UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Levstik

Preverjanje in merjenje hitrosti testne izvedbe prevajalnika za MRuby

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE

STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Boštjan Slivnik

Ljubljana, 2014

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00503 / 2013
Datum: 12.4.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **PETER LEVSTIK**

Naslov: **PREVERJANJE IN MERJENJE HITROSTI TESTNE IZVEDBE PREVAJALNIKA ZA MRUBY**
TESTING AND BENCHMARKING THE PROTOTYPE MRUBY COMPILER

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Programski jezik MRuby je poenostavitev programskega jezika Ruby, ki je namenjena za vgrajene sisteme in kot skriptni jezik v večjih aplikacijah. Trenutno je še v fazi razvoja, kljub temu pa poleg uradnega interpreterja že obstaja tudi prevajalnik. Preverite kakovost delovanja prevajalnika za programski jezik MRuby in izmerite, kako hitro se izvajajo programi prevedeni s tem prevajalnikom v primerjavi z ekvivalentnimi programi napisanimi v primerljivih programskih jezikih. Pri meritvah hitrosti poskusite uporabiti že uveljavljene testne primere.

Mentor: B. Slivnik
doc. dr. Boštjan Slivnik

Fakulteta za računalništvo in informatiko 2 UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Peter Levstik, z vpisno številko **63060484**, sem avtor diplomskega dela z naslovom:

Preverjanje in merjenje hitrosti testne izvedbe prevajalnika za MRuby

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom **doc. dr. Boštjana Slivnika**,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne: _____ Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Boštjanu Slivniku za pomoč, usmerjanje in nasvete pri izdelavi diplomskega dela in Kateřini za vso spodbudo.

VSEBINA

POVZETEK

Programski jezik MRuby je poenostavitev programskega jezika Ruby, ki je namenjena za vgrajene sisteme in vgradnjo v večje aplikacije. Trenutno je še v fazi razvoja, kljub temu pa poleg uradnega »interpreterja« že obstaja tudi prevajalnik. Preverite kakovost delovanja prevajalnika za programski jezik MRuby in izmerite, kako hitro se izvajajo programi prevedeni s tem prevajalnikom v primerjavi z ekvivalentnimi programi napisanimi v primerljivih programskih jezikih. Pri meritvah hitrosti poskusite uporabiti že uveljavljene testne primere.

ABSTRACT

The MRuby programming language is a simplified version of Ruby. It is meant to be used in embedded systems and as a sub module in larger applications. At the time being it is still being developed. Nevertheless, apart from the official interpreter a prototype compiler exists as well. Estimate the quality of the prototype MRuby compiler and compare the speed of programs compiled with it against the equivalent programs written in similar programming languages. Use some well-known benchmark programs if at all possible.

UPORABLJENE KRATICE IN SIMBOLI

- C, C++, Ruby, MRuby, Python - programski in skriptni jeziki
- GIT - orodje za nadzor revizij in upravljanja izvorne kode
- GEM - Ruby-jev upravljalec vtičnikov
- CPE - centralna procesorska enota
- PBM - prenosni format bitne slike
- RAKE - Ruby-jev ukaz za sestavljanje (ang. *make*)
- STDIN, STDOUT - standardni vhod / izhod
- DNK, RNK - deoksi- in ribonukleinska kislina
- FASTA - eden izmed algoritmov za iskanje podobnosti v molekulah DNK, RNK ali beljakovinah
- GCC - skupina orodij za prevajanje (ang. *GNU Compiler Collection*)

1. UVOD

V današnjem času je na voljo kar precej programskih jezikov in prevajalnikov. Njihovo število pa iz dneva v dan še narašča. Vsak izmed njih ima svoj pristop k reševanju problema in tako pokriva svoje področje. Starejši prevajalniki praviloma delujejo hitreje, medtem ko so novejši počasnejši, vendar jih je lažje pisati.

Jezik MRuby je relativno nov, zato si poglejmo njegove prednosti, pomanjkljivosti, primerjajmo hitrost izvajanja testnih primerov z ostalimi jeziki in ga navsezadnje tudi uvrstimo mednje.

V pogon bomo spravili trinajst programov pod jeziki C, C++, Python, Ruby in MRuby ter rezultate primerjali. Programi so sicer napisani v različnih programskih jezikih, vendar pri obdelavi podatkov uporabljajo enak algoritem.

Zanima nas ali ima MRuby sploh primerljive hitrosti izvajanja in/ali je morda počasnejši oz. hitrejši od svojih tekmecev oz. kako se ponaša program preveden s prevajalnikom MRuby_cc v primerjavi z njim. Ob tem nas zanima tudi kje nastopijo težave, saj sta, kot že vemo, sam MRuby in zanj prevajalnik relativno mlada.

2. RUBY IN MRUBY

2.1 VRSTE PROGRAMSKIH JEZIKOV

Programski jeziki se v splošnem delijo v pet skupin:

- **Proceduralni oz. postopkovni** jeziki izvršujejo zaporedje stavkov oz. izražajo postopek, ki ga je potrebno upoštevati, da dobimo željeni rezultat. Tipično uporabljajo veliko spremenljivk in zank. Funkcije tu ne vračajo le rezultata, ampak imajo tudi drugačno vlogo, npr. spreminjajo vrednost spremenljivkam izven obsega funkcije oz. izpisujejo podatke.
- **Funkcionalni jeziki** v nasprotju s postopkovnimi, težijo k manjšem obsegu programske kode. V izogib zankam se uporabljajo rekurzivne metode. Primarna naloga funkcij je vračanje vrednosti. Odsvetujejo se izpisi ali hramba podatkov izven obsega funkcije. Tipično so sintaktično enostavnejši, vendar njihov programski model lahko povzroči preglavice pri programiranju sistema.
- **Objektno usmerjeni jeziki** gledajo na vse kot na množico objektov, v kateri ima vsak svoje lastnosti in funkcije. Ideja je razbiti problem na več objektov, kateri ponujajo storitve s katerimi si pomagamo rešiti problem. Vsaka stvar je objekt in jo lahko vključimo znotraj drugega objekta, zanje značilne funkcije so enkapsulacija, dedovanje in polimorfizem.
- **Skriptni jeziki** so navadno proceduralni in lahko vsebujejo elemente objektno usmerjenih jezikov, vendar jih uvrščamo v svojo skupino, saj niso namenjeni programiranju sistema v velikem obsegu. Velikokrat so preprostejši v sintaksi, zahtevajo manj predznanja, vendar z njimi lahko kaj hitro povzročimo veliko zmedo.
- **Logični jeziki** omogočajo programerju vnašanje deklarativnih izjav, nato pa sam računalnik razmisli o posledicah teh izjav. Logično programiranje s tega stališča nima nič skupnega z ostalimi jeziki, saj računalniku navajamo le kaj naj bi se zgodilo, ne pa kako naj se to zgodi.

2.2 KRATEK OPIS PROGRAMSKIH JEZIKOV

Pri merjenju hitrosti bomo programe pognali pod naslednjimi jeziki: C, C++, Python in Ruby. Naj vsakemu izmed njih namenimo nekaj besed.

- **C** je splošno namenski programski jezik, zelo popularen posebej v igralskih vodah. Od C++ ga loči npr. dejstvo, da ni objektno usmerjen, a je zaradi tega v nekaterih primerih hitrejši, velikost prevedenih datotek pa manjša.
- **C++** je zelo primeren za velike projekte, saj je objektno usmerjen in skupaj s knjižnicami, ki ga dopolnjujejo, pokriva večino področij programskega razvoja. Primeren je tudi za delo v skupini, saj se ga lahko razdeli na več delov, na katerih je možno delati v skupini oz. individualno, a sočasno. Dele kode lahko večkrat uporabimo, zahvaljujoč njegovi strukturi. Potrebno je omeniti tudi, da je zelo hiter in efektiven jezik.
- **Python** je izredno zmogljiv, modularen, hiter in dinamičen skriptni jezik. Ima čisto in berljivo sintakso. Lahko ga integriramo z nekaterimi objektnimi modeli kot npr.: COM, .NET in CORBA. Ima tudi zelo bogato zbirko vtičnikov.
- **Ruby** je prav tako dinamičen, odprtokodni skriptni jezik s poudarkom na enostavnosti in produktivnosti. Poleg bogate zbirke modulov ponuja tudi interaktivno spletno lupino, ki živi v brskalniku.

2.3 PROGRAMSKI JEZIK RUBY

Yukihiro "Matz" Matsumoto je leta 1995 javno objavil prvo verzijo jezik Ruby. Ustvaril ga je iz različnih delov svojih najljubših jezikov (Perl, Smalltalk, Eiffel, ADA, in Lisp) z željo, da bi jezik naredil čimbolj naraven – ne enostaven. K temu dodaja še:

»Ruby je na videz preprost, vendar zelo zapleten znotraj, nekako tako, kot človeško telo.«

Ruby je konglomerat veliko metodologij, kar ga naredi težkega za učenje in efektivno programiranje. Se pa lahko brez težav primerja z ostalimi skriptnimi programskimi jeziki. Njegovi oboževalci mu pravijo tudi »čudoviti, pretkani jezik«. Ima veliko skupnega v primerjavi z npr. Pythonom. Oba sta visokonivojska programska jezika, imata smetarja (ang. *garbage collector*), sta dinamično pisana, imata interaktivno lupino in bogate standardne knjižnice. Oba lahko uporabimo za reševanje enakih programskih problemov.

Ruby podpira bloke kode (ang. *blocks*) obdane z zavitimi oklepaji, spremenljivke objekta poizveduje prek spremenljivke *attr_accessor*, medtem ko Python uporablja klice metod (). Pythonova sintaksa je izredno preprosta. Modeliran je po Fortranovih načelih, medtem ko Ruby

izhaja iz Lispa. Drži se ga filozofija »obstaja samo eden način, kako rešiti zadevo« in ga dela hitrejšega ter enostavnejšega za učenje, nasprotno kot Ruby, ki je poln zvijač. Na spletu lahko najdemo tudi spletno različico interaktivne lupine, kjer lahko v realnem času izvajamo delčke kode. Dosegljiva je na naslovu: http://tryruby.org/

Primer sorodnosti s Pythonom

Želimo pognati del kode, za tem oz. pred tem pa izvesti nekatera dodatna opravila, kot npr. samodejno zapreti datoteko po pisanju vanjo. Pri Ruby-ju lahko uporabimo »blocks«. Pri Python-u pa stavek *with*.

Python:

```
with open('somefile.txt') as f:
print(f.readline())
```

Ruby:

```
File.open('somefile.txt') do |f
puts f.readline
end
```

IZPISA 1.1 IN 1.2 - VZORCA KODE, PRIMER SORODNOSTI

Primer raznolikosti s Pythonom: klic metode/funkcije

Razlog, zakaj Python uporablja oklepaje za klice metod/funkcij je enostaven. Želimo vedeti in prav tako videti, kdaj se kliče funkcija in kdaj se le sklicujemo oz. dostopamo do spremenljivke. Pri Ruby-ju na to lahko pozabimo, saj imamo **vedno** klic metode.

Python:

```
class Test1:
  tekst = "Lep pozdrav!"

  def pozdravi(self):
    print self.tekst


Test1().pozdravi()
```

Oba izpišeta: »Lep pozdrav!«

Ruby:

```
class Test1
      attr_accessor :tekst

      def pozdravi
          puts "#{@tekst}\n"
      end
  end

  # kreiranje instance
  b = Test1.new
  b.tekst = "Lep pozdrav!"

  # izpis "Lep pozdrav!"
  b.pozdravi
```

IZPISA 2.1 IN 2.2 - VZORCA KODE, PRIMER RAZNOLIKOSTI

2.4 PROGRAMSKI JEZIK MRUBY

MRuby je relativno mlada, odprtokodna, modularna in minimalistična različica jezika Ruby, ki ga je začel razvijati avtor Ruby-ja sam. V času pisanja tega besedila najdemo na spletu komaj kaj dokumentacije, le-ta pa zajema bolj osnove. Deluje kot tolmač (ang. *interpreter*) za Ruby. V pričakovanju, da bo zaživel znotraj drugih programskih aplikacij, skuša ostati lahek in preprost za vgradnjo. Glavni namen MRuby-ja je zagotoviti različico Ruby-ja, katero je moč vgraditi, prevesti in spojiti z drugimi aplikacijami. V taki obliki naj bi imel tudi manjši obseg. MRuby cilja na razvijalce računalniških iger in vgrajenih aplikacij.

