ELEKTROTEHNIČKI FAKULTET, UNIVERZITET U BEOGRADU
PARALELIZAM U ARHITEKTURI RAČUNARA I PROGRAMSKIM PREVODIOCIMA

OPTIMIZACIJE PROGRAMA NA GLOBALNOM NIVOU

doc. dr Maja Vukasović
as. ms Nikola Stanković

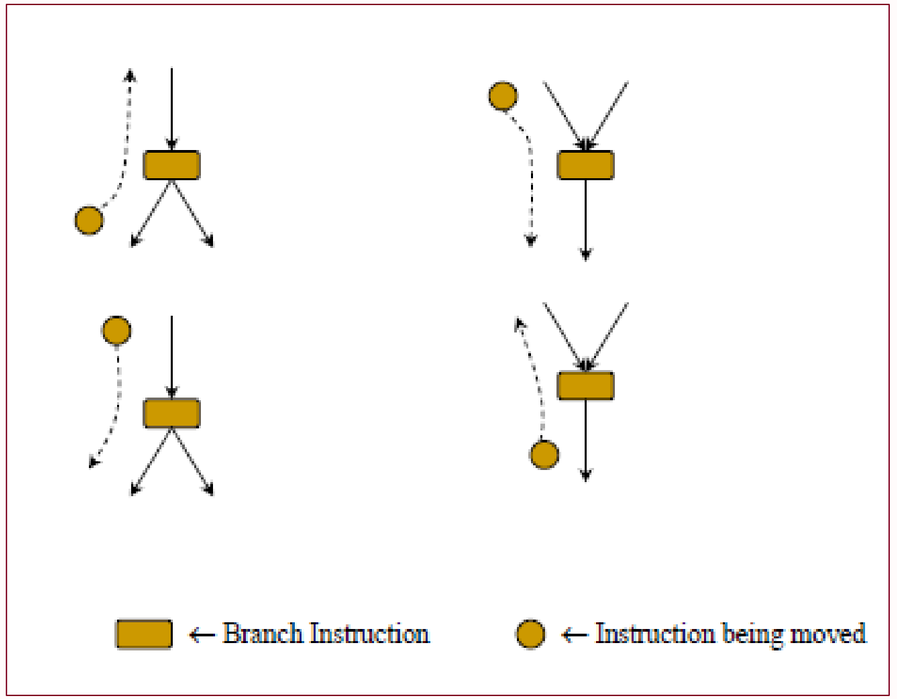
Optimizacija programa na globalnom nivou

- **Graf toka kontrole** – opisivanje strukture programa:
  - Čvorovi – bazični blokovi
  - Grane – uslovna grananja
- **Dinamički trag**
- List Scheduling algoritam – paralelizacija na nivou bazičnog bloka.
- Nedostatak: Ne koristi mogućnost optimizacije izvan granica bazičnih blokova.
- Optimizacija instrukcija na dinamičkom tragu programa – ukoliko bi bio poznat u *compile time* -u, sveo bi se na optimizaciju velikog bazičnog bloka.
- Optimizacija koda za sve moguće dinamičke tragove – **Trace Scheduling** algoritam.



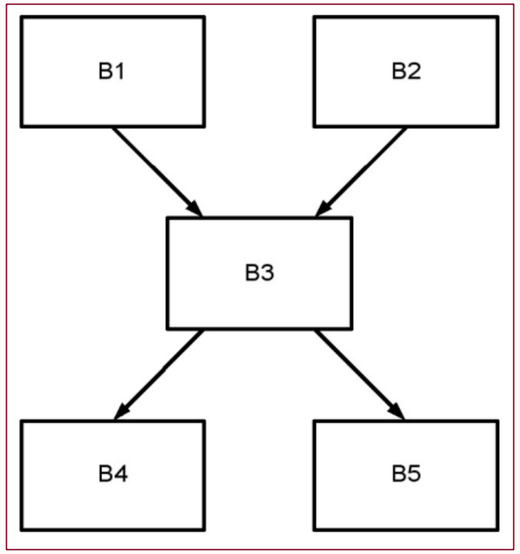
Selidba operacija

- Selidba operacija izvan granica bazičnog bloka – redefinisanje pripadnosti operacija bazičnim blokovima.
- Ubrzavanje izvršavanja bazičnih blokova.
- Kandidati za selidbe operacija:
  - Naviše se mogu seliti isključivo operacije slobodne na vrhu – nezavisne od drugih operacija.
  - Naniže se mogu seliti samo operacije slobodne na dnu – ne postoje operacije zavisne od njihovog rezultata.
- Kada su selidbe operacija moguće?



