EFFICIENCY OF FORTRAN AND COMMON LISP

ALEXANDRU DAN CORLAN

Misconceptions persist regarding the execution speed of lisp code compared to classic compiled languages such as FORTRAN.

Due to the dynamic nature of Lisp, in early implementations, when compilers using proper type declarations were not yet deployed, a reputation of "slowness" emerged for Lisp.

Around 40 years ago, very efficient compilers were developed that compile code as fast as FORTRAN, if the program is properly specified, that is, if the types of variables and functions are restricted to integers, floats, characters and arrays of them.

Table 1. Efficiency of compiled code via f2cl translation, followed by sbcl Common Lisp compilation, vs direct compilation of FORTRAN programs with gfortran on two recent microprocessors.

processor	version		tespol exec (ms)		compile+run (ms)	
	sbcl	gfortran	sbcl	gfortran	sbcl	gfortran
Ryzen 5 5600G	2.1.11	11.4.0	120	107	137	161
Xeon E3-1230V2	1.1.14	4.8.4	208	193	245	245

Lisp can be used as a dynamic language, that allocates and frees lots of memory while interpreting constructs that are known only at runtime.

However, this is just an optional feature. When and where this type of dynamic expressivity is not necessary, ubiquitous Common Lisp implementations, such as sbcl, compile properly written Lisp code with exactly the same efficiency as FORTRAN compilers.

Here, we reiterate the argument and also present a simple method through which any FORTRAN program can be easily tested against the Common Lisp equivalent, using the well known [2] developed by Broughan and Willcock about 30 years ago.

1. Results

About 20 years ago, we evaluated the same algorithm compiled with CMU Common Lisp (cmucl) and FORTRAN and the result was only about 50%

Date: January 3, 2026.

faster for FORTRAN [1]. In the meantime, SBCL, the succesor of cmucl, improved and despite simultaneous improvements in the FORTRAN compiler, this difference has been reduced to less than 10%.

The results are in table 1. One of the timings is for running the tested function, tespol, in lisp, and respectively for loading and running the gfortran generated binary. The other is for the whole process, loading the lisp environment, converting the program from FORTRAN, compiling and running it and, respectively compiling the program with gfortran and running it. For some unexplored reason, the whole FORTRAN process is somewhat slower.

An important point is that the common-lisp execution of tespol does not cons anything (as reported by the internal 'time' function), sa no garbage collection debt is created.

2. Methods

Unlike in the previous experiment [1], we did not write the Common Lisp code by hand. Instead, we start from the same FORTRAN source code, that is either compiled with gfortran or translated to lisp through f2cl [2] and compiled and run with sbcl.

To streamline this, process we built an executable, named bablisp, that combines sbcl, the f2cl translator and our previously published shelisp interface [3]. This is done by running the script shown in listing 1.

The embedding of FORTRAN programs in Common Lisp is shown in listing 2. f77 is the function previously defined in bablisp. The `#[ ... ]#` is a reader macro defined in shelisp that allows pieces of text to be given as strings without escapes for quotes and other characters. At read time, the text between these delimiters is converted into a simple string that is then provided as a first argument to function f77. f77 invokes f2cl to convert this string, as FORTRAN source, into the lisp function tespol, which is compiled.

Listing 3 shows the result of the compilation, the Common Lisp tespol function.

3. Run the test on another machine

The associated archive, clfortest.tgz, contains the script babltest.runme. This archive has a md5sum of `c3180d7f001842b66eefcf05823974c2`.

You must have gfortran, unzip and sbcl installed. On a debian based system, you can install them with a command like:

```
apt install gfortran sbcl unzip
```

The f2cl distribution is included in clfortest. Make a directory, cd into it, untar the archive with:

```
tar xvzf clfortest.tgz
```

then run: `./babltest.runme`

The first set of timings is for the function tespol, the next is for the overall compilation, loading and running.

Of course, the tespol benchmark can be replaced with any other FORTRAN program.

4. Conclusions

The difference in efficiency between native fortran compilers and, at least, the compiler from sbcl, have narrowed to less than 10% in the last couple of decades.

Transpiling other languages, both with static and dynamic data structures, into Common Lisp is an increasingly appealing approach to increasing speed and reducing power consumption of computation.