Razvojni navdušenec in tehnični arhitekt iz podjetja Appirio, Hiroshi Nakamura, ga je opisal na prav poseben način:

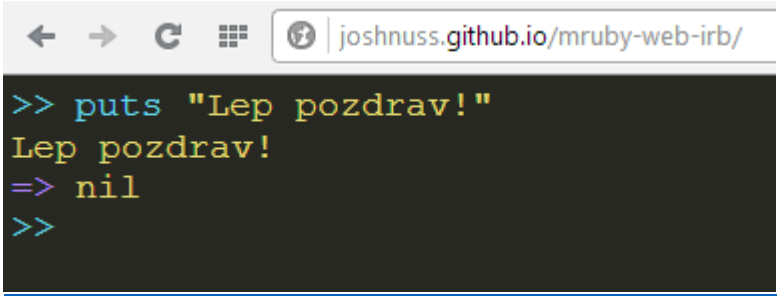
mruby ::= RiteVM + parser + code generator + mruby Ruby lib + mruby C lib + binary I/O

Avtor Yukihiro "Matz" Matsumoto, pa o njem pravi takole. Citiram:

»Sintaksa je na višjem nivoju. Alternative imajo težave in požrejo preveč resursov. Niso primerne za umeščanje v vgrajene sisteme.«

Ruby-jev virtualni stroj, ki ga uporablja MRuby se imenuje RiteVM in je posebej namenjen integriranim sistemom. Tudi MRuby že podpira spletno interaktivno lupino, dosegljivo na:

http://joshnuss.github.io/mruby-web-irb/



SLIKA 1 - PRIKAZ IZGLEDA SPLETNE INTERAKTIVNE LUPINE ZA MRUBY [1]

Glede na to, da MRuby sintaktično posnema Ruby imata je težko najti kako razliko. Kot smo že omenili, nekaj področij kode ostaja odprtih in ta koda se pod MRuby-jem ne prevede. Ima pa eno zelo lepo lastnost in sicer MRuby kodo se lahko prevede tudi s pomočjo C prevajalnika – GCC in uporabo MRuby knjižnice kot reference. Program, ki ga prevajamo, mora sedaj

sintaktično odgovarjati pravilom jezika C, umeščena MRuby koda pa še pravilom jezika MRuby.

Poglejmo si primer. Sledečo kodo zapišimo v datoteko *pozdrav.c*:

```
#include <stdlib.h>
#include <stdio.h>

/* Vključimo MRuby glavo */
#include <mruby.h>
#include <mruby/compile.h>

int main(void)
{
  mrb_state *mrb = mrb_open();
  char code[] = "p 'Lep pozdrav iz MRuby-ja!'";
  printf("Lep pozdrav iz C-ja!\n");

  mrb_load_string(mrb, code);
  return 0;
}
```

IZPIS 3 - VZOREC IZVORNE KODE MRUBY

Zgornjo datoteko prevedemo s komando:

```
gcc -Iinclude pozdrav.c lib/libmruby.a -lm -o pozdrav.out
```

IZPIS 4 - PRIMER PREVAJANJA Z GCC

Kot rezultat dobimo objektno datoteko, ki jo poženemo z ukazom: *./pozdrav.out*

Nato prejmemo izpis:

»Lep pozdrav iz C-ja!«
»Lep pozdrav iz MRuby-ja!«

2.5 NEPOKRITI DELI SINTAKSE JEZIKA MRUBY

MRuby zaenkrat še ne pokriva celotne zmogljivosti Ruby-ja, ampak le osnove. Skozi delo z njim se soočimo z manjkajočimi oz. še ne zasnovanimi področji, ki so pri njegovem starejšem bratu že od davno v polnem zagonu. Tako npr. kaj hitro opazimo, da MRuby ne zna uporabljati programske besede *require*, ki je potrebna, da lahko v programski jezik vključimo knjižnice. Ta funkcionalnost je sicer delno že omogočena preko GEM paketa *mruby-require*, ki ga lahko namestimo preko paketnega upravljalca GEM za MRuby oz. snamemo iz GIT-a na naslovu:

https://github.com/mattn/mruby-require

Vendar je potrebno omeniti, da je GEM paket še v povojih in ne deluje ravno po pričakovanjih. Kasneje bomo spoznali tudi postopek namestitve teh paketov.

Naslednja manjkajoča funkcionalnost je funkcija *eval*, namreč zelo uporabna, saj omogoča zaganjanje niza znakov oz. bloka kode, kot bi bil izraz. Z drugimi besedami izvede eno ali več vrstic kode, kot da bi bili vključeni v program namesto same vrstice, ki vključuje besedo *eval*.

Manjka tudi aritmetika z arbitrarno natančnostjo oz. podatkovni tip *bignum*, ki je izredno uporaben pri računanju z dolgimi celimi števili in tako ohranjamo natančnost vse do enic. Lahko bi navajali dalje, naj le na kratko omenimo še manjkajoče regularne izraze, vhodno-izhodne storitve, datotečne storitve, niti in mehanizme za zaklepanje delov kode, razred Process, Encoding in Marshal itd.

3. PREVAJALNIK ZA MRUBY

3.1 KOMPILIRANJE MRUBY IZ IZVORNE KODE

MRuby pri prevajanju kode in navzkrižnem prevajanju knjižnic uporablja RAKE, preprost Ruby-jev program za prevajanje v objektne datoteke.

Predpogoji

Za prevajanje izvorne kode v mruby potrebujemo sledeča orodja:

- Prevajalnik C
- Povezovalnik (ang. *linker*)
- Orodje za arhiviranje
- Generator razčlenjevalnika (ang. *parser generator*)
- Ruby (verzija 1.8, 1.9, …. 2.0)

Uporaba

Znotraj korenske mape izvorne kode mruby se nahaja datoteka *build_config.rb*, ki vsebuje nastavitve za prevajanje v kodo MRuby. Izgleda takole:

```
MRuby::Build.new do |conf|
  toolchain :gcc
end
```

IZPIS 5 - STRUKTURA DATOTEKE BUILD_CONFIG.RB

Datoteko lahko po potrebi spremenimo in dodamo v proces vsa ostala orodja ali pripomočke potrebne za prevajanje kode. Dodamo lahko nastavitve prevajalnika C in GCC, povezovalnika, nastavitve orodja Visual Studio, nastavitve za Android. Lahko izbiramo tudi posamična orodja, ki bodo prevedena med postopkom itd.

Začetek prevajanja povzročimo s komando *./minirake* v korenskem imeniku. Za generiranje in izvajanje testnih primerov poženemo *./minirake test*. Za čiščenje izhodne mape poženemo *./minirake clean*.

Postopek prevajanja programa MRuby se odvija takole:

1. Prevajanje vseh datotek v mapi *src*
2. Izdelava gramatike razčlenjevalnika s pomočjo datoteke *src/parse.y*
3. Prevajanje datoteke *build/host/src/y.tab.c* in nato gradnja *build/host/src/y.tab.o*
4. Izdelava datoteke *build/host/lib/libmruby_core.a* iz vseh objektnih datotek (samo C)
5. Izdelava datoteke *build/host/bin/mrbc* s pomočjo prevajanja *tools/mrbc/mrbc.c* in povezovanja z *build/host/lib/libmruby_core.a*
6. Izdelava *build/host/mrblib/mrblib.c* s prevajanjem vseh .rb datotek pod **mrblib* z *build/host/bin/mrbc*
7. Prevajanje *build/host/mrblib/mrblib.c* v *build/host/mrblib/mrblib.o*
8. Izdelava *build/host/lib/libmruby.a* iz vseh objektnih datotek (C in Ruby)
9. Izdelava *build/host/bin/mruby* s prevajanjem *mrbgems/mruby-bin-mruby/tools/mruby/mruby.c* in povezovanjem z *build/host/lib/libmruby.a*
10. Izdelava *build/host/bin/mirb* s prevajanjem *mrbgems/mruby-bin-mirb/tools/mirb/mirb.c* in povezovanjem z *build/host/lib/libmruby.a*

3.2 KRATEK OPIS PREVAJALNIKA ZA MRUBY

MRuby prevajalnik je objektna datoteka z imenom ***mrbc***. Najdemo jo v mapi *bin* na korenskem nivoju izvorne kode MRuby. Podpira naslednje bolj uporabljane vhodne parametre:

- -c preverjanje sintakse
- -o<izhodna datoteka > postavi izhod v <izhodno datoteko>
- -v izpiši verzijo in vklopi funkcija podrobnega izpisa
- -g generiraj informacije za razhroščevanje

Datoteko z MRuby kodo lahko enostavno prevedemo do vmesne kode, ang. *bytecode* s komando: mrbc *ime_datoteke.rb*. Kot rezultat dobimo novo datoteko s končnico mrb.

Za tem jo lahko izvršimo s komando: mruby –b *ime_datoteke.mrb*.

3.3 KRATEK OPIS PREVAJALNIKA ZA MRUBY_CC

MRuby_cc prevajalnik je še prototip. Za zdaj najdemo na spletu le alfa verzijo. Po uspešnem prevajanju izvorne kode se v mapi *bin* pojavi objektna datoteka, tokrat z imenom ***mrbcc***. Pri instalaciji prevajalnika pa lahko nastavimo pot, kamor se bo namestil.

Podpira naslednje uporabne vhodne parametre:

- -P predhodna obdelava datoteke
- -M kompiliranje v mruby vmesno kodo
- -C kompiliranje v datoteko v C kodo
- -c kompiliranje v objektno datoteko
- -o destinacija izhodne datoteke

MRuby kodo enostavno prevedemo z ukazom: *mrbcc vhodna_datoteka.rb –o izhodna.rb*

3.4 ZAGON MRUBY PROGRAMOV

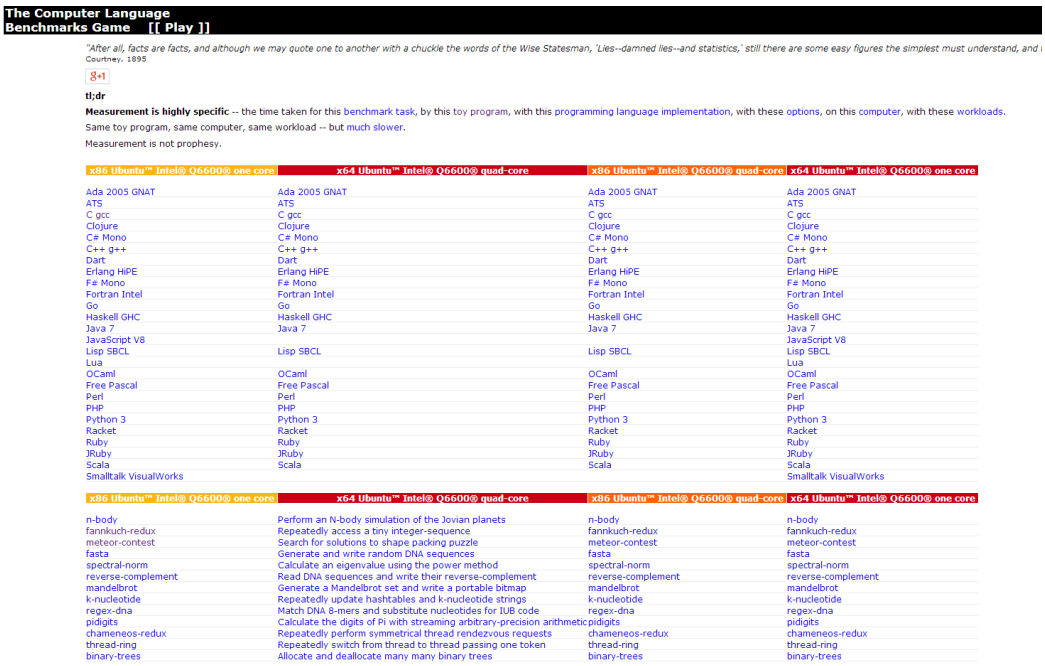
MRuby programe lahko poženemo na različne načine. Npr. preko:

- **interpretiranja kode**, kar je značilno za skriptne jezike. Podobno kot pri Ruby-ju kodo postavimo v datoteko s končnico .rb in jo poženemo z ukazom *mruby*. Prednost te oblike je hiter razvojni ciklus. Slabost je, da moramo datoteko shraniti na datotečni sistem in hkrati program mruby, ki bo kodo izvedel. Koda se pred izvedbo pretvori v vmesno kodo (ang. *byte-code*).
- **interaktivne lupine (mirb)**, ki je izredno praktična, saj lahko kodo vanjo neposredno vnašamo in poganjamo ter se tako izognemo nepotrebnemu pisanju v datoteko in prevajanju oz. poganjanju le-te. Prav tako nam omogoča hitro preverjanje pravilnosti sintakse. Ne moremo pa je uporabiti za resnejše namene.
- **bitne-kode**, kjer MRuby prevajalnik kodo prevede v vmesno kodo, ki se nato požene, podobno kot pri Javi. Prevajanje do vmesne kode zahtevamo z ukazom *mrbc*, ki je prav tako del MRuby-ja. Vmesna koda se hrani v datoteki s končnico .mrb. Podobno kot pri interpretiranju, poženemo vmesno kodo s programom *mruby*, tokrat s parametrom *-b* in imenom datoteke. Pri tej obliki končnemu uporabniku ni potrebno poslati izvorne kode, sama koda pa se pred izvajanjem ponovno ne prevaja.
- **kode C**, ki omogoča vgradnjo MRuby kode znotraj kode C. Za to poskrbi program *mrbc* v kombinaciji s parametrom *-C*, ki zna surovo MRuby kodo prevesti v C in jo nato oviti v C funkcijo, ki jo lahko kasneje tudi spreminjamo. Primer je opisan v podpoglavju 2.4.
- **binarne kode C**, ki za razliko od prejšnje točke ustvari le zbirko bajtov, ločenih z vejico. Ponovno končni uporabnik za delo ne potrebuje izvorne kode. Tak program je povsem samostojen, koda pa kompaktnejša. Za slabost štejemo težje vzdrževanje in oteženo delo s samo kodo, saj imamo tako daljši razvojni cikel: programiranje MRuby -> prevajanje (*mrbc*) -> vključevanje kode C -> Prevajanje kode C (*gcc*) -> Test.

