Original article:
http://dan.corlan.net/bench.html

Programski jezik mjerila

Ova stranica sadrži isti program, sprovodi na isti način, u C, Ada, FORTRAN, Lisp, dalje, Java, Perl, R i Ruby i pokrenuti na 300MHz Pentium pod "Woody" Debian GNU/Linux, koristeći isključivo alata koji dolaze sa Debian distribucije, osim za sastavljanje Java sam koristio jikes kompajler iz IBM-a.
Ovo je relevantna za aplikacije koje uključuju intenzivne proračune za više korisnika, kao što su simulatori s otvorenim web pristup (kao što suwarfarissimo i autovaca). Za takve aplikacije, performanse prevodi direktno u troškove: broj mašina trebate kupiti i održavati kako bi se služiti dati broj korisnika ili da se postigne potrebno vrijeme odziva je direktno proporcionalna vremenu računanja.
Program smo mjerilo izračunava na isti 100-term polinom 500.000 puta, koristeći isti algoritam. U svim programima se čuvaju indeksi polinoma u lokalnoj vektor float. U ovom, program testira samo kvalitet koda koji pristupa lokalne vektora i obavlja jednostavne aritmetiku u petlje, i bez razlike u standardnoj biblioteci, operativni sistem poziva i, zaista, prisustvo bilo koje napredne mogućnosti jezika.
Postoje dvije verzije programa: (1) svi proračuni se obavljaju unutar jednog tijela funkcije; (2) svaki polinom se izračunava putem poziva na funkciju; vektor od 100 koeficijenata se čuvaju u statički dimenzionirani niz unutar ove funkcije; to se zove 500000 puta.
Brzina prve verzije izvršenja je također istražen kroz mjerila na AMD 64×2.
Rezultati su ispod (u sekundama potrebno za obavljanje jednostavan zadatak gore navedene). Kliknite na link s imenom jezikom vidjeti detalje (program i kako da sastavi i Run) za svaki jezik:

Jezik	jedinstveno tijelo (s)	s call(s)
FORTRAN, g77 V2.95.4	2.73	2.73
Ada 95, gnat V3.13p	2.73	2.74
C, hand optimized **, gcc V2.95.4	2.73	
Java, gcj V3.0	3.03	15.53
D, gcc V4.0.3+	1)3.43	1)3.98
C, gcc V2.95.4	3.61	3.57
R translated to lisp korišćenje R2cl v0.1 i sastavio sa cmucl	3.69	
Lisp, CMU Common Lisp V3.0.8 18c+, build 3030	4.69	10.69
Java, jikes V1.15 (bytecompiled)	8.23	13.54
FORTH, hand optimized ** Gforth 0.6.1	1)18.21	
FORTH,** Gforth 0.6.1	1)27.26	
Python** +psyco (interpreted)	1)168.50	
Perl, more optimized$ V5.6.1 (natively compiled)	209.20	
Perl, more optimized$ V5.6.1 (interpreted)	258.64	

Perl, hand optimized*** V5.6.1 (bytecompiled)	306.18
Perl* V5.6.1 (natively compiled)	367.23
Python** V2.1.2 (interpreted)	505.50
Perl* V5.6.1 (bytecompiled)	515.04
RUBY*** (interpreted)	1074.52
R V1.5.1 (interpreted)	5662.64

* Doprinijela Matei Conovici
** Doprinosi Mihai Manolescu
*** Doprinosi zgrim
$ Doprinosi Radu Greab
+ Doprinosi Mihai Militaru
1) procijenjena

- naivna (bez ruku-optimizaciju) gcc-sastavio C je oko 30% sporiji od Fortran i ada, inače nijedna nije pogođena otkriti iznad prilikom dodjele lokalni vektor u funkciji;
- je gcj kompajler radi dobar posao, povećavajući brzinu koda više od dva puta; Međutim, dodjela lokalne varijable (na `pol 'niz na primjer) poništava sve korist pribavljena native kompiliranje;
- performanse CMU-CL Lisp kompajler je Stellar; to je bio bolji od Java, čak i sastavio prirodno, uprkos vrlo dinamične prirode; od 10.69 sekundi, oko 6,5 su za odvoz smeća.
- bytecompiled Perl kod je porediti u brzini sa tumači R koda i nije puno sporiji od Perl koda sastavio nativno; primijetiti činjenicu da je u R koristimo vectorial operatora koji također čine tekst vrlo kompaktan.

