Profile R Code for Speed

Presented at Chicago RUG
May 14, 2014

FOSS Finance

What is this “profiling” you speak of?

- Wikipedia:
  ...profiling is a form of dynamic program analysis that measures, for example, the memory or time complexity of a program, the usage of particular instructions, or frequency and duration of function calls.
- Why should you profile your code?
  - Primary reason:
    - You need your code to run faster or use less memory
  - Secondary reasons:
    - You're impatient
    - Your code takes forever, and you have nothing else to do

FOSS Finance

What is this “profiling” you speak of?

- The obligatory quote:

  We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified.

  -Donald Knuth

Before trying to make it faster...

- Think carefully and ask yourself:
  - Is the potential decreased run-time worth *my* time?
    - Will this function/code be called often?
    - Am I often waiting for this function/code to run?
  - How likely is it that I'll break something?
    - Make a backup! Better yet, use version control.
    - Test your changes to ensure identical output.

Notes from *Writing R Extensions*

- Profiling imposes a small performance penalty
- Output files from profiling long runs at small intervals can be very large
- Profiling short runs can give misleading results due to garbage collection

Notes from *Writing R Extensions*

- What is `Rprof`?

  "The command `Rprof` is used to control profiling, and its help page can be consulted for full details. Profiling works by recording at fixed intervals (by default every 20 milliseconds) which line in which R function is being used, and recording the results in a file (default `Rprof.out` in the working directory). Then the function `summaryRprof` or the command-line utility `R CMD Rprof Rprof.out` can be used to summarize the activity."

Other things to keep in mind

- Special primitives will not be recorded by `Rprof`
  - `for`, `while`, `repeat` loops; `log`, `round`, `||`, `&&`
  - several others (see *R Internals*, "Special Primitives")
  - For example:

```
mm <- 1:1e7
x=runif(mm)
Rprof(); y=log(x); Rprof(NULL); summaryRprof()
Error in summaryRprof() : no events were recorded
Rprof(); for(i in mm) TRUE && FALSE; Rprof(NULL); summaryRprof()
Error in summaryRprof() : no events were recorded
Rprof(); for(i in mm) TRUE || FALSE; Rprof(NULL); summaryRprof()
Error in summaryRprof() : no events were recorded
```

Other things to keep in mind

- `<Anonymous>` can be hard to find
  - Caused by, e.g.: `lapply(foo, function(x) x)`
  - Using named functions makes profiling easier
- To line profile when profiling a package, the DESCRIPTION needs:
  - `ByteCompile: FALSE`
    `KeepSource: TRUE`

Profiling quantstrat

- Code:
  - DF_GE.R contains parameter setup
  - DF_strat.R contains actual strategy code
    - Calls to `add.indicator`, `add.signal`, `add.rule`, `applyStrategy`, etc.
- Data:
  - 1 instrument, 5-second data
    - 2012-07-09 22:00:00/2014-01-01 22:00:00
    - 6,353,883 rows, 13 columns

Profiling quantstrat

- Calling `Rprof`:

```
Rprof(interval=0.02, memory.profiling=FALSE, line.profiling=TRUE, gc.profiling=FALSE)
applyStrat <- applyStrategy(Strat, Portfolio, symbols=Symbols)
Rprof(NULL)
summaryRprof(lines="both")
```

- `summaryRprof` output: what are we looking at?
  - self.time: only time in function, **not** function(s) it called
  - total.time: self.time plus time in function(s) it called
- Low-hanging fruit: function calls with
  - High total.time and high self.time/total.time
  - High self.time

Profiling quantstrat (rev 1532)

```
> summaryRprof(lines='both')
$by.self
                self.time self.pct total.time total.pct
"match"           29665.28    62.75   29671.56     62.76
"!"                3275.90     6.93    3276.90      6.93
"diff.default"     3035.34     6.42    3035.40      6.42
"is.na"            2669.46     5.65    2669.58      5.65
"sort.int"         2644.84     5.59    3166.48      6.70
"match.call"       2266.60     4.79    2266.64      4.79
"which"            1227.30     2.60   36201.16     76.57
"NextMethod"        262.50     0.56     263.94      0.56
"[.xts"             179.90     0.38   37045.40     78.36

$by.total
                total.time total.pct self.time self.pct
"applyStrategy"   47277.36    100.00      0.00     0.00
"applyRules"      47257.66     99.96      0.00     0.00
strategy.R#158    47256.86     99.96      0.00     0.00  # applyRules
"[.xts"           37045.40     78.36    179.90     0.38
"do.call"         36999.96     78.26      1.62     0.00
"ruleProc"        36999.28     78.26      0.00     0.00
rules.R#704       36981.94     78.22      0.18     0.00  # ruleProc/do.call
"ruleSignal"      36980.22     78.22      0.90     0.00
rules.R#641       36810.74     77.86      0.08     0.00  # ruleProc
ruleSignal.R#62   32806.76     69.39      3.00     0.01  # if
```