4. PREVERJANJE DELOVANJA PREVAJALNIKA Z MRUBY

4.1 IZBIRA SPLETNE STRANI TESTNIH PROGRAMOV

Uporabili bomo testne primere s strani: http://benchmarksgame.alioth.debian.org.

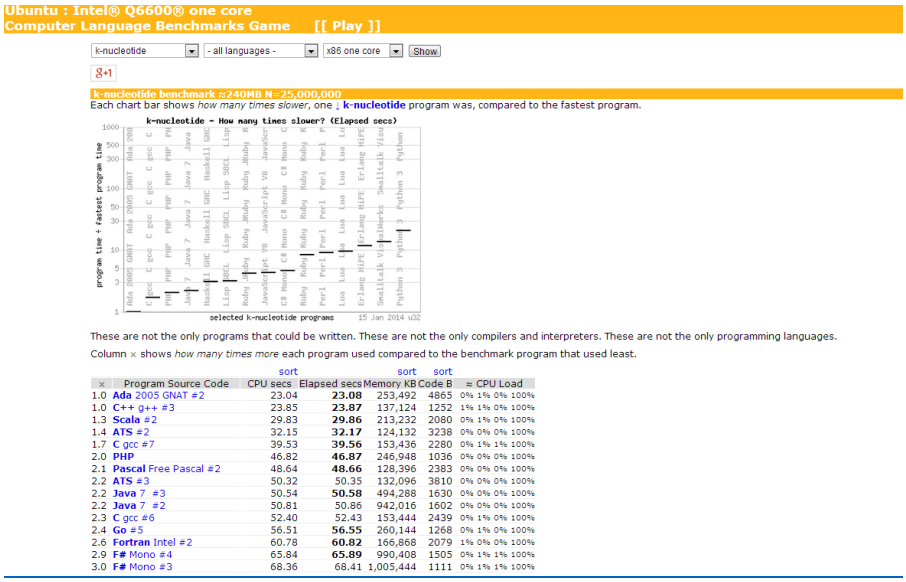


SLIKA 2 - SPLETNA STRAN S TESTNIMI PRIMERI [2]

Na njej najdemo 13 različnih programov oz. algoritmov, napisanih v več verzijah za jezike: C, C++, Python, Ruby, Perl, Java, C#, Lisp, Fortran, Haskell, Ada, itd. Vsak program ima svojo podstran, vsaka verzija pa svoj razdelek v tabeli, kjer je zabeležen čas izvajanja, količina porabljenega pomnilnika in obremenjenost jeder.

Izbrani so bili vsi testni primeri: *n-body, fannkuch-redux, meteor-contest, fasta, spectral-norm, reverse-complement, mandelbrot, k-nucleotide, regex-dna, pidigits, chameneos-redux, thread-ring, binary-trees.*

Vsak izmed programov ima na spletni strani povezave do podstrani, na katerih je mogoče najti primere izvorne kode, napisane za večino bolj poznanih programskih jezikov. Naj omenimo še dejstvo, da se algoritmi objavljeni na spletni strani dnevno posodabljajo ali celo nadomestijo z novimi, hitrejšimi, zato obstaja velika verjetnost, da so ti trenutno povsem drugačni kot za časa pisanja tega dela.



Ubuntu : Intel® Q6600® one core
Computer Language Benchmarks Game [[Play]]

k-nucleotide | - all languages - | x86 one core | Show

k-nucleotide benchmark ≈240MB N=25,000,000

Each chart bar shows *how many times slower*, one ↓ **k-nucleotide** program was, compared to the fastest program.

k-nucleotide - How many times slower? (Elapsed secs)

program time ÷ fastest program time

selected k-nucleotide programs 15 Jan 2014 u32

These are not the only programs that could be written. These are not the only compilers and interpreters. These are not the only programming languages.

Column × shows *how many times more* each program used compared to the benchmark program that used least.

×	Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	≈ CPU Load
1.0	Ada 2005 GNAT #2	23.04	**23.08**	253,492	4865	0% 1% 0% 100%
1.0	C++ g++ #3	23.85	**23.87**	137,124	1252	1% 1% 0% 100%
1.3	Scala #2	29.83	**29.86**	213,232	2080	0% 1% 0% 100%
1.4	ATS #2	32.15	**32.17**	124,132	3238	0% 0% 0% 100%
1.7	C gcc #7	39.53	**39.56**	153,436	2280	0% 1% 1% 100%
2.0	PHP	46.82	**46.87**	246,948	1036	0% 0% 0% 100%
2.1	Pascal Free Pascal #2	48.64	**48.66**	128,396	2383	0% 0% 0% 100%
2.2	ATS #3	50.32	50.35	132,096	3810	0% 0% 0% 100%
2.2	Java 7 #3	50.54	**50.58**	494,288	1630	0% 0% 0% 100%
2.2	Java 7 #2	50.81	50.86	942,016	1602	0% 0% 0% 100%
2.3	C gcc #6	52.40	52.43	153,444	2439	0% 1% 0% 100%
2.4	Go #5	56.51	**56.55**	260,144	1268	0% 1% 0% 100%
2.6	Fortran Intel #2	60.78	**60.82**	166,868	2079	1% 0% 0% 100%
2.9	F# Mono #4	65.84	**65.89**	990,408	1505	0% 1% 1% 100%
3.0	F# Mono #3	68.36	68.41	1,005,444	1111	0% 1% 1% 100%

SLIKA 3 - PODSTRAN PROGRAMA K-NUCLEOTIDE [3]

4.2 SPECIFIKACIJE SISTEMA

Da bi bili rezultati in ugotovitve smiselni, navajam specifikacije sistema, kjer se bodo izvajale meritve.

Sistem: Lenovo ThinkPad W500

- CPE: Intel® Core™ 2 Duo CPU T9600 @ 2.80 GHz (2 jedra, 2 niti)
- Pomnilnik: 8,00 GB (DDR2 533 MHz)
- Operacijski sistem: Microsoft Windows 7 SP1, 64-bit

Virtualka: Oracle VM Virtualbox, verzija 4.2.18 r88780

- Dodeljen 1 CPU, 1 nit in 4GB pomnilnika
- Operacijski sistem: Ubuntu 10.04.4 LTS (Lucid Lynx), 32-bit

Programska orodja in prevajalniki:

- C prevajalnik: gcc, verzija 4.7.2 (Ubuntu/Linaro 4.7.2-2ubuntu1), i686
- C++ prevajalnik: g++, verzija 4.7.2 (Ubuntu/Linaro 4.7.2-2ubuntu1), i686
- Python, verzija 2.7.3
- Ruby, verzija 2.0.0p0 (2013-02-24, rev. 39474), i686-linux
- MRuby, verzija v1.0.0
- Prevajalnik za MRuby_cc: mrbcc, verzija 0.1-alpha (revision mruby @ b783311)

4.3 IZBIRA IN PRILAGODITEV TESTNIH PROGRAMOV

Vsi programi pod jeziki C, C++, Python in Ruby so bili uspešno prevedeni in pognani. Nekateri za delovanje potrebujejo namestitev dodatnih knjižnic oz. uporabljajo kodo zajeto v zunanjih datotekah, ki so sicer na voljo na spletu, spet drugi se naslanjajo na vtičnike ipd.

Jezike C, C++ in Python bomo uporabil le za primerjavo. Podrobneje nas zanimajo le: Ruby, MRuby in zanj prevajalnik, MRuby_cc. Pri slednjih dveh so nastopile velike težave, zato sem kodo karseda prilagodil tako, da jo je moč prevesti in izvesti pod vsemi tremi kombinacijami. Prilagojen del kode, ki se pred tem ni prevedel bodisi pod enim ali drugim jezikom sem umestil v vse tri programe, vključno z Ruby-jem, da bi bile primerjave hitrosti in sklepi smiselni. Težave na katere sem naletel so opisane v nadaljevanju. Nanašajo se izključno na MRuby oz. prevajalnik MRuby_cc.

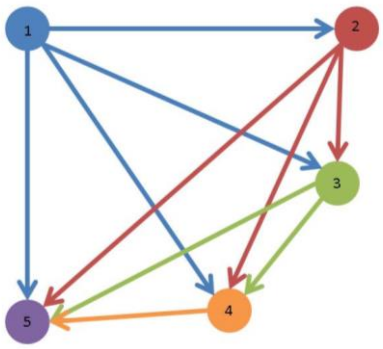
MRuby ima večje težave pri interpretiranju sledečih programov: *reverse-complement*, *regex-dna*, *chameneos-redux*, *thread-ring* in *meteor-contest*, pri MRuby_cc pa dodatno še pri programih *k-nucleotide in fasta*, kar lahko razberemo tudi v spodnji tabeli, ki prikazuje uspešnost prevajanja in zaganjanja.

	PREVAJANJE			ZAGON						SKUPAJ
	C	C++	mruby_cc	C	C++	Python	Ruby	mruby	mruby_cc	
n-body	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
fannkuch-redux	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
mandelbrot	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
binary-trees	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
spectral-norm	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
pidigits	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
k-nucleotide	✓	✓	×	✓	✓	✓	✓	✓	×	×
fasta	✓	✓	✓	✓	✓	✓	✓	✓	napaka	×
reverse-complement	✓	✓	×	✓	✓	✓	✓	napaka	×	×
regex-dna	✓	✓	×	✓	✓	✓	✓	napaka	×	×
chameneos-redux	✓	✓	×	✓	✓	✓	✓	napaka	×	×
thread-ring	✓	✓	×	✓	✓	✓	✓	napaka	×	×
meteor-contest	✓	✓	×	✓	✓	✓	✓	napaka	×	×

TABELA 1 - PRIKAZ USPEŠNOSTI PRILAGODITVE, PREVAJANJA IN ZAGONA TESTNIH PRIMEROV

4.3.1 N-BODY

Potrebno je modelirati orbite Jupitrovih planetov z uporabo enostavnega, enakovrednega simplektičnega integratorja. Gre za starodavni, klasičen problem, ki obravnava interakcije med telesi v množici. Govorimo o problemu N teles. V množici N je potrebno izračunati medsebojne sile za vsak par teles. Začnemo tako, da izberemo poljubno točko. Zanjo moramo izračunati N-1 operacij, vsaka naslednja točka pa mora preračunati operacijo manj, kot točka pred njo. Imamo N * (N-1) operacij. Časovna zahtevnost je torej $O(n^2)$ in močno narašča s številom točk.



SLIKA 4 - INTERAKCIJE MED N TELESI [3]

Napake oz. težave iste sorte so omenjene le enkrat. Poglejmo, kje se je zataknilo.

1. Težava: datoteka *nbody.rb*, vrstica št. 9 – branje parametrov ukazne vrstice:

```
n = Integer(ARGV[0])
```

IZPIS 6 – VZOREC PROBLEMATIČNE KODE: MANJKAJOČI RAZRED INTEGER

MRuby ob prvem zagonu javi napako, saj ne pozna razreda Integer.

```
trace:
        [0] nbody_m.rb:8
nbody_m.rb:8: undefined method 'Integer' for main (NoMethodError)
```

IZPIS 7 – JAVLJENA NAPAKA ZARADI MANJKAJOČEGA RAZREDA

Rešitev za MRuby je enostavna.

```
n = ARGV[0].to_i
```

IZPIS 8 – POPRAVLJEN DEL KODE PROGRAMA N-BODY ZA MRUBY

Pri naslednjem poizkusu javi napako MRuby_cc, saj ne zna dostopati do parametrov podanih v ukazni vrstici:

```
Uncaught exception:
NameError: uninitalized constant ARGV
```

IZPIS 9 – JAVLJENA NAPAKA OB ZAGONU POD MRUBY_CC

Ena izmed možnosti je, da vrednost spremenljivke vprogramiramo v samo kodo. V našem primeru si to lahko dovolimo, saj je cilj izmeriti hitrost izvajanja danega algoritma, koda pa ni produkcijska.

