poznaje vrednost $q[0]$ u polaznom stanju, pa se u narednom stanju ne poznaje vrednost $q[-1]$. Isti ovaj problem javio bi se i u postupku pomeranja operacija opisanom u narednom poglavlju, pa će se ovakva pojava eliminisati. Pored toga, pojava da u izgledu iteracije učestvuju dve instance istog uslova na nekom iteracionom rastojanju većem od 1, a da neka od instanci istog uslova između njih ne učestvuje nije realna, jer se uslovno izvršavanje neke operacije uvek "proteće" preko svih iteracija koje se nalaze između krajnjih iteracija obuhvaćenih nekim pomeranjem. Zbog svega ovoga, mi ćemo ograničiti posmatranje tako što će predikatske matrice biti konstruisane samo tako što u

jednoj vrsti ne postoje " b -rupe", to se formalno može iskazati na sledeći način. Ne dozvoljava se da u nekoj vrsti predikatske matrice postoje dve različite ne- b vrednosti, između kojih postoji neka b vrednost.

Takođe, može se primetiti da opisani model zapravo predstavlja beskonačno izvršavanje petlji, to je zapravo podrška principu periodičnosti. Problem ulaska u petlju i izlaska iz petlje koji zapravo uključuje problem pretpetlje i postpetlje neće se ovde razmatrati. Praktično se razmatranje ograničava na jezgro optimizovane petlje, jer je ono kritično za veliki broj izvršenih iteracija. Opisani model naziva se *PSP modelom*.

Prema tome, predikatska matrica za polazni primer treba da izgleda ovako:

$$\begin{bmatrix} p[-1] & p[0] & p[1] & p[2] \\ q[-1] & q[0] & q[1] & q[2] \\ b & b & r[1] & r[2] \\ b & s[0] & s[1] & b \end{bmatrix}$$

to znači da konačni automat koji predstavlja model petlje iz stanja¹⁶

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ b & b & 0 & 1 \\ b & 1 & 0 & b \end{bmatrix}$$

prelazi u stanja koja su određena svim matricama koje umesto x imaju 0 ili 1:

$$\begin{bmatrix} 0 & 1 & 0 & x \\ 1 & 1 & 0 & x \\ b & b & 1 & x \\ b & 0 & x & b \end{bmatrix}$$

Jasno je sada kako se pronalaze stanja u koja automat prelazi iz nekog stanja S . Matrice tih stanja se dobijaju tako što se iz matrice stanja S svi elementi koji sa leve strane imaju b susede (zvaćemo ih ne- b ivični elementi sa leve strane) odbace, matrica stanja S pomeri ulevo za jedno mesto, i na mesta ivičnih ne- b elemenata sa desne strane upiše u sve varijacije elemenata 0 i 1. Slično, stanja iz kojih se u S dolazi određuju se na sledeći način: svi ne- b ivični elementi sa desne strane se odbace, matrica stanja S se pomeri udesno za jedno mesto, i na mesta ivičnih ne- b elemenata sa leve strane upiše u sve varijacije elemenata 0 i 1. Prema tome, u svako stanje S se dolazi iz 2^m stanja, i u isto toliko stanja se iz stanja S dolazi. Pri tome polazno i dolazno stanje ne moraju biti različita; iz stanja koje ima sve ne- b elemente jednake 0 se može doći i do istog stanja (isto važi i za sve 1).

Evo jednog jednostavnog primera: neka je predikatska matrica data sa

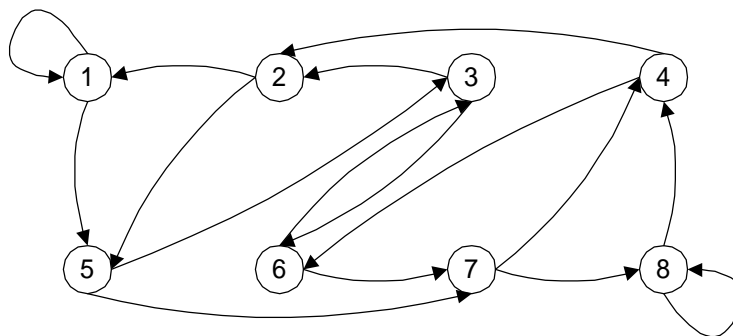
$$\begin{bmatrix} p[-1] & p[0] & p[1] \end{bmatrix}$$

to znači da petlja poseduje samo jednu uslovnu operaciju i da se razmatraju tri susedne iteracije, tekuća, jedna prethodna i jedna naredna. PSP model dat ovom predikatskom matricom ima 8 stanja (sve varijacije dužine 3 elemenata 0 i 1). Iz svakog stanja se prelazi u dva stanja koja se dobijaju tako što se matrica pomeri ulevo i na krajnje desno mesto upiše 0 ili 1. Ako se stanja obeleže na sledeći način:

¹⁶Umesto fraze "stanja određene matricom stanja" koristimo ponekad kraće samo "stanje" uz navođenje njegove matrice stanja.

$p[-1]$	$p[0]$	$p[1]$	Stanje
0	0	0	1
1	0	0	2
0	1	0	3
1	1	0	4
0	0	1	5
1	0	1	6
0	1	1	7
1	1	1	8

graf prelaza automata koji predstavlja PSP model dat je na Slici 22.

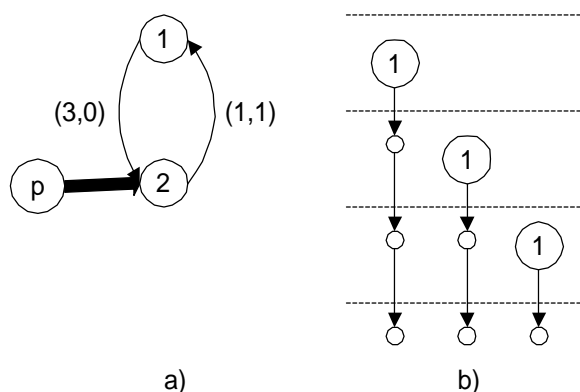


Slika 22: Graf prelaza stanja PSP modela jedne jednostavne petlje.

Posmatranjem grafa na Slici 22 može se uočiti jedno važno svojstvo. Za svako stanje S u grafu postoje 2 stanja P_1 i P_2 u koja se iz tog stanja S prelazi. U oba ova stanja P_1 i P_2 može se doći samo iz jednog stanja Q , što znači da se prelasci odvijaju "2 stanja u 2 stanja". Na primer, iz stanja 1 i 2 se prelazi u stanja 1 i 5. Iz stanja 5 i 6 se prelazi u stanja 3 i 7. Ovo svojstvo važi u opštem slučaju i jednostavno se dokazuje, što će biti ukinjeno u sledećoj glavi. Ovde ćemo pokazati to svojstvo na datom primeru. Naime, iz dva stanja $[x \ a \ b]$, gde su a i b konstantne vrednosti 0 ili 1, a umesto x stoji bilo šta, prelazi se u dva stanja $[a \ b \ x]$. Grupu polaznih stanja nazivamo *izvorišnim klasterom* (engl. *source cluster*, predlog), a grupu dolaznih stanja *odredišnim klasterom* (engl. *destination cluster*, predlog).

5.2. Pomeranja operacija

Sada će biti prikazano kako se PSP model može primeniti u procesu optimizacije koda petlji sa uslovnim grananjima. Posmatrajmo jedan jednostavan primer petlje čiji je ciklički graf zavisnosti dat na Slici 23a. Za slučaj da je ishod uslova p jednak T , trajanje iteracije je 4 ciklusa. Za slučaj da je ishod F , operacija 1 koja traje ukupno 3 ciklusa može da se rasporedi na sledeće ili sledeće dve iteracije, tako da svaka od tih iteracija traje po 1 ciklus. Na to ukazuje i izgled cikličkog grafa u slučaju ishoda F , koji se tada svodi samo na jednu operaciju 1.



Slika 23: Jedna jednostavna petlja sa uslovnim grananjem. (a) Ciklički graf zavisnosti. (b) Graf razmotane petlje za slučaj da je ishod u tri susedne iteracije F .

Ovakvom analizom zaključuje se da se efekat pomeranja operacija (zapravo prostiranja jedne operacije na višu iteraciju) prostire na tri susedne iteracije. Ako označimo tri ciklusa operacije 1 sa 1/1, 1/2 i 1/3, onda se u nekoj iteraciji može naći 1/2 ili 1/3 iz prethodne iteracije, odnosno dve iteracije ranije. Ovo navodi na intuitivan zaključak da predikatska matrica treba da izgleda ovako:

$$\begin{bmatrix} p[-2] & p[1] & p[0] \end{bmatrix}$$

Prema tome, PSP model ima 8 stanja sa prelazima kao na Slici 22. Pretpostavimo da je raspored operacija po stanjima u početku onakav kako to odgovara inicijalnoj petlji: u svim stanjima gde $p[0]$ ima vrednost 1 pojavljuju se sva tri dela operacije 1 i operacija 2, a tamo gde je vrednost $p[0]$ 0, pojavljuju se samo sva tri dela operacije 1:

Stanje [000] 1	Stanje [100] 2	Stanje [010] 3	Stanje [110] 4
1/1[0]	1/1[0]	1/1[0]	1/1[0]
1/2[0]	1/2[0]	1/2[0]	1/2[0]
1/3[0]	1/3[0]	1/3[0]	1/3[0]
Stanje [001] 5	Stanje [101] 6	Stanje [011] 7	Stanje [111] 8
1/1[0]	1/1[0]	1/1[0]	1/1[0]
1/2[0]	1/2[0]	1/2[0]	1/2[0]
1/3[0]	1/3[0]	1/3[0]	1/3[0]
2[0]	2[0]	2[0]	2[0]

Oznaka $op[i]$ kao i do sada označava instancu operacije op iz i iteracija dalje od tekuće iteracije. Nadalje, samo delove operacije 1 (1/1, 1/2 i 1/3) posmatrati potpuno ravnopravno sa običnom operacijom 2.

Sada nastupa ključni momenat primene PSP modela. Vratimo se preme stanju operacija u susedne iteracije u cilju smanjenja dužine izvršavanja stanja (smanjenje intervala iniciranja II). Pri tome se mora postaviti semantika programa koja se ogleda u tome da se odgovarajuće operacije mora izvršiti u svakom konkretnom izvršavanju koje prolazi kroz stanje S , ukoliko se ona nalazi u iteraciji opisanoj stanjem S u početnom rasporedu. Ovo pravilo se zadovoljava ukoliko se neka instanca operacije premesti iz svih stanja izvorišnog klastera u sva stanja odredišnog klastera ili obrnuto.

Na primer, iz stanja 1 i 2 koja pripadaju izvorišnom klasteru, može se premestiti operacija 1/3[0] koja je slobodna na dnu u ovim stanjima, u stanja 1 i 5 odredišnog klastera. Pri tome, u ovim novim stanjima operacija postaje 1/3[-1], jer se radi o pomeranju u narednu iteraciju. Takođe, u ovim

stanjima se ispituju zavisnosti po podacima od prido le operacije prema postojećim, i sve operacije raspoređuju prema tim zavisnostima. Kako u konkretnom slučaju od operacije $1/3[-1]$ ne zavisi ni jedna operacija, ova stanja se ne produžavaju, a kako se izvršavanje stanja 1 i 2 skraćuje, ukupno se dobija na vremenu izvršavanja. Rezultat opisanog pomeranja je sledeći:

Stanje [000] 1 1/1[0] 1/3[-1] 1/2[0]	Stanje [100] 2 1/1[0] 1/2[0]	Stanje [010] 3 1/1[0] 1/2[0] 1/3[0]	Stanje [110] 4 1/1[0] 1/2[0] 1/3[0]
Stanje [001] 5 1/1[0] 1/3[-1] 1/2[0] 1/3[0] 2[0]	Stanje [101] 6 1/1[0] 1/2[0] 1/3[0] 2[0]	Stanje [011] 7 1/1[0] 1/2[0] 1/3[0] 2[0]	Stanje [111] 8 1/1[0] 1/2[0] 1/3[0] 2[0]

Isti postupak se može nastaviti i dalje. Operacije $1/2[0]$ i $1/3[0]$ iz stanja 3 i 4 prelaze u stanja 2 i 6 i postaju $1/2[-1]$ i $1/3[-1]$. Zatim operacije $1/3[-1]$ i $1/2[0]$ iz stanja 1 i 2 prelaze u stanja 1 i 5 i postaju $1/3[-2]$ i $1/2[-1]$. Konačan raspored posle ovih pomeranja izgleda ovako:

Stanje [000] 1 1/1[0] 1/2[-1] 1/3[-2]	Stanje [100] 2 1/1[0] 1/2[-1]	Stanje [010] 3 1/1[0]	Stanje [110] 4 1/1[0]
Stanje [001] 5 1/1[0] 1/2[-1] 1/3[-2] 1/2[0] 1/3[-1] 1/3[0] 2[0]	Stanje [101] 6 1/1[0] 1/2[-1] 1/2[0] 1/3[-1] 1/3[0] 2[0]	Stanje [011] 7 1/1[0] 1/2[0] 1/3[0] 2[0]	Stanje [111] 8 1/1[0] 1/2[0] 1/3[0] 2[0]

Na ovaj način se dobija izvršavanje jedne iteracije u proseku od 2,5 ciklusa takta (prosečan interval iniciranja II), pod pretpostavkom da su verovatnoće svih stanja podjednake. Ovo upravo predstavlja raspored koji je intuitivno bio postignut posmatranjem grafa razmotane petlje na Slici 23b (trajanje 1 ili 4 ciklusa). Treba ipak napomenuti da se isti postupak može nastaviti i dalje, primenom novih pomeranja, čime se dobija još manji prosečan interval iniciranja, ali samo se mi ovde zaustaviti.

Samo iz razloga generisanja inicijalnog rasporeda, nadalje može se zahtevati da predikatska matrica poseduje kolonu sa indeksom [0]. Na ovaj način se inicijalni raspored dobija unošenjem operacija samo u odnosu na ishode uslova iz tekuće iteracije, što se može odrediti u fazi prevođenja.

Kao i kod tehnike EPS, i ovde se naglašava da je postupak pomeranja operacija potuno odvojen od heuristika izbora operacija za pomeranje. Postupak pomeranja operacija je deo modela PSP i bio bi formalno definisan i teorijski razmatran u Glavi 6. Heuristike za izbor pomeranja operacija mogu se proizvoljno dopunjavati i usavršavati, a neki predlozi biće izneseni u Glavi 7. Ovde se sada bitno ponovljena samo pravila za pomeranje operacija.

Zaključak 7: (O pomeranju operacija u PSP modelu) Operacija $op[i]$ se može premestiti iz izvorišnog u odredišni klaster akko ona postoji u svim stanjima izvorišnog klastera i u svima njima je slobodna na dnu [15]. Pri pomeranju se operacija $op[i]$ briše iz svih stanja izvorišnog klastera i pojavljuje u odredišnom klasteru kao operacija $op[i-1]$ (pomeranje u kasniju iteraciju, pomeranje "nadole"). U svim stanjima odredišnog klastera se ova operacija pojavljuje kao slobodna na vrhu, a razmatraju se zavisnosti po podacima između nje i već raspoređenih

operacija, čime se postiže novi raspored tih stanja. Analogno je pomeranje iz odredišnog u izvorišni klaster (pomeranje u prethodnu iteraciju, pomeranje "nagore"), samo što operacija $op[i]$ mora biti slobodna na vrhu u odredišnom klasteru, i što u izvorišnom klasteru postaje $op[i+1]$ i slobodna na dnu.

Ispitivanje zavisnosti po podacima pri ovako opisanom pomeranju operacija PSP modela je veoma jednostavno. Naime, ne postoji potreba za analizom promenljivih iteracionih distanci. Jednostavno, ako se u nekom stanju pojave operacije $op1[i]$ i $op2[j]$, njihova zavisnost postoji ili ne postoji, to se može jednostavno utvrditi posmatranjem njihovih izvorišnih i odredišnih operacija, i to je elementarna procedura (npr. kao u EPS). Na ovaj način se može vršiti i dinamičko preimenovanje ili kombinovanje kao u EPS. Efekat promenljive iteracione distance se pojavljuje kao jednostavno postojanje ili ne postojanje neke operacije u odgovarajućim stanjima, ba kao u grafu razmotrane petlje za konkretno izvršavanje.

Najzad, navedimo samo neke ideje za formiranje heuristika za izbor operacija koje se pomerati. Očigledno je, najpre, da je pomeranje neke operacije dobro izvršiti ukoliko se ukupna dužina izvršavanja svih stanja datim pomeranjem smanjuje. To se dešava ukoliko je broj stanja iz kojih se data operacija uzima i kojima se tim uzimanjem skraćuje izvršavanje veći od broja stanja u koja se operacija smešta i kojima se tim smeštanjem izvršavanje produžava. Pri tom, između raznih pomeranja koja donose istu promenu trajanja izvršavanja, treba izabrati ono koje je po nekom kriterijumu najbolje; to može biti npr. kriterijum najmanjeg spekulativnog izvršavanja. Eventualno, ukoliko se ukupno vreme izvršavanja ne menja, mogu se formulisati uslovi pod kojima pomeranje otvara mogućnost za dobitak pri narednim pomeranjima, to će biti detaljnije analizirano u Glavi 7.

5.3. Generisanje rezultujućeg koda

Razmotrimo sada kako se iz postignutog rasporeda PSP modela može dobiti izvršni kod optimizovane petlje. Pri tome se smatra da ne postoji nikakva hardverska podrška za direktnu implementaciju PSP modela uz koju bi, naravno, proces generisanja koda bio pojednostavljen. Proces ćemo posmatrati na istom primeru iz prethodnog poglavlja (Slika 23). Pretpostavimo, radi jednostavnosti, da je uslovna operacija zapravo operacija 1 i da se uslov razrešava u poslednjem delu (1/3) ove operacije. Ovo je donekle nerealna pretpostavka, jer postoji zavisnost po podacima između operacije 1 i kontrolno zavisne operacije 2, ali promena ove pretpostavke nimalo ne menja opšta razmatranja¹⁷.

Opiti princip je očigledan: stanja treba pretvoriti u razgranatu CASE strukturu u odnosu na vrednost predikata. Posmatrajmo ponovo završni raspored iz prethodnog poglavlja:

Stanje [000] 1 1/1[0] 1/2[-1] 1/3[-2]	Stanje [100] 2 1/1[0] 1/2[-1]	Stanje [010] 3 1/1[0]	Stanje [110] 4 1/1[0]
Stanje [001] 5 1/1[0] 1/2[-1] 1/3[-2] 1/2[0] 1/3[-1] 1/3[0] 2[0]	Stanje [101] 6 1/1[0] 1/2[-1] 1/2[0] 1/3[-1] 1/3[0] 2[0]	Stanje [011] 7 1/1[0] 1/2[0] 1/3[0] 2[0]	Stanje [111] 8 1/1[0] 1/2[0] 1/3[0] 2[0]

¹⁷Uostalom, može postojati antizavisnost između uslovne operacije i kontrolno zavisne operacije, to se na ovaj način jednostavno razrešava.

Rezultujući kod bi u principu izgledao ovako:

```
CASE (p[-2]p[-1]p[0]) OF
  000: ... // kôd za stanje 1
  100: ... // kôd za stanje 2
  ...
END CASE
```

Međutim, uoči se sledeći problem. U trenutku ispitivanja vrednosti predikata $p[0]$, $p[-1]$ i $p[-2]$ (na početku iteracije) nije izvršena operacija 1 koja određuje tu vrednost u svim stanjima ($1/3[0]$, $1/3[-1]$ i $1/3[-2]$). Zbog toga prikazana struktura nije moguća. Prema tome, sve razlike u sadržaju (rasporedu operacija) između stanja koja se razlikuju po nekom predikatu čiju vrednost nije moguće odrediti na početku iteracije predstavljaju zapravo spekulativno raspoređene operacije. Na primer, stanja 1 i 5 se razlikuju po vrednosti predikata $p[0]$ koji se u slučaju ishoda T istog predikata razrešava u istoj iteraciji u trećem ciklusu, a u slučaju ishoda F u nekoj od narednih iteracija (u stanjima koji su sledbenici stanja 1). Ovo ukazuje na isti problem koji je uočen u Glavi 4 (Zaključak 6): duina izvršavanja stanja (iteracija) 1 i 5 je različita, a zavisi od vrednosti predikata $p[0]$ koji se ne razrešava u toku delova koda koji su duine jednake duini kraja iteracije (u ovom slučaju ne razrešava se u prvom ciklusu). Zbog toga je potrebno primeniti taktiku izjednačavanja duina stanja koja se razlikuju po duini, a ta razlika je spekulativna. U ovom slučaju, treba izjednačiti po duini stanja 1 i 5, 2 i 6, 3 i 7, 4 i 8, ukoliko se uslov $p[0]$ ne razrešava na vreme; slično, treba izjednačiti po duini stanja 1 i 3, 2 i 4, 5 i 7, 6 i 8, ukoliko se uslov $p[-1]$ ne razrešava na vreme; isto važi i za stanja 1 i 2, 3 i 4, 5 i 6, 7 i 8 i uslov $p[-2]$.

Izjednačavanje se može sprovesti pomeranjem operacija iz duine stanja u naredne iteracije. Na primer, stanja 1 i 5 nisu iste duine; prebacimo iz stanja 5 i 6 sve operacije iz ciklusa 2, 3 i 4 u stanja 3 i 7:

Stanje [000] 1 1/1[0] 1/2[-1] 1/3[-2]	Stanje [100] 2 1/1[0] 1/2[-1]	Stanje [010] 3 1/2[-1] 1/3[-2] 1/3[-1] 2[-1] 1/1[0]	Stanje [110] 4 1/1[0]
Stanje [001] 5 1/1[0] 1/2[-1] 1/3[-2]	Stanje [101] 6 1/1[0] 1/2[-1]	Stanje [011] 7 1/2[-1] 1/3[-2] 1/3[-1] 2[-1] 1/1[0] 1/2[0] 1/3[0] 2[0]	Stanje [111] 8 1/1[0] 1/2[0] 1/3[0] 2[0]

Sada opet stanja 3 i 7 nisu iste duine, pa treba prebaciti tri poslednje operacije iz stanja 7 i 8 u stanja 4 i 8.

Stanje [000] 1 1/1[0] 1/2[-1] 1/3[-2]	Stanje [100] 2 1/1[0] 1/2[-1]	Stanje [010] 3 1/2[-1] 1/3[-2] 1/3[-1] 2[-1] 1/1[0]	Stanje [110] 4 1/2[-1] 1/3[-1] 2[-1] 1/1[0]
Stanje [001] 5 1/1[0] 1/2[-1] 1/3[-2]	Stanje [101] 6 1/1[0] 1/2[-1]	Stanje [011] 7 1/2[-1] 1/3[-2] 1/3[-1] 2[-1]	Stanje [111] 8 1/2[-1] 1/3[-1] 2[-1]

		1/1[0]	1/1[0]
--	--	--------	--------

Opet stanja 1 i 3 nisu jednake dužine, a uslov $p[-1]$ nije razrešen na vreme. Zato se operacije iz tri poslednja ciklusa stanja 3 i 4 sele u stanja 2 i 6:

Stanje [000] 1 1/1[0] 1/2[-1] 1/3[-2]	Stanje [100] 2 1/1[0] 1/3[-2] 2[-2] 1/1[-1] 1/2[-1]	Stanje [010] 3 1/2[-1] 1/3[-2]	Stanje [110] 4 1/2[-1]
Stanje [001] 5 1/1[0] 1/2[-1] 1/3[-2]	Stanje [101] 6 1/1[0] 1/3[-2] 2[-2] 1/1[-1] 1/2[-1]	Stanje [011] 7 1/2[-1] 1/3[-2] 1/3[-1] 2[-1] 1/1[0]	Stanje [111] 8 1/2[-1] 1/3[-1] 2[-1] 1/1[0]

Sada opet stanja 3 i 7 nisu iste dužine, pa se operacije iz tri poslednja ciklusa stanja 7 i 8 sele u stanja 4 i 8:

Stanje [000] 1 1/1[0] 1/2[-1] 1/3[-2]	Stanje [100] 2 1/1[0] 1/3[-2] 2[-2] 1/1[-1] 1/2[-1]	Stanje [010] 3 1/2[-1] 1/3[-2]	Stanje [110] 4 1/3[-2] 2[-2] 1/1[-1] 1/2[-1]
Stanje [001] 5 1/1[0] 1/2[-1] 1/3[-2]	Stanje [101] 6 1/1[0] 1/3[-2] 2[-2] 1/1[-1] 1/2[-1]	Stanje [011] 7 1/2[-1] 1/3[-2]	Stanje [111] 8 1/3[-2] 2[-2] 1/1[-1] 1/2[-1]

Konačno, dobijen je raspored iz koga se može generisati kod. Naime, sada su stanja 1 i 5, 2 i 6, 3 i 7, 4 i 8 iste dužine, što omogućava i za stanja 1 i 3, 2 i 4, 5 i 7, 6 i 8. Stanja 1 i 2, 3 i 4, 5 i 6, 7 i 8 su različitih dužina (1 i 4 ciklusa), ali se vrednost predikata $p[-2]$ saznaje u prvom ciklusu, pa se može odrediti da li se ostatak stanja od tri ciklusa izvršava ili ne. Tada, stanja 1 i 5, 2 i 6, 3 i 7, 4 i 8 sadrže potpuno identične rasporede, što znači da raspored uopšte ne zavisi od predikata $p[0]$, pa se ovaj predikat može izbaciti. Zato ćemo nadalje posmatrati samo stanja 1, 2, 3 i 4. Stanja koja se razlikuju po predikatu $p[-1]$ (1 i 3, 2 i 4) se razlikuju po rasporedu, a predikat $p[-1]$ se ne razrešava na vreme da bi se te razlike izvršavale samo kada je potrebno. Zato su sve ove razlike spekulativne. Uzevši to u obzir, mogu se spojiti stanja 1 i 3, 2 i 4, tako da se operacija 1/1[0] u oba slučaja izvršava spekulativno. Najzad, ako se posmatraju stanja 1 i 2, njihova razlika počev od drugog ciklusa može se "razdvojiti" uslovom koji se razrešava u prvom ciklusu (operacija 1/3[-2]), ali razlika u prvom ciklusu (operacija 1/2[-1]) predstavlja takođe spekulativno izvršavanje. Tako se dobija sledeći kod u simboličkom zapisu koji je prilagođen VLIW mašini, pri čemu se uslovni skok izvršava na kraju prvog ciklusa [7] (spekulativne operacije označene su slovom s):

Ciklus:	Operacije:
LOOP	
1	1/1[0]s 1/2[-1]s 1/3[-2]
	IF (rezultat 1/3[-2] je T)
2	2[-2]
3	1/1[-1]
4	1/2[-1]

END IF

END LOOP

Za sluĉaj skalarnih operacija, u općem sluĉaju ostaje problemivotnog veka promenljive. Ovaj problem se rešava ili promenom u vektorsku operaciju, ili razmotavanjem optimizovane petlje potreban broj puta uz preimenovanje [39]. Drugi problem je generisanje kompenzacionih operacija spekulativnog izvršavanja. Ovi problemi neće se dalje razmatrati u ovom radu, pa ostaju za dalja istraživanja. Ovaj rad se dominantno bavi raspoređivanjem operacija, a u Glavi 7 će biti postavljeni samo neki osnovni principi potrebni za proces generisanja koda.

5.4. Intuitivna svojstva modela

Opisani PSP model se na pokazanim primerima ponaša prilično pravilno, tako da se intuitivno mogu naslutiti mnoga njegova bitna svojstva. Ovde će ta svojstva biti objašnjena.

5.4.1. Svojstvo težnje ka optimalnom

Intuitivno je jasno da u općem sluĉaju petlje sa uslovnim grananjima može da se dobije bolji raspored u smislu kraćeg proseĉnog intervala iniciranja II , ukoliko se posmatra kombinacija ishoda uslova iz većeg broja susednih iteracija. Pokazano kako se ovo intuitivno shvatanje uklapa u PSP model.

