Profajliranje uzimanjem uzoraka. Algoritmi uzorkovanja.

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Vladimir Bošković, 1086/2018
vladimir.boskovic89@gmail.com

12. decembar 2018

Sažetak

Profajliranje predstavlja vid dinamičke analize koda koji za cilj ima prikupljanje informacija o radu programa u toku njegovog izvršavanja, kako bi se na osnovu tih rezultata izvršile odgovarajuće optimizacije koda. Postupak profajliranja se zasniva na umetanju određenih instrukcija u određenim delovima koda, čiji je zadatak da beleže stanje programa u trenutku poziva. Te instrukcije nekada mogu uticati na performanse samog programa. U ovom tekstu biće opisan metod profajliranja uzimanjem uzoraka koji za cilj ima smanjenje negativnog uticaja profajliranja na perforamnse samog programa. Potom će biti opisani algoritmi uzorkovanja sa i bez duplirajućeg koda. Na kraju ćemo uporediti opisane tehnike, koje ćemo prikazati kroz rezultate njihovog rada i izvesti odgovarajuće zaključke.

Sadržaj

1 Uvod

Da bi se programski kod optimizovao, neophodno je izvršiti odgovarajuću analizu. Analiza koda se može podeliti na statičku i dinamičku. Statička analiza koda se sprovodi bez izvršavanja samog programa, dok se dinamička analiza programa sprovodi u toku njegovog izvršavanja. Profajliranje spada u vid dinamičke analize programa. Rezultat profajliranja je skup podataka - tzv. *profil* programa, dobijen u toku njegovog rada, na osnovu kojih je moguće izvući odgovarajuće zaključke i potom optimizovati kod. Ovaj postupak podrazumeva dodavanje odgovarajućih instrukcija unutar programa, čiji je zadatak da beleže stanje programa u datom trenutku (na primer stanje u memoriji, broj promašaja u keš memoriji), broje poziv odgovarajućeg bloka koda itd. Postupak dodavanja ovih instrukcija naziva se *instrumentalizacija*. Poželjno je da se instrumentalizacijom dobijaju samo potrebni podaci o programu, da ne utiče na funkcionalnost i ne usporava program [2]. U određenim slučajevima instrumentalizovan kod može značajno usporiti rad programa. Jedan od načina kojim se može umanjiti usporavanje programa instrumentalizacijom jeste profajliranje uzimanjem uzoraka (eng. *sampling*). Karakteristika ovog pristupa je da se u određenim vremenskim intervalima izvršava instrumentalizovani kod, a u određenim intervalima originalni kod. Na taj način se umanjuje usporavanje programa izazvanog instrumentalizacijom, a opet se mogu dobiti podaci o samom programskom kodu i formirati profil programa sa velikom tačnošću (u odnosu na potpuno istrumentalizovan kod). Pokazuje se da se ovim algoritmima usporenje programa smanjuje na 6% u odnosu na orignialni kod, dok preciznost dobijenih profila se može poklokpiti i do 98-99% sa idealnim profilom [1].

U poglavlju 2 opisaćemo sam algoritam profajliranja uzimanjem uzoraka. U poglavlju 3 opisaćemo varijante ovog algoritma koji troše manje memorije (eng. *space saving varations*). U poglavlju 4 ćemo uporediti opisane algoritme i prikazati rezultate njihovog izvršavanja, sa posebnim osvrtom na vreme izvršavanja instrumentalizovanog koda i preciznosti profila koje ovi algoritmi postižu, kao i izneti zaključke.

2 Profajliranje uzimanjem uzoraka

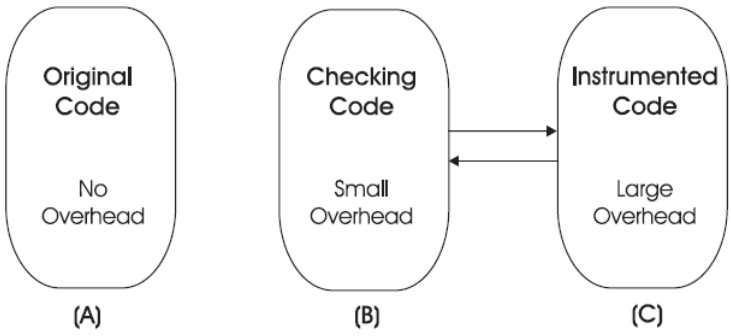
Kod nekih aplikacija, kao što su neke serverske aplikacije koje dugo vremena rade, jedini način da se izvrši optimizacija koda jeste da se izvrši njihovo profajliranje i da se na osnovu dobijenih podataka izvedu odgovarajući zaključci o budućim optimizacijama. Kod takvih aplikacija proces instrumentalizacije može negativno uticati na performanse, a u nekim slučajevima je usporenje programa toliko da ga je instrumentalizaciju čak nemoguće primeniti. U takvim slučajevima možemo primeniti algoritam profajliranja uzimanjem uzoraka.

