Using the R Profiler

Biostatistics 140.776

Why is My Code So Slow?

- Profiling is a systematic way to examine how much time is spend in different parts of a program
- Useful when trying to optimize your code
- Often code runs fine *once*, but what if you have to put it in a loop for 1,000 iterations?
- Profiling is better than guessing

On Optimizing Your Code

- “We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil” –Donald Knuth
- Getting biggest impact on speeding up code depends on **knowing where the code spends most of its time**—this cannot be done without performance analysis or profiling

General Principles of Optimization

- Design first, then optimize
- Remember: Premature optimization is the root of all evil
- Measure (collect data), don’t guess. We’re statisticians after all!

Good Ol' `system.time()`

- Takes an arbitrary R expression as input (can be wrapped in curly braces)
- Computes the time (in seconds) needed to execute an expression
  - If there's an error, gives time until the error occurred
- Returns an object of class 'proc_time'
  - **user time**: time charged to the CPU for this expression
  - **elapsed time**: "wall clock" time

Good Ol' `system.time()`

- Usually, the user time and elapsed time are relatively close, for straight computing tasks
- Elapsed time may be > user time if the CPU spends a lot of time waiting around
- The user time may be larger than the elapsed time if your machine has multiple cores/processors (and is capable of using them)
  - Multi-threaded BLAS libraries (vecLib, ATLAS, ACML)
  - Parallel processing via the **parallel** package

Good Ol' `system.time()`

```
## Elapsed time > user time
system.time(readLines("http://www.jhsph.edu"))
   user  system elapsed
  0.004   0.002   0.431

## User time > elapsed time
hilbert <- function(n) {
        i <- 1:n
        1 / outer(i - 1, i, "+")
}
x <- hilbert(1000)
system.time(svd(x))
   user  system elapsed
  1.605   0.094   0.742
```

Timing Longer Expressions

```
system.time({
        n <- 1000
        r <- numeric(n)
        for(i in 1:n) {
                x <- rnorm(n)
                r[i] <- mean(x)
        }
})
 user  system elapsed
  0.087   0.001   0.088
```

microbenchmark

- The microbenchmark package allows for more accurate timing of small R expressions
- Uses system timers that have much higher temporal resolution (nanosecond)
- Can be used to compare multiple versions of the an expression to see which one is faster

microbenchmark

microbenchmark

```
> microbenchmark(3*3, 3^2, times = 10000L)

Unit: nanoseconds
  expr min  lq     mean median  uq   max neval
 3 * 3  99 173 217.0352    206 249 30081 10000
   3^2 141 225 285.5195    263 312 34227 10000
```

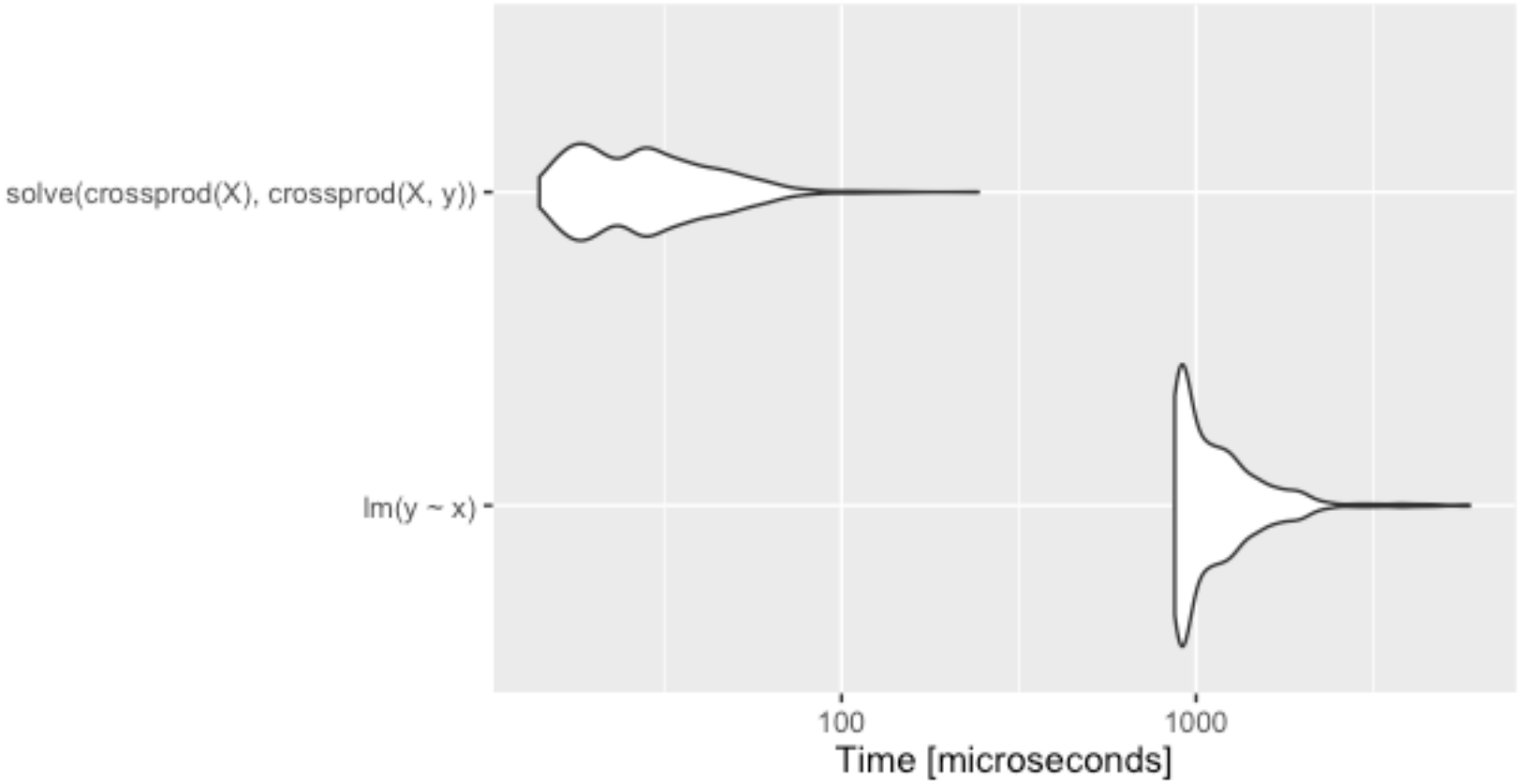
microbenchmark

```
x <- rnorm(500)
y <- 3 + 2*x + rnorm(500)
X <- cbind(1, x)

m <- microbenchmark(
   lm(y ~ x),
   solve(crossprod(X), crossprod(X, y)),
   times = 1000
)

autoplot(m)
```