Posmatrajmo isti primer petlje sa Slike 23. Pogledajmo kako se ponaša PSP model u sluĉaju da se razmatraju ishodi samo jedne iteracije; drugim reĉima, neka predikatska matrica ima jednostavno jedan elemenat: $[p[0]]$. Tada postoje samo dva stanja 1 i 2 sa sledećim inicijalnim rasporedom:

Stanje	[0]	1
	1/1[0]	
	1/2[0]	
	1/3[0]	

Stanje	[1]	2
	1/1[0]	
	1/2[0]	
	1/3[0]	
	2[0]	

Graf prelaza automata koji predstavlja ovaj model je jednostavan: iz stanja 1 i 2 prelazi se u stanja 1 i 2. Operacija 1/1[0] je jedina slobodna na vrhu u oba stanja odredi nog klastera 1 i 2, pa se jedino ova operacija može pomeriti u prethodnu iteraciju, tj. u oba stanja izvori nog klastera 1 i 2 (pomeranje nagore). Posle ovog pomeranja dobija se sledeći raspored:

Stanje	[0]	1
1/2[0]	1/1[+1]	
1/3[0]		

Stanje	[1]	2
1/2[0]		
1/3[0]		
2[0]		
1/1[+1]		

Sada je jedina operacija koja se može dalje pomeriti nagore, a koja postoji u oba stanja i slobodna je na vrhu, operacija 1/2[0]. Međutim, pomeranje ove operacije ne bi donelo nikakav dobitak, jer bi raspored bio sledeći:

Stanje	[0]	1
1/3[0]	1/1[+1]	
	1/2[+1]	

Stanje	[1]	2
1/3[0]		
2[0]		
1/1[+1]		
1/2[+1]		

Naravno, nijedna operacija u polaznom rasporedu nije mogla biti preseljena nadole, jer ne postoji ista operacija koja je slobodna na dnu u oba stanja.

Uzrok nemogućnosti dalje optimizacije leži upravo u tome što predikatska matrica dozvoljava praćanje efekta slaganja ishoda uslova samo jedne iteracije, dok petlja ima inherentno svojstvo slaganja ishoda iz višeg od jedne iteracije. Proširimo sada predikatsku matricu na $[p[0] \ p[1]]$ i počnimo ponovo od početnog rasporeda:

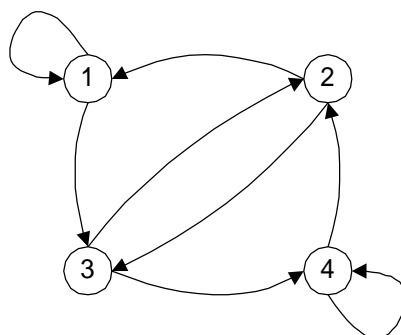
Stanje	[00]	1
	1/1[0]	
	1/2[0]	
	1/3[0]	

Stanje	[10]	2
	1/1[0]	
	1/2[0]	
	1/3[0]	
	2[0]	

Stanje	[01]	3
	1/1[0]	
	1/2[0]	
	1/3[0]	

Stanje	[11]	4
	1/1[0]	
	1/2[0]	
	1/3[0]	
	2[0]	

Graf prelaza ovog modela dat je na Slici 24.



Slika 24: Graf prelaza PSP modela sa 4 stanja

Izvršimo ponovo pomeranje operacije 1/1[0] iz stanja 1 i 3 u stanje 1 i 2, a zatim iste operacije iz stanja 2 i 4 u stanja 3 i 4:

Stanje	[00]	1
	1/2[0] 1/1[1]	
	1/3[0]	

Stanje	[10]	2
	1/2[0]	
	1/3[0]	
	2[0]	
	1/1[1]	

Stanje	[01]	3
	1/2[0] 1/1[1]	
	1/3[0]	
	1/1[1]	

Stanje	[11]	4
	1/2[0]	
	1/3[0]	
	2[0]	
	1/1[1]	

Ovde ćemo primetiti jedno važno svojstvo koje izgleda da važi u opštem slučaju i koje će biti formalno definisano u Glavi 6 (za sada je bez dokaza). Naime, dobili smo isti raspored kao za PSP model sa predikatskom matricom $[p[0]]$ za svako stanje u kome $p[0]$ ima odgovarajuću vrednost 0 odnosno 1. Drugačije posmatrano, u PSP modelu sa proširenom predikatskom matricom mogli smo da izvršimo sva pomeranja koja smo mogli da izvršimo i u početnom modelu. Zbog toga se početni raspored za PSP model sa proširenom predikatskom matricom može dobiti od proizvoljnog postignutog rasporeda polaznog PSP modela, s tim da se za svako stanje polaznog modela (u primeru stanja [0] i [1]) generiše po 2 stanja sa istim rasporedom (u primeru nova stanja [00] i [01], odnosno [10] i [11]). Ovo svojstvo je intuitivno sasvim očigledno, jer se PSP model može proizvoljno proširivati tako što se proširuje opseg "posmatranja" iteracija, a rasporedi po stanjima ne zavise od novododatih predikata.

Zaključak 8: (O proširenju PSP modela) Sva pomeranja operacija koja se mogu izvršiti u nekom PSP modelu, mogu se izvršiti i u PSP modelu dobijenom proširivanjem predikatske matrice

u širinu novim predikatima. Zbog toga se polazni raspored operacija proširenog modela može dobiti "kopiranjem" rasporeda stanja prvog modela u sva stanja koja imaju iste vrednosti onih predikata koji su postojali i u prvom modelu.

Ako postupak pomeranja operacija nastavimo dalje, možemo preseliti operaciju 1/1[1] iz stanja 1 i 3 u stanja 1 i 2, zatim operaciju 1/2[0] iz stanja 1 i 3 u stanja 1 i 2, a zatim operaciju 1/2[0] iz stanja 2 i 4 u stanja 3 i 4. Tada se dobija sledeći raspored:

Stanje	[00]	1
1/3[0]	1/2[1]	1/1[2]

Stanje	[10]	2
1/3[0]	1/1[2]	
2[0]		
1/1[1]		
1/2[1]		

Stanje	[01]	3
1/1[1]	1/3[0]	
1/2[1]		

Stanje	[11]	4
1/3[0]		
2[0]		
1/1[1]		
1/2[1]		

Dakle, prosečno vreme izvršavanja se donekle smanjilo. Uzrok je u tome što je omogućeno da se operacija pod slovom F u dve susedne iteracije, što obuhvata i periodičan slučaj ishoda F u tri iteracije (prelaz iz stanja [00] u isto to stanje). Za ovaj slučaj je dobijeni prosečni II jednak $11/4$ (to je manje od $7/2$) ciklusa. Bez dokaza možemo ovde navesti da se daljim proširivanjem modela može postići bolji raspored, a da se za model sa matricom $[p[0] p[1] p[2]]$ može postići potpuno ekvivalentan raspored onom koji je postignut u modelu sa matricom $[p[-2] p[-1] p[0]]$; ovo drugo tvrđenje je posledica potpune ekvivalencije modela sa matricom $[p[0] p[1] p[2]]$ sa pomeranjima operacija nagore, i modela sa matricom $[p[-2] p[-1] p[0]]$ sa pomeranjima operacija nadole. Dakle, proširenjem "opsega posmatranja" dobili smo bolji raspored za neke staze izvršavanja. Ovo navodi na intuitivan, takođe veoma važan zaključak koji će biti i formalno dokazan u sledećoj glavi:

Zaključak 9: (O težnji PSP modela ka optimalnom) PSP model teži ka optimalnom u sledećem intuitivnom značenju: Ako postoji neka staza izvršavanja za koju postoji bolji raspored operacija od onog postignutog datim PSP modelom, onda se može pronaći PSP model sa proširenim predikatskom matricom u kome se može postići takav bolji raspored.

Dakle, PSP model "koži" bolje raspoređivanje nekih operacija za neku stazu izvršavanja na taj način što te operacije ili ne postoje u svim stanjima klastera iz koga ih je potrebno premestiti, ili nisu u svim stanjima tog klastera slobodne na vrhu odnosno dnu. Dokaz navedenog svojstva te nje ka optimalnom može se upravo zasnivati na tvrdnji da prošireni PSP model obezbeđuje postojanje i "slobodu" ovih operacija u celom klasteru.

5.4.2. Mogućnost eliminacije proizvoljne predikcije

U poglavlju 5.3. pokazano je na primeru kako se u principu eliminiše problem "proizvoljne predikcije". Ovaj problem se, kao što je rečeno, sastoji u tome da se neka dva stanja razlikuju (čak i po dužini izvršavanja), ali ishod predikata koji razlikuje ta dva stanja se ne saznaje na vreme. Problem, u stvari, nastaje zbog toga što PSP model polazi od apstraktne postavke predikatske matrice koja može da uključi predikate sa proizvoljnom "dalekovidišću", tj. da se stanja razlikuju po predikatima koji će budućnosti biti rešeni, a uzimaju se pri raspoređivanju kao poznati.

Primer iz poglavlja 5.3. navodi na sledeće zaključke vezane za proces eliminacije ovakve proizvoljne predikcije u smislu izjednačavanja dužina izvršavanja odgovarajućih stanja, uz spekulativno izvršavanje.

Prvo, može se pomisliti da se opisani postupak pmeranja operacija dok se sva potrebna stanja ne izjednače po dužini izvršavanja uvek završava.

Drugo, može se verovati da su potrebna pomeranja uvek izvodljiva, u smislu da u celom izvorišnom (određenom) klasteru postoje iste operacije koje je potrebno uzeti iz nekog stanja čije se skraćuje. Naime, u datom primeru, mi smo prvo eleli da izjednačimo dužine stanja 5 i 1,

to je uĹinjeno selidbom operacija iz stanja 5 i 6. Pri tome smo u stanju 6 prona Ĺli sve iste operacije koje je bilo potrebno uzeti iz stanja 5. Da li je to u op Ĺtem sluĹaju uvek zadovoljeno?

TreĹe na kraju se dobio raspored koji po srednjem vremenu nije ni Ĺta du Ĺ od polaznog (2,5 ciklusa po iteraciji u proseku), dakle postupak je sproveden bez gubitka kvaliteta rasporeda. Da li je to uvek tako?

Odgovori na ova pitanja ostaju za sada samo intuitivni zakljuĹci bez dokaza. Dalja istra Ĺivanja treba da poka u da li neke od ovih pretpostavki uvek va e. Ĺak i ako nije tako, postoje re Ĺenja problema koja mogu da daju manje dobar raspored, ali iz koga je moguĹe generisati kôl. Ova re Ĺenja biĹe diskutovana u Glavi 7.

6. Teorijski model i njegova analiza

U prethodnoj glavi neformalno je prikazan PSP model. Na primerima su uoĹene mnoge njegove pravilnosti. Ove pravilnosti umnogome povećavaju upotrebljivost i znaĹaj modela. Zbog toga će neke od njih u ovoj glavi biti i formalno dokazane. Naravno, pre toga ćemo sam PSP model biti formalno postavljen. Ovakva formalna matematiĹka postavka modela otvara mogućnost za dalja njegova ispitivanja, to opet daje veću ansku za njegovu upotrebljivost. Zbog svega toga rezultati ove teorijske analize imaju dalekose an znaĹaj.

6.1. Pretpostavke i ograniĹenja

Kod formalne postavke PSP modela biće uvedena neka ograniĹenja. Uslovi za ukidanje tih ograniĹenja biće analizirani u sledećoj glavi.

Prvo, smatraćese da sve operacije traju po jedan ciklus takta. U principu, operacije koje traju više ciklusa potrebno je podeliti na delove pri Ĺemu svaki deo traje jedan ciklus, i dalje posmatrati nezavisno, kao što je to uĹinjeno u prethodnoj glavi.

Drugo, smatraćese da se u fazi rasporeĹivanja može detektovati da li postoji zavisnost po podacima između dve instance nekih operacija iz iteracija koje su na poznatom rastojanju. Pri tome se ne pravi razlika između pravih, izlaznih ili antizavisnosti: mogu se posmatrati sve ili samo neke od njih, u zavisnosti od toga da li se vrši preimenovanje ili ne. Zavisnosti će dakle, biti posmatrane jedinstveno.

Dalje, pretpostavlja se da ne postoji ograniĹenje u resursima. Broj raspoloivih resursa je konaĹan, ali neograniĹen: raspored dobijen PSP modelom će koristiti broj resursa ograniĹen nekim konaĹnim celim brojem. U toku rasporeĹivanja neće se voditi računa o ograniĹenjima u resursima.

Smatraćese takoĹ da je rezultujućol prilagoĹen VLIW mašini i da se može u principu generisati u skladu sa zakljuĹcima iz prethodne glave. Ovo podrazumeva i neograniĹenu dubinu spekulativnog izvršavanja, odnosno konaĹnu, ali neograniĹenu "daljinu gledanja unapred".

Pojmovi i oznake u daljem tekstu biće u skladu sa dosadašnjim izlaganjem.

6.2. Formalna definicija PSP modela

U ovom poglavlju biće formalno definisan PSP model. Najpre će biti uvedeni pojmovi operacije, zavisnosti po podacima, predikata i rasporeda operacija. Zatim će biti definisani pojmovi vezani za predikatsku matricu PSP modela. Posle toga će biti date definicije pomeranja operacija. Najzad, biće dokazana neka osnovna jednostavna svojstva PSP modela i definisan pojam proirrenog PSP modela.

6.2.1. Formalne definicije osnovnih elemenata

Polazi se od konaĹnog skupa operacija: $OP = \{op1, op2, \dots, opk\}$. To su operacije iz tela polazne petlje koja se optimizuje, ukljuĹujući uslovne operacije. Sledi definicija pojma *instance operacije* (engl. *instance of operation*, predlog).

Definicija 3: (*Instanca operacije*) UreĹeni par $(opi, j) \in OP \times Z$, gde je Z skup celih brojeva, naziva se *instancom j operacije opi*. Dalje će se koristiti oznaka $opi[j]$ i kraće govoriti samo "*operacija opi[j]*". Skup instanci operacija biće oznaĹavan sa $OPI = OP \times Z$. \square

Definicija 4: (Zavisnost po podacima) Relacija strogog uređenja¹⁸ $d \hat{=} OPI \times OPI$ naziva se relacijom zavisnosti po podacima. Dalje će se koristiti oznaka $d(op1[j1], op2[j2])$ i govoriti da "op2[j2] zavisi od op1[j1]" ukoliko je $(op1[j1], op2[j2]) \hat{=} d$. \square

Sledeći polazni pojam je konačni skup *predikata* (engl. *predicates*): $P = \{p1, p2, \dots, pm\}$. Za broj elemenata ovog skupa biće rezervisan simbol m : $card(P) = m$. Sledi definicija *instance predikata* (engl. *instance of predicate*, predlog).

Definicija 5: (Instanca predikata) Uređeni par $(pi, j) \hat{=} PxZ$, gde je Z skup celih brojeva, naziva se instancom j predikata pi . Dalje će se koristiti oznaka $pi[j]$ i kraće govoriti samo "predikat $pi[j]$ ". Skup instanci predikata biće označavan sa $PI = PxZ$. \square

Definicija 6: (Raspored) Preslikavanje $R: OPI' \rightarrow N$, gde je OPI' konačan podskup skupa OPI , a N skup prirodnih brojeva, naziva se rasporedom (engl. *schedule*), akko (po def.) važi:

$$("op1[j1] \hat{=} OPI') ("op2[j2] \hat{=} OPI') (d(op1[j1], op2[j2]) \hat{=} R(op1[j1]) < R(op2[j2]))$$

$R(op[i])$ se naziva ciklusom u koji je operacija raspoređena. Dalje će se koristiti oznaka $Dom(R)$ (domen) za skup OPI' . \square

Očigledno je, po to je $Dom(R)$ konačan, da postoji broj $L \in N$ takav da je to najveći broj koji predstavlja $R(op[i])$ za neko $op[i]$. Ovaj broj L biće nazivan *dužinom* rasporeda R , sa oznakom $Len(R)$.

Definicija 7: (Minimalni raspored) Raspored $R: OPI' \rightarrow N$ je minimalan akko (po def.) za svaku instancu operacije $op[i]$ iz OPI' , za koju je $R(op[i]) = k$, postoji niz instanci $op_1[i_1], op_2[i_2], \dots, op_{k-1}[i_{k-1}], op_k[i_k] = op[i]$ iz OPI' , takav da je $d(op_j[i_j], op_{j+1}[i_{j+1}])$, $j = 1, \dots, k-1$. \square

Ovakva definicija minimalnog rasporeda potpuno se poklapa sa Definicijom 1 vremenski optimalnog izvršavanja. Sledeće svojstvo je intuitivno jasno, ali ga navodimo u obliku leme, jer ćemo se kasnije pozivati na nju.

Lema 1: (O jedinstvenosti minimalnog rasporeda) Neka je OPI' neki konačni skup instanci operacija i d data relacija zavisnosti. Tada postoji jedan i samo jedan minimalni raspored $R: OPI' \rightarrow N$.

Dokaz: Dokazaćemo najpre da minimalni raspored R postoji. Kako je OPI' konačan, a d relacija strogog uređenja, može se jednostavno pokazati da postoji $op[i]$ takav da ne postoji $op'[j]$ takav da je $d(op'[j], op[i])$. Neka je $R(op[i]) = 1$ za sve ovakve $op[i]$. Neka se sada iz OPI' izbace svi ovakvi $op[i]$. Može se jednostavno pokazati da relacija d na ovako suenom skupu ostaje relacija strogog uređenja. Zato se postupak ponavlja, pri čemu se u sledećem koraku postavlja $R(op[i]) = 2$, itd. Kako je OPI' konačan, ovaj postupak se završava. Lako se pokazuje da je ovakav raspored minimalan¹⁹.

Pokazaćemo sada da je opisani raspored jedinstven. Pretpostavimo suprotno, da postoje dva minimalna rasporeda $R1 \neq R2$. Sledi da postoji $op[i]$ takav da je $R1(op[i]) \neq R2(op[i])$. Bez gubitka opštosti, pretpostavimo da je $R1(op[i]) < R2(op[i])$. Neka je $R1(op[i]) = c1$, $R2(op[i]) = c2$. Prema definiciji minimalnog rasporeda, postoji niz dužine $c2$ operacija $op_j[i_j]$, $j = 1, \dots, c2-1$, $op_{c2}[i_{c2}] = op[i]$, takav da važi $d(op_j[i_j], op_{j+1}[i_{j+1}])$, $j = 1, \dots, c2-1$. Prema definiciji rasporeda, sledi da mora biti $R1(op_j[i_j]) < R1(op_{j+1}[i_{j+1}])$, $j = 1, \dots, c2-1$. Kako su ovi brojevi prirodni i međusobno različiti, svi manji ili jednaki $c1$, a ima ih ukupno $c2 > c1$, dolazi se do kontradikcije. Ovim je dokaz završen. \square

¹⁸Nije refleksivna, jeste antisimetrična i jeste tranzitivna.

¹⁹Opisani postupak zapravo predstavlja postupak topološkog sortiranja grafa zavisnosti datog relacijom d .

6.2.2. Definicije predikatske matrice i stanja

Definicija 8: (Predikatska matrica) Preslikavanje $PM: P \times Z' \rightarrow PI \cup \{b\}$, gde je Z' konačan podskup skupa celih brojeva Z , P skup predikata i PI skup onstanci predikata P , naziva se predikatskom matricom (engl. predicate matrix, predlog) akko (po def.) zadovoljava sledeće uslove:

- 1° $(\text{"}i, j\hat{I}Z')(\text{"}k\hat{I}Z)(i \neq k \rightarrow PM(p, i) \neq PM(p, k))$
- 2° $0\hat{I}Z'$
- 3° $PM(p, i) \in \{p[i], b\}$
- 4° $PM(p, 0) = p[0]$
- 5° $(\text{"}z_{\min}\hat{I}Z')(\text{"}z\hat{I}Z)(z \geq z_{\min} \rightarrow PM(p, z) \neq b)$
- 6° $(\text{"}z_{\max}\hat{I}Z')(\text{"}z\hat{I}Z)(z \leq z_{\max} \rightarrow PM(p, z) \neq b)$
- 7° $(\text{"}p\hat{I}P)(\text{"}i, j\hat{I}Z)(i < j \rightarrow PM(p, i) \neq b \rightarrow PM(p, j) \neq b)$

Za broj elemenata skupa Z' biće rezervisan simbol n : $\text{card}(Z') = n$. $PM(p, i)$ naziva se elementom predikatske matrice. Skup Z' naziva se skupom indeksa. Predikatska matrica biće predstavljana u obliku pravougaone šeme - matrice sa m vrsta i n kolona, pri čemu odgovarajući elementi mogu da budu b . Elementi koji su jednaki $p[i]$ nazivaće se $ne-b$ elementima, a elementi jednaki b - b -elementima. Indeks z_{\min} naziva se najmanjim, a indeks z_{\max} najvećim indeksom matrice. \square

Prvi uslov iz navedene definicije zahteva zapravo da elementi podskupa celih brojeva Z' budu susedni. Drugi uslov zahteva da u tom skupu bude indeks 0, samo iz razloga generisanja početnog rasporeda, kako je to rešeno u prethodnoj glavi. Treći uslov specifikuje zapravo da su elementi matrice ili instance predikata, ili simboli b . Četvrti uslov opet zahteva da se u svakoj vrsti matrice nalazi $ne-b$ element $p[0]$. Peti i šesti uslov specifikuju da krajnja leva i desna kolona matrice na sadrže samo simbole b , čime bi se matrica nepotrebno proširivala. Sedmi uslov predstavlja zahtev da u jednom redu matrice ne postoje " b -praznine" između $ne-b$ elemenata, kako je to pokazano u prethodnoj glavi.

Definicija 9: (Ivični $ne-b$ element i ivica predikatske matrice) $Ne-b$ element predikatske matrice $PM(p, i)$ naziva se levim ivičnim elementom akko (po def.) je i najmanji indeks ili ako je $PM(p, i-1) = b$. Skup svih levih ivičnih elemenata naziva se leva ivica. Analogno za desni ivični element i desnu ivicu. \square

Definicija 10: (Matrica stanja) Neka je PM predikatska matrica sa skupom predikata P i skupom indeksa Z' . Preslikavanje $SM: P \times Z' \rightarrow \{0, 1, b\}$ naziva se matricom stanja (engl. state matrix, predlog), akko važi:

$$(\text{"}p\hat{I}P)(\text{"}i\hat{I}Z)(PM(p, i) = b \rightarrow SM(p, i) = b)$$

$SM(p, i)$ nazivaće se elementom matrice stanja. SM će se prikazivati u obliku šeme (matrice) kao i PM , samo sa elementima 0, 1 i b . Ivičnim elementima (levim i desnim) i ivicama biće nazivani pojmovi potpuno analogno definisani kao i za PM . \square

Definicija 11: (Stanje) Za datu predikatsku matricu PM , uređeni par (SM, R) gde je SM neka matrica stanja, a R neki minimalni raspored, naziva se stanjem (engl. state). \square

Definicija 12: (Kompletna skup stanja) Za datu predikatsku matricu PM , skup stanja (SM, R_i) , takav da se svaka postojeća matrica stanja SM predikatske matrice PM pojavljuje u ovom skupu parova tačno jednom, naziva se kompletna skup stanja date matrice PM (engl. complete state set, predlog). \square

6.2.3. Definicije grafa prelaza i klastera

Definicija 13: (Stanje sledbenik i stanje prethodnik) Neka su $S1=(SM1,R1)$ i $S2=(SM2,R2)$ dva stanja definisana predikatskom matricom PM . Akko (po def.) za svaki ne- b element $SM1(p,i)$ koji ne pripada levoj ivici $SM1$, važi da je $SM1(p,i)=SM2(p,i-1)$, onda je stanje $S1$ prethodnik stanja $S2$, odnosno $S2$ sledbenik stanja $S1$. Govori se još i da iz stanja $S1$ postoji prelaz u stanje $S2$. \square

Ovakva definicija zapravo odslikava Ĺinjenicu da se stanja sledbenici dobijaju od stanja prethodnika pomeranjem ne- b sadr aja matrice za jedno mesto ulevo. Treba primetiti da u nekoj vrsti predikatske matrice (pa i matrice stanja) mo e postojati i samo jedan ne- b element, a to je $p[0]$, kada za tu vrstu va i da ima isti element koji pripada i levoj i desnoj ivici, pa se pri navedenom pomeranju ulevo on "gubi", tako da u sledbeniĹkom stanju mo e uzimati bilo koju vrednost.

Definicija 14: (Graf prelaza) Usmereni graf u kome Ĺvorovi predstavljaju stanja kompletnog skupa stanja neke predikatske matrice PM , a svaka grana polazi od Ĺvora koji predstavlja stanje-prethodnika do Ĺvora koji predstavlja stanje-sledbenika, naziva se grafom prelaza date predikatske matrice PM . \square

Sada Ĺemo dokazati ranije najavljeno svojstvo da svih 2^m stanja koja predstavljaju prethodnike nekog stanja S , imaju ukupno 2^m sledbenika, odnosno da svako od ovih stanja-prethodnika S ima isti skup sledbenika. Ovo svojstvo Ĺenam omoguĹa da formalno defini emo ranije intuitivno uveden pojam klastera.

Lema 2: (O svojstvu prelazaka PSP modela) Neka iz stanja S postoje prelazi u stanja P_i , $i=1,...,k$. Neka se u stanje P_i prelazi iz stanja S_{ij} , $j=1,...,l$, pri Ĺemu je jedno od njih baš S . Tada su svi skupovi $\{S_{ij} | j=1,...,l\}$, $i=1,...,k$ jednaki, i broj njihovih elemenata je jednak 2^m .

Dokaz: OĹigledno je da je $k=l=2^m$, jer je broj iviĹnih elemenata (levih, kao i desnih) matrice ba 2^m . Iz stanja S se prelazi u 2^m onih stanja P_j . Ĺije matrice stanja imaju ne- b elemente koji nisu na desnoj ivici, jednake redom ne- b elementima koji nisu na levoj ivici matrice stanja S . Prema tome, iz svakog od ukupno 2^m stanja, meĹ kojima je i S , prelazi se u 2^m stanja, meĹ kojima je i P_j . Kako svako stanje ima taĹno 2^m sledbenika i isto toliko prethodnika, svi skupovi $\{S_{ij} | j=1,...,2^m\}$ su jednaki. \square

Definicija 15: (Izvorišni i odredišni klaster) Skup svih stanja prethodnika nekog stanja S naziva se izvorišni klaster. Skup svih stanja sledbenika stanja iz izvorišnog klastera naziva se odredišni klaster. \square

Potpuno ekvivalentno, izvori nim klasterom naziva se skup svih 2^m stanja koji imaju zajedniĹkih 2^m sledbenika, i analogno za odredi ni klaster.

6.2.4. Definicija pomeranja operacija

Definicija 16: (Pomeranja operacija) Neka je PM neka predikatska matrica i S neki kompletan skup stanja $S_i=(SM_i,R_i)$ date predikatske matrice.

Kaže se da se kompletan skup S' stanja $S'_i=(SM'_i,R'_i)$ iste predikatske matrice PM može dobiti pomeranjem operacije $op[i]$ iz odredišnog klastera Cd nagore u izvorišni klaster Cs , u oznaci: $moveup(Cd,op[i])$, od skupa S , akko (po def.) važi:

- 1° ($"S_i \hat{I} S)("S'_i \hat{I} S')(SM_i=SM'_i \hat{U} S_i \hat{I} Cs \hat{U} S'_i \hat{I} Cd \hat{P} R_i=R'_i)$)
- 2° ($"S_i \hat{I} Cd)(op[i] \hat{I} Dom(R_i))$)
- 3° ($"S_i \hat{I} Cd)("op'[i'] \hat{I} Dom(R_i))(\emptyset d(op'[i'],op[i])$)
- 4° ($"S_i \hat{I} Cd)("S'_i \hat{I} S')(SM_i=SM'_i \hat{P} Dom(R'_i)=Dom(R_i) \setminus \{op[i]\})$)
- 5° ($"S_i \hat{I} Cs)("S'_i \hat{I} S')(SM_i=SM'_i \hat{P} Dom(R'_i)=Dom(R_i) \hat{E} \{op[i+1]\})$)

Analogno, kaže se da se kompletan skup S' stanja $S'_i = (SM'_i, R'_i)$ može dobiti pomeranjem operacije $op[i]$ iz izvorišnog klastera Cs nadole u odredišni klaster Cd , u oznaci: $movedown(Cs, op[i])$, od skupa S , akko (po def.) važi:

- 1° $(\hat{S}_i \hat{I} S)(\hat{S}'_i \hat{I} S')(SM_i = SM'_i \hat{U} S_i \hat{I} Cs \hat{U} S'_i \hat{I} Cd \hat{P} R_i = R'_i)$
- 2° $(\hat{S}_i \hat{I} Cs)(op[i] \hat{I} Dom(R_i))$
- 3° $(\hat{S}_i \hat{I} Cs)(\hat{S}'_i \hat{I} S')(SM_i = SM'_i \hat{P} Dom(R'_i) = Dom(R_i) \setminus \{op[i]\})$
- 4° $(\hat{S}_i \hat{I} Cs)(\hat{S}'_i \hat{I} S')(SM_i = SM'_i \hat{P} Dom(R'_i) = Dom(R_i) \hat{E} \{op[i-1]\})$
- 5° $(\hat{S}_i \hat{I} Cd)(\hat{S}'_i \hat{I} S')(SM_i = SM'_i \hat{P} Dom(R'_i) = Dom(R_i) \hat{E} \{op[i-1]\})$

□

Prvi stav u definiciji predstavlja Linjenicu da se pri pomeranju operacije rasporedi stanja koja se ne nalaze ni u izvorišnom ni u odredišnom klasteru ne menjaju. Drugi stav predstavlja zahtev da se operacija koja se pomera nalazi u svim stanjima klastera iz kog se pomera. Treći stav zahteva da je operacija koja se pomera slobodna na vrhu, odnosno dnu. Četvrti stav navodi da se stanja klastera iz kog se operacija pomera menjaju tako da se iz rasporeda izbacuje operacija koja se pomera. Poslednji stav navodi da se stanja klastera u koji se operacija pomera menjaju tako da se u rasporedu dodaje pomerena operacija, uz odgovarajuću promenu indeksa.