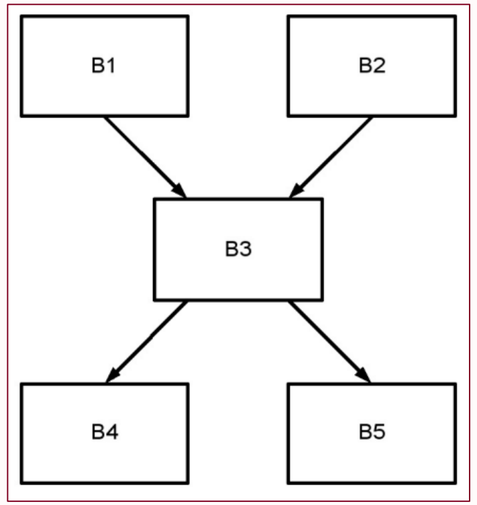
Selidba operacije preko spoja naviše

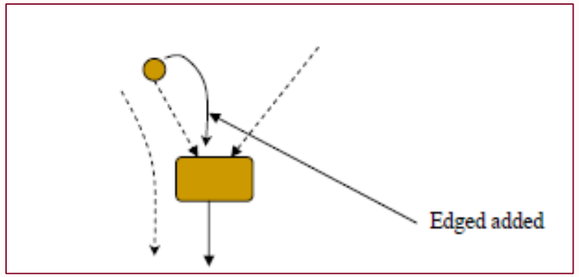
- Operacije slobodne na vrhu bazičnog bloka B3.
- Kopiranje operacija u obe grane.
- Ažuriranje skupova slobodnih operacija na vrhu i na dnu.
- Dalje mogućnosti za selidbu.
- Povećava veličinu generisanog koda.



Selidba operacije preko spoja naniže

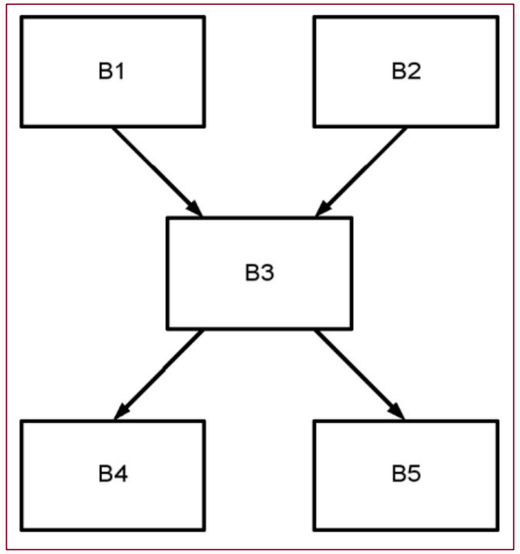
- Inverzna operacija.
- Operacije slobodne na dnu bazičnih blokova B1 i B2.
- Uslov za korektnost je postojanje identičnih operacija u obe grane.
- Smanjenje veličine generisanog koda.
- Ukoliko operaciju ne treba preseliti, dodaje se grana u grafu i time se onemogućava da se ova operacija preseli preko spoja naniže.
- Razlike u performansama kod paralelnih i sekvencijalnih mašina?