Diskusiju. Relevantnost tih testova počiva u činjenici da u svakom jeziku, performanse upravlja koliko brzo operacije, petlje, pristup elemenata niza, pozivi funkcija, itd, se izvode. Machine-orijentisanih jezika (C, FORTRAN, Ada i donekle Java) obavlja u prosjeku 100 brže od ljudskog izražavanja orijentiranih jezika (Perl, R, Python, Ruby) puta.
Programiranje 1 GHz Pentium [najbrže, $ 1000, 2002 PC] u Perl je kao programiranja 10 MHz nešto [overclockana, 50 $, 1982 Z80-bazirane ZX Spectrum] u FORTRAN!
Ovo nije veliko iznenađenje što gledamo klasične trade-off između moć jezika (omogućava programeru da izrazi nešto u kompaktnom način) i performace za implementaciju (koja se odnosi na klasično jezik bude blizu zastupljenosti mašinu). Međutim, ogromni izuzetak je CommonLisp. Lisp je najmoćniji jezik koji je, predstavlja klasičnu ekstremne izbor za izražajne snage umjesto efikasnu implementaciju.
Ovaj izbor čini se da se isplatilo. S obzirom dovoljno izraz snage bilo je moguće napisati kompajler koji konkurira bilo kojem jeziku rublja orijentisan.
Zaključci (za mene):

- Za nove projekte u kojima je jedan slobodan da izabere jezik, treba odabrati Lisp.
Primjer je na mreži generacija web stavove (kao što je ova) kompleksnih, heterogene baze podataka u kojima postoje dva uslova: brza implementacija novih ideja i sklonost za ažuriranje statički web stranice za 10 minuta, a od 15 sati.
- Ako efikasnost je vrlo kritičan i samo ukoliko je zahtjev mora biti multithreaded, treba odabrati Ada. Multithreaded, online simulacije servere, poput onih gore navedenih, tipični su primjeri.
- Drugim jezicima, s kojim bi se moglo biti veoma poznat zbog istorijskih razloga, mogla postati efikasna jezika se prevodi u CommonLisp i sastavio sa CMU-CL.

Dodatak. Detalji svakog programa i kako da sastavi i pokrenite ga:
FORTRAN
tespol.f

```
      program tespol
      dimension pol(100)
      real pol
      integer i,j,n
      real su,pu,mu
      real x

      n = 500000
      x = 0.2
```

```
      mu = 10.0
      pu = 0.0
      do i = 1,n
         do j=1,100
            mu = (mu + 2.0) / 2.0
            pol(j) = mu
         enddo
         su = 0.0
         do j=1,100
            su = x * su + pol(j)
         enddo
         pu = pu + su
      enddo
      write (*,*) pu
      end
```

Sastaviti i pokrenuti sa:

```
f77 tespol.f -O6 -o tespol
time ./tespol
```

tespol2.f

```
      function dopoly(x)
      real x
      real su,mu
      integer j
      dimension pol(100)
      real pol

         do j=1,100
            mu = (mu + 2.0) / 2.0
            pol(j) = mu
         enddo
         su = 0.0
         do j=1,100
            su = x * su + pol(j)
         enddo

      dopoly = su
      end

      program tespol

      integer i
      real pu
      real x


      n = 500000
      x = 0.2
      mu = 10.0
      pu = 0.0
      do i = 1,n
         pu = pu + dopoly(x)
      enddo
      write (*,*) pu
      end
```

