

4. Spekulativno izvršavanje

Pod spekulativnim izvršanjem podrazumevamo izvršavanje operacije, koja možda nije uopšte morala da se izvrši. Pritom, postoji neka verovatnoća da će ta operacija sa datim ulaznim podacima zaista biti neophodna. Ovakvo izvršavanje može biti korisno za ubrzavanje izvršavanja programa kod paralelnih mašina. Ako mašina ima dovoljno resursa, spekulativno izvršavanje ne može da uspori izvršavanje programskog kôda. Ukoliko je operacija zaista trebalo da se obavi, ona se obavila ranije. Njeno ranije izvršavanje omogućava drugim, ili takođe spekulativnim operacijama ili pak nespekulativnim operacijama da se ranije izvrše. Ako se pokaže da je operacija urađena nepotrebno, neophodno je poništiti sve efekte spekulativno izvršene operacije. Naravno, sve druge operacije koje su koristile rezultate takve spekulativne operacije bile su takođe spekulativne, pa je potrebno poništiti i njihove efekte. Pod rezultatima spekulativne operacije podrazumevaju se rezultat operacije i vrednosti odgovarajućih bita u procesorskoj statusnoj reči nastali kao posledica izvršavanja spekulativne operacije.

Postoje dve kategorije spekulativnog izvršavanja: **spekulativno izvršavanje po kontroli** i **po podacima**. *Spekulativno izvršavanje po kontroli* podrazumeva da se na osnovu pretpostavke da će niz predikata, koji određuju tok kontrole, imati neke vrednosti, unapred obavimo operacije iz jedne ili više grana. To može da se uradi tokom prevođenja ili tokom izvršavanja. Mora se obaviti poništavanje efekata spekulativnog izvršavanja po kontroli, kada operacija nije trebalo da se izvrši. Neophodno je dakle poništiti i rezultate operacija i eventualno nastale izuzetke zbog obavljanja operacija koje nisu ni trebale da se obave na osnovu toka kontrole.

Jedna od tehnika za poništavanje efekata spekulativnog izvršavanja je da se uz rezultat svake spekulativne operacije pridruži i predikat kojim se definiše pod kojim logičkim uslovima se prihvata rezultat operacije {PS 91}{SM 92}{MA 92}{MJ 98A}. Predikat se formira na tako da definiše neophodan ishoda svih grananja koje je spekulativna operacija prešla prilikom selidbe na gore. Najjednostavniji način predstavljanja je pomoću niza bita čija je dužina jednaka broju uslovnih grananja koje je spekulativna operacija prešla prilikom selidbe na gore, a vrednosti bita odgovaraju potrebnim ishodom skoka. Ako se tokom izvršavanja dogodi da je vrednost predikata takva da na osnovu toka izvršavanja treba poništiti efekte spekulativno urađene operacije, odmah se oslobađaju resursi i svi efekti takvih operacije se poništavaju. Takvo izvršavanje se naziva predikatskim izvršavanjem, kao podvrsta spekulativnog izvršavanja {MA 92}. Ovim postupkom efektivno se eliminišu kontrolne zavisnosti operacija, jer se operacija može izvršiti pre određivanja jednog ili više predikata za grananje.

Druga tehnika je da se spekulativno izvršavaju operacije samo duž jednog pretpostavljenog dinamičkog traga. Tada se pamti samo niz predikata tog pretpostavljenog traga i, u osnovi, je zajednički za sve operacije. U trenucima izračunavanja uslovnih grananja, ispituje se da li je obavljeni skok u skladu sa jedinim pretpostavljenim tragom i ako jeste, rezultati svih operacija koje su pripadale grani u koju se skače se prihvataju kao finalni. Ako se utvrdi da je došlo do greške u predikaciji (predviđanju traga), moraju se poništiti efekti svih operacije koje su se zasnivale na toj pogrešnoj pretpostavci. Ključna komponenta u ovom slučaju je prediktor grananja, a pojedine operacije nemaju sopstvene predikate. Naravno, za sve operacije se nekako

mora pamtiti kom bazičnom bloku na tragu su pripadale, a to će detaljnije biti objašnjeno na primeru superskalarnih procesora.

Spekulativno izvršavanje po podacima podrazumeva da se mogu pretpostaviti ulazni podaci za neku operaciju i da na osnovu toga unapred obavimo tu operaciju. Naknadno otkrivena greška u pretpostavci o ulaznim podacima zahteva poništenje efekata te operacije. Razlog za spekulativno izvršavanje po podacima je pokušaj eliminacije direktne prave zavisnosti po podacima, kada se može predvideti argument. U realnosti, ovakvo spekulativno izvršavanje može se najčešće uraditi kada nije izvesno da li zavisnosti po podacima postoje ili ne. Ukoliko zavisnosti ne postoje, spekulativnim izvršavanjem se postiže ranije izvršavanje operacije, jer se koristi ranije izračunata (stara) vrednost iz lokacije. Ukoliko zavisnost postoji, tada se operacija ponovo obavlja sa novom vrednošću, ali sada sa validnim ulaznim podacima, dakle ne spekulativno. Naravno, efekat prethodno izvršene spekulativno operacije nad pogrešnim argumentom(ima) mora se tada poništiti.

Spekulativno izvršavanje po podacima može ubrzati izvršavanje naročito kada se tokom prevođenja, u nekim slučajevima, ne može sa sigurnošću utvrditi da li postoje ili ne neke zavisnosti po podacima. Bez spekulativnog izvršavanja moralo se pretpostaviti da zavisnosti postoje, kako bi se garantovala korektnost kôda. Zavisnosti po podacima kod programskih petlji i pokazivači su najčešći uzročnici pojava relacija zavisnosti između operacija, kada se u toku prevođenja, ne može dati definitivan odgovor o postojanju prave zavisnosti po podacima. Kako će kasnije biti pokazano,

Kao rezime, nameće se pitanje otkuda tako agresivne tehnike optimizacije kôda kod kojih promašaj u predikciji dovodi do značajnog rasipanja resursa. Pokazalo se da je minijaturizacijom vrlo brzo postignut paralelizam u procesorima (broj mašinskih resursa različitih tipova kako je bilo navedeno u *List Scheduling-u*) koji je veći nego prosečan paralelizam na instrukcijskom nivou. Samim tim postavilo se pitanje: čime uposliti slobodne resurse? Kao prirodno rešenje se nametnulo spekulativno izvršavanje.

Kontrolne zavisnosti zahtevale su da se neka operacija ne može izvršiti dok se na uslovnom grananju ne odredi da će tok izvršavanja sigurno obuhvatiti tu operaciju. Spekulativno izvršavanje po kontroli omogućava ranije izvršavanje instrukcija kojim se efektivno eliminišu kontrolne zavisnosti. Ukoliko je zaista trebalo da se izvrši takva operacija, eliminacijom kontrolne zavisnosti ubrzava se izvršavanje. Spekulativno izvršavanje po podacima omogućava teoretski da se eliminišu zavisnosti po podacima, ako se mogu predvideti argumenti. Vrlo retko je u praksi moguće sa značajnijom verovatnoćom predviđati argumente, osim ako nije u pitanju neka ranije izračunata vrednost za koju se pretpostavlja da je neće promeniti operacije koje mogu, a ne moraju, da je promene. Ako se pretpostavi da će zbog prirode zavisnosti, vrednost promenljive sa malom verovatnoćom biti menjana, korisno je uvesti spekulativno izvršavanje po podacima. Tada efektivno eliminišemo zavisnosti po podacima, ukoliko se stara vrednost ne promeni.

4.1. Eliminacija kontrolnih zavisnosti

Više autora je realizovalo eksperimente {WA 91}{LW 92} kojima je analizirali paralelizam u kôdu koji se može postići potpunom eliminacijom kontrolnih zavisnosti. U tom slučaju ostaju samo direktne prave zavisnosti po podacima. Rezultati simulacija

ukazuju da je prosečan maksimalan paralelizam, zavisno od tipa programskog kôda koji je bio analiziran, u opsegu između 10 i 100. Ovaj nivo paralelizma izuzetno je privlačan, s obzirom da je znatno teže efikasno iskoristiti paralelizam na višem nivou, na nivou procesa ili niti {MJ 99}.

Kontrolnu zavisnost moguće je eliminisati na nekoliko različitih načina. Zbog kompletnog uvida, na ovom mestu će se razmatrati i rešenja koja nisu striktno vezana samo za model instrukcijskog nivoa paralelizma, već i za implementaciju.