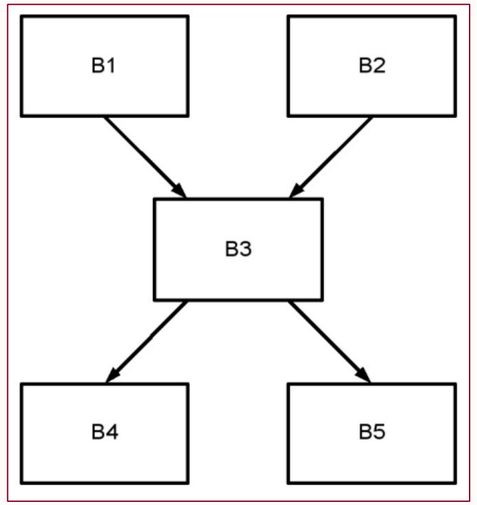
Selidba operacije preko grananja naniže

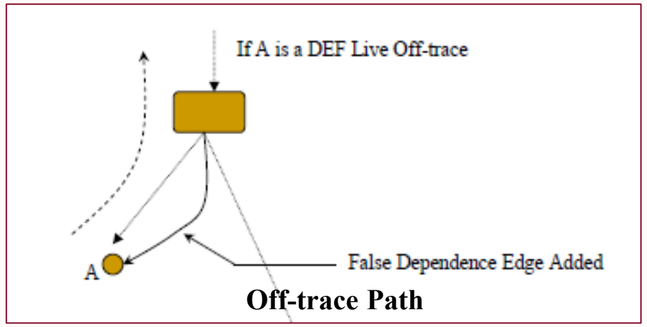
- Operacije slobodne na dnu bazičnog bloka B3.
- Kopiraju se operacije u obe grane.
- Da li operacija generiše uslov grananja?
- Povećanje veličine generisanog koda.
- Ukoliko je operacija mrtva u jednoj grani, onda se ona ne mora replicirati u kodu te grane i nema ekspanzije koda.
  - Primer: *code motion* optimizacija.



Selidba operacije preko grananja naviše

- Uslov korektnosti – kopiraju se identične operacije iz obe grane slobodne na vrhu bazičnih blokova B4 i B5.
- Smanjenje veličine generisanog koda.
- Moguće je preseliti operaciju čak iako ne postoji duplikat u drugoj grani, ako je u toj grani rezultat mrtav.
- Ako nije u redu izvršiti selidbu, dodaje se lažna grana zavisnosti po podacima od operacije koja generiše uslov grananja.

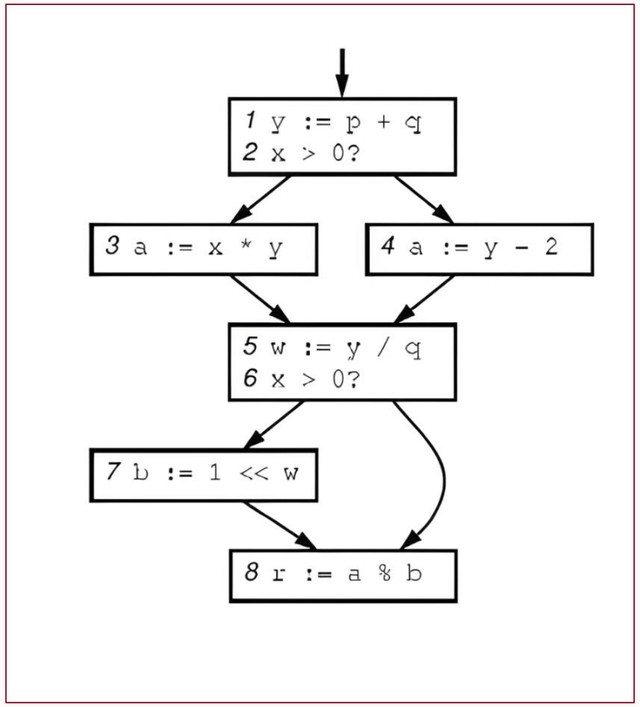




Kontrolne zavisnosti (1)

