Profajliranje Haskell programa

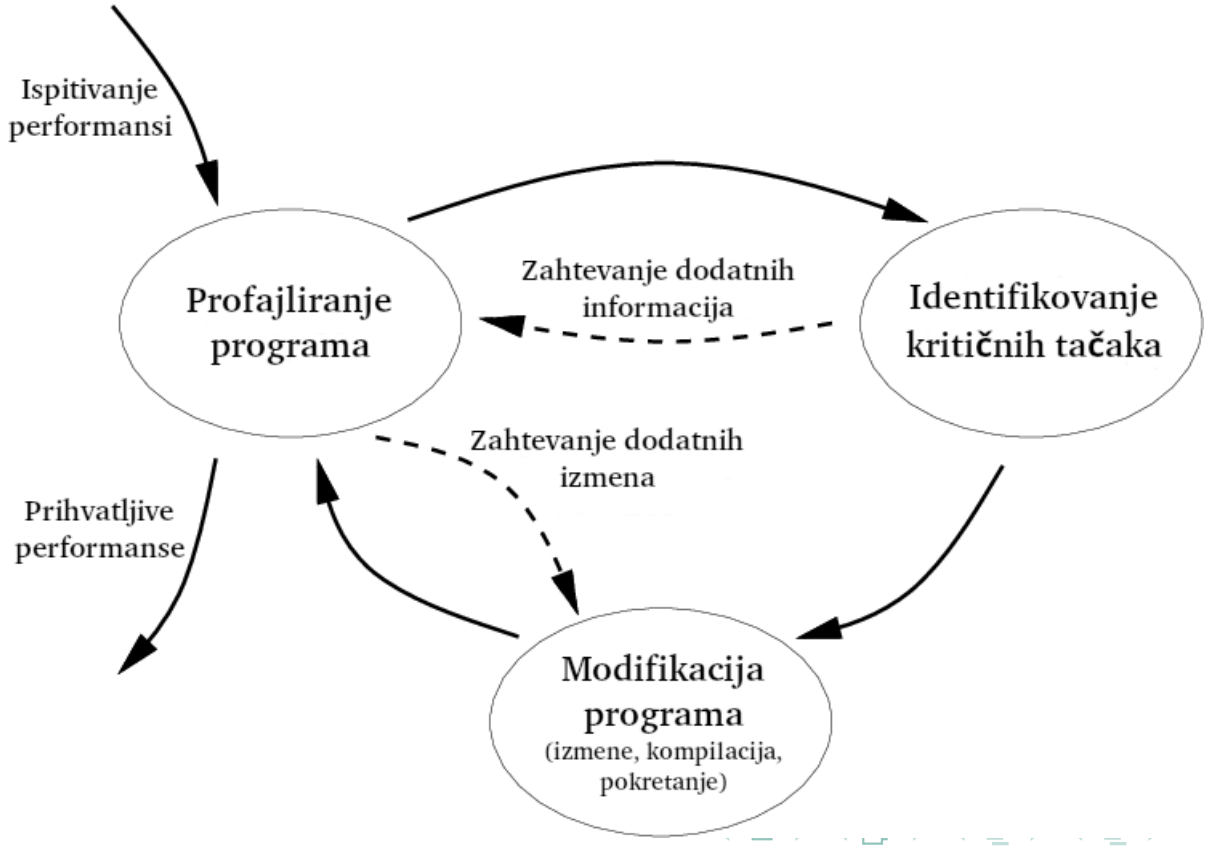
Jovana Bošković, Ana Jakovljević
Nikola Perić, Mateja Trtica

Matematički fakultet
Univerzitet u Beogradu

22. april 2020.

Uvod

- Deo svakog razvoja softvera
- Metoda koja upoznaje programera sa kodom
- Olakšava proces pisanja i održavanja, povecava produktivnosti



Profajliranje

- Alat - Profajler
- Pronalazi kritične tačke izvršavanja koda, prati uticaj određene izmene na kod
- Neke od mogućih upotreba:
  1. Projektanti hardvera - provera programa na različitim arhitekturama
  2. Programeri - provera programa da li zadovoljavaju potrebne perfomanse
  3. Dizajniranje kompajlera
  4. itd..

