

Speedtest PicoLisp vs Emacs Lisp

Thorsten Jolitz
tj@data-driven.de

2012-04-23

Speedtest PicoLisp vs Emacs Lisp

The Tests

Function Call/Arithmetic Cost

Shell Script Approach

The classic Fibonacci function was used for measuring function call/arithmetic cost.

Here is the PicoLisp script:

```
#!/usr/bin/picolisp
(de fibo (N)
  (if (> 2 N)
    1
    (+ (fibo (dec N)) (fibo (- N 2))) ) )
(fibo 35)
(bye)
```

Here is the (uncompiled) Emacs Lisp script:

```
#!/usr/bin/emacs --script

(defun fibo (N)
  (if (> 2 N)
    1
    (+ (fibo (1- N)) (fibo (- N 2))) ) )

(fibo 35)
```

Here is the script that calls a byte-compiled Emacs Lisp file with the above function definition and call:

```
#!/bin/sh
":"; exec emacs --no-site-file --script "/home/tj/shellscripts/tj-fibo-compiled.elc" # --emacs-lisp
```

The following shell command was used to measure the performance:

```
[tj@arch ] timescript
```

with *script* being one of the three scripts above.

Command Line Approach

This is an alternative, more elegant and efficient way to run the tests. Just produce these two files:

```
$ cat > fibo.el << .
(defun fibo (N)
  (if (> 2 N)
      1
      (+ (fibo (1- N)) (fibo (- N 2))) ) )

(fibo 35)
.
```

```
$ cat > fibo.l << .
(de fibo (N)
  (if (> 2 N)
      1
      (+ (fibo (dec N)) (fibo (- N 2))) ) )
(fibo 35)
.
```

Then byte-compile fibo.el and run the following commands:

```
$ time emacs --no-site-file --script fibo.el
$ time emacs --no-site-file --script fibo.elc
$ time pil fibo.l -bye
```

As a side note: Emacs can be invoked noninteractively from the shell to do byte compilation with the aid of the function batch-byte-compile. In this case, the files to be compiled are specified with command-line arguments. Use a shell command of the form

```
emacs -batch -f batch-byte-compile files...
```

for example

```
$ emacs --no-site-file -batch -f batch-byte-compile fibo.el
```

List Manipulation Cost

The costs of list manipulation were tested with the “extensive list manipulations” code from Alexander Burger:

```
$ cat > tst.l << .
(de tst ()
  (mapcar
    (quote (X)
      (cons
        (car X)
        (reverse (delete (car X) (cdr X)))) ) )
    '((a b c a b c) (b c d b c d) (c d e c d e) (d e f d e f)) ) )
(do 1000000 (tst))
.
```

```
$ cat > tst.el << .
(defun tst ()
  (mapcar
   (lambda (X)
     (cons
      (car X)
      (reverse (delete (car X) (cdr X))) ) )
   '( (a b c a b c) (b c d b c d) (c d e c d e) (d e f d e f) ) )
  (dotimes (i 1000000) (tst)))
.
```

Results

32bit

System Information

```
$ uname -a
Linux icz 2.6.32-5-686 #1 SMP Mon Jan 16 16:04:25 UTC 2012 i686
GNU/Linux
$ cat /proc/cpuinfo |grep "model name" | cut -d: -f2
Pentium(R) Dual-Core CPU          T4200  @ 2.00GHz
Pentium(R) Dual-Core CPU          T4200  @ 2.00GHz
```

Function Calls

These are the results for running fibo (N) with N=35:

PicoLisp	0m5.662s	1x	
Elisp	0m13.854s	ca 2.5x	
Elisp (compiled)	0m5.882s	ca 1x	

PicoLisp is 2.5x faster than interpreted Emacs Lisp and as fast as compiled Emacs Lisp.

List Manipulation

These are the results for running tst with (do 1000000 (tst) or (dotimes (i 1000000) (tst))):

PicoLisp	0m1.208s	1x	
Elisp	0m8.311s	ca 7x	
Elisp (compiled)	0m5.622s	ca 4.5x	

PicoLisp is 7x faster than interpreted Emacs Lisp and 4.5x faster than compiled Emacs Lisp. Looks like the Emacs compiler can't improve much in that function and it's still 4.6-6.9x slower than PicoLisp.

64bit

System Information

```
$ uname -a
```

```
Linux arch 3.3.2-1-ARCH #1 SMP PREEMPT Sat Apr 14 09:48:37 CEST 2012
x86_64 AMD Athlon(tm) 64 X2 Dual Core Processor 5000+ AuthenticAMD
GNU/Linux
$ cat /proc/cpuinfo |grep "model name" | cut -d: -f2
AMD Athlon(tm) 64 X2 Dual Core Processor 5000+
AMD Athlon(tm) 64 X2 Dual Core Processor 5000+
```

Function Calls

These are the results for running fibo (N) with N=35:

```
| PicoLisp          | 0m3.191s | 1x   |
| Elisp             | 0m12.731s | ca 4x |
| Elisp (compiled) | 0m6.635s | ca 2x |
```

PicoLisp is 4x faster than interpreted Emacs Lisp and 2x faster than compiled Emacs Lisp.

These are the results for running fibo (N) with N=40:

```
| PicoLisp          | 0m35.982s | 1x   |
| Elisp             | 2m14.352s | ca 4x |
| Elisp (compiled) | 1m13.304s | ca 2x |
```

Again PicoLisp is ca. 2x faster than compiled Elisp and 4x faster than interpreted Elisp.

List Manipulation

These are the results for running tst with (do 1000000 (tst) or (dotimes (i 1000000) (tst))):

```
| PicoLisp          | 0m1.635s | 1x   |
| Elisp             | 0m9.582s | ca 6x |
| Elisp (compiled) | 0m7.129s | ca 4.5x |
```

PicoLisp is 6x faster than interpreted Emacs Lisp and 4.5x faster than compiled Emacs Lisp.

Just to remind you - PicoLisp is always interpreted, but the interpreter is designed with the need for speed¹.

32bit vs 64bit

All other things equal, 64-bit PicoLisp is usually slower than the 32-bit version, due to a poorer memory cache performance (the cells are twice as large size). On the other hand, arithmetics are faster, due to the additional short number type in pil64.

¹<http://picolisp.com/5000/!wiki?needforspeed>