

Turbocharging Computation Efficiency: A Tutorial on Hierarchical Parallel Computing in R

Mahmoud Ali Afifi¹, Samer Mouksassi^{1,2}

¹Certara, Radnor PA, USA

²University of California San Francisco, San Francisco CA, USA

Background

Modern modeling and data analysis require speed. Sequential processing slows exploration and delivery. Parallel computing—via R’s **foreach** and **doParallel**—unlocks efficiency by running tasks across multiple cores, streamlining complex workflows.

Objective

Introduce key parallel computing tools in R, demonstrate real-world use cases, and empower attendees to build scalable, efficient workflows for faster decision-making.

Methods

- Introducing how to recode for loops into parallelized foreach equivalent
- Compared toy implementations of *lapply*, *foreach*, *future_lapply*, *furrr*, and *parLapply* to demonstrate parallelization strategies in R.
- Sharing real-life project applications with benchmarks and impact

Case Study 1: Safety–Exposure Analysis

Use case: Assess relationships between multiple exposure metrics and dozens of adverse events across dose levels and covariates.
Why Parallelization: Stepwise modeling (e.g., forward/backward selection) is computationally demanding. Parallel execution dramatically boosts efficiency and cuts runtime.

10 X
Speed
up

Steps to Replace *for* Loops with *foreach*

STEP 1

Backend Initialization
Select and register a parallel backend suitable for the computing environment (e.g., doParallel for multicore execution).

```
library(doParallel)  
cl <- makeCluster(detectCores()-2) # keep some cpu for other tasks  
registerDoParallel(cl) # on windows default "PSOCK"
```

STEP 2

Define Nested Iteration Structure
Use the foreach package with %:% to construct hierarchically nested loops that reflect the structure of the computational problem (e.g., Subpopulations x AE terms x exposure metrics). %:% Automatically expand and flatten the nested simulation into parallel ones

```
results <- foreach(supp = subpopulations, .combine = 'list') %:%  
  foreach(ae = ae_terms, .combine = 'list') %:%  
    foreach(exp = exposure_metrics, .combine = 'list') %dopar% {  
      simulate_and_fit_model(supp, ae, exp,...) }  
    }
```

STEP 3

Ensure Reproducibility across cores
Apply the doRNG package to enable reproducible parallel execution by automatically managing random seed propagation

```
library(doRNG) # Replace %dopar% with %dorng% for reproducibility;  
They are identical %dorng% but enables seed control
```

STEP 4

Post-Processing and Resource Management
After computation, aggregate results, save outputs, and release computing resources.

```
stopCluster(cl)  
saveRDS(results, "model_output.rds")
```

Case Study 2: Vaccine Efficacy Estimation

Use case: Estimate efficacy using a Bayesian simulation-based framework across multiple virus subtypes, clinical endpoints, and trial scenarios.
Why parallelization: The approach loops over many subtype-endpoint-scenario combinations, each requiring simulation and Bayesian model fitting. Parallelization speeds up this intensive process.

20 X
Speed
up

Other parallelization packages

Heavy task function (replaceable with any real pharmacometric task)
cpu_heavy_task <- function(n = 5e7) { x <- runif(n) y <- runif(n) inside <- sum(x^2 + y^2 <= 1) 4 * inside / n }

parLapply (Base R Parallel)
1. Load base parallel and set up cluster
library(parallel)
cl <- makeCluster(4)
2. Export function to workers
clusterExport(cl, "cpu_heavy_task")

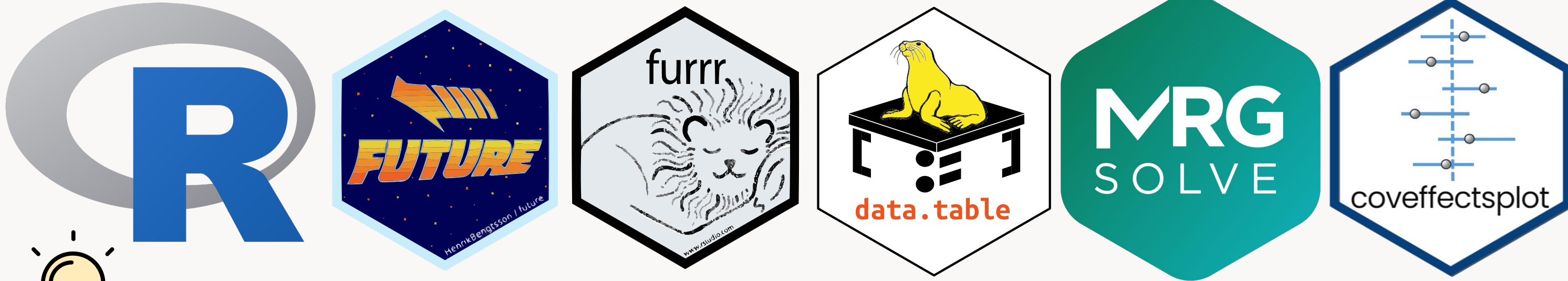
3. Run tasks in parallel
result <- parLapply(cl, 1:20, function(i)
cpu_heavy_task(i))
4. Clean up
stopCluster(cl)

furrr::future_map() (Tidyverse Style)
library(furrr)
plan(multisession, workers = 4)
Enable parallel backend
2. Run parallel purrr::map
result <- future_map(1:20,
~cpu_heavy_task())

