Writing Efficient R Code

Stephen Weston

Yale Center for Research Computing

November 2, 2016

Resources

I'll demonstrate some R scripts during this workshop which you can download at the URL listed below.

I've also included some good resources for writing efficient R code.

Examples	`http://github.com/steveweston/efficient-R`
R Book	`http://adv-r.had.co.nz`
R Manuals	`http://cran.r-project.org/manuals.html`
R Inferno	`http://www.burns-stat.com/documents/books/the-r-inferno`

Is R slow?

R programs can be slow, but well written R programs are usually fast enough.

- Speed was not the primary design criteria
- Designed to make programming easier
- Slow programs often a result of bad programming practises or not understanding how R works
- There are various options for calling C or C++ functions from R

The goal of this bootcamp is to help you write better R programs that are less likely to require later optimization.

General R programming advice

- If you don't understand something in R, try some experiments
- R has a number of quirks: learn about them
- Download and browse the R source
- Study well written R programs
- Browse the R documentation
- Break your code into functions when appropriate
- Use functions to reduce the need for global variables
- Write lots of tests for your functions
- Learn how to make R packages
- Use version control (such as git) to keep track of changes

Code tuning advice

Tuning code is tricky and not intuitive, so be methodical.

- Profile your code and focus your efforts on problem areas
- Run benchmarks to determine how differences effect performance
- Don't get carried away with micro-optimizations
- Consider a better algorithm
- Preallocate result vectors
- Be careful to avoid duplication of large objects
- Become familiar with R's vector functions and "apply" functions
- Learn different vector and matrix indexing techniques
- Compile your R functions into byte code using *cmpfun*
- Learn to use a parallel computing package
- Consider specialized packages: data.table, bigmemory, plyr, RSQLite
- If you know C, C++, or Fortran, learn to call it from R
- Use monitoring tools such as *top*, *Activity Monitor*, etc

Are for loops in R slow?

Not all **for** loops are bad, but many of the most common mistakes involve **for** loops. The classic mistake is not preallocating a result vector.

classic bad for loop

```
n <- 1000000
x <- NULL
for (i in 1:n) {
  x[i] <- sqrt(i)
}
```

This example is a problem due to a combination of issues:

- large number of iterations
- tiny amount of computation per iteration
- result vector is reallocated and copied on each iteration eventually triggering garbage collection periodically

Preallocate result vectors

Preallocating the result vector avoids memory management problems.

preallocate x

```
n <- 1000000
x <- double(n)
for (i in 1:n) {
  x[i] <- sqrt(i)
}
```

This is a great improvement over the previous example, but it's still slow because of the many tiny iterations. Fortunately we can replace the **for** loop with a vector function:

```
x <- sqrt(1:n)
```

Profiling

Profiling helps you focus on the slow parts of your code thus saving you programming time.

R has builtin support for profiling, but there are additional packages available:

- proftools
- profvis (RStudio support)

Basic profiling is quite easy:

```
Rprof('prof.out')
slowfunction(x, 1000)
Rprof(NULL)

print(summaryRprof('prof.out')
```

Benchmarking

Once you know what section of your code is slow via profiling, benchmarking tools will help you to time your code.

- more acurate than standard profiling tools
- **system.time** is useful for long running code
- the **microbenchmark** package is useful for analyzing short running code
- I like to put code into a function
- benchmark different versions of code for comparison