Vrlo često se limitira dubina do koje je dozvoljeno spekulativno izvršavanje instrukcija po kontroli. Dubina spekulativnog izvršavanja najčešće se definiše kao broj uslovnih grananja preko kojih je operacija spekulativno prebačena na gore, prilikom eliminacije kontrolnih zavisnosti. Ta dubina može se, ali ne mora, ograničavati u postupku eliminacije kontrolnih zavisnosti tokom prevođenja. Ako se ograniči, smanjuje se broj operacija koje se mogu izvršiti, a da se nakon toga njihov rezultat kasnije poništava zbog izvršavanja dinamičkog traga koji ne obuhvata tu operaciju.

Slična logika primenjuje se i kod metoda eliminacije kontrolnih zavisnosti u toku izvršavanja (run-time metoda). Tada se ograničava broj predviđenih uslovnih grananja za koje nije još razrešeno da li su dobro predviđena tokom izvršavanja. U implementaciji run-time metoda eliminacije kontrolnih zavisnosti, u hardveru na nekom mestu mogu da postoje tag-ovi, u kojima su zapisani pretpostavljeni ishodi grananja pod kojim rezultat operacije postaje validan. Kako se u hardveru mora odrediti maksimalna dužina tag-a, postoje hardverska ograničenja za dužinu predikata za grananje pridruženog spekulativnoj instrukciji, a samim tim i maksimalan broj grananja preko kojih se operacija može prebaciti u cilju spekulativnog izvršavanja. Naravno, pored hardverskih ograničenja važe i stavovi vezani za ograničavanje broja grananja za spekulativnost u vreme prevođenja.

4.1.1. Eliminacija kontrolnih zavisnosti prilikom prevođenja (compile time)

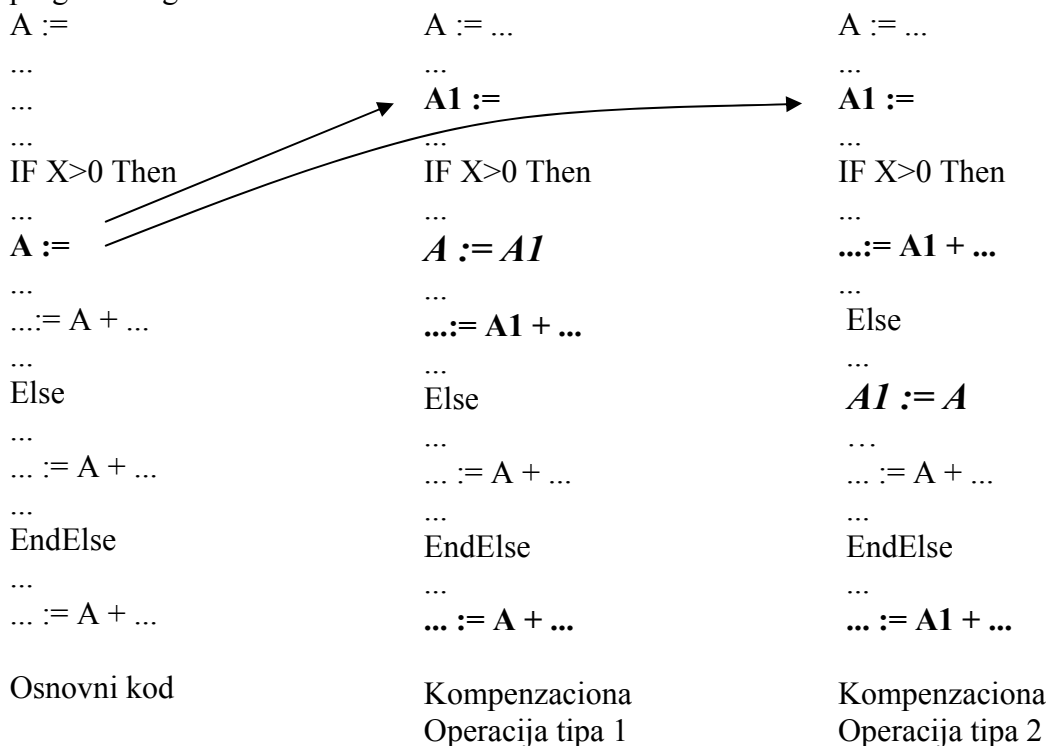
Transformacija kojom je operacija iz jedne grane seljena preko uslovnih grananja na gore, kada nije postojala operacija identična toj operaciji slobodna na vrhu u drugoj grani, mogla se obaviti samo ako je rezultat operacije bio mrtav u drugoj grani. Poštujući strogo definiciju spekulativnog izvršavanja, ova transformacija je specijalan slučaj spekulativnog izvršavanja po kontroli. Takva konstatacija važi, jer se izračunati rezultat koristi samo u jednoj od grana, a rezultat nije potreban u drugoj grani.

Kako je u ovom slučaju spekulativnog izvršavanja po kontroli eliminisan rezultat operacije, kada je operacija nepotrebno izvršena? Pošto je rezultat mrtav u drugoj grani, automatski se poništava rezultat operacije kada kontrola toka prolazi kroz drugu granu. Kako je već napomenuto, treba samo poništiti efekte eventualnog izuzetka (exception), ako bi se dogodio prilikom izvođenja te operacije i ako bi tok kontrole bio takav da nije trebalo ni da se prođe tom granom. Poništavanjem uticaja izuzetaka se nismo detaljnije bavili kada smo razmatrali osnovne selidbe operacija, kada je operacija seljena samo iz jedne grane preko grananja na gore. To smo činili zbog pojednostavljivanja razmatranja. Jasno je da se efekat nepotrebno izuzetka i u tim slučajevima mora rešavati na isti način kao i kod svih ostalih spekulativnih operacija.

U slučaju da je rezultat živ u drugoj grani, spekulativno izvršavanje dovodi do prepisivanja ranijeg rezultata i efekta sličnog antizavisnosti, jer stari rezultat nije pročitao, a već je prepisan novom vrednošću. Zbog analogije sa antizavisnošću, ovaj hazard tipa upis pre čitanja nastao spekulativnim prebacivanjem operacije preko uslovnih grananja na gore naziva se još i **uslovnom antizavisnošću** {PJ 96}. U daljem tekstu pod **spekulativnom selidbom operacije preko uslovnih grananja na gore** će se podrazumevati slučaj kada postoji uslovna antizavisnost {JP 92}.

Na osnovu analogije, nameće se ideja da se uradi preimenovanje promenljive u koju se upisuje pri ovakvom spekulativnom izvršavanju. Takvim preimenovanjem automatski se generiše promenljiva mrtva u drugoj grani, ali reference čitanja te promenljive u grani iz koje je izvučena operacija moraju takođe da se preimenuju. Ukoliko u grani postoji ponovni upis u tu promenljivu, preimenovanje referenci obavlja se samo do tačke novog upisa. Preimenovanje se može posmatrati kao generisanje rezultata operacije koji je mrtav u drugoj grani, kako bi se mogla primeniti osnovna selidba operacija ranije nazvana: *Selidba operacije preko grananja na gore, kada nema identične operacije slobodne na vrhu u drugoj grani.*

Da bi preimenovanje bilo korektno i u slučaju ponovnog objedinjavanja toka kontrole, zbog spojeva, mora se sprovesti ponovno stapanje nepreimenovane i preimenovane vrednosti. Na taj način se u trenutku spajanja ponovo javljaju iste promenljive. Da bi se u primeru jasno naznačilo da su neki rezultati živi u granama, napisane su operacije u kojima se sa desne strane izraza nalazi jedan označeni argument. Naravno da i dalje moraju da se poštuju sve prave zavisnosti po podacima. Motivi za izvlačenje operacije pre grananja zavise od algoritma optimizacije, a na ovom mestu je ideja samo da se prikažu osnovne transformacije spekulativnog izvršavanja po kontroli i pokaže njihova ispravnost. Preimenovanje i stapanje, kojim se mogu objasniti osnovne transformacije programskog kôda ilustrovani su na Sl. 4.1.



Sl. 4.1. Preimenovanja prilikom spekulativnog izvršavanja

Napomene uz Sl. 4.1.:

Strelice označavaju spekulativnu selidbu sa neophodnim preimenovanjem.

Iste vrste označavaju ekvivalentne operacije, izuzev kada je razmak poremećen kompenzacionim operacijama naglašenim *italic slovima*.

Operacija + i ispitivanje uslova >0 samo su primeri operacija i uslova, čime se ne umanjuje generalnost pokazanih transformacija.

