ACM *Lisp Pointers* **V**,3 (July-Sept. 1992), 22-23.

A Tachy[1] 'TAK'

Henry G. Baker

Nimble Computer Corporation, 16231 Meadow Ridge Way, Encino, CA 91436
(818) 986-1436 (818) 986-1360 (FAX)

We show how to speed up the Tak Benchmark by an order of magnitude—5X faster than the Cray-1—on a Common Lisp system (40MHz 80860-based OKIstation) using memoizing. The list-based Takl Benchmark improves even more—30X faster than the Cray-1. Given the speed attainable through memoizing, the possibility of further speedups using parallelism seems unlikely.

A. INTRODUCTION

The Tak benchmark is John McCarthy's mis-remembered version of the Takeuchi function [Gabriel85]. The Tak benchmark is one of the most commonly used benchmarks because its reliance on only recursive function-calling and integer arithmetic allows it to be used early in hardware debugging, and because it is short enough to memorize and type surreptitiously into a competitor's computer at a trade show. While some benchmarks have been criticized for running "entirely within the cache", the Tak benchmark typically runs "entirely within the register set" of a RISC architecture, and therefore deserves a double dose of the same criticism. It is generally assumed that because of its ubiquity that Tak cannot be speeded up by non-intelligent means; we show that this assumption is erroneous.

We show that Tak can be speeded up by the technique of "memoization" [Bird80] [Keller86], which requires only that the function be "functional"—i.e., contain no side-effects. Since the lack of side-effects can often be statically assured at compile time by simple syntactic tests, a compiler could decide to utilize memoization for Tak as one of its standard optimizations.

B. STANDARD 'TAK'

According to [Gabriel85], the Tak benchmark contains 63,609 recursive calls to `tak`, as well as 47,706 decrement operations, when performed on the arguments `(18 12 6)` to produce the answer `7`. None of the arguments to `tak` ever becomes negative, nor does any ever exceed `18`. The first arm of the conditional is executed 75% of the time.

```
(defun tak (x y z)
  (if (not (< y x)) z
    (tak (tak (1- x) y z)
         (tak (1- y) z x)
         (tak (1- z) x y))))
```

C. MEMOIZING 'TAK'

A simple measurement shows that `tak` is called with only 281 distinct combinations of arguments, so memoization can work splendidly. However, in order to memoize, we must construct a single "key" from the triple of integers passed to `tak` as arguments. The Lispiest way to do this is to construct a Lisp list of the 3 arguments, and then use this as a key to a Common Lisp `equal` hash table, as in the following code:

```
(defparameter *memo-table* (make-hash-table :test #'equal)
  "Those who don't remember the past are condemned to recompute it"[2])

(defun make-key (x y z) `(,x ,y ,z))

(defun tak (x y z)
  (let ((key (make-key x y z)))[3]
    (or (gethash key *memo-table*)
        (setf (gethash key *memo-table*)
              (if (not (< y x)) z
                (tak (tak (1- x) y z)
                     (tak (1- y) z x)
                     (tak (1- z) x y)))))))
```

[1]Look it up in your *Funk&Wagnall's*.

[2]Apologies to Santayana.

[3]One might also utilize `&rest` arguments to construct the key list, as in `(defun tak (&rest key) ... )`.

This implementation works, and can already out-perform many standard `tak` implementations. It can be speeded up by the straight-forward technique of "hash consing" [Ershov58] [Goto74] [Deutsch73], which allows the `equal` hash table to be replaced by an `eq` hash table. But the fastest implementation utilizes the fact that the argument integers are bounded, and we can therefore *pack* them into a single fixnum:

```
(defparameter *memo-table* (make-hash-table :test #'eq))

(defun make-key (x y z) (+ (ash x 16) (ash y 8) z))
```

D. STANDARD 'TAKL'

The Gabriel Takl benchmark is obtained from the Tak benchmark by replacing integer counters with list counters; i.e., lists of length *n* are used to represent the integer *n*. Intuitively, one would presume that Takl would run a small factor slower than Tak, since list counters would appear to be only a small factor slower than fixnum counters (assuming that the lists are in the cache). However, it is much more difficult to implement the < predicate on lists than on fixnums; therefore, `shorterp` takes time proportional to the smaller of its arguments instead of taking only a small constant amount of time. On the standard benchmark versions, we find non-memoized Takl to be about 5.7X slower than non-memoized Tak.

E. MEMOIZING 'TAKL'

Memoizing Takl is slightly more difficult than memoizing Tak, because we cannot utilize packed integers as the keys to our memo table, but must construct unique keys using hash consing. However, our table still consists of only 281 active entries, so it will likely remain entirely within the cache.

In Takl, we actually have a choice about whether to memoize `mas`, `shorterp` or both. While memoizing `shorterp` should dramatically shorten its time, we would still execute the entire 63,609 number of calls to `mas`. If we memoize `mas`, then we are left with very few calls to `shorterp`, in which case its timing won't matter very much. Thus, it is only necessary to memoize `mas` to get most of the benefits of memoization.

F. RESULTS

The memoization optimization improves Tak by about an order of magnitude. We achieve a Tak time of 0.008 seconds on the 40Mhz 80860-based OKIstation™, which time is 5 X faster than the Cray-1 on the old benchmark.[4] By utilizing memoization with hash consing on the Takl benchmark, we achieve a Takl time of 0.01 seconds, which is 30 X faster than the Cray-1 on the old benchmark. Interestingly enough, Takl is only 25% slower than Tak when both are memoized; these numbers indicate that the memo table lookup dominates both computations.

G. REFERENCES

Anderson, J.Wayne, *et al*. "Implementing and Optimizing Lisp for the Cray". *IEEE Software* (July 1987),74-83.
Bird, R.S. "Tabulation Techniques for Recursive Programs". *ACM Comp. Surv. 12*,4 (Dec. 1980),403-417.
Deutsch, L. Peter. "An Interactive Program Verifier". Xerox PARC TR CSL-73-1, 1973.
Ershov, A.P. "On Programming of Arithmetic Operations". *Doklady, AN USSR 118*,3 (1958),427-430, transl. Friedman, M.D., *CACM 1*,8 (Aug. 1958),3-6.
Gabriel, R.P. *Performance and Evaluation of Lisp Systems*. MIT Press, Camb., MA, 1985.
Goto, Eiichi. "Monocopy and Associative Algorithms in Extended Lisp". TR. 74-03, U. Tokyo, 1974.
Keller, R.M., and Sleep, M.R. "Applicative Caching". *ACM TOPLAS 8*,1 (Jan. 1986),88-108.
Steele, Guy L. *Common Lisp, The Language; 2nd Ed*. Digital Press, Bedford, MA, 1990,1029p.

[4]We are using the "old" Cray-1 numbers from [Gabriel85]; newer numbers for the Cray-1 are given in [Anderson87].

