

Where does X spend its time?

A small Forth profiler

EuroForth 2021
Philip Zembrod - pzembrod@gmail.com

Motivation

- cc64 compiler written in ITC VolksForth on C64 felt slow ...
- ... unreasonably slow ...
- Solution: optimize hotspots ...
- ... which were unknown

Where did cc64 spend its time?

Wish for a profiler that works

- at different levels - module group, module, word group, word
- ideally self-hosted

Prior art & tips

- timing individual words
- let NEXT log all words to stdout
- a C-written VM could be instrumented
- per-word e2e time tracking
 - <https://sourceforge.net/p/forth-brainless/code/HEAD/tree/trunk/profiler.fs>
 - gross time rather than net time

No clear fit for my problem ...

... there seemed to be an opportunity^wexcuse for a new tool. :-)

What could I do with NEXT?

- count invocations of a word
- count NEXT cycles within a word
 - # of IP fetches at addresses between : and ;
- count NEXT cycles and sum up time within a word
- count NEXT cycles and sum up time within a range of words
- split cc64 code into N ranges aka buckets
 - count NEXT cycles and sum up time per bucket
- split a bucket into N sub-buckets, rinse & repeat

... this could fly ...

Some details

- NEXT should remain fast
 - only single-interval buckets
 - only 8 buckets -> 3-cmp binary search
 - unrolled loop
- What about Forth core code?
 - core NEXT cycles & time added to calling bucket
- What about non-core code outside buckets?
 - default bucket 0 collects rest of NEXT cycles & time
- Time measurement
 - 2 cascaded 16-bit timers (MOS 6526 CIA) running at CPU clock

```
: compareIp
  IP 1+ lda >buckets[ ,x cmp 0= ?[ IP lda <buckets[ ,x cmp ]? ;

: findBucket
  0 # ldx compareIp CC ?[
    currentBucket ldx
  ][ inx compareIp CC ?[
    dex
  ][
    5 # ldx
    compareIp 0<> ?[ CC ?[ dex dex ][ inx inx ]?
    compareIp 0<> ?[ CC ?[ dex ][ inx ]?
    compareIp CC ?[ dex ]? ]? ]?

  IP 1+ lda >]buckets ,x cmp 0= ?[ IP lda <]buckets ,x cmp ]?
    CS ?[ 0 # ldx ]?

  txa .a asl .a asl tax
  ]?
  currentBucket stx
  ]? ;
```

```

Label prNext
  timerActrl lda pha $fe # and timerActrl sta
  calcTime
  findBucket
  incCountOfBucket
  addTimeToBucket
  setPrevTime
  incMainCount
  pla timerActrl sta
  0 # ldx clc IP lda Next $c + jmp

```

```

Code install-prNext
prNext 0 $100 m/mod
  # lda Next $b + sta
  # lda Next $a + sta
$4C # lda Next $9 + sta
Next jmp end-code

```

Buckets defined inline in code

```

\prof profiler-bucket [input]
include input.fth
\prof [input] end-bucket

```

- Easy & straightforward
 - for both start & end of bucket
- Only defined in instrumented mode

```

\prof profiler-bucket [scanner]
include scanner.fth
\prof [scanner] end-bucket

```

Alternative considered:

- define bucket with `word`
 - end of bucket less intuitive
 - difficult with headerless words

```

\prof profiler-bucket [syntab]
include symboltable.fth
include preprocessor.fth
\prof [syntab] end-bucket

```

Nested buckets for drilling down

```

\prof profiler-bucket [scanner-nextword]
\prof profiler-bucket [scanner-fetchword]

: fetchword ( -- tokenvalue token ) BEGIN (nextword is-comment? WHILE
  2drop skip-comment REPEAT \ ." fetchword: " 2dup u. u. word' 2! ;

: accept ( -- ) 1 word# +! fetchword ;

\prof [scanner-fetchword] end-bucket
\prof profiler-bucket [scanner-thisword]

: thisword ( -- tokenvalue token ) word' 2@ ;

\prof [scanner-thisword] end-bucket
\prof [scanner-nextword] end-bucket