Metode profajliranja

- Postoji više metoda profajliranja koje nam daju različite profile
- Profajliranje vremena:
  1. Uzimanje uzoraka
  2. Brojač frekvencija
  3. Prolazno vreme procedure
- Profajliranje prostora:
  1. Profili alokacije
  2. Profili curenja
  3. Hip profili

O Haskell-u i profajliranju

- Haskell - čist funkcionalni programski jezik visokog nivoa
- Koncepti:
  1. Lenjo izračunavanje
  2. Funkcije višeg reda
  3. Polimorfizam
- Informacije koje profajler pruža moraju da odgovaraju realnom izvršavanju:
  1. Poredak izvršavanja ostaje isti
  2. Bez narušavanja lenje semantike
  3. Optimizacije kompajlera ne treba isključivati
  4. Zanemarljiva cena profajliranja

Uticaj koncepata na profajliranje

- Ponovno korišćenje funkcija
  - Koncepti funkcija višeg reda i polimorfizam
  - Postojanje opšte funkcije višeg reda koja može biti specijalizovana nekom drugom funkcijom. Informacije o troškovima takve funkcije moraju biti odvojeni po mestima poziva
- Lenjo izračunavanje
  - Izraz se izračunava kada njegov rezultat postane potreban
  - Delovi koda se ne izvršavaju
  - Ispoljava se problem povezivanja dinamički prikupljenih informacija sa mestima u izvornom kodu
  - Pitanje pridruživanja troška izračunavanja

Transformacija i optimizacija

- Od apstraktnog koda do izvršnog nivoa preko niza transformacija
- Optimizacione transformacije na međureprezentaciji Core jezika
- Kroz optimizacije i transformacije troškovi ostaju povezani sa odgovarajućim izvorom

Proces profajliranja

- GHC (*The Glasgow Haskell Compiler*) - kompajler i interaktivno okruženje za programski jezik Haskell
- Proces profajliranja pomoću GHC se sastoji iz tri koraka:
  1. Rekompajlirati program uz opciju za profaliranje
  2. Pokrenuti program da bi se generisao profil
  3. Ispitati generisan profil, koristeći dobijene informacije razmotriti proces optimizacije i po potrebi ponoviti proces

Centar troškova

- **Centar troškova** (eng. *cost center*) - struktura deklarisana označenim delom koda čije su informacije o izvršavanju od interesa
- Trošak - vremenska ili memorijska zahtevnost izraza
- Centar troškova se povezuje na izraz od interesa
- GHC pamti stek centara troškova za svaki izraz u toku izvršavanja i generiše stablo poziva (eng. call-tree) sa pripisanim troškovima

Dodela centara troškova

- Centara torškova može biti automatski generisan ili ručno unesen
- Opcija -fprof-auto za automatsko generisanje oznaka
- Ručno dodavanje centra troškova se reguliše SCC (Set Cost Center) anotacijom
- Sintaksa ručnog označavanja centra troškova:
  - {-# *SCC "name" #-}* *expression*
  - {-# *SCC func #-}*
  - {-# *SCC func "name" #-}*
- Semantika troškova - koje cene se povezuju na koji centar troškova
- lenji izraz (eng. *thunk*) - izraz koji se izračunava po potrebi

Priprema za profajliranje

- Vrste profajliranja:
  - Vremensko i alokacijsko profajliranje
  - Prostorno profajliranje
- Program se priprema za profajliranje navođenjem opcije *-prof* pri prevođenju, čime je omogućeno osnovno vremensko i prostorno profajliranje
- Dodatne opcije: *-fprof-auto*, *-fprof-auto-top*, *-fprof-auto-exported*, *-fprof-cafs* ...
- Opcije se nadove između +RTS i -RTS zastavica, pa se nazivaju i RTS opcijama

Vremensko i alokacijsko profajliranje


```
    Tue Mar 31 23:52 2020 Time and Allocation Profiling Report  (Final)

        Main +RTS -p -RTS

    total time  =        0.06 secs   (56 ticks @ 1000 us, 1 processor)
    total alloc =  56,947,248 bytes  (excludes profiling overheads)
```

```
COST CENTRE        MODULE    SRC                    %time %alloc

process.word_occs  Main      Main.hs:(30,9)-(31,46)   67.9   33.4
process.ws         Main      Main.hs:(25,9)-(27,20)   26.8   57.5
process.ws.\       Main      Main.hs:26:27-60          3.6    0.0
main               Main      Main.hs:(10,1)-(19,27)    1.8    7.8
```

