Run Code in Parallel in R

Fan Wang

2023-11-01

Contents

1 Parallel Loop in R

Go to the **RMD**, **R**, **PDF**, or **HTML** version of this file. Go back to fan's REconTools research support package, R4Econ examples page, PkgTestR packaging guide, or Stat4Econ course page.

1.1 Setting Up and First Run

First, install several packages.

```
install.packages(c("parallel", "doParallel", "foreach"))
```

Second, we load the libraries, and check on the parallel processing capacities on the local machine.

```
# Load libraries
library(dplyr)
library(readr)
library(tibble)
library(iterators)
library(parallel)
library(foreach)
library(doParallel)

# Check number of cores
it_n_cores_computer <- parallel::detectCores()
glue::glue("Number of cores on computers:{it_n_cores_computer}")
```

```
## Number of cores on computers:20
```

Third, we might want to use less than the total number of cores available. Specifying the number of cores to be used, we can initiate a local cluster. "PSOCK" below copies everything to each worker.

```
# Start cluster
ob_cluster <- parallel::makeCluster(
  it_n_cores_computer - 2,
  type = "PSOCK"
  )
# Register cluster
doParallel::registerDoParallel(cl = ob_cluster)
```

Fourth, run first parallel task, concurrent base-10 exponentiation.

```
# c(a,b,c,d) outputs together with combine
ar_test_parallel <- foreach(
  it_power = seq(1, 10), .combine = 'c'
) %dopar% {
  return(10^(it_power))
}
glue::glue("dopar outputs: {ar_test_parallel}")
```

```
## dopar outputs: 10
## dopar outputs: 100
## dopar outputs: 1000
## dopar outputs: 10000
## dopar outputs: 1e+05
## dopar outputs: 1e+06
## dopar outputs: 1e+07
## dopar outputs: 1e+08
## dopar outputs: 1e+09
## dopar outputs: 1e+10
```

Fifth, close cluster. When work is done, close the cluster.

```
parallel::stopCluster(cl = ob_cluster)
```

1.2 Parallel Function Run with Different Parameters, Aggregate Output Files

In this example, we create a function, we run the function with different parameters, each time generating a data output file to be stored, and then review results after.

First, we create a function. In this function, we generate a random matrix, the `it_nrow` parameter controls the number of rows in this random matrix. We store this matrix as csv.

Note, for each function used, such as `as_tibble` below, we should write it as `tibble::as_tibble`, to declare package and function jointly.

```
ffi_rand2csv <- function(
    spt_path_out,
    it_nrow = 3,
    st_file_prefix = "prefix") {

    # Generate a matrix and tibble
    mt_rnorm_a <- matrix(
      rnorm(it_nrow*3, mean=0, sd=1),
      nrow=it_nrow, ncol=3)
    tb_test <- tibble::as_tibble(mt_rnorm_a)

    # File output path
    spn_output_file <- file.path(
      spt_path_out,
      paste0(st_file_prefix, '_nrow', it_nrow, '.csv'),
      fsep = .Platform$file.sep)

    # Write file out
    readr::write_csv(tb_test, spn_output_file)
    print(glue::glue(
      "File saved successfully: ", spn_output_file))
```

```
}
```

Second, we initialize the cluster.

```
# Get the number of cores
it_n_cores_computer <- parallel::detectCores()
glue::glue("Number of cores on computers:{it_n_cores_computer}")
```

```
## Number of cores on computers:20
```

```
# Start cluster
ob_cluster <- parallel::makeCluster(
  it_n_cores_computer - 2,
  type = "PSOCK"
  )
# Register cluster
doParallel::registerDoParallel(cl = ob_cluster)
```

Third, we run the function in parallel.

```
# Define shared Path

spt_root <- "C:/Users/fan/"
spt_rmd <- "R4Econ/development/parallel/_file/"
spt_path_out <- file.path(spt_root, spt_rmd, fsep = .Platform$file.sep)

# Parallel Run
foreach(
  it_nrow = seq(2, 4)
) %dopar% {
  # Run function
  ffi_rand2csv(
    spt_path_out,
    it_nrow = it_nrow,
    st_file_prefix = "ffi_para_test")
}
```

```
## [[1]]
## File saved successfully: C:/Users/fan//R4Econ/development/parallel/_file/ffi_para_test_nrow2.csv
##
## [[2]]
## File saved successfully: C:/Users/fan//R4Econ/development/parallel/_file/ffi_para_test_nrow3.csv
##
## [[3]]
## File saved successfully: C:/Users/fan//R4Econ/development/parallel/_file/ffi_para_test_nrow4.csv
```

Fourth, adapting the parallel loop to other functions. Note that:

1. In the `forach` loop ablow, we iterate over `seq(2,4)`, assigning in parallel `2`, `3`, and `4` to the parameter `it_nrow`.
2. `it_nrow` is a parameter for the `ffi_rand2csv` function, so we will generate different outputs associated with `it_nrow=2`, `it_nrow=3`, and `it_nrow=4`.
3. The code above can be adapted to other functions that one wants to run in parallel by changing only one parameter of a function. For example, suppose we want to run `ffp_demo_loc_env_inequality(spt_path_data, fl_temp_bound=fl_temp_bound)`, where `spt_path_data` is common across parallel calls, but we want to update `fl_temp_bound` for each parallel call, then we need to iterate over `fl_temp_bound`. See example below:

```r
# Some path
spt_path_data <- "C:/Users/fan/"
# Parallel Run
foreach(
  fl_temp_bound = seq(-40, 40, by=1)
) %dopar% {
  # Run function
  ffp_demo_loc_env_inequality(
    spt_path_data,
    fl_temp_bound=fl_temp_bound)
}
```