```

Up to 8 active buckets

Buckets grouped in metrics

```
\prof include prof-metrics.fth
```

prof-metrics.fth:

```
profiler-metric:[ profile-cc64
  [strings]
  [memman-etc]
  [file-handling]
  [input]
  [scanner]
  [symtab]
  [parser]
  [pass2]
]profiler-metric
```

```
profiler-metric:[ profile-scanner
  [scanner-alphanum]
  [scanner-identifier]
  [scanner-operator]
  [scanner-char/string]
  [scanner-(nextword)]
  [scanner-comment]
  [scanner-nextword]
  [scanner-rest]
]profiler-metric
```

```
profiler-metric:[
profile-scanner-nextword
  [scanner-nextword-vars]
  [scanner-fetchword]
  [scanner-thisword]
  [scanner-nextword-mark]
  [scanner-nextword-advanced?]
]profiler-metric
```

Design overview

- Hooks into NEXT routine
- Max 8 buckets active per measurement
 - e.g. per instrumented e2 test run
 - 16 or 32 buckets also feasible
- Define arbitrary # buckets inline
- Metrics: bundles of max 8 buckets
 - defined in separate central file
- A metric, invoked interactively, activates its buckets
 - same compiled turn-key binary can run different metrics

Result #1: Don't list source during compile

```
profiler report PROFILE-CC64-1
timestamps
919.522.732 1.078.906.060
```

```
buckets
b# nextcounts clockticks name
0 475419 52822277 (etc)
1 1037243 114416784 [MEMMAN]
2 1384162 153154008 [FILE-HDL]
3 797224 122822197 [INPUT]
4 2695157 299076306 [SCANNER]
5 153639 17403250 [SYMTAB]
6 1826434 197679185 [PARSER]
7 1100491 121509788 [PASS2]
8 0 0 [SHELL]
```

```
profiler report PROFILE-CC64-1
timestamps
830.786.988 989.908.172
```

```
buckets
b# nextcounts clockticks name
0 475419 52854657 (etc)
1 1035458 114210370 [MEMMAN]
2 1384162 153117590 [FILE-HDL]
3 313708 34373231 [INPUT]
4 2695157 299094501 [SCANNER]
5 153639 17396511 [SYMTAB]
6 1826434 197594285 [PARSER]
7 1100491 121235385 [PASS2]
8 0 0 [SHELL]
```

Result #2: Eliminate loops in operator scanning

profiler report **PROFILE-SCANNER2**
timestamps
831.180.204 **990.563.532**

buckets			
b#	nextcounts	clockticks	name
0	6291642	691659740	(etc)
1	247516	29308255	[ALPHANUM]
2	66366	7040651	[ID]
3	1075233	116814924	[OPERATOR]
4	27320	3487498	[NUMBER]
5	237738	26065205	[CHR/STR]
6	150400	15851841	[NEXTWORD]
7	43874	5189985	[COMMENT]
8	844379	95121698	[REST]

profiler report PROFILE-SCANNER2
timestamps
725.077.164 **884.722.893**

buckets			
b#	nextcounts	clockticks	name
0	6294398	692597293	(etc)
1	247516	29374038	[ALPHANUM]
2	66366	7035251	[ID]
3	93760	10011918	[OPERATOR]
4	27320	3483433	[NUMBER]
5	237738	26075738	[CHR/STR]
6	150400	15872668	[NEXTWORD]
7	43874	5198813	[COMMENT]
8	844379	95104859	[REST]