6.2.5. Definicija PSP modela

Definicija 17: (PSP model) Uređeni par $PSP = (PM, S)$, gde je PM neka predikatska matrica, a S kompletan skup stanja ove matrice, naziva se PSP modelom. □

Efekat pomeranja operacija bio je uveden pojmom ekvivalencije dva PSP modela: ako se jedan PSP model može dobiti pomeranjem operacija iz drugog modela, onda se ovi modeli smatraju ekvivalentnim.

Definicija 18: (Ekvivalentni PSP model) Dva PSP modela, $PSP1 = (PM1, S1)$ i $PSP2 = (PM2, S2)$ su ekvivalentni akko (po def.) je $PM1 = PM2$ i $S2$ se može dobiti od $S1$ pomoću konačnog niza pomeranja operacija. □

Ova definicija nije sasvim precizna, u smislu da je preskočeno definisanje pojma dobijanja skupa stanja pomoću konačnog niza pomeranja operacija (ranije je definisan efekat samo jednog pomeranja). Međutim, sasvim je očišćen način na koji bi se i ovaj pojam definisao, a mi ga izostavljamo radi konciznosti. Jednostavno se pokazuje da je ovako definisana relacija ekvivalencije, to se ovde izostavlja. Ova relacija ekvivalencije deli skup PSP modela na klase ekvivalencije. Svaka klasa ekvivalencije zapravo predstavlja skup usputnih rasporeda pri pomeranju operacija, odnosno korake u procesu optimizacije koda neke petlje pomoću predikatske matrice.

6.2.6. Definicija proširenja PSP modela

Sada ćemo definisati pojmove koji će biti potrebni u formulaciji i dokazu svojstva te nje ka optimalnom. To su pojmovi proširene predikatske matrice i proširenog PSP modela.

Definicija 19: (Proširena predikatska matrica) Neka je PM predikatska matrica sa skupom indeksa Z' i skupom predikata P . Predikatska matrica $PM' \supset PM$, sa skupom indeksa Z'' i skupom predikata P' , naziva se proširenom predikatskom matricom (engl. extended predicate matrix, predlog) matrice PM akko (po def.) važi:

- 1° $P = P'$
- 2° $Z' \hat{I} Z''$
- 3° $(\hat{p} \hat{I} P)(\hat{i} \hat{I} Z')(PM(p, i) \supset PM'(p, i) \supset b)$

Pri tom, akko je leva ivica PM jednaka levoj ivici PM' , PM' se naziva desnim proširenjem. Analogno, akko je desna ivica PM jednaka desnoj ivici PM' , PM' se naziva levim proširenjem. Akko

nijedno od ta dva nije tačno, PM' se naziva obostranim proširenjem matrice PM . Matrica PM se naziva osnovnom ili polaznom matricom proširenja. \square

Treba primetiti da se u ovoj definiciji dozvoljava da je skup indeksa isti za osnovnu i proširenu matricu. Pri tom se, praktično, u proširenoj matrici neki b -element osnovne matrice "pretvara" u ne- b element. Potpuno analogno se definiše i pojam proširene matrice stanja.

Definicija 20: (Proširena matrica stanja) Neka je PM predikatska matrica i PM' njena proširena matrica. Neka je SM neka matrica stanja za PM , a SM' neka matrica stanja za PM' . Matrica SM' naziva se proširenom matricom stanja (engl. extended state matrix, predlog) matrice SM ako (po def.) važi:

$$(\forall p \in P)(\forall i \in Z)(SM(p,i) \rightarrow SM'(p,i) = SM(p,i))$$

Pri tom, ako je PM' levo proširenje matrice PM , SM' se naziva levim proširenjem matrice SM . Analogno, ako je PM' desno proširenje matrice PM , SM' se naziva desnim proširenjem matrice SM . \square

Va enje sledećeg je očigledno:

Lema 3: (O stanjima proširene predikatske matrice) Kompletan skup stanja proširene predikatske matrice PM' se surjektivno preslikava na kompletan skup stanja osnovne predikatske matrice PM tako da svakom stanju iz kodomena odgovara isti broj (to je neki stepen broja 2) stanja iz domena. Ovo preslikavanje je određeno time da se nekom stanju S' sa proširenom matricom stanja SM' pridružuje ono stanje S osnovne matrice sa matricom stanja SM , tako da je SM' proširena matrica stanja matrice SM . \square

Ovakvo surjektivno preslikavanje omogućava da se formuliše sledeća definicija:

Definicija 21: (Prošireni PSP model) Neka su $PSP1=(PM1,S1)$ i $PSP2=(PM2,S2)$ dva PSP modela. Model $PSP2$ se naziva proširenjem (ili proširenim modelom) modela $PSP1$ ako (po def.) je $PM2$ proširena predikatska matrica matrice $PM1$ i postoji model $PSP2'$ ekvivalentan modelu $PSP2$ takav da je raspored svakog stanja S' iz skupa stanja modela $PSP2'$ jednak rasporedu odgovarajućeg (prema prethodno navedenoj surjekciji) stanja S iz polaznog modela PSP .

Pri tom, ako je $PM2$ levo proširenje matrice $PM1$, $PSP2$ se naziva levim proširenjem modela $PSP1$. Analogno, ako je $PM2$ desno proširenje matrice $PM1$, $PSP2$ se naziva desnim proširenjem modela $PSP1$. \square

Smisao ove definicije je upravo taj da se prošireni model dobija tako što se proširena predikatska matrica, i iz svakog stanja osnovnog modela generiše potreban broj stanja (neki stepen broja 2) proširenog modela tako da svi oni imaju isti raspored kao i ovo polazno stanje.

6.2.7. Neka osnovna svojstva PSP modela

Najpre ćemo definisati pojam sekvence izvršavanja neke petlje, a zatim dokazati važno svojstvo koje je intuitivno jasno: u svakom PSP modelu neka beskonačna staza (put) u grafu prelaza zapravo predstavlja određenu beskonačnu sekvencu izvršavanja petlje predstavljene tim PSP modelom.

Definicija 22: (Sekvence i podsekvence izvršavanja) Neka je P skup predikata tela date petlje²⁰ i neka je PI skup svih preslikavanja $P \rightarrow \{0,1\}$. Preslikavanje $S:Z \rightarrow PI$, gde je Z skup celih brojeva, naziva se beskonačnom sekvencom izvršavanja date petlje. Element $S(i)(p)$ biće kraće označavan sa $S(i,p)$.

Neka je Z' konačan podskup skupa Z takav da važi:

$$(\forall i,j \in Z')(\forall k \in Z)(i \neq j \rightarrow S(i,p) \neq S(j,p))$$

²⁰Zapravo skup uslovnih operacija tela polazne petlje.

Preslikavanje $s:Z' \rightarrow PI$ naziva se konačnom sekvencom izvršavanja date petlje. Konačna sekvenca se ponekad označavati kao uređeni niz elemenata skupa PI na sledeći način: $PI1 \rightarrow PI2 \rightarrow PI3 \rightarrow \dots \rightarrow PIk$.

Neka je s neka konačna sekvenca sa domenom Z' , a S neka beskonačna sekvenca. Akko (po def.) postoji neko $k \in \mathbb{N}$, takvo da za svako $i \in \mathbb{N}$ važi da je $S(i+k)=s(i)$, onda se kaže da je s konačna podsekvenca sekvence S .

Neka su $s1$ i $s2$ dve konačne sekvence sa domenima $Z1'$ i $Z2'$. Akko (po def.) postoji neko $k \in \mathbb{N}$, takvo da za svako $i \in Z1'$ važi da je $i+k \in Z2'$ i da je $s1(i)=s2(i+k)$, onda se kaže da je $s1$ podsekvenca sekvence $s2$. \square

Definicija 23: (Jednakost sekvenci izvršavanja) Dve beskonačne sekvence izvršavanja $s1$ i $s2$ su jednake akko (po def.) postoji neko $k \in \mathbb{N}$ takvo da za svako $i \in \mathbb{N}$ važi $s1(i+k)=s2(i)$.

Dve konačne sekvence $s1$ i $s2$ su jednake akko (po def.) važi da je $s1$ podsekvenca $s2$ i $s2$ podsekvenca $s1$. \square

Definicija 24: (Jednakost staza u grafu prelaza) Dve beskonačne staze (puta) u grafu prelaza nekog PSP modela $P1=(...,S1_{-1}, S1_0, S1_1,...)$ i $P2=(...,S2_{-1}, S2_0, S2_1,...)$ su jednake akko (po def.) postoji neko $k \in \mathbb{N}$ takvo da za svako $i \in \mathbb{N}$ važi $S1_{i+k}=S2_i$. \square

Bez dokaza navodimo tvrđenje da su ovako definisane relacije jednakosti sekvenci izvršavanja i staza u grafu - relacije ekvivalencije.

Sada definišiti postavljen pojam *bijekcije izvršavanja* (engl. *execution bijection*, predlog), koja predstavlja formalizam koji dozvoljava da se svakoj beskonačnoj stazi u grafu prelaza PSP modela pridruži (bijektivno) neka sekvenca izvršavanja.

Definicija 25: (Bijekcija izvršavanja) Neka je skup P skup predikata i SA skup svih beskonačnih sekvenci izvršavanja nad skupom P i PA skup svih beskonačnih staza (puteva) u grafu prelaza nekog PSP modela sa predikatskom matricom nad skupom P . Bijektivno preslikavanje $E:SA \rightarrow PA$ naziva se bijekcija izvršavanja, akko (po def.) zadovoljava sledeće svojstvo: za svaku sekvencu S iz SA i odgovarajuću stazu u grafu $P=E(S)$ iz PA postoji bijekcija f između elemenata S i elemenata P takva da važi:

$$("i,j \in \mathbb{N} (f(S(i))=P(j) \wedge f(S(i+1))=P(j+1)) \text{ i još}$$

$$("i,j \in \mathbb{N} ("p \in \mathbb{N} (f(S(i))=P(j) \wedge S(i,p)=SMj(p,0)))$$

gde je SMj matrica stanja $P(j)$. Bijekcija f biće nazivana internom bijekcijom bijekcije izvršavanja E . \square

Treba primetiti da se u navedenim definicijama stalno koriste beskonačne sekvence izvršavanja i staze u grafu. Ovo je posledica prilagodljivosti PSP modela proizvoljnom broju (teorijski beskonačnom) iteracija petlje, odnosno poštovanje principa periodičnosti. Dokažemo najpre jedno pomoćno tvrđenje:

Lema 4: (O jednom svojstvu matrice stanja na stazi) Neka je $P=(...,S0,S1,...)$ beskonačna staza u grafu prelaza datog PSP modela, i SMi matrica stanja Si iz P . Tada za svako $i \in \mathbb{N}$ i svaki $n \in \mathbb{N}$ element $SMi(p,j)$ važi sa je $SMi(p,j)=SMi+j(p,0)$.

Dokaz: Neka je Si posmatrano, bilo koje stanje iz P , i neka je $j \geq 0$. Posmatrajmo n -b element $SMi+j(p,0)$. Po definiciji sledbeničke veze između stanja, ako je $SMi+j-1(p,1)$ n -b element, kako on sigurno nije na levoj ivici (jer postoji n -b kolona sa indeksom 0), onda je $SMi+j-1(p,1)=SMi+j(p,0)$. Slično, ako je $SMi+j-2(p,2)$ n -b element, onda je $SMi+j-2(p,2)=SMi+j-1(p,1)=SMi+j(p,0)$. Ako se postupak nastavi i dalje, zbog svojstva da su indeksi susedni celi brojevi, ako je $SMi+j-j(p,j)$ n -b element (iz čega sledi i da su $SMi(p,1), \dots, SMi(p,j-1)$ n -b elementi), onda je u sledbeničkoj matrici $SMi(p,j)=SMi+1(p,j-1)=SMi+j(p,0)$. Dokaz za $j < 0$ se sprovodi analogno, povećanjem indeksa u SMi . \square

Ovime je dokazano i op tije svojstvo da za neka dva stanja na stazi (konačnoj ili beskonačnoj), odnosno njihove matrice stanja, va i $SM_i(p,j)=SM_{i+k}(p,j-k)$, pri čemu su oba ne- b elementi.

Teorema 2: (O postojanju bijekcije izvršavanja) Za svaki PSP model postoji tačno jedna bijekcija izvršavanja.

Dokaz: Najpre ćemo konstruisati preslikavanje E , zatim dokazati da je ono bijekcija, i najzad da je jedinstveno. Neka je $S \in SA$ neka beskonačna sekvenca izvršavanja nad skupom predikata P . Konstruisaćemo beskonačnu stazu $P \in PA$, $P=(...,S_{-1},S_0,S_1,...)$ koja predstavlja $E(S)$ na sledećem način. Neka je S_i ono stanje datog PSP modela koje ima matricu stanja takvu da je svaki ne- b element $SM_i(p,j)$ jednak $S(i+j,p)$. Pokačimo da je niz ovako izabranih stanja uopšte staza u grafu. Kako je S_i određeno matricom SM_i takvom da je svaki ne- b element van leve ivice $SM_i(p,j)$ jednak $S(i+j,p)$, onda je u matrici stanja S_{i+1} element $SM_{i+1}(p,j-1) \neq b$, zapravo jednak $S(i+1+j-1,p)=S(i+j,p)=SM_i(p,j)$. Time je pokazano da između čvorova koji predstavljaju S_i i S_{i+1} postoji prelaz, odnosno da je P staza. Pri tome postoji jedinstveno S_i za dato $S(i)$ zbog naćina konstrukcije.

Iz naćina konstruisanja staze P očigledno je da je preslikavanje $f(S(i))=S_i$ zapravo bijekcija koja zadovoljava svojstva iz definicije bijekcije izvršavanja. Takoć, očigledno je da za svako S postoji jedinstveno $E(S)$, tako da je E preslikavanje. Da bi se dokazalo da je E bijekcija, treba pokazati da je E preslikavanje "1-1" i "na".

Dokačimo da je E preslikavanje "na". Potrebno je dokazati da za svaku stazu $P \in PA$ postoji sekvenca S takva da je $P=E(S)$. Neka je S takva sekvenca da je $S(i,p)$ jednako $SM_i(p,0)$, gde je SM_i matrica stanja S_i iz P . Tada je matrica stanja za $f(S(i))$, prema gornjoj konstrukciji, matrica SM_i takva da je ne- b element $SM_i(p,j)=S(i+j,p)=SM_{i+j}(p,0)=SM_i(p,j)$, za svako j , prema Lemi 4. Ovime je pokazano da je E preslikavanje "na", a pri tom i definisano preslikavanje E^{-1} kada se pokače da je E bijekcija: $E^{-1}(P)$ se dobija kao niz kolona sa indeksom 0 matrica stanja koja ćine stazu u grafu.

Dokačimo sada da je E preslikavanje "1-1". Potrebno je dokazati da ako je $P_1=P_2$, i ako je $E(S_1)=P_1$ i $E(S_2)=P_2$, onda je $S_1=S_2$. Kako je $P_1=P_2$, postoji neko $k \in \mathbb{Z}$ takvo da je za svako $i \in \mathbb{Z}$ $P_1(i)=P_2(i+k)$, odnosno $SM_{1,i}=SM_{2,i+k}$. Kako je, prema definiciji, $S_1(i,p)=SM_{1,i}(p,0)=SM_{2,i+k}(p,0)=S_2(i+k,p)$ za svako $p \in P$, sledi da je za posmatrano k i svako i : $S_1(i)=S_2(i+k)$, odakle sledi da je $S_1=S_2$. Ovime je dokazano da je E bijekcija.

Ostaje joć da se pokače da je E jedinstvena. Neka je S bilo koja sekvenca izvršavanja. Posmatrajmo bilo koju bijekciju E sa internom bijekcijom f . Tada postoji neko $k \in \mathbb{Z}$ takvo da je za svako $i \in \mathbb{Z}$ $f(S(i))=P(i+k)$, to se jednostavno dokazuje na osnovu prvog uslova za f iz definicije bijekcije izvršavanja. Neka je SM_{i+k} matrica stanja $P(i+k)$. Po definiciji bijekcije izvršavanja je $S(i,p)=SM_{i+k}(p,0)$, za svako $p \in P$. Prema Lemi 4 je $S(i+j,p)=SM_{i+j+k}(p,0)=SM_{i+k}(p,j)$ za svako j takvo da je $SM(p,j)$ ne- b element. Dakle, SM_{i+k} je potpuno i jednoznaćno određena sekvencom S . Kako ovo vaći za bilo koju E , jedina neodrećnost je konstanta k , odakle se jednostavno pokazuje da bilo koja bijekcija E preslikava bilo koje S u neku od jednakih staza P . Time je dokazano da je E jedinstvena. \square

Isti pojam bijekcije izvršavanja uvećemo i za PSP modele mećusobno.

Definicija 26: (Bijekcija izvršavanja) Neka su $PSP1$ i $PSP2$ dva modela nad istim skupom predikata P . Neka su $PA1$ i $PA2$ skupovi svih beskonaćnih staza u grafovima prelaza ova dva modela, redom. Bijektivno preslikavanje $E:PA1 \rightarrow PA2$ naziva se bijekcija izvršavanja, akko (po def.) zadovoljava sledeće svojstvo: za svaku stazu $P1 \in PA1$ i odgovarajuću stazu $P2=E(P1)$ iz $PA2$ postoji bijekcija f između elemenata $P1$ i elemenata $P2$ takva da vaći:

$$("i,j \in \mathbb{Z})(f(P1(i))=P2(j) \wedge f(P1(i+1))=P2(j+1)) \text{ i još}$$

$$("i,j \in \mathbb{Z})(("P1 \in P)(f(P1(i))=P2(j)) \rightarrow SM_{1,i}(p,0)=SM_{2,j}(p,0))$$

gde je $SM1i$ matrica stanja $P1(i)$, a $SM2j$ matrica stanja $P2(j)$. Bijekcija f biće nazivana internom bijekcijom stanja bijekcije izvršavanja E . \square

Teorema 3: (O postojanju bijekcije izvršavanja) Između svaka dva PSP modela $PSP1$ i $PSP2$ nad istim skupom predikata P postoji tačno jedna bijekcija izvršavanja.

Dokaz: Posmatrajmo skup SA svih beskonačnih sekvenci izvršavanja nad skupom P . Kako između SA i $PA1$ postoji tačno jedna bijekcija izvršavanja $E1$, a isto tako i između SA i $PA2$ bijekcija $E2$, bijekcija $E(P1)=E2(E1^{-1}(P1))$ je tražena bijekcija. Lako je dokazati da je ovo jedina bijekcija koja zadovoljava uslove iz definicije, posmatranjem odgovarajućih sekvenci izvršavanja za staze $P1$ i $P2$, koje moraju biti jednake što sledi iz uslova definicije bijekcije E . \square

Kako su neki PSP model i njegov prošireni model specijalni slučaj modela nad istim skupom predikata P , onda između njih postoji tačno jedna bijekcija izvršavanja.

6.2.8. Svojstva proširenja PSP modela

Ovde će biti definisani neki pojmovi i pokazana neka svojstva koja su potrebna u postavci i dokazu te nje kao optimalnom PSP modela. To su najpre pojam dužine izvršavanja neke staze u grafu prelaza, pojam združenog rasporeda neke staze u grafu prelaza, pojam semantičke ekvivalencije dva PSP modela i svojstvo semantičke ekvivalencije proširenja PSP modela.

Definicija 27: (Dužina izvršavanja staze) Neka je G graf prelaza nekog PSP modela, i neka je $p=(S0,S1,...,Sk)$ neka staza (put) konačne dužine u ovom grafu G . Dužinom izvršavanja staze (puta) p naziva se broj:

$$\text{Len}(p) = \sum_{i=0}^k \text{Len}(Si)$$

pri čemu oznaka $\text{Len}(Si)$ predstavlja dužinu rasporeda stanja Si . \square

Definicija 28: (Združeni raspored staze) Neka je G graf prelaza nekog PSP modela, i neka je $p=(S0,S1,...,Sk)$ neka staza (put) konačne dužine u ovom grafu G . Združeni raspored (engl. composite schedule, predlog) je minimalni raspored čiji se skup instanci operacija sastoji od instanci operacija iz $S0$, i instanci operacija iz svakog Si , $i=1,...,k$, pri čemu je svaka instanca $op[j]$ iz Si promenjena u $op[j+i]$. Oznaka: $CR(p)$. $\text{Len}(CR(p))$ će biti nazivano i dužinom kritičnog puta (engl. critical path) staze p . \square

Definicija 29: (Skup instanci operacija na stazi) Neka je $P=(...,S0,S1,...)$ beskonačna staza (put) u grafu prelaza nekog PSP modela. Skupom instanci operacija staze P sa pomerajem k , gde je k neki ceo broj, naziva se skup sastavljen od instanci operacija iz svakog Si , $i \in \mathbb{Z}$, pri čemu je svaka instanca $op[j]$ iz Si promenjena u $op[j+i+k]$. \square

Definicija 30: (Semantička ekvivalencija modela) Dva PSP modela $PSP1$ i $PSP2$ su semantički ekvivalentna akko (po def.) postoji bijekcija izvršavanja E između $PSP1$ i $PSP2$, pri čemu za svaku beskonačnu stazu P u grafu prelaza $PSP1$ postoji neko $k \in \mathbb{Z}$ takvo da je skup instanci operacija staze P sa pomerajem k jednak skupu instanci operacija staze $E(P)$ sa pomerajem 0 . Oznaka: $SE(PSP1,PSP2)$. \square

Bez dokaza navodimo tvrdnje da je ovako definisana relacija SE relacija ekvivalencije (dokaz nije teško izvesti). Dokazano sada najpre da su dva ekvivalentna PSP modela istovremeno i semantički ekvivalentna, a zatim i da se proširenjem dobija semantički ekvivalentan model.

Lema 5: (O semantičkoj ekvivalenciji pri pomeranju) Ekvivalentni PSP modeli su i semantički ekvivalentni.

Dokaz: Dokazano najpre da se jednim pomeranjem zadržava semantička ekvivalentnost. Pretpostavimo da se od modela $PSP1$ dobija model $PSP2$ jednim pomeranjem operacije $movedown(Cs, op[i])$. Pokazano da su tada $PSP1$ i $PSP2$ semantički ekvivalentni. Bijekcija E koja preslikava staze grafa $G1$ u staze grafa $G2$ je trivijalna (praktično predstavlja identitet), jer su matrice stanja oba modela iste. Za sve staze oba grafa koje ne prolaze kroz stanja iz klastera Cs i Cd važi da imaju iste skupove instanci operacija, jer se raspoređuje stanja na tim stazama ne razlikuju.

Posmatrajmo stazu $P1$ grafa $G1$ koja prolazi kroz jedno stanje $S1s$ (i odgovarajuću stazu $P2$ koja prolazi kroz stanje $S2s$ grafa $G2$ sa istom matricom stanja) iz Cs i jedno stanje $S1d$ (i odgovarajuću $S2d$) iz Cd . Kako je staza beskonačna, ovakvi $S1s$ i $S1d$ sigurno oba pripadaju stazi. Domeni rasporeda stanja $S1s$ i $S2s$ se razlikuju po tome što prvi sadrži $op[i]$, a drugi ga ne sadrži. Domeni rasporeda stanja $S1d$ i $S2d$ se razlikuju po tome što drugi sadrži $op[i-1]$, a prvi ga ne sadrži. Domeni ostalih stanja van dva navedena klastera na stazama se ne razlikuju. U skupu instanci operacija staze $P1$ postoje $op[i+k]$, gde je k redni broj stanja $S1s$ (i $S2s$) u stazi, dok u skupu instanci operacija staze $P2$ postoji $op[i-1+k+1]=op[i+k]$, jer je redni broj $S1d$ (i $S2d$) jednak $k+1$, i to za svako pojavljivanje ovih stanja na stazi. Sve ostale operacije u domenima bitne. Prema tome, skupovi instanci operacija na stazama $P1$ i $P2$ su bitni jednaki. Kako ovo važi za sve staze koje prolaze kroz Cs i Cd , tvrdnje je dokazano za sve staze grafa.

Potpuno analogno se izvodi dokaz za pomeranje nagore. Navedeni dokaz se odnosi na jedno pomeranje. Kako su relacije ekvivalencije i semantičke ekvivalencije PSP modela obe refleksivne, simetrične i tranzitivne, tvrdnje važi za svaki konačan niz pomeranja, čime je lema dokazana. \square

Teorema 4: (O semantičkoj ekvivalenciji proširenja PSP modela) Dati model $PSP1$ i njegov prošireni model $PSP2$ su semantički ekvivalentni.

Dokaz: Dokazano samo da su semantički ekvivalentni model $PSP1$ i model koji se dobija proširenjem predikatke matrice $PM1$, uz iste rasporede po stanjima (kao u definiciji proširenog PSP modela); ovaj drugi model označimo sa PSP' . Kako su modeli PSP' i $PSP2$ ekvivalentni prema definiciji proširenja, a prema prethodnoj lemi time i semantički ekvivalentni, zbog tranzitivnosti ove relacije to je dovoljno za dokaz.

Kako su raspoređuje stanja modela PSP i odgovarajuća stanja iz PSP' (prema surjekciji iz definicije proširenog PSP modela) identični, potrebno je samo pronaći bijekciju koja preslikava beskonačne staze grafa $G1$ modela $PSP1$ na staze grafa $G2$ modela PSP' , tako da između \mathcal{L} -vorova staze iz $G1$ i \mathcal{L} -vorova staze iz $G2$ postoji bijekcija f koja preslikava stanje $S1$ na stanje $S2$ sa proširenom matricom stanja. To je upravo bijekcija izvršavanja E , a njeno svojstvo da preslikava matricu stanja u proširenu matricu stanja je jednostavno videti posmatranjem odgovarajuće sekvence izvršavanja, što je formalno pokazano u narednom poglavlju. \square

6.3. Definicija i dokaz svojstva težnje ka optimalnom

Sada je formalno biti definisan pojam težnje nekog modela ka optimalnom u ovom slučaju, a zatim i dokazano to svojstvo za predloženi PSP model. Najpre je opisno biti objašnjen povod za nalet na koji je ovaj pojam definisan.

6.3.1. Intuitivna postavka pojma i dokaza svojstva težnje ka optimalnom

Za svako konkretno izvršavanje neke petlje može se formirati graf zavisnosti po podacima koji opisuje tok podataka (engl. *data flow*) za to konkretno izvršavanje, kao što je to učinjeno u poglavlju 3.1.2.²¹ Vremenski optimalno izvršavanje petlje prema Definiciji 1 pretpostavlja da je

²¹To je upravo združen raspored definisan u prethodnom odeljku.

du ina trajanja izvršavanja jednaka dužini kritičnog puta u ovom grafu. Mi smo ovde izvršavanje posmatrati u kontekstu softverske protokolnosti prema Definiciji 2. To nam nalaže da dati graf koji predstavlja neko konkretno izvršavanje podelimo na onoliko podgrafova (u smislu njegovog topološkog redosleda), koliko iteracija sadrži to izvršavanje u polaznoj petlji. Tako smo iteracijom transformisane petlje smatrati jedan podgraf tog grafa. Svaka posebna podela ovog grafa zapravo predstavlja jednu transformaciju polazne petlje. Uz ovakve pretpostavke, iznećemo jednu tvrdnju koju nećemo dokazivati, jer ona predstavlja samo usputni zaključak koji nije bitan za dalju analizu PSP modela.

Teorema 5: *(O vremenskoj optimalnosti izvršavanja petlje) Dato izvršavanje petlje je vremenski optimalno akko je svaki deo svakog kritičnog puta celog grafa izvršavanja, koji pripada podgrafu jedne iteracije, ujedno i kritičan put tog podgrafova iteracije. □*

PSP model predstavlja model transformisane petlje. Za svako konkretno izvršavanje petlje postoji odgovarajuća konačna staza u grafu prelaza PSP modela koja predstavlja to izvršavanje. Vremenska optimalnost ovog izvršavanja ekvivalentna je Linjenici da je svaki deo kritičnog puta po celoj stazi, koji pripada jednom stanju, ujedno i kritičan put rasporeda tog stanja. Prema tome, ukoliko dati PSP model ne predstavlja petlju transformisanu tako da je svako izvršavanje optimalno, to bi značilo da ovaj PSP model predstavlja optimalno rešenje, onda postoji staza u grafu prelaza koja ne zadovoljava navedeno svojstvo o delovima kritičnog puta.

