

Optimizacije u okviru kompajlera GCC/LLVM

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Tamara Stojković, Emilija Stošić, Teodora Isailović
tamara.stojkovic.1998@gmail.com, emilijazstosic@gmail.com,
teodora.isailovic@gmail.com

12. novembar 2022.

Sažetak

Ovo se piše na kraju.

Sadržaj

1	Uvod	2
2	Osnovna podela optimizacija	2
2.1	Optimizacije međukoda	2
2.1.1	Lokalne optimizacije	2
2.1.2	Globalne optimizacije	3
2.1.3	Međuproceduralne optimizacije	4
2.2	Optimizacije koda	4
3	Prvi naslov	4
3.1	Prvi podnaslov	4
3.2	Drugi podnaslov	4
3.3	... podnaslov	4
4	n-ti naslov	5
4.1	... podnaslov	5
5	Zaključak	5
	Literatura	5
A	Dodatak	5

1 Uvod

Optimizacija predstavlja tehniku transformacije dela programa, tako da kod bude što je moguće efikasniji. Za cilj ima poboljšanje performansi koda, a ne savršen rezultat (uglavnom se ne može se reći da predstavlja pronalazak „optimalnog rešenja“). Optimizacija je oblast u kojoj se danas vrši većina istraživanja kompajlera. U okviru savremenih kompajlera dostupan je veliki broj optimizacija koje sve međusobno deluju na razne načine i imaju uticaj na kvalitet koda, veličinu koda, vreme kompilacije, potrošnju energije itd. Kompajleri obično obezbeđuju ograničen broj standardnih optimizacionih nivoa, kao što su O1, -O2, -O3 i -Os. Na ovim nivoima ostvaruju se kompromisi između različitih mera kao što su kvalitet koda, veličina koda i vreme kompilacije. U ovom radu ćemo upravo predstaviti vrste optimizacija, sa akcentom na one koje su dostupne u okviru kompajlera GCC i LLVM.

2 Osnovna podela optimizacija

Postoje razne tehnike u primeni procesa optimizacije. Razlikujemo optimizacije koje se primenjuju na međukod, kao deo generisanja ciljnog koda, ali i nakon generisanja ciljnog koda. **Optimizacija međukoda** primenjuje se da se kod pojednostavi, preuredi ili sažme. Predstavlja optimizaciju koja ne uzima u obzir specifičnost ciljne arhitekture. Neke od stavki koje **optimizacija kao deo generisanja ciljnog koda** omogućava su biranje instrukcija i određivanje načina alociranja objekata. Na kraju, može se izvršiti i **optimizacija ciljnog koda**, gde se pokušava prerada samog asemblerskog koda u nešto efikasnije. U ovom slučaju zahteva se detaljno poznavanje ciljne arhitekture, kao i asemblerskog i mašinskog jezika ciljnog programa.

2.1 Optimizacije međukoda

U okviru optimizacije međukoda razlikujemo :**lokalne, globalne i međuproceduralne optimizacije**.

2.1.1 Lokalne optimizacije

Lokalne optimizacije služe za ubrzavanje malih delova neke funkcije i rade sa konkretnim naredbama unutar osnovnog bloka(eng. basic block) One su obično najlakše za izvođenje jer nije neophodno raditi analizu kontrole toka(eng. *Control - flow*). U nastavku navodimo neke od tehnika lokalne optimizacije koje se vrše nad međukodom.

- **Eliminacija čestih podizraza (eng. *Common subexpression elimination*)**
Podrazumeva izbegavanje izračunavanja čestog izraza više puta(ako se javlja više od jednom). U slučaju da dve operacije daju isti rezultat, kažemo da su česte, pa je bolje izračunati izraz jednom i pozvati ga naredni put na mestu gde je neophodno.
- **Slaganje konstanti (eng. *Constant folding*)**
Podrazumeva da se konstantni izrazi mogu evaluirati u vreme kompilacije. To uključuje utvrđivanje da svi operandi u izrazu imaju konstantnu vrednost, a zatim menjanje izraza njegovom vrednošću. Često je i lepše za pisanje i čitanje.

- **Propagacija kopija (eng. *Copy propagation*)**
Podrazumeva izbegavanje uvođenja promenljivih, koje samo čuvaju vrednosti nekih promenljivih koje već postoje. Posebno je značajna jer je u stanju da eliminiše veliki broj instrukcija, koje služe samo za kopiranje vrednosti iz jedne promenljive u drugu.
- **Smanjenje snage operatora (eng. *Operator strength reduction*)**
Podrazumeva zamenu operatora „jeftinijim operatorom“. Od same arhitekture zavisi koje operacije se smatraju najjeftinijim, a koje najskupljim, tako da je za dobru optimizaciju neophodno poznavati ciljnu mašinu.
- **Eliminacija mrtvog koda (eng. *Dead code elimination*)**
Podrazumeva da ako se rezultat neke instrukcije nadalje ne koristi, instrukcija se smatra „mrtvom“ i može biti uklonjena. Ovo omogućava uprošćavanje koda, izbacivanjem nepotrebnih izračunavanja.
- **Algebarsko pojednostavljenje i reasocijacija (eng. *Algebraic simplification and reassociation*)**
Podrazumeva pojednostavljenje izraza korišćenjem algebarskih svojstava (zakona algebre). Reasocijacija podrazumeva korišćenje svojstava kao što su asocijativnost, komutativnost i distributivnost kako bi se nakon toga omogućila lakša primena drugih optimizacija.
- **Kompozicija lokalnih transformacija**
Različite optimizacije koje smo do sada videli, brinu o malom delu koda. Za maksimalan efekat, možda će morati neke optimizacije da se primene više puta.