Result #3: nextword/backword -> thisword/accept

profiler report **PROFILE-SCANNER3**
timestamps
725.011.628 **884.395.213**

buckets			
b#	nextcounts	clockticks	name
0	6322534	695897987	(etc)
1	247516	29341777	[ALPHANUM]
2	66366	7045787	[ID]
3	93760	9996178	[OPERATOR]
4	237738	26092982	[CHR/STR]
5	150400	15872100	[(NEXTWORD)]
6	43874	5190433	[COMMENT]
7	844359	95006489	[NEXTWORD]
8	20	2405	[REST]

profiler report PROFILE-SCANNER3
timestamps
635.489.452 **794.873.037**

buckets			
b#	nextcounts	clockticks	name
0	6225621	685133179	(etc)
1	247524	29335672	[ALPHANUM]
2	66366	7032118	[ID]
3	93760	9997433	[OPERATOR]
4	237738	26098202	[CHR/STR]
5	150422	15861717	[(NEXTWORD)]
6	43885	5193847	[COMMENT]
7	149888	16295019	[NEXTWORD]
8	14	1679	[REST]

Result #4: alphanum? in code, len-indexed string search

profiler report **PROFILE-CC64-1**
timestamps
644.730.028 **805.359.052**

buckets			
b#	nextcounts	clockticks	name
0	216494	22948435	(etc)
1	480396	56645293	[STRINGS]
2	947278	104421341	[MEMMAN]
3	1396465	154541959	[FILE-HDL]
4	316896	34711112	[INPUT]
5	832390	90610649	[SCANNER]
6	155294	17603763	[SYMTAB]
7	1858844	201343115	[PARSER]
8	1109617	122555758	[PASS2]

profiler report PROFILE-CC64-1
timestamps
579.980.460 **740.543.948**

buckets			
b#	nextcounts	clockticks	name
0	216506	22950222	(etc)
1	114310	12976436	[STRINGS]
2	742175	83029931	[MEMMAN]
3	1396517	154575151	[FILE-HDL]
4	316896	34740882	[INPUT]
5	832390	90869725	[SCANNER]
6	155294	17623890	[SYMTAB]
7	1858844	201608564	[PARSER]
8	1109737	122235615	[PASS2]

Result overall: 31% time saved, 45% speed gain

profiler report **PROFILE-CC64-1**
timestamps
919.522.732 **1.078.906.060**

profiler report PROFILE-CC64-1
timestamps
579.980.460 **740.543.948**

buckets			
b#	nextcounts	clockticks	name
0	475419	52822277	(etc)
1	1037243	114416784	[MEMMAN]
2	1384162	153154008	[FILE-HDL]
3	797224	122822197	[INPUT]
4	2695157	299076306	[SCANNER]
5	153639	17403250	[SYMTAB]
6	1826434	197679185	[PARSER]
7	1100491	121509788	[PASS2]
8	0	0	[SHELL]

buckets			
b#	nextcounts	clockticks	name
0	216506	22950222	(etc)
1	114310	12976436	[STRINGS]
2	742175	83029931	[MEMMAN]
3	1396517	154575151	[FILE-HDL]
4	316896	34740882	[INPUT]
5	832390	90869725	[SCANNER]
6	155294	17623890	[SYMTAB]
7	1858844	201608564	[PARSER]
8	1109737	122235615	[PASS2]

Links

All to be found under <https://github.com/pzembrod/cc64>:

- profiler: <src/common/profiler.fth>
- profiler activation: <src/cc64/invoke.fth>
- instrumented code: <src/cc64/cc64.fth> & <src/cc64/scanner.fth>
- metrics definitions: <src/cc64/prof-metrics.fth>
- profiling results: <tests/e2e/profile-register>
- cc64 input scripts for different metrics: tests/e2e/*.pfs

Conclusion

- Profiler proved practical & easy to use
- Good overview & drilldown with multiple metrics
- Different metrics within one compiled binary
- Result: Small to moderate optimizations yielded 45% speed increase
- 190 lines of code
- Gross runtime penalty for instrumentation < 4x
- Prerequisite: ITC or DTC

Thank you for your attention!

Questions?