R Packages

Using C to Speed Up R

Jingyu Sun

March 29, 2025

Ocean University of China, High Performance Computing Club, Qingdao 266100

Why R Packages?

Normally we would use basic R packages that are currently provided by R, but sometimes we need some specific functions, and it is easy to do so.

Now we assume that you have already learned some basic grammar of C, in this slides we will use C to speed up R.

We have to admit that R is not fast as many languages, this would be interpreted later (compared with Julia).

If you still remember something that probably mentioned in C Programming Course, you would know that C allow programmers to use pointers, which is a powerful and almost unique tool to speed up the program.

Begin with a Simple Example

Suppose we have a function that calculate the sum of a vector, we can use the following code to do so:

```
sum_vector <- function(x) {
    sum <- 0
    for (i in 1:length(x)) {
        sum <- sum + x[i]
    }
    return(sum)
}
```

Begin with a Simple Example

Now we can use the following code to test the function:

```
x <- 1:1000000
sum_vector(x)
```

But at what cost? Let's use the following code to test the time:

```
system.time(sum_vector(x))
```

Begin with a Simple Example

If we try that in C, we can use the following code:

```
#include <R.h>
#include <Rinternals.h>
SEXP sum_vector(SEXP x) {
    int n = LENGTH(x);
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += REAL(x)[i];
    }
    return ScalarReal(sum);
}
```

What we found interesting here is that the code is not as the same as our previous C codes.

We have to include some header files, and we have to clarify our function differently, here we use SEXP.

To begin with, start downloading Rtools from `https://cran.r-project.org/bin/windows/Rtools/` or simply, we would just use RStudio command to do so.

Start Compiling

To start compiling, we can use the following command in RStudio:

```
system("R␣CMD␣SHLIB␣sum.c")
```

This command will generate a file called `sum.dll` in the current working directory.

Test Compilation

Now we can use the following code to test the function:

```
dyn.load("sum.dll")
.Call("sum_vector", as.numeric(x))
```

We can also use the following code to test the time:

```
system.time(.Call("sum_vector", as.numeric(x)))
```

Compare Efficiency

Also remember that the code we use here should use as.numeric since the input value of our written function is a SEXP, which is a special type of object in R.

Now if installed and compiled correctly, we can find the time of the C function is much faster than the R function.

Another Methods

Also, we can also use `.C()` in R to do so, but this method is not recommended mainly since it is not as fast and widely used as `.Call()`.

If we still have some time, we would like to try that or we would use this as some practicals.

More about R Extensions

R project provides a much detailed document to assist us to have own packages, you can find that on `https://cran.r-project.org/doc/manuals/r-devel/R-exts.html`.

This document actually focuses on how to write R packages, which is a little bit different from what we are doing here. But in general, the key things that we want to present here is that using R does not means low efficiency, sometimes it's even more efficiency.

Some Further Examples

Above we discussed how to run C code in R with basic operations and shows the standard steps to develop a C program, now some other examples here are also provided.

The most common places that we want the algorithm to be fast is for matrix operations, now consider we want two big matrices doing simple multiplication, we would see the time is faster than R code.

Some Further Examples: Matrix Multiplication

We can use the following code to test the time of R matrix multiplication:

```
A <- matrix(1:1000000, nrow = 1000, ncol = 1000)
B <- matrix(1:1000000, nrow = 1000, ncol = 1000)
C <- A %*% B
system.time(C)
```

Some Further Examples: Matrix Multiplication

Now consider implement that in C, we would use the code provided in the `multi.c` file.

Just imagine, the complexity should be $O(n^3)$, and we can use the following code to test the time:

```
A <- matrix(1:1000000, nrow = 1000, ncol = 1000)
B <- matrix(1:1000000, nrow = 1000, ncol = 1000)
C <- .Call("multi", A, B)
system.time(C)
```

Some Further Examples: Matrix Multiplication

Now we might see that the R code itself is much more efficient, so the key we want to show in this example is that choose the method wisely.

About Julia

Actually there is a new language called Julia, which is similar to R, but much faster. This language changed the logic when designing, that is, the logic of the compiler is somehow different when using it.

Julia is a new language that is designed for high performance and many of its language features are similar with R, so learning R would be much benifical for using Julia in the future.

In later courses, we would like to use Julia and introduce its features.