Pretpostavimo da rešenje predstavljeno datim PSP modelom nije optimalno za svako izvršavanje i posmatrajmo jednu stazu u grafu prelaza koja nije optimalna u navedenom značenju. Ako se stanja koja čine ovu stazu posmatraju kao niz rasporeda i ti rasporedi ujedine u jedinstven graf zavisnosti po podacima, a zatim se taj graf podeli na isto onoliko delova koliko je stanja u nizu tako da se zadovolji kriterijum o optimalnosti ovih delova prema celom grafu (svaki deo kritičnog puta grafa je kritičan put u svom delu grafa), i ovakva podela smatra "ciljnim" rasporedom po stanjima, onda ovakva podela ukazuje na pomeranja operacija koja treba izvršiti između stanja u nizu da bi se dobilo optimalno izvršavanje posmatrane staze.

Prema tome, može se definisati konačan niz pomeranja operacija između stanja susednih u nizu (na datoj stazi) koji dovodi do optimalnog izvršavanja te staze. Međutim, u datom PSP modelu neka od tih pomeranja može da nije moguće izvršiti, jer posmatrana operacija ne postoji u svim stanjima klastera iz koga je treba pomeriti.

Pojam rešenja ka optimalnom nekog modela zasniva se upravo na ovom razmatranju: smatraćemo da model te i ka optimalnom ako za svako izvršavanje koje nije optimalno model nudi efektivan postupak kojim se to izvršavanje optimizuje, a pri tome ostala izvršavanja ne produžavaju. Kod PSP modela, taj efektivan postupak biće zasnovan na proirenju PSP modela i konačnom nizu pomeranja operacija u proirenom modelu koja dovode do optimalnosti odgovarajućestaze u grafu prelaza, dok se ostale staze u grafu koje su disjunktne sa ovom neće menjati.

Neka je niz stanja u grafu prelaza PSP modela koji ne daje optimalno izvršavanje niz S_1, S_2, \dots, S_k . Pretpostavimo, radi jednostavnosti ovog preliminarog razmatranja, da skup predikata sadrži samo jedan predikat p , tako da matrice imaju samo jednu vrstu bez b -elemenata²². Opet radi jednostavnosti i konkretnosti objašnjenja, pretpostavimo da je skup indeksa $Z' = \{-1, 0, 1\}$. Neka su matrice stanja navedenog niza stanja redom:

$$SM_1 = [x_{1_{-1}} \ x_{1_0} \ x_{1_1}]$$

$$SM_2 = [x_{2_{-1}} \ x_{2_0} \ x_{2_1}]$$

...

$$SM_k = [x_{k_{-1}} \ x_{k_0} \ x_{k_1}]$$

²²Za predikatsku matricu sa jednom vrstom, prema definiciji predikatske matrice, uvek važi da nema b -elemenata.

gde je $x_i \in \{0,1\}$. Iz definicije prelaza stanja sledi da je:

$$\begin{aligned}x_{1_0} &= x_{2_{-1}} \\ x_{1_1} &= x_{2_0} = x_{3_{-1}} \\ x_{2_1} &= x_{3_0} = x_{4_{-1}} \\ &\dots\end{aligned}$$

Prema tome, niz ishoda predikata koji opisuju ovo izvršavanje može se predstaviti u obliku niza:

$$[x_{1_{-1}} x_{1_0} x_{2_0} \dots x_{k_0} x_{k_1}]$$

Intuitivno je ranije pokazano da se bolji raspored može dobiti proirenjem PSP modela. Logično proirenje matrice za ovaj slučaj izvršavanja je taj da se matrica proiri tako da se postigne potrebna "dalekovidost" i "istorija" ishoda predikata: da se u stanju S'_1 "znaju" unapred ishodi svih instanci predikata zaključno sa x_{k_1} , i slično da se u S'_k "znaju" ishodi svih prethodnih instanci predikata zaključno sa $x_{1_{-1}}$. Na ovaj način, skup indeksa proirenog modela biće $Z' = \{-k, \dots, -1, 0, 1, \dots, k\}$. Posmatrajmo skup svih stanja datih pojedinačnim matricama iz proirenog modela, pri čemu oznaka x predstavlja bilo koju vrednost 0 ili 1:

$$\begin{aligned}SM'_1 &= [x \quad x \quad \dots \quad x \quad x_{1_{-1}} \quad x_{1_0} \quad x_{2_0} \quad \dots \quad x_{k-1_0} \quad x_{k_0} \quad x_{k_1}] \\ SM'_2 &= [x \quad x \quad \dots \quad x_{1_{-1}} \quad x_{1_0} \quad x_{2_0} \quad x_{3_0} \quad \dots \quad x_{k_0} \quad x_{k_1} \quad x] \\ &\dots \\ SM'_k &= [x_{1_{-1}} \quad x_{1_0} \quad \dots \quad x_{k-2_0} \quad x_{k-1_0} \quad x_{k_0} \quad x_{k_1} \quad \dots \quad x \quad x \quad x]\end{aligned}$$

U skupu stanja određenih sa SM'_i su stanja koja odgovaraju stanju S_i , $i=1, \dots, k$, to znači da su sva stanja određena sa SM'_i zapravo dobijena "proirenjem" stanja S_i , pa su njihovi rasporedi jednaki rasporedu stanja S_i . Zbog načina konstruisanja SM'_i , svako pomeranje koje je bilo potrebno izvršiti između neka dva stanja S_i i S_{i+1} može se izvršiti između svih stanja određenih sa SM'_i i SM'_{i+1} , jer su ova stanja povezana sledbeničkim vezama. Pri tome, sva stanja "nastala" proirenjem nekog stanja S koje se ne nalazi u datom nizu, ne nalaze se u skupu stanja određenih pomoćnim datih SM'_i . Na taj način je obezbeđeno da se ostale disjunktne staze izvršavanja ne menjaju.

Međutim, ovakav postupak još uvek ne obezbeđuje postizanje optimalnog rasporeda za datu sekvencu izvršavanja. Posmatrajmo ovaj problem na primeru izvršavanja datom sledećom sekvencom matrica stanja:

$$[1 \ 0 \ 1] \rightarrow [0 \ 1 \ 0] \rightarrow [1 \ 0 \ 1]$$

Označimo stanje određeno matricom $[1 \ 0 \ 1]$ sa $S1$, a matricom $[0 \ 1 \ 0]$ sa $S2$. Ako se primeni opisano proirivanje, dobijaju se skupovi stanja predstavljeni sledećom tabelom (elementi koji čine sekvencu ishoda su podvučeni radi lakše orijentacije):

	C1	C2	C3
R1	[00 <u>1</u> 0101]	[0101010]	[1010100]
R2	[0110101]	[0101011]	[1010101]
R3	[1010101]	[1101010]	[1010110]
R4	[1110101]	[1101011]	[1010111]

Prelazi izgledaju ovako: iz $(C1, R1)$ i $(C1, R3)$ prelazi se u $(C2, R1)$ i $(C2, R2)$; iz $(C1, R2)$ i $(C1, R4)$ prelazi se u $(C2, R3)$ i $(C2, R4)$; iz $(C2, R1)$ i $(C2, R3)$ prelazi se u $(C3, R1)$ i $(C3, R2)$; konačno, iz $(C2, R2)$ i $(C2, R4)$ prelazi se u $(C3, R3)$ i $(C3, R4)$.

Sada se vidi ideja predložene rešenja. Za sve ostale staze koje su disjunktne sa stazom $S1 \rightarrow S2 \rightarrow S1$, ne postoje stanja koja su generisana na ovaj način u proirenom modelu, a nalaze se u datom skupu stanja, pa se pomeranjem operacija njihovi rasporedi ne menjaju. Drugo, sva stanja u

C1 "nastala" su od stanja $S1$ (jer imaju isti središnji deo matrice stanja), pa imaju isti raspored; slično je za $C2$ i stanje $S2$, i $C3$ i stanje $S2$.

Međutim, ovo još ne obezbeđuje da se primena celog niza pomeranja operacija biti uopšte moguća, kao ni to da, ako je moguće, daje optimalni raspored duž ove staze. Naime, problem je u tome što se stanje određeno matricom $[1010101]$ pojavljuje u dve kolone date tabele ($C1$ i $C3$). To znači da ako je bilo potrebno primeniti neko pomeranje između $S1$ i $S2$ sa početka sekvence, isto pomeranje se reflektovati i na stanje $[1010101]$ iz kolone $C1$ i na isto to stanje iz kolone $C3$. Zato, ako je potrebno izvršiti neko pomeranje između $S2$ i $S1$ sa kraja sekvence, to pomeranje može da bude nemoguće zato što je raspored stanja $[1010101]$ promenjen prethodnim pomeranjem. Ako to nije slučaj, onda se u najmanju ruku može dobiti raspored stanja $[1010101]$ različit od onoga koji bi se dobio kada bi se pomeranje za to stanje izvršavalo nezavisno za dva slučaja pojavljivanja $S1$ u sekvenci. Na primer, ako se neka operacija iz $S1$ premesti nadole u $S2$, ona će biti istovremeno premećena iz svih stanja grupe $C1$ u sva stanja grupe $C2$, ali će i ta operacija biti premećena nadole iz stanja $[1010101]$ iz grupe $C3$. Analogno je za pomeranja nagore iz $S2$ u $S1$. Baš ovakvo pomeranje nagore može da doprinese da se stanje $[1010101]$ produži za operaciju koja nije predviđena na kraju sekvence (grupa $C3$).

Suština problema leži u principu periodičnosti. Naime, PSP model, kao model softverske protočnosti, prilagođen je proizvoljno dugačkom (teorijski beskonačnom) izvršavanju petlje. Zbog toga se mora postaviti princip periodičnosti. Ovaj princip dovodi do toga da se raspored mora prilagoditi svakoj beskonačnoj sekvenci izvršavanja koja u sebi sadrži datu sekvencu $S1 \rightarrow S2 \rightarrow S1$, pa i za onu beskonačnu sekvencu koja je periodična: $\dots \rightarrow S1 \rightarrow S2 \rightarrow S1 \rightarrow S2 \rightarrow S1 \rightarrow \dots$. Zbog toga postoji stanje $[1010101]$ koje predstavlja "uvod" u ovakvu periodičnu beskonačnu sekvencu, i sve promene jedne periode beskonačne sekvence moraju se odslikati na svaku njenu periodu, a time i na svaku "instancu" stanja $[1010101]$ u sekvenci.

Zbog ovoga beskonačne periodične sekvence predstavljaju poseban slučaj izvršavanja petlje koje mogu da dovedu do neoptimalnosti. Uostalom, posledica istog ovog principa periodičnosti je i svojstvo petlje bez uslovnih grananja opisano u 3.1.1. koje ne dozvoljava teorijski optimalno izvršavanje u kontekstu softverske protočnosti za petlje čiji količnik zbira dužina i zbira iteracionih distanci po kritičnom zatvorenom putu u grafu petlje nije ceo broj. Pokazano u sledećem odeljku i formalno da se opisani slučaj pojavljivanja istog stanja u proširenom modelu na višim mestima javlja upravo onda i samo onda kada posmatrana sekvenca predstavlja deo neke beskonačne periodične sekvence, pri čemu je taj deo dužine od periode te beskonačne sekvence. Ovakve konačne sekvence nazivamo *parcijalno periodičnim sekvencama* (engl. *partially periodic sequence*, predlog).

Za sada se čini da postojanje periodičnih beskonačnih sekvenci izvršavanja za koje ne može da se pronađe optimalni raspored predstavlja prirodno ograničenje paralelizma petlji sa uslovnim grananjima (kao i bez njih). Čini se da ovo prirodno ograničenje ne može da se prevaziđe tehnikom softverske protočnosti kao što je ona ovde definisana. Zbog toga ćemo mi uvesti oslabljeni uslov te nje ka optimalnom, koji će se zasnivati na sledećem razmatranju. Neka je S neka parcijalno periodična sekvenca za koju raspored u datom PSP modelu nije optimalan. Oslabljeni uslov te nje ka optimalnom zahteva da postoji efektivan postupak kojim se izvršavanje nekih sekvenci koje sadrže podsekvencu S optimizuje.

Ovaj efektivan postupak za PSP model zasniva se na "razbijanju" periodičnosti. Naime, "neke" sekvence za koje se poboljšati raspored biće neke od onih beskonačnih sekvenci koje sadrže datu sekvencu S a nisu periodične, ili jesu periodične, ali sa periodom većim od dužine sekvence S . Za to će biti potrebno da se data sekvenca S produži konačnim nizom stanja, tako da ovako produžena sekvenca nije više parcijalno periodična. Dalje će se primenjivati isti onaj postupak koji je ranije opisan.

Na primer, sekvenca $1 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 1 \rightarrow 2$ (stanja su označena samo brojevima) je parcijalno periodična, jer se završava istom sekvencom $1 \rightarrow 2$ kojom i počinje. Slično, sekvenca $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow$

$2 \rightarrow 3 \rightarrow 1$ je parcijalno periodična, kao i sekvenca $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$ ili sekvenca $1 \rightarrow 1$. Dokazano u narednom poglavlju da se svaka parcijalno periodična sekvenca može produžiti u konačnu sekvenču koja nije parcijalno periodična. Dokaz se zasniva na tome što postoji ponuđen efektivan postupak produžavanja sekvenče tako da dobijena sekvenca nije parcijalno periodična.

Ovaj postupak sastoji se u sledećem: na kraj sekvenče treba dodati onaj element e , koji je različit od elementa kojim sekvenca počinje (takav uvek postoji, jer model ima bar dva stanja). Zatim treba obezbediti, dodavanjem istog elementa e na kraj sekvenče, da se sekvenca završava sa k elemenata e , takvim da je k veće od najvećeg broja podsekvenče elemenata e u polaznoj sekvenči. Na primer, za parcijalno periodičnu sekvenču $1 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 1$, na kraj treba dodati četiri elementa 2, čime se dobija sekvenca $1 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2$ koja nije parcijalno periodična. Slično, sekvenči $1 \rightarrow 1$ treba dodati element 2, čime se opet dobija parcijalno neperiodična sekvenca $1 \rightarrow 1 \rightarrow 2$.

Primenom ranije navedenog postupka, poboljšaće se sva ona izvršavanja koja sadre problematiku sekvenču koja nije parcijalno periodična, dok preostale sekvenče, među kojima je i periodična sekvenca biti nepromenjene, pa time i eventualno neoptimalne. Na ovaj način, biće dokazano da PSP model zadovoljava oslabljeni uslov te nije ka optimalnom u svakom slučaju, dok u posebnim slučajevima zadovoljava i strogi uslov. Formalne postavke i dokazi navedenih tvrdnji biće izneseni u narednim odeljcima.

6.3.2. Definicija svojstva težnje ka optimalnom

Sledeće definicije svojstva te nije ka optimalnom su donekle neprecizne, u smislu da se do sada nije definisao pojam vremena izvršavanja. Ovo je učinjeno zato što su definicije sročene tako da budu primenjive na sve modele optimizacije koda programskih petlji sa uslovnim grananjima, a ne samo na PSP model. Definicije ovog svojstva za PSP model biće precizno navedene uz oslanjanje na pojmove koji su do sada formalno uvedeni.

Definicija 31: (Svojstvo težnje ka optimalnom) Neki model teži ka optimalnom (engl. is near-optimal) akko (po def.) za svaku petlju važi sledeće: ako postoji konačna sekvenca izvršavanja S čije izvršavanje nije vremenski optimalno, onda u modelu postoji efektivan postupak kojim se petlja transformiše tako da se smanjuje vreme izvršavanja svih konačnih sekvenči koje sadrže podsekvenču S , a vreme izvršavanja svih ostalih konačnih sekvenči koje su disjunktne sa S ne povećava. \square

Definicija 32: (Slabiji uslov težnje ka optimalnom) Neki model slabo teži ka optimalnom (engl. is weakly near-optimal, predlog) akko (po def.) za svaku petlju važi sledeće: ako postoji konačna sekvenca izvršavanja S čije izvršavanje nije vremenski optimalno, onda u modelu postoji efektivan postupak kojim se petlja transformiše tako da se smanjuje vreme izvršavanja nekih konačnih sekvenči koje sadrže podsekvenču S , a vreme izvršavanja svih ostalih konačnih sekvenči koje su disjunktne sa ovima ne povećava. \square

Ova poslednja definicija se od prethodne razlikuje praktično samo u tome što je reč "svih" zamenjena rečju "nekih", u skladu sa analizom iz prethodnog odeljka.

6.3.3. Definicija svojstva težnje PSP modela ka optimalnom

Sada ćemo postaviti tvrdnje da PSP model zadovoljava slabiji uslov te nije ka optimalnom, bez formalnog dokaza da je tumačenje iz sledeće teoreme ekvivalentno upravo iznesenoj ovoj definiciji. Ipak, ova ekvivalencija je intuitivno sasvim očitljiva.

Teorema 6: (Svojstvo težnje PSP modela ka optimalnom) PSP model nad skupom predikata P zadovoljava slabiji uslov težnje ka optimalnom sa sledećim značenjem. Ako postoji konačna sekvenca izvršavanja S nad skupom P čije izvršavanje nije vremenski optimalno, onda

postoji efektivan postupak kojim se dobija novi, semantički ekvivalentan PSP model iste petlje u kome je vreme izvršavanja nekih konačnih sekvenci koje sadrže podsekvencu S smanjeno, a vreme izvršavanja ostalih konačnih sekvenci koje su disjunktne sa njima nije povećano. \square

Zbog dokazanog postojanja bijekcije izvršavanja E , dokazane semantičke ekvivalencije pro irenog i osnovnog PSP modela, i uz korišćenje pojmova združenog rasporeda i dužine staze i rasporeda definisanih u 6.2.8., sledi sledeći teorema o te nji PSP modela ka optimalnom je ekvivalentna prethodnoj, ali je neposredno prilagođena dokazu.

Teorema 7: (Svojstvo težnje PSP modela ka optimalnom) PSP model zadovoljava slabiji uslov težnje ka optimalnom sa sledećim značenjem. Ako postoji konačna staza p u grafu prelaza datog PSP modela, takva da je $\text{Len}(p) > \text{Len}(\text{CR}(p))$, onda postoji semantički ekvivalentan model PSP' u kome važi da je $\text{Len}(p') = \text{Len}(\text{CR}(p'))$, za neke p' od konačnih staza u grafu modela PSP' koje sadrže podstaze iz skupa staza određenih kao $f(p)$ za sve beskonačne staze P koje sadrže p , pri čemu je f interna bijekcija bijekcije izvršavanja E između PSP i PSP' , dok se $\text{Len}(p'')$ za ostale p'' koje su disjunktne sa p' iz grafa modela ne menjaju. \square

Postavljamo ovde i drugu teoremu koja daje uslov da se sve sekvence izvršavanja koje sadrže neoptimalnu sekvencu optimizuju, odnosno opisuje slučajev kada PSP model zadovoljava strožiji uslov težnje ka optimalnom. Ovu teoremu dokazavamo u istom postupku dokaza prethodne teoreme, pa je zato odmah ovde navodimo. Uslov će biti u skladu sa intuitivnim razmatranjem sa početka ovog poglavlja: strožiji uslov težnje ka optimalnom zadovoljen je samo za sekvence izvršavanja koje nisu parcijalno periodične. Zbog toga je neophodno najpre definisati pojmove periodičnosti i parcijalne periodičnosti.

Definicija 33: (Periodični niz) Preslikavanje $A: \mathbb{Z} \rightarrow S$, gde je S neki skup, naziva se beskonačnim nizom A . Beskonačni niz A naziva se periodičnim akko (po def.) važi:

$$(\forall i \in \mathbb{Z})(\exists T)(A(i) = A(i+T)).$$

Najmanja od konstanti T iz definicije naziva se periodom niza A . \square

Pojmove konačnog niza, konačnog podniza beskonačnog ili konačnog niza, kao i dužine konačnog niza nećemo formalno navoditi, jer smo neke analogne definicije imali za sekvence izvršavanja.

Definicija 34: (Parcijalno periodični niz) Konačni niz A se naziva parcijalno periodičnim (engl. partially periodic, predlog) akko (po def.) postoji beskonačni periodični niz A' čiji je A podniz, pri čemu je dužina niza A veća od perioda tog niza A' . \square

Evo sada i najavljene teoreme o slučajevima kada PSP model zadovoljava strožiji uslov težnje ka optimalnom.

Teorema 8: (Strožiji uslov težnje PSP modela ka optimalnom) PSP model zadovoljava strožiji uslov težnje ka optimalnom samo u nekim slučajevima, sa sledećim značenjem. Ako postoji konačna staza p u grafu prelaza datog PSP modela koja nije parcijalno periodična, takva da je $\text{Len}(p) > \text{Len}(\text{CR}(p))$, onda postoji semantički ekvivalentan model PSP' u kome važi da je $\text{Len}(p') = \text{Len}(\text{CR}(p'))$, za svaku stazu p' iz skupa staza određenih kao $f(p)$ za neku beskonačnu stazu P koja sadrži p , pri čemu je f interna bijekcija bijekcije izvršavanja E između PSP i PSP' , dok se $\text{Len}(p'')$ za staze p'' iz grafa modela koje su disjunktne sa $f(p)$ ne menjaju. \square

6.3.4. Dokazi pomoćnih svojstava

U ovom odeljku dokazavamo neka svojstva koja su potrebna u dokazu težnje PSP modela ka optimalnom, a intuitivno su bila pokazana u 6.3.1. Najpre ćemo dokazati da se svaki parcijalno periodičan niz može "produžiti" u konačni niz koji nije parcijalno periodičan.

Lema 6: (O "pretvaranju" parcijalno periodičnog niza u neperiodičan) Za svaki parcijalno periodični niz A nad skupom S koji sadrži bar dva elementa, postoji konačni niz A' koji sadrži podniz A i nije parcijalno periodičan.

Dokaz: Dokaz se izvodi konstrukcijom niza A' . Neka je niz $A=(a_1,a_2,\dots,a_k)$, $a_i \in S$. Neka je $x \in S$, $x \neq a_1$. Ovakav x sigurno postoji, jer S sadrži bar dva elementa. Neka je l dužina najdužeg podniza niza A koji se sastoji samo od elemenata x (l može biti i 0; l postoji jer je A konačan). Tada je $A'=(a_1,a_2,\dots,a_k,x,\dots,x)$, takav da se završava sa $l+1$ elemenata x , traženi niz. U daljem dokazu ćemo, bez gubitka općitosti, pretpostaviti da je $A=(A(1),A(2),\dots,A(k))$, i da je $A'=(A(1),A(2),\dots,A(k),\dots,A'(L))$.

Dokazujemo da A' nije parcijalno periodičan. Pretpostavimo suprotno, to bi po definiciji značilo da postoji neko $T \in \mathbb{N}$, $T < L$, da za svako $i \in \mathbb{N}$, takvo da je $1 \leq i+T \leq L$, važi $A'(i)=A'(i+T)$. Kako je $A'(1)=A(1) \neq x$, i $A'(L-l)=A'(L-l+1)=\dots=A'(L)=x$, mora biti $T < L-l-1$. Međutim, tada sa desne strane $A'(T+1)$ postoji podniz od $l+1$ elemenata x , to ne postoji nigde između $A'(1)$ i $A'(T+1)$, time se dolazi do kontradikcije. Na taj način je dokazano da A' nije parcijalno periodičan. \square

Uvećamo pojam proširenog PSP modela prilagođenog nekoj konačnoj stazi p u grafu prelaza. Ovaj pojam ćemo koristiti u narednim dokazima pomoću svojstava i u dokazu te nje kao optimalnom.

Definicija 35: (PSP model prilagođen konačnoj stazi u grafu) Neka je $p=(S_0,S_1,\dots,S_L)$ neka konačna staza u grafu prelaza modela PSP sa predikatskom matricom PM . Prošireni model PSP' sa matricom PM' naziva se modelom prilagođenim stazi p ako (po def.) PM' zadovoljava sledeće svojstvo za svaki predikat p : ako je $PM(p,k)$ levi ivični element ($k \neq 0$), onda je $PM'(p,k-L)$ takođe levi ivični element i ako je $PM(p,k)$ desni ivični element ($k \neq 0$), onda je $PM'(p,k+L)$ takođe desni ivični element. \square

Sada treba dokazati svojstvo da se u odgovarajućoj stazi grafa prelaza PSP modela prilagođenog stazi p može pojaviti jedno isto stanje na više mesta ako i samo ako je staza p u osnovnom modelu parcijalno periodična. Ovo svojstvo je potrebno u dokazu da PSP model zadovoljava strogi uslov te nje kao optimalnom za staze koje nisu parcijalno periodične. Ranije je već pokazano kako se intuitivno dolazi do ovog zaključka. Najpre ćemo definisati relaciju između konačnih staza u grafovima osnovnog i proširenog PSP modela.

Definicija 36: (Relacija između konačnih staza u grafu osnovnog i proširenog modela) Kaže se da neka staza p' u grafu prelaza proširenog PSP' modela odgovara stazi p u grafu prelaza osnovnog modela PSP ako (po def.) postoji beskonačna staza P koja sadrži podstazu p i beskonačna staza P' koja sadrži podstazu p' , tako da je $P'=E(P)$, i p' je određeno kao $f(p)$, što znači da se p' sastoji iz slike elemenata staze p , gde je E bijekcija izvršavanja, a f njena interna bijekcija. \square

Na primeru u 6.3.1. se vidi da ovako definisana relacija u općtem slučaju nije funkcija, jer za datu stazu p postoji više staza p' u grafu proširenog modela koje joj odgovaraju. Sledeće svojstvo se lako dokazuje i direktno sledi iz konstrukcije bijekcije izvršavanja:

Lema 7: (O vezi "pokrivanja" između osnovnog i modela prilagođenog stazi) Neka je $p=(S_0,S_1,\dots,S_L)$ neka konačna staza u grafu modela PSP, gde je SM_i matrica stanja S_i , $i=0,\dots,L$, i PSP' prošireni model. Tada za svaku stazu $p'=(S'_0,S'_1,\dots,S'_L)$ u grafu prelaza PSP' koja odgovara stazi p , gde je SM'_i matrica stanja S'_i , $i=0,\dots,L$, važi da je SM'_i proširena matrica stanja matrice SM_i . Ako je PSP' model prilagođen stazi p , onda važi i sledeće: svako stanje sa matricom stanja SM'_i iz staze p' ne pripada nijednoj drugoj stazi $p''=(S''_0,S''_1,\dots,S''_i=S'_i,\dots,S''_L)$ koja ne odgovara stazi p .

Dokaz: Neka je S beskonačna sekvenca izvršavanja koja figure u konstrukciji bijekcije izvršavanja E koja preslikava elemente p u elemente p' , odnosno P u P' . Prema konstrukciji E za svaki ne- b element je $SM_i(p,j)=S(i+j,p)=SM'_i(p,j)$, Ĺime je dokaz prvog tvrĹenja završen.

Dokazano sada i drugo tvrĹenje iz leme. Posmatrajmo bilo koju stazu $p''=(S''_0, S''_1, \dots, S''_L)$. Neka je $p=(S_0, S_1, \dots, S_L)$ staza u PSP takva da je p'' njena odgovarajuća staza u PSP' . Prema prvom tvrĹenju, za p va i da je $SM_{i+j}(p,k)=SM''_{i+j}(p,k)=SM''_i(p,j+k)=SM'_i(p,j+k)$, za sve j takve da je $0 \leq i+j \leq L$ i sve k takve da je $SM(p,k) \neq b$. Prema definiciji prilagoĹenog modela, ovakvo $SM'_i(p,j+k)$ je uvek ne- b element. MeĹtim, ovakvo svojstvo va i upravo za matrice SM staze p iz postavke leme, jer je za njih $SM_{i+j}(p,k)=SM'_i(p,k)=SM'_i(p,j+k)$, pa je i drugo tvrĹenje dokazano. \square

Na ovaj naĹin je pokazano da se skup staza u pro Ĺirenom prilagoĹenom modelu i skupovi pro Ĺirenih matrica stanja sa staze meĹsobno "pokrivaju": svaka staza p' iz prilagoĹenog modela koja odgovara stazi p prolazi samo kroz stanja "nastala" pro Ĺirivanjem stanja sa staze p , i svako stanje sa pro Ĺirenom matricom u prilagoĹenom modelu nalazi se samo na stazama p' koje odgovaraju stazi p . Ovo Ĺe omoguĹiti pomeranja operacija samo izmeĹ stanja nastalih pro Ĺirivanjem stanja sa staze koja se optimizuje i to sa istim rasporedima. Evo sada i najavljenog svojstva.

Lema 8: (*O svojstvu modela prilagoĹenog stazi koja je parcijalno periodična*) Neka je $p=(S_0, S_1, \dots, S_L)$ neka konačna staza u grafu prelaza modela PSP , i neka je PSP' model prilagoĹen ovoj stazi. Neka su $p'=(S'_0, S'_1, \dots, S'_L)$ staze u grafu prelaza modela PSP' koje odgovaraju stazi p i neka je skup SS_i skup svih stanja S'_i , $i=0, \dots, L$. Tada u neka dva skupa SS_i i SS_j postoji isto stanje S' ako i samo ako je p parcijalno periodična.