2. Težava: datoteka *nbody.rb*, vrstica št. 134:

```
puts "%.9f" % energy(BODIES)
```

IZPIS 10 – VZOREC PROBLEMATIČNE KODE: FORMATIRANI IZPIS

Napaka, ki jo javi MRuby:

```
        [0] nbody_m.rb:131
nbody_m.rb:131: undefined method '%' for "%.9f" (NoMethodError)
```

IZPIS 11 – JAVLJENA NAPAKA: FORMATIRANI IZPIS, MRUBY

Napaka, ki jo javi MRuby_cc:

```
Uncaught exception:
NoMethodError: undefined method '%' for "%.9f"
```

IZPIS 12 – JAVLJENA NAPAKA: FORMATIRANI IZPIS, MRUBY_CC

Formatirani izpis pri ukazu *puts* obema povzroča preglavice. Skupna rešitev je enostavna, ukaz zamenjamo z ukazom *printf*.

4.3.2 FANNKUCH-REDUX

Fannkuch je okrajšava za nemško besedo *Pfannkuchen*, oz. palačinke (analogija obračanja palačink). Gre za izvajanje permutacij

Ideja pri temu algoritmu je:

1. Vzeti poljubno permutacijo {1,...,n}, npr.: {4,2,1,5,3}.
2. Vzeti prvi element, v našem primeru 4, in obrniti vrstni red prvih 4 elementov: {5,1,2,4,3}.
3. To ponavljati dokler prvi element ni 1, tako da obračanje ne povzroči več nobene spremembe: {3,4,2,1,5}, {2,4,3,1,5}, {4,2,3,1,5}, {1,3,2,4,5}.
4. Preštejemo število obračanj, v našem primeru 5.
5. Zadržimo kontrolno vsoto (v nadaljevanju vsota)
    - vsota = vsota + (če je indeks permutacije sod potem + št.obračanj sicer - št.obračanj)
    - vsota = vsota + (predznak_1_1 * št.obračanj)
6. To storimo za vse n! permutacije, in zabeležimo maksimalno število obračanj potrebnih za katerokoli permutacijo.

Domnevno bo maksimalno število aproksimirano s približkom n*log(n) ko gre n proti neskončnosti.

1. Težava – datoteka *fann.rb*, vrstica št. 21:

```
q[0..-1] = p
```

IZPIS 13 - PROBLEMATIČNA KODA: OBRAČANJE VRSTNEGA REDA ELEMENTOV V ZBIRKI

Zgornja koda, napisana za Ruby, spremenljivki *q* priredi obrnjeno zbirko elementov iz spremenljivke *p*. Pri prevajanju dobimo napako. MRuby in MRuby_cc te sintakse ne poznata.

Zgornji ukaz zamenjamo z ukazom *reverse* in že se nam prevede pod obema:

```
q = p.reverse
```

IZPIS 14 – POPRAVLJEN DEL KODE

2. **Težava** – datoteka *fann.rb*, vrstica št. 47:

```
p[1], p[2] = p[2], p[1]
```

IZPIS 15 – VZOREC PROBLEMATIČNE KODE: OBRAČANJE VREDNOSTI ELEMENTOV V ZBIRKI

Ne eden ne drugi ne prepozna ukaza za obračanje vrednosti elementov v zbirki v enem koraku. Ukaz je potrebno razbiti na več delov:

```
t = p[2]      # shranjevanje začasne vrednosti
p[2] = p[1]   # obračanje prvega elementa
p[1] = t      # obračanje prvega elementa
```

IZPIS 16 – POPRAVLJEN DEL KODE ZA OBRAČANJE VREDNOSTI ELEMENTOV V ZBIRKI

3. **Težava** – datoteka *fann.rb*, vrstica št. 72. MRuby ne pozna funkcije, ki vstavi vrednost *t* v zbirko, takoj za elementom *i*.

```
p.insert(i, t)
```

IZPIS 17 - VZOREC IZ KODE: MANJKAJOČA FUNKCIJA

Ideja je zbirko na poziciji *i* razdeliti na dva dela, preveriti ali imamo na vsaki strani po vsaj en element (MRuby novi spremenljivki dodeli nično vrednost v kolikor je prirejen razdeljen seznam prazen), uporabiti metodo za dodajanje elementa na konec zbirke:

```
left = p[0, i]          # leva polovica
if left.nil?            # preverjanje nične vrednosti
  left = []
end
left.push(t)            # dodajanje elementa levi polovici
right = p[i, p.length] # desna polovica
if not right.nil?       # če je desna polovica prazen seznam, smo zaključili
left[left.length, 0] = right # sicer spojimo zbirki še desno polovico
end
p = left                # novo zbirko priredimo spremenljivki p
```

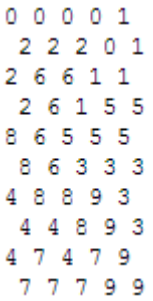
IZPIS 18 – IMPLEMENTACIJA BLOKA KODE, KI OPONAŠA FUNKCIJO *INSERT(INDEX, OBJECT)*

4.3.3 METEOR-CONTEST

Program lahko ponazorimo s sestavljanko iz 10 vrst po 5 šesterokotnih celic. Na desko postavljamo 10 ploščic, označili jih bomo od 0 do 9. Vsaka ploščica je sestavljena iz 5 heksagonalnih celic. Ena izmed možnih rešitev problema je sledeča: obdelujemo vsako vrstico od leve proti desni in od zgoraj navzdol. Zapišemo si številko ploščice v vsaki celici na deski in ustvarimo niz znakov iz vseh 50 števil. Najkrajša rešitev sestavljanke bi bila:

00001222012661126155865558633348893448934747977799

Izpišemo najkrajšo in najdaljšo rešitev v formatu, ki posnema šesterokotno ploščo sestavljanke.



SLIKA 5 - ŠESTEROKOTNA TABELA [5]

1. Težava – datoteka *meteor.rb*, vrstica št. 35:

```
if ((blank_board & mask == 0 && !prunable(blank_board | mask, 0, true))
```

IZPIS 19 - VZOREC PROBLEMATIČNE KODE: BITNI OPERATORJI

Pri prvem poizkusu zagona pod MRuby ugotovimo, da MRuby ne pozna bitnih operatorjev. Težavo sem odpravil z uporabo logičnih operatorjev vendar me je pri naslednjem prevajanju presenetila kompleksnejša napaka, ki jo v 281. vrstici iste datoteke povzroči dostop do elementa zbirke, ki pri dostopu uporablja bitni ključ.

```
if (i[bit] == 1) then
```

IZPIS 20 - VZOREC PROBLEMATIČNE KODE: DOSTOP DO ZBIRKE, BITNI KLJUČ

Po nekaj poizkusih (predelava na drugi tip zbirke, implementacija zunanje funkcije za poizvedovanje itd.) sem prišel do zaključka, da se rešitve ne da realizirati le z osnovnim znanjem jezika MRuby. Program izključimo iz seznama za merjenje hitrosti za MRuby.

4.3.4 FASTA

Algoritem FASTA poravnava zaporedje DNK, RNK oz. beljakovinskih molekul. Namen je ugotoviti regije podobnosti, ki so lahko posledica funkcionalnih, strukturnih ali evolucijskih razmerij med zaporedji. Med prvimi sta ga opisala David J. Lipman in William R. Pearsonin leta 1985. Njegova zapuščina je FASTA format zapisa, ki dandanes navzoč v bioinformatiki.

Program med delovanjem generira dolg izpis, ki ga kasneje uporabimo pri nekaterih ostalih, kot npr.: *k-nucleotide* ali *reverse-complement*.

Pri tem programu naletimo na dve manjši in eno večjo težavo. Manjši se nanašata na prvi del algoritma in se ju da zaobiti. Večja se pojavi v drugem delu v metodi *make_random_fasta* in se nanaša na ukaz *eval*, ki pa, kot že vemo, v MRuby-ju ni na voljo. Obstaja alternativa (omenili jo bomo kasneje pri obravnavanju MRuby GEM paketov), vendar še ni ravno zrela in ne deluje pravilno.

1. Težava – datoteka *fasta.rb*, funkcija *slice!(index, length)*, ki briše elemente od indeksa *n* do dolžine *l* in vrednost priredi obstoječemu nizu znakov pod MRuby-jem še ni zasnovana:

```
def make_repeat_fasta(src, n)
    v = nil
    width = 60
    l = src.length
    s = src * ((n / l) + 1)
    s = s.slice!(0, n)        # funkcija slice!
```

IZPIS 21 - VZOREC IZ KODE: MANJKAJOČA FUNKCIJA *SLICE!(INDEX, LENGTH)*

Uporabimo lahko npr. funkcijo *slice(index, length)*, ki deluje na podoben način, vendar iz danega niza znakov vrača elemente od indeksa n in dolžine l. Sestavimo nov ukaz:

```
s = s.slice(0, n) + s.slice(n+l, n.length)
```

IZPIS 22 - PRIREJENA KODA Z UPORABO FUNKCIJE *SLICE*

Opomba: algoritem je bil med izdelavo tega dela posodobljen. Stara verzija ni več na voljo, nova pa ne uporablja več funkcije *slice!*. Merjenje hitrosti sem tako izvedel na **novi** verziji.

2. Težava – strnjen ukaz *scan* z uporabo regularnih izrazov in izpis.

```
puts (s.scan(/.{1,#{width}}/).join("\n"))
```

IZPIS 23 - VZOREC PROBLEMATIČNE KODE: UKAZ *SCAN*

Zgornji košček kode opravlja kar veliko nalogo in sicer se sprehaja po nizu s in izpisuje vrstico za vrstico podane dolžine. Ponovno se moramo poglobiti globoko v dokumentacijo, da bi razumeli kako deluje funkcija *scan* v kombinaciji z regularnim izrazom. Sprogramiral sem proceduro, katera oponaša delovanje omenjene funkcije:

```
$toPrint = s.length
$printed = 0

while $toPrint-$printed > 1 do
  puts s.slice($printed, width.to_i)
  s.slice($printed, width.to_i)
  $printed = $printed + width.to_i
  $toPrint = $toPrint - $printed
end
```

IZPIS 24 - POPRAVLJENA KODA

Opomba: nova verzija algoritma je popolnoma spremenjena in hkrati tudi nekoliko hitrejša. Spremenil se je tudi algoritem za izpis v prvem delu. Časa izvajanja za MRuby_cc še vedno ni možno izmeriti, saj se program predčasno prekine z napako »*memory corruption*«.

3. Težava – uporaba ukaza *eval* za zagon korakanja po vrsticah tabele in pogojenega izpisa nekaterih števil v kombinaciji s sofisticirano idejo, ki predhodno kreira te pogoje.

```
eval <<-EOF
   (1..(n/width)).each do |i|
     puts rwidth.collect{#{collector}}.join
   end
   if n%width != 0
     puts (1..(n%width)).collect{#{collector}}.join
   end
EOF
```

IZPIS 25 - VZOREC IZ KODE: MANJKAJOČA FUNKCIJA EVAL

Predelava te funkcije zahteva širše poznavanje programskega jezika Ruby in delovanje funkcije *eval*. Med testiranjem opazimo tudi, da ima MRuby_cc težave pri veliki količini izpisa, ki neprestano odteka na standardni izhod. Odločil sem se, da drugi del izpisa FASTA izpustim in program ustrezno skrajšam.

4.3.5 SPECTRAL-NORM

Gre za razširitev algoritma normalizacije vektorja za matrike. Algoritem obsega matematično računsko operacijo, ki se izvaja nad neskončno matriko A z vrednostmi: $a_{11}=1$, $a_{12}=1/2$, $a_{21}=1/3$, $a_{13}=1/4$, $a_{22}=1/5$, $a_{31}=1/6$, itd.

Prilagajanje kode je potekalo brez večjih zapletov.

4.3.6 REVERSE-COMPLEMENT

Algoritem služi računanju inverznega komplementa (oz. komplementarne verige) zaporedja molekule DNK. Na vhodu potrebujemo datoteko *revcomp-input25000000.txt*, katero sem generiral s pomočjo programa FASTA. Program bere vrstico za vrstico in na standardni izhod izpisuje id, opis in inverzni komplement zaporedja v formatu FASTA.

SLIKA 6 - INVERZNI KOMPLEMENT (NAPREJ) [6]

1. Težava – datoteka *reverse.rb*, vrstica št. 11:

```
seq.reverse!.tr!('wsatugcyrkmbdhvnATUGCYRKMBDHVN',
'WSTAACGRYMKVHDBNTAACGRYMKVHDBN')
```

IZPIS 26 - VZOREC IZ KODE: MANJKAJOČA FUNKCIJA *TR!*

Nepoznan ukaz *tr!*. Ukaza *tr!* in *tr* še nista implementirana. Poizkus uporabe metode *gsub* v zanki se je izkazal za dovolj počasnega, da je povzročil napako v pomnilniku, ang. *Out of memory exception*. Ostali pristopi so časovno še bolj zahtevni in prav tako ne obrodijo sadov.

