TIME: Where Did It Go?

Alan Graham

IBM Technical Computing
Dept 6FR/Bldg 32, mail drop 35A
1510 Page Mill Road,
Palo Alto, CA 94304
(415) 855-4465

Abstract

One of the most popular features of APL2 Release 3 is the performance analysis tool. It consists of a single external APL function, called *TIME*, that allows the user to extract relative timing information by function[1] or by line. Timing information may be extracted for selected functions or for all functions in the workspace. Monitoring may be selectively enabled or disabled.

This paper discusses the use of the *TIME* function and illustrates how cover functions can be used to customize and enhance the facility.

Introduction

In the course of application development, developers may need to do performance analysis. Often, performance problems appear in the form of an application that seems to run *too slowly* compared with the programmer's expectation. Are there critical sections of code that consume an inordinate amount of CPU time? In other words, what are the *hot spots*?

There have been several APL packages written that will monitor where the CPU time is used. PARAETO is one such package. Typically, such an APL timing package modifies your application to include code that looks at the accumulated CPU time (`1↑1↓⎕AI`) in between the execution of each line. This generally works quite well, but because your functions are modified, subtle differences may be introduced. For example, if cover functions are introduced and the function call stack is queried (`⍴⎕LC`) it will be deeper than expected. Also, there is always the fear that you will accidentally `)SAVE` the modified application over top of your original.

Performance monitoring is something the *system* can do without modifying the user's code. The system knows when a line begins execution and when it ends. In APL2 Release 2, the hooks were put in place for the system to keep statistics during execution of functions and to initialize, enable, and disable monitoring. The *TIME* function was built as an interface and after some experimenting, it was included as part of APL2 Release 3.

Overview

The *TIME* function is established in the workspace using `⎕NA`. Processor 11 has been extended in Release 3 to access APL objects outside of the active workspace. From the user's point of view *TIME* looks like any other locked function.

```
      3 11 ⎕NA 'TIME'
1
```

If you don't get a `1` result from `⎕NA` it is probably because you already have an object in the workspace named *TIME*, you are not running on APL2 *Release 3*, or there is a problem with the installation of APL2.

---

[1] Throughout this paper the term *function* is used to mean defined function or defined operator.

NL TIME 0 initializes the timing facility for the functions in the name list *NL*. The name list may be a simple string (naming one function), a vector of strings, or a simple character matrix. If no left argument is given, the initialization applies to all the functions in the workspace (including *TIME* itself).

Internally, a pair of counters are appended onto each line of every function being monitored and set to zero.[2] They are used to hold the CPU time and a count of the number of times a line is executed. The counters of a function are deleted if any change is made to the function by editing or *□FX* or if the function is transferred via *)OUT* or *)COPY*. For an external or locked function (such as *TIME* itself) the counters are appended only to the header.

NL TIME 1 returns a four-column matrix with one row for each function in the name list that has accumulated some time or has been called at least once, sorted in descending order by CPU time.[3] If no left argument is given, *TIME* 1 reports all functions that have have been called at least once.

The four columns are:

1. The number of times the function was called
2. CPU seconds the function accumulated (excluding subfunctions)
3. Percent of the total CPU time
4. Function name

On some very fast machines a function may execute so fast that a clock tick does not occur during its execution. Therefore, *TIME* may report a function with no accumulated accumulated time.

NL TIME 2 returns a five-column matrix with one row for each line of every function in the name list that has accumulated some time, sorted in descending order by CPU time. If no left argument is given, *TIME* 2 reports all functions that have used some time. The first four columns are the same as *TIME* 1 except that the fourth column is function name and line number. The fifth column is the line of code. For locked and external functions only a single row appears showing a summary for the entire function. (No Martha, you can't deduce details of a locked function by using *TIME*.)

NL TIME 3 returns a five-column matrix with one row for each line of every function in the name list, in ascending order by line within function. In other words, it is a function list with timing information. If no left argument is given, *TIME* 3 reports all functions that are being monitored.

NL TIME ¯3 deletes the timing information from the functions in the name list. If no left argument is given, *TIME* ¯3 deletes the timing information from all functions in the workspace. (Note: *TIME* ¯3 is how you clean up a workspace that has been accidentally :apl)SAVEd with the timing counters.)