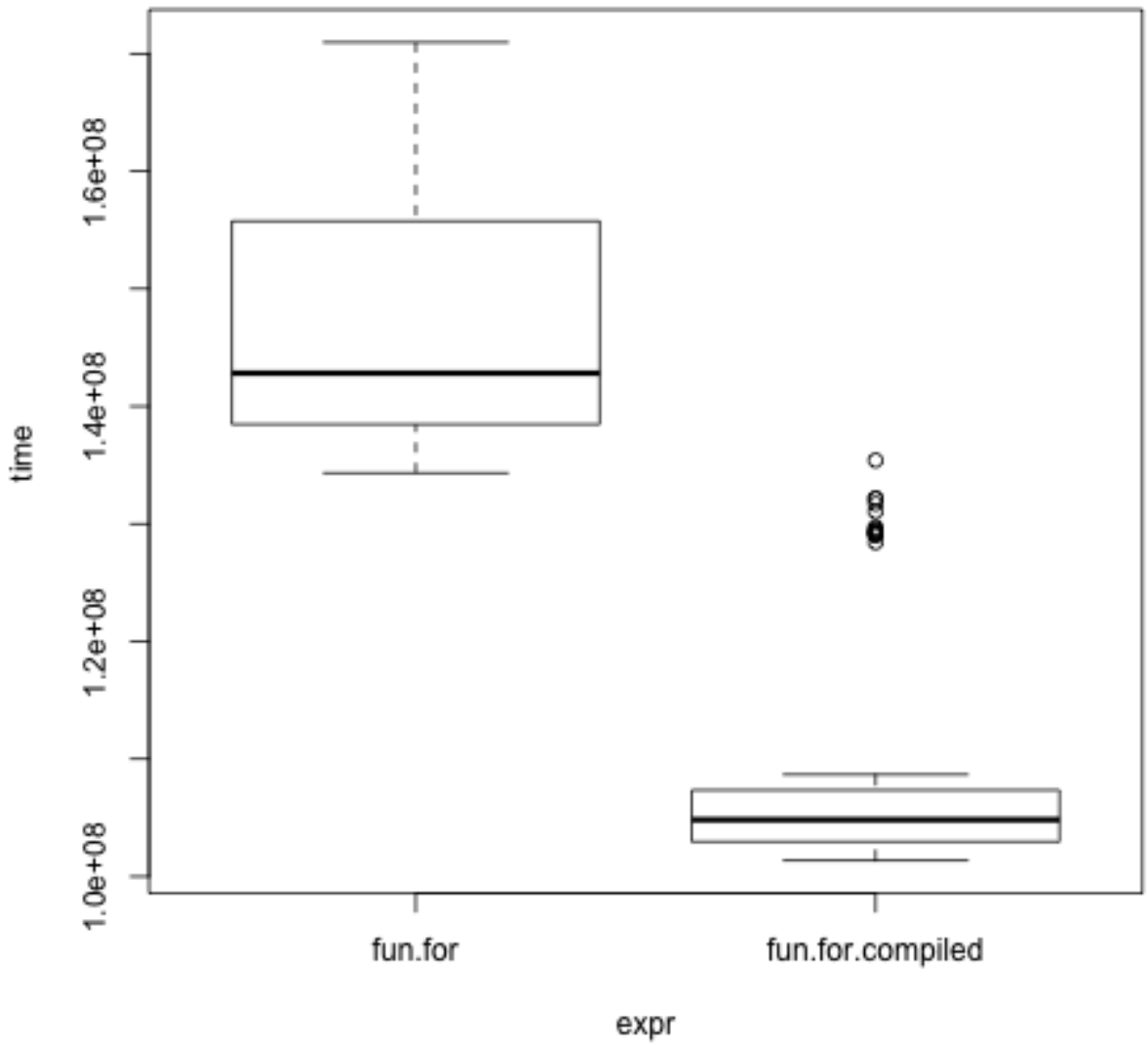
Benchmarking with the microbenchmark package

The microbenchmark package is particularly good for timing very short running code.