Stapanje u srednjoj koloni obavlja se tako što se, u grani iz koje je preseljena operacija na gore, originalnoj promenljivoj dodeli vrednosti preimenovane (nove) promenljive. Neophodna dodatna operacija nazvana je **kompenzacionom operacijom tipa 1**. Ovo dodeljivanje može da se uradi u bilo kom delu grane. Dakle, kompenzaciona operacija tipa 1 slobodna je i na vrhu i na dnu bazičnog bloka iz koga je izvučena operacija, jer se u grani koriste preimenovane vrednosti argumenata. Zato je moguće kompenzacionu operaciju tipa 1 uraditi u bilo kom ciklusu tog bazičnog bloka.

Na dovoljno paralelnom hardveru ovakva operacija ne može da dovede do produžavanja izvršenja programskog kôda. Izuzetak je samo slučaj kada se prilikom optimizacije kôda dogodi da se sve operacije eliminišu iz bloka, osim kompenzacione operacije tipa 1. Tada se kompenzaciona operacija može prebaciti u drugu granu, tako da se u njoj radi preimenovanje. Time se preimenuje ceo kôd iza uslovnih grananja i spojeva, izuzev dela kôda iz grane iz koje nije seljena operacija u kome se koristi stara vrednost promenljive. Ovo se naziva **kompenzaciona operacija tipa 2**. Ukoliko ima upisa u promenljivu u grani iz koje nije seljena operacija, tada se upis može definisati kao upis u preimenovanu promenljivu i kompenzaciona operacija tipa 2 nije neophodna.

Kompenzaciona operacija tipa 2 takođe je slobodna i na vrhu i na dnu bazičnog bloka. Ova tvrdnja proizilazi iz sledećih stavova:

- a. Kompenzaciona operacija potrebna je samo ako u grani iz koje nije spekulativno pomerana operacija nema upisa u promenljivu za koju bi moralo da se radi preimenovanje.
- b. Sve operacije u grani koje čitaju vrednost promenljive čitaju staru vrednost do eventualnog ponovnog upisa.

Izborom kompenzacione operacije tipa 1 ili tipa 2, uvek se može postići da spekulativno izvršavanje sa kompenzacionom operacijom ne dovede do produžavanja trajanja izvršenja programa, ako postoji dovoljan paralelizam u mašini. Zanimljivo je da jednom preseljena operacija preko uslovnog grananja na gore, kod koje je moralo da se uradi preimenovanje, može da se dalje seli na gore preko proizvoljnog broja uslovnih grananja. To je moguće jer je njen rezultat sigurno mrtav u svim drugim granama uslovnih grananja preko kojih je preseljena na gore. Tako se jednim preimenovanjem dozvoljava da se operacija preseli preko proizvoljnog broja uslovnih grananja na gore.

Ako se posmatra ukupan efekat spekulativne selidbe operacije preko uslovnih grananja na gore i nabroje zaključci prethodnog razmatranja, sledi:

- a. **Operacija se uvek može preseliti preko uslovnih grananja na gore**, bez obzira na to da li je njen rezultat živ u drugoj grani
- b. Ako je rezultat živ u drugoj grani, mora se raditi preimenovanje.
- c. Ukoliko na bilo kom tragu može da dođe do ponovnog spajanja tragova nakon uslovnih grananja i spojeva, može biti potrebna kompenzaciona operacija kojom se stapaju preimenovana i originalna promenljiva.

- d. Kompenzaciona operacija nije neophodna ako se pre spajanja tragova javlja upis bar u jednoj od grana. Tada se stapanje originalne i preimenovane promenljive izvodi preko operacije upisa u jednoj od grana. Naravno, upisi ne moraju da budu odmah u granama iza grananja, a ovde je naveden samo najjednostavniji slučaj.
- e. Izborom grane u kojoj se nalazi kompenzaciona operacija, odnosno tipa kompenzacione operacije, može se postići da se **na dovoljno paralelnoj mašini nikada ne produžava trajanje izvršavanja po bilo kom tragu, zbog postojanja kompenzacionih operacija.**

Kako u današnjim mašinama tipično postoji dovoljno paralelizma, spekulativna selidba operacija preko uslovnih grananja na gore predstavlja veoma privlačnu transformaciju, ravnopravnu sa bazičnim transformacijama selidbe operacija {PS 91} {SM 92}. Osnovna podrška za agresivnu primenu ove vrste spekulativnog izvršavanja je broj registara koji se koriste za preimenovanja. Osim toga, neophodno je da postoji dovoljno paralelizma za jednostavne kompenzacione operacije kojima se radi stapanje promenljivih preko prepisivanja sadržaja registara. Kompenzacione operacije najčešće se svode na izbor argumenta iz odgovarajućeg registra, tako da je zamena za kompenzacione operacije u stvari selekciona logika za izbor registra sa odgovarajućim argumentom.

4.1.1.2. Najbrže izvršavanje po svim tragovima

Pretpostavimo kôd bez programskih petlji. Kao ekstremum veoma agresivne primene spekulativnih selidbi preko uslovnih grananja na gore, sve operacije mogu se preseliti do svog najranijeg trenutka izvršavanja određenog samo pravim zavisnostima po podacima. Pritom će se događati selidbe preko spojeva na gore i preko uslovnih grananja na gore. Prilikom svake selidbe operacije preko spoja na gore, javljaju se dve kopije operacije. Istovremeno, i sva uslovna grananja bi se selila na gore do svog najranijeg trenutka izvršavanja. Taj trenutak određen je najranijim trenutkom završetka operacije koja izračunava uslov za skok. Najranijim izvršavanjem grananja smanjuje se ukupan broj spekulativnih izvršavanja ostalih operacija. Na kraju bi se operacija (odnosno kopirane i spekulativne operacije proizašle iz te operacije) našle u bazičnom(im) bloku(ovima) u kome(jima) mogu da se izvršavaju u svom najranijem trenutku za sve tragove. Na taj način bi se, za dovoljno paralelnu mašinu, postiglo optimalno vreme izvršavanja za sve dinamičke tragove.

Eksplzija kôda pri ovakvim transformacijama veoma je velika, ali uzimajući u obzir veličinu memorija današnjih računara i razvoj tehnologije, može se u budućnosti očekivati primena ovako agresivne optimizacije kôda. U specijalnim slučajevima se ovako agresivna optimizacija može raditi (npr. za kôd unutrašnje petlje kod ugnježenih petlji, kada postoje grananja). Ovakvom agresivnom optimizacijom pokazano je da se, zasada više teoretski nego praktično, spekulativnom selidbom preko grananja na gore postiže potpuna eliminacija kontrolnih zavisnosti i samim tim se kontrolne zavisnosti mogu zanemariti kao ograničavajući faktor u paralelizaciji kôda na instrukcijskom nivou.

4.1.1.3. Detaljan primer dobijanja optimalnog kôda po svim tragovima.

Prikaz eliminacije kontrolne zavisnosti kod grananja u programskom kôdu otežan je činjenicom da se vrlo lako generiše veliki broj tragova izvršavanja. Najjednostavniji primer je IF-THEN-ELSE struktura sa po jednim bazičnim blokom pre grananja i posle spoja. Njen kôd u višem programskom jeziku predstavljen je na Sl. 4.2.b.

REGISTRI ŽIVI NA VRHU: X1, X2, X3, X4, X5, X6, X7, **Z6, Z7**, Z10, Z13

MEĐUKOD:

OP1: Y1 := X3 * X4

OP2: Y2 := X5 - X6

OP3: T3 := X1 * X2

OP4: Y4 := T3 + 2

OP5: Y5 := X1 * X7

THEN

OP6: Z6 := X1 - X3 OP11: T11 := Z7 + Y4

OP7: Z7 := Z6 * Z6 OP12: T12 := Z6 - T11

OP8: T8 := Y1 * Z7 OP13: Z13 := Y5 / T12

OP9: T9 := Y2 + T8

OP10: Z10 := 7 * T9

ELSE

OP14: T14 := Z10 * Z13

OP15: Z15 := T3 - T14

a.