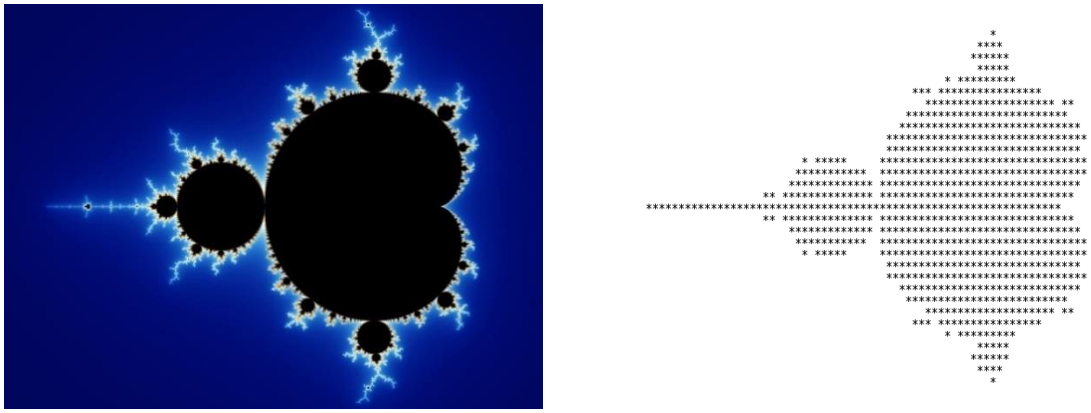
2. Težava – branje toka podatkov preko STDIN ali datoteke.

```
STDIN.each do |line|
File.open("revcomp-input25000000.txt", "r").each_line do |line|
```

IZPIS 27 – VZORCA IZ KODE: OBLIKI BRANJA PODATKOV NA VHODU

4.3.7 MANDELBROT

Znova imamo pred seboj klasičen algoritem. Program izriše znani fraktal z imenom *Mandelbrot set* [-1.5-i,0.5+i] na NxN bitni sliki. Izpis izriše bit po bitu v prenosnem formatu bitne slike - PBM.



SLIKA 7 - MANDELBROT IZRIS V VISOKI LOČLJIVOSTI (LEVO) IN TEKSTOVNI OBLIKI [7,8]

Prilagajanje za vse tri jezike iz družine Ruby je tudi tokrat potekalo brez zapletov.

4.3.8 K-NUCLEOTIDE

Program na vhodu bere vrstico za vrstico iz datoteke, predhodno generirane s pomočjo programa FASTA, in skuša izluščiti DNK sekvenco »THREE«. Za vsak prebrani okvir računa vrednosti in jih preko ključa shranjuje v tabelo. Sešteti je potrebno vse 1-verižne in 2-verižne sekvence in jih izpisati ter izračunati relativno frekvenco, na koncu pa še sortirati po frekvenci padajoče in nato naraščajoče po ključu *k-nucleotide*.

Program pod MRuby_cc za zdaj ne deluje. Delovanje pod MRuby-jem mu ponovno omogoča GEM paket *mruby-io*.

1. Težava – datoteka *nucl.rb*, vrstica št. 14:

```
(f ... n).step(length) do |i|
```

IZPIS 28 - VZOREC IZ KODE: MANJKAJOČA METODA *STEP*

Metoda *step* ni na voljo. Njeno funkcionalnost sem analiziral in prilagoditev realiziral takole:

```
s1 = length
(f ... n).each do |i|
   s1 = s1 - 1
   if s1 == 0
      table[seq[i,length]] += 1
      s1 = length
   end
end
```

IZPIS 29 - PRIREJENA KODA ZA FUNKCIJO *STEP*

2. Težava – vrstice št. 42 - 47:

```
line = STDIN.gets while line !~ /^>THREE/
line = STDIN.gets
while (line !~ /^>/) & line do
    seq << line.chomp
    line = STDIN.gets
end
```

IZPIS 30 - VZOREC TEŽAVNE SINTAKSE

Sintaksa je tu precej problematična, saj vrstica 42 prepreči nadaljnjo analizo kode, saj se MRuby tukaj zmede in ni mu več jasno kje oz. kako se vrstica konča. Po daljši analizi, programiranju in testiranju sem prišel do nekoliko počasnejšega ekvivalenta:

```
last_line = ""
found = 0
File.open("knucleotide-input25000000.txt", "r").each_line do |line|
  last_line = line
  if found == 1 and not (line.index(">") == 0)
    seq << line.chomp
  end
  if found == 0 && line.index(">THREE") == 0
   found = 1
  end
end
```

IZPIS 31 – POPRAVLJEN BLOK KODE

3. Težava – vrstice 42, 43 in 46:

```
line = STDIN.gets while line
```

IZPIS 32 – VZOREC NEDELUJOČE KODE: BRANJE PODATKOV PREKO STANDARDNEGA VHODA

Branje datoteke ali toka podatkov preko standardnega vhoda v MRuby-ju še ni podprto. Problema sem se lotil z iskanjem alternativnih možnosti, da bi MRuby pripravil do branja toka podatkov bodisi iz datoteke bodisi iz standardnega vhoda. Datoteka za branje je prevelika, da bi jo lahko integrirali znotraj datoteke z izvorno kodo ali kaj podobnega. Poslužimo pa se lahko GEM paketa *mruby-io* in tako pridemo do zaključka, ki za zdaj deluje le za MRuby:

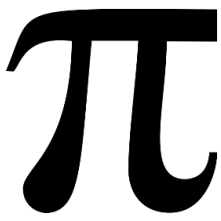
```
File.open("revcomp-input25000000.txt", "r").each_line do |line|
```

IZPIS 33 - PRIMER: BRANJE IZ DATOTEKE

Verzija MRuby_cc, ki je trenutno dostopna v GIT-u (https://github.com/slivnik/mruby_cc) ima težave pri grajenju omenjenega GEM paketa. Verzija MRuby-ja, ki je umeščena v MRuby_cc, ni kompatibilna s stabilno verzijo GEM paketa, ki je trenutno na razpolago. Branje iz datoteke ali standardnega vhoda pod MRuby_cc zaenkrat **še ni možno**. Program je odstranjen iz seznama za merjenje hitrosti pod MRuby_cc.

4.3.9 PIDIGITS

Kot morda že naslov pove, ta program računa decimalke števila PI. Pri računanju se uporablja postopkovni Spigotov algoritem za računanje decimalk. Izračun prvih N decimalk se izpiše na standardni izhod v vrsticah dolžine 10.



SLIKA 8 - ŠTEVILO PI [9]

Algoritmi, ki so na voljo za računanje števila PI, uporabljajo razred Bignum, ki omogoča ohranitev natančnosti za števila, ki prekoračijo meje razreda Fixnum. Instance razreda Bignum se ustvarijo tik preden matematična operacija povzroči preliv. Razred Bignum nam, kot vemo, ni na voljo, zato lahko ta program pod MRuby-jem izračuna le tri decimalke. Za dodano vrednost bi bilo potrebno sprogramirati lastni podatkovni tip in prav tako vse osnovne matematične operacije, ki bi se izvajale nad njim, kar presega moje znanje jezika Mruby.

4.3.10 REGEX-DNA

Podobno kot *k-nucleotide* in *reverse-complement*, program na vhodu uporablja FASTA datoteko, ki jo prebere v celoti. Iz prebranega nato z uporabo regularnih izrazov odstrani opise sekvenc, poišče vse DNK verige oz. njihove komplemente in zadetke prešteje. Na koncu izpiše tri zabeležene dolžine sekvenc.

1. **Težava** – datoteka *regex.rb*, vrstica 12:

```
seq.gsub!(/>.*\n|\n/,"")
```

IZPIS 34 - VZOREC PROBLEMATIČNE KODE

Težave z regularnimi izrazi smo že omenili, podajmo le možno rešitev na zgornji primer:

```
reg = HsRegexp.new(">.*\n|\n")
begin
match_data = reg.match(seq)
seq = seq.sub(match_data[0], "")
end while match_data.nil?
```

IZPIS 35 - POPRAVLJENA KODA, Z UPORABO GEM PAKETA

V nadaljevanju lahko opazimo, da je program zasnovan z uporabo niti, ki jih pod MRuby-jem in MRuby_cc ne moremo pognati brez nameščanja razširitev. V GIT-u imamo sicer na voljo GEM paket *mruby-thread*, vendar je bil vsakršen poizkus namestitve paketa neuspešen.

4.3.11 CHAMENEOX-REDUX

Algoritem zajema prilagojeno verzijo igre z imenom ang. *Chameneos*, ki opisuje paradigmo sodelovanja »vsak z vsakim«. V izvirniku je povzet v delu »Chameneos, a Concurrency Game for Java, Ada and Others«, dosegljivem na naslovu: http://cedric.cnam.fr/PUBLIS/RC474.pdf

Program ustvari tri različno poimenovana bitja različnih barv: rdeče, rumeno in modro. Vsako bitje večkrat odide na kraj zbirališča in tam spozna ali čaka na spoznanje drugega *chameneos* bitja. V trenutku, ko bitje odide na sestanek še ne vemo ali je na zbirališču navzoče drugo bitje, prav tako ne vemo, ali bo navzoče v prihodnje. V primeru da se bitja srečata, zamenjata barvo,

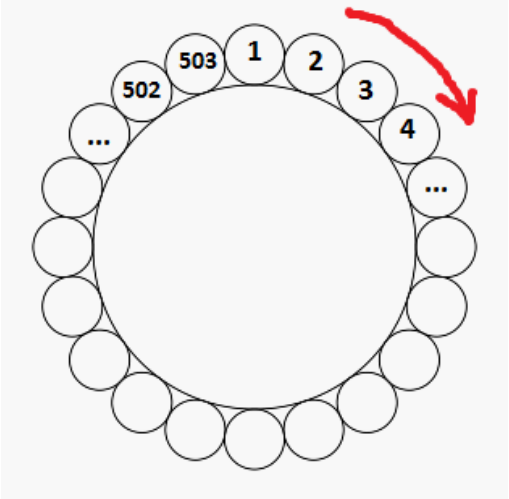
ki sovpada s komplementom barve bitja, katerega spoznata. Izpisati je potrebno število srečanj in vse barvne spremembe modrih, rdečih in rumenih bitij.

Primer. Rdeče bitje A sreča modro bitje B. Oba zamenjata barvo v rumeno.

Algoritem je realiziran z uporabo niti, bi usposobitev pod MRuby zahtevala celovito prenovo izvorne kode.

4.3.12 THREAD-RING

Ideja tega algoritma je ustvariti 503 niti, označenih od 1 do 503 in jih med seboj povezati. Vsaka nit N je povezna z nitjo N+1, razen zadnje niti št. 503, ki mora biti povezana z nitjo št. 1 in tako ustvariti neprekinjen krog. Niti št. 1 predamo žeton, nato si ga niti n-krat podajo med seboj. Izpišemo ime zadnje niti (od 1 do 503), ki sprejme žeton.

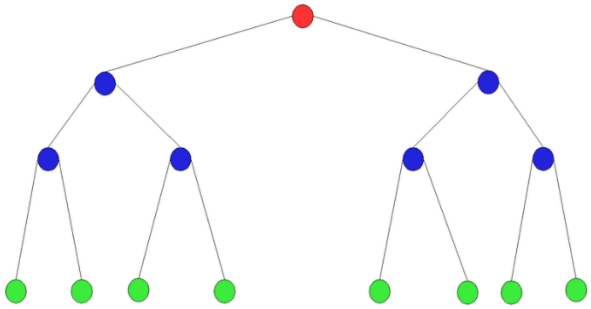


SLIKA 9 – PRIKAZ NITI

Ponovno imamo težave pri prilagajanju kode, saj je sam algoritem zgrajen na nitih. Program za zdaj ne deluje pod MRuby ali MRuby_cc.

4.3.13 BINARY-TREES (BINARNA-DREVESA)

Algoritem docela obremeni delovanje drevesne strukture s sprehajanjem po njem, ustvarjanjem, dodeljevanjem in uničevanjem novih, kratkoživih dreves. Obstaja eno dolgoživo drevo, ostala izmenično nastajajo in umirajo. Računajo se tudi kontrolne vsote vozlišč, med operacijami pa se preverja ali dolgoživo drevo še živi.



SLIKA 10 - BINARNO DREVO [10]

Posebnih težav povezanih s prilagajanjem izvorne kode ni bilo.

5. PRIPRAVA OKOLJA

Posodobitev sistema Ubuntu:

```
sudo apt-get update
sudo apt-get upgrade
```

IZPIS 36 - UKAZNI NIZ ZA POSODOBITEV SISTEMA

Namestitev prevajalnikov

S spodnjim ukazom sprožimo pričetek namestitve.

```
sudo apt-get install build-essential
```

IZPIS 37 - NAMESTITEV PREVAJALNIKA C, C++ IN G++

Namestitev Ruby-ja, verzije 2.0.0-p0

To je prva stabilna verzija serije 2.0. Dodano je bilo veliko novih funkcij in izboljšav, da bi zadovoljili rastočim in raznolikim zahtevam tega jezika.

