OPTIMIZING R CODE WITH RCPP

Welcome to the course

Romain François
Consulting Datactive, ThinkR

R vs C++

R

- Great, flexible
- Slow
- Interpreted

C++

- Compiled
- More difficult
- Fast

Motivation

- Use Rcpp to make your code faster
- No need to know all of C++
- Focus on writing simple C++ functions

Course structure

- Introduction - basic C++ syntax
- C++ functions and control flow
- Vector classes
- Case studies

Measure performance

☐ loopy version of `max`

```
slowmax <- function(x){
   res <- x[1]
   for ( i in 2:length(x) ){
       if( x[i] > res ) res <- x[i]
   }
   res
}
```

Comparing performance with the `microbenchmark` ☐

```
library(microbenchmark)

x <- rnorm(1e6)
microbenchmark( slowmax(x), max(x) )
Unit: milliseconds
       expr       min        lq      mean    median        uq       max neval
 slowmax(x) 31.649452 34.29454 36.344912 35.435299 37.188249 90.363038   100
    max(x)  1.563559  1.74036  1.939367  1.847045  2.014684  3.340052   100
```


DataCamp
PAID COURSE
Writing Efficient R Code
Start Course For Free
Play Intro Video

Evaluating simple C++ expressions with evalCpp

```
library(Rcpp)

evalCpp( "40 + 2" )
42

evalCpp( "exp(1.0)" )
2.718282

evalCpp( "sqrt(4.0)" )
2
```

Using `std::numeric_limits<int>::max()` to get the biggest representable

32 bit signed integer, aka `int`

```
evalCpp( "std::numeric_limits<int>::max()")
2147483647

2^31-1
2147483647
```

Basic number types

C++ has rich set of number types

- Integer numbers: `int`
- Floating point numbers: `double`

Basic number types

R

Literal numbers are `double`

```
x <- 42

storage.mode(x)
"double"
```

Integers need the `L` suffix

```
y <- 42L
storage.mode(y)
"integer"

z <- as.integer(42)
storage.mode(z)
"integer"
```

C++

Suffixing with `.0` forces a `double`

```
y <- evalCpp( "42.0" )

storage.mode(y)
"double"
```

Literal integers are `int`

```
library(Rcpp)

# Literal integers are `int`
x <- evalCpp( "42" )

storage.mode(x)
"integer"
```

Casting

Explicit casting with `(double)`

```
# Explicit cast
y <- evalCpp( "(double)(40 + 2)" )

storage.mode(y)
"double"
```

Beware of the integer division

```
# Integer division
evalCpp( "13 / 4" )
3

# Explicit cast, and hence use of double division
evalCpp( "(double)13 / 4" )
3.25
```

```
# Automatic conversion in R
13L / 4L
3.25
```

OPTIMIZING R CODE WITH RCPP

Let's practice!

OPTIMIZING R CODE WITH RCPP

Inline functions with cppFunction

Romain François
Consulting Datactive, ThinkR

Limits of evalCpp

```
evalCpp("40 + 2")
42

evalCpp("PI")
3.141593

evalCpp("exp(1)")
2.718282
```

cppFunction

Define a C++ function with `cppFunction()`

```
library(Rcpp)

cppFunction("int fun(){
   int x = 37 ;
   return x ;
}" )
```

Call that function from R

```
fun()

37
```

Internal things Rcpp does on your behalf

```
cppFunction("int fun(){
   int x = 37 ;
   return x ;
}", verbose = TRUE )

Generated code for function definition:
------------------------------------------------------------

#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
int fun(){
  int x = 37 ;
  return x ;
}

Generated extern "C" functions
------------------------------------------------------------


#include <Rcpp.h>
// fun
int fun();
RcppExport SEXP sourceCpp_1_fun() {
BEGIN_RCPP
    Rcpp::RObject rcpp_result_gen;
    Rcpp::RNGScope rcpp_rngScope_gen;
    rcpp_result_gen = Rcpp::wrap(fun());
    return rcpp_result_gen;
END_RCPP
}
```