VIŠI PROGRAMSKI JEZIK:

Y1 := X3 * X4;

Y2 := X5 - X6;

Y4 := X1 * X2 + 2;

Y5 := X1 * X7;

IF Y2 > 0 THEN DO (P=0,4)

Z6 := X1 - X3;

Z7 := Z6 ** 2;

Z10 := 7 * (Y1 * Z7 + Y2); END

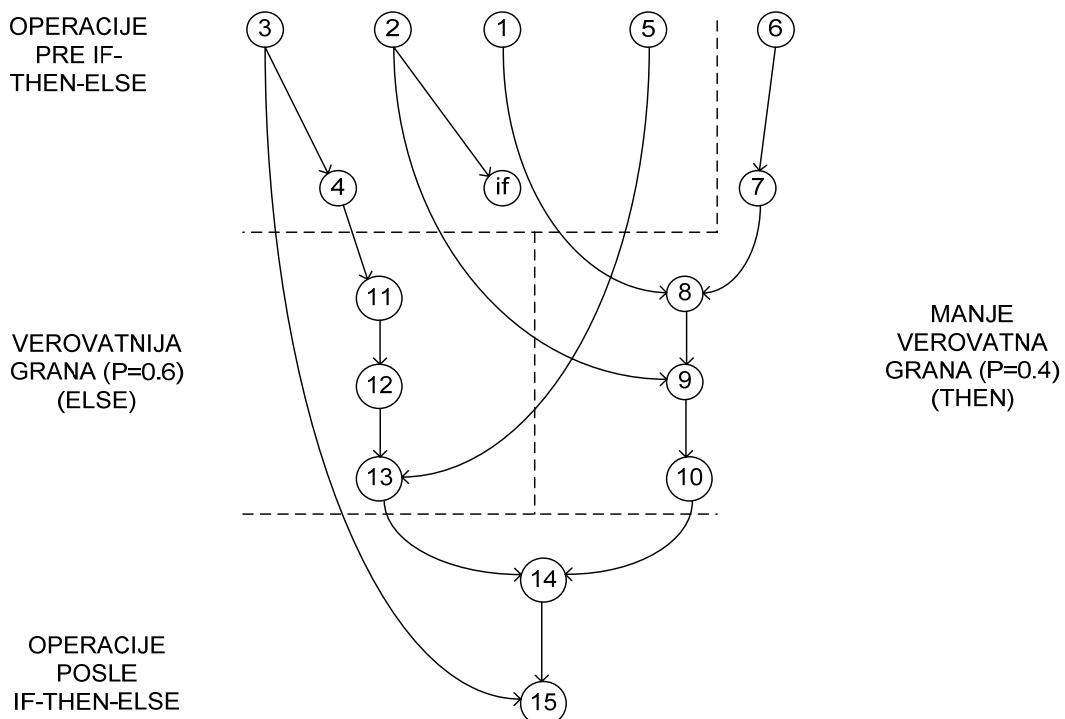
ELSE (P=0,6) Z13 := Y5 / (Z6 - (Z7+Y4));

Z15 := X1 * X2 - Z10 * Z13;

b.

REGISTRI ŽIVI NA DNU: Y5, **Z6, Z7**, Z10, Z13, Z15, ...

Sl. 4.2. Kôd IF-THEN-ELSE strukture i susednih bazičnih blokova.



Sl 4.3. Graf zavisnosti po podacima za oba traga

Primenom elementarnih metoda optimizacije iz klasičnih prevodilaca, dobijen je međukod sa Sl. 4.2.a, pri čemu su registri za čuvanje međurezultata označeni sa T_i , a operacije sa O_{P_i} . Definisani su registri živi na vrhu i živi na dnu kôda, zbog ostatka programa koji nije vidljiv u primeru. Uslov za skok, na osnovu koga se određuje grana za izvršavanje, dobija se na osnovu rezultata drugog reda kôda u obe predstave.

Na osnovu međukoda mogu se formirati grafovi zavisnosti po podacima za svaki od bazičnih blokova. Formiranjem takvih grafova maskirala bi se informacija potrebna za globalnu optimizaciju koda. Druga alternativa je da se formiraju aciklički grafovi zavisnosti po podacima za svaki od tragova (bazični blok pre grananja, kôd jedne od grana, bazični blok iza spoja). Da bi se lakše dočarao metod eliminacije uslovne antizavisnosti, grafovi zavisnosti po podacima oba moguća traga su integrisani, uz označavanje granica bazičnih blokova. Takva reprezentacija sa prikazom granica bazičnih blokova pomoću isprekidanih linija data je na Sl. 4.3. Ako svaka operacija traje po 1 ciklus, najkraće vreme izvršavanja traga sa *ELSE* granom (verovatniji trag) je 7 ciklusa na paralelnoj mašini, zbog postojećih zavisnosti po podacima.

Određivanje kritičnog puta drugog traga znatno je kompleksnije. Prvo je potrebno odrediti broj ciklusa potrebnih da se odredi test uslov na osnovu koje se vrši grananje. To vreme je dva ciklusa, jer je potrebno prvo odrediti rezultat OP_2 , a dodatni ciklus je neophodan da se ispita uslov za grananje, Preostali deo traga u *THEN* grani ima dužinu 7 ciklusa, tako da je ukupna dužina traga 9 ciklusa. Metodama selidbe operacija između bazičnih blokova koje ne koriste spekulativno izvršavanje, nije moguće preseliti OP_6 pre uslovnog grananja, jer je promenljiva Z_6 živa u drugoj grani.

Primer ponašanja klasičnih metoda optimizacije prikazan je za raspoređivanje po tragu (*Trace Scheduling*) na levoj koloni Sl. 4.4.a., po *Trace Scheduling* algoritmu bez spekulativnog izvršavanja. U primeru, ponovo je korišćeno uprošćenje da sve operacije traju jedan ciklus.

Istovremena optimizacija oba traga može se sprovesti zahvaljujući metodi eliminacije kontrolnih zavisnosti preko preimenovanja uslovnih antizavisnosti. U prvom koraku utvrđuje se dužina segmenta kritičnog puta koji je ispred grananja za svaki od tragova. Dužina segmenta kritičnog puta traga ispred grananja za *ELSE* granu je 2, a za *THEN* granu 0, jer je pretpostavljeno da sve operacije traju 1 ciklus. Dužina kritičnog puta za određivanje uslova za grananje je 2 ciklusa, pa se delovi koda koji moraju biti u granama mogu najranije javiti tek u trećem ciklusu. Segment kritičnog puta traga *ELSE* grane i kritičan put za izračunavanje test uslova imaju istu dužinu - ujedno i najveću dužinu među segmentima pre grananja. Deo kritičnog puta iz *THEN* grane dužine 2 ciklusa (operacije OP_6 i OP_7) mora se izvući ispred grananja, ako se želi najviši nivo paralelizma. To je moguće sprovesti samo eliminacijom uslovne antizavisnosti preimenovanjem i uvođenjem pomoćnih promenljivih Z_{6K} i Z_{7K} . U *THEN* grani se tada, ako se koristi kompenzaciona operacija 1, mora dodati kompenzacioni kod kojim se dodeljuju vrednosti preimenovanih promenljivih originalnim promenljivama, ako se prolazi kroz tu granu. Kompenzacioni kod se može izvršiti u bilo kom ciklusu u grani.

RASPOREĐIVA NJE PO TRAGU (PR. 1)	RASPOREĐIVANJE SPEKULATIVNA SELIDBA
1, 2, 3	
4, 5	
6 11	2, 3, 6
7 12	1, 4, 7
8 13	5, 8, K6 11, 5
9	9, K7 12
10	10 13
14	14
15	15
THEN 9 ELSE 7	THEN 7 ELSE 7
BROJ CIKLUSA POD PRETPOSTAVKOM DA SVE OPERACIJE TRAJU PO 1 CIKLUS	

VIŠI PROGRAMSKI JEZIK:

Y1 := X3 * X4;

Y2 := X5 - X6;

Y4 := XI * X2 + 2;

Y5 := XI * X7;

Z6K := X1-X3;

Z7K := Z6K ** 2;

IF Y2>0 THEN DO (P=0,4)

 Z10 := 7 * (Y1 * Z7K + Y2);

 Z6 := Z6K;

 Z7 := Z7K; END

ELSE (P=0,6)

 Z13 := Y5 / (Z6 - (Z7 + Y4));

Z15:= XI * X2 - Z10 * Z13;

b. KOD POSLE SELIDBI I SA
DODATIM KOMPENZACIONIM
OPERACIJAMA

a. RASPOREDI OPERACIJA ZA
LIW (Large Instruction Word)
MAŠINU SA 3 PROCESORSKE
JEDINICE

K6 i K7 su kompenzacione operacije tipa 1

Sl. 4.4. Rasporedi za raspoređivanje po tragu, eliminaciju uslovne antizavisnosti i kôd u višem jeziku posle eliminacije uslovne antizavisnost.