Dokaz: Ako je p parcijalno periodična, onda po definiciji postoji beskonačna periodična staza P koja sadr i stazu p sa periodom T manjim od du Ĺine staze p . Neka je S sekvenca izvršavanja koja po bijekciji izvršavanja E odgovara stazi P . Po konstrukciji bijekcije E , oĹigledno je da je S periodična, tako da je du Ĺina konačne sekvence $s=f^1(p)$ veće od T , pa u s postoje dva ista elementa $P(i)$ i $P(i+T)$. Neka je $S'_i=f'(P(i))$, $S'_{i+T}=f'(P(i+T))$, gde je f' interna bijekcija bijekcije izvršavanja E' modela PSP' . OĹigledno je da su S'_i i S'_{i+T} elementi neke p' . Zbog naĹina konstrukcije S'_i , takve da je za svaki ne- b element $SM'_i(p,j)=P(i+j,p)$, i zbog toga to va i da je $(\forall j \in Z)(P(j)=P(j+T))$, sledi da je $SM'_i(p,j)=P(i+j,p)=P(i+j+T,p)=SM'_{i+T}(p,j)$, pa je $S'_i=S'_{i+T}$. Ovime je jedan smer ekvivalencije dokazan. Treba primetiti da ovaj smer ekvivalencije va i za bilo koji, a ne samo prilagoĹeni model.

Dokazimo sada drugi smer ekvivalencije: ako postoje dva (ista) stanja $S'_i=S'_{i+T}$ iz neka dva SS_i i SS_{i+T} , tako da je $0 < T \leq L$ i $0 \leq i < i+T \leq L$, treba dokazati da je p parcijalno periodična. Neka je SM_1 i matrica stanja S'_i koje pripada stazi p_1' , a SM_2 i matrica stanja S'_{i+T} koje pripada stazi p_2' , pri Ĺemu i p_1' i p_2' odgovaraju stazi p . Po pretpostavci je $SM_1=S_2$. Kako je prema Lemi 7 SM_1 i pro Ĺirena matrica matrice SM_i (takoĹ je i SM_2 i pro Ĺirena matrica matrice SM_{i+T}), za svaki predikat p i svaki indeks k za koji je $SM(p,k)$ ne- b element, i svako j takvo da je $0 \leq j < j+T \leq L$, prema Lemi 4 va i: $SM_j(p,k)=SM_1j(p,k)=SM_1i(p,k+j-i)$. Kako je i $0 \leq i \leq L$, to je $-L \leq j-i \leq L$, pa je po definiciji prilagoĹenog pro Ĺirenja $SM_1i(p,k+j-i)$ ne- b element. Po pretpostavci je $SM_1i(p,k+j-i)=SM_2i+T(p,k+j-i)$, a kako za stazu p_2' va i da je $SM_2i+T(p,k+j-i)=SM_2j+T(p,k)$, i kako je $SM_2j+T(p,k)=SM_j+T(p,k)$, sledi da je $SM_j(p,k)=SM_j+T(p,k)$, odnosno $SM_j=SM_j+T$, to znaĹi da je p parcijalno periodična, Ĺime je i drugi smer ekvivalencije dokazan. Za ovaj smer je veće bilo neophona pretpostavka da je PSP' prilagoĹeno pro Ĺirenje modela. \square

Ostalo je jo da se formalno opi e postupak dobijanja optimalnog rasporeda du jedne staze. Neka je $p=(S_0, S_1, \dots, S_L)$ neka konačna staza Ĺije izvršavanje nije optimalno prema datoj definiciji (razlikuju se du Ĺina staze i du Ĺina zdru enog rasporeda). Optimalno izvršavanje ove

staze se dobija podelom po ciklusima zdru enog rasporeda na $L+1$ delova (novih iteracija). Ovakva podela oĖigledno nije jedinstvena. Neka su delovi ovakve podele oznaĖeni sa $0,1,\dots,L$. Ako se u ovakvoj podeli instanca $op[i+j]$, potekla od instance $op[i] \in S_j$, nalazi u delu oznaĖenom sa k , onda treba izvr ititi pomeranje operacije $op[i]$ iz stanja S_j nagore (*moveup*) $j-k$ puta, ako je $k < j$, odnosno nadole (*movedown*) $k-j$ puta, ako je $k > j$, i ako su ta pomeranja moguæa prema uslovima pomeranja.

6.3.5. Dokaz svojstva teŖnje ka optimalnom

Sada se mo e konstruisati ceo dokaz svojstva te nje PSP modela ka optimalnom. Dokaz æebiti samo formalno ponavljanje intuitivnih zakljuĖaka iz odeljka 6.3.1. Radi preglednosti navodimo ponovo iskaz teoreme koja se na ovo svojstvo odnosi.

Teorema 7: (*Svojstvo teŖnje PSP modela ka optimalnom*) *PSP model zadovoljava slabiji uslov teŖnje ka optimalnom sa sledeæem znaenjem. Ako postoji konaæna staza u grafu prelaza datog PSP modela, takva da je $Len() \leq Len(CR())$, onda postoji semantiæki ekvivalentan model PSP' u kome vaŖi da je $Len() = Len(CR())$, za neke ' od konaænih staza u grafu modela PSP' koje sadrŖe podstaze iz skupa staza odreĖenih kao $f()$ za sve beskonaæne staze koje sadrŖe , pri èemu je f interna bijekcija bijekcije izvršavanja E izmeĖu PSP i PSP' , dok se $Len()$ za ostale " koje su disjunktne sa ' iz grafa modela ne menjaju.*

Dokaz: Ako postoji ovakva konaĖna staza p Ėije izvr avanje nije optimalno, onda postoji konaĖni niz pomeranja operacija koja treba izvr ititi izmeĖ stanja na ovoj stazi, kako bi se dobilo optimalno izvr avanje. Ovaj niz pomeranja opisan je na kraju prethodnog odeljka. Neka je PSP' pro ireni model datog modela, prilagoĖn stazi p , pri èemu su rasporedi svih stanja S' "nastalih" iz stanja S (koji imaju pro irenu matricu stanja S) jednaki rasporedu S . U Lemi 7 pokazano je da sve staze p' koje odgovaraju stazi p prolaze kroz odgovarajuæa stanja sa pro irenim matricama, i da sva ova stanja sa pro irenim matricama pripadaju nekoj stazi p' . Prema tome, svako od potrebnih pomeranja moguæe je izvr ititi izmeĖ neka dva klastera u PSP' , pri èemu su sva stanja ta dva klastera "nastala" pro irenjem dva susedna stanja sa staze p . Pri tom, kako je u Lemi 8 pokazano, ako p nije parcijalno periodiĖna, onda u svim klasterima PSP' izmeĖ kojih treba vr ititi pomeranja ne postoji jedno isto stanje na dva mesta. Zbog toga se sva pomeranja vr e potuno nezavisno: jedno pomeranje izmeĖ dva klastera nema "sporednog" efekta na neko stanje koje je nastalo od stanja S koje se nalazi vi e puta na stazi p . Kako prema Lemi 7 stanja izmeĖ kojih se vr e pomeranja u PSP' ne pripadaju nijednoj drugoj stazi koja odgovara nekoj stazi koja je disjunktne sa stazom p ²³, izvr avanja ovih drugih staza se ne menjaju. Na taj naĖin, ako p nije parcijalno periodiĖna, raspored na svim stazama p' se minimizuje. Na ovaj naĖin je dokazana i Teorema 8.

Ako je p parcijalno periodiĖna, onda prema Lemi 6 postoji neka konaĖna staza koja sadr i p a nije parcijalno periodiĖna, jer iz svakog stanja postoje prelazi u bar dva stanja. Tada se pomeranja mogu na isti opisani naĖin izvr ititi u modelu prilagoĖenom ovoj produ enoj stazi, tako da se rasporedi samo nekih staza koje odgovaraju stazi p skraæu, dok se za ostale izvr avanje ne menja.

□

6.3.6. Alternativna postavka svojstva teŖnje ka optimalnom

PSP model je pogodan i za jo jednu postavku svojstva te nje ka optimalnom koja je intuitivno daleko jasnija od navedene, ali koja nije stavljena u prvi plan zato to je specifiĖna za PSP model. Naime, ova postavka se zasniva na sledeæem razmatranju. PSP model sa svojim stanjima zapravo

²³Ako je p_1 disjunktne sa p , sve staze koje odgovaraju p_1 imaju stanja sa matricama koje su pro irene matrice stanja na stazi p_1 , pa su one razliĖite od matrica stanja na stazama koje odgovaraju p . Zbog toga sva stanja iz staza koje odgovaraju stazi p , pripadaju samo stazama koje odgovaraju stazama iz PSP koje nisu disjunktne sa p .

predstavlja model izvršavanja petlje sa uslovnim grananjima, pri čemu svakoj sekvenci izvršavanja odgovara jedna staza u grafu prelaza. Svako stanje na stazi odgovara jednoj iteraciji petlje, a prelazi iz stanja u stanje određuju se prema ishodima uslovnih operacija. Pod pretpostavkom da su verovatnoće svih ishoda uslova jednake, intuitivno se dolazi do postavke da je verovatnoća svih stanja jednaka²⁴. Na taj način se zaključuje da je prosečan interval iniciranja transformisane petlje, to je zapravo osnovni parametar celog postupka, jednak prosečnoj dužini rasporeda svih stanja u modelu.

Ovo navodi na zaključak da svojstvo te nje ka optimalnom treba postaviti tako da ono znači mogućnost proizvoljnog smanjivanja prosečne dužine rasporeda stanja u modelu, pod uslovom da model većije optimalan. Upravo ovakvu postavku dajemo u sledećoj teoremi.

Teorema 9: (Alternativna postavka težnje PSP modela ka optimalnom) *PSP model teži ka optimalnom sa sledećim značenjem. Ako postoji konačna staza p u grafu prelaza datog PSP modela, takva da je $Len(p) > Len(CR(p))$, onda postoji semantički ekvivalentan model PSP' u kome je prosečna dužina rasporeda stanja manja od prosečne dužine rasporeda stanja u PSP .*

Dokaz: Kao što je ranije već pokazano, ako u modelu PSP postoji staza p čije izvršavanje nije optimalno, onda postoji konačan niz (potencijalnih) pomeranja operacija između stanja na ovoj stazi koji dovodi do optimalnosti te staze. To znači i da od ukupnog broja stanja na stazi p postoji veći broj stanja čiji se rasporedi skraćuju ovim pomeranjima, od broja onih stanja čiji se rasporedi povećavaju (jer se ukupna dužina staze skraćuje). Neka je PSP' prošireni model prilagođen ovoj stazi p . U ovom modelu, prema Lemi 7, svako stanje S_i odgovara stanju Si , tako što ima isti raspored i proširenu matricu. Neka je SSi skup svih stanja S_i . Lako je videti da je broj elemenata SSi za svako i jednak, jer su u SSi , prema Lemi 7, sva i samo ona stanja koja imaju proširenu matricu stanja Si . Pomeranja operacija u PSP' , prema lemi koja je opisana, imaju isti efekat na sva stanja S_i kao što bi imala pomeranja operacija na Si . Osim toga, nijedno stanje van ovih skupova ne trpi nikakvu izmenu ovim pomeranjem. Prema tome, broj stanja u PSP' čiji se rasporedi skraćuju opet će biti veći od broja stanja kojima se rasporedi produžavaju, pa je prosečna dužina rasporeda stanja u PSP' kraćeg od u PSP . Naravno, ukoliko je p parcijalno periodična, treba je produžiti u stazu koja to nije, kako bi pomeranja operacija bila nezavisna u PSP' , kao što je to i ranije rađeno. \square

Naravno, ako ovakva staza p ne postoji, dati model predstavlja optimalno izvršavanje petlje.

6.4. Ostala svojstva PSP modela

Svojstvo te nje ka optimalnom je sigurno najvažnije svojstvo PSP modela koje omogućava "proizvoljno dobro" raspoređivanje, naravno uz povećanje složenosti modela. Međutim, ranije je već ukazano da postoje i druga važna svojstva ovog modela koja omogućavaju njegovu praktičnu primenu. To je pre svega mogućnost eliminacije proizvoljne predikcije. Ova mogućnost se oslanja na jedno svojstvo proširenja modela. Najzad, ovde će biti pokazano kako se softverska protočnost petlji bez uslovnih grananja može posmatrati kao specijalni slučaj PSP modela.

6.4.1. Neka svojstva proširenja modela

U Glavi 5 pokazano je da je jedna prepreka za mogućnost pomeranja operacija nepostojanje operacije u svim stanjima klastera iz kog operaciju treba pomeriti (ili ona nije slobodna u svim tim stanjima). Ova prepreka može doći do izražaja i pri pomeranju operacija u cilju eliminacije proizvoljne predikcije opisane u 5.4.2. U ovom slučaju, kako je u prethodnom poglavlju

²⁴Na ovo razmatranje može se detaljnije vratiti u 7.2.4.

pokazano, ova prepreka se može eliminisati proirivanjem modela tako da bude prilagođen stazi (u ovom slučaju stazi dužine dva) na kojoj se vrše pomeranja. Međutim, ovakvo proširenje je "maksimalističko" (jer je obostrano) i uvodi veliki broj dodatnih instanci predikata u predikatsku matricu, što unosi nove probleme u postupak eliminacije proizvoljne predikcije.

Posmatrajmo problem na jednom primeru. Neka su dva stanja iz kojih treba premestiti operacije nadole zbog izjednačavanja dužina nekih stanja data matricama stanja $[0\ 0\ 1]$ i $[1\ 0\ 1]$. Pomeranje treba izvršiti u odredišta stanja $[0\ 1\ 0]$ i $[0\ 1\ 1]$. Neka je npr. stanje $[0\ 0\ 1]$ ono iz kog treba premestiti neku operaciju koja ne postoji u stanju $[1\ 0\ 1]$. U proširenom modelu koji je prilagođen stazi $[0\ 0\ 1] \rightarrow [0\ 1\ 0]$ bi se potrebno pomeranje izvršilo samo između novih stanja "nastalih" proširivanjem ova dva stanja $[0\ 0\ 1]$ i $[0\ 1\ 0]$. Međutim, proširenje koje je ovako prilagođeno je obostrano, pa unosi nove instance predikata i sa leve i sa desne strane. Kritičan je predikat koji se unosi sa desne strane jer on ima veći indeks, pa se time kasnije i razrešava. Na ovaj način obostrano proširenje može da unese u model nove instance predikata koje se odnose na još "dalju budućnost" nego već postojeće instance, što ukazuje na neograničeni postupak eliminacije proizvoljne predikcije.

Na sreću ovakvo obostrano proširenje u posmatranom slučaju nije neophodno. Kako treba premestiti operaciju samo iz stanja nastalih od $[0\ 0\ 1]$, a ta operacija može biti pomerena u stanja nastala od oba stanja $[0\ 1\ 0]$ i $[0\ 1\ 1]$, dovoljno je model proširiti samo nalevo. Ako se to učini, onda od stanja $[0\ 0\ 1]$ nastaju stanja $SM'11=[0\ 0\ 0\ 1]$ i $SM'12=[1\ 0\ 0\ 1]$, od stanja $[0\ 1\ 0]$ nastaju stanja $SM'21=[0\ 0\ 1\ 0]$ i $SM'22=[1\ 0\ 1\ 0]$, a od stanja $[0\ 1\ 1]$ nastaju stanja $SM'31=[0\ 0\ 1\ 1]$ i $SM'32=[1\ 0\ 1\ 1]$. Iz stanja $SM'11$ i $SM'12$ prelazi se u stanja $SM'21$ i $SM'31$, pa je time pomeranje moguće i dovoljno je da iz rasporeda svih stanja nastalih od $[0\ 0\ 1]$ izbaci suvišne operacije radi smanjenja njihove dužine.

Prema tome, sledi jedno zanimljivo svojstvo PSP modela: ako se iz stanja $S1$ prelazi u stanja $S2$ i $S3$, onda u levo proširenom modelu "za jedan", postoji klaster koji se sastoji samo od stanja nastalih od stanja $S1$, iz koga se prelazi u klaster koji se sastoji od stanja nastalih od $S2$ i $S3$. Analogno važi i za desno proširenje "za jedan". Ovo svojstvo može i formalno postaviti za opšti slučaj.

Lema 9: (*O jednom svojstvu levog i desnog proširenja*) Neka je PSP dati model i neka u njemu postoji prelaz iz stanja S u stanja $S1, S2, \dots, Sk$. Neka je PSP' levo proširenje ovog modela, tako da je levi ivični element svake vrste PM različit od levog ivičnog elementa PM'. Tada u PSP' postoji izvorišni klaster koji se sastoji samo od stanja koja imaju proširenu matricu stanja S . Analogno važi za desno proširenje i odredišni klaster.

Pri tom, ako je levo proširenje "za jedan", što znači da levi ivični element u svakoj vrsti PM ima za jedan veći indeks od levog ivičnog elementa u istoj vrsti PM', onda postoji izvorišni klaster koji se sastoji od svih stanja koja imaju proširenu matricu stanja S . Analogno za desno proširenje.

Dokaz: Kako je u modelu PSP', leva ivica matrice sasvim različita (u svim vrstama PM) od leve ivice PM, onda za svaki ne-b element $SM'(p, i)$ koji je jednak $SM(p, i)$ važi da je van leve ivice PM'. Kako izvorišnom klasteru u PSP' pripadaju samo stanja sa matricama koje imaju jednake ne-b elemente van leve ivice, sledi da postoji izvorišni klaster sa opisanim svojstvom da se sastoji samo od stanja "nastalih" od S . Ako je još proširenje "za jedan", onda po definiciji izvorišnog klastera sva stanja koja su "nastala" od S pripadaju istom izvorišnom klasteru. Dokaz za desno proširenje je potpuno simetričan. \square

Još jedno svojstvo proširenja je bitno u praktičnoj primeni PSP modela. Ovo svojstvo je intuitivno jasno i pokazuje se kao tačno na svim testiranim primerima, ali za sada ostaje bez formalnog dokaza. Ono se kratko i neformalno može iskazati na sledeći način: za svako pomeranje koje se može izvesti u nekom PSP modelu, postoji niz pomeranja koja se mogu izvesti i u proširenom modelu sa istim efektom. Evo sada formalnog iskaza ovog tvrdjenja.

Teorema 10: (O jednom svojstvu pomeranja u osnovnom i proširenom modelu) Neka je PSP dati model i $PSP1$ njemu ekvivalentan model. Neka je PSP' prošireni model modela PSP takav da je raspored svakog stanja S iz PSP jednak rasporedu stanja S' iz PSP' , za svako S' koje odgovara stanju S . Neka je $PSP1'$ na isti način formiran prošireni model modela $PSP1$, sa istom predikatskom matricom kao model PSP' . Tada je PSP' ekvivalentan sa $PSP1'$. \square

Treba primetiti da bi dokaz ove teoreme bio jednostavan kada bi bila dokazana sledeća lema.

Lema 10: (O uslovu ekvivalencije semantički ekvivalentnih modela) Ako su dva modela $PSP1$ i $PSP2$ sa istom predikatskom matricom semantički ekvivalentna, onda su oni i ekvivalentni (u smislu da se jedan od drugog može dobiti konačnim nizom pomeranja operacija). \square

Naime, dokaz Teoreme 10 bi, uz vaenje Leme 10, i ao ovako. Kako su PSP i $PSP1$ ekvivalentni, oni su i semantički ekvivalentni: $SE(PSP, PSP1)$. Kako je $SE(PSP, PSP')$ i $SE(PSP1, PSP1')$ po dokazanom svojstvu semantičke ekvivalencije proširenja, i kako je SE relacija ekvivalencije, to je i $SE(PSP', PSP1')$. Kako su PSP' i $PSP1'$ modeli sa istom predikatskom matricom, prema Lemi 10 sledi da su oni i ekvivalentni.

Evo za to je ovo svojstvo od praktičnog značaja. Ono dozvoljava da se optimizacija petlje započne na PSP modelu koji ima predikatsku matricu sa malim brojem ne-b elemenata, što znači da ima i mali broj stanja. Na ovom modelu treba izvršiti sva moguća pomeranja u cilju optimizacije. Zatim model treba proširiti prema nekom pravilu, i na proširenom modelu izvršiti samo pomeranja koja nisu bila moguća u osnovnom modelu. Ovaj postupak treba iterativno ponavljati do neke granice. Na taj način se smanjuje vreme rada na optimizaciji, jer se sva moguća pomeranja vrše na manjem broju stanja. Suprotni pristup, koji bi počeo od neke velike predikatske matrice i na njoj izvršio sva pomeranja trajao bi dugo, jer je intuitivno jasno da jedno pomeranje u osnovnom modelu zahteva više pomeranja u proširenom modelu da bi se postigao isti efekat.

6.4.2. Eliminacija proizvoljne predikcije

Sumirajmo sada zaključke koji su vezani za problem eliminacije proizvoljne predikcije opisan u 5.4.2. Postupak eliminacije sastoji se u tome da se najpre pronađu dva stanja čija je dužina rasporeda različita, a trenutak razrešavanja uslovne operacije po čijem se ishodu ta dva stanja razlikuju nalazi se posle završetka kraćeg stanja. Tada iz dužeg stanja S treba prebaciti sve operacije koje su "višak", nadole u sledeća stanja, kako bi se dužina posmatrana dva stanja izjednačila. Ovaj postupak u sebi sadrži sledeće probleme:

1. Moguća je da neka operacija koju treba pomeriti nadole ne postoji u svim stanjima izvornog klastera kome pripada S , ili ona u svim tim stanjima nije slobodna na dnu. Tada treba pribegavati proširenju modela "za jedan", kao što je opisano u prethodnom odeljku, čime se ovaj problem rešava.

2. Levim proširenjem se uvode nove instance predikata. Na sreću levo proširenje uvodi "starije" instance predikata nego one koje već postoje. Ovo dovodi do intuitivnog zaključka da se levim proširenjima, odnosno ponavljanjem proširivanja ulevo, ne utiče na konačnost postupka eliminacije proizvoljne predikcije. Ovaj intuitivni zaključak ostaje zasad bez potpunog formalnog dokaza.

3. Potrebno je da se navedeni postupak eliminacije sigurno završava. Ovo je zasad otvoreno pitanje na koje nema ni intuitivnog zaključka. Naime, pomeranjem operacija nadole rasporedi stanja u određenom klasteru se menjaju, pa se potencijalno menjaju i njihove dužine, što može dovesti do pojave novih parova stanja čije rasporede treba izjednačiti po trajanju.

4. Na svojstvo da se postupak uvek završava utiče i sledeći efekat: ako je neka od operacija koja se pomera nadole uslovna operacija, onda se trenutak razrešavanja uslova ovim pomeranjem kasni, pa je moguća nastajanje novih stanja čije razlike u trajanju treba eliminisati.

5. Najzad, dobro bi bilo da se navedenim postupkom eliminacije krajnji raspored ne pogorava, to opisno znači da se ukupna dužina trajanja svih stanja ne povećava. Ovo je sasvim moguće s obzirom da se u levom proširenju "za jedan" sigurno smanjuje dužina 2^m stanja za neku vrednost l , jer se premeštaju sve operacije sa kraja rasporeda, dok se dužina 2^m stanja u najgorem slučaju povećava za istu vrednost l (dužina može i da se ne povećava toliko, već manje).

Treba primetiti da se isti postupak eliminacije potpuno analogno može sprovesti dualnom logikom: pomeranjem nagore i desnim proširenjem "za jedan". Ovakva logika dualizuje i neke probleme: uslovne operacije se pomeraju nagore, pa se uslovi razrešavaju ranije; međutim, desno proširenje unosi nove instance predikata koje su još dalje "u budućnosti".

Kao što se vidi, većina ovih pitanja zasad ostaje i bez intuitivnog odgovora, a naročito bez formalnog dokaza. Opirniji eksperimenti treba da pokažu u koje od ovih svojstava potencijalno vodi u opštem slučaju, da bi se uložio napor u njihovo dokazivanje. Eksperimenti mogu da pokažu na primeru i da neko od ovih svojstava ne vodi.

6.4.3. Petlje bez uslovnih grananja kao specijalni slučaj PSP modela

Petlje bez uslovnih grananja predstavljaju samo specijalni slučaj PSP modela. PSP model za petlje bez uslovnih grananja ima samo jedno stanje, jer postoji 0 uslovnih operacija i 0 njihovih instanci. Prema tome, iz stanja S se trivijalno prelazi u isto ovo stanje S . Pomeranja operacija nagore i nadole su potpuno analogna: ako se iz S premesti operacija $op[i]$ nadole u isto to S , ona postaje $op[i-1]$; ako se iz S premesti operacija $op[i]$ nagore u isto to S , ona postaje $op[i+1]$. Na ovaj način se postiže softverska protočnost. Kriterijum za izbor operacija za premeštanje može biti potpuno isti kao i kod petlji sa uslovnim grananjima: osnovno je smanjiti trajanje jedne iteracije.

Posmatrajmo kako se ovaj trivijalni model ponaša u kontekstu te nje kao optimalnom. U grafu prelaza postoji samo jedna grana iz S u S . Zato je svaka staza u grafu parcijalno periodična. Kako skup stanja nema dva stanja, ova parcijalno periodična staza se ne može "pretvoriti" u neperiodičnu. Formalno, ovakav PSP model ne zadovoljava uslov te nje kao optimalnom, sa sledećim značenjem: ako izvršavanje neke staze u modelu nije optimalno, ne mora postojati efektivan postupak kojim se izvršavanje te staze optimizuje.

Ovo je zapravo formalno tumačenje poznate činjenice da je optimalni interval iniciranja II jednak najbližem celom broju ne manjem od količnika dužine i iteracione distance kritičnog zatvorenog puta u cikličkom grafu zavisnosti petlje. Kada navedeni količnik nije ceo broj, izvršavanje svake konačne staze u PSP modelu nije vremenski optimalno, a ne može ni biti. Ovo tumačenje navodi na već pomenuto mišljenje da parcijalno periodične staze u PSP modelu predstavljaju prirodno ograničenje optimizacije petlji sa uslovnim grananjima u kontekstu softverske protočnosti, na suštinski isti način kao što periodičnost predstavlja ograničenje optimizacije petlji bez grananja.

7. Model za realni slučaj i heuristike

U ovoj glavi biće najpre razmatrana prilagođanja predloženog apstraktnog PSP modela na realni slučaj optimizacije koda petlji sa uslovnim grananjima. U drugom delu glave predložene su heuristike za raspoređivanje operacija, odnosno za izbor i porenje predikatske matrice, izbor pomeranja operacija, kao i smernice za generisanje koda.

7.1. Modifikacije PSP modela za realni slučaj

U teorijskoj postavci PSP modela ukinjene su mnoge pretpostavke i ograničenja u cilju jednostavnije formalizacije modela i dokazivanja nekih njegovih općih svojstava. Logično je očekivati da PSP model bez ovakvih ograničenja nema garantovana sva svojstva dokazana u prethodnoj glavi, ali da bar pokazuje tendenciju ka tim svojstvima. Ovde će biti razmatrani praktični aspekti implementacije PSP modela za realni slučaj optimizacije koda. Implementacija PSP modela sadrži nekoliko problema koji do sada nisu razmatrani: način tretiranja operacija trajanja više od jednog ciklusa takta, slučaj ugneženih IF struktura, slučaj ograničene količine resursa, problem eliminacije proizvoljne predikcije i problem konačnog generisanja koda.

7.1.1. Operacije proizvoljnog trajanja

U teorijskoj postavci PSP modela ukinjena je pretpostavka da sve operacije traju po jedan ciklus takta. Navedena je i preporuka da u slučaju da operacija traje duže, operaciju treba podeliti na delove od po jedan ciklus, i dalje ove delove tretirati nezavisno.

Ovakva strategija može donekle da se zadrži i u realnom slučaju. Naime, razlikuju se dve vrste mašina za koje se opisana tehnika optimizacije koda petlje može primeniti: prva vrsta su mašine kod kojih zaista sve operacije traju jedan ciklus takta, npr. mikroprogramske jedinice ili VLIW mašine sa eksplicitnom kontrolom protočnih resursa (svako napredovanje protočne obrade se eksplicitno navodi u instrukciji); druga vrsta su sve ostale mašine kod kojih operacije mogu da traju više ciklusa, npr. RISC protočni procesori, superskalari i VLIW mašine sa protočnim resursima kod kojih se instrukcijom zadaje samo inicijalizacija protočne obrade, a napredovanje protočnih resursa se obavlja interno (hardverski). Kod ove druge vrste mašina, operacije koje traju više ciklusa se mogu podeliti na delove u PSP modelu, ali se njihova pomeranja ne mogu obavljati sasvim nezavisno.