Internal things Rcpp does on your behalf

```
Generated R functions
-------------------------------------------------------

`.sourceCpp_1_DLLInfo` <- dyn.load('/private/var/folders/r_/1b2gjtsd7j92jbbpz4t7ps340000gn/T/Rtmpl
-x86_64-apple-darwin15.6.0-0.12.16/sourcecpp_4bb22c766031/sourceCpp_2.so')

fun <- Rcpp:::sourceCppFunction(function() {}, FALSE, `.sourceCpp_1_DLLInfo`, 'sourceCpp_1_fun')

rm(`.sourceCpp_1_DLLInfo`)

Building shared library
-------------------------------------------------------

DIR: /private/var/folders/r_/1b2gjtsd7j92jbbpz4t7ps340000gn/T/Rtmpl8dL6H/sourceCpp-x86_64-apple-da
.16/sourcecpp_4bb22c766031

/Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB -o 'sourceCpp_2.so'  'file4bb247d077c.cp
clang++  -I/Library/Frameworks/R.framework/Resources/include -DNDEBUG   -I"/Library/Frameworks/R.f
ons/3.4/Resources/library/Rcpp/include" -I"/private/var/folders/r_/1b2gjtsd7j92jbbpz4t7ps340000gn/
sourceCpp-x86_64-apple-darwin15.6.0-0.12.16" -I/usr/local/include   -fPIC  -Wno-unused-result -Wno
-namespace -O3 -c file4bb247d077c.cpp -o file4bb247d077c.o
clang++ -dynamiclib -Wl,-headerpad_max_install_names -undefined dynamic_lookup -single_module -mul
suppress -L/Library/Frameworks/R.framework/Resources/lib -L/usr/local/lib -o sourceCpp_2.so file4b
-F/Library/Frameworks/R.framework/.. -framework R -Wl,-framework -Wl,CoreFoundation
```

cppFunction

Define a C++ function with `cppFunction()`

```
library(Rcpp)

cppFunction("int fun(){
   int x = 37 ;
   return x ;
}" )
```

Call that function from R.

```
fun()

37
```

Dynamic vs Static

R is **dynamically** typed

```
x <- "hello"

x
"hello"

typeof(x)
"character"

x <- 42L

x
42

typeof(x)
"integer"
```

C++ is **statically** typed.

```
// Define x as a double

double x = 42.0 ;

// Cannot redefine it as an int
// ----> that would not compile

int x = 14 ;
```

Type declaration of function arguments and return value

```
    types of arguments x and y: double
                |          |
                v          v
```

```
double add( double x, double y ){

    // body of the function
    // ...

}
```

Type declaration of function arguments and return value

```
 return type: double
    |
    v
```

```
double add( double x, double y ){

    // body of the function
    // ...
    // ...
}
```

Type declaration of function arguments and return value

```
 return type: double
  |
  |  name of the function: add
  |   |
  |   |    types of arguments x and y: double
  |   |      |          |
  |   |      |          |
  v   v      v          v
```

```
double add( double x, double y ){

    // body of the function
    // ...
    // ...
}
```

An addition example

C++ function to add two `double`.

```
cppFunction( "
double add( double x, double y){
    double res = x + y ;
    return res ;
}
" )

add( 30, 12 )
```

Equivalent R code

```
addr <- function(x, y) {
    res <- x + y
    res
}
```

OPTIMIZING R CODE WITH RCPP

Let's practice!

OPTIMIZING R CODE WITH RCPP

Debugging

Romain François
Consulting Datactive, ThinkR

Print to the console

A function that uses `Rprintf()` to print a message on the console

```
cppFunction( '
int fun(){

    // Some values
    int x = 42 ;

    // Printing a message to the R console
    Rprintf( "some message in the console, x=%d\\n", x ) ;

    // Return some int
    return 76 ;
}
')
```

Rprintf and placeholders

Integer placeholder with `%d`

```
int x = 42 ;

Rprintf( "some message in the console, x=%d\n", x ) ;

// Prints the following in the console:
some message in the console, x=42
```

String placeholder with `%s`

```
Rprintf( "roses are %s, violets are %s\n", "red", "blue" ) ;

// Prints the following:
roses are red, violets are blue
```

Print to the console

A function that uses `Rprintf()` to print a message on the console

```
cppFunction( '
int fun(){
    // some values
    int x = 42 ;

    // printing a message to the R console
    Rprintf( "some message in the console, x=%d\\n", x ) ;

    // return some int
    return 76 ;
}
')
```

Calling the function

```
fun()
some message in the console, x=42
76
```

Error messages

A function that only takes numbers between 0 and 20

```
cppFunction( 'int fun(int x){
    // A simple error message
    if( x < 0 ) stop( "sorry x should be positive" ) ;

    // A formatted error message
    if( x > 20 ) stop( "x is too big (x=%d)", x ) ;

    // Return some int
    return x ;
}')
```

Calling the function

```
fun(-2)
Error in fun(-2) : sorry x should be positive

fun(23)
Error in fun(23) : x is too big (x=23)

tryCatch( fun(24), error = function(e){
    message("C++ exception caught: ", conditionMessage(e))
})
C++ exception caught: x is too big (x=24)
```

OPTIMIZING R CODE WITH RCPP

Let's practice!