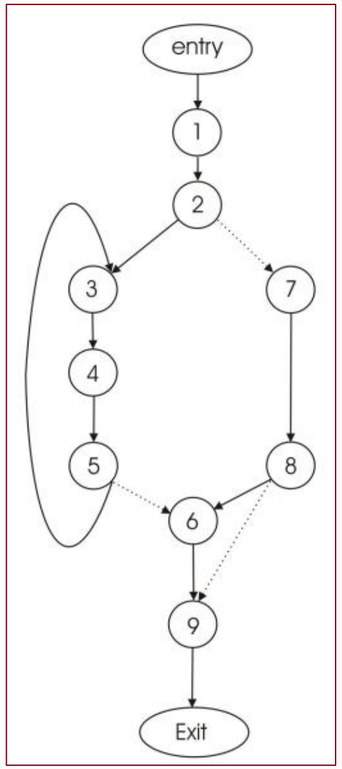


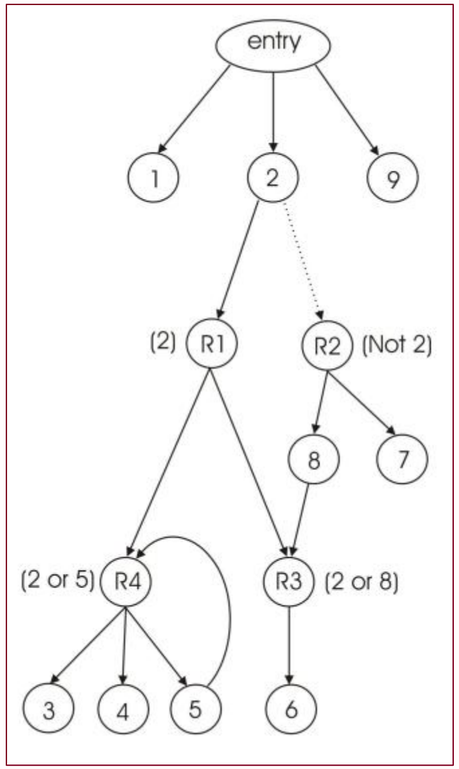
- Zavisnosti u nekom programu – po podacima i kontrolne zavisnosti.
- Čvor B zavisi po kontroli od čvora A (instrukcije modelovane čvorovima) ukoliko se u okviru čvora A izračunava predikat na osnovu koga se odredjuje da li će se nakon A izvršiti instrukcija modelovana čvorom B.
- Da li B mora da čeka sa izvršavanjem iako ne postoji zavisnost po podacima od A?
- Neka dva čvora C i D mogu zavisiti od istih uslova (predikata), iako su udaljeni u grafu toka kontrole.



Kontrolne zavisnosti (2)

- Regionski čvorovi imaju pridružene globalne predikate za modelovanje zavisnosti.





Strategije primene selidbi (1)

1. Lokalna optimizacija na nivou bazičnih blokova.
2. Selidbe operacija izmedju bazičnih blokova.
3. Provera isplativosti selidbi operacija.

- Lokalne optimizacije bez informacija o selidbama.
- Posmatraju se samo susedni bazični blokovi.
- Zauzeće resursa mašine lokalnim optimizacijama.

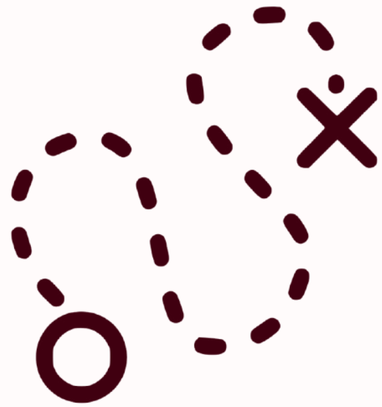
Strategije primene selidbi (2)

- Poboljšana strategija:
  1. Provera isplativosti selidbi operacija u susednim blokovima.
  2. Selidbe operacija.
  3. Lokalna optimizacija na nivou bazičnih blokova.

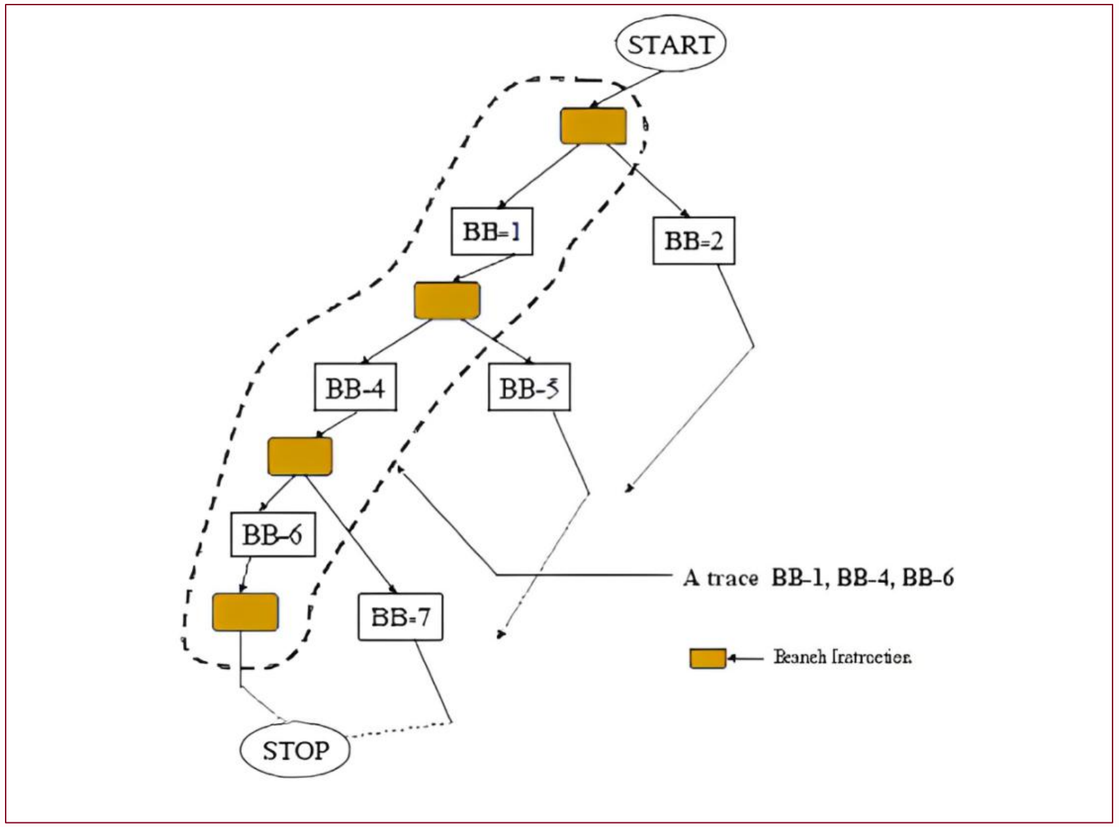
- Optimizuju se modifikovani bazični blokovi (nakon selidbi).
- Selidba se i dalje vrši samo izmedju susednih bazičnih blokova.