Naime, kod pomeranja delova operacija nezavisno može nastati sledeći problem: jedan deo operacije može se nalaziti nekoliko iteracija pre narednog dela operacije, tako da se ta dva dela, koji inače treba da budu susedni, izvršavaju sa vremenskim rascepom.

Predlaže se rešenje ovog problema koje se sastoji u ograničavanju pomeranja delova operacija. Naime, delovi operacija se mogu, osim zavisnostima po podacima (koji sprežavaju njihovo "sabijanje"), vezati i tako da se spreži njihovo udaljavanje za više od jednog ciklusa. To bi značilo da između delova operacije postoje grane zavisnosti koje imaju dužinu koja u rasporedu predstavlja i donje i gornje ograničenje: dva dela ne smeju biti ni na manjem ni na većem vremenskom udaljenju nego što je dužina grane. Na taj način bi se, osim postojećih, u uslove pomeranja operacije uvrstio i sledeći. Ako neki deo operacije treba pomeriti nagore, to se ne može izvršiti ukoliko naredni deo iste operacije ne ostaje da se izvrši odmah u narednom ciklusu. Analogno treba da važi za pomeranje nadole.

Na primer, neka su $S1$, $S2$ i $S3$ redom tri susedna stanja na nekoj stazi duž koje se vrši pomeranje dela operacije op_1 iz $S2$ u $S1$. Ako se naredni deo op_2 iste operacije nalazi u $S3$,

naravno pod pretpostavkom da je du ina rasporeda stanja S_2 jednaka 1, i ako se pomeranjem dela op_1 zadrava du ina 1 rasporeda S_2 , ovo pomeranje je zabranjeno, jer bi između op_1 u S_1 i op_2 u S_3 nastao vremenski procep od jednog ciklusa takta.

Druga, jo ne sasvim ispitana ideja za re enje istog problema sastoji se u tome da se u raspoređivanje uvrste samo operacije kao celine, bez obzira koliko traju, i da se one kao takve raspoređuju. Pri raspoređivanju se uzima u obzir trajanje operacije tako to se određuje najraniji trenutak početka neke operacije, u odnosu na trajanje operacija od kojih ona zavisi. Nejasno je sasvim kako pri tome eksplicitno odrediti du ina rasporeda jednog stanja, ukoliko se dozvoljava rasprostiranje jedne operacije preko vi e susednih stanja. Ova du ina rasporeda stanja je osnovni kriterijum pri raspoređivanju. Sa druge strane, utisak je da ovakvo re enje mo e bolje da se uklopi u slučaj kada postoji ograničenje na resursima. Ovaj predlog ostaje da se ispita.

7.1.2. Ugnježdene IF strukture

Teorijska postavka PSP modela uslovne operacije, odnosno njihove ishode, tretira potpuno nezavisno, kao predikate. Međtim, u realnom slučaju, kada u telu petlje postoji vi e uslovnih operacija, njihovi ishodi mogu da budu zavisni. To je slučaj kod ugnježdenih IF struktura, npr.:

```

LOOP
  ...
  IF p1 THEN
    ...
    IF p2 THEN
      ...
    END IF
    ...
  ELSE
    ...
  END IF
  ...
END LOOP

```

U ovom primeru predikati p_1 i p_2 nisu nezavisni: ako je ishod p_1 jednak F , ishod predikata p_2 nije bitan, odnosno ne postoji. Za PSP model to praktično znači da oba stanja sa matricama stanja u kojima je vrednost nekog $SM(p_1, i)$ jednaka 0, a vrednost $SM(p_2, i)$ bilo koja ne-b vrednost, predstavljaju isto stanje sa istim rasporedom.

U ovom slučaju, kada postoji ugnježđivanje IF struktura, predla e se sledeći postupak. U PSP modelu treba uvek obezbediti (i inicijalno i pri pro irivanju) da međsobno zavisni predikati p_1 i p_2 imaju ne-b elemente u istim kolonama predikatske matrice, Lime se obezbeđuje da se u matricama stanja uvek znaju ishodi oba ova predikata. Sva stanja koja imaju vrednosti $SM(p_1, i)$ jednaku 0 (za kombinaciju kao u primeru) i proizvoljnu vrednost $SM(p_2, i)$ treba inicijalno da imaju isti raspored. Intuitivno je jasno da se mo e obezbediti da i konačni rasporedi ovih stanja budu jednaki, pa se sva ova stanja mogu ujediniti u jedno, zdru eno stanje pri generisanju koda.

Opisani postupak je praktično ekvivalentan sledećem. Sve zavisne uslovne operacije, njih $k > 1$ (u primeru p_1 i p_2), treba zameniti jednim predikatom u predikatskoj matrici PSP modela, ali koji u matricama stanja imati vi e od 2, a manje od 2^k vrednosti, npr. $\{1, 2, 3, \dots, l\}$. Svaka od ovih vrednosti odgovara jednom kombinovanom ishodu zdru enih predikata. Dalje se PSP model formira tako to se svakoj matrici stanja pridru uje jedno stanje sa inicijalnim rasporedom na uobičajeni način, a prelazi formiraju prema istim većavedenim pravilima. Na ovaj način se mo e doći do uop tenja PSP modela kod koga vrednosti elemenata matrice stanja pripadaju proizvoljnom konačnom skupu $\{1, 2, \dots, l\}$. Ostaje da se ispituju sva svojstva navedena u ovom radu za ovako uop teni PSP model.

Očigledno je da ovaj problem zahteva dalju iscrpniju teorijsku i praktičnu analizu, ali da ne predstavlja nepremostivu prepreku za implementaciju PSP modela.

7.1.3. Ograničenja u resursima

Kada postoji ograničenje u resursima, za sada postoji samo predlog rešenja koje se sastoji u tome da se konačan raspored transformisane petlje dobijen pomoću PSP modela propusti kroz neki od postojećih algoritama za globalno raspoređivanje, npr. *List Scheduling*. Ostaje mnogo posla oko preciznog definisanja ovakvog rešenja, ili oko formulisanja potpuno novog rešenja koje bi bilo posebno prilagođeno PSP modelu, i koje bi se sastojalo u tome da se kriterijum ograničenja u resursima unese u korake algoritma pomeranja operacija u PSP modelu. Na taj način bi se konflikti na resursima rešavali tokom raspoređivanja, kao što je to kod raspoređivanja po modulu ili EPS. Za sada je najveći problem kod ovakvog iterativnog rešenja u tome što PSP model rastavlja stanja po ishodima predikata, dok se pri generisanju koda može dogoditi da se rasporedi nekoliko stanja ujedinijuju.

7.1.4. Eliminacija proizvoljne predikcije

O ovom problemu već je bilo dosta reči. Ovde ćemo se osvrnuti na još neke aspekte rešavanja ovog problema u praksi.

Predlaže se sledeći postupak. U konačnom PSP modelu koji je dobijen pomeranjima operacija, treba krenuti od neke instance predikata koji u predikatskoj matrici ima najveći indeks. Za ovu instancu predikata treba uzeti neka dva stanja čije se matrice razlikuju samo po vrednosti te instance. Ako se u njima rasporeda ova dva stanja ne razlikuju, prelazi se na novu instancu predikata. Ako se razlikuju, utvrđuje se trenutak razrešavanja ishoda posmatrane instance predikata.

Ovaj postupak pronalaznja trenutka razrešavanja ishoda neke instance predikata nije sasvim jednostavan i svodi se na pronalaznje odgovarajuće instance uslovne operacije. Teoretski, ovo pronalaznje se svodi na pretragu svih staza koje prolaze kroz posmatrana stanja. Kako su ove staze beskonačne, a i njihov broj je neograničen, za sada se ne vidi pouzdan ukazatelj da je postupak izvodljiv. Ipak, intuitivno se zaključuje da se do tražene operacije brzo dolazi, jer se ona mora nalaziti na svakoj stazi koja prolazi kroz neko od datih stanja, što znači da se nalazi u nekom od rasporeda stanja neke od staza koju je najlakše konstruisati. To može da bude neka periodična staza koja prolazi kroz konačan broj stanja. Takva staza uvek postoji, jer je skup stanja svakog PSP modela konačan. Detaljniju analizu zahteva pitanje šta se dešava ako se tražena operacija nalazi na različitim mestima u različitim stazama.

Ako se negde na stazi pronađe tražena operacija, lako se određuje da li se ona izvršava pre ili posle poslednjeg ciklusa kraćeg od dva posmatrana stanja. Ako se ta operacija izvršava posle pomenutog ciklusa, treba pribegati izjednačavanju dužina dva posmatrana stanja pomeranjem operacija iz dužeg stanja nadole, uz eventualno levo proširenje modela za jedan, kako je to u prethodnoj glavi objašnjeno.

Posle izjednačavanja dužina dva posmatrana stanja, postupak se ponavlja za nova dva stanja iste instance predikata ili naredne instance predikata. Postupak je završen ako ne postoji potreba da se neka dva stanja izjednače po dužini, prema opisanom kriterijumu. Kao što je rečeno, za sada nije dokazano da se ovo uvek dešava. Zato se može usvojiti neki drugi, heuristički kriterijum završetka ovog postupka. Eksperimentalna analiza ovog problema ostaje za dalji rad. U svakom slučaju, ovaj problem ne predstavlja nepremostivu prepreku za primenu PSP modela u optimizaciji, jer se čak i bez eliminacije proizvoljne predikcije može dobiti izvršnog koda koji samo može da bude manje efikasan.

7.1.5. Generisanje koda

Kada se dobije konačan izgled PSP modela, posle eventualne eliminacije proizvoljne predikcije, potrebno je generisati kod. Postupak za generisanje koda tela petlje je ranije već intuitivno opisan i sastoji se redom u objedinjavanju rasporeda po dva stanja koja se razlikuju samo u vrednosti jedne instance predikata u matrici stanja. Ovo objedinjavanje se sastoji u tome da se u združeni raspored

uvek uključuje ona operacija koja se u oba polazna rasporeda nalazi u istom ciklusu. Ova operacija se izvršava bezuslovno. Za ostale operacije se postupa na sledeći način. Najpre se utvrđuje ciklus izvršavanja operacije koja razrešava ishod instance predikata po kome se dva stanja razlikuju. Problem koji za sada nema rešenje nastaje ako se ovaj ciklus razlikuje za dva posmatrana stanja. Pitanje je da li se ovo uopšte može dogoditi. Zanimarimo za sada ovaj problem. Tada se sve operacije koje su raspoređene pre ovog ciklusa, a razlikuju se u dva posmatrana stanja, izvršavaju spekulativno. Ostale operacije se raspoređuju u dve grane IF strukture koja se generiše od operacije koja razrešava ishod posmatranog predikata.

Ostaje mnogo detalja koji nisu precizno definisani u ovom postupku. Neki od njih se mogu rešiti već postojećim metodama, kao što je npr. pitanjeivotnog veka neke promenljive koji se protekne na višej iteracija. Ostaje sasvim otvoren problem generisanja pretpetlje i postpetlje. Ipak, treba naglasiti da je uloga predloženog PSP modela prvenstveno *raspoređivanje*, a ne *generisanje koda*. Generisanje koda ionako jako zavisi od modela ciljne mašine za koju se vrši optimizacija.

7.2. Heuristike za raspoređivanje

Ovde će biti opisan postupak optimizacije koda pomoću PSP modela sa više praktičnih detalja. Ovaj postupak sprovodi se, kao što je rečeno, samo u cilju raspoređivanja, a pre eliminacije proizvoljne predikcije i generisanja koda. Ograničenja na resursima ne uzimaju se u obzir. Biće predložene i neke heuristike koje se mogu primeniti tokom postupka.

7.2.1. Globalni postupak

Postupak raspoređivanja pomoću PSP modela sprovodi se iterativno, tako da se u svakom koraku vrši niz pomeranja u modelu sa jednom predikatskom matricom koja se u narednom koraku proširuje. Dakle, u svakom koraku algoritma, dati PSP model je tekuci on se u narednom koraku proširuje. Polazi se od PSP modela sa predikatskom matricom koja ima samo jednu kolonu sa indeksom 0 i onoliko vrsta koliko predikata postoji u telu polazne petlje. Početni rasporedi stanja ovakvog modela generišu se tako što se u neko stanje unose one i samo one operacije koje se izvršavaju za odgovarajućishod predikata.

U svakom koraku se nad datim PSP modelom vrše sva pomeranja operacija koja su moguća i koja su heuristikama odabrana za primenu. Moguće su pomeranja onih operacija koje postoje u svim stanjima klastera iz koga se vrši pomeranje, i u svim tim stanjima su slobodne na dnu/vrhu. Samo pomeranje se vrši prema opisanim pravilima, uz proveru zavisnosti po podacima i eventualnu primenu dinamičkog preimenovanja i kombinovanja [24].

Izbor pomeranja vrši se na sledeći način. Za dati trenutni izgled PSP modela generiše se skup svih kandidata za pomeranje. Svakom pomeranju iz ovog skupa pridružuje se nekakva ocena njegove dobrote. Ove ocene se formiraju na osnovu heuristika koje će biti opisane kasnije u ovom poglavlju. Iz datog skupa se bira pomeranje sa najvišom ocenom i ono se sprovodi.

Kada se iscrpe sva moguća pomeranja koja imaju ocenu veću od neke minimalne vrednosti, vrši se proširenje modela, tako što se predikatska matrica proširi prema kriterijumima opisanim u narednom odeljku. U ovako proširenom PSP modelu stanja koja imaju proširenu matricu matrice iz osnovnog modela imaju i isti početni raspored kao i odgovarajućestanje iz osnovnog modela. Zatim se navedeni postupak ponavlja za ovaj prošireni model.

Kriterijumi za završetak celog algoritma mogu da budu različiti, ili čak kombinovani. Oni će biti opisani u narednom odeljku. Jedan uvek primenjiv kriterijum je iscrpljivanje predviđenog računarskog vremena (ili maksimalnog broja dozvoljenih koraka) celog algoritma preko kojeg se ne eli

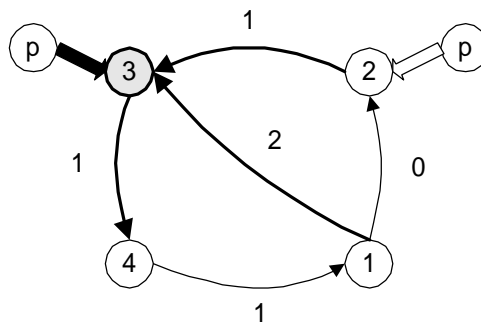
7.2.2. Proširenje predikatske matrice

Predlažu se dve strategije pri proširivanju predikatske matrice. Prva se zasniva na teorijskoj postavci svojstva te matrice ka optimalnom. Kod ove strategije potrebno je najpre u tekućem modelu izabrati konačnu stazu u grafu prelaza. Lije bi se izvršavanje pomeranjem operacija između stanja te staze skratilo. Kada se ovakva staza pronađe, treba pronaći najmanje proširenje modela u kome je moguće izvršiti pomeranja. U najgorem slučaju, postoji proširenje prilagođeno datoj stazi u kome je moguće izvršiti ova pomeranja, prema teorijskoj analizi. Ostaje za dalji rad da se precizno definiše algoritam za pronalazjenje najmanjeg zadovoljavajućeg proširenja.

Pri traženju ovakve staze treba krenuti od staza najmanje dužine (dva) pa naviše. Jedan kriterijum za kraj ove pretrage može da bude taj da staza ne prolazi više puta kroz isto stanje, osim to polazno i završno stanje na stazi mogu da budu ista. Ovo garantuje završetak pretrage, jer je najduža ovakva staza ona koja prolazi kroz sva stanja po jednom osim kroz polazno stanje dva puta, a broj stanja je inače konačan. Drugi kriterijum može jednostavno da bude maksimalna dozvoljena dužina izabrane staze.

Na ovaj način je određen i prvi kriterijum završetka globalnog postupka raspoređivanja. Ovaj globalni postupak treba završiti onda kada nije moguće pronaći odgovarajuću stazu koja se može optimizovati.

Druga strategija se zasniva na posmatranju cikličkog grafa petlje i uticaja različitih ishoda raznih uslova na izgled tog grafa, kao što je to rađeno u Glavi 4. Tamo je zaključeno da uticaj na izgled grafa imaju različite kombinacije ishoda uslova iz različitih iteracija. Posmatrajmo ciklički graf na Slici 25. Ovaj ciklički graf menja svoj oblik kada se pretpostave određeni ishodi predikata p to, kao što je pokazano, otvara mogućnost za paralelizaciju.



Slika 25: Način određivanja maksimalnog proširenja. Prikazan je ciklički graf petlje samo sa iteracionim distancama. Naglašene su grane izabranog razapinjućeg stabla.

Međutim, ovako definisan graf nosi zapravo i više informacija nego što je pogodno za optimizaciju. Naime, ako se posmatra operacija $op2[i]$, odnosno ishod predikata $p[i]$, postoji uticaj na operaciju $op3[i+1]$ i operaciju $op1[i]$. Ako se posmatra operacija $op3[i]$, odnosno ishod predikata $p[i]$, postoji uticaj na operaciju $op2[i-1]$ i operaciju $op1[i-2]$. Ako se sada posmatraju zajednički uticaj predikata $p[i]$ (vezano za $op2$) i $p[i+1]$ (vezano za $op3$), u obzir ulaze operacija $op1[i]$ (od $op2$ prema $op1$) i $op1[i-1]$ (od $op3$ prema $op1$). Ovo znatno komplikuje razmatranje uticaja više instanci predikata na graf petlje. Suština problema je u tome što ciklički graf poseduje *zatvorene puteve*, što uzrokuje da jedan isti čvor zapravo predstavlja beskonačno mnogo instanci operacija, i to različito udaljenih od nekog drugog čvora. Na primer, ako se dodaju u razmatranje i staze preko $op4$, stvar se dodatno komplikuje. Na ovaj način nije jednostavno odrediti koliko i na koji način definisati "prostiranje" pomeranja operacija na više iteracija.

Predlaže se sledeće rešenje. U cikličkom grafu treba pronaći neko razapinjuće stablo. Na tom stablu treba izabrati jedan čvor koji se nazivati korenom. Na ovaj način do svakog čvora u grafu postoji jedna i samo jedna staza kroz razapinjuće stablo koja polazi od korena. Koren će

predstavljati referentnu instancu operacije. Na primer, neka je na Slici 25 operacija *op3* koren za prikazano stablo koje ima grane 3-1, 3-2 i 3-4. Ova operacija predstavlja instancu *op3[0]*. Prolaskom kroz razapinjuaestablo, u Lvoru 4 pridru uje se instanca *op4[1]*, u Lvoru 2 pridru uje se instanca *op2[-1]*, a u Lvoru 1 instanca *op1[-2]*. Predikati predstavljaju instance *p[0]* i *p[-1]*.

Ovako postavljeno razapinjuaestablo određuje neku vrstu periode petlje. Naime, grane koje pripadaju razapinjuaan stablu trpeaizmene usled različitih ishoda predikata i pomeranja operacija. Grane van stabla predstavljaju ograničenja instanci iz jedne periode (stabla) prema instancama susedne periode. Na primer, posmatrajmo granu između Lvorova 4 i 1. Ako se posmatra operacija *op4[1]*, onda ova grana predstavlja ograničenje ove operacije prema operaciji *op1* iz naredne periode. Slično je i u obrnutom smeru. Pretpostavimo da se operacija *op1[-2]* pomera nadole. Time se iteraciona distanca grane 4-1 povećava, ali samo ako se posmatra udaljenje posmatrane *op1[-2]* od *op4* iz prethodne periode. Operacija *op4[1]* iz tekuće periode je i dalje udaljena jednu iteraciju od operacije *op1* iz naredne periode, jer se ova *op1* (mo da) uopšte nije pomerila²⁵.

Ovakva promena iteracione distance na granama van stabla može se modelovati tako što se kao iteraciona distanca koristi uređeni par dva broja: prvi predstavlja iteracionu distancu koja se uzima u obzir kada se ide *niz* granu (od *op4* iz tekuće periode prema *op1* iz naredne periode), a drugi iteracionu distancu koja se uzima u obzir kada se ide *uz* granu (od *op1* iz tekuće periode prema *op4* iz prethodne periode). Pri pomeranju neke operacije, pomeranje se zapravo odslikava na instancu operacije iz tekuće periode. Zbog toga se distance grana koje ulaze ili izlaze iz Lvora koji se pomera, a pripadaju stablu, menjaju na uobičajeni način, dok se distance grana koje ne pripadaju stablu menjaju tako da se menja samo jedna njihova komponenta. Ako se Lvor pomera nadole za *d* iteracija, i grana van stabla ulazi u Lvor, njena "uzlazna" distanca se povećava za *d*, dok se "silazna" distanca ne menja. Dualno je za izlazne grane, kao i za pomeranje nagore.

Ovakvo razmatranje može da ukaže na način ponašanja petlje pri optimizaciji i na granice prostiranja pomeranja operacija. Potrebno je, pošto se izabere razapinjuaestablo i koren, da se izračuna najveće udaljenost (u iteracijama) od korena prema svakoj operaciji koja se nalazi na jednom kraju grane, u odnosu na referentnu udaljenost operacije na drugom kraju grane koja ne pripada stablu. Na primer, za granu 4-1, operacija *op1* ima udaljenost +2 (dužina grane 1 plus referentna udaljenost 1 operacije *op4*) a operacija *op4* udaljenost -3, a za granu 1-2, operacija *op1* ima udaljenost -1, a operacija *op2* udaljenost -2. Dalje, za svaku instancu predikata *p[i]*, gde je *i* određeno kao udaljenje od korena, treba definisati opseg indeksa u predikatskoj matrici, tako što se od broja *i* oduzme redom maksimalna i minimalna udaljenost od korena dobijena na malopre opisani način. Na primer, za *p[0]* (vezan za *op3*), opseg je [-2,+3], a za *p[-1]* opseg je [-3,2]. Kako i *p[0]* i *p[-1]* (služajno) predstavljaju isti predikat, pa se oba bita u istoj vrsti predikatske matrice, ukupan opseg za ovu vrstu je [-3,+3]. Naravno, svaka vrsta predikatske matrice treba da obuhvata indeks 0, pa se opseg po potrebi proširuje tako da uključuje i 0.

Na ovaj način je određon krajnji opseg indeksa za svaki predikat u predikatskoj matrici. Taj krajnji opseg predstavlja ujedno i jedan od kriterijuma za završetak globalnog postupka raspoređivanja: postupak se završava kada se iscrpe sva pomeranja u modelu sa ovako definisanom "maksimalno širokom" predikatskom matricom. Sa druge strane, jasno je i kako treba proširiti tekućem matricu: treba proširiti one vrste u kojima nisu dostignute granice opsega indeksa datog predikata. Kriterijum izbora redosleda vrsta koje se proširuju nije za sada definisan. I ovaj algoritam iziskuje dodatne napore za rešavanje još nekih problema, kao što je, na primer, postupak u slučaju promenljive iteracione distance, kao i za eksperimentalnu evaluaciju ispravnosti cele postavke na razapinjuaestablo kao "periode" optimizacije. Ostaje, takođe, da se eksperimentalno ispita koja je od dve navedene strategije uspešnija.

²⁵Njeno pomeranje zavisi od ishoda uslova iz naredne periode.

7.2.3. Kriterijumi pomeranja operacija

Za potrebe analize u ovom odeljku, raspored jednog stanja PSP modela biće tretiran kao aciklički graf određen zavisnostima po podacima između instanci operacija. Ovaj graf ima čvorove slobodne na vrhu (u koje ne ulaze grane) i čvorove slobodne na dnu (iz kojih ne izlaze grane). Takođe, u ovom grafu postoji jedan ili više kritičnih puteva. Kako je cilj raspoređivanja smanjenje trajanja iteracije, pomeranje će se skoncentrisati na operacije sa kritičnih puteva u grafu stanja, i to one koje su slobodne na dnu (za pomeranje nadole), odnosno na vrhu (za pomeranje nagore). Operacije koje su slobodne na vrhu i pripadaju nekom kritičnom putu biće nazivane *vršnim kritičnim operacijama*, a operacije slobodne na dnu koje pripadaju nekom kritičnom putu - *krajnjim kritičnim operacijama*. Dakle, u skup potencijalnih pomeranja ulaze samo pomeranja operacija koje su slobodne na dnu i postoje u svim stanjima klastera iz kog se pomeraju. Nadalje će se takođe uzimati u obzir samo pomeranje nadole, a svi zaključci potpuno dualno mogu da se izvedu i za pomeranja nagore. Zbog toga u obzir za pomeranje dolazi samo operacija koja postoji u svim stanjima izvornog klastera, u svim tim stanjima je slobodna na dnu, a u nekom od tih stanja se nalazi na nekom kritičnom putu. Ovakve operacije, odnosno njihova pomeranja, ulaze u skup potencijalnih pomeranja.

Ocena o kojoj je bilo reči i koja se pridružuje svakom potencijalnom pomeranju posmatra se kao uređena k -torka celih brojeva. Relacija poretka ocena pri izboru najboljeg pomeranja data je relacijom leksikografskog poretka ovih k -torki. Svaki element k -torke je jedan od kriterijuma ocene pomeranja kojih može biti proizvoljno mnogo. Na ovaj način se mogu dodavati proizvoljne heuristike u odlučivanje. Leksikografski poredak zapravo uzrokuje da su kriterijumi poređani po prioritetu i da se svaki od tih kriterijuma može kvantifikovati.

Za sada se predlaže u sledećim kriterijumi predstavljani redom od najvažnijeg prema najmanje važnom, odnosno po redosledu u sledećim k -torki-oceni. Za svaki kriterijum definisana je numerička vrednost V koja je element k -torke:

1. Smanjenje ukupne dužine rasporeda stanja. Kako se iz svakog izvornog klastera uzima operacija slobodna na dnu, dužina stanja iz izvornog klastera se može smanjiti za 1 ili ostati ista (ako operacija nije na kritičnom putu ili ako ima više krajnjih kritičnih operacija). Slično, neko stanje određeni klastera se može produžiti za 1, ili njegova dužina može ostati ista. Za svako stanje izvorni i određeni klastera definiše se v kao razlika dužine rasporeda stanja pre i posle pomeranja. V se definiše kao zbir svih ovih v . Na taj način se favorizuju ona pomeranja koja više skraćuju ukupno izvršavanje. Ovakvo V predstavlja ukupni dobitak u ukupnoj dužini svih stanja.

2. Smanjenje broja krajnjih kritičnih operacija. Za svako stanje izvorni i određeni klastera definiše se v kao razlika broja krajnjih kritičnih operacija tog stanja pre i posle pomeranja. V se definiše kao zbir svih ovih v . Na taj način se među pomeranjima koja imaju isti efekat na ukupnu dužinu stanja favorizuju ona koja više smanjuju broj krajnjih kritičnih operacija. Ovo stoga može se pretpostavljati da će na taj način biti veće verovatnoće da budu pomeranja uzrokuju smanjenje dužine stanja, jer je ostao manji broj operacija koje je potrebno pomeriti da bi se došlo do skraćivanja (to su upravo krajnje kritične operacije).

3. Smanjenje broja vršnih kritičnih operacija. Za svako stanje izvorni i određeni klastera definiše se v kao razlika broja vršnih kritičnih operacija tog stanja pre i posle pomeranja. V se definiše kao zbir svih ovih v . Na taj način se među pomeranjima koja imaju isti efekat na ukupnu dužinu stanja favorizuju ona koja smanjuju broj vršnih kritičnih operacija. Ovo stoga može se pretpostavljati da će na taj način biti veće verovatnoće da budu pomeranja ne uzrokuju povećanje dužine stanja svojih određeni klastera, jer postoji veći prostor gde se mogu "smestiti" operacije pomerene nadole od kojih zavise operacije slobodne na vrhu u određeni stanju.

Ovakva postavka otvara mogućnost i da se definiše kriterijum za izbor staze prema kojoj treba proiriti predikatsku matricu, kao što je to opisano u prethodnom odeljku, po prvoj strategiji. Naime, pored skupa potencijalnih pomeranja koja se sigurno mogu izvršiti, može se formirati i skup pomeranja koja se ne mogu izvršiti samo zbog toga što data operacija ne postoji u svim stanjima

izvori nog klastera ili nije u svima njima slobodna na dnu. Za ovakva pomeranja se formiraju ocene na prikazani naLin, s tim da se ne uzimaju u obzir stanja iz kojih je operaciju nemoguæe pomeriti. Kada se izabere pomeranje sa najvi om ocenom, model se pro iruje prema onim stazama (du ine 2) koje polaze od stanja iz kojih se operacija mo e pomeriti. Na taj naLin se model pro iruje ulevo.