Konačni raspored operacija za LIW mašinu sa 3 univerzalne aritmetičko-logičke jedinice dat je u desnoj koloni Sl. 4.4.a. Postignuto je da se kôd po oba traga izvršava za samo 7 ciklusa, primenom spekulativnog izvršavanja. Nijedna od osnovnih selidbi operacija nije mogla da ostvari ovaj rezultat za trag kroz THEN granu, jer je bilo neophodno eliminisati kontrolne zavisnosti, a to se moglo postići samo spekulativnim operacijama, uz dodatak kompenzacionih operacija. Konačni ekvivalentni kôd u višem programskom jeziku zajedno sa kompenzacionim kôdom dat je na Sl. 4.4.b.

4.1.1.4. Detaljan primer kada je potrebna kompenzaciona operacija tipa 2.

Neka je dat kôd u višem programskom jeziku koji predstavlja jednu IF-THEN-ELSE strukturu zajedno sa bazičnim blokovima pre grananja i posle spoja:

```

Registri živi na vrhu: X1, X2, X3, X4, X5, X6, X7, X8, Z6, Z7, Z9, Z10, Z11,...
Y1 := X5 + X6;
Y2 := X2 - X3;
Y3 := X1 * X4;
Y5 := (Y3 + X7) / 2;
IF Y2 > 0 THEN (P = 0,6)
    Z6 := Y5 ** 2;
    Z7 := Z6 - X5;
    Z9 := (Z7 + Y2) * X3;
END
ELSE (P = 0,4)
    Z10 := Y1 / X8;
    Z11 := 3 * Z10;
END
Y13 := (Z9 - Z11) * (Y3 + X7);

```

Registri živi na dnu: Y1, Y2, Y3, Y5, Y13, Z6, Z7, Z9, Z10, Z11,...

Sl. 4.5. Kôd u višem programskom jeziku za kompenzacionu operaciju tipa 2

U zagradama su naznačene verovatnoće izvršavanja THEN i ELSE grana određene na osnovu podataka koje je dao programer ili rada profilera. Klasični programski prevodilac dao bi sledeći međukod (Ti su pomoćni registri):

```

OP1: Y1 := X5 + X6
OP2: Y2 := X2 - X3
OP3: Y3 := X1 * X4
OP4: T4 := Y3 + X7
OP5: Y5 := T4 / 2

```

THEN grana	ELSE grana
OP6: Z6 := Y5 * Y5	OP10: Z10 := Y1 / X8
OP7: Z7 := Z6 - X5	OP11: Z11 := 3 * Z10
OP8: T8 := Z7 + Y2	
OP9: Z9 := T8 * X3	

```

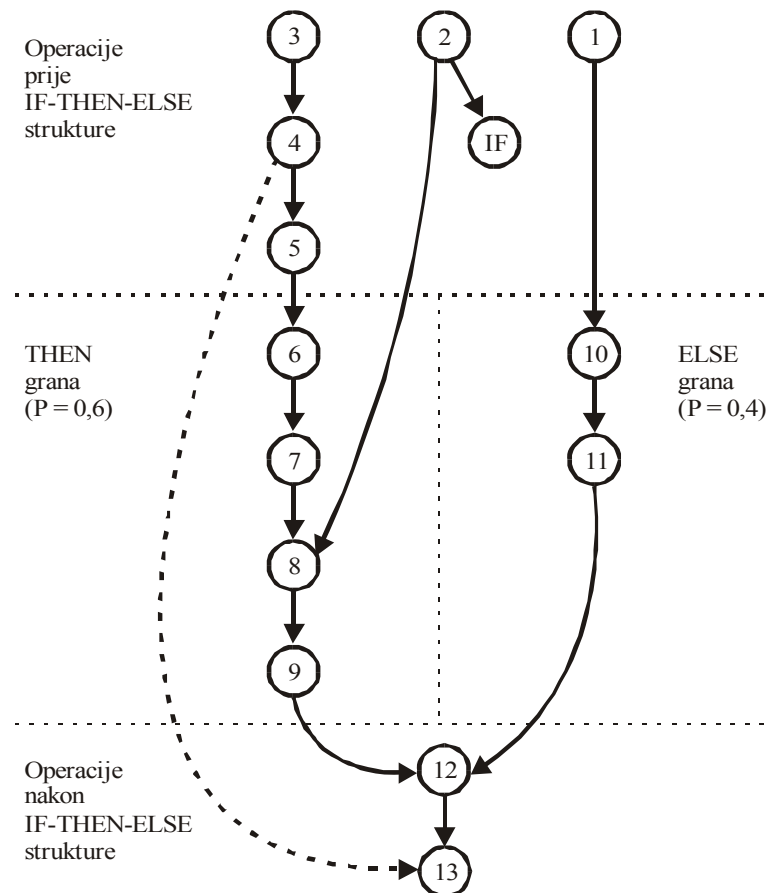
OP12: T12 := Z9 - Z11
OP13: Y13 := T12 * T4

```

Sl. 4.6. Međukod za kompenzacionu operaciju tipa 2

Uslov za grananje izračunava se u drugoj operaciji koda tako da izvršavanje operacija u THEN ili ELSE grani zavisi od ishoda ove operacije. Ta situacija može se modelirati kontrolnom zavisnošću. Graf zavisnosti po podacima može se formirati za oba

dinamička traga koji postoje u ovom segmentu koda. Na slici 1. prikazan je objedinjeni graf zavisnosti po podacima koji uključuje oba traga.



Slika 4.7. Objedinjeni graf zavisnosti po podacima za oba dinamička traga u međukodu. Isprekidanom linijom je prikazana tranzitivna zavisnost

Ukoliko pretpostavimo da trajanje svake od operacija iznosi jedan ciklus, onda je najkraće vreme izvršavanja dinamičkog puta koji uključuje verovatniju THEN granu 9 ciklusa. Na dinamičkom tragu koji sadrži ELSE granu, za određivanje i ispitivanje uslova grananja, takođe su potrebna 2 ciklusa. Pošto operacija OP10 ne zavisi po podacima od operacije OP5, trajanje ostatka ovog dinamičkog traga iznosi 4 ciklusa, odnosno, minimalno trajanje celog dinamičkog traga, ukoliko se ne koristi spekulativno izvršavanje, iznosi 6 ciklusa. Ovo optimalno trajanje može se postići selidbom operacije OP5 preko grananja u obe grane na dole.

Dodatna optimizacija ovog segmenta koda može se postići korišćenjem spekulativnog izvršavanja. Naime, operacije OP10 i OP11 iz ELSE grane ne zavise po podacima od operacija OP4 i OP5 te njihova selidba na gore preko grananja ima potencijal da ubrza izvršavanje. Ovakva selidba rezultuje kodom u višem programskom jeziku u kojem je ELSE grana sada ostala prazna, zbog čega se nameće potreba za korišćenjem kompenzacione operacija tipa 2 {RI 05}.

```

Y1 := X5 + X6;
Y2 := X2 - X3;
Y3 := X1 * X4;
Y5 := (Y3 + X7) / 2;
Z10K := Y1 / X8;
Z11K := 3 * Z10K;
IF Y2 > 0 THEN
    Z6 := Y5 ** 2;
    Z7 := Z6 - X5;
    Z9 := (Z7 + Y2) * X3;
    Z10K := Z10;
    Z11K := Z11;
END
Y13 := (Z9 - Z11K) * (Y3 + X7);

```

Sl. 4.8. Kôd u višem programskom jeziku sa kompenzacionom operacijom tipa 2

Da bi se sačuvala semantička ispravnost kôda i u ostatku programa ispod ovog segmenta, neophodno je izvršiti preimenovanja promjenljivih Z10 i Z11 u Z10K i Z11K, respektivno.

Međukod ima sledeći oblik:

```

OP1: Y1 := X5 + X6
OP2: Y2 := X2 - X3
OP3: Y3 := X1 * X4
OP4: T4 := Y3 + X7
OP5: Y5 := T4 / 2
OP10: Z10K := Y1 / X8
OP11: Z11K := 3 * Z10K

```

THEN grana
<pre> OP6: Z6 := Y5 * Y5 OP7: Z7 := Z6 - X5 OP8: T8 := Z7 + Y2 OP9: Z9 := T8 * X3 OP10K: Z10K := Z10 OP11K: Z11K := Z11 </pre>

```

OP12: T12 := Z9 - Z11K
OP13: Y13 := T12 * T4

```

Sl. 4.9. Međukod sa kompenzacionom operacijom tipa 2

Primenom algoritma raspoređivanja po tragu i imajući u vidu da su kompenzacione operacije OP10K i OP11K slobodne i na vrhu i na dnu bazičnog bloka dobija se sledeći raspored:

1,2,3	
4, 10	
5, 11	
6, K10	
7, K11	
8	
9	
12	
13	

Sl. 4.10. Finalni raspored sa uklonjenom ELSE granom

Trajanje dinamičkog traga koji sadrži THEN granu ostalo je nepromenjeno, 9 ciklusa, dok je trajanje dinamičkog traga koji je u polaznom kodu sadržao ELSE granu, spekulativnim izvršavanjem operacija OP10 i OP11, smanjeno na 5 ciklusa. ELSE grana je naravno nestala, a to je i bio razlog za uvođenje kompenzacione operacije tipa 2.