TIME ¯2 disables the time monitor, but preserves the counters. A left argument is not allowed.

TIME ¯1 enables the time monitor after it has been disabled. A left argument is not allowed.

---

[2] *TIME* 0 is a mnemonic for *zero* the counters.

[3] *TIME* 1is a mnemonic for *one* row per function.

Sample Use

Below is a sample use of the timing facility against a function named *ON* that appends its arguments along the leading dimension. The workspace consists of the *ON* function and its subfunction *MAT*.[4]

```
     ∇
[0]    Z←Y ON X;⎕IO;N
[1]   ⍝ put left argument ON top of right argument
[2]    ⎕IO←0
[3]    Y←⍕MAT Y                    ⍝ character matrix
[4]    X←⍕MAT X                    ⍝ character matrix
[5]    N←(↑⌽⍴Y)⌈(↑⌽⍴X)             ⍝ more columns
[6]    Z←(N↑[1]Y),[0](N↑[1]X)      ⍝ attach vertically
     ∇ 1987-11-04 10.25.09 (GMT-8)

     ∇
[0]    Z←MAT X
[1]   ⍝ MATrix given any array
[2]    →(2=⍴⍴Z←X)/0                ⍝ escape if already matrix
[3]    Z←((×/¯1↓⍴X),¯1↑1,⌽⍴X)⍴X     ⍝ rows by columns
     ∇ 1987-11-04 10.23.54 (GMT-8)
```

To prepare for timing analysis bring in the application and the *TIME* function, initialize the counters, exclude the *TIME* function from being timed, set the print width and print precision, and accumulate timing information by running the application.

```
       )CLEAR                ⍝ start fresh
CLEAR WS

       )IN ON                ⍝ get application .

       3 11 ⎕NA 'TIME'       ⍝ get TIME function
1

       TIME 0                ⍝ zero counters

       'TIME' TIME ¯3        ⍝ don't time TIME

       ⎕PW←320               ⍝ no wrap please

       ⎕PP←4                 ⍝ limit detail

       ⍴Z←'TOP' ON (20 10⍴⍳100) ON 'BOTTOM'
22 30
```

To look at statistics on a function basis use *TIME* 1. The result is sorted in descending order by CPU time. Functions that are not called are not shown.

```
       ⍴⎕←TIME 1
 2 0.036 92.31  ON
 4 0.003  7.692 MAT
2 4
```

Now we want more detail. Since the application consists of only seven lines of executable code, it is reasonable to look at all of them. In a typical application, you'd use an expression such as *N*↑[⎕*IO*]*TIME* 2 where *N* is a small positive integer such as 15.

---

[4] For presentation here some of the displays are truncated on the right.

```
      TIME 2
2 0.029 74.36  ON[3]  Y←⍕MAT Y                    ⍝ char
2 0.002  5.128 MAT[3] Z←((×/¯1↓⍴X),¯1↑1,⍴X)⍴X    ⍝ row
2 0.002  5.128 ON[2]  ⎕IO←0
2 0.002  5.128 ON[5]  N←(↑⌽⍴Y)⌈(↑⌽⍴X)            ⍝ more
2 0.002  5.128 ON[6]  Z←(N↑[1]Y),[0](N↑[1]X)     ⍝ atta
4 0.001  2.564 MAT[2] →(2=⍴⍴Z←X)/0               ⍝ esc
2 0.001  2.564 ON[4]  X←⍕MAT X                   ⍝ char
4 0      0     MAT[0] Z←MAT X
2 0      0     ON[0]  Z←Y ON X;⎕IO;N
```