Ukoliko se pri rasporeðivanju koriste samo pomeranja nadole (ili samo pomeranja nagore), treba primeniti prvu strategiju pro irenja predikatske matrice opisanu u prethodnom odeljku, i obratno, ako se izabere ova strategija, vr e se samo pomeranja nadole (ili samo pomeranja nagore). Pri tom se pro irenja obavljaju samo ulevo (ili samo udesno). Ako se izabere druga strategija, potrebno je kombinovati i pomeranja nadole i pomeranja nagore, tako da obe vrste pomeranja ravnopravno ulaze u skup potencijalnih pomeranja. Pro irenja su tada u op tem sluLaju obostrana. Meðtim, u tom sluLaju treba modifikovati metod formiranja ocene, tako da u njega uð i parametri koji su prilagoðeni pomeranjima nagore. Predla e se sledeæa konstrukcija k-torke-ocene. Prva komponenta ostaje ista - ona opisuje promenu du ine rasporeda stanja. Druga komponenta treba da bude zbir V definisanog pod 2. za pomeranja nadole, i dualno definisanog V za pomeranje nagore. Ovo dualno V ukljuLuje broj vr nih kritiLnih operacija, umesto broj krajnjih kritiLnih operacija. Na sliLan naLin se defini e i treæa komponenta, kao zbir V definisanog pod 3. za pomeranje nadole i dualnog V za pomeranje nagore. Treæa komponenta postaje tako jednaka drugoj, a time i nepotrebna.

Ostaje da se detaljno eksperimentalno ispituju ove intuitivne postavke i da se eventualno predlo e izmene i dopune navedenih kriterijuma.

7.2.4. PSP model kao vrsta Markovljevog lanca

Do sada uop te nisu uzimane u obzir verovatnoæe ishoda predikata. Videæmo ovde da su se dosada nje postavke zasnivale zapravo na pretpostavci da su oba ishoda T i F svakog predikata podjednako verovatna. To u op tem sluLaju ne mora da bude tako. Pokazaæmo da informacija o dinamiLkoj verovatnoæi u toku izvr avanja programa) ishoda svakog predikata mo e da ima bitnu ulogu u rasporeðivanju i da se jako dobro uklapa u PSP model. DinamiLke informacije o ishodima mogu se prikupiti probnim izvr avanjem programa i sakupljanjem statistike (engl. *profiling*) ishoda uslovnih operacija u polaznoj petlji.

Kada se uzmu u obzir verovatnoæe ishoda predikata, PSP model se mo e smatrati posebnom vrstom Markovljevog lanca [5]. Markovljev lanac se sastoji iz odreðnog broja Lvorova (to su stanja PSP modela) izmeð kojih postoje grane (prelazi). Svakoj grani pridru en je realan broj iz opsega $[0,1]$ tako da je zbir brojeva pridru enih ulaznim granama svakog Lvora jednak 1, i isto va i za zbir brojeva pridru enih svim izlaznim granama. Zami ljenja "Lestica" kreæe se stohastiLki od Lvora do Lvora lanca, tako da iz Lvora A prelazi u Lvor B sa verovatnoæom pridru enom grani iz A u B .

Izvr avanje petlje predstavljene PSP modelom upravo predstavlja ovakav Markovljev lanac. Svako stanje predstavlja jednu iteraciju, a prelaz "Lestice" iz stanja u stanje predstavlja prelaz izvr avanja iz iteracije u iteraciju. Verovatnoæa koja se pridru uje grani iz stanja A u stanje B predstavlja zapravo uslovnu verovatnoæu da se u tekuej iteraciji dogode ishodi predikata predstavljeni matricom stanja B , pod uslovom da su se u prethodnoj iteraciji dogodili ishodi predstavljeni matricom stanja A . Ove uslovne verovatnoæe mogu da se uzmu kao statistiLki podaci dobijeni izvr avanjem polazne petlje.

Tako se svakoj grani PSP modela pridru uje verovatnoæa. Ako se posmatraju beskonaLne sekvence izvr avanja, to u praksi znaLi veoma veliki broj izvr enih iteracija petlje, mogu se izraLunati verovatnoæe svakog stanja ("boravka Lestice" u nekom Lvoru), ako se znaju verovatnoæe prelaza (grana), re avanjem sledeæeg sistema jednaLina po P_i :

$$\sum_{j=1}^N P_j p_{ji} = P_i, \quad i = 1, \dots, N$$

gde je p_{ji} verovatnoća pri prelazu iz stanja j u stanje i , P_i verovatnoća stanja i , a N broj svih stanja. Ovaj sistem je homogen, pa mu treba pridružiti i sledeće jednačine da bi imao jedinstveno rešenje:

$$\sum_{i=1}^N P_i = 1$$

Na ovaj način se dobijaju verovatnoće P svih stanja. Prosečan interval iniciranja transformisane petlje II , koji je osnovni parametar optimizacije, izražava se kao:

$$II = \sum_{i=1}^N P_i \cdot Len(S_i)$$

gde je $Len(S_i)$ dužina rasporeda stanja sa oznakom i .

Do sada se prosečan II izražavao kao:

$$II = \frac{1}{N} \sum_{i=1}^N Len(S_i)$$

to je značilo da su sva stanja podjednako verovatna, odnosno da sve grane koje izlaze iz stanja imaju jednaku verovatnoću. Ovaj kriterijum je uvršten kao prvi pri izboru pomeranja iz skupa mogućih pomeranja, kako je objašnjeno u prethodnom odeljku. Međutim, ovaj kriterijum može i da se izmeni tako da uključuje proizvoljne verovatnoće stanja. Za to treba izmeniti izražavanje V tako da se opisano v za svako stanje i pomnoži sa P_i . Na taj način se favorizuje skraćivanje onih stanja koja imaju najveću verovatnoću u izvršavanju, dok se može tolerisati i ukupno produžavanje stanja. Bitno je da se prosečni II smanjuje. Ostaje da se pokaže kako se menjaju (ili ostaju iste) verovatnoće P_i pri pročišćivanju modela.

7.2.5. Računska svojstva modela

Na kraju, dajemo kratak komentar kompleksnosti celog algoritma. Kako za sada nisu precizno definisani svi detalji algoritma, nije lako odrediti ni njegovu kompleksnost.

Globalni algoritam predstavlja petlju koja u svakom koraku pročišćuje PSP model. Kriterijum završetka ove petlje je prvi važan parametar kompleksnosti algoritma. Kako je ovaj kriterijum još uvek prilično nedefinisan, ne može se pouzdano odrediti njegova dimenzija.

Unutar ove glavne petlje izvršavaju se sva moguća pomeranja u tekućem PSP modelu. Teško je odrediti i koliko ovakvih pomeranja uopšte ima. Intuitivno je jasno da to zavisi od prirode same petlje.

Jedna primedba može da bude ta da se mnogi postupci u algoritmu zasnivaju na ispitivanju svih stanja u modelu. Kako je broj stanja uvek jednak nekom stepenu broja 2, pri čemu je taj stepen srazmeran proizvodu broja predikata i broja indeksa matrice, može se pomisliti da je PSP model intenzivno eksponencijalan. To jeste tako, ali je sreću tome da se dimenzije predikatske matrice uvek mogu kontrolisati tako da broj stanja ne pređe neku kritičnu granicu. Naime, broj predikata je retko veći od tri, a opseg indeksa matrice se veoma lako kontrolisati, tako da veći opseg daje bolje rezultate.

Uopšte, ceo algoritam se može postaviti tako da ne zahteva vreme veće od neke unapred definisane vrednosti. Suština je u tome da efekat PSP modela upravo zavisi od ove vrednosti, što je predstavljeno njegovim svojstvom te nije ka optimalnom. To je ova vrednost već PSP model može dati bolje rezultate, a ako je ona suviše mala za datu petlju, PSP će ipak dati određeni rezultat. Ostaje, naravno, da se ovi zaključci teorijski i eksperimentalno proveravaju.

8. Eksperimentalna analiza

U ovoj glavi prikazani su rezultati preliminarne eksperimentalne analize predlo enog PSP modela. Najpre je eksperimentalno potvrđeno svojstvo te nje PSP modela ka optimalnom primenom predlo enih heuristika na est izabranih petlji sa uslovnim grananjima. Ovim eksperimentom je pokazano i da predlo eni algoritam vrlo brzo konvergira, to znači da se dobri rezultati mogu postići sa malim predikatskim matricama, odnosno na modelu malih dimenzija. Drugo, predlo eni PSP model upoređen je sa tehnikom EPS [24], jedinom značajnijom tehnikom koja daje promenljiv interval iniciranja II , a time i jedinom primerenom za upoređivanje sa predlo enim modelom. Preliminarni rezultati eksperimenata sa tri realne petlje pokazuju da se PSP model pona a jednako dobro kao i EPS kada ne postoje ograničenja na resursima, a da se pri strogim ograničenjima na resursima dobijaju znatno bolji rezultati primenom PSP modela.

8.1. Uslovi eksperimenta

Prototip predlo enog PSP modela realizovan je na jeziku C++, za PC DOS platformu. Ovaj prototip ima mogućnost formiranja PSP modela za proizvoljnu petlju sa uslovnim grananjima, pomeranja operacija i pro irenja modela ulevo i udesno. Prototip nema mogućnost eliminacije proizvoljne predikcije niti generisanja koda u bilo kom obliku. Preliminarna eksperimentalna analiza sastojala se iz dva dela. U prvom delu ispitana su svojstva PSP modela pri pro irivanju. U drugom delu PSP model je upoređen sa tehnikom EPS.

8.1.1. Analiza PSP modela

Model je testiran pomoću est izabranih petlji. Tri prve petlje, označene sa S1, S2 i S3 su sintetičke; to su primeri iz ovog rada. Tri ostale petlje su realne: *bubble* je unutra nja petlja *bubble sort* algoritma, *filter* je petlja koja u jedan novi niz upisuje indekse onih elemenata datog niza koji zadovoljavaju neko jednostavno svojstvo (npr. veća su od neke konstantne vrednosti), i *merge* je glavna petlja procedure za spajanje dva sortirana niza u novi sortirani niz. Spisak ovih petlji dat je u Tabeli 1, a izvorni kod tri realne petlje na jeziku C dat je u nastavku.

```
/* Bubble sort */
for (j=0; j<i; j++)
    if (a[j]>a[j+1]) {
        temp=a[j]; a[j]=a[j+1]; a[j+1]=temp;
    }

/* Filter */
for (i=0; i<N; i++)
    if (a[i]>T) b[j++]=i;

/* Merge */
for (ia=0,ib=0; (ia<Na) && (ib<Nb)); ic++)
    if (a[ia]<=b[ib])
        c[ic]=a[ia++];
    else
        c[ic]=b[ib++];
```

Petlja	Objašnjenje
S1	Sintetička, Slika 23
S2	Sintetička, Slika 15
S3	Sintetička, Slika 17
<i>bubble</i>	<i>Bubble sort</i>
<i>filter</i>	Izbor elemenata niza koji zadovoljavaju svojstvo
<i>merge</i>	Spajanje dva sortirana niza u jedan

Tabela 1: Test benchmark petlji

Za sintetičke petlje model je formiran prema definisanim grafovima zavisnosti po podacima. Za realne petlje najpre je formiran assemblerski kod. Izabran je skup instrukcija prema arhitekturi SU-MIPS procesora [11]. Zavisnosti po podacima su ustanovljene takođe prema ovoj arhitekturi, pri čemu su za jedinicu vremena uzeta dva ciklusa takta, koliko iznosi period dohvaćanja instrukcija kod ovog procesora. Pokazuje se da je na taj način kašnjenje koje je potrebno zadovoljiti između dve zavisne operacije uvek jedna jedinica (dva takta), za bilo koju vrstu zavisnosti. Na ovaj način su se realne petlje u potpunosti uklopile u model operacija trajanja jedan ciklus, što je potrebno za primenu PSP modela. Nisu uzimane u obzir zavisnosti koje se mogu eliminisati pomoću preimenovanja, kombinovanja ili promene indeksa niza.

Ispitivano je ponašanje PSP modela pri proigravanju. Merena je prosečna dužina stanja PSP modela. Primenjivane su heuristike opisane u prethodnoj glavi. Prvi ciklus merenja se odnosio na model u kome su se vršila samo pomeranja nadole i proigravanja modela ulevo. Drugi ciklus merenja se odnosio na model u kome su se vršila samo pomeranja nagore i proigravanja udesno. Treći ciklus merenja odnosio se na model koji je proigravan u obe strane podjednako, a vršena su pomeranja i nagore i nadole. U svim slučajevima polazilo se od modela sa matricom koja ima jednu kolonu (i jednu vrstu, jer sve petlje imaju samo jednu IF strukturu), a u svakom koraku su vršena sva moguća pomeranja koja dovode do smanjenja prosečne dužine stanja, pa se tek onda vršilo proigravanje.

8.1.2. Upoređenje sa tehnikom EPS

Predloženi PSP model upoređen je sa tehnikom EPS na tri realne petlje opisane u prethodnom odeljku: *bubble*, *filter* i *merge*. Optimizacija je vršena ručno u oba slučaja. Transformacije su vršene na nivou assemblerskog jezika, pod istim uslovima opisanim u prethodnom odeljku. U obe tehnike vršene su sve transformacije koda koje mogu dovesti do boljeg rezultata: pored preimenovanja i kombinovanja, vršene su i promene indeksa pri čitanju ili upisu u niz. Promenom indeksa se postiže da operacija inkrementiranja indeksa niza i operacija pristupa nizu postanu nezavisne. Operacija pristupa nizu (*load/store*) se time nimalo ne produžava, jer procesor poseduje registarsko indirektno adresiranje sa pomerajem, pa se promenom indeksa menja samo pomeraj u instrukciji. Za generisanje koda korišćen je model IBM VLIW mašina koja izvršava instrukcije predstavljene u obliku stabla [24].

Ispitivano je ponašanje obe tehnike u dva slučaja. Prvi slučaj je bez ograničenja u resursima. Ova pretpostavka je sasvim realna, jer su posmatrane petlje jednostavne, imaju mali broj operacija, pa i potpuno paralelizovan kod koristi istovremeno mali broj resursa. Ukoliko procesor poseduje visok nivo paralelizma, ova pretpostavka je u praksi zadovoljena.

U drugom slučaju su postavljena relativno stroga ograničenja u resursima. Ova ograničenja su formirana u odnosu na broj i vrste operacija koje data petlja poseduje, tako da se može posmatrati uticaj ograničenja na optimizaciju. Kod PSP modela primenjivana su najpre pomeranja operacija prema opisanim heuristikama, a zatim je vršeno raspoređivanje operacija dobijene iteracije uz poštovanje ograničenja na resursima (*List scheduling*). Pri ovom raspoređivanju viši prioritet su

imale operacije sa kritičnog puta u grafu zavisnosti iteracije. U Tabeli 2 prikazana su ograničenja u resursima koja su pretpostavljena za svaku posmatranu petlju, za obe tehnike podjednako.

Petlja	Ograničenje
<i>bubble</i>	1 x ld, 1 x st, 1 x br, total 3 ops
<i>filter</i>	ld+st=1, 1 x br, total 2 ops
<i>merge</i>	1 x ld, 1 x st, 1 x br, total 3 ops

Tabela 2: Ograničenja u resursima za tri posmatrane realne petlje. Oznake: 1 x ld - jedna *load* operacija po instrukciji; 1 x st - jedna *store* operacija po instrukciji; ld+st=1 - ukupno jedna *load* ili *store* operacija po instrukciji; 1 x br - jedna *branch* (uslovna) operacija po instrukciji; total n ops - ukupno n operacija po instrukciji.

8.2. Rezultati eksperimenta

Rezultati dobijeni u eksperimentima opisanim u prethodnom odeljku prikazani su ovde tabelarno i grafički. Najpre su prikazani rezultati ispitivanja ponašanja PSP modela pri proirivanju, a zatim i rezultati upoređivanja PSP modela sa tehnikom EPS.

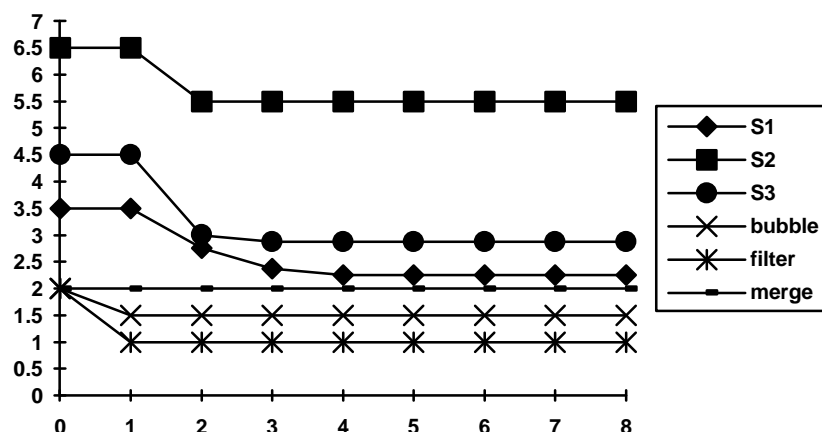
8.2.1. Analiza PSP modela

U nastavku su dati rezultati posmatranja ponašanja PSP modela pri proirivanju. U tabelama su date vrednosti prosečne dužine stanja u modelu. U prvoj koloni su date početne vrednosti neoptimizovane petlje, a u ostalim kolonama vrednosti posle pomeranja, za razne vrednosti irine predikatske matrice.

U Tabeli 3 i na Slici 26 prikazani su tabelarno i grafički rezultati eksperimenta za slučaj pomeranja samo nadole i proirivanja modela samo ulevo. U Tabeli 4 i na Slici 27 prikazani su tabelarno i grafički rezultati eksperimenta za slučaj pomeranja samo nagore i proirivanja modela samo udesno. U Tabeli 5 i na Slici 28 prikazani su tabelarno i grafički rezultati eksperimenta za slučaj pomeranja i nadole i nagore, i proirivanja modela u obe strane.

Petlja	Poč.	irina predikatske matrice							
		1	2	3	4	5	6	7	8
S1	3.5	3.5	2.75	2.375	2.25	2.25	2.25	2.25	2.25
S2	6.5	6.5	5.5	5.5	5.5	5.5	5.5	5.5	5.5
S3	4.5	4.5	3	2.875	2.875	2.875	2.875	2.875	2.875
<i>bubble</i>	2	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
<i>filter</i>	2	1	1	1	1	1	1	1	1
<i>merge</i>	2	2	2	2	2	2	2	2	2

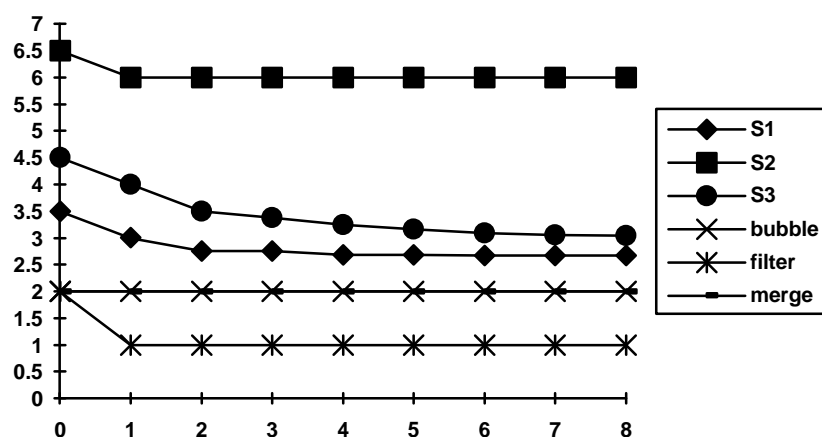
Tabela 3: Rezultati optimizacije pomoću PSP modela za slučaj pomeranja nadole i proirivanja ulevo. Date vrednosti predstavljaju prosečnu dužinu stanja u modelu. U prvoj koloni date su početne vrednosti neoptimizovane petlje, a u ostalim kolonama vrednosti posle pomeranja, za razne vrednosti irine predikatske matrice.



Slika 26: Rezultati optimizacije pomoću PSP modela za slučaj pomeranja nadole i proirivanja ulevo. Date vrednosti na apscisi predstavljaju širinu predikatske matrice, a na ordinati prosečnu dužinu stanja u modelu. Prva vrednost 0 na apscisi predstavlja početno (neoptimizovano) stanje petlje.

Petlja	Poč.	širina predikatske matrice							
		1	2	3	4	5	6	7	8
S1	3.5	3	2.75	2.75	2.688	2.688	2.672	2.672	2.668
S2	6.5	6	6	6	6	6	6	6	6
S3	4.5	4	3.5	3.375	3.25	3.156	3.094	3.055	3.039
<i>bubble</i>	2	2	2	2	2	2	2	2	2
<i>filter</i>	2	1	1	1	1	1	1	1	1
<i>merge</i>	2	2	2	2	2	2	2	2	2

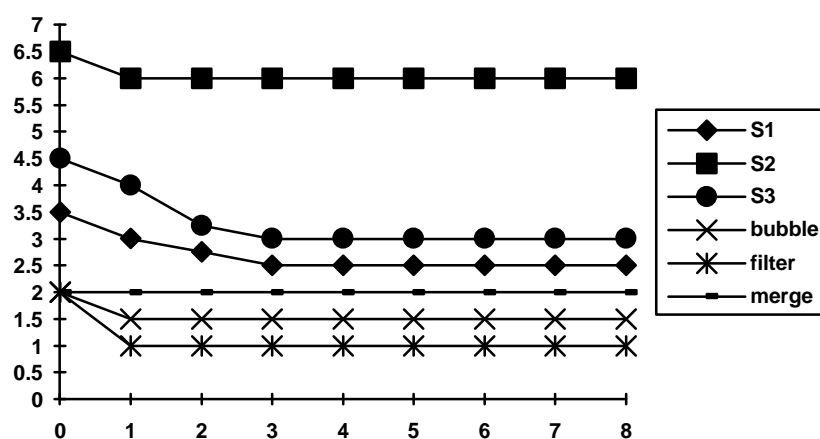
Tabela 4: Rezultati optimizacije pomoću PSP modela za slučaj pomeranja nagore i proirivanja udesno. Date vrednosti predstavljaju prosečnu dužinu stanja u modelu. U prvoj koloni date su početne vrednosti neoptimizovane petlje, a u ostalim kolonama vrednosti posle pomeranja, za razne vrednosti širine predikatske matrice.



Slika 27: Rezultati optimizacije pomoću PSP modela za slučaj pomeranja nagore i proirivanja udesno. Date vrednosti na apscisi predstavljaju širinu predikatske matrice, a na ordinati prosečnu dužinu stanja u modelu. Prva vrednost 0 na apscisi predstavlja početno (neoptimizovano) stanje petlje.

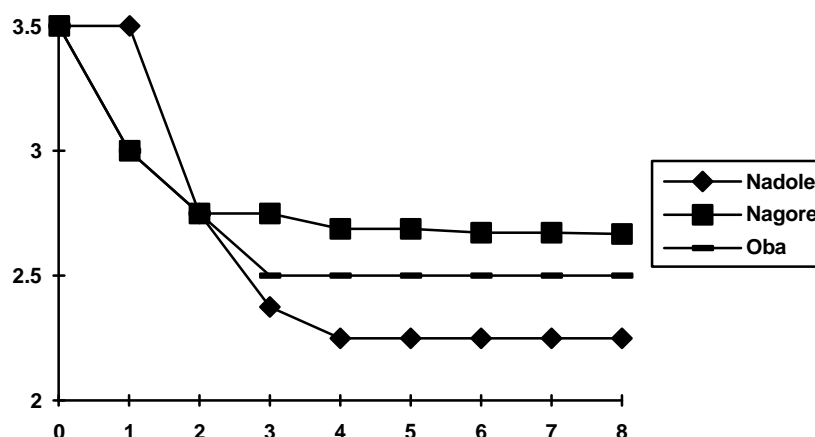
Petlja	PoŁ.	irina predikatske matrice							
		1	2	3	4	5	6	7	8
S1	3.5	3	2.75	2.5	2.5	2.5	2.5	2.5	2.5
S2	6.5	6	6	6	6	6	6	6	6
S3	4.5	4	3.25	3	3	3	3	3	3
<i>bubble</i>	2	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
<i>filter</i>	2	1	1	1	1	1	1	1	1
<i>merge</i>	2	2	2	2	2	2	2	2	2

Tabela 5: Rezultati optimizacije pomoću PSP modela za slučaj pomeranja i nadole i nagore, i proirivanja u obe strane. Date vrednosti predstavljaju prosečnu dužinu stanja u modelu. U prvoj koloni date su početne vrednosti neoptimizovane petlje, a u ostalim kolonama vrednosti posle pomeranja, za razne vrednosti irine predikatske matrice.



Slika 28: Rezultati optimizacije pomoću PSP modela za slučaj pomeranja i nadole i nagore, i proirivanja u obe strane. Date vrednosti na apscisi predstavljaju irinu predikatske matrice, a na ordinati prosečnu dužinu stanja u modelu. Prva vrednost 0 na apscisi predstavlja početno (neoptimizovano) stanje petlje.

Radi upoređivanja sve tri strategije, na Slici 29 su prikazani rezultati za petlju S1, kod koje se uočavaju najzanimljivije razlike.



Slika 29: Upoređenje sve tri strategije: pomeranja nadole, pomeranja nagore i pomeranja u oba smera, za petlju S1.

Posmatranjem rezultata optimizacije petlje *merge* primećeno je jedan nedostatak sada nje implementacije PSP tehnike. Naime, kod ove petlje nije postignut nikakav dobitak ni u jednom slučaju, iako se jednostavno može postići prosek od jednog ciklusa po stanju. Naime, kod ove petlje postoje dve operacije koje su vrhne kritične i dve koje su krajnje kritične, a dužina kritičnog puta je 2. Pomeranje samo jedne, bilo koje vrhne ili krajnje kritične operacije pojedinačno ne donosi nikakvo poboljšanje, jer ostaje i dalje kritičan put dužine 2. Tek pomeranje obe vrhne ili obe krajnje kritične operacije istovremeno dovodi do poboljšanja. Zbog toga u sadašnjoj implementaciji, koja uzima u obzir pomeranje samo jedne operacije, nije došlo do željenog rezultata. Ova pojava ukazuje i na jednostavno unapređenje predložene tehnike: potrebno je analizirati efekat pomeranja svih vrhnih/krajnjih kritičnih operacija jednog stanja odjednom. Ovo poboljšanje je uvedeno pri optimizaciji *merge* petlje u narednom odeljku, kod upoređivanja sa EPS tehnikom. Treba uočiti da je implementacija ovog unapređenja sasvim jednostavna.

8.2.2. Upoređenje sa tehnikom EPS

U Tabeli 6 prikazani su rezultati optimizacije tri realne petlje *bubble*, *filter* i *merge* pomoću tehnika EPS i PSP za slučaj bez ograničenja u resursima, a u Tabeli 7 za slučaj ograničenja u resursima. Date vrednosti predstavljaju dužinu iteracije za jedan i drugi ishod uslova. Kako su rezultati obe tehnike za prvi slučaj isti, na Slici 30 su grafički prikazane prosečne vrednosti za obe tehnike, za slučaj ograničenja u resursima.

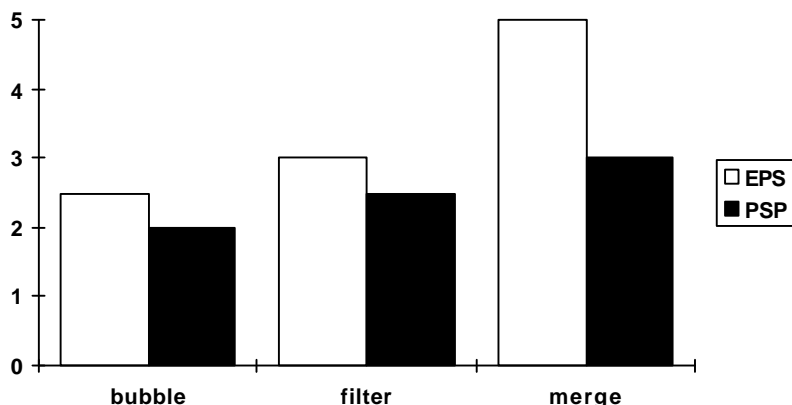
Petlja	EPS	PSP
<i>bubble</i>	1+2	1+2
<i>filter</i>	1+1	1+1
<i>merge</i>	1+1	1+1
Prosek	1.33	1.33

Tabela 6: Rezultati optimizacije pomoću EPS tehnike i PSP modela za slučaj nepostojanja ograničenja u resursima. Date vrednosti predstavljaju dužinu iteracije za jedan i drugi ishod uslova.

Petlja	EPS	PSP
<i>bubble</i>	2+3	2+2
<i>filter</i>	3+3	2+3
<i>merge</i>	5+5	3+3

Prosek	3.5	2.5
--------	-----	-----

Tabela 7: Rezultati optimizacije pomoću EPS tehnike i PSP modela za slučaj postojanja ograničenja u resursima. Date vrednosti predstavljaju dužinu iteracije za jedan i drugi ishod uslova.



