Lisp Implementations

Walter van Roggen
Digital Equipment Corporation
77 Reed Rd, Hudson, MA, 01749, HLO2-3/E9
vanroggen%bach.dec@decwrl.dec.com

There's a lot more to describing Lisp implementations than providing a list of features. One such aspect I'd like to gather information on is performance.

A long, long time ago in a place far, far away, someone named Richard P. Gabriel spent many, many years studying the performance of lisp systems. He gathered together some benchmarks and many results on different systems in "Performance and Evaluation of Lisp Systems" (MIT Press, 1985). Although the primary conclusion may well have been that analyzing lisp systems is at least as complex as the lisp system itself, and that trying to compare lisps by one or a few benchmark results is useful only if you understand what specific features are really being compared, the study was greatly useful in making the lisp community aware of the limitations of benchmarks and in giving us a common background of programs to consider during a performance evaluation.

But since the early 1980's, when that study was done, there have been many changes in the lisp community. There are many new implementations, and there are even new implementation technologies. Comparing the implementations and the results available then with those available now produces quite a contrast–and the impression that the lisp community is still in quite a flux.

So I'd like to ask all implementations to send me their most recent CPU and GC times for the Gabriel benchmarks. Please include the size of the heap used, which lisp and version of the lisp, which operating system and version of the operating system, and what hardware were used. Feel free to add correct declarations where appropriate, but please don't make any algorithmic changes to the code. I will publish the results in a future issue.

I cannot publish all the benchmarks here. If you need the sources, I can try to send them via netmail if you send me netmail I can reply to.

But I do include the source for one Gabriel benchmark, and a new benchmark which uses more of Common Lisp in what may be a slightly more realistic program.

The first one has been timed on many different lisp and non-lisp systems. It is probably the third most frequently tried function when trying out a new lisp system, after factorial of 100 and fibonacci of 20. It is highly recursive and primarily tests function calling.

```
;;; TAK
(defun tak (x y z)
  (declare (type (integer 0 19) x y z))
  (if (not (< y x))
      z
      (tak (tak (1- x) y z)
           (tak (1- y) z x)
           (tak (1- z) x y))))

;;; Time (tak 18 12 6)
```

The second one takes an input file, counts the number of times each word occurs, and writes out a file with those counts. The good thing about this benchmark is that it tests many normally untested aspects of Common Lisp, including characters, strings, sequences, hash-tables, closures, and I/O (particularly FORMAT). The bad thing about this benchmark is that it tests so many things, which makes it hard to determine where the time is really going. But it makes for another kind of "representative" benchmark.

```
;;; WORDS
;;; Count the frequency of words in a file.
;;; Walter van Roggen, 22 July 1983.

;;; (WORD-COUNT "file.txt" "freq.res") where "file.txt" is just text
;;; file and "freq.res" is the name of the result file to be written.
;;; The output is in "<word> = <# of occurrences>" pairs.
;;; A ''word'' is assumed to be any consecutive sequence of alphabetic
;;; characters or a single-quote character (to allow words like
;;; "don't"), or a hyphen.

(defun alpha-or-quote-p (ch)
  (declare (string-char ch))
  (or (alpha-char-p ch) (char= ch #\') (char= ch #\-)))

(defun word-count (infile outfile)
  (let ((hashtab (make-hash-table :test #'equal :size 1000))
        (total 0))
    (declare (hash-table hashtab) (fixnum total))
    ;; this hash tables holds the "words" we've found so far.
    (with-open-file (inf infile :direction :input)
      ;; read in each line, parse each "word", and increment count
      (do ((buf (read-line inf nil nil) (read-line inf nil nil)))
          ((null buf))
        (declare (simple-string buf))
        (setq buf (nstring-upcase buf))
        (do ((start (position-if #'alpha-or-quote-p buf :start 0)
                    (position-if #'alpha-or-quote-p buf :start end))
             end)
            ((null start))
          (declare (type (or fixnum null) start end))
          (setq end (or (position-if-not #'alpha-or-quote-p buf
                                         :start (the fixnum (1+ start)))
                        (length buf)))
          (incf total)
          (incf (the fixnum (gethash (subseq buf start end) hashtab 0)))))))
    (let ((words (let ((list ()))
                   (maphash #'(lambda (key val)
                                (push (cons key val) list))
                            hashtab)
                   list)))
```

```
      (princ "Sorting ... ")
      (setq words (sort words #'> :key #'cdr))
      (princ "Writing word list ... ")
      (with-open-file (outf outfile :direction :output :if-exists :new-version)
        (format outf "Word frequency count for ~A~%"
                (namestring (probe-file infile)))
        (format outf "~D different words in ~D total~%~%"
                (hash-table-count hashtab) total)
        (mapc #'(lambda (pair)
                  (format outf "~A = ~D~%" (car pair) (cdr pair)))
              words))))
  (namestring (probe-file outfile)))

;;; Time (word-count "boyer.lsp" "boyer.wc")
```

Of course to run this benchmark everyone has to have the same data file to read. The longest reasonable file I could think of that everyone running benchmarks would have is the source for the BOYER benchmark. Note that the source provided in "Performance and Evaluation of Lisp Systems" includes only one comment, at the end of the file. It also has some constructs which are not part of Common Lisp, so if you already have a running copy of the BOYER benchmark, there may be some minor differences in the number of words.

The resulting file should say the three most common words are:

```
X = 206
EQUAL = 151
Y = 117
```