Trace Scheduling algoritam

- Mnogo potencijalnih izvršavanja programa.
- Izabrati najverovatniji trag u vreme prevođenja – analize izvršavanja.
- Najverovatniji dinamički trag predstavlja jedan veliki bazični blok – optimizacija List Scheduling algoritmom.
- Petlje se obradjuju na poseban način.
- Nakon optimizacija, promenom rasporeda (paralelizacijom) nekih instrukcija, gube se mesta nekadašnjih grananja i spojeva, pa se mora raditi obnavljanje grananja i spojeva na izabranom dinamičkom tragu.

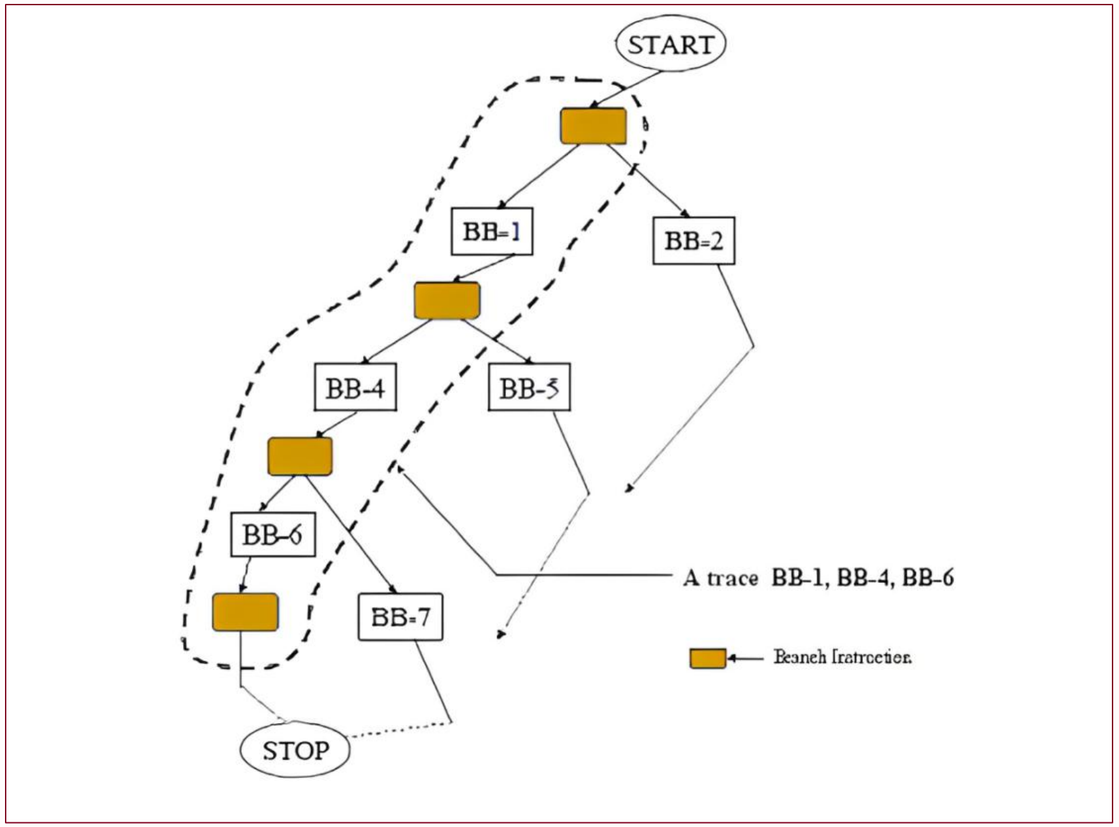


Izbor najverovatnijeg dinamičkog traga (1)



- Procenjuje se ponašanje programa u cilju odabira verovatnijih ishoda pri grananju.
- Podaci se mogu dobiti statističkim analizama ili na osnovu istorije izvršavanja.