Slika 30: Upoređnje tehnika EPS i PSP za tri realne petlje, za slučaj postojanja ograničenja u resursima. Prikazane su dobijene prosečne vrednosti dužine iteracije.

8.3. Zaključak eksperimenta

U ovom poglavlju navešćemo zaključke koji se mogu izvesti iz dobijenih rezultata eksperimenata. Najpre će biti izneseni zaključci u vezi sa analizom ponašanja PSP modela, a zatim i oni u vezi sa upoređenjem PSP modela sa tehnikom EPS.

8.3.1. Analiza PSP modela

Prvi zaključak koji se može izvesti iz eksperimenta koji je ispitivao ponašanje PSP modela pri proćirivanju je da se zaista postižu bolji rezultati kada se model proćiri, pod uslovom da bolje rećenje postoji. Ovo se u potpunosti uklapa sa teorijskom analizom, kao i sa intuitivnim oćekivanjima.

Mećutim, ostali zaključci su još bitniji za praktićnu primenu predloćenog modela. Prvo, primećuje se da PSP model vrlo brzo teći ka vrednostima koje se postižu u "zasićenju". Moće se rećalati se za sve posmatrane petlje, a naroćito za one realne, postižu zasićenja za matrice ćirine 1 do 2, a najviše 4. Ovo je izuzetno znaćajno, jer se time kompleksnost algoritma svodi na prihvatljiv nivo.

Drugo, moće se primetiti da strategija pomeranja nadole i proćirenja ulevo uglavnom daje bolje rezultate za veće ćirine matrice od strategije pomeranja nagore i proćirenja udesno. Strategija obostranog proćirenja je negde izmeću. Za neke petlje strategija pomeranja nagore za male ćirine matrice (jedan) daje bolje rezultate. Za realne petlje ove razlike su daleko manje znaćajne. Ova pojava moće se objasniti strukturom petlji: one poseduju najpre niz zavisnosti po podacima operacija koje ne zavise od uslova, a zatim i niz operacija koje zavise od uslova. Kontrolno nezavisne operacije se mogu lako pomerati nagore za male ćirine matrice, dok se one kontrolno zavisne mogu pomerati nadole tek posle proćirivanja matrice.

Ovakav zaključak ukazuje na dalje pravce poboljšanja heuristika: pomeranja nagore treba vrćiti u poćetku, kada je matrica male ćirine, a zatim, posle proćirivanja, treba vrćiti i pomeranja

nadole. Najzad, ponovimo zaključak koji je već iznesen: potrebno je poboljšati tehniku tako što se u obzir uzimati efekat pomeranja viših ili krajnjih kritičnih operacija nekog stanja odjednom, a ne samo pojedinačno.

8.3.2. Upoređenje sa tehnikom EPS

Za posmatrane realne petlje i slučaj nepostojanja ograničenja u resursima obe tehnike su dale potpuno iste rezultate. Ovo se može objasniti time što su sve posmatrane petlje krajnje jednostavne: one poseduju samo po jednu do dve međiteracione zavisnosti, i to uvek distance jedan (ili promenljive distance po lev od jedan). Dužine kritičnih puteva su za sve petlje male i iznose dva ciklusa. Zbog toga se obe tehnike vrlo brzo zaustavljaju na istom rešenju koje se postiže preklapanjem samo dve susedne početne iteracije.

Za slučaj strogih ograničenja u resursima, predloženi PSP model se znatno bolje ponaša nego tehnika EPS. Uzrok je u tome što kod tehnike EPS stroga ograničenja u resursima sprečavaju neke operacije da pređu u susednu iteraciju, iako bi se u njoj našlo "prostora" za njihovo izvršavanje. Posmatranje ponašanja EPS tehnike dovelo je do sledećeg zaključka: ova tehnika je prvobitno konstruisana za slučaj van petlji i njemu je i prilagođena. Za slučaj van petlji ova tehnika eliminiše mnoge nedostatke drugih globalnih tehnika. Međutim, za slučaj petlji EPS nije "prirodan": za razliku jednostavnog prilagođavanja globalne strategije optimizaciji koda petlji, gubljeno je iskorišćavanje osnovnog izvora paralelizma u petljama - preklapanje operacija iz različitih, udaljenih iteracija.

Predloženi PSP model nema ove nedostatke. Pretpostavljamo da se za slučaj umerenih ograničenja u resursima ove dve tehnike ponašaju približno, i daju rezultate negde između datih vrednosti za krajnje slučajeve. Ipak, može se očekivati da za petlje koje imaju složenije međiteracione zavisnosti PSP model daje znatno bolje rezultate nego tehnika EPS.

Naravno, sigurno je da se mogu pronaći petlje i uslovi ograničenja u resursima pri kojima EPS daje bolje rezultate nego PSP, pri čemu se može izraziti prednost globalne strategije EPS u odnosu na *List scheduling*. Verujemo da su ovi slučajevi ipak ređi.

9. Pravci budućeg rada

U dosadašnjem izlaganju izloženi su mnogi još nerešeni problemi vezani za teorijsku analizu i praktičnu primenu predloženog PSP modela. U ovoj glavi bit će jednom navedeni svi važniji otvoreni problemi, kao smernice za buduću rad.

9.1. Potpuna teorijska analiza modela

PSP model najpre zahteva potpuniju teorijsku analizu. Ova analiza uključuje sledeća pitanja. Potrebno je najpre još jednom proveriti ispravnost cele postavke svojstva te nje ka optimalnom i eventualno predložiti alternative.

Drugo, potrebno je postaviti opšti slučaj PSP modela prilagođenog proizvoljno zavisnim (ugneđenim) IF strukturama u telu petlje, kao što je to prikazano u 7.1.2. Ovakav opšti model bi se od predloženog u osnovi razlikovao samo po tome što ne- b vrednosti matrice stanja za svaku vrstu mogu da budu iz proizvoljnog konačnog skupa. Potrebno je ispitati sva navedena svojstva za ovako uopšten model.

Dalje, važno je razrešiti pitanja vezana za eliminaciju proizvoljne predikcije koja su opisana u 6.4.2. To je, kao prvo, pitanje da li se predloženi postupak eliminacije uvek završava. Uticaj na ovo imaju, da podsetimo, pomeranja operacija nadole, jer se tako pomeraju eventualno i uslovne operacije i time kasne trenuci razrešenja predikata, a i dodatno menjaju rasporedi određeni stanja, čime se unose nove razlike u dužinama koje treba eliminisati. Takođe je pitanje da li se proširenjima modela uveliko utiče na konačnost algoritma. Sledeće pitanje je da li se i koliko navedenim postupkom gubi na prosečnoj dužini rasporeda modela.

Naredno otvoreno polje proučavanja je posmatranje PSP modela kao Markovljevog lanca, kao što je opisano u 7.2.4. Moguće je da ovakvo posmatranje otvara nove probleme, ali i mogućnosti PSP modela.

Najzad, otvoreno teorijsko pitanje je kompleksnost celog modela, odnosno njemu pridruženog algoritma, što je diskutovano u 7.2.5.

9.2. Definisanje svih elemenata raspoređivanja

Čini se da je problem raspoređivanja od svih otvorenih problema najlakši. U 7.2. su predložene neke heuristike za izbor operacija za pomeranje, dok se nove heuristike mogu praktično neograničeno dodavati, a predložene poboljšavati.

Jedna heuristika koja nije ovde razmatrana, a potrebna je, treba da uzima u obzir spekulativnost operacija, kao što je to navedeno u [24]. Pri izboru treba favorizovati pomeranja uslovnih operacija nagore, ili pomeranja operacija na kritičnom putu do uslovnih operacija nagore, ili pomeranja ostalih operacija tako da budu što manje spekulativne. Ovaj poslednji kriterijum zateva definisanje postupka kojim bi se utvrđivao nivo spekulativnosti svake operacije. Dobro bi bilo kad bi se ovaj nivo izražavao iterativno, kao u [24].

Daleko složeniji problem je prilagođenje raspoređivanja služaju ograničenih resursa. Ovaj problem diskutovan je u 7.1.3. Potrebno je predložiti tehniku koja bi iterativno ispitivala konflikte na resursima tokom samog pomeranja operacija, ali tako da se ne izgubi na kvalitetu rezultujućeg koda u odnosu na predloženu strategiju.

Dalja poboljšanja mogu da se predlože i za kriterijume i strategije proširenja PSP modela opisane u 7.2.2.

9.3. Definisanje svih detalja generisanja koda

Ostaje da se detaljno formuliše algoritam generisanja konačnog koda iz završnog PSP modela. Problemi vezani za ovaj algoritam opisani su u 7.1.5. Pitanja koja ostaju nerešena vezana su za način generisanja spekulativnih i kompenzacionih operacija i operacija koje nisu spekulativne, odnosno pripadaju nekoj grani IF strukture koja se može dovoljno rano razrešiti.

Važno otvoreno pitanje je kako razrešiti slučaj kada je jedna ista uslovna operacija raspoređena u različite cikluse u stanjima koja se razlikuju samo prema ishodu te iste operacije. Pitanje je i da li se uopšte ovaj slučaj može dogoditi. Sve ovo zahteva detaljnu teorijsku i praktičnu analizu.

Problem generisanja pretpetlje i postpetlje ostaje takođe potpuno otvoren.

9.4. Hardverska podrška

Pravilnost PSP modela otvara široke mogućnosti za delimičnu ili potpunu hardverizaciju. Ovde ćemo samo na kratko opisati neke ideje za moguću hardversku podršku izvršavanju petlje koja je optimizovana primenom PSP modela. Najpre ćemo izneti neke ideje za konstrukciju arhitekture koja direktno podržava PSP model, a zatim i predloge za iskoriscavanje postojećih arhitekture - prediktora grananja u procesoru.

9.4.1. Arhitekture prilagođene PSP modelu

Izvršavanje petlje predstavljene PSP modelom, kao što je pokazano, može se posmatrati kao funkcionisanje konačnog automata koji u svakom koraku prelazi iz trenutnog u naredno stanje. Svako stanje sadrži skup operacija koje treba izvršiti. Iz svakog stanja se može predefinisati određeni broj novih stanja (ne u sva postojeća), to je određeno ishodima instanci predikata koji se nalaze na desnoj ivici predikatske matrice. Prema tome, pri svakom prelasku se tretiraju ishodi samo m uslovnih operacija, onoliko koliko ih ima u polaznoj petlji.

Prva ideja koja se nameće je da se ovakav konačni automat implementira u hardveru. Pitanje je kako to treba izvesti i kakav bi efekat bio. Problem je kako odrediti tačan prelazak ako svi ishodi na desnoj ivici matrice nisu razrešeni. Drugim rečima, za slučaj spekulativnog izvršavanja, ovakav automat je nedeterministički. Nedeterminizam automata zapravo predstavlja postojanje više niti izvršavanja (engl. *threads*), sve dok se ne razreše potrebni uslovi, kada se broj niti redukuje.

Preciznije, automat se u svakom trenutku nalazi zapravo u nekoliko stanja koja predstavljaju različite niti izvršavanja. U odnosu na ishode uslovnih operacija izvršenih u tim stanjima, neka od stanja se odbacuju kao pogrešne pretpostavke, odnosno pogrešne niti izvršavanja. Iz onih stanja koja preostaju, prelazi se u nova stanja, opet nedeterministički, jer nove instance predikata nisu razrešene.

Problem kod ovakve postavke predstavlja korišćenje po jednog nezavisnog skupa resursa (registara i funkcionalnih jedinica) za svako stanje, bez obzira da li se to stanje zadržava ili odbacuje. Ovo je praktično pandam spekulativnom izvršavanju operacija, samo što se spekulativno izvršavaju cele iteracije. Nedostatak je u tome što se može da neka operacija izvršava u više trenutnih stanja istovremeno, što znači da ona nije spekulativna, a koristi resurse u svim tim stanjima (nitima izvršavanja). Drugi problem je kako dimenzionisati broj stanja (skupova resursa) i broj mogućih prelaza koji inače zavisi od dimenzija predikatske matrice PSP modela, odnosno od prirode same petlje.

Osim ovakve "maksimalističke" postavke arhitekture prilagođene PSP modelu, mogu se proučavati i samo delovi koji u klasičnoj arhitekturi olakšavaju izvršavanje. Pretpostavka je da je kod generisan za klasičnu mašinu, kao što je to u ovom radu činjeno. U tom slučaju, moguće je da se u transformisanoj petlji pojavljuju uslovne (IF) konstrukcije koje ispituju složenije uslove

(konjunkcije) vi e instanci predikata iz razliĹitih iteracija. Kada bi se ove konjunkcije realizovale kao klasiĹne operacije, a instance predikata iz prethodnih iteracija pamtile u registrima op te namene, do lo bi do prevelike potro nje vremena i funkcionalnih jedinica (za izraĹunavanje slo enih uslova), kao i registara.

Kao re enje se predla e namenski registarski fajl za pam e i ishoda predikata, kao to je to u [30] ili [37]. Ovaj registarski fajl bi posedovao mali broj registara, onoliko koliko se oĹekuje da postoji uslovnih operacija u petlji (npr. 4). Svaki registar bi bio pomeraĹki, sa onoliko razreda kolika se daljina "gledanja unazad" oĹekuje (pribli no broj indeksa predikatske matrice, npr. 8). Svakoj uslovnoj operaciji u petlji pridru en je jedan ovakav registar, tako da uslovna operacija upisuje svoj binarni rezultat u odgovaraju e krajnji bit svog registra. Na kraju svake iteracije, prevodilac generi e posebnu instrukciju koja pomera sve ove predikatske registre za jedno mesto, Ĺime se zapravo obezbeĹuje da jedan predikatski registar Ĺuva istoriju ishoda jednog predikata.

Da bi se jednostavno izvr io uslovni skok u odnosu na vrednosti vi e instanci predikata iz razliĹitih iteracija, potrebno je na izlaze predikatskih registara povezati kombinacionu mre u Ĺija se funkcija mo e "programirati" pomo e posebnog upravljaĹkog registra. Ovaj upravljaĹki registar je adresibilan i prevodilac pre ulaska u petlju obezbeĹuje instrukciju koja u ovaj registar upisuje potrebnu vrednost. Mogu e je da postoji i vi e ovakvih upravljaĹkih registara, za razliĹite kombinacije instanci predikata. Uslovni skok se tako obavlja u odnosu na izlaz kombinacione mre e "programirane" prema upravljaĹkom registru koji je specifikovan samom instrukcijom uslovnog skoka. Mo e se obezbediti i da uslovni skok bude razgranati, na vi e odredi nih adresa.

9.4.2. Veza sa prediktorima grananja

Jedan od osnovnih problema u optimizaciji koda pomo e PSP modela, kao to je vi e puta isticano, predstavlja pretpostavka o mogu eosti proizvoljne predikcije. Algoritam koji je predlo en ima za cilj da u vreme prevoĹnja elimini e eventualne posledice pogre ne pretpostavke o budu e ishodima predikata.

MeĹtim, ovaj isti problem odavno je re avan hardverski - prediktorima grananja. Naime, uloga hardverskih prediktora grananja je upravo da u vreme izvr avanja programa obezbede to manju verovatno e gre ke u pretpostavkama o budu e ishodima predikata. Dana nji prediktori posti u izuzetne rezultate, Ĺak i preko 95% pogodaka.

Ova Ĺinjenica se mo e iskoristiti u implementaciji PSP modela. Naime, PSP model se prirodno zasniva na proizvoljno "dalekoj" pretpostavci o ishodima budu e predikata. Pretpostavke o budu e ishodima ugraĹne su zapravo u matrice stanja PSP modela. Ove informacije se ve aalaze u internoj memoriji prediktora grananja. Slo eniji prediktori imaju mogu eost predviĹanja vi e uslovnih operacija, i to vi e instanci unapred. Tako se pretpostavke o ishodima predikata mogu iskoristiti u izvr avanju PSP modela da bi se umesto vi e niti, kao to je opisano u prethodnom odeljku, pratila samo jedna najverovatnija (ili samo nekoliko najverovatnijih). Problem je kako u sluĹaju proma aja pretpostavke restaurirati prethodno stanje petlje i kakav tetan efekat to ima po ukupno vreme izvr avanja.

10. Zaključak

U radu je razmatran problem optimizacije koda programskih petlji sa uslovnim grananjima u kontekstu softverske protočnosti. Najpre je problem definisan i objašnjeni su razlozi za njegovu analizu. Zatim su prikazani neki postojeći načini njegovog rešavanja.

Potom je problem detaljno analiziran, u smislu pronalaska izvora paralelizma koje u petlji unose uslovna grananja. Ovi izvori su znatno rari nego kod petlji bez uslovnih grananja, ali i znatno složeniji za pronalazak i iskoriscenje. Formirani su određeni zaključci koji govore o izvorima paralelizma. Ponovljenom ovde samo tri najvažnija. Prvi je taj da se zbog dejstva uslovnih operacija graf zavisnosti po podacima menja tako što se određene operacije gube iz grafa, ili iteracione distance nekih grana povećavaju. Kao posledica ovoga, drugi zaključak je taj da se na paralelizmu dobija dupliranjem različitih instanci iste operacije iz različitih iteracija polazne petlje, u istoj iteraciji transformisane petlje. Ove višestruke instance su kontrolno zavisne od različitih instanci predikata. Treći zaključak govori da važan uticaj na paralelizam imaju složenije kombinacije (konjunkcije) višestrukih instanci jednog ili višestrukih predikata iz višestrukih iteracija, ali ne samo susednih iteracija, već i onih koje su svojstvene iteracionim distancama grana u cikličnom grafu zavisnosti petlje.

Predloženo je zatim originalni način modelovanja petlji sa uslovnim grananjima u kontekstu softverske protočnosti. Model je nazvan PSP modelom i zasniva se na posmatranju izvršavanja petlje kao vrste konačnog automata. Automat ima određen broj stanja. Svako stanje je opisano matricom stanja. Svaka vrsta ove matrice predstavlja ishode jedne uslovne operacije u nekoliko susednih iteracija. Između stanja automata postoje prelazi koji odslikavaju "napredovanje" ishoda predikata; opisno rečeno, matrica sledbeničkog stanja dobija se od matrice prethodničkog stanja pomeranjem za jedno mesto (ulevo). Svakom stanju je pridružen raspored operacija koje se izvršavaju u jednoj iteraciji pod pretpostavkom da su ishodi predikata onakvi kako je to opisano matricom tog stanja. Definisana su pomeranja operacija: operacija se može pomeriti nadole ako postoji kao slobodna na dnu u svim prethodničkim stanjima; ovakva operacija se pomera u sva sledbenička stanja. Analogno je za pomeranje nagore. Pri pomeranju operacija menja svoj indeks za jedan. Ova pomeranja pružaju mogućnost za optimizaciju petlje.

Predloženi PSP model je formalno matematički postavljen. Uočena su i dokazana mnoga zanimljiva svojstva modela. Predložena je definicija svojstva te njegova optimalnost i to svojstvo dokazano za PSP model. Svojstvo se zasniva na prodirivanju PSP modela koje se sastoji u prodirivanju matrice modela tako da obuhvata ishode predikata višestrukih susednih iteracija. Može se reći da najveću vrednost predloženiog PSP modela leži i upravo u njegovoj suptilnoj jednostavnosti i pravilnosti, koje omogućavaju njegovu formalnu postavku.

Dalje su razmatrani problemi vezani za praktičnu implementaciju PSP modela u cilju optimizacije koda petlji. Predložene su heuristike za izbor operacija za pomeranje. Komentarisani su i neki delimično rešeni ili još nerešeni problemi vezani za generisanje koda dobijenog pomoću PSP modela.

Izvršena je i početna eksperimentalna analiza predloženiog rešenja. Osnovni zaključak je da PSP model postiže dobre rezultate i pri malim dimenzijama predikatske matrice. PSP model je upoređen sa tehnikom EPS. Za slučaj kada ne postoje ograničenja u resursima, obe tehnike daju iste rezultate. Kada postoje stroga ograničenja u resursima, PSP model je znatno efikasniji.

Najzad, opisani su otvoreni problemi vezani za PSP model. To su najpre teorijski problemi u vezi sa još nekim nedokazanim svojstvima PSP modela. Ukazane su smernice za formiranje novih heuristika raspoređivanja, kao i za rešavanje problema generisanja koda. Predložene su na kraju i neke ideje za hardversku podršku izvršavanja petlje transformisane pomoću PSP modela.

Smatramo da predloženi model nudi široku mogućnost daljeg istraživanja i primene u optimizaciji koda. Rezultate ovog rada mogu da iskoriste svi oni koji se bave istraživanjima u ovoj

oblasti, kao i oni koji se bave konstrukcijom prevodilaca za superračunare. Ako se ovaj model i ne primeni odmah, verujemo da će originalni formalni pristup celom problemu, kao i mnoga dokazana svojstva i zaključci iz ovoga rada pokrenuti dalje analize koje će dovesti do efikasnijeg rešavanja problema optimizacije koda programskih petlji sa uslovnim grananjima.

Reference

- [1] Bacon D. F., Graham S. L., Sharp O. J., "Compiler Transformations for High-Performance Computing", *ACM Computing Surveys*, Vol. 26, No. 4, December 1994, pp. 345-420
- [2] Aho A., Sethi R., Ullman J., *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA, USA, 1986.
- [3] Aiken A., Nikolau A., "Optimal Loop Parallelization", *Proceedings of The SIGPLAN 1988 Conference on Programming Language Design and Implementation*, June 1988, pp. 308-317
- [4] Chou H.-C., Chung C.-P., "An Optimal Instruction Scheduling for Superscalar Processor", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 3, March 1995
- [5] Cvetković D., *Diskretne matematičke strukture*, Naučna knjiga, Beograd 1987.
- [6] De Gloria A., Faraboschi P., Olivieri M., "A Non-Deterministic Scheduler for a Software Pipelining Compiler", *Proceedings of the 25th Annual International Symposium on Microarchitecture (MICRO-25)*, December 1992, pp. 41-44
- [7] Ebcioglu K., "A Compilation Technique for Software Pipelining of Loops With Conditional Jumps", *Proceedings of the 20th Annual Workshop on Microprogramming (MICRO-20)*, December 1987, pp. 69-79
- [8] Ebcioglu K., Nakatani T., "A New Compilation Technique for Parallelizing Loops with Unpredictable Branches on a VLIW Architecture", *Second Workshop on Languages and Compilers for Parallel Computing*, 1989
- [9] Eichenberger A. E., Davidson E. S., Abraham S. G., "Minimum Register Requirements for a Modulo Schedule", *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO-27)*, December 1994, pp. 75-84
- [10] Gasperoni F., "Compilation Techniques for VLIW Architectures", Technical report #435, New York University, Computer Science Dept., NY, USA, March 1989
- [11] Gill J., Gross T., Hennessy J., Jouppi N., Przybylski S., Rowen C., "Summary of MIPS Instructions", Technical Note No. 83-237, Stanford University, Stanford, CA, USA, November 1983
- [12] Govindarajan R., Altman E. R., Gao G. R., "Minimizing Register Requirements under Resource-Constrained Rate-Optimal Software Pipelining", *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO-27)*, December 1994, pp. 85-94
- [13] Jones R. B., Allan V. H., "Software Pipelining: An Evaluation of Enhanced Pipelining", *Proceedings of the 24th Annual International Symposium on Microarchitecture (MICRO-24)*, November 1991, pp. 82-92
- [14] Jovanović Z., "Software Pipelining of Loops by Pipelining Strongly Connected Components", *Proceedings of the 21st Hawaii International Conference on System Sciences*, 1991

-
- [15] Jovanović Z., Petković D., Milić D., *Paralelni računarski sistemi - Instrukcijski nivo paralelizma*, u pripremi, Beograd, 1996.
- [16] Lam M. S., Wilson R. P., "Limits of Control Flow on Parallelism", *Proceedings of 19th Annual International Symposium on Computer Architecture (ISCA)*, May 1992, SIGARCH Comp. Arch. News, Vol. 20, No. 2, pp.46-57
- [17] Lam M., "Software Pipelining: An Effective Scheduling Technique for VLIW Machines", *Proceedings of The SIGPLAN 1988 Conference on Programming Language Design and Implementation*, June 1988, pp. 318-328
- [18] Lam M., *A Systolic Array Optimizing Compiler*, PhD thesis, Carnegie Mellon University, Pittsburg, PA, USA, 1987
- [19] Larus J. R., "Loop-level Parallelism in Numeric and Symbolic Programs", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 7, July 1993, pp. 812-826
- [20] Lee G., "Parallelizing Iterative Loops with Conditional Branching", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 2, February 1995, pp. 185-189
- [21] Liu D., Giloi W. K., "A Loop Optimization Technique Based on Scheduling Table", *Proceedings of the 22nd Annual International Workshop on Microprogramming and Microarchitecture (MICRO-22)*, August 1989, pp. 135-140
- [22] Milutinović V., *Računarski VLSI sistemi*, Nauka, Beograd, 1994.
- [23] Moon S.-M., Ebcioglu K., "A Study on the Number of Memory Ports in Multiple Instruction Issue Machines", *Proceedings of the 26th Annual International Symposium on Microarchitecture (MICRO-26)*, December 1993, pp. 49-58
- [24] Moon S.-M., Ebcioglu K., "An Efficient Resource-Constrained Global Scheduling Technique for Superscalar and VLIW processors", *Proceedings of the 25th Annual International Symposium on Microarchitecture (MICRO-25)*, December 1992, pp. 55-71
- [25] Müller T., "Employing Finite Automata for Resource Scheduling", *Proceedings of the 26th Annual International Symposium on Microarchitecture (MICRO-26)*, December 1993, pp. 12-20
- [26] Nakatani T., Ebcioglu K., " 'Combining' as a Compilation Technique for VLIW Architectures", *Proceedings of the 22nd Annual International Workshop on Microprogramming and Microarchitecture (MICRO-22)*, August 1989, pp. 43-55
- [27] Nikolau A., "Percolation Scheduling: A Parallel Compilation Technique", Technical Report TR-85-678, Cornell University, 1985
- [28] Rajagopalan M., Allan V. H., "Efficient Scheduling of Fine Grain Parallelism in Loops", *Proceedings of the 26th Annual International Symposium on Microarchitecture (MICRO-26)*, December 1993, pp. 2-11
- [29] Rau B. R., Glaeser C. D., "Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing", *Proceedings of the 20th Annual Workshop on Microprogramming and Microarchitecture*, October 1981, pp.183-198

-
- [30] Rau B. R., Yen D. W. L., Yen W., Towle R. A., "The Cydra 5 Departmental Supercomputer", *IEEE Computer*, January 1989, pp. 12-35
- [31] Rau B., Fisher J. A., "Instruction-level Parallel Processing: History, Overview, and Perspective", *Journal on Supercomputing*, Vol. 7, No. 1/2, May 1993, pp. 9-50
- [32] Rau R., "Iterative Modulo Scheduling: An Algorithm For Software Pipelining Loops", *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO-27)*, December 1994, pp. 63-74
- [33] Schwiegelshohn U., Gasperoni F., Ebcioglu K., "On Optimal Parallelization of Arbitrary Loops", *Journal of Parallel and Distributed Computing* 11, 1991, pp. 130-134
- [34] Su B., Ding S., Wang J., Xia J., "GURPR A Method for Global Software Pipelining", *Proceedings of the 20th Annual Workshop on Microprogramming (MICRO-20)*, December 1987, pp. 88-96
- [35] Su B., Wang J., "GURPR*: A New Global Software Pipelining Algorithm", *Proceedings of the 24th Annual Workshop on Microprogramming and Microarchitecture (MICRO-24)*, November 1991, pp.212-216
- [36] Tang Z., Chen G., Zhang C., Zhang Y., Su B., Habib S., "GPMB Software Pipelining Branch-Intensive Loops", *Proceedings of the 26th Annual International Symposium on Microarchitecture (MICRO-26)*, December 1993, pp. 21-29
- [37] Ugurdag H. F., Papachristou C. A., "A VLIW Architecture Based on Shifting Register Files", *Proceedings of the 26th Annual International Symposium on Microarchitecture (MICRO-26)*, December 1993, pp. 263-268
- [38] Uht A. K., "Requirements for Optimal Execution of Loops with Tests", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 5, September 1992, pp. 573-581
- [39] Warter N. J., Bockhaus J. W., Haab G. E., Subramanian K., "Enhanced Modulo Scheduling for Loops with Conditional Branches", *Proceedings of the 25th Annual International Symposium on Microarchitecture (MICRO-25)*, December 1992, pp.170-179

Zahvalnice

Neizmernu zahvalnost dugujem svom mentoru, profesoru Zoranu Jovanoviću ne samo na pomoć i savetima ukazanim tokom izrade ovog rada, već na izuzetnoj podršci koju mi je pružio u celokupnoj dosadašnjoj saradnji.

Zahvaljujem kolegama, prijateljima i roditeljima na podršci, a posebno supruzi Sneani na razumevanju.

U Beogradu, novembra 1995.

Dragan Milićev