If you are a user who needs to work with Rcpp-based packages, or you are a maintainer of one of such packages, you may be interested in the recent development of the unwind API, which can be leveraged to boost performance since [the last Rcpp update](http://dirk.eddelbuettel.com/blog/2018/07/23/#rcpp_0.12.18). In a nutshell, until R 3.5.0, every R call from C++ code was executed inside a try-catch, which is really slow, to avoid breaking things apart. From v3.5.0 on, this API provides a new and safe fast evaluation path for such calls.

**Some motivation**

Here is a small comparison of the old and the new APIs. The following toy example just calls an R function N times from C++. A pure R for loop is also provided as a reference.

Rcpp::cppFunction('

void old\_api(Function func, int n) {

for (int i=0; i)

Rcpp::cppFunction(plugins = "unwindProtect", '

void new\_api(Function func, int n) {

for (int i=0; i)

reference <- function(func, N) {

for (i in 1:N) func()

}

func <- function() 1

N <- 1e6

system.time(old\_api(func, N))

## user system elapsed

## 17.863 0.006 17.950

system.time(new\_api(func, N))

## user system elapsed

## 0.289 0.000 0.290

system.time(reference(func, N))

## user system elapsed

## 0.216 0.000 0.217

Obviously, there is still some penalty compared to not switching between domains, but the performance gain with respect to the old API is outstanding.

**A real-world example**

This is a quite heavy simulation of an M/M/1 system using simmer:

library(simmer)

system.time({

mm1 <- trajectory() %>%

seize("server", 1) %>%

timeout(function() rexp(1, 66)) %>%

release("server", 1)

env <- simmer() %>%

add\_resource("server", 1) %>%

add\_generator("customer", mm1, function() rexp(50, 60), mon=F) %>%

run(10000, progress=progress::progress\_bar$new()$update)

})

In my system, it takes around **17 seconds** with the old API. The new API makes it in **less than 5 seconds**. As a reference, if we avoid R calls in the timeout activity and precompute all the arrivals instead of defining a dynamic generator, i.e.:

system.time({

input <- data.frame(

time = rexp(10000\*60, 60),

service = rexp(10000\*60, 66)

)

mm1 <- trajectory() %>%

seize("server", 1) %>%

timeout\_from\_attribute("service") %>%

release("server", 1)

env <- simmer() %>%

add\_resource("server", 1) %>%

add\_dataframe("customer", mm1, input, mon=F, batch=50) %>%

run(10000, progress=progress::progress\_bar$new()$update)

})

then the simulation takes **around 2.5 seconds**.

**How to start using this feature**

First of all, you need R >= 3.5.0 and Rcpp >= 0.12.18 installed. Then, if you are a user, the easiest way to enable this globally is to add CPPFLAGS += -DRCPP\_USE\_UNWIND\_PROTECT to your ~/.R/Makevars. Packages installed or re-installed, as well as functions compiled with Rcpp::sourceCpp and Rcpp::cppFunction, will benefit from this performance gains. If you are a package maintainer, you can add -DRCPP\_USE\_UNWIND\_PROTECT to your package’s PKG\_CPPFLAGS in src/Makevars. Alternatively, there is a plugin available, so this flag can be enabled by adding [[Rcpp::plugins(unwindProtect)]] to one of your source files.

Note that this is fairly safe according to reverse dependency checks, but there might be still issues in some packages. But the sooner we start testing this feature and reporting possible issues, the sooner it will be enabled by default in Rcpp.