HPC-R Exercises: Basics

Drew Schmidt

11/12/2016

Basics

Debugging

1. Find the bug:

```
x <- 0:9
if (x[1] = 999){
  print(x)
}
```

2. Find the bug:

```
x <- 0:9
if (x[0] == 999){
  print(x)
}
```

3. Find the bug:

```
myfactorial <- function (x)
{
  if (x==1)
    return(1)
  else
    return( x*myfactorial(x) )
}
```

4. Use the `debug()` function to debug this function:

```
f <- function(X)
{
  scl <- sum(as.numeric(X$a))
  ans <- scl * (as.numeric(X$a)+X$b)
  ans <- crossprod(ans)

  return(ans)
}

X <- list(a=factor(-2:2), b=matrix(1:30, nrow=10))
f(X)
```

The correct output is:

```
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```

5. Find the bug:

```
f <- function(n)
{
  if (n==1)
    return(1)
  else {
    if (n%%2==0)
      return(n/2)
    else
      return(3*x+1)
  }
}

x <- 1
f(x)
n <- 3
f(n)
```

Profiling

1. For `x <- matrix(rnorm(1000*250), 1000, 250)`, which is faster (single execution):
    - `t(x) %*% x`
    - `crossprod(x)` ?
2. Explore the call stack of `example(glm)` with `Rprof()`.
3. Re-run exercise 2 with `Rprof(memory.profiling=TRUE)`, and examine with `summaryRprof(memory="both")`. See the help files for an explanation of the new output.

Benchmarking

1. Which function is faster on average? Try several values of n.

```
f <- function(n)
{
  x <- c()
  for (i in 1:n)
    x[i] <- i*i

  return(x)
}

g <- function(n)
{
  x <- numeric(n)
  for (i in 1:n)
    x[i] <- i*i

  return(x)
}
```

2. Which function is faster on average? Try several values of n.

```
h <- function(n) sapply(1:n, function(i) i*i)

i <- function(n) (1:n)*(1:n)
```

Answers

Debugging

1. Use `==` for comparison, not `=` (which can be used for assignment).
2. Vectors in `R` are indexed from 1, not 0 like in `C`. The vector `x` contains no 0'th element.
3. Calling `f(x)` from inside any function `f` will cause infinite recursion. The call should instead be `x*myfactorial(x-1)`.
4. The conversion of factors to numeric data is often not straight-forward. Try casting the factor as character first in the `scl <-` assignment.
5. Type `rm(x)` then re-run `f(n)`. Now look at the variable names in the function definition…

Profiling

1. `crossprod()` is faster (and also uses much less memory):

```
x <- matrix(rnorm(1000*250), 1000, 250)

system.time(t(x) %*% x)
```

```
##    user  system elapsed
##   0.004   0.008   0.006
```

```
system.time(crossprod(x))
```

```
##    user  system elapsed
##   0.004   0.004   0.002
```

2. Run:

```
Rprof()
example(glm)
Rprof(NULL)

summaryRprof()
```

in your R session.

3. Run:

```
Rprof(memory.profiling=TRUE)
example(glm)
Rprof(NULL)

summaryRprof(memory="both")
?summaryRprof ### help files
```

in your R session.

Benchmarking

1. `g()` is faster, because it preallocates the storage it needs:

```
library(rbenchmark)
n <- 1000
benchmark(f(n), g(n), columns=c("test", "replications", "elapsed", "relative"))
```

```
##   test replications elapsed relative
## 1 f(n)          100   0.153    1.889
## 2 g(n)          100   0.081    1.000
```

2. `i()` is faster, because it is vectorized:

```
library(rbenchmark)
n <- 1000
benchmark(h(n), i(n), columns=c("test", "replications", "elapsed", "relative"))
```

```
##   test replications elapsed relative
## 1 h(n)          100   0.066       66
## 2 i(n)          100   0.001        1
```