Algoritam profajliranja uzimanjem uzoraka se zasniva postepenom izvršavanju originalnog i instrumentalizovanog koda, pri čemu su intervali izvršavanja jednog i drugog koda precizno definisani. Karakteristike rešenja koje ćemo predstaviti su:

- Algoritam ne zavisi od hardvera ili operativnog sistema na kojem se izvršava
- Algoritam je fleksibilan i mogu mu se podešavati i vrednosti intervala u kojima će se izvršavati originalni ili instrumentalizovani kod

- Instrumentalizacija se može izvršavati duži vremenski period, bez preteranog uticaja na pad performansi samog programa
- Mogu se koristiti različite tehnike instrumentalizacije na različitim delovima koda
- Usporenje programa je kontrolisano od strane samog algoritma

Kao što je prikazano na slici 1, algoritam razlikuje tri verzije koda programa - originalni kod, proveravajući kod (eng. *checking code*) i instrumentalizovani kod (eng. *instrumented code*).



Slika 1: Tri verzije koda

Poveravajući kod je gotovo identičan originalnom kodu, pri čemu su mu dodate instrukcije kojima se proverava uslov prelaska u instrumentalizovani kod. Te instrukcije za proveru uslova ne utiču značajno na performanse samog programa. Osnovna ideja ovog algoritma jeste da se najveći deo vremena program izvršava unutar metoda proveravajućeg koda i time ne uspori značajno. Ipak u precizno određenim uzorcima (eng. *samples*), program će izvršavati instrumentalizovane blokove koda i tada doprineti formiranju njegovog profila.

Na slici 2 može se videti tok između proveravajućeg i instrumentalizovanog koda.

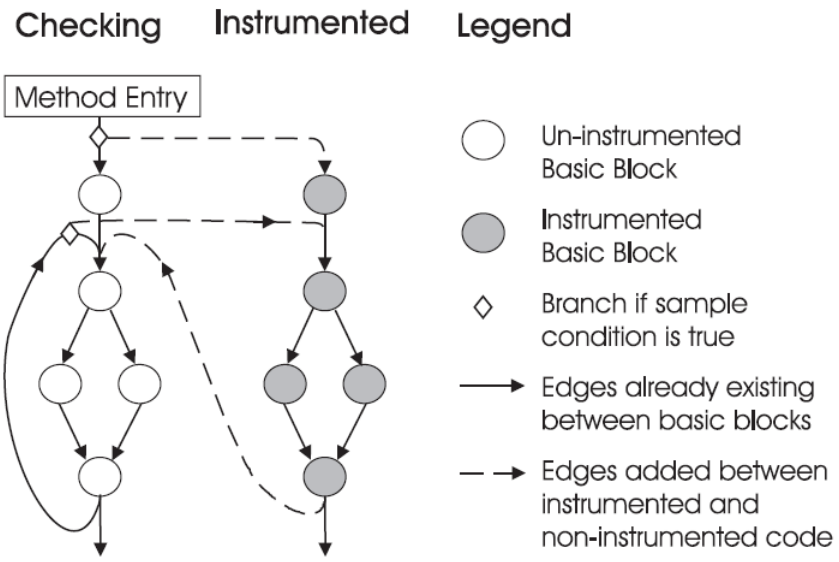
U proveravajući kod se dodaju na određenim mestima uslovi prelaska (eng. *checks*) koji proveravaju uslove prelaska (eng. *sample condition*). Kada je uslov prelaska ispunjen, izvršavanje prelazi u instrumentalizovani kod. Uslovi prelaska su postavljeni na početak svake metode, kao i na sve povratne grane (eng. *backedges*).

Instrumentalizovni kod je takođe modifikovan na način da nema povratnih grana, već se na tim mestima kontrola vraća na odgovarajuće mesto u proveravjućem kodu. Odnos između vremena izvršavanja programa unutar proveravajućeg i instrumentalizovanog koda se može podešavati i potrebno ga je precizno odrediti kako bi se umanjilo usporenje programa i u isto vreme dobili dovoljno precizni podaci.

Broj izvršenih provera uslova je nezavisan od samog načina instrumentalizacije i jednak je zbiru broja ulazaka u metodu (ili blok koda) i broja povratnih grana [1].