Sastaviti i pokrenuti sa:

```
f77 tespol2.f -O6 -o tespol2
time ./tespol2
```

Ada-95

tpol.adb

```
with Ada.Command_Line; use Ada.Command_Line;
with Ada.Text_Io; use Ada.Text_Io;

procedure Tpol is
```

```
   Pol: array(1..100) of Float;
   N: Integer:= Integer'Value(Argument(1));
   X: Float:= Float'Value(Argument(2));
   S: Float;
   Mu: Float:= 10.0;
   Pu: Float:= 0.0;

begin
   for I in 1..N loop
      for J in 1..100 loop
         Mu := (Mu + 2.0) / 2.0;
         Pol(J) := Mu;
      end loop;
      S := 0.0;
      for J in 1..100 loop
         S := X * S + Pol(J);
      end loop;
      Pu := Pu+S;
   end loop;
   Put_Line(Float'Image(Pu));
end Tpol;
```

Sastaviti i pokrenuti sa:

```
gnatmake -O6 tpol
time ./tpol 500000 0.2
```

tpol2.adb

```
with Ada.Command_Line; use Ada.Command_Line;
with Ada.Text_Io; use Ada.Text_Io;

procedure Tpol2 is

   N: Integer:= Integer'Value(Argument(1));
   X: Float:= Float'Value(Argument(2));
   Pu: Float:= 0.0;

   function Dopol(X: Float) return Float is

      Pol: array(1..100) of Float;
      S: Float;
      Mu: Float:= 10.0;

   begin
      for J in 1..100 loop
         Mu := (Mu + 2.0) / 2.0;
         Pol(J) := Mu;
      end loop;
      S := 0.0;
      for J in 1..100 loop
         S := X * S + Pol(J);
      end loop;
      return S;
   end Dopol;

begin
   for I in 1..N loop
      Pu := Pu+Dopol(X);
   end loop;
   Put_Line(Float'Image(Pu));
end Tpol2;
```

Sastaviti i pokrenuti sa:

```
gnatmake -O6 tpol2
time ./tpol2 500000 0.2
```

Java

```
public class tpoly {

    static public void main(String argv[]) {
     float mu = (float)10.0;
     float x,s;
     float pu = (float)0.0;
     int su, i, j, n;
     float pol[] = new float[100];

     n = 500000;
     x = (float)0.2;
     for(i=0; i<n; i++) {
       for (j=0; j<100; j++) {
         mu =  (mu + (float)2.0) / (float)2.0;
         pol[j] = mu;
       }
       s = (float)0.0;
       for (j=0; j<100; j++) {
         s = x*s + pol[j];
       }
       pu += s;
     }
     System.out.println(pu);
  }
}
```

Sastaviti i pokrenuti sa:

```
gcj-3.0 --main=tpoly --classpath=/usr/share/java/libgcj.jar -O2 -o tpoly
tpoly.java
time ./tpoly
```

or compile to bytecode and run with:

```
jikes -O tpoly.java
time java tpoly
public class tpoly2 {

    static float dopoly(float x) {
        float pol[] = new float[100];
        int j;
        float mu = (float)10.0;
        float s;

        for (j=0; j<100; j++) {
            mu =  (mu + (float)2.0) / (float)2.0;
            pol[j] = mu;
        }
        s = (float)0.0;
        for (j=0; j<100; j++) {
            s = x*s + pol[j];
        }
        return s;

    }

    static public void main(String argv[]) {
     float x;
     float pu = (float)0.0;
     int i, n;

     n = 500000;
     x = (float)0.2;
     for(i=0; i<n; i++) {
       pu += dopoly(x);
     }
```

```
     System.out.println(pu);
  }
}
```

Sastaviti i pokrenuti sa:

```
gcj-3.0 --main=tpoly --classpath=/usr/share/java/libgcj.jar -O2 -o tpoly2
tpoly2.java
time ./tpoly2
```

or compile to bytecode and run with:

```
jikes -O tpoly2.java
time java tpoly2
```