The single line *ON*[3] stands out as taking the majority of the CPU time. Full line comments are not shown because they do not accumulate any time. The time reported for *ON*[3] excludes the time taken for the call to *MAT*, which is detailed in other rows. Therefore, we can conclude that the majority of time is spent in the Format (⍕) primitive. Notice that *ON*[4] is the same line as *ON*[3] except for the right argument instead of the left argument, but uses far less time. In this sample run, a left argument Format is relatively expensive compared with a right argument Format. Formatting a simple integer matrix, although fairly fast, is much more expensive than formatting a character array (which is a no-op!).

```
      TIME 3
4 0      0     MAT[0] Z←MAT X
0 0      0     MAT[1] ⍝ MATrix given any array
4 0.001  2.564 MAT[2] →(2=⍴⍴Z←X)/0               ⍝ esc
2 0.002  5.128 MAT[3] Z←((×/¯1↓⍴X),¯1↑1,⍴X)⍴X    ⍝ row
2 0      0     ON[0]  Z←Y ON X;⎕IO;N
0 0      0     ON[1]  ⍝ put left argument ON top of
2 0.002  5.128 ON[2]  ⎕IO←0
2 0.029 74.36  ON[3]  Y←⍕MAT Y                    ⍝ char
2 0.001  2.564 ON[4]  X←⍕MAT X                   ⍝ char
2 0.002  5.128 ON[5]  N←(↑⌽⍴Y)⌈(↑⌽⍴X)            ⍝ more
2 0.002  5.128 ON[6]  Z←(N↑[1]Y),[0](N↑[1]X)     ⍝ atta
```

Listing all lines of both functions, we notice that prologue comments not only consume zero time, but actually never execute! The APL2 interpreter begins execution on the first non-comment line of a function.

TOP: A Simple Set of TIME Cover Functions

Although the *TIME* function is fairly easy to use directly, it is provided as more of a tool than an end-user report function. The most common problem is getting **too much** information. Executing *TIME* 2 on an application of 1,000 lines of code will produce a matrix of up to 1,000 rows! The CPU times and percents will display with up to ⎕*PP* (usually 10) digits of precision, although typically only four digits are significant. Cover functions can be built to take this matrix and select only the top few slow functions or lines.

TOP is a simple set of four cover functions that allow an application's *hot spots* to be quickly discovered and neatly displayed with titles and summaries. The functions are shown in the appendix.

Initialize and run the *CROSS* application.

```
      )CLEAR          ⍝ start fresh
CLEAR WS
      )IN CROSS       ⍝ get application
      )IN TOP         ⍝ get TIME, TOP, etc.
      TIME 0
      CROSS 'CROSS'   ⍝ run application
```

Show all functions called.

```
      TOP FNS
COUNT   TIME PERCENT PROGRAM
   86 1.231   43.78 IDENTS
   97  .460   16.36 ON
   13  .399   14.19 NAMES
   14  .307   10.92 CROSS
   15  .149    5.30 DETAIL
   28  .113    4.02 MEMBER
   27  .098    3.49 ROWS
   13  .049    1.74 ΔSS
    1  .004     .14 CROSS
    2  .002     .07 UNQUOTE
  296 2.812  100.00
```

Slowest four functions and the slowest four lines.

```
      TOP 4 FNS
COUNT   TIME PERCENT PROGRAM
   86 1.231   43.78 IDENTS
   97  .460   16.36 ON
   13  .399   14.19 NAMES
   14  .307   10.92 CROSS
  210 2.397   85.24

      TOP 4 LNS
COUNT   TIME PERCENT PROGRAM     LINE
   86  .300   10.67 IDENTS[16] Z←(⍴Λ)⍴(,Λ←Λ∘.≥⍳0⌈⌈/Λ
   84  .252    8.96 NAMES[15]  ZΔ←(∨⌿<\ZΔ∧.=⍉ZΔ)⌿ZΔ←
   15  .212    7.54 CROSS[15]  □←QV,PE,((PE MEMBER I
   86  .168    5.97 IDENTS[6]  QS←(⍬(-Q)⍬∧\(Q←¯1+(⍬B
  271  .932   33.14
```