2.1 Mehanizmi provere uslova

Uslovi prelaska u instrumentalizovani bi trebalo pažljivo da budu odabrani tako da se veći deo vremena kod izvršava u proveravajućem kodu,



Slika 2: Tok između proveravajućeg i instrumentalizovanog koda

kako se performanse programa ne bi narušile. Druga, čak i važnija osobina izbora uslova prelaska jeste da broj poziva instrumentalizovanih blokova koda bude proporcionalan broju poziva tih blokova u proveravajućem kodu.

Jedan pristup za realizaciju uslova prelaska jeste da se koristi odgovarajuća hardverska komponenta ili sistem prekida operativnog sistema. Na primer može se koristi odgovarajući bit prekida koju bi uslov prelaska proveravao, a koji bi postavljao operativni sistem ili odgovarajući hardver. Međutim, mana ovog pristupa jeste da učestalost prelaska u instrumentalizovani kod zavisi od učestalosti samog prekida operativnog sistema. Druga mana mana ovog pristupa jeste činjenica da se ne može preći u instrumentalizovani kod istog trenutka kada se desi prekid, već tek kada se dođe do koda za proveru uslova prelaska.

Drugi pristup za realizaciju uslova prelaska jeste da se koristi sofversko brojanje pristupa određenom bloku i na taj način donosi odluka o eventualnom prelasku u instrumentalizovani blok. Na taj način moći ćemo da omogućimo proporcionalan broj poziva koda za uslov prelaska (u proveravajućem kodu) i broja poziva odgovarajućeg instrumentalizovanog bloka. Ovaj mehanizam nazivamo uzorkovanje zasnovano na brojanju (eng. *counter-based sampling*) i koje ćemo detaljnije opisati.

Za realizaciju mehanizma zasnovanom na brojanju, dovoljno je samo da kompajler u originalni kod ubaci deo koda za proveru uslova. To je jednostavna sekvenca instrukcija koja dekrementira i proverava brojač i prikazana je na slici 3. Postoji više načina da se realizuje ovaj način, a jedan od njih jeste da kompajler baš ubaci na mesta provere kod koji je prikazan, a da se za brojač koristi registar ili keš memorija. U tom slučaju mehanizam provere uslova troši jako malo resursa i veoma malo utiče na usporenje proveravajućeg koda.

Pored toga što je mehanizam provere zasnovan na brojanju jednosta-

van za implementaciju i minimalno zauzima resurse u toku izvršavanja programa, postoje i druge prednosti. Ovim pristupom se može precizno definisati i jednostavno sofverski postaviti učestalost prelaska u instrumentalizovani kod, bez oslanjanja na hardver ili operativni sistem [1].

```
eventCounter--;
if (eventCounter == 0) {
   takeSample();
   eventCounter = resetValue;
}
```

Slika 3: Provera brojača u kod provere uslova zasnovane na brojanju

3 Varijante algoritma bez duplirajućeg koda

Algoritam opisan u prethodnom poglavlju koristi po dve verzije (u proveravajućem i instrumentalizovanom kodu) svakog bloka koji se instrumentalizuje i samim tim zauzima duplo više prostora za svaki blok u odnosu na originalni kod. To može u nekim slučajevima biti značajno uvećanje prostora, a takođe može dovesti i do povećanog vremena kompajliranja.

Opisivajući algoritam u poglavlju 2, naveli smo osobinu ovog pristupa da je ukupan broj provera uslova prolaska u instrumentalizovanu verziju neke metode ili bloka, jednak zbiru broja poziva te metode (ili bloka) i broju povratnih grana u toj metodi. Ukoliko bismo želeli da smanjimo veličinu instrumentalizovanog bloka, uklanjajući njegove delove, tj. čvorove koji se ne instrumentalizuju, morali bismo da dodamo dodatne uslove prelaska (u nekim slučajevima) i na taj način narušimo pomenuto svojstvo, što je prikazano na slici 4.