References

[1] Programming language benchmarks. Alexandru Dan Corlan et al, 2002. http://dan.corlan.net/bench.html

[2] FORTRAN to Lisp Translation using f2cl K.A. Broughan, D.M.K. Willcock, Software Practice and Experience, 26(10), p. 1127–1139, 1996 https://doi.org/10.1002/(SICI)1097-024X(199610)26:10<1127::AID-SPE50>3.0.CO;2-Q

[3] SHELISP: Unix shell commands from Common Lisp. Alexandru Dan Corlan 2006. http://dan.corlan.net/shelisp/

Listing 1. bablispgen, a simple script that loads shelisp v3.2, the f2cl translator, defines a simpler compilation function (f77) and saves everything as an executable named bablisp. The executable has about 47MB and uses libm, libc, libz and libpthreads.

```
#! /usr/bin/sbcl --script
;;;; BABel LISP. a framework to use other syntaxes in lisp
;;;; Copyright (c) 2022 Alexandru Dan Corlan MD PhD
;;;;
;;;; This program loads and compiles other programs, then generates
;;;; a standalone executable as an enhanced sbcl.
;;;;
;;;; v0.1. January 8, 2022. Using shelisp and f2cl to embed FORTRAN in lisp

(load "shelisp32.lisp")

(load "f2cl-master/src/f2cl0.l")
(load "f2cl-master/src/f2cl1.l")
(load "f2cl-master/src/f2cl2.l")
(load "f2cl-master/src/f2cl3.l")
(load "f2cl-master/src/f2cl4.l")
(load "f2cl-master/src/f2cl5.l")
(load "f2cl-master/src/f2cl6.l")
(load "f2cl-master/src/f2cl7.l")
(load "f2cl-master/src/f2cl8.l")
(load "f2cl-master/src/macros.l")

(defun f77 (source &key (tempfile #P"temp.f77")
                        (templisp #P"temp.lisp")
                        (keep-temp-files nil)
                        (declare-common nil))
  (with-open-file (file tempfile :direction :output
                                    :if-exists :overwrite
                                    :if-does-not-exist :create)
    (write-string source file)
    )
  (f2cl::f2cl tempfile :output-file templisp
              :declare-common declare-common)
  (load templisp)
  (unless keep-temp-files
    (delete-file tempfile)
    (delete-file templisp)
    )
  )

(save-lisp-and-die "bablisp" :executable t :purify t)
```

Listing 2. The lisp program that runs the test.

```
#! ./bablisp --script

(f77 #[

      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

]# :keep-temp-files t)

(time (tespol))
```

Listing 3. The bablispgen script that combines the standard Common Lisp environment with f2cl and shelisp to produce the bablisp executable.

```
;;; Compiled by f2cl version:
;;; ("f2cl1.l,v 95098eb54f13 2013/04/01 00:45:16 toy $"
;;;  "f2cl2.l,v 95098eb54f13 2013/04/01 00:45:16 toy $"
;;;  "f2cl3.l,v 96616d88fb7e 2008/02/22 22:19:34 rtoy $"
;;;  "f2cl4.l,v 96616d88fb7e 2008/02/22 22:19:34 rtoy $"
;;;  "f2cl5.l,v 95098eb54f13 2013/04/01 00:45:16 toy $"
;;;  "f2cl6.l,v 1d5cbacbb977 2008/08/24 00:56:27 rtoy $"
;;;  "macros.l,v 1409c1352feb 2013/03/24 20:44:50 toy $")

;;; Using Lisp SBCL 1.1.14.debian
;;;
;;; Options: ((:prune-labels nil) (:auto-save t) (:relaxed-array-decls t)
;;;           (:coerce-assigns :as-needed) (:array-type ':array)
;;;           (:array-slicing t) (:declare-common nil)
;;;           (:float-format single-float))

(in-package :common-lisp-user)


(defun tespol ()
  (prog ((x 0.0) (su 0.0) (pu 0.0) (mu 0.0) (i 0) (j 0) (n 0)
         (pol (make-array 100 :element-type 'single-float)))
    (declare (type (array single-float (100)) pol)
             (type (f2cl-lib:integer4) n j i)
             (type (single-float) mu pu su x))
    (setf n 500000)
    (setf x 0.2)
    (setf mu 10.0)
    (setf pu 0.0)
    (f2cl-lib:fdo (i 1 (f2cl-lib:int-add i 1))
                  ((> i n) nil)
      (tagbody
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j 100) nil)
          (tagbody
            (setf mu (/ (+ mu 2.0) 2.0))
            (setf (f2cl-lib:fref pol (j) ((1 100))) mu)
           label100001))
        (setf su 0.0)
        (f2cl-lib:fdo (j 1 (f2cl-lib:int-add j 1))
                      ((> j 100) nil)
          (tagbody
            (setf su (+ (* x su) (f2cl-lib:fref pol (j) ((1 100)))))
           label100002))
        (setf pu (+ pu su))
       label100000))
    (f2cl-lib:fformat t :list-directed pu)
   end_label
    (return nil)))

(in-package #-gcl #:cl-user #+gcl "CL-USER")
#+#.(cl:if (cl:find-package '#:f2cl) '(and) '(or))
(eval-when (:load-toplevel :compile-toplevel :execute)
  (setf (gethash 'fortran-to-lisp::tespol
                 fortran-to-lisp::*f2cl-function-info*)
          (fortran-to-lisp::make-f2cl-finfo :arg-types 'nil :return-values 'nil
                                            :calls 'nil)))
```