Percent of total time.

```
      TOP 33 PERCENT LNS
COUNT   TIME PERCENT PROGRAM     LINE
   86  .300   10.67 IDENTS[16] Z←(⍴Λ)⍴(,Λ←Λ∘.≥⍳0⌈⌈/Λ
   84  .252    8.96 NAMES[15]  ZΔ←(∨⌿<\ZΔ∧.=⍉ZΔ)⌿ZΔ←
   15  .212    7.54 CROSS[15]  □←QV,PE,((PE MEMBER I
   86  .168    5.97 IDENTS[6]  QS←(⍬(-Q)⍬∧\(Q←¯1+(⍬B
  271  .932   33.14

      TOP 50 PERCENT FNS
COUNT   TIME PERCENT PROGRAM
   86 1.231   43.78 IDENTS
   97  .460   16.36 ON
  183 1.691   60.14
```

Conclusions

APL2 Release 3 includes a powerful time monitor tool, the *TIME* function. It can be used either directly or with cover functions to find an application's *hot spots*. It is not uncommon to get performance improvements of 50% after modifying one or two lines of code found to be particularly CPU intensive.

References

1. IBM Corporation, *APL2 Programming: Using the Supplied Routines*, SH20-9233

2. IBM Corporation, *APL2 Programming: System Services Reference*, SH20-9218

TOP functions

Report TOP (Slowest) Functions

```
     ∇
[0]    TOP X;⎕IO;⎕PW;N;H;F;T
[1]   ⍝ TOP (slowest) few lines or fns (or ops)
[2]   ⍝ syntax:  TOP [n|n PERCENT] (LNS|FNS)
[3]   ⍝ example: TOP FNS                ⍝ all functions
[4]   ⍝ example: TOP 5 LNS              ⍝ slowest 5 lines
[5]   ⍝ example: TOP 50 PERCENT FNS     ⍝ fns up thru 50%
[6]   ⍝ attributes: 0 1 0 1 ⎕FX ⎕CR 'TOP'
[7]    ⎕IO←0                            ⍝ zero origin
[8]    ⎕PW←320                          ⍝ don't wrap display
[9]    →(2=ρρX)/L1                      ⍝ just timing matrix?
[10]   (N X)←X                          ⍝ row-count, matrix
[11]   X←(N⌊↑ρX)↑[0]X                   ⍝ top N rows
[12]  L1:                               ⍝ format with header/totals
[13]   H←(↑1↓ρX)↑'COUNT' ' TIME' 'PERCENT' 'PROGRAM' 'LINE'
[14]   F←(2×ρH)↑0 0 0 3 0 2             ⍝ Format vector
[15]   T←(ρH)↑(+⌿3↑[1]X),'' ''          ⍝ Totals
[16]   ⎕←F⍕H,[0]X,[0]T                  ⍝ format and display
     ∇ 1987-10-30 16.34.06 (GMT-8)
```

Timing Matrix by Lines or Functions

```
     ∇
[0]    Z¯←LNS;US¯
[1]   ⍝ LiNeS with non-zero times or counts
[2]    US¯←'TOP' 'LNS' 'FNS' 'PERCENT' 'TIME'    ⍝ my fns
[3]    →0ρUS¯ TIME ¯3                ⍝ don't monitor us
[4]    Z¯←TIME 2                     ⍝ all lines monitored
     ∇ 1987-10-30 15.47.00 (GMT-8)

     ∇
[0]    Z¯←FNS;US¯
[1]   ⍝ FNS (and ops) with non-zero times or counts
[2]    US¯←'TOP' 'LNS' 'FNS' 'PERCENT' 'TIME'    ⍝ my fns
[3]    →0ρUS¯ TIME ¯3                ⍝ don't monitor us
[4]    Z¯←TIME 1                     ⍝ all fns and ops monitored
     ∇ 1987-10-30 15.48.51 (GMT-8)
```

Selects N Percent of Total Time

```
     ∇
[0]    Z←N PERCENT X;⎕IO
[1]   ⍝ select N percent of the slowest from timing matrix
[2]   ⍝ attributes: 0 1 0 0 ⎕FX ⎕CR 'PERCENT'
[3]    ⎕IO←0                              ⍝ zero origin
[4]    ⎕ES(0=⎕NC 'N')/5 1                 ⍝ must be dyadic
[5]    Z←((↑ρX)⌊1++/N>+\X[;2])↑[0]X       ⍝ slowest thru N percent
     ∇ 1987-10-30 16.34.18 (GMT-8)
```