4.1.2. Eliminacija kontrolnih zavisnosti u vreme izvođenja

4.1.2.1. Jedinствен tok kontrole (prediktori grananja)

Osnovni mehanizam sprovođenja spekulativnog izvršavanja po kontroli u cilju eliminacije kontrolnih zavisnosti u toku izvođenja je dinamička predikcija grananja. Neophodno je da mašina ima deo koji na osnovu istorije toka programa i eventualno nekih podataka iz vremena prevođenja obavi predikciju grananja. Tada se operacije predviđenog bazičnog bloka neposredno iza uslovnog grananja obavljaju spekulativno sve do trenutka u kome se odredi da li je predikcija grananja bila tačna ili ne. Ukoliko je predikcija grananja bila tačna, operacije gube karakter spekulativnih i operacije koje su zavisile po podacima od tih operacija, bar po toj osnovi, ne moraju više da budu spekulativne. Naravno da se na osnovu ranijeg predviđenog skoka može raditi i predikcija narednog skoka, čime se efektivno u toku izvršavanja radi spekulativna selidba operacija na gore preko većeg broja grananja. Ove selidbe će biti detaljnije razmatrane u poglavlju o superskalarnim procesorima.

Osnovna ideja uvođenja dinamičkih prediktora je da se formira pretpostavljeni dinamički trag koji se ograničava po veličini i zato se često zove instrukcijski prozor. U tom dinamičkom tragu se može nalaziti nekoliko predviđenih skokova, tako da većina operacija kandidata za izvršavanje u tom instrukcijskom prozoru predstavljaju spekulativne operacije. Ubrzanje dobijeno ovakvim načinom eliminacije kontrolnih zavisnosti jako zavisi od kvaliteta prediktora grananja, jer pogrešna predikcija može da dovede do poništavanja rezultata velikog broja operacija i do potrebe da se briše sadržaj skoro kompletnog instrukcijskog prozora. Osnovni kvant spekulativnosti u ovom slučaju je bazični blok, jer se spekulativno u izvršavanje prebacuje najmanje ceo bazični blok, iako nije poznat ishod grananja. Zbog ogromnog značaja dobre predikcije grananja, samim prediktorima je posvećen deo 4.1.2.5. Današnji superskalarni procesori, koji dominiraju među najnovijim procesorima, upravo eliminišu kontrolne zavisnosti u vreme izvođenja primenom veoma kompleksnih prediktora grananja.

Osnovna karakteristika ovog pristupa eliminacije kontrolnih zavisnosti je da se primenjuju vrlo agresivne metode optimizacije sa velikim instrukcijskim prozorima, a

samim tim i tragovima. Ovde je namerno izbegnuta diskusija o tome kako se unutar tog prozora znaju zavisnosti po podacima i kako se dinamički trag unutar prozora paralelizuje. To se razmatra u poglavlju o superskalarnim procesorima.

4.1.2.2. Više tokova kontrole (paralelni tragovi)

Ova kategorija podrazumeva da postoji istovremeno više jedinica za dohvaćanje instrukcija, da se dohvaćanje obavlja za različite procese i samim tim su različiti dinamički tragovi. Ako se uvede apstrakcija instrukcijskog prozora za deo dinamičkog traga, tada mora da postoji više instrukcijskih prozora, za sve predviđene različite dinamičke tragove procesa. Time se postiže da se delovi ukupnog kôda, koji su kontrolno nezavisni, istovremeno dohvataju ukoliko postoji paralelizam na nivou procesa. Na ovaj način smanjuje se potrebna dubina spekulativnosti za svaki pojedini trag, jer se koristi paralelizam procesa. Pogreške u predikciji unutar nezavisnog zadatka na ovaj način ne utiču na ostale zadatke, pa se postiže efekat smanjenja prosečne štete nastale zbog pogrešne predikcije, zahvaljujući manjoj potrebnoj spekulativnosti. Naravno da bi se greške u predikciji propagirale u slučaju komunikacije zadataka. Zato se komunikacija zadataka obavlja tek kada je sigurno da komunikacija treba da se obavi i nijedan podatak u komunikaciji nije samo predviđen, već je i potvrđen.

Ovakve mašine moraju da imaju više prediktora grananja koji za nezavisne zadatke obavljaju simultanu predikciju grananja. Osim toga, mašina mora da podržava istovremene oporavke od istovremenih pogrešaka u predikciji grananja različitih procesa. U ovu kategoriju mašina spadaju multiskalarni i spekulativni višenitni (multithreaded) procesori. Takve osobine imaju neki najnoviji procesori, jer imaju podršku za istovremenu obradu više niti.

4.1.2.3. Jedan tok kontrole sa hijerarhijskom sekvencom tragova

Generalizacija spekulativnog izvršavanja operacija je da se u paketu spekulativno izvršavaju tragovi. U ovom slučaju neophodno je da se kompletni tragovi izvršavaju na osnovu spekulativnih podataka i spekulativne kontrole. Hijerarhija sekvenci tragova se odigrava na sledeći način: svaki trag dat je početnom adresom traga i sekvencom uslova kod grananja. Kraj traga određen je brojem instrukcija. Predikcija se radi na nivou N tragova, pri čemu su tragovi nanizani na osnovu predviđenih grananja u svim tragovima. Tragovi počinju da se rade na osnovu spekulativnosti i po podacima i po kontroli. Kada se izračunaju stvarne vrednosti kod spekulativnosti po podacima, radi se ponovno izračunavanje za operacije kod kojih je pogrešno predviđen podatak, ali ne i za operacije kod kojih je predikcija bila korektna. Na sličan način postupa se kada dođe do pogrešne predikcije po kontroli. Zanimljiv je oporavak u ovom slučaju. Ako je pogrešna predikcija bila vezana za IF-THEN-ELSE strukturu, tada se oporavak radi samo za taj trag i nalazi se novo mesto spajanja tragova. Naravno da se i ovom prilikom čuvaju vrednosti svih operacija koje su bile i na ispravnom tragu, a ponovo se izračunavaju operacije do tačke spajanja sa spekulativno već izračunatim tragovima {RO 99}.

Ova izuzetno agresivna metoda spekulativnog izvršavanja još je u fazi ispitivanja na simuliranim mašinama. Još nije jasno da li će procesori sa ovako agresivnim metodama spekulativnog izvršavanja na nivou tragova biti široko primenjeni.

4.1.2.4. Dinamička predikcija grananja

Predikcija grananja može se obaviti statički i dinamički. Statička predikcija već je bila donekle objašnjena u poglavlju o Trace Scheduling-u, kada je opisan postupak određivanja najverovatnijeg traga. Osnovna razlika između statičke i dinamičke predikcije grananja javlja se zbog načina na koji se definiše pretpostavljeni ishod grananja. Kod statičke predikcije, verovatniji ishod određen je u vreme prevođenja i ne može se menjati u toku izvođenja. Svaki dolazak do tog grananja u toku izvršavanja uvek dovodi do iste predikcije. Kod dinamičke predikcije, u toku izvršavanja se za isto grananje može menjati predviđeni ishod skoka. Samim tim, dinamička predikcija ima velikih potencijalnih prednosti u odnosu na statičku. Statički dobijena informacija može da posluži za povećavanje verovatnoće pogotka dinamičkog prediktora. Takvi prediktori često se zovu hibridni prediktori grananja.

U predikciji grananja, neophodno je uraditi predikciju dve stvari, ishoda skoka i adrese skoka. Metode za predikciju adrese skoka nisu od interesa u ovoj knjizi, jer nisu striktno vezane za paralelizaciju kôda na instrukcijskom nivou. Pažnja je usredsređena na predikciju ishoda skoka, a kod nekih metoda će adresa skoka biti parametar koji se određuje paralelno sa procenom ishoda skoka.

Istorijski je zanimljivo da su postojala hardverska rešenja koja su predviđala ishod skoka na osnovu statičke predikcije. Ova jednostavna rešenja pokazala su relativno loše performanse, pa je početkom osamdesetih godina 20. veka započeto intenzivno istraživanje u oblasti dinamičkih prediktora.