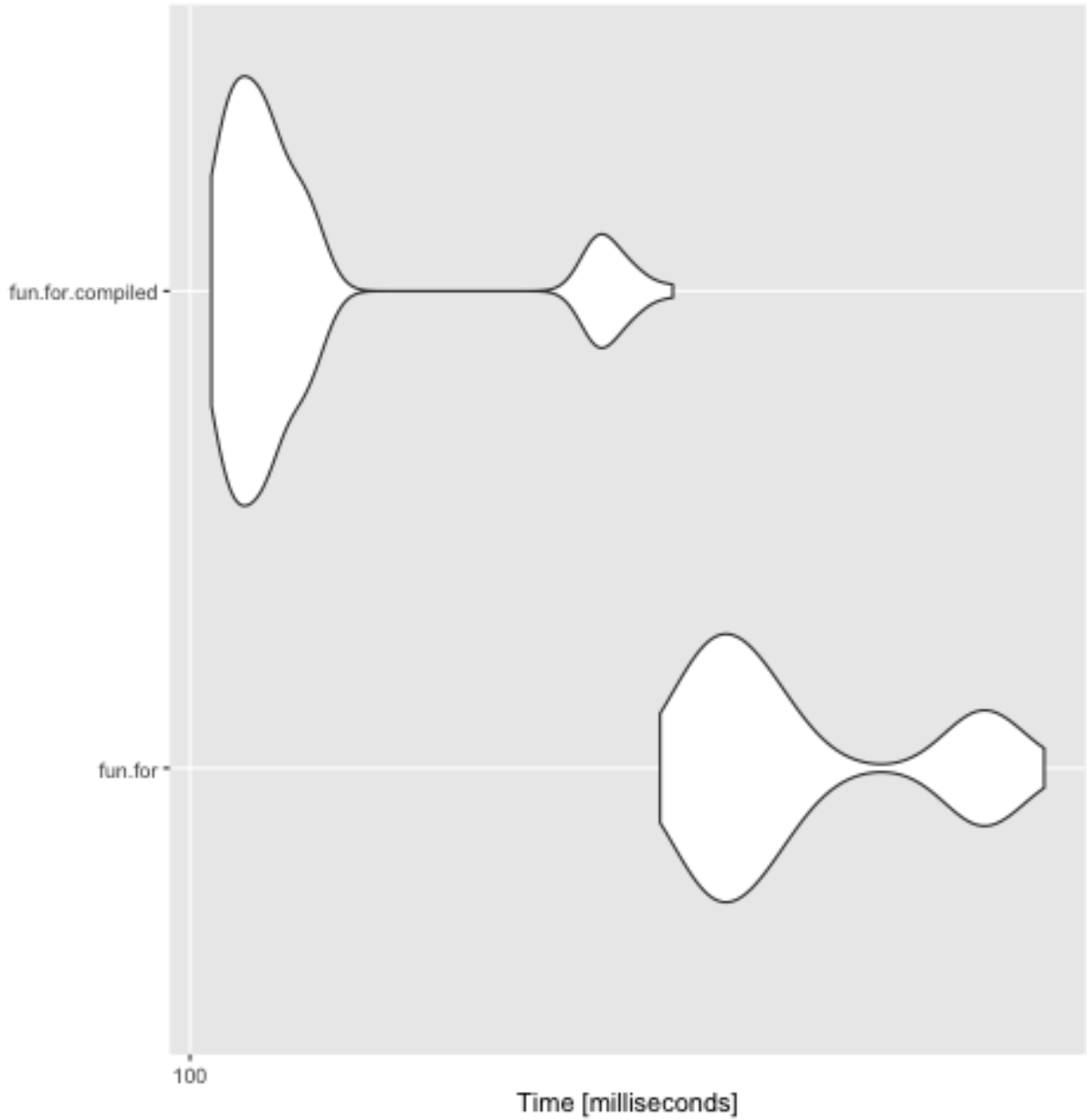
```
library(microbenchmark)
res <- microbenchmark(
  fun.for=fun.for(X),
  fun.for.compiled=fun.for.compiled(X)
)
print(res)
plot(res)

library(ggplot2)
autoplot(b)
```

microbenchmark plot



microbenchmark autoplot



Use R byte code compiler

The cmpfun function from the standard compiler package can significantly improve the performance of R functions. **for** loops in particular may run much faster after compiling them.

```
fun.for <- function(x, seed=1423) {
  set.seed(seed)
  y <- double(length(x))
  for (i in seq_along(x)) {
    y[i] <- rnorm(1) * x[i]
  }
  y
}

library(compiler)
fun.for.compiled <- cmpfun(fun.for)
```

Note that byte code is not the same as machine code.

R vector functions

Vector functions are central to good R programming.

- fast since implemented as a single C or Fortran function
- concise and easy to read
- nested calls to vector functions can often replace for loops
- heavy use of vector functions can use a lot of memory

Useful vector functions include:

- math operators: +, -, *, /, ^, %/%, %%
- math functions: abs, sqrt, exp, log, log10, cos, sin, tan, sum, prod
- logical operators: &, |, !
- relational operators: ==, !=, <, >, <=, >=
- string functions: nchar, tolower, toupper, grep, sub, gsub, strsplit
- conditional function: ifelse (pure R code)
- misc: which, which.min, which.max, pmax, pmin, is.na, any, all, rnorm, runif, sprintf, rev, paste, as.integer, as.character

Vector indexing

Vectors can be used as indices to vectorize index operations.

```
x <- rnorm(10)

# Extract subvector
x[3:6]

# Extract elements using result of vector relational operation
x[x > 0]

# Set NA's to zero
x[is.na(x)] <- 0
```

Matrix indexing

Vectors and matrices can be used as indices to vectorize index operations.

```
m <- matrix(rnorm(100), 10, 10)

# Extract submatrix (non-consecutive columns)
m[3:4, c(5,7,9)]

# Extract arbitrary elements as vector
m[cbind(3:6, c(2,4,6,9))]

# Extract elements using result of vector relational operation
m[m > 0]

# Set NA's to zero
m[is.na(m)] <- 0
```

Beware of object duplication

R uses *pass by value* semantics for function arguments. In general, this requires making copies of objects, although R tries to avoid copying unless necessary.