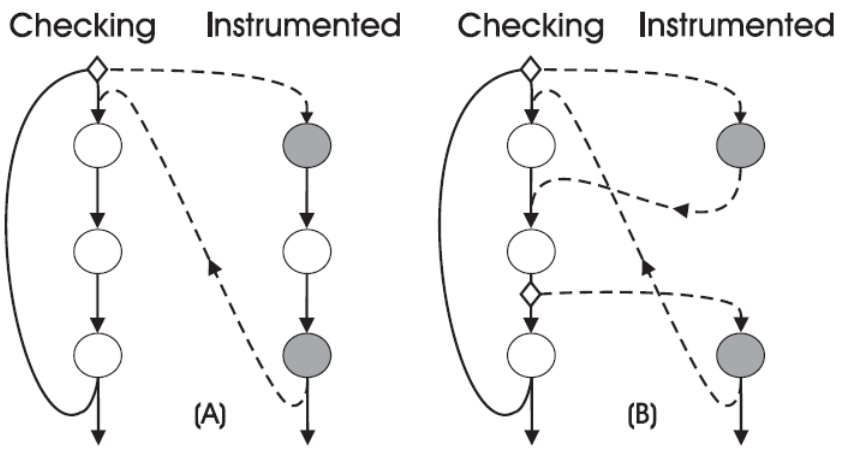
U ovom poglavlju opisaćemo dve varijante algoritma profajliranja zasnovanog na uzimanju uzoraka. Prva varijanta ne narušava opisano svojstvo, dok ga druga narušava.

3.1 Varijanta 1 algoritma bez duplirajućeg koda

Varijanta 1 algoritma jednostavno izbacuje sve neinstrumentalizovane čvorove iz instrumentalizovanog koda tako da ne naruši svojstvo o broju uslova provere koje je opisano u poglavlju 2. To se može postići izbacivanjem dva tipa čvorova - čvorova na vrhu bloka (eng. *top-nodes*) i čvorova na dnu bloka (eng. *bottom-nodes*).

Čvorom na vrhu bloka označavamo bilo koji čvor u bloku takav da takav da sve putanje od ulaska u blok do tog čvora prolaze isključivo kroz neinstrumentalizovane čvorove. Izbacivanjem čvorova na vrhu bloka neće se narušiti svojstvo o broju uslova provera prelaska, mada ipak neke modifikacije u proveravajućem kodu moramo napraviti. Potrebno je izmeniti putanje ka čvorovima instrumentalizovanog bloka koji su uklonjeni, tako da pokazuju na instrumentalizovani čvor koji nije uklonjen.

Čvorom na dnu bloka označavamo svaki neinstrumentalizovani čvor u instrumentalizovanom bloku takav da od njega nijedna putanja ne vodi



Slika 4: Izbacivanje neinstrumentalizovanih čvorova iz instrumentalizovanog koda tako da dovodi do povećanja broja uslova provere

do instrumentalizovanog bloka. Uklanjanjem svih čvorova na dnu bloka nećemo narušiti svojstvo o broju uslova provera [1].

3.2 Varijanta 2 algoritma bez duplirajućeg koda

Druga varijanta algoritma bez duplirajućeg koda narušava svojstvo o broju uslova provere prelaska. U ovoj varijanti algoritma se jednostavno izbacuju svi neistrumentalizovani čvorovi u instrumentalizovanom bloku i na taj način se postiže maksimalna ušteda prostora u kojoj zapravo nema duplirajućeg koda (neinstrumentalizovanog). To se postiže dodavanjem dodatnih provera uslova prelaska kao što je prikazano na slici 4. Iako postoji određenih razlika u realizaciji prelaska iz proveravajućeg koda u instrumentalizovani u odnosu na algoritam sa duplirajućim kodom, odnos broja izvršavanja čvorova u proveravajućem kodu i njihovih instrumentalizovanih verzija ostaje isti.

Mana ove varijante u odnosu na varijantu 1 jeste u tome što to može izvršiti veći broj provera uslova prelaska u proveravajućem kodu što u nekim slučajevima može dovesti do značajnijeg usporavanja proveravajućeg koda [1].

4 Eksperimentalni rezultati

U eksperimentima koji su sprovedeni meri se usporenje profajliranog koda u odnosu na originalni, kao i preciznost profila dobijenih opisanim algoritmima profajliranja zasnovanih na uzorkovanju.

Na slici 5 prikazano je usporavanje programa izazvanog profajliranjem. U prvoj koloni prikazani su test primeri. U drugoj koloni su procentualno izražena usporavanja programa primenom instrumentalizacije koja broji pozive metoda, dok su u drugoj koloni prikazane vrednosti usporavanja programa primenom instrumentalizacije koja broji pristupe atributima klasa [1].

Benchmark	Call edge	Field accesse
201_compress	77.1	286.5
202_jess	140.7	90.2
209_db	4.5	15.3
213_javac	75.3	15.4
222_mpegaudio	132.5	59.2
227_mtrt	121.6	76.4
228_jack	28.3	113.2
opt-compiler		
pBOB		
Volano		
Geom. Mean	82.8	94.1

Slika 5: Usporenje originalnog programa izavanog profajliranjem (bez primene algoritama uzorkovanja)

Na slikama 6 i 7 prikazano je usporavanje proveravajućeg koda izazvanog dodavanjem uslova provere prelaska. Vrednosti se ne odnose na isntrumentalizovani kod. Varijanta 0 na slici se odnosi na algoritam profajliranja uzorkovanjem sa duplirajućim kodom. Može se videti da je u proseku ukupno usporenje proveravajućeg koda 6% [1].