FOSS Finance

Profiling quantstrat (rev 1532)

- Lots of time spent calling `match`, what is calling it?

```
> library(proftools)
> profData <- readProfileData("Rprof.out")
> printProfileCallGraph(profData)
Call graph

index    % time     % self   % children     name

                                             eval <cycle 1> [3]
[1]      100.00       0.00     100.00     applyStrategy <cycle 1> [1]
                                             applyIndicators <cycle 1> [47]
                                             applyRules <cycle 1> [6]
                                             applySignals <cycle 1> [32]
-----------------------------------------------
<snip>
-----------------------------------------------
                                             [.xts <cycle 1> [9]
<snip>
[10]      64.40       2.61      61.80     which <cycle 1> [10]
<snip>
                     50.49       0.00         match [13]
-----------------------------------------------
```

Profiling quantstrat (rev 1532)

- `[.xts` calls `match`, and we call `[.xts` a lot

```
`[.xts` <-
function(x, i, j, drop = FALSE, which.i=FALSE,...)
{
    <snip>

    if (timeBased(i)) {
      if(inherits(i, "POSIXct")) {
        i <- which(!is.na(match(.index(x), i)))  # <- the culprit
      } else if(inherits(i, "Date")) {
        i <- which(!is.na(match(.index(x), as.POSIXct(as.character(i),tz=indexTZ(x)))))
      } else {
        # force all other time classes to be POSIXct
        i <- which(!is.na(match(.index(x), as.POSIXct(i,tz=indexTZ(x)))))
      }
      i[is.na(i)] <- 0
    } else

    <snip>
}
```

Profiling quantstrat (rev 1532)

- What can we do?
  - Change `[.xts`?
    - Very likely to break lots of things
  - Call `[.xts` less?
    - A good idea, but probably not easy to do
  - Change how we call `[.xts`?
    - Ah ha!

Profiling quantstrat (rev 1532)

- Pass integer to `[.xts` instead of a `POSIXct` object
  - Avoids the `match` call
  - quantstrat's dimension-reduction loop already calculates all the integer index values we need to evaluate

```
> library(microbenchmark)
> x <- .xts(matrix(NA_real_, 6353883, 13), 1:6353883)
> i <- 3176942
> p <- index(x)[i]
> microbenchmark(x[i], x[p])
# Unit: microseconds
#  expr       min         lq     median         uq        max neval
#  x[i]    39.811    41.3225    56.6335    58.6695     81.803   100
#  x[p] 82893.943 85034.0615 85757.5605 86749.7375 128808.081   100
```

Profiling quantstrat (rev 1561)

```
> summaryRprof(lines='both')
$by.self
                 self.time self.pct total.time total.pct
".Call"              360.96    24.80     360.96     24.80
"NextMethod"         242.04    16.63     243.92     16.76
"[.xts"              158.32    10.88     513.80     35.30
"[.POSIXct"          110.40     7.59     111.10      7.63
"unclass"             95.12     6.54      95.12      6.54
"dimnames<-.xts"      64.62     4.44     225.12     15.47
"c"                   64.38     4.42      64.38      4.42
"colnames<-"          63.40     4.36     256.52     17.63
"rownames<-"          32.90     2.26      32.90      2.26
signals.R#275         31.44     2.16     286.60     19.69  # colnames<-

$by.total
                total.time total.pct self.time self.pct
"applyStrategy"    1455.42    100.00      0.00     0.00
"applyRules"       1437.16     98.75      0.00     0.00
strategy.R#178     1436.06     98.67      0.00     0.00  # applyRules
rules.R#654        1185.28     81.44      0.02     0.00  # nextIndex
"nextIndex"        1185.26     81.44      0.00     0.00
rules.R#461         723.62     49.72      0.06     0.00  # sigThreshold/nextIndex
"sigThreshold"      723.60     49.72      0.00     0.00
"["                 629.88     43.28      1.08     0.07
"[.xts"             513.80     35.30    158.32    10.88
```

FOSS Finance

Profiling quantstrat (rev 1561)

- What can we do?
  - Make `sigThreshold` faster
    - Don't call `colnames<-` unless we have to; it copies
    - We only need the first cross; breaking a for loop may be faster if the first cross is near the beginning of the vector
    - R loop might have been faster, but C loop would be faster still