C

tepol.c

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

main(short argc, char **argv) {
  float mu = 10.0;
  float x,s;
  float pu = 0.0;
  int su, i, j, n;
  float pol[100];

  n = atol(argv[1]);
  x = atof(argv[2]);
  for(i=0; i<n; i++) {
    for (j=0; j<100; j++) {
      pol[j] = mu = (mu + 2.0) / 2.0;
    }
    s = 0.0;
    for (j=0; j<100; j++) {
      s = x*s + pol[j];
    }
    pu += s;
  }
  printf("%f\n",pu);
}
```

Sastaviti i pokrenuti sa:

```
gcc -O6 tepol.c -o tepol
time ./tepol 500000 0.2
```

tepol2.c

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

float dopoly(float x) {

  float mu = 10.0;
  float s;
  int j;
  float pol[100];

  for (j=0; j<100; j++) {
    pol[j] = mu = (mu + 2.0) / 2.0;
  }
  s = 0.0;
  for (j=0; j<100; j++) {
    s = x*s + pol[j];
  }
  return s;
}

main(short argc, char **argv) {
  float x;
  float pu = 0.0;
```

```
  int  i, n;

  n = atol(argv[1]);
  x = atof(argv[2]);
  for(i=0; i<n; i++) {
    pu += dopoly(x);
  }
  printf("%f\n",pu);
}
```

Sastaviti i pokrenuti sa:

```
gcc -O6 tepol2.c -o tepol2
time ./tepol2 500000 0.2
```

Hand optimised C

bench1.c (by Mihai Manolescu)

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

main(short argc, char **argv) {
  float mu = 10.0;
  float x,s;
  float pu = 0.0;
  int su, i, j, n;
  float pol[100];
  register tm1;
  register int tp1;

  n = atol(argv[1]);
  x = atof(argv[2]);
  for(i=0; i<n; i++) {
    tp1=2;
    tm1=1/2.0;
    for (j=0; j<100; j++) {
      pol[j] = mu = (mu + tp1) * tm1;
    }
    s = 0.0;
    for (j=0; j<100; j++) {
      s = x*s + pol[j];
    }
    pu += s;
  }
  printf("%f\n",pu);
}
```

Common-Lisp

testpol.lisp

```
(defun eval-pol (n x)
  (declare (fixnum n) (single-float x))
  (let ((su 0.0) (mu 10.0) (pu 0.0)
        (pol (make-array 100 :element-type 'single-float)))
    (declare (single-float su) (single-float mu) (single-float pu))
    (dotimes (i n)
             (declare (fixnum i))
             (setf su 0.0)
             (dotimes (j 100)
                      (declare (fixnum j))
                      (setf mu (the single-float (/ (+ mu 2.0) 2.0)))
                      (setf (aref pol j)
                            (the single-float mu)))
             (dotimes (j 100)
                      (declare (fixnum j))
                      (setf su (the single-float
                                    (+ (aref pol j) (the single-float (* su
x))))))
             (setf pu (the single-float (+ pu su)))
```

```
            )
    (prin1 pu)
    ))
```

Sastaviti i pokrenuti sa:

```
lisp
```

… then load, compile and run with:

```
(load "testpol.lisp")
(compile #'eval-pol)
(time (eval-pol 500000 0.2))
```

testpol2.lisp

```
(proclaim '(optimize))

(declaim (start-block dopoly eval-pol))

  (defun dopoly (x)
    (declare (ftype (function (single-float) single-float) dopoly))
    (declare (single-float x))
    (let ((su 0.0) (mu 10.0)
          (pol (make-array 100 :element-type 'single-float)))
      (declare (single-float su) (single-float mu))
      (dotimes (j 100)
        (declare (fixnum j))
        (setf mu (the single-float (/ (the single-float (+ mu 2.0)) 2.0)))
        (setf (aref pol j)
              (the single-float mu)))
      (dotimes (j 100)
        (declare (fixnum j))
        (setf su (the single-float
                      (+ (aref pol j) (the single-float (* su x))))))
      su)
    )


