Week 14 Assignment

2024-04-23

Assignment

Purpose

The goal of this assignment is to practice using parallelization.

Task

Write R code to successfully answer each question below.

Criteria for Success

- Code is within the provided code chunks or new code chunks are created where necessary
- Code chunks run without errors
- Code chunks have brief comments indicating which code is answering which part of the question
- Code will be assessed as follows:
  - Produces the correct answer using the requested approach: 100%
  - Generally uses the right approach, but a minor mistake results in an incorrect answer: 90%
  - Attempts to solve the problem and makes some progress using the core concept, but returns the wrong answer and does not demonstrate comfort with the core concept: 50%
  - Answer demonstrates a lack of understanding of the core concept: 0%
- Any questions requiring written answers are answered with sufficient detail

Due Date

April 29 at midnight MST

Assignment Exercises

1. Set Up (10 points)

We will use a number of packages to complete this week's assignments. Load in the following packages: `tidyverse`, `foreach`, and `doParallel`.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(foreach)
```

```
##
## Attaching package: 'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
library(doParallel)
```

```
## Loading required package: iterators
## Loading required package: parallel
```

2. Use apply functions (30 points)

Let's practice using `lapply` and the mutli-core cousin, `mclapply`.

Run the following code chunk to create a list to use as our data object.

```
seq_list <- as.list(seq(1, 500, by = 5))
```

Use the `head` function to look at the `seq_list` object. You will notice that the output looks different from the vectors and dataframes that we are used to looking at. That is because it is a list, a collection of data objects. But don't worry! The code for `lapply` and `mclapply` is the same as in the lesson.

(a) Use `lapply` to take the square root of each number in `seq_list` (using the `sqrt` function). Then use `do.call` to covert the output into a dataframe. Use the `head` function to print the first 6 rows of the output.

```
output <- lapply(seq_list, sqrt)
output <- do.call(rbind, output)
head(output)
```

```
##          [,1]
## [1,] 1.000000
## [2,] 2.449490
## [3,] 3.316625
## [4,] 4.000000
## [5,] 4.582576
## [6,] 5.099020
```

(b) Use `mclapply` to do the same thing. Remember to set up the number of cores you will use, first.

```
nCores <- detectCores() /2

output2 <- mclapply(seq_list, sqrt, mc.cores = nCores)
output2 <- do.call(rbind, output2)
head(output)
```

```
##          [,1]
## [1,] 1.000000
## [2,] 2.449490
## [3,] 3.316625
## [4,] 4.000000
## [5,] 4.582576
## [6,] 5.099020
```

3. Use `foreach` (30 points)

Use the `source` function to bring in the `mass_from_length` function (from Week 12's assignment) in the `dino_allometry_fxn.R` script. You will also want to read in the `dinosaur_lengths.csv` file.

```
source("dino_allometry_fxn.R")

dino_data <- read_csv("../Week12_Iteration/data_raw/dinosaur_lengths.csv")
```

```
## Rows: 500 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): species
## dbl (1): lengths
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

(a) First, write out a `for` loop that uses the `mass_from_length` function and stores the results in an empty vector. This is the same as Question 2a from Week 12.

```
lengths <- dino_data$lengths
species <- dino_data$species
masses <- vector(length = length(lengths))
for (i in 1:length(lengths)){
    masses[i] <- mass_from_length(lengths[i], species[i])
}
head(masses)
```

```
## [1] 24341.68 27017.90 67453.38 22114.19 53884.76 52026.34
```

(b) Now, do the same thing, except use the `foreach` and `%do%` operator. Use `.combine = c` in place of the `.combine = rbind`.

```
results2 <- foreach::foreach(i = 1:nrow(dino_data), .combine = c) %do% {
  mass_from_length(dino_data$lengths[i],
                   dino_data$species[i])
}
head(results2)
```

```
## [1] 24341.68 27017.90 67453.38 22114.19 53884.76 52026.34
```

(c) Now, use parallel processing with the `%dopar%` operator. Remember to register and stop your cluster of cores.

```
nCores <- parallel::detectCores() / 2

doParallel::registerDoParallel()

results3 <- foreach::foreach(i = 1:nrow(dino_data), .combine = c) %dopar% {
  mass_from_length(dino_data$lengths[i],
                   dino_data$species[i])
}

doParallel::stopImplicitCluster()

head(results3)
```

```
## [1] 24341.68 27017.90 67453.38 22114.19 53884.76 52026.34
```

(d) Go back through all of your code for question 3 and explicitly call all of the functions that you use. The ones you've written yourself do not need to be explicilty called.

4. Forest Change through Time (30 points)

The following code chunk reads in data about how much forested area is in each country. You might need to edit the file path to match your project structure.

```
forest <- read_csv("forest_per_country.csv",
                   skip = 4, col_names = TRUE) %>%
  select(-`2022`)
```

```
## Rows: 266 Columns: 35
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): Country Name, Country Code
## dbl (32): 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, ...
## lgl  (1): 2022
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The `forest` data is not a fully cleaned dataset, especially when we look at the column names. You can either use the `rename()` function to rename them or encompass the names with back ticks in your `for` loops.

(a) Write a for loop that calculates the change in forest per country from 1990 to 2021 by subtracting the 1990 value from the 2021 value.

The output should be stored in a vector called `change_in_forest`. Print the head of `change_in_forest`.

```
change_in_forest <- vector(mode = "numeric", length = nrow(forest))

for (i in 1:nrow(forest)) {
  change_in_forest[i] <- forest$`2021`[i] - forest$`1990`[i]
}

head(change_in_forest)
```

```
## [1]       0.0 -832420.0       0.0 -275433.8 -132104.7       1.0
```

(b) Perform the same task as in (a) using the `foreach` function and the `%do%` operator. Use the `.combine = rbind` argument to save the output as a dataframe.

```
change_in_forest2 <- foreach(i = 1:nrow(forest), .combine = c) %do% {
  area <- forest$`2021`[i] - forest$`1990`[i]
}

head(change_in_forest2)
```

```
## [1]       0.0 -832420.0       0.0 -275433.8 -132104.7       1.0
```

(c) Do the same as in (b) but use parallel processing and the `%dopar%` operator.

```
registerDoParallel(nCores)

change_in_forest3 <- foreach(i = 1:nrow(forest), .combine = c) %dopar% {
  area <- forest$`2021`[i] - forest$`1990`[i]
}

stopImplicitCluster()

head(change_in_forest3)
```

```
## [1]       0.0 -832420.0       0.0 -275433.8 -132104.7       1.0
```