OPTIMIZING R CODE

Jacalyn Huband

UVA Research Computing

https://github.com/jhuband/Optimizing-R/

What is Code Optimization?

- The process of making code more efficient (either with time or memory)
- Important caveats:

  - The correctness of the code must be preserved.
  - The code should run faster “on average”.
  - There is a tradeoff between your efforts and the computer’s efforts. (**Should you spend a week trying to make a program faster by 1 sec?**)

https://github.com/jhuband/Optimizing-R/

What are the basics for writing optimal R codes?

https://github.com/jhuband/Optimizing-R/

Vectorization

- **Avoid using loops.**

  Take advantage of applying operations to entire arrays/lists.

Before:

```
total <- 0.0

for (x in values){
   total <-  total + sin(x)^2 – 3*x
}
```

After:

```
total <- sum( sin(values)^2 – 3*values )
```

https://github.com/jhuband/Optimizing-R/

Tidyverse

- **Use *tidyverse* to manipulate data**

  tidyverse is a collection of packages designed specifically for ease of manipulating data. Designed by Hadley Wickham, it has a different approach to its syntax.

Before:

```
myData <- read.csv(filename)
quality <- sort(unique(myData$quality))

N <- length(quality) ;   avg_chl <- rep(0.0, N)
for (i in 1:N) {
  qual <- quality[i]
  ndx <- which(myData$quality == qual)
  avg_chl[i] <- mean(myData$chlorides[ndx])
}
df <- data.frame(quality, avg_chl)
```

After:

```
library(tidyverse)
 myData <- read_csv(filename)
 myData %>% group_by(quality) %>%
                    summarize(avg_chl = mean(chlorides))
```

Pre-allocate Memory

- **Don't "grow" a list or array.**

  If you know the size and type of the list or array, reserve memory for the entire list/array before doing calculations.

Before:

```
y = c( )

for (x in values){
   if ( x > 0.5) {
      y  <- c(y, myFunc1(x))
   } else {
      y <- c(y, myFunc2(x))
   }
}
```

After:

```
y <-  rep(0.0, length(values);  i = 1

for (x in values){
   if ( x > 0.5) {
      y[i] <- myFunc1(x)
   } else {
      y[i] <- myFunc2(x)
   }
 i = i + 1
}
```

Avoid Overuse of Parentheses

- **R treats parentheses as function calls.**

  The contents inside parentheses are evaluated and stored in a special list structure.
  There is overhead for allocating the list, storing, and retrieving the results.

Before:

```
y <- ( (x)^2 + 3*( sin(x) ) )
```

After:

```
y <- x^2 + 3*sin(x)
```

Use Exponentials

- **Squaring a number is okay.**

  When squaring a value or expression, exponentiation is okay.

Before:

```
y <- x*x
z <- x*x*x*x
```

After:

```
y <- x^2
z <- (x^2)^2
```

Unlist without names

- **Avoid having names turned on in lists.**

  Whether the items in our list are named or not, we should turn off the use.names option in the unlist function.

Before:

```
x <- lapply(1:myCount, create_list)

y <- unlist(x)
```

After:

```
x <- lapply(1:myCount, create_list)

y <- unlist(x, use.names=FALSE)
```

ifelse

- **Avoid ifelse for vectors.**

  Although it is expected to be more concise, the ifelse statement can be less efficient than a more detailed approach.

Before:

```
x <- runif(numValues, min=1, max=20)

y <- ifelse(x > 10, 1, -1)
```

After:

```
x <- runif(numValues, min=1, max=20)

y <- rep(-1, length(x))
y[x > 10] <- 1
```

Next Up . . .

- Rebecca Belshe and Gil Speyer

https://github.com/jhuband/Optimizing-R/