Preden lahko uspešno namestimo Ruby 2.0.0, moramo sistem pripraviti. Za časa pisanja diplomske naloge te verzije Ruby-ja ne moremo namestiti neposredno preko lastnega paketnega upravljalca, ampak ga je potrebno ročno prevesti in sestaviti iz izvorne kode.

Namestitev knjižnic

Namestiti je potrebno vse knjižnice, ki so v odvisnosti z Ruby-jem verzije 1.9.1

```
sudo apt-get build-dep ruby1.9.1
```

IZPIS 38 - UKAZNI NIZ ZA NAMESTITEV KNJIŽNIC

Namestitev novih knjižnic, potrebnih eksplicitno za verzijo 2.0.0-p0:

```
sudo apt-get install zlib1g-dev libssl-dev libreadline6-dev libyaml-dev
```

IZPIS 39 - UKAZNI NIZ ZA NAMESTITEV DODATNIH KNJIŽNIC

Namestitev Ruby

```
cd /tmp                                                    # pomik v mapo /tmp
wget http://cache.ruby-lang.org/pub/ruby/2.0/ruby-2.0.0-p353.tar.gz # prenos Ruby-ja
tar –xvzf ruby-2.0.0-p353.tar.gz                           # ekstrakcija paketa
cd ruby-2.0.0-p353/                                        # pomik v mapo z izvorno kodo Ruby
./configure –prefix=/usr/local                             # avtom. nastavitev
Make                                                       # sestavljanje paketa
sudo make install
```

IZPIS 40 - INSTALACIJA RUBY-JA

Namestitev GEM modulov za Ruby

```
sudo gem install activesupport
```

IZPIS 41 - UKAZNI NIZ ZA INSTALACIJO POTREBNIH GEM PAKETOV

Namestitev MRuby

MRuby izvorna koda se nahaja v sistemu za nadzor različic GIT na naslovu: https://github.com/mruby/mruby.git

Zadnje meritve so potekale na verziji z dne: Dec 23, 2013 in kontrolno vsoto d957a67841. K sebi jo povlečemo z ukazom:

```
git clone https://github.com/mruby/mruby.git
cd mruby # premaknemo se v mapo mruby
git pull # izvedemo GIT PULL, da preidemo na zadnjo verzijo

make # zgradimo MRuby
```

IZPIS 42 - PRENOS IN KOMPILACIJA ZADNJE VERZIJE MRUBY-JA

Namestitev GEM modulov za MRuby:

Najprej iz GIT-a prenesemo pakete, ki jih želimo namestiti. Celotno mapo paketa prenesemo v *examples/mrgems/* (relativno na korenski direktorij izvorne kode MRuby). Na korenskem nivoju je potrebno prilagoditi še datoteko *build_config.rb* (prikazano spodaj) in nato MRuby ponovno prevesti:

```
  # Use mrbgems
  conf.gem 'examples/mrbgems/mruby-io'
  conf.gem 'examples/mrbgems/mruby-thread'
  #conf.gem 'examples/mrbgems/mruby-dir'
  #conf.gem 'examples/mrbgems/mruby-tempfile'
  #conf.gem 'examples/mrbgems/mruby-require'
```

IZPIS 43 - DODAJANJE NOVIH GEM PAKETOV V VERIGO

Namestitev MRuby_cc

Izvorno kodo prenesemo iz GIT-a in se pomaknemo na zadnjo verzijo. Poženemo avtomatsko konfiguracijo in če je odgovor pozitiven, lahko nadaljujemo s kompiliranjem in instalacijo. Korake avtomatiziramo s pomočjo spodnje skripte za lupino Bash:

```
git clone https://github.com/slivnik/mruby_cc.git # ustvar. lokalne kopije
cd mruby_cc/
git pull                                          # pomik na zadnjo verzijo
./configure CFLAGS="-g -O2" --prefix=/tmp/local   # avtomatska konfiguracija
make all                                          # začetek gradnje
make check                                        # test
make install                                      # namestitev
```

IZPIS 44 - SKRIPTA ZA PRENOS, KONFIGURACIJO IN NAMESTITEV MRUBY_CC

6. MERITVE HITROSTI

Testiranje je potekalo v lupini Bash. Prevajanje in merjenje časa izvajanja posamičnega programa sem izmeril s pomočjo skript v Python-u in Shell-u, ki so pomagale avtomatizirati prevajanje, gradnjo objektnih datotek in izvajanje programov.

Prevajanje - postopek prevajanja za jezike C, C++ in MRuby_cc

```
#! /usr/bin/env python
import os, time, shutil

def run(cmd):
	print "\t", cmd
	os.system(cmd)

tests = open("tests2run", "r").readlines()
tests = map(lambda x: x[:-1], tests)

for test in tests:
	print "Compiling", test
	os.chdir(test)
	run("./compile_all.sh")
	os.chdir("..")
```

IZPIS 45 - SKRIPTA ZA PREVAJANJE: COMPILE_ALL.PY

Merjenje hitrosti

Izbiro programa, za katerega bi radi izmerili hitrost delovanja upravlja datoteka *tests2run.*

```
mandelbrot
nbody
binary-trees
fasta
chameneos-redux
fannkuch-redux
k-nucleotide
meteor-contest
pidigits
regex-dna
reverse-complement
thread-ring
#spectral-norm      <- program se ne bo izvedel
```

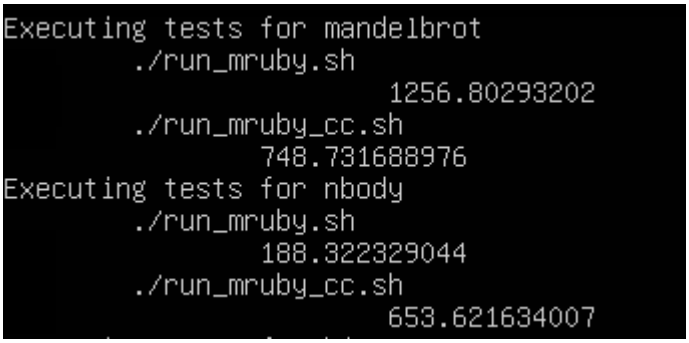
IZPIS 46 - STRUKTURA DATOTEKE TESTS2RUN

Glavna skripta za krmiljenje izvajanja testnih programov:

```
#! /usr/bin/env python

import os, time, shutil

def run(cmd):
	print "\t", cmd
	start_time = time.time()
	os.system(cmd+" > /dev/null")
	print "\t\t", time.time()-start_time

tests = open("tests2run", "r").readlines()
tests = map(lambda x: x[:-1], tests)

for test in tests:
	if test.startswith("#"): continue
	print "Executing tests for", test

	os.chdir(test)
	run("./run_c.sh")
	run("./run_cpp.sh")
	run("./run_py.sh")
	run("./run_ruby.sh")
	run("./run_mruby.sh")
	run("./run_mruby_cc.sh")
	os.chdir("..")
```

IZPIS 47 - SKRIPTA ZA MERJENJE HITROSTI: BENCH.PY

Deluje v kombinaciji z datoteko *tests2run*, ki vsebuje imena programov, katere želimo pognati. Ko poženemo skripto *bench.py*, postopoma dobivamo izpise programov in čase izvajanja:



SLIKA 11 – PRIMER IZPISA PRI MERJENJU HITROSTI ZA MANDELBROT IN NBODY

7. REZULTATI MERITEV

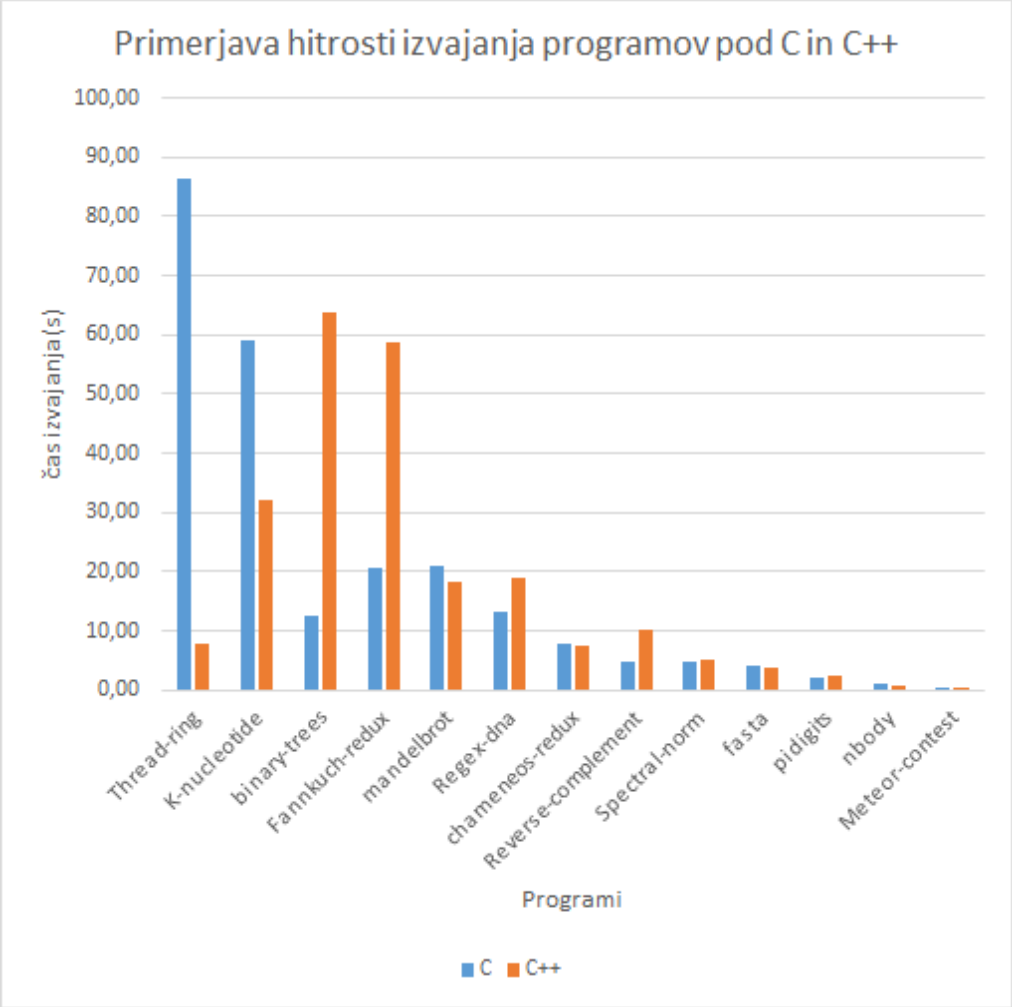
V spodnji tabeli so podani rezultati meritev testnih programov. Časi izvajanja vsakega programa posebej so bili izmerjeni 3-krat. Preverjeno je bilo tudi koliko rezultati odstopajo med seboj. Povprečne vrednosti meritev so bile vnesene v spodnjo tabelo. Zaradi mobilnosti sem se odločil delati na virtualnem stroju, na katerem sem kasneje tudi izvajal meritve. To dejstvo je potrebno izpostaviti, saj si gostiteljski in gostujoči operacijski sistem delita resurse, kar pomeni da bi v skrajnem primeru lahko neka aplikacija na gostiteljskem operacijskem sistemu odžirala npr. procesorsko moč gostujočemu sistemu in tako bi bile meritve povsem napačne. Pomnilnik se sicer da rezervirati, procesorske moči pa v mojem primeru žal ne. Da bi se temu izognil, sem v gostiteljskem sistemu pred meritvami naredil čistko, zaprl vse aktivne programe in programe v ozadju ter tako sprostil pomnilnik in CPE. Opazil sem, da rezultati med seboj bistveno ne izstopajo.