Izbor najverovatnijeg dinamičkog traga (2)



- Kreiranje dinamičkog traga započinje od prve instrukcije izvršavanja za kod bez petlje. Nakon toga u skup operacija se dodaju ili operacije trenutnog bazičnog bloka ili, u slučaju grananja, ili skoka operacija koja je najverovatnija za naredno izvršavanje prema statističkim podacima.
- Kasnije izabrani dinamički tragovi se optimizuju sa većim ograničenjima – redosled izbora dinamičkih tragova je bitan!

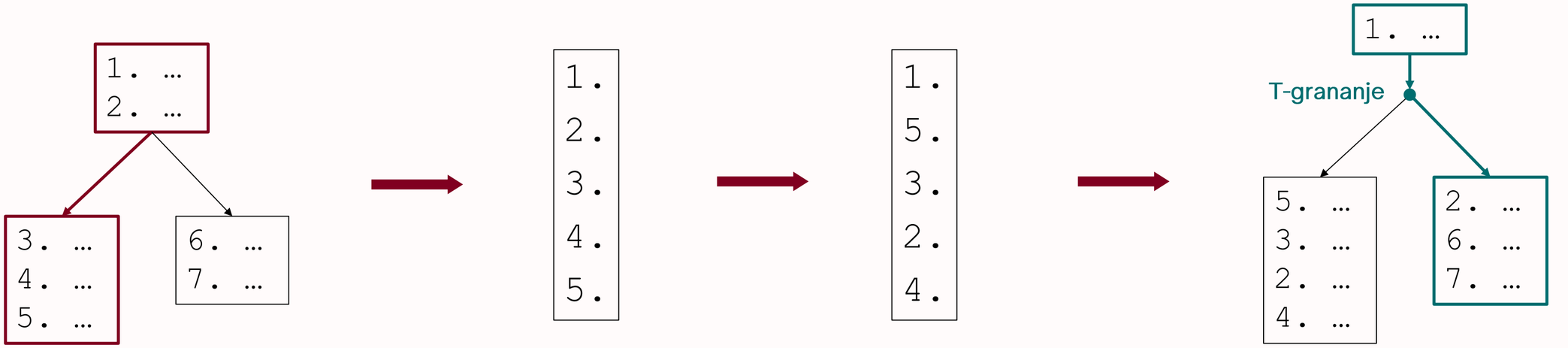
Book Keeping algoritam (1)

- Optimizacijama na nivou traga mešaju se operacije različitih bazičnih blokova i eliminišu se uz skakanja i iskakanja u tragu.
- Potrebno je pronaći nova mesta za grananja i spojeve kako bi se održala korektnost celokupnog koda (*off-trace* grana).
- Obnavljanje grananja i spojeva – **Book Keeping algoritam**.

Book Keeping algoritam (2)

- Unutar traga, promenjen je redosled operacija – neke operacije su preseljene preko spojeva i grananja.
- Pronaći novo mesto grananja za koje je moguće selidbama dobiti semantički ispravan program.
- Novo mesto grananja (T-grananje) – iznad najranije operacije na tragu koja je u originalnom kodu bila iza tačke grananja.
- Zbog izmenjenog redosleda operacija na optimizovanom tragu, T-grananja će se pomeriti pre odredjenog broja operacija koje su se u originalnom kodu nalazile iznad grananja (pravilo selidbi preko grananja naniže).
- Povećanje generisanog koda može biti veliko.
- Bira se najkasnija pozicija za novo mesto grananja koja zadovoljava uslov.
- Da li se može izabrati još kasnija pozicija?

Book Keeping algoritam (3)