Variations 0 & 1		
Both	Backedges	Method entry
11.20	9.36	5.62
5.65	4.32	5.23
1.29	1.65	1.10
1.97	1.22	4.51
10.28	7.40	3.94
5.83	0.73	5.00
5.40	3.20	3.30
5.94	3.98	4.10

Slika 6: Usporenje proveravajućeg koda za originalni algoritam i varjantu 1 algoritma

Na slici 8 prikazano je usporenje programa i preciznost dobijenog profila (o odnosu na profil koji se dobija izvršavanjem samo instrumentalizovanog koda - prvi red). Prva kolona *interval uzorkovanja* (eng. *sample*

Variation-2		
Call edges	Field accesses	Space Reduction (K)
5.62	101.9	
5.23	31.8	
1.10	4.2	
4.51	3.2	
3.94	22.8	
5.00	59.0	
3.30	29.3	
4.10	36.0	

Slika 7: Usporenje proveravajućeg koda za varijantu 2 algoritma

interval) označava učestalost prelaska u instrumentalizovani kod. Na primer ukoliko je vrednost u toj koloni 100, to znači da će svaka 100. provera uslova u proveravajućem kodu završiti u instrumentalizovanom, što znači da kako ova vrednost raste, opada preciznost dobijenog profila, ali i usporenje čitavog programa [1].

	Call Edge			
	Variation-0		Variation-2	
Sample Interval	Overhead	Accuracy	Overhead	Accuracy
1	89.03	100	108.90	100
10	8.10	99	11.24	99
100	0.72	99	1.33	98
1,000	0.05	95	0.18	97
10,000	0.07	89	0.07	89
100,000	0.03	62	0.09	67

Slika 8: Precizsnost i usporenje programa na koji je primenjen algoritam profajliranja uzorkvanjem (originalni algoritam i varijanta 2)

Na slici 9 prikazana je preciznost dobijenih profila kod algoritama profajliranja uzorkovanjem koji koriste mehanizam provere uslova zasnovanog na bitu prekida i algoritma koji koristi mehanizam provere uslova zasnovanog na brojanju [1]. Vrednosti su izražene u procentima i predstavljaju preciznost dobijenih profila u odnosu na idealni profil. Može se primetiti da su profili kod verzije algoritma koja koristi mehanizam provere uzorka zasnovan na softverskom brojanju u proseku precizniji [1].

Benchmark	Time-based	Counter-based
201_compress	89	99
202_jess	88	94
209_db	51	94
213_javac	66	72
222_mpegaudio	73	94
227_mtrt	83	78
228_jack	85	91
opt-compiler		
pBOB		
Volano		
Geom. Mean	76	89

Slika 9: Preciznost profila kod algoritma profajliranja uzorkovanjem koji koristi mehanizam provere uslova zasnovanog bitu prekida (hardverski ili OS) i algoritma koji koristi mehanizam provere zasnovanog na brojanju

5 Zaključak

Na osnovu rezultata sa slike 5 možemo videti da u nekim slučajevima profajliranje može dovesti do značajnog usporavanja originalnog koda, što nekad onemogućava primenu samog profajliranja, već je potrebno primeniti neku drugu metodu kao što je profajliranje uzorkovanjem.

S obzirom da profajliranje uzorkovanjem originalni kod menja proveravajućim kodom, poželjna osobina proveravajućeg koda je da ne bude previše usporen dodavanjem instrukcija za proveru prelaza u instrumentalizovani kod. Rezultati sa slika 6 i 7 to pokazuju.

Takođe profajliranje uzorkovanjem bi imalo smisla jedino ukoliko bi profili nastali njegovom primenom bili dovoljno precizni da se iz njih mogu izvesti odgovarajući zaključci i na osnovu njih optimizovao kod. Rezultati eksperimenata prikazanih na slici 8 pokazju da se veoma precizni profili mogu dobiti i u situacijama kada se puno više vremena kod izvršava u proveravajućem kodu.

Literatura

[1] M. Arnold and B. G. Ryder. A Framework for Reducing the Cost of Instrumented Code. *Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, (42):168–179.

[2] master rad. Nikola Prica. Podrška za profajliranje softvera za uređaje sa ugrađenim računarom. *Univerzitet u Beogradu, Matematički fakultet*, (42).