09 - High Performance Computing

Flemming Wu

2022-10-26

```
library(parallel)
```

Learning goals

In this lab, you are expected to learn/put in practice the following skills:

- Evaluate whether a problem can be parallelized or not.
- Practice with the parallel package.
- Use Rscript to submit jobs
- Practice your skills with Git.

Problem 1: Think

Give yourself a few minutes to think about what you just learned. List three examples of problems that you believe may be solved using parallel computing, and check for packages on the HPC CRAN task view that may be related to it.

- pvclust for hierarchical clustering with multiscale bootstraps
-

Problem 2: Before you

The following functions can be written to be more efficient without using parallel:

1. This function generates a n x k dataset with all its entries distributed Poisson with mean lambda.

```
set.seed(1235)
fun1 <- function(n = 100, k = 4, lambda = 4) { # returns matrix with 100 (n) rows and 4 (k) columns
  x <- NULL

  for (i in 1:n)
    x <- rbind(x, rpois(k, lambda))

  return(x)
```

```
}


fun1alt <- function(n = 100, k = 4, lambda = 4) { # create matrix all at once and populate with set num
  x <- matrix(rpois(n*k, lambda), ncol = 4)
}

# Benchmarking
microbenchmark::microbenchmark(
  fun1(),
  fun1alt()
)
```

```
## Unit: microseconds
##       expr     min      lq      mean   median       uq      max neval
##     fun1() 409.907 634.621 883.50258 685.6155 710.5525 21399.86   100
##  fun1alt()  27.467  30.215  60.06415  31.9105  34.7035  2659.48   100
```

The second approach is much faster.

2. Find the column max (hint: Checkout the function max.col()).

```
# Data Generating Process (10 x 10,000 matrix)
set.seed(1234)
x <- matrix(rnorm(1e4), nrow=10)

# Find each column's max value
fun2 <- function(x) {
  apply(x, 2, max)
}

fun2alt <- function(x) {
  # YOUR CODE HERE
}

# Benchmarking
microbenchmark::microbenchmark(
  fun2(),
  fun2alt()
)
```

```
fun2(x)
```

Problem 3: Parallelize Everything

We will now turn our attention to non-parametric (bootstrapping)[https://en.wikipedia.org/wiki/Bootstrapping_(statistics)]. Among its many uses, non-parametric bootstrapping allow us to obtain confidence intervals for parameter estimates without relying on parametric assumptions.

The main assumption is that we can approximate many experiments by resampling observations from our original dataset, which reflects the population.

This function implements the non-parametric bootstrap:

```
my_boot <- function(dat, stat, R, ncpus = 1L) {

  # Getting the random indices
  n <- nrow(dat)
  idx <- matrix(sample.int(n, n*R, TRUE), nrow=n, ncol=R)

  # Making the cluster using `ncpus`
  # STEP 1: GOES HERE
  # STEP 2: GOES HERE

    # STEP 3: THIS FUNCTION NEEDS TO BE REPLACES WITH parLapply
  ans <- lapply(seq_len(R), function(i) {
    stat(dat[idx[,i], , drop=FALSE])
  })

  # Coercing the list into a matrix
  ans <- do.call(rbind, ans)

  # STEP 4: GOES HERE

  ans

}
```

1. Use the previous pseudocode, and make it work with parallel. Here is just an example for you to try:

```
# Bootstrap of an OLS
my_stat <- function(d) coef(lm(y ~ x, data=d))

# DATA SIM
set.seed(1)
n <- 500; R <- 1e4

x <- cbind(rnorm(n)); y <- x*5 + rnorm(n)

# Checking if we get something similar as lm
ans0 <- confint(lm(y~x))
ans1 <- my_boot(dat = data.frame(x, y), my_stat, R = R, ncpus = 2L)

# You should get something like this
t(apply(ans1, 2, quantile, c(.025,.975)))
##                   2.5%      97.5%
## (Intercept) -0.1372435 0.05074397
## x            4.8680977 5.04539763
ans0
##                   2.5 %     97.5 %
## (Intercept) -0.1379033 0.04797344
## x            4.8650100 5.04883353
```

2. Check whether your version actually goes faster than the non-parallel version:

```
system.time(my_boot(dat = data.frame(x, y), my_stat, R = 4000, ncpus = 1L))
system.time(my_boot(dat = data.frame(x, y), my_stat, R = 4000, ncpus = 2L))
```

Problem 4: Compile this markdown document using Rscript

Once you have saved this Rmd file, try running the following command in your terminal:

```
Rscript --vanilla -e 'rmarkdown::render("[full-path-to-your-Rmd-file.Rmd]")' &
```

Where [full-path-to-your-Rmd-file.Rmd] should be replace with the full path to your Rmd file... :).