Navedimo prvo ulazne podatke koji su u poznatim metodama korišćeni za predikciju grananja. Pre svega to mogu da budu podaci vezani za profiler ili dinamički prediktor koji su:

1. Statički određena verovatnoća grananja sa eventualnom procenom verodostojnosti te verovatnoće.
2. Verovatnoće pojavljivanja kombinacija prethodnih true i false vrednosti tog uslova grananja u nizovima određene dužine za sopstvenu istoriju grananja
3. Verovatnoće pojavljivanja kombinacija prethodnih true i false vrednosti okolnih uslova grananja u nizovima određene dužine za istoriju okolnih grananja
4. Zavisnosti prethodnih vrednosti od određujućih adresa
5. Verovatnoće izvršavanja bazičnih blokova
6. Smer skoka (povratni skok ili skok unapred)
7. Vrste instrukcije uslovnog skoka (vrsta flag-a u procesorskoj statusnoj reči, ...)

U ovom nabrajanju su navedeni samo osnovni ulazni podaci potrebni za predikciju grananja. U vreme profiling-a, može se ispitati ponašanje svakog grananja i naći optimalan prediktor za dinamičku predikciju grananja, za zadate ulazne skupove podataka. U tako opštem slučaju, postojao bi širi skup prediktora grananja, od kojih bi se birao odgovarajući tip za svako od grananja. Kada je to jednom određeno tokom profiling-a, ulazna informacija za hibridni detektor treba da bude klasa prediktora koja najviše odgovara nekom tipu grananja. Tako se mogu dobiti veoma kvalitetni dinamički prediktori, kod kojih je verovatnoće pogotka veoma visoka.

Svi ulazni parametri mogu da se koriste za statičku, a većina može da se koristi i za dinamičku predikciju, ali je razlika u tome što hardver mašine sa dinamičkim

prediktorom mora da podrži dinamičko određivanje svakog od navedenih ulaznih podataka i mehanizam preslikavanja tih vrednosti preko tabele ili automata u predviđeni ishod grananja.

Kod dinamičke predikcije, prethodna istorija grananja mora da bude zapamćena na neki način u mašini i ažurirana nakon svakog grananja koje se odigra. Za pamćenje te istorije koriste se tabele i automati, a ulazni podaci su sledeći:

1. Kombinacija nizova true i false vrednosti u pomeračkim registrima određene dužine za sopstvenu (lokalnu) istoriju grananja – "sopstveni trag"
2. Kombinacija nizova true i false vrednosti u pomeračkim registrima određene dužine za globalnu istoriju svih prethodnih grananja na tragu – "korelacioni trag"
3. Adresa odredišta skoka (uključuje smer skoka)

Za različite kategorije grananja mogu se predvideti različiti prediktori po tipu i time ostvariti upareni prediktori. Pokazalo se da najuspešnije metode dinamičke predikcije grananja daju izuzetno visoke verovatnoće pogodaka. Neke od njih daju verovatnoće pogotka od preko 95%. To je i razlog što se u najnovijim procesorima može primeniti veoma agresivno spekulativno izvršavanje po kontroli.

4.1.2.5. Primeri dinamičkih prediktora

U daljem tekstu predstavljena su dva dinamička prediktora, od kojih je jedan opšteg tipa i predstavlja generalizaciju prediktora koji su predložili {YP 91}, a drugi je specifičan prediktor za grananja na kraju ugnježdenih programskih petlji.

Tronivoski prediktor grananja

Tronivoski prediktor grananja koristi tri nivoa ulaza za predikciju. Oni se nazivaju nivo korelacije, nivo istorije sopstvenog traga i automat koji pamti stanja istorije sopstvenog traga i radi predikciju. Informacija se na sva tri nivoa prikuplja dinamički i za to postoje odgovarajući registri i automati.

Nivo korelacije prikuplja se tako što se u pomerački registar zapamti korelacioni trag koji predstavlja snimak Boolean vrednosti uslova za grananje za poslednjih K grananja na tragu. Na ovaj način pamte se ishodi svih K prethodnih grananja na dinamičkom tragu. Ova ulazna informacija neophodna je zato što se pokazalo da ishod nekog grananja značajno zavisi od ishoda grananja koja su mu prethodila na dinamičkom tragu. Zbog postizanja agresivne spekulativnosti i pretpostavke da se postiže visoka verovatnoća pogotka, pomerački registar korelacionog traga popunjava se i predviđenim ishodima grananja, tako da predikcija može da napreduje nezavisno od stvarnih ishoda. Dubina ovog propagiranja najčešće je ograničena nekim malim prirodnim brojem (tipično je 4 – 8 spekulativno predviđenih grananja).

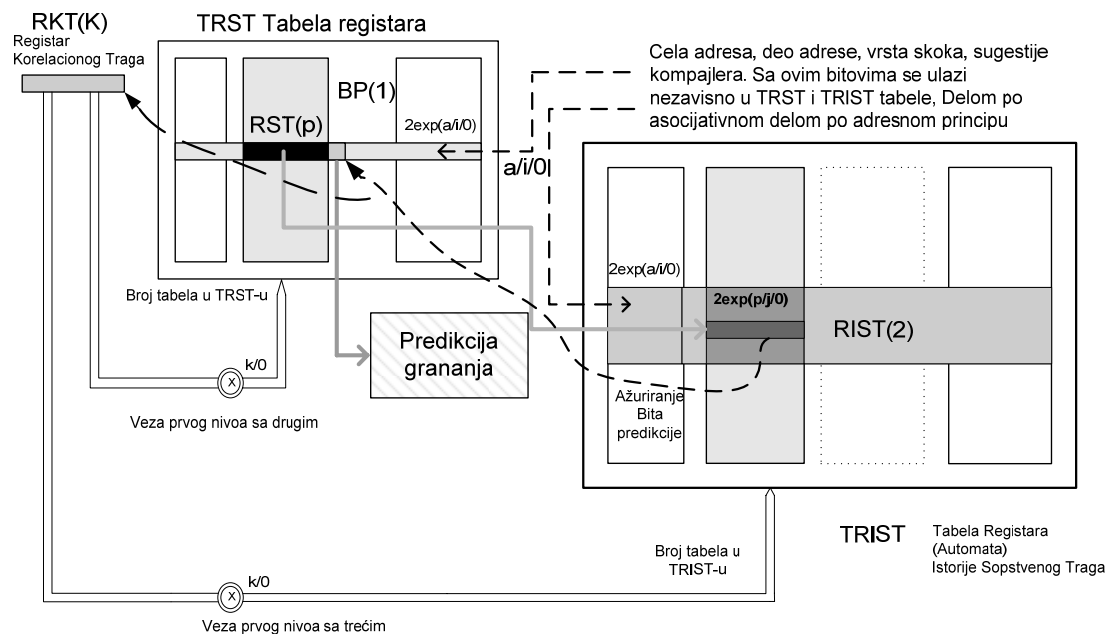
Drugi nivo nazvan je istorijom sopstvenog traga. Na tom nivou pamti se šta se događalo poslednjih p puta kao ishod izvršavanja te operacije grananja. Kako svako grananje ima svoju istoriju, neophodno je da postoji Tabela Registara Sopstvenog Traga (TRST). Osnovna varijanta podrazumeva da se istorija skuplja posebno za svako grananje, a zbog isticanja pojedinačnog grananja, to se naziva još i Per-Address istorija (svako grananje ima svoju adresu).

Jedna od kompleksnijih varijanti zasnovana je na ideji da se istorija vodi zajednički za skup grananja definisan po nekom kriterijumu. Ovo rešenje naziva se Per-Set i omogućava da se objedinjavanjem koriste zajednički elementi nekih tipova grananja u generalnom prediktoru. Osim toga, moguće je da se kombinuje informacija za grananja koja su imala zajednički korelacioni trag, kada se pamti sopstveni trag. Ovaj izlaz se dobije kao P bita iz tabele registara sopstvenog traga.

Izlazi prva dva nivoa služe za određivanje adrese kojom će se naći konačni automat koji preko svog stanja pamti istoriju sopstvenog traga. Dakle, to je automat koji preko svog stanja pamti deo istorije sopstvenog traga definisan sa prva dva nivoa i vrstom i stanjem automata. Prelazi stanja automata tog grananja obavljaju se dakle samo za određene vrednosti korelacionog traga. Ovaj nivo vrši predikciju grananja i radi odmah i ažuriranje bita za korelacioni i sopstveni trag, pored prelaza stanja automata. Treći nivo je u osnovi tabela automata koji se selektuju na bazi informacija sa prva dva nivoa.