	čas izvajanja (s)					
benchmark	C	C++	Python	Ruby	MRuby	MRuby_cc
nbody	1,17	0,92	122,40	167,26	210,45	651,98
Fannkuch-redux	20,49	58,76	2456,3	3795	5617,34	4710,36
mandelbrot	21,15	18,38	2389,14	2379,8	1256,82	748,73
binary-trees	12,51	63,62	273,21	211,17	2641,44	449,9
fasta	4,16	3,75	271,81	1,23	4,36	
Reverse-complemen	4,76	10,34	41,76	17,14		
Spectral-norm	4,81	5,11	871,33	2155,86	483,59	376,93
K-nucleotide	59,18	32,25	227,39	1352,11	4940,48	
Regex-dna	13,36	18,92	28,20	46,82		
pidigits	2,26	2,49	3,44	25,20	0,05	0,04
chameneos-redux	7,96	7,68	410,41	261,85		
Thread-ring	86,26	7,79	208,71	1878,63		
Meteor-contest	0,13	0,07	7,54	9,11		

TABELA 2 - REZULTATI MERITEV

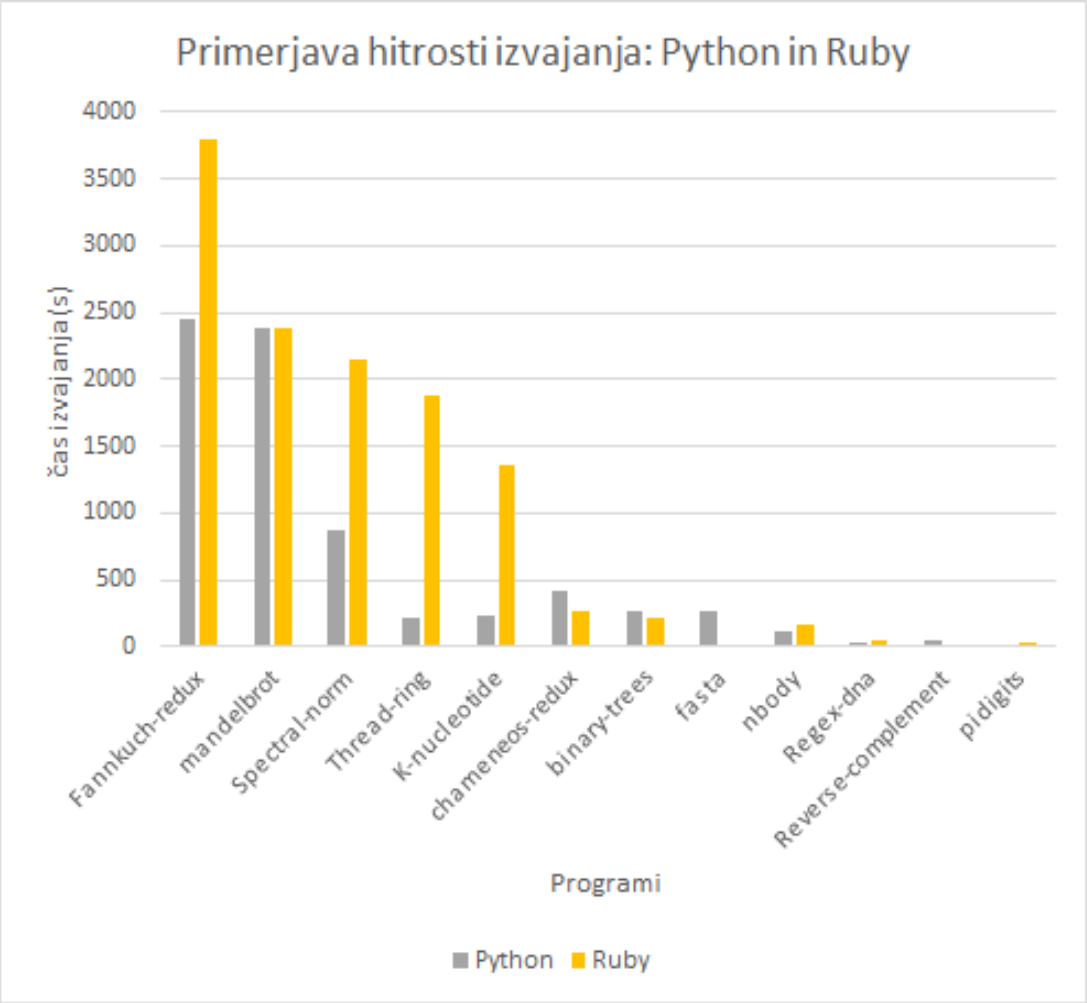
Poglejmo si najprej čase izvajanja na C in C++, da dobimo občutek kateri algoritmi so hitrejši in kateri počasnejši.

Iz spodnjega grafa ne moremo zagotovo trditi kateri od dvojice je hitrejši. Presenetljivo je recimo dejstvo, da se je pri algoritmu *thread-ring* (oz. obroč niti) jezik C izkazal za približno 10x počasnejšega. Situacija pa se je, kot vidimo, obrnila pri *binary-trees*, delu z binarnimi drevesi.



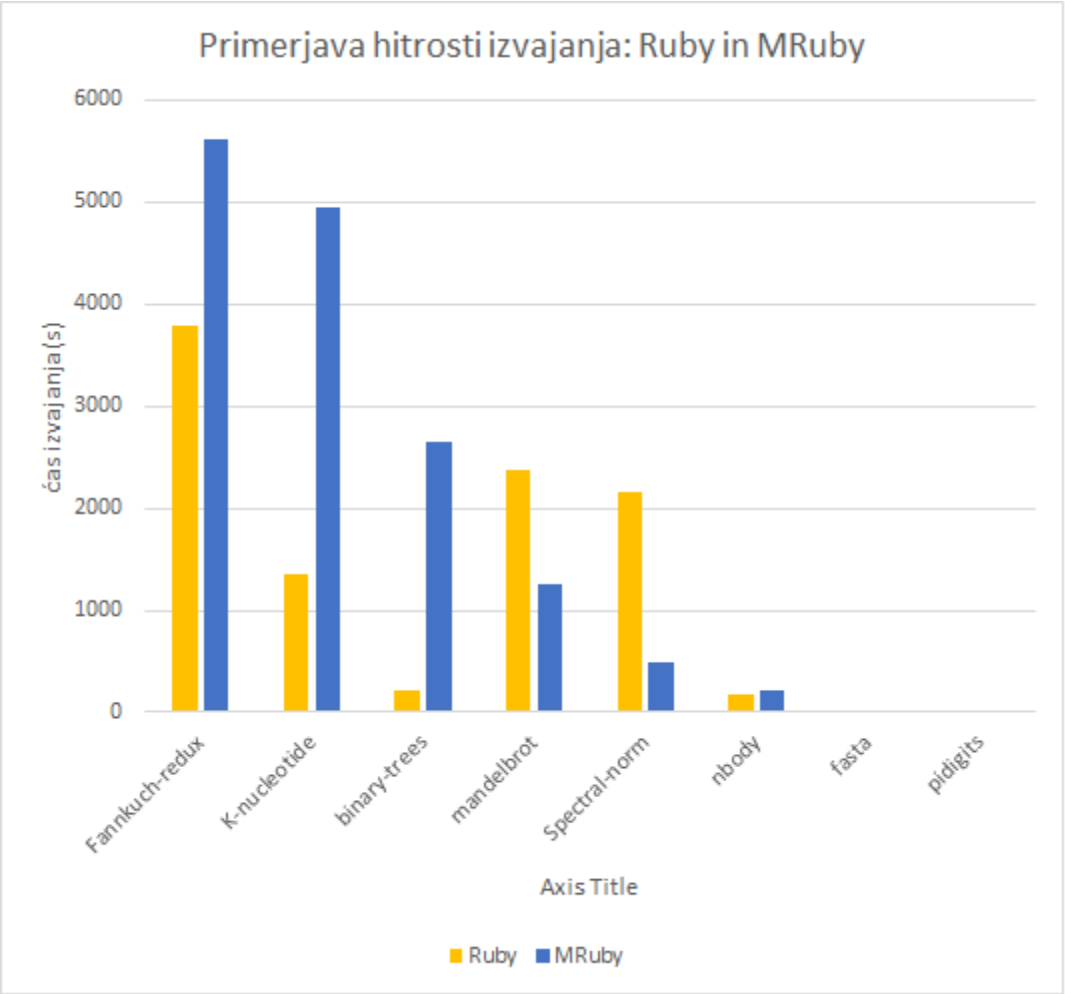
GRAF 1 - PRIMERJAVA HITROSTI IZVAJANJA C, C++

Ob pogledu na naslednji graf lahko razberemo, da je Python pri dalj časa trajajočih algoritmih velikokrat hitrejši, medtem ko sta si pri ostalih z Ruby-jem precej blizu.



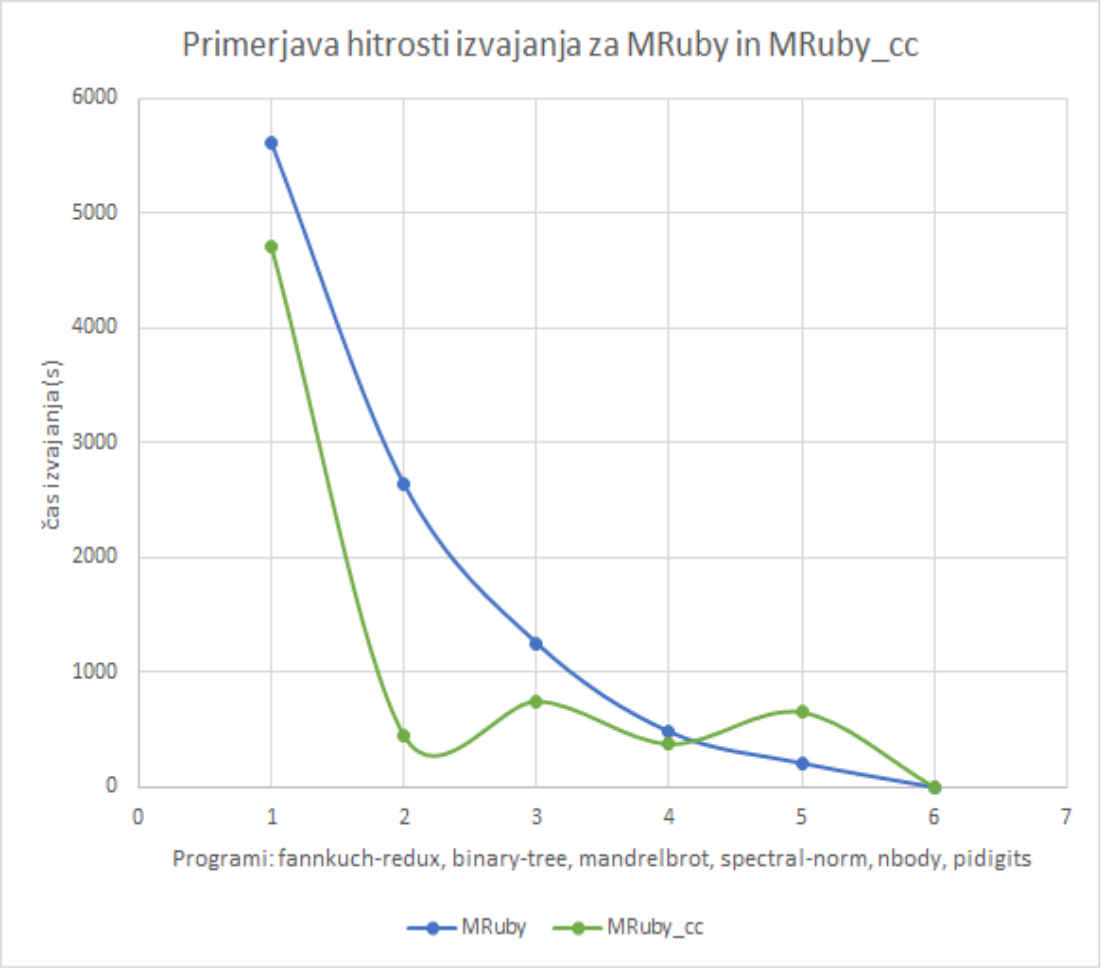
GRAF 2 - PRIMERJAVA HITROSTI IZVAJANJA PYTHON, RUBY

Kakršenkoli sklep o primerjavi hitrosti izvajanja med Ruby in MRuby bi bil verjetno brc v temo. V nekaterih primerih vodi prvi v nekaterih drugi. Nikakor pa ne moremo trditi, kateri je v splošnem hitrejši. Zaleži od algoritma oz. vrste dela, ki ga opravljata. Poudariti je potrebno tudi, da Ruby v primerjavi z MRuby-jem »stari maček«, medtem ko je MRuby praktično še prototip, ki je vse prej kot optimiziran. V sistemu GIT se zanj vsakodnevno vrstijo številni novi prispevki s popravki kode. Zato je lahko pri nekaterih operacijah zelo potraten, kar pa se pa pokaže v razgibanih rezultatih.



GRAF 3 - PRIMERJAVA HITROSTI IZVAJANJA RUBY, MRUBY

Zanimiva pa je primerjava MRuby z MRuby_cc, saj oba uporabljata enako izvorno kodo. Na spodnjem grafu pa lahko razberemo, da je koda prevedena z MRuby_cc v kar štirih primerih opazno hitrejša od MRuby-ja, kar je zelo dober znak.



GRAF 4 – PRIMERJAVA HITROSTI: MRUBY, MRUBY_CC

8. SKLEPNE UGOTOVITVE

V okviru diplomske naloge smo predstavili programski jezik MRuby in opisali prevajalnik MRuby_cc. Pogledali smo si kako poteka namestitev obeh in kako se ju uporablja. Omenili smo tudi nekatere pozitivne in negativne lastnosti, ki jih imata in obravnavali številne napake, ki so se vrstile pri poizkusih zagona testnih programov.

Kot smo že nekajkrat omenili, bi rad ponovno izpostavil, da je programski jezik MRuby zelo mlad z veliko prostora za optimizacijo. Zanj imamo na voljo zelo omejene vire dokumentacije in ostalega gradiva. Prevajalnik MRuby_cc je prav tako še prototip, vendar kot smo videli, že prva verzija kaže presenetljive in zanimive rezultate. Da bi lahko z večjo gotovostjo diskutirali o hitrosti bi definitivno morali obdelati oz. usposobiti nekaj dodatnih testnih primerov. Morda en dan to postane predmet dodatnih raziskav na tem področju.