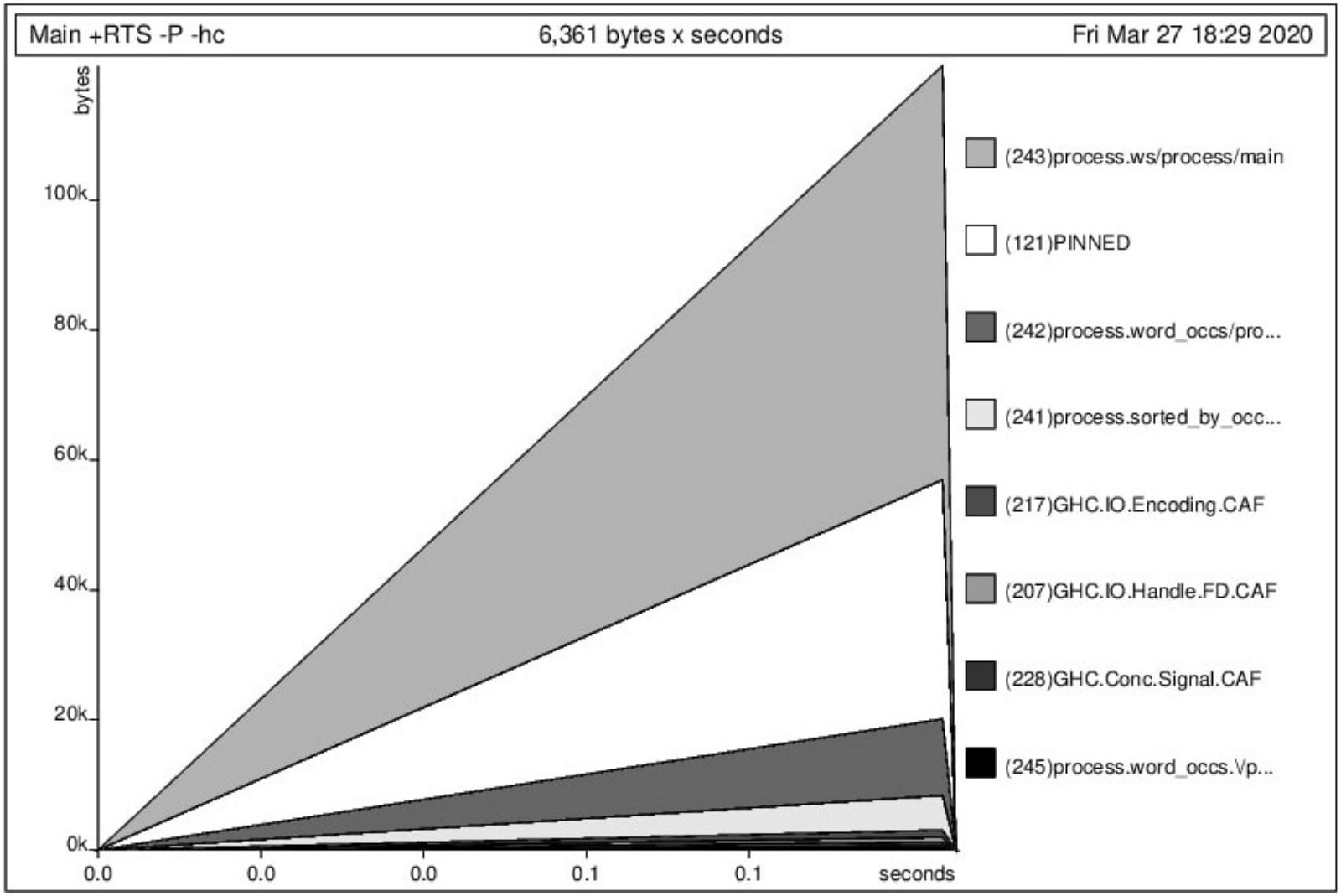
```
                                                                 individual     inherited
COST CENTRE                MODULE                SRC                 no.  entries  %time %alloc  %time %alloc

MAIN                       MAIN                  <built-in>          119        0    0.0    0.0  100.0  100.0
 CAF                       Main                  <entire-module>     237        0    0.0    0.0    0.0    0.0
  main                     Main                  Main.hs:(10,1)-(19,27) 238     1    0.0    0.0    0.0    0.0
 CAF                       GHC.Conc.Signal       <entire-module>     228        0    0.0    0.0    0.0    0.0
 CAF                       GHC.IO.Encoding       <entire-module>     217        0    0.0    0.0    0.0    0.0
 CAF                       GHC.IO.Encoding.Iconv <entire-module>     215        0    0.0    0.0    0.0    0.0
 CAF                       GHC.IO.Exception      <entire-module>     209        0    0.0    0.0    0.0    0.0
 CAF                       GHC.IO.Handle.FD      <entire-module>     207        0    0.0    0.1    0.0    0.1
 CAF                       GHC.IO.Handle.Internals <entire-module>   206        0    0.0    0.0    0.0    0.0
 CAF                       System.Posix.Internals <entire-module>    179        0    0.0    0.0    0.0    0.0
 CAF                       GHC.IO.FD             <entire-module>     146        0    0.0    0.0    0.0    0.0
 CAF                       GHC.IO.Handle.Text    <entire-module>     145        0    0.0    0.0    0.0    0.0
 main                      Main                  Main.hs:(10,1)-(19,27) 239     0    1.8    7.8  100.0   99.9
  process                  Main                  Main.hs:(23,1)-(37,42) 240     1    0.0    0.9   98.2   92.1
   process.sorted_by_occs  Main                  Main.hs:34:9-63     241        1    0.0    0.3    0.0    0.3
   process.word_occs       Main                  Main.hs:(30,9)-(31,46) 242     1   67.9   33.4   67.9   33.4
    process.word_occs.\    Main                  Main.hs:30:32-54    245      301    0.0    0.0    0.0    0.0
   process.ws              Main                  Main.hs:(25,9)-(27,20) 243     1   26.8   57.5   30.4   57.5
    process.ws.\           Main                  Main.hs:26:27-60    244   103834    3.6    0.0    3.6    0.0
```