- you can pass a matrix to a function and not worry that it will be modified as a side effect
- if modifications are desired, function must return modified object
- duplication takes time and memory
- objects are sometimes duplicated when not strictly necessary, sometimes causing serious performance problems
- see section 1.1.2 of the "R Internals" manual for more information
- for information on Luke Tierney's work to implement reference counting to reduce object duplication, see https://developer.r-project.org/Refcnt.html

Example of object duplication

The **tracemem** function reports when an object is duplicated which is very useful for debugging performance problems.

In this example, object duplication is expected and helpful.

```
> x <- double(10)
> tracemem(x)
[1] "<0x7fd2eb256750>"
> y <- x
> y[1] <- 10
tracemem[0x7fd2eb256750 -> 0x7fd2eb1cbff0]:
> .Internal(inspect(x))
@7fd2eb256750 14 REALSXP g0c5 [NAM(2),TR] (len=10, tl=0) 0,0,0,0,0,
> .Internal(inspect(y))
@7fd2eb1cbff0 14 REALSXP g0c5 [NAM(1),TR] (len=10, tl=0) 10,0,0,0,0,
```

Example of unexpected object duplication

Passing a matrix to a non-primitive function such as **nrow** will set the NAMED bit, causing it to be duplicated when next modified. This doesn't seem helpful, but is presumably necessary to insure that the object isn't modified.

```
> m <- matrix(0, 3, 3)
> tracemem(m)
[1] "<0x7fc168d29df0>"
> m[1,1] <- 1
> nrow(m)
[1] 3
> m[1,1] <- 2
tracemem[0x7fc168d29df0 -> 0x7fc168d21f58]:
```

So be careful what you do with large objects that you modify in a for loop.

Simple parallel computing using mclapply

The standard **parallel** package includes a very useful function called **mclapply**.

- **mclapply** is nearly a drop-in replacement for **lapply**
- use the **mc.cores** argument to specify the number of workers to use
- does not execute in parallel on Windows (depends on **fork** system call)
- not generally safe to use in R GUIs (such as RStudio)

parallel randomForest example

```
library(parallel)
library(randomForest)

x <- matrix(runif(500), 100)
y <- gl(2, 50)
ntree <- 1000
cores <- detectCores()
vntree <- rep(ntree %/% cores, cores)
worker <- function(n) randomForest(x, y, ntree=n)
rf <- do.call('combine',
              mclapply(vntree, worker, mc.cores=cores))
```

Split problem into smaller tasks

R makes it easy to read entire data sets in one operation, but reading it in parts can be much more efficient.

- Splitting the problem into smaller tasks is compatible with parallel computing techniques
- The foreach/iterators packages provide tools to split inputs into smaller pieces
- Use Linux commands (split, awk, etc) to preprocess data files by splitting data files and removing unneeded fields

Beware of read.table

The read.table function is commonly used for reading data files, but it can be very slow on large files.

- Use of the colClasses argument can improve performance
- colClasses can be used to skip a column, using less memory
- It can be faster to read a file in smaller chunks using the nrows argument
- The scan function can be faster
- Consider using similar functions from different packages, such as data.table, sqldf, and bigmemory

bigmemory package

The bigmemory package defines new matrix objects that are mutable, allowing memory to be used more efficiently since the matrices are never automatically duplicated.

- Written by Mike Kane and Jay Emerson of Yale University
- Works very well in conjunction with parallel computing
- big.matrix – can use a backing file that is memory mapped
- package biganalytics – apply, biglm, bigglm, bigkmeans, colmax
- package bigtabulate – bigsplit, bigtabulate, bigtable, bigtsummary
- package synchronicity – boost.mutex, lock, unlock

Save data in binary format

Saving data in a binary format can make it much faster to read the data later. There are a variety of functions available to do that:

- save/load
- writeBin/readBin
- write.big.matrix/read.big.matrix (from the bigmemory package)

SQLite

Consider putting data into an SQLite database.

- RSQLite packages is easy to use
- Easy to get subsets of the data into a data frame
- Command line tool very useful for experimenting with queries
- Database can be accessed from many different languages
- The sqldf package may be useful, also
- Can be quite slow