Optimizacije kroz GCC, LLVM i Native Image

Tamara Stojković, Emilija Stošić, Teodora Isailović

Metodologija stručnog i naučnog rada
Matematički fakultet
Univerzitet u Beogradu

Beograd, decembar 2022.

Sadržaj

Uvod

- Cilj optimizacije - poboljšanje performansi koda
- Osnovna podela: Optimizacije međukoda i Optimizacije koda
- Optimizacije međukoda: lokalne, globalne i međuproceduralne
- Lokalne optimizacije služe za ubrzavanje malih delova neke funkcije
- Globalne optimizacije primenjuju se na jednu po jednu funkciju
- Međuproceduralne optimizacije :
  - Vrše se na nivou celog programa tj. više funkcija
  - Najpoznatija tehnika - uvlačenje definicija funkcija

Optimizacije međukoda

- Tehnike lokalne optimizacije koje razlikujemo:
  - Eliminacija čestih podizraza
  - Slaganje konstanti
  - Propagacija kopija
  - Smanjenje snage operatora
  - Eliminacija mrtvog koda
  - Algebarsko pojednostavljenje i reasocijacija
  - Kompozicija lokalnih transformacija
- Globalne optimizacije koje razlikujemo:
  - tehnike lokalne koje se primenjuju globalno (npr. globalna eliminacija mrtvog koda)
  - Optimizacija kretanje koda (pomeranje invarijantog koda, parcijalna eliminacija suvišnosti)

Optimizacije koda

- Prerada asemblerskog koda, kako bi se dobio što efikasniji kod
- Zahteva se detaljno poznavanje ciljne arhitekture
- Razlikujemo sledeće tehnike:
  - Optimizacija redosleda instrukcija - obuhvata fazu odabira instrukcija, alokacije registara i raspoređivanja instrukcija
  - Optimizacija upotrebom keša - zasniva se na prostornoj i vremenskoj lokalnosti, cilj da bude što bolja

Optimizacije u okviru kompajlera GCC

- Opcije koje su vrlo značajne u procesu kompilacije, su opcije za optimizaciju
- Nivo optimizacije koju kompajler vrši se kontroliše opcijom -On, gde je n nivo zahtevane optimizacije
- Postoji sedam nivoa optimizacije : -O0, -O1, -O2, -O3, -Os, -Og, -Ofast

```
emilija@SONY:~/Desktop$ gcc -Wall -O0 primer.c -lm
emilija@SONY:~/Desktop$ time ./a.out
sum = 4e+38

real    0m1.707s
user    0m1.701s
sys     0m0.000s
emilija@SONY:~/Desktop$ gcc -Wall -O1 primer.c -lm
emilija@SONY:~/Desktop$ time ./a.out
sum = 4e+38

real    0m0.419s
user    0m0.412s
sys     0m0.000s
emilija@SONY:~/Desktop$ gcc -Wall -O2 primer.c -lm
emilija@SONY:~/Desktop$ time ./a.out
sum = 4e+38

real    0m0.381s
user    0m0.360s
sys     0m0.009s
emilija@SONY:~/Desktop$ gcc -Wall -O3 primer.c -lm
emilija@SONY:~/Desktop$ time ./a.out
```

Optimizacije u okviru kompajlera LLVM

- Optimizacije u vidu prolaza
- Svi LLVM prolazi su podklase klase Pass
- Funkcionalnosti su implementirane tako što prevazilaze virtuelne metode nasleđene od klase Pass
- Analysis Passes prikuplja informacije koje služe za otklanjanje grešaka ili vizuelizaciju programa
- Transform Passes se koriste za optimizaciju koda
- Utility Passes služe za dobijanje raznih informacija, koje su bitne za razvoj drugih prolaza

Optimizacije u okviru kompajlera LLVM

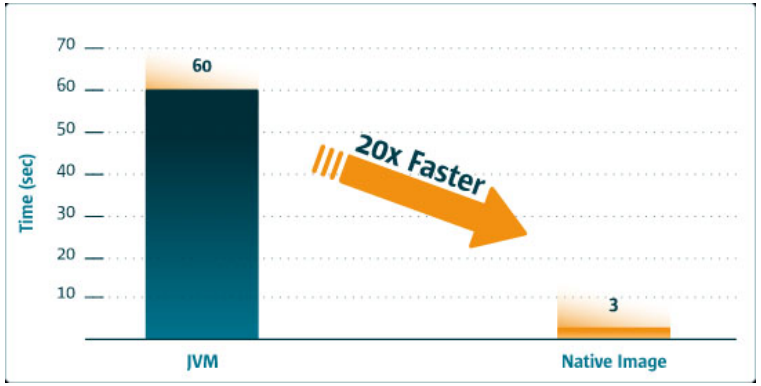
- Postoji sedam nivoa optimizacije : -O0, -O1, -O2, -O3, -O4, -Os, -Oz

```
emilija@SONY:~/Desktop$ clang -Wall -O0 main.c -lm
emilija@SONY:~/Desktop$ time ./a.out
sum = 4e+38

real    0m1.743s
user    0m1.703s
sys     0m0.008s
emilija@SONY:~/Desktop$ clang -Wall -O1 main.c -lm
emilija@SONY:~/Desktop$ time ./a.out
sum = 4e+38

real    0m0.771s
user    0m0.765s
sys     0m0.004s
emilija@SONY:~/Desktop$ clang -Wall -O2 main.c -lm
emilija@SONY:~/Desktop$ time ./a.out
sum = 4e+38

real    0m0.370s
user    0m0.369s
sys     0m0.000s
emilija@SONY:~/Desktop$ clang -Wall -O3 main.c -lm
emilija@SONY:~/Desktop$ time ./a.out
sum = 4e+38

real    0m0.364s
user    0m0.361s
sys     0m0.000s
```

Slika: Rezultati optimizacije kod LLVM kompajlera

Native Image - pojam

- Od bytecode-a do izvršivog koda (native image) određene platforme
- Ahead-Of-Time kompilacija
- Rezultujući kod se izvršava brže i zahteva manje memorije u odnosu na VM
- Najpoznatiji primeri Native Image iz GraalVM i Ngen (CrossGen) za .NET Framework (Core)



Slika: Ubrzanje native image-a u odnosu na JVM

Prevođenje

- Statička analiza (Closed World Assumption)
- Problemi i optimizacije:
  - Reflection API
  - Dynamic proxy
  - Java Native Interface
- Za nerazrešene delove potreban konfiguracioni fajl (tracing agent)
- Fallback image

Zaključak

- Mnogi optimizacioni problemi su NP-teški pa koriste aproksimacije i heuristike
- Optimizacija ne sme da menja ispravnost i ponašanje koda
- Optimizacije su ključno mesto za postizanje dobrih performansi
- Uvek aktuelna tema zbog večite težnje za što boljim performansama

Literatura

- Kompajleri Stanford

  `https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/`
- The Dragon Book Compilers: Principles, Techniques (Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman)
- Options That Control Optimization

  `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`
- LLVM's Analysis and Transform Passes

  `https://llvm.org/docs/Passes.html`
- GraalVM Manual - Native Image

  `https://www.graalvm.org/22.0/reference-manual/native-image/`