Profiling quantstrat (rev 1561)

```
> svn diff -r1561:1562 quantstrat
Index: quantstrat/R/rules.R
===================================================================
--- quantstrat/R/rules.R        (revision 1561)
+++ quantstrat/R/rules.R        (revision 1562)
@@ -457,10 +459,9 @@
           }
           if (is.na(col)) stop("no price discernable for limit in applyRules")
       }
-      # use sigThreshold
-      cross<-sigThreshold(label='x',data=mktdata,column=col,threshold=tmpprice,relationship=relationship)
-      cross <- cross[timespan][-1]  # don't look for crosses on curIndex
-      if(any(cross)){
+      # use .firstThreshold to find the location of the first tmpprice that crosses mktdata[,col]
+      cross <- .firstThreshold(data=mktdata, col, tmpprice, relationship, start=curIndex+1)
+      if(cross < nrow(mktdata)){
           # find first index that would cross after this index
           #
           # current index = which(cross[timespan])[1]
@@ -468,7 +469,7 @@
           # need to subtract 1 index==1 means current position
           #
-          newidx <- curIndex + which(cross)[1]
+          newidx <- cross

           #if there are is no cross curIndex will be incremented on line 496
           # with curIndex<-min(dindex[dindex>curIndex]).
```

Profiling quantstrat (rev 1561)

- Simple R/C functions to find the index value for the first TRUE element for a given logical comparison
  - `.firstThreshold` is very similar to `sigTheshold`

```
.firstThreshold <- function(Data, threshold=0, relationship, start=1) {
    rel <- switch(relationship[1],
                  '>'    =  ,
                  'gt'   = 1,
                  '<'    =  ,
                  'lt'   = 2,
                  'eq'   = 3,
                  'gte'  =  ,
                  'gteq' =  ,
                  'ge'   = 4,
                  'lte'  =  ,
                  'lteq' =  ,
                  'le'   = 5)
    .Call('firstThreshold', Data, threshold, rel, start)
}
```

Profiling quantstrat (rev 1561)

```
#include <R.h>
#include <Rinternals.h>
SEXP firstThreshold(SEXP x, SEXP th, SEXP rel, SEXP start)
{
    int i, int_rel, int_start;
    double *real_x=NULL, real_th;

    real_th = asReal(th);
    int_rel = asInteger(rel);
    int_start = asInteger(start)-1;
    SEXP result = ScalarInteger(nrows(x)); /* return if never TRUE */

    switch(int_rel) {
        case 1:  /* >  */
            real_x = REAL(x);
            for(i=int_start; i<nrows(x); i++)
                if(real_x[i] >  real_th) {
                    result = ScalarInteger(i+1);
                    break;
                }
            break;
/* snip */
    }
    return(result);
}
```

Profiling quantstrat (rev 1562)

```
> summaryRprof(lines='both')
$by.self
                self.time self.pct total.time total.pct
".Call"            182.44    33.50     182.46     33.50
"[.POSIXct"        169.10    31.05     169.66     31.15
"NextMethod"        16.44     3.02      18.56      3.41
"dim"                9.62     1.77       9.62      1.77
"[.xts"              8.68     1.59     161.14     29.59
".External"          8.06     1.48       8.06      1.48
"match"              7.14     1.31      11.38      2.09
".xts"               7.06     1.30      33.86      6.22
"grep"               7.06     1.30       9.48      1.74

$by.total
                 total.time total.pct self.time self.pct
"applyStrategy"      544.58    100.00      0.00     0.00
"applyRules"         522.80     96.00      1.48     0.27
"["                  335.80     61.66      2.56     0.47
"nextIndex"          207.54     38.11      2.86     0.53
".Call"              182.46     33.50    182.44    33.50
"[.POSIXct"          169.66     31.15    169.10    31.05
".firstThreshold"    167.78     30.81      0.10     0.02
"[.xts"              161.14     29.59      8.68     1.59
```

Profiling quantstrat (rev 1562)

- What can we do?
  - Call `[.xts` less
    - Wasn't easy...
    - Re-factored to store specific `mktdata` columns (e.g., price) in an intermediate object before entering the main loop
  - Move other checks outside the main loop
    - Fairly easy...
    - Fairly marginal improvement

Conclusion

- Make good choices
  - Don't waste your time trying to save computing time
- Look for “low-hanging fruit”
- Anticipate spending more time looking for the bottleneck than writing a faster solution
- Don't get discouraged if your first attempt(s) don't work as well as you expect