Originalan kod sa grananjem

Najverovatniji trag

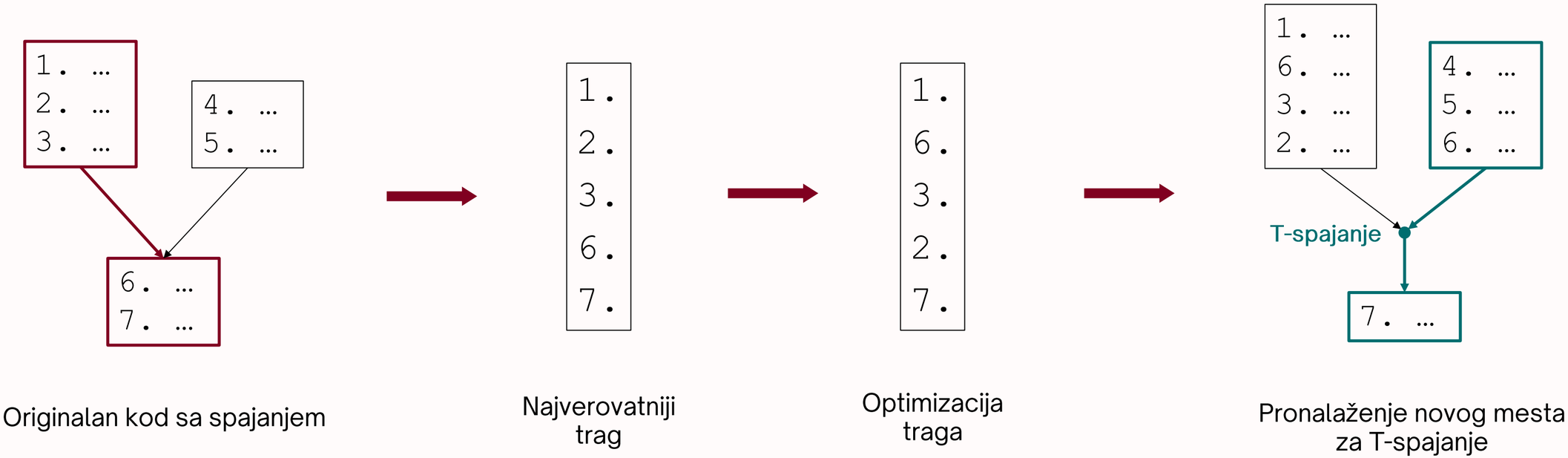
Optimizacija traga

Pronalaženje novog mesta za T-grananje

Book Keeping algoritam (4)

- Novo mesto spajanja (T-spajanje) - iza najkasnije operacije na tragu koja je u originalnom kodu bila iznad spoja.
- Zbog izmenjenog redosleda operacija na optimizovanom tragu, neke operacije koje su se u originalnom kodu nalazile ispod tačke spajanja, sada će se nalaziti iznad T-spajanja (pravilo selidbi preko spoja naviše).
- Povećanje generisanog koda.
- Prebacivanje grananja preko spoja naviše?

Book Keeping algoritam (5)