3. Reset plan
plan(sequential)

future_lapply (Future Base)
1. Load future and set plan
library(future)
plan(multisession, workers = 4)
Choose parallel strategy
2. Use parallel lapply
result <- future_lapply(1:20,
function(i) cpu_heavy_task(i))

3. Reset plan
plan(sequential)



Tips and Tricks

- Before parallelizing over a local or cloud cluster the modeler should make every effort to optimize single core computations:
- Reduce memory usage but keeping only the needed data for the task
 - Perform **vectorized** operations whenever possible
 - Use efficient automatically parallel packages such as **data.table**
 - While in theory the more we use cores the better in practice we reach a saturation due to various factors such as input output of data between cores
 - Perform trials to understand the relationship between computation time vs n cores
 - One decision we often need to make is whether we parallelize over subject or replicates if you have n cores > replicates then parallelize all the replicates , usually we don’t want to parallelize simulations over subject unless we do not have many replicates or we have n cores > replicates X subjects

Tip	Description
Use .combine = smartly	Set .combine = rbind (or similar) to merge outputs cleanly. Avoid growing objects inside the loop.
Clean up clusters	Always call stopCluster(cl) after foreach loops to release system resources.
Plan your chunks and cores	Use parallelization for expensive computations. Avoid overhead on very fast/simple tasks.
Load packages inside foreach	Use .packages = c("mrgsolve", "data.table") to ensure each worker has access to needed libraries.
Seed carefully	Use set.seed() before parallel calls, or control RNG per iteration for reproducibility.
Profile first	Use bench::mark() or tic()/toc() to measure where time is spent before parallelizing.

Case Study 3: Covariate Effect Simulation

Use case: Simulate how covariates (e.g., race, weight, age) influence drug concentration by running the model 1000 times with varying individual parameters.
Why parallelization: Each run requires data preparation and ODE solving. Parallelization distributes simulations across CPU cores, significantly reducing runtime.

4 X
Speed
up

sequential (single-core, data.table)
nbsvsubjects <- 100000
idata <- data.table(ID=1:nbsvsubjects, WT=85, SEX=0, ALB=45)
ev1 <- ev(time = 0, amt = 100, cmt = 1)
data.dose <- ev(ev1)
data.dose <- setDT(as.data.frame(data.dose))
data.all <- data.table(idata, data.dose)
tic()
outputsim <- modcovsim %>%
 data_set(data.all) %>%
 mrgsim(end = 24, delta = 0.25) %>%
 as.data.frame %>%
 as.data.table
toc()
#23.15 sec elapsed

mrgsim.parallel
plan(multisession)
tic()
outputsim.future <-
 future_mrgsim_d(modcovsim,
 data.all, nchunk = 8L)
toc()
outputsim.future <-
 data.table(outputsim.future)
#6.1 sec elapsed

Compute exposures

data.table
tic()
refbsv <- outputsim[,
 derive.exposure(time, CP),
 by=.(ID, WT, SEX, ALB)]
toc()
Processed 100000 groups out of 100000. 100% done. Time elapsed: 7s. ETA: 0s.
#6.75s

dplyr
tic()
refbsv <- outputsim %>%
 group_by(ID, WT, SEX, ALB)
 %>%
 reframe(derive.exposure(time, CP))
toc()
#29.29 s

foreach, data.table and parallelization over replicates
cl <- makeCluster(detectCores() - 2) ;
registerDoParallel(cl)
nsim <- 1000
sim_parameters data.table holding uncertainty
iter_sims_parallel <- foreach(i = 1:nsim,
 .combine = rbind,
 .packages = c("data.table", "mrgsolve", "dplyr"))
 %dopar% {
 data.all <- data.table(idata, data.dose,
 sim_parameters[i])
 out <- modcovsim %>%
 data_set(data.all) %>%
 zero_re() %>%
 mrgsim(start = 0, end = 24, delta = 0.25) %>%
 as.data.table()
 out[, rep := i]
 out}
 }

References:

- Metrum Research Group. Parallel simulation in mrgsolve. GitHub Issue #1178. 2023. Available at: github.com/metrumresearchgroup/mrgsolve/issues/1178.
- Revolution Analytics. The foreach package: Foreach looping construct for R. CRAN vignette. 2023. Available at: cran.r-project.org/package=foreach.
- Henry L, Vaughan D. furrr: Apply mapping functions in parallel using futures. Futureverse Project. 2023. Available at: furrr.futureverse.org.
- Bengtsson H. future: Unified parallel and distributed processing in R for everyone. Futureverse Project. 2023. Available at: future.futureverse.org.