Ker se verzija MRuby-ja dnevno spreminja, bi bil prvi korak MRuby-ju dodati vse manjkajoče funkcionalnosti iz Ruby-ja in počakati na stabilno verzijo oz. predloge in popravke iz skupnosti. To verzijo bi nato umestili tudi v MRuby_cc in kodo ponovno prilagodili, tako kot je za prejšnjo verzijo to že storil doc. dr. Boštjan Slivnik. Morda poiskati dodatne možnosti optimizacije.

Vredno je omeniti tudi, da je vzdrževanje takega projekta precej kompleksna zadeva, saj lahko že en prispevek v GIT povzroči nekoliko preglavic, pomislimo sedaj na grmado prispevkov ali celo kritične spremembe v zasnovi, ki se vrstijo iz dneva v dan. Potrebno bi bilo tudi odpraviti GEM pakete za osnovne funkcije in nato poizkusiti prevesti še ostale nedelujoče programe. Ker smo delali z že uveljavljenimi algoritmi, bi pri večji količini primerkov lahko podali tehtne rezultate.

9. ZAKLJUČEK

Moja velika želja skozi diplomsko delo je bila, da bi ob koncu dobili predstavo o hitrosti prevajalnika MRuby_cc. Na poti sem se srečal s številnimi težavami, ki sem jih po svojih zmožnostih poizkusil odpraviti, nekatere bolj, druge manj uspešno. Iz popolne teme, kjer se niti eden program ni uspešno prevedel pod MRuby_cc, smo z odprtimi očmi zagledali luč, ki jo predstavlja teh nekaj delujočih testnih programov. MRuby_cc ima vsekakor potencial, ki ga moramo le podpreti.

SEZNAM SLIK

VIRI

- [1] Spletna stran z interaktivno MRuby lupino, dostopno na: http://joshnuss.github.io/mruby-web-irb/ (dostop september 2013)
- [2] Spletna stran s testnimi primeri, dostopno na: http://benchmarksgame.alioth.debian.org/ (dostop september 2013)
- [3] Podstran testnega programa k-nucleotide, dostopno na: http://benchmarksgame.alioth.debian.org/u32/performance.php?test=knucleotide (dostop september 2013)
- [4] http://stellar.cct.lsu.edu/2012/01/solving-n-body-problem-using-hpx/ (dostop september 2013)
- [5] http://benchmarksgame.alioth.debian.org/u32/performance.php?test=meteor (dostop oktober 2013)
- [6] http://freepages.genealogy.rootsweb.ancestry.com/~langolier/BerryDNA/dna_recloh.html (dostop november 2013)
- [7] http://en.wikipedia.org/wiki/Mandelbrot_set (dostop november 2013)
- [8] http://codegolf.stackexchange.com/questions/3105/generate-a-mandelbrot-fractal (dostop november 2013)
- [9] http://www.nilesjohnson.net/teaching/pi.png (dostop december 2013)
- [10] http://commons.wikimedia.org/wiki/File:Binary_tree.png (dostop december 2013)
- https://github.com/mruby/mruby (dostop september 2013)
- http://matt.aimonetti.net/posts/2012/04/25/getting-started-with-mruby/ (dostop september 2013)
- http://www.youtube.com/watch?feature=player_embedded&v=sB-IifjyeLI (dostop september 2013)
- http://c2.com/cgi/wiki?PythonVsRuby (dostop september 2013)
- http://www.senktec.com/2013/06/ruby-vs-python/ (dostop september 2013)
- http://www.ruby-lang.org/en/documentation/ruby-from-other-languages/to-ruby-from-python/ (dostop september 2013)
- https://github.com/mruby/mruby/tree/master/doc/compile (dostop oktober 2013)
- http://en.wikipedia.org/wiki/N-body_problem (dostop oktober 2013)
- http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=09561EA252CE8F783AD0813DD408F1DA?doi=10.1.1.35.5124&rep=rep1&type=pdf (dostop november 2013)
- https://www.ruby-lang.org/en/about/ (dostop november 2013)

- http://stackoverflow.com/questions/125367/dynamic-type-languages-versus-static-type-languages (dostop november 2013)
- http://www.jvoegele.com/software/langcomp.html (dostop november 2013)
- http://en.wikipedia.org/wiki/FASTA (dostop december 2013)
- http://www.python.org/about/ (dostop december 2013)
- https://www.ruby-lang.org/en/ (dostop december 2013)
- http://regebro.wordpress.com/2009/07/12/python-vs-ruby/ (dostop december 2013)
- http://en.wikipedia.org/wiki/Sequence_alignment (dostop december 2013)
- http://en.wikipedia.org/wiki/Matrix_norm (dostop januar 2014)
- https://www.ruby-lang.org/en/news/2013/02/24/ruby-2-0-0-p0-is-released/ (dostop januar 2014)
- https://www.ruby-lang.org/en/about/ (dostop januar 2014)
- http://blog.mruby.sh/201205231844.html (dostop januar 2014)
- http://matt.aimonetti.net/posts/2012/04/20/mruby-and-mobiruby/ (dostop januar 2014)
- http://blog.mruby.sh/201207020720.html (dostop januar 2014)

PRILOGA A: SKRIPTE ZA PREVAJANJE

N-BODY

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -mfpmath=sse -
msse3 nbody.c -o nbody.gcc-4.gcc_run -lm

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native -mfpmath=sse -
msse3 --std=c++11 nbody.cpp -o nbody.gpp-3.c++.o && /usr/bin/g++ nbody.gpp-
3.c++.o -o nbody.gpp-3.gpp_run -fopenmp

# mruby_cc
/usr/local/bin/mrbcc nbody_cc.rb -o nbody_cc
```

Izpis A.1 - datoteka *compile_all.sh* za prevajanje izvorne kode za C, C++ in MRuby_cc

FANNKUCH-REDUX

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -falign-
labels=8 fann.c -o fannkuchredux.gcc-4.gcc_run

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native  -pthread fann.cpp
-o fannkuchredux.gpp-4.c++.o &&  /usr/bin/g++ fannkuchredux.gpp-4.c++.o -o
fannkuchredux.gpp-4.gpp_run -lpthread -lboost_thread

# mruby_cc
/usr/local/bin/mrbcc fann_cc.rb -o fann_cc
```

Izpis A.2 - datoteka *compile_all.sh* za prevajanje izvorne kode za C, C++ in MRuby_cc

METEOR-CONTEST

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native  meteor.c -o
meteor.gcc_run

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native  meteor.cpp -o
meteor.gpp-6.c++.o && /usr/bin/g++ meteor.gpp-6.c++.o -o meteor.gpp-6.gpp_run
```

Izpis A.3 - datoteka *compile_all.sh* za prevajanje izvorne kode za C in C++.

FASTA

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -std=c99 -
mfpmath=sse -msse3 fasta.c -o fasta.gcc-4.gcc_run

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native -mfpmath=sse -
msse3 --std=c++11 fasta.cpp -o fasta.gpp-2.c++.o && /usr/bin/g++ fasta.gpp-
2.c++.o -o fasta.gpp-2.gpp_run

# mruby_cc
/usr/local/bin/mrbcc fasta_cc.rb -o fasta_cc
```

Izpis A.4 - datoteka *compile_all.sh* za prevajanje izvorne kode za C, C++ in MRuby_cc

SPECTRAL-NORM

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -fopenmp -
mfpmath=sse -msse2 spec.c -o spectralnorm.gcc-5.gcc_run -lm

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native -mfpmath=sse -
msse2 -fopenmp -O0 spec.cpp -o spectralnorm.gpp-5.c++.o && /usr/bin/g++
spectralnorm.gpp-5.c++.o -o spectralnorm.gpp-5.gpp_run -fopenmp

# mruby_cc
/usr/local/bin/mrbcc spec_cc.rb -o spec_cc # ne potrebujemo
```

Izpis A.5 - datoteka *compile_all.sh* za prevajanje izvorne kode za C, C++ in MRuby_cc

REVERSE-COMPLEMENT

```
#c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -std=c99 -
pthread reverse.c -o revcomp.gcc-2.gcc_run

#cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native  -fopenmp -
mfpmath=sse -msse2 reverse.cpp -o revcomp.gpp-4.c++.o && /usr/bin/g++
revcomp.gpp-4.c++.o -o revcomp.gpp-4.gpp_run -fopenmp
```

Izpis A.6 - datoteka *compile_all.sh* za prevajanje izvorne kode za C in C++.

MANDELBROT

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -mfpmath=sse -
msse2 -fopenmp mandel.c -o mandelbrot.gcc-4.gcc_run

# c++
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native -mfpmath=sse -
msse2 -fopenmp -mfpmath=sse -msse2 mandel.cpp -o mandelbrot.gpp-9.c++.o &&
/usr/bin/g++ mandelbrot.gpp-9.c++.o -o mandelbrot.gpp-9.gpp_run -fopenmp

# mruby_cc
/usr/local/bin/mrbcc mandel_cc.rb -o mandel_cc
```

Izpis A.7 - datoteka *compile_all.sh* za prevajanje izvorne kode za C, C++ in MRuby_cc

K-NUCLEOTIDE

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -pthread -
std=c99 -include simple_hash3.h nucl.c -o knucleotide.gcc-6.gcc_run

# c++
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native  -std=c++0x
nucl.cpp -o knucleotide.gpp-3.c++.o && /usr/bin/g++ knucleotide.gpp-3.c++.o -o
knucleotide.gpp-3.gpp_run -lpthread
```

Izpis A.8 - datoteka *compile_all.sh* za prevajanje izvorne kode za C in C++.

PIDIGITS

```
# predpogoji
aptitude install libgmp3-dev python-gmpy

# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native pidi.c -o
pidigits.gcc_run -lgmp

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native   pidi.cpp -o
pidigits.gpp-3.c++.o && /usr/bin/g++ pidigits.gpp-3.c++.o -o pidigits.gpp-
3.gpp_run -lgmp -lgmpxx

# mruby_cc
/usr/local/bin/mrbcc pidi_cc.rb -o pidi_cc
```

Izpis A.10 – datoteka *compile_all.sh* za prevajanje C, C++ in MRuby_cc. (za nekatere od jezikov, potrebujemo za uspešno gradnjo objektne datoteke še nekatere druge knjižnice.

REGEX-DNA

```
# predpogoji
aptitude install libgtk2.0-dev tcl8.5-dev

# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -pthread -
I/usr/include/tcl8.5 `pkg-config --cflags --libs glib-2.0`regex.c -o
regexdna.gcc_run -ltcl8.5 -lglib-2.0

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native  -fopenmp
regex.cpp -o regexdna.gpp-4.c++.o && /usr/bin/g++ regexdna.gpp-4.c++.o -o
regexdna.gpp-4.gpp_run -fopenmp
```

Izpis A.9 – datoteka *compile_all.sh* za prevajanje C, C++ in MRuby_cc. (za nekatere od jezikov, potrebujemo za uspešno gradnjo objektne datoteke še nekatere druge knjižnice.

CHAMENEOX-REDUX

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -pthread cham.c
-o chameneosredux.gcc-5.gcc_run

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native cham.cpp -o
chameneosredux.c++.o &&  /usr/bin/g++ chameneosredux.c++.o -o
chameneosredux.gpp_run -lboost_thread -lboost_system
```

Izpis A.11 - datoteka *compile_all.sh* za prevajanje izvorne kode za C in C++.

THREAD-RING

```
# c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -pthread ring.c
-o threadring.gcc-2.gcc_run

# cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native ring.cpp -o
threadring.gpp-5.c++.o && /usr/bin/g++ threadring.gpp-5.c++.o -o
threadring.gpp-5.gpp_run -lboost_system -lpthread
```

Izpis A.12 - datoteka *compile_all.sh* za prevajanje izvorne kode za C in C++.

```
# predpogoji
aptitude install libapr1-dev libboost-dev-all

#c
/usr/bin/gcc -pipe -Wall -O3 -fomit-frame-pointer -march=native -fopenmp -
D_FILE_OFFSET_BITS=64 -I/usr/include/apr-1.0 binary.c -o binarytrees.gcc-
7.gcc_run -lapr-1 -lgomp -lm

#cpp
/usr/bin/g++ -c -pipe -O3 -fomit-frame-pointer -march=native  -fopenmp
binary.cpp -o binarytrees.gpp-6.c++.o && /usr/bin/g++ binarytrees.gpp-6.c++.o
-o binarytrees.gpp-6.gpp_run -fopenmp

# mruby_cc
/usr/local/bin/mrbcc binary_cc.rb -o binary_cc
```

Izpis A.8 - datoteka *compile_all.sh* za prevajanje izvorne kode za C, C++ in MRuby_cc.