Prostorno profajliranje

- Curenje memorije - zadržavanje memorije koja se ne upotrebljava
- Posledica je intenzivna aktivnost sakupljača podataka koji iziskuje dodatne resurse
- Generisanje memorijskog profila:
  1. prevođenje programa uz odgovarajuće opcije
  2. pokretanje programa sa nekom od RTS opcija za profajliranje hipa
  3. pokretanje alata hp2ps nad dobijenim izlazom
  4. prikaz hip profila pomoću Postscript pregledača

Prostorno profajliranje



Literatura

[1] CORPORATE Adobe Systems Inc. *PostScript Language Reference (3rd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., USA, 1999.

[2] Richard Bird and Philip Wadler. *An Introduction to Functional Programming*. Prentice Hall International (UK) Ltd., GBR, 1988.

[3] John Goerzen Bryan O'Sullivan, Don Stewart. *Real World Haskell*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2008.

[4] K. Hammond, D.N. Turner, and P.M. Sansom. *Functional Programming, Glasgow 1994: Proceedings of the 1994 Glasgow Workshop on Functional Programming, Ayr, Scotland, 12–14 September 1994*. Workshops in Computing. Springer London, 2013.

[5] David J. King, Jon G. Hall, and Philip W. Trinder. A strategic profiler for glasgow parallel haskell. pages 88–102, 1998.

[6] Dušan Okanović. *Model adaptivnog sistema za praćenje i predikciju rada distribuiranih aplikacija*. PhD thesis, Univerzitet u Novom Sadu, Fakultet tehničkih nauka u Novom Sadu, September 2012.

[7] Simon Peyton Jones and Andre Santos. A transformation-based optimiser for haskell. *Science of Computer Programming*, 32(1):3–47, October 1997.

[8] Patrick M. Sansom. *Execution Profiling for Non-strict Functional Languages*. PhD thesis, University of Glasgow, April.

[9] Artifex Software. Ghostview, 2019. on-line at: `https://gsview.com/`.

[10] The Glasgow Haskell Compiler Team. Glasgow Haskell Compiler User's Guide, 2015. on-line at: `https://downloads.haskell.org/ghc/latest/docs/html/users_guide/`.