Julia as a Compiler for R Packages

Michael Kane - MD Anderson Cancer Center, Telperian Inc.

Why do I use R primarily?

R minimizes my development time

Unmatched in the following:

- syntactic ergonomics and language extensibility
- polyglot solution development - R for orchaestrating computations
- package ecosystem for some domains (statistics, clinical trials, etc.)

Ergonomics mean that we sometimes trade execution time for development time

Why I reconsidered julia

From the TheCedarPrince/InteroperableJuliaBinaries] and `juliac` scripts from here using `1.12.0-DEV.1314` (2024-10-06) at commit ab6df86f77b.

```julia
module JuliaTest

    Base.@ccallable function add_r(a::Ptr{Csize_t},
      b::Ptr{Csize_t},
      out::Ptr{Csize_t})::Csize_t
        a = unsafe_load(a)
        b = unsafe_load(b)
        out = unsafe_wrap(Array,
                          out::Ptr{Csize_t},
                          1::Int)
        out[1] = a + b
    end
end
```

Compile it and call it from R

```
bash> julia +nightly  juliac.jl --output-lib simple.so
  --compile-ccallable --experimental --trim simple.jl
bash> ls -lah simple.so
-rwxr-xr-x@ 1 mike  staff   1.0M Apr 11 12:09 simple.so
```

```
bash> LD_LIBRARY_PATH=. LD_PRELOAD=simple.so R
R> dyn.load("simple.so")
R> res <- .C('add_r',
             a = as.integer(2),
             b = as.integer(2),
             output = c(as.integer(0)))
R> res$output
4
```

Scaling Out vs. Up (Martin Schultz)

Scaling out (horizontal scalability, embarrasingly parallel problems) - perform the same computation on different data.

Scaling up (vertical scalability) make a single computation faster.

- Horizontal scalabling includes parallel apply functions, `foreach`, `furrr`, etc.
- Vertical scaling include writing better R code, `C/C++`, `Rcpp`, `torch`, etc.

Vertical scalability often means external computing solutions.

Why Julia how does it fit?

- Syntax is somewhere between R and Python but supports low(er)-level programming.
- It's fast - sometimes.
- Great metaprogramming via macros.
- Optional typing and built-in multiple dispatch.
- It can be compiled
- It has great GPU development tools

We (R-developers) should consider it a great option for scaling up.

Predictive distribution of clinical trial: R code

```
library(tibble)

# Function to simulate survival data for a clinical trial
simulate_clinical_trial <- function(n_subjects, hazard_ratio, follow_up_time) {
  lambda_control <- 0.05  # Hazard rate for the control group

  lambda_treatment <- lambda_control / hazard_ratio

  treatment <- rbinom(n_subjects, 1, 0.5)  # 50% treated

  survival_times <- numeric(n_subjects)
  event_indicators <- integer(n_subjects)

  for (i in 1:n_subjects) {
    lambda <- if (treatment[i] == 1) lambda_treatment else lambda_control

    t <- rexp(1, rate = lambda)
    survival_times[i] <- min(t, follow_up_time)  # Apply censoring
    event_indicators[i] <- ifelse(t <= follow_up_time, 1, 0)
  }

  tibble(
    SubjectID = 1:n_subjects,
    Treatment = treatment,
    SurvivalTime = survival_times,
    Event = event_indicators
  )
}
```

Predictive distribution of clinical trial: Julia code

```julia
using Random, Distributions, DataFrames, Survival

function simulate_clinical_trial(n_subjects::Int, hazard_ratio::Float64,
                                 follow_up_time::Float64)

     _control = 0.05  # Hazard rate for the control group

     _treatment =  _control / hazard_ratio  # Adjust hazard by the hazard ratio

    treatment = rand(Bernoulli(0.5), n_subjects)  # 50% treated

    survival_times = Float64[]
    event_indicators = Int[]  # 1 = event occurred, 0 = censored

    for i in 1:n_subjects
        # Assign hazard based on treatment group
          = treatment[i] == 1 ?  _treatment :  _control

        # Generate survival time using exponential distribution
        t = rand(Exponential( ))
        push!(survival_times, min(t, follow_up_time))  # Apply censoring
        push!(event_indicators, t <= follow_up_time ? 1 : 0)
    end

    DataFrame(
        SubjectID = 1:n_subjects,
        Treatment = treatment,
        SurvivalTime = survival_times,
        Event = event_indicators
    )
end
```

Simulation

Run for 1,000,000 subjects/patients.

Language	Lines of code	First Run (sec)	Second Run (sec)	Speedup
R	50	1.638	1.459	1.123
Julia	50	1.275	0.054	23.611

Summary

Julia is a great option for scaling up computations.

We are about to see much better shared object support.

- Can already (kind of) use `.C`
- Reflection of `SEXP` is available in `.Call` (code from Doug Bates)
- Julia becomes a viable compiler target for R packages.