(defun eval-pol (n x)
  (declare (fixnum n) (single-float x))
  (let ((pu 0.0))
    (declare (single-float pu))
    (dotimes (i n)
             (declare (fixnum i))
             (setf pu (the single-float (+ pu (the single-float (dopoly x)))))
             )
    (prin1 pu)
    ))

(declaim (end-block))
```

Sastaviti i pokrenuti sa:

```
lisp
```

… then load, compile and run with:

```
(load "testpol2.lisp")
(compile #'eval-pol)
(time (eval-pol 500000 0.2))
```

D

tepol.d (by Mihai Militaru)

```
//tepol.d

import std.stdio;
import std.conv;

int main(char[][] args)
{
        float mu = 10.0;
        float x,s;
        float pu = 0.0;
```

```
        int su, i, j, n;
        float pol[100];

        n = toLong(args[1]);
        x = toFloat(args[2]);
        for(i=0; i<n; i++)
        {
                for (j=0; j<100; j++)
        {
                pol[j] = mu = (mu + 2.0) / 2.0;
        }
        s = 0.0;
        for (j=0; j%lt;100; j++)
        {
                s = x*s + pol[j];
        }
                pu += s;
        }
        writefln("%f\n",pu);
        return 0;
}
```

Prevesti s:

```
gdmd -O -release -inline tepol.d
```

Run with:

```
time ./tepol 500000 0.2
```

tepol2.d (by Mihai Militaru)

```
//tepol2.d

import std.stdio;
import std.conv;

float dopoly(float x)
{
        float mu = 10.0;
        float s;
        int j;
        float pol[100];

        for (j=0; j<100; j++)
        {
                pol[j] = mu = (mu + 2.0) / 2.0;
        }
        s = 0.0;
        for (j=0; j<100; j++)
        {
                s = x*s + pol[j];
        }
        return s;
}

int main(char[][] args)
{
        float x;
        float pu = 0.0;
        int  i, n;

        n = toLong(args[1]);
        x = toFloat(args[2]);
        for(i=0; i<n; i++)
        {
        pu += dopoly(x);
        }
        printf("%f\n",pu);
```

```
        return 0;
}
```

Prevesti s:

```
gdmd -O -release -inline tepol2.d
```

Run with:

```
time ./tepol2 500000 0.2
```

Python

tpytpol.py (by Mihai Manolescu)

```
n = 500000
x = 0.2

def t(x):
    mu = 10.0
    pu = 0.0
    pol =[0] * 100
    r = range(0,100)

    for i in range(0,n):
        for j in r:
            pol[j] = mu = (mu + 2.0) / 2.0
        su = 0.0
        for j in r:
            su = x * su + pol[j]
        pu = pu + su
    print pu

t(x)
```

Trčanje s:

```
time python tpytpol.py
```

Perl

tperlpol.pl (by Matei Conovici)

```
#! /usr/bin/perl


$n = $ARGV[0];
$x = $ARGV[1];

$mu = 10;
$pu = 0;

@pol = ();

for ($i = 0; $i < $n; $i++) {
        for ($j = 0; $j < 100; $j++) {
                $mu = ($mu + 2) / 2;

                $pol[$j] = $mu;
        }

        $s = 0;
        for ($j = 0; $j < 100; $j++) {
                $s = $x * $s + $pol[$j];
        }

        $pu += $s;
}

print "$pu\n";
```

Sastaviti i pokrenuti sa:

```
perlcc -O -o tperlpol tperlpol.pl
time ./tperlpol 500000 0.2
```

or, for byte compilation:

```
perlcc -B -o tperlpol tperlpol.pl
time ./tperlpol 500000 0.2
```

Perl (hand optimized)

pl.pl (by zgrim)