2.1.2 Globalne optimizacije

Predstavljaju optimizacije koje su slične lokalnim i koje možemo primeniti sa nekim dodatnim analizama. „Globalno“ u ovom slučaju ne znači u celom programu, već se optimizacije primenjuju na jednu po jednu funkciju. Ova analiza je znatno moćnija od lokalne, ali dosta komplikovanija. Mnoge lokalne optimizacije se mogu se primenljivati i globalno, dok postoje i one koje se mogu primeniti samo na globalnom nivou. Neke globalne optimizacije date su u nastavku.

- **Globalna eliminacija mrtvog koda**
U ovom slučaju jedina razlika je što se informacije o tome gde je neka promenljiva živa, moraju dobiti globalnom analizom, što je značajno komplikuje u odnosu na lokalnu eliminaciju mrtvog koda.
- **Globalno propagiranje konstante**
Ovo je optimizacija koja svaku promenljivu za koju se zna da je konstantna menja sa tom konstantom. Potrebno je pratiti vrednosti koje mogu biti dodeljene promenljivoj u svakoj tački programa.
- **Globalna eliminacija čestih podizraza**
Česti podizrazi mogu se eliminisati i na globalnom nivou, poznavanjem skupa dostupnih izraza. Izrazi su dostupni u jednom trenutku ako su živi pri ulasku u blok, što se utvrđuje ospežnom analizom.
- **Optimizacija kretanje koda (eng. *Code motion*)**
Postoje optimizacije koje se mogu primeniti samo globalno i jedna takva je ova i ona objedinjuje sekvence koda zajedničke za jedan ili više blokova. Cilj je smanjiti veličinu koda i eliminisati eventualna

skupa ponovna izračunavanja. Dva oblika optimizacije izdizanjem koda koje ćemo izdvojiti su :

- **Pomeranje invarijantnog koda**(eng. *loop-invariant*)
Podrazumeva da se računanje vrednosti neke promenljive može izdvojiti izvan petlje, ako je nezavisno od brojača petlje.
- **Parcijalna eliminacija suvišnosti** (eng. *Partial redundancy elimination*)
Za neki račun u programu kaže se da je suvišan, ako računa već poznatu vrednost. Parcijalno redundantno izračunavanje je ono čija je vrednost poznata samo u nekim delovima.

2.1.3 Međuproceduralne optimizacije

U okviru ove optimizacije jedna od najznacajnijih tehnika je **uvlačenje definicija funkcija**(eng. *Inlining*). Uglavnom se jednostavnije funkcije uvlače, a složenije ne. Premalo uvlačenja dovodi do troškova vezanih za pozive funkcija, a previše uveća veličinu koda i dovodi do neefikasnosti, pa treba uspostaviti ravnotežu.

2.2 Optimizacije koda

Optimizovani međukod se u fazi generisanja koda prevodi u asemblerski tj. mašinski kod. U finalnom generisanju koda, bitno je donošenje pametnih odluka kako bi generisani ciljni kod bio što efikasniji. U ovom slučaju bitne su specifične karakteristike mašine za proizvodnju optimizovanog koda za tu konkretnu arhitekturu. Ralikujemo sledeće tehnike optimizacija koda:

- **Optimizacija redosleda instrukcija**
Razlikujemo fazu odabira instrukcija, fazu alokacije registara i fazu raspoređivanja instrukcija.
- **Optimizacija upotrebom keša**
Upotreba keša se zasniva na prostornoj i vremenskoj lokalnosti, a cilj je da postane što bolja. Ako je nekoj memoriji skoro pristupano, vremenska podrazumeva da će biti uskoro opet, a prostorna da će i njeni susedni objekti biti uskoro korišćeni.

Postoje razne optimizacije koje se mogu sprovesti i na nivou izgenerisanog ciljnog koda.

3 Prvi naslov

Ovde pišem tekst.

3.1 Prvi podnaslov

Ovde pišem tekst.

3.2 Drugi podnaslov

Ovde pišem tekst.

3.3 ... podnaslov

Ovde pišem tekst.

4 n-ti naslov

Ovde pišem tekst.

4.1 ... podnaslov

Ovde pišem tekst.

5 Zaključak

Ovde pišem zaključak.

Literatura

A Dodatak

Ovde pišem dodatne stvari, ukoliko za time ima potrebe.