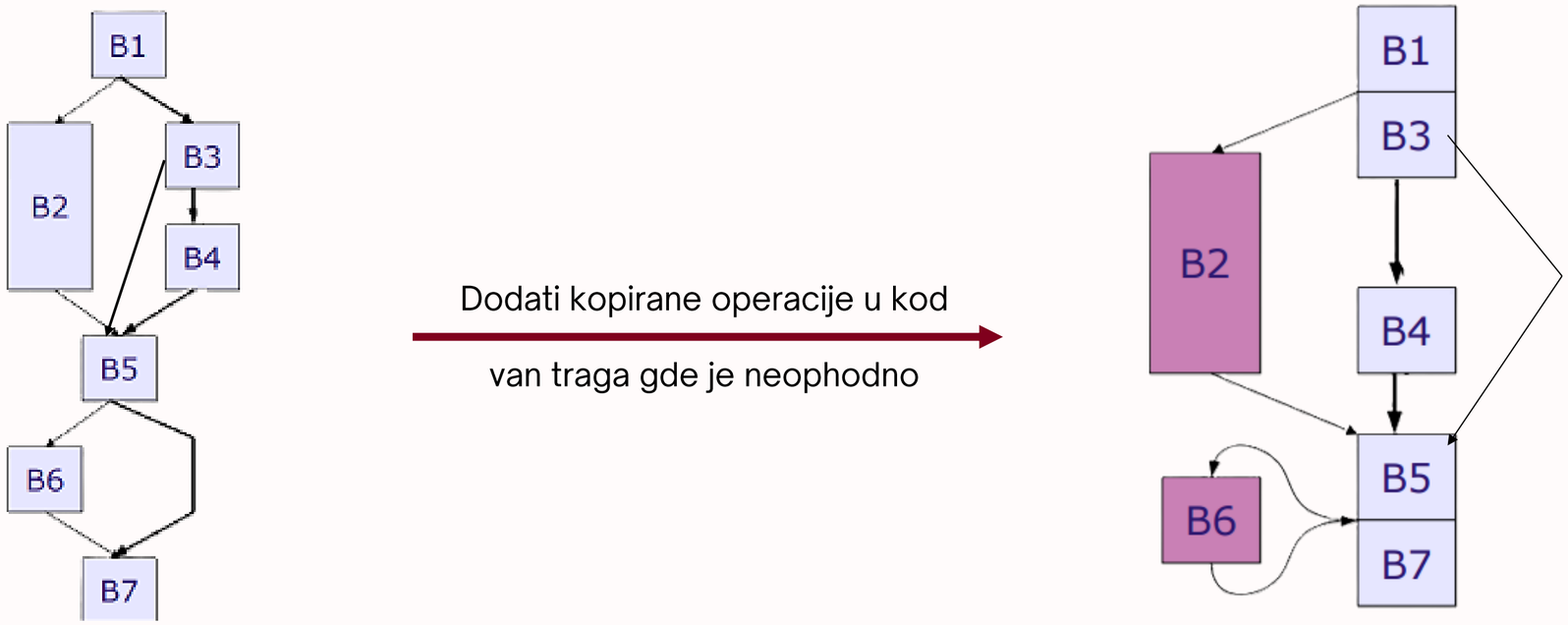
Originalan kod sa spajanjem

Najverovatniji trag

Optimizacija traga

Pronalaženje novog mesta za T-spajanje

Kopiranje operacija



- Na osnovu podataka izabrana je putanja koja uključuje bazične blokove B1, B3, B4, B5 i B7.
- Moguće ubacivanje dodatnih operacija na neku off-trace granu u blokovima B2 i B6.

Tretiranje petlje

- Programske petlje - veliki potencijalni izvor paralelizma.



Razmotavanje petlje

- Osnovna ideja je višestruko ugradjivanje tela petlje direktno u kod u da bi se smanjio broj ukupnih provera uslova, dekrementiranja broja preostalih iteracija i skakanja na adresu početka koda iteracije.
- Broj repliciranih iteracija – **faktor razmotavanja petlje**.
- Inline funkcije – kompajlerska optimizacija.
- Potpuno razmotavanje neprihvatljivo za jako veliki broj iteracija ili slučaj kada ukupan broj iteracija nije poznat u *compile time* -u.

```
sum = 0;
for(i = 0; i < n; i++)
  sum += array[i];
```

```
sum = 0;
for(i = 0; i < n; i += 4) {
   sum0 += array[i];
   sum1 += array[i+1];
   sum2 += array[i+2];
   sum3 += array[i+3];
}
sum = sum0 + sum1 + sum2 + sum3;
```

Operacije invarijantne u odnosu na petlju

- Operacije koje se ne menjaju u zavisnosti od iteracija u kojima se izvršavaju.
- Operacija invarijantna u odnosu na petlju može da se izvrši pre petlje, a njen rezultat da se koristi u okviru onih iteracija petlje gde je potreban.
- Procena vremena ukupnog izvršenja u skladu sa postojećim resursima.

```
for (i = 0; i <= n; i++) a[i] = i + n * 25;
```

Loop peeling

- Nekada je potrebno reformatirati petlju u cilju povećanja paralelizma zbog postojanja odredjenog broja iteracija koje onemogućuju paralelizaciju.
- Odredjeni broj prvih i/ili poslednjih iteracija se izdvajaju ispred petlje i izvršavaju odvojeno.
- U slučaju fiksnog broja iteracija ispitivanje nije potrebno.

```
for (i = 1; i <= n; i++) a[i] = a[1] * b[i];
```

izdvajanje prve iteracije

```
if (n > 0) a[1] *= b[1];
for (i = 2; i <= n; i++) a[i] = a[1] * b[i];
[a(2:n) = a(1) * b(2:n);]
```

Loop peeling - primer

```
w = 0; j = 1;
for (i = 1; i <= n; i++) {
  a[i] = b[j] + b[i];
  c[i] = b[i] * c[j];
  j = i - w;
  w = 1;
}
```



```
if (n > 0) {
  a[1] = b[j] + b[1];
  c[1] = b[1] * c[j];
  j = 1 - w;
  w = 1;
}
if (n > 1) {
  a[2] = b[j] + b[2];
  c[2] = b[2] * c[j];
  j = 1;
}
for (i = 3; i <= n; i++) {
  a[i] = b[i-2] + b[i];
  c[i] = b[i] * c[i-2];
}
```

- Prve dve iteracije se razlikuju od ostalih – moguća eliminacija zavisnosti po j.
- Dodela vrednosti promenljivoj w – invarijantna u odnosu na petlju.