```
my($n,$x,$mu,$pu,@pol)=(500000,0.2,10,0,(0..99));
for(1..$n) {
    for (0..$#pol)  {   $pol[$_] = $mu = ( $mu + 2 ) * 0.5      }
    $s = 0;
    for(0..$#pol)   {   $s = $x * $s + $pol[$_]                 }
    $pu += $s
}
print qq[$pu\n];
```

Sastaviti i pokrenuti sa:

```
perlcc -B -o pl pl.pl
time ./pl
```

Perl (hand optimized for native compilation)

perl3.pl by Radu Greab

```
#!/usr/bin/perl -w

use strict;

my $n = $ARGV[0];
my $x = $ARGV[1];

my $mu = 10;
my $pu = 0;

my @pol;

foreach (0 .. $n - 1) {
    foreach (0 .. 99) {
        $pol[$_] = $mu = ($mu + 2) / 2;
    }

    my $s = 0;
    foreach (0 .. 99) {
        $s = $x * $s + $pol[$_];
    }

    $pu += $s;
}

print "$pu\n";
```

Sastaviti i pokrenuti sa:

```
perlcc -O -o perl3 perl3.pl
time ./perl3
```

R (or S, Splus)

trpol2.R

```
trpol2 <- function(n,x) {
  mu <<- 10.0
  pu <<- 0.0
  pol <<- 1:100
  tp1 <<- 2.0
  tm1 <<- 1/2.0
  for (i in 1:n) {
    for (j in 1:100) {
      mu <<- (mu + tp1) * tm1
      pol[j] <<- mu
    }
    s <<- 0.0;
    for (j in 1:100) {
```

```
      s <<- x*s + pol[j];
    }
    pu <- s+pu;
  }
  print(pu)
}
```

Izvršiti program uz:

```
time echo " source(\"trpol2.R\") ; trpol2(500000,0.2) ; q() " | R --no-save
```

Ruby (by `zgrim`)

pol.rb

```
n = 500000
x = 0.2
mu = 10
pu = 0
pol = []

n.times do
    0.upto(99) { |j| pol[j] = mu = (mu + 2) * 0.5 }
    s = 0
    0.upto(99) { |j| s = x * s + pol[j] }
    pu += s
end

print pu,"\n"
```

Izvršiti program uz:

```
time ruby pol.rb
```

FORTH (by Mihai Manolescu)

bench.fs

```
--- bench.fs:

\ gforth izvorni kod za test brzine
\                                                       mmihai, sept '04
\

100 floats allocate throw
constant farray

: init_farray
    0e0
    farray
    100 0 do
        dup fdup f!
        float+
    loop
    drop fdrop
;

: my_loop
    init_farray
    0e0                 \ x pu
    10e0                \ x pu mu

    0 do
        farray
        100 0 do
            2e0 f+ f2/
            dup fdup f!
            float+
        loop
        drop                    \ x pu mu
        frot frot               \ mu x pu
        fswap 0e0               \ mu pu x su
        farray
```

```
        100 0 do
            fover  f*
            dup f@ f+
            float+
        loop
        drop
        frot f+                 \ mu x pu
        frot                    \ x pu mu
    loop
    fdrop f. cr fdrop
;

0.2e0 500000 my_loop
```

Izvršiti program uz:
`time gforth-fast bench.fs -e bye`
Dalje, ruke optimizirana (Mihai Manolescu)
bench2.fs
\ Gforth izvorni kod za test brzine
\ Mmihai, septembar ’04
\
falign ovdje 100 plovcima izdvojiti konstantan farray
: my_loop
0e0 \ x PU
10e0 \ x PU MU
0 do
farray
2E0 fswap
100 0 Do
fover F + fover f /
dup fdup F!
float +
petlja
fnip
drop \ x PU MU
Frot Frot \ MU x PU
fswap 0e0 \ mu PU x su
farray
100 0 Do
dup
fover F *
F @ F +
float +
petlja
ispustiti
Frot F + \ MU x PU
Frot \ x PU MU
petlja
fdrop F. CR fdrop
;
0.2e0 500000 my_loop
Izvršiti program uz:
Vrijeme gforth-brzo bench2.fs -e bye