Generalni tronivoski model podrazumeva da svaki naredni nivo koristi izlaze prethodnih nivoa za izbor odgovarajućih polja ili automata. Ako se ukinu veze između nivoa kojima se smanjuje dubina veza, dobijaju se dvonivoske šeme koje su svojom implementacijom u Pentium-Pro procesoru dovele do revolucije u kompleksnosti prediktora grananja. Dvonivoske šeme su manje kompleksne i prilagođene su današnjem nivou integracije komponenti u procesoru.

Moguće je još i da postoje različiti tipovi automata. Izbor kod preslikavanja skupa instrukcija uslovnog grananja na tip automata obavlja se na osnovu trenutne adrese instrukcije u operativnoj memoriji, operacionog koda naredbe i klase grananja koju određuje kompajler (a može da bude i rezultat profiling-a). Generalni prediktor prikazan je na Slici 4.11.



Slika 4.11. Opšta šema tronivoskog dinamičkog prediktora

Prvi nivo prediktora obeležen je sa RKT (Registar Korelacionog Traga), jer se odnosi na pomerački registar koji pamti ishode okolnih (prethodnih) grananja. Njegova dužina je k bita, a pokazalo se analizama na realnom kôdu da su vrednosti u opsegu od 4 do 14 bita dovoljne za dobru predikciju. Ovakav rezultat statistike mogao se i očekivati, jer samo relativno bliska ostala grananja na dinamičkom tragu mogu značajnije da utiču na neki skok.

Za sopstveni trag, koriste se pomerački registri obeleženi sa RST (Registar Sopstvenog Traga) i dužine P bita. Broj ovih registara je veliki i u najopštijem slučaju, za svaki skok, je 2^{k+a} , ukoliko postoji veza od prvog nivoa ka drugom i ukoliko se za posebne adrese pamti u nezavisnom registru sopstveni trag. Tabela koja čuva vrednosti tih registara naziva se Tabela Registara Sopstvenog Traga (TRST). Zbog veličine hardvera, retko se pravi veza prvog i drugog nivoa i retko se za različite adrese skoka koriste odvojeni registri sopstvenog traga. Svaki RST ima pridružen jedan ili više bita predikcije (BP) u koje se ubacuje(u) predviđena(e) vrednost(i) bita skoka. Prilikom potvrđivanja predikcije, BP se pomera u RST tog skoka. Ukoliko dođe do pogrešne predikcije, ubacuje se prava vrednost i poništava BP. Promene u prediktoru su male, ali usporavanje izvršavanja u procesoru je ogromno, jer dolazi do poništavanja velikog broja spekulativnih operacija.

Osnovni element trećeg nivoa je registar istorije sopstvenog traga – RIST veličine N bita koji pamti stanje automata. U njemu se preko stanja automata pamti sopstvena istorija i to na način na koji to automat definiše. Ukupan broj ovih automata varira od 1 do 2^{k+a+p} .

Predikcija se obavlja svaki put kada se nakon dekodovanja naiđe na instrukciju uslovnog skoka. Tada je već stari sadržaj RKT mogao da odabere ulaz u TRST tabeli, ako postoji veza prvog i drugog nivoa. Vrednost RST (zajedno sa RKT) i eventualnim ostalim delovima ulaza u TRST tabeli adresira Registar Istorije Sopstvenog Traga RIST (koji u osnovi sadrži stanje automata). Automat zatim radi predikciju grananja i rezultatom predikcije popunjava BP polja (bit predikcije) i same registre RKT i RST. Da bi prediktor obavljao posao za paralelizam današnjih procesora, neophodno je da u jednom ciklusu radi predikciju, ili da bar protočni niz prediktora svakog ciklusa izbacuje novu prediktovanu vrednost bita skoka. Za pamćenje stanja automata u raznim implementacijama imaju različite veličine memorije u bitima, ali često vrednosti N koje su samo 2 (4 stanja) daju dobre rezultate.

Jedan od problema u implementaciji je da broj grananja prevazilazi broj lokacija memorije predviđenih za tabele prediktora. Tada se primenjuje logika cache memorije ili preslikavanja više grananja na iste lokacije. Time se smanjuje tačnost prediktora, ali je to i neminovnost, jer prediktori grananja već imaju zavidan udeo u broju tranzistora današnjih najnovijih superskalarnih procesora. To i jeste razlog za uvođenje varijacija prediktora u kojima više različitih uslovnih grananja deli isti hardver u delovima vezanim za sopstvenu istoriju.

Spekulativna predikcija grananja može se raditi veoma daleko unapred, a ekstremum je doživela u trace procesorima {RO 99} {RS 99}. Danas je možda u prediktorima grananja najveći potencijal za dalju paralelizaciju kôda na instrukcijskom nivou.

Prediktor grananja ugnježenih petlji

Ideja ovog prediktora je da se izdvoje grananja za petlje od ostalih grananja i da se za njih realizuju odvojeni automati u okviru prediktora. Osnovni razlog za ovakvo odvajanje je činjenica da se za uslov iskakanja iz petlji, registar sopstvenog traga može pamtitu u specifičnoj komprimovanijoj formi. Tako se može sa istim brojem bita pamtitu mnogo duža istorija sopstvenog traga. Sama instrukcija uslovnog skoka petlje može se klasifikovati u vreme prevođenja, ali i u toku izvođenja, bilo na osnovu smera skoka ili kôda instrukcije.

Za nadgnježdene petlje takođe se može uspostaviti ista logika pamćenja duže istorije. Nadgnježdenost se može lakše odrediti analizama u vreme prevođenja, metodama za otkrivanje bazičnih blokova. Registar sopstvenog traga takvih uslovnih grananja sadrži niz brojača petlji koji broji broj iteracija petlje do izlaska iz nje.

Uvedena je pretpostavka o linearnoj zavisnosti broja iteracija unutrašnje petlje od vrednosti brojača nadgnježenih petlji (čest slučaj u kôdu). Dakle, ukupan broj iteracija tekuće petlje je linearna kombinacija trenutnih vrednosti m brojača nadgnježenih petlji. To se može predstaviti izrazom:

$$i_0 = a_0 + a_1 * i_1 + a_2 * i_2 + a_3 * i_3 + \dots + a_m * i_m$$

gde je i_0 ukupan broj iteracija tekuće ugnježdene petlje, i_1 trenutna vrednost broja završenih iteracija prve nadgnježdene petlje, a i_m trenutna vrednost broja završenih iteracija m -te najbliže nadgnježdene petlje. Koeficijenti a_i su koeficijenti linearnosti koje u najkraćem intervalu učenja treba da odredi prediktor grananja, a da zatim sa najvećom verovatnoćom vrši predviđanje grananja. Naravno i u fazi učenja, prediktor treba najčešće da pogađa.

Učenje se izvodi na sledeći način: Pretpostavimo da su sve petlje normalizovane {AK87} u trenutku prevođenja. Tada se nakon n_{11} iteracija dogodi da se iskoči iz petlje. Tako se odmah određuje da je $a_0 = n_{11}$. Nakon toga obavi se jedna iteracija nadređene petlje. Ako se sada nakon n_{12} iteracija iskoči iz unutrašnje petlje, tada se dobija vrednost

$$a_1 = n_{12} - a_0.$$

Tako se na kraju dobijaju svi koeficijenti i ukoliko je ispunjena pretpostavka, dobija se dalje pogađanje sa verovatnoćom od 100 %. Hardver kojim se obavljaju ovakve predikcije opisan je u {ČE 95}.

Puštanjem velikog broja reprezentativnih programa, utvrđeno je da pretpostavka važi u velikom broju slučajeva ugnježenih petlji, ali da bazični prediktor, čija je logika upravo opisana, puno greši kada pretpostavka nije ispunjena. Modifikacija prethodnog prediktora, u smislu uprošćavanja, dala je izuzetno dobre rezultate. Uprošćena pretpostavka proizašla iz skupa reprezentativnih tragova {MU 94} {KI 95} ukazivala je da broj iteracija u svakom narednom izvršavanju predstavlja prethodnu vrednost uvećanu za fiksni korak. Ovim se eliminišu sve zavisnosti predikcije od broja iteracija nadgnježenih petlji. Pokazalo se da ovakav prediktor, za sva grananja petlji na kojima je ispitan, daje izuzetno visoku prosečnu verovatnoću pogotka za uslovna grananja kod petlji od 97%. Detalji su opisani u {ČE 95}.