microbenchmark



Beyond Benchmarking

- Using `system.time()` and `microbenchmark()` allows you to test certain functions or code blocks to see if they are taking excessive amounts of time
- Assumes you already know where the problem is and can call `system.time()` on it
- What if you don’t know where to start?

The R Profiler

- The `Rprof()` function starts the profiler in R
  - R must be compiled with profiler support (but this is usually the case)
- The `summaryRprof()` function summarizes the output from `Rprof()` (otherwise it's not readable)
- DO NOT use `system.time()` and `Rprof()` together or you will be sad

The R Profiler

- `Rprof()` keeps track of the function call stack at regularly sampled intervals and tabulates how much time is spend in each function
- Default sampling interval is 0.02 seconds
- NOTE: If your code runs very quickly, the profiler is not useful

R Profiler Output

```
## lm(y ~ x)

sample.interval=10000
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
```

summaryRprof()

- The summaryRprof() function tabulates the R profiler output and calculates how much time is spend in which function
- There are two methods for normalizing the data
- “by.total” divides the time spend in each function by the total run time
- “by.self” does the same but first subtracts out time spent in functions above in the call stack

By Total

$by.total

	total.time	total.pct	self.time	self.pct
"lm"	7.41	100.00	0.30	4.05
"lm.fit"	3.50	47.23	2.99	40.35
"model.frame.default"	2.24	30.23	0.12	1.62
"eval"	2.24	30.23	0.00	0.00
"model.frame"	2.24	30.23	0.00	0.00
"na.omit"	1.54	20.78	0.24	3.24
"na.omit.data.frame"	1.30	17.54	0.49	6.61
"lapply"	1.04	14.04	0.00	0.00
"[.data.frame"	1.03	13.90	0.79	10.66
"["	1.03	13.90	0.00	0.00
"as.list.data.frame"	0.82	11.07	0.82	11.07
"as.list"	0.82	11.07	0.00	0.00

By Self

```
$by.self
                        self.time self.pct total.time total.pct
"lm.fit"                     2.99    40.35       3.50     47.23
"as.list.data.frame"         0.82    11.07       0.82     11.07
"[.data.frame"               0.79    10.66       1.03     13.90
"structure"                  0.73     9.85       0.73      9.85
"na.omit.data.frame"         0.49     6.61       1.30     17.54
"list"                       0.46     6.21       0.46      6.21
"lm"                         0.30     4.05       7.41    100.00
"model.matrix.default"       0.27     3.64       0.79     10.66
"na.omit"                    0.24     3.24       1.54     20.78
"as.character"               0.18     2.43       0.18      2.43
"model.frame.default"        0.12     1.62       2.24     30.23
"anyDuplicated.default"      0.02     0.27       0.02      0.27
```

summaryRprof() Output

```
$sample.interval
[1] 0.02

$sampling.time
[1] 7.41
```

Summary

- `Rprof()` runs the profiler for performance of analysis of R code
- `summaryRprof()` summarizes the output of `Rprof()` and gives percent of time spent in each function (with two types of normalization)
- C or Fortran code is **not** profiled
- Good to break your code into functions so that the profiler can give useful information