Speeding up R with Rcpp

Stephen Cristiano
Department of Biostatistics
Johns Hopkins University

November 27, 2018

What is Rcpp?

- Rcpp: Seamless integration between R and C++.
- Extremely simple to connect C++ with R.
- Maintained by Dirk Eddelbuetter and Romain Francois

Simple examples

```
library('Rcpp')
cppFunction('int square(int x) { return x*x; }')
square(7L)

## [1] 49

cppFunction('
            int add(int x, int y, int z) {
                int sum = x + y + z;
                return sum;
            }'
            )
add(1, 2, 3)

## [1] 6
```

Everything revolves around .Call

C++ Level:

```
SEXP foo(SEXP a, SEXP b, SEXP C, ...);
```

R Level:

```
res <- .Call("foo", a, b, C, ..., package="mypkg")
```

Why C++?

- One of the most frequently used programming languages. Easy to find help.
- Speed.
- Good chance what you want is already implemented in C++.
- From wikipedia: 'C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose, powerful programming language.'

Why not C++?

- More difficult to debug.
- more difficult to modify.
- The population of potentials users who understand both R and C++ is smaller.

Why Rcpp

- Easy to use (honest).
- Clean and approachable API that enable for high performance code.
- R style vectorized code at C++ level.
- Programmer time vs computer time: much more efficient code that does not take much longer to write.
- Enables access to advanced data structures and algorithms implented in C++ but not provided by R.
- Handles garbage collection and the Rcpp programmer should never have to worry about memory allocation and deallocation.

C++ in 2 minutes

```
cppFunction('
  double sumC(NumericVector x) {
    int n = x.size();
    double total = 0;
    for(int i = 0; i < n; ++i) {
      total += x[i];
    if(total > 100)
    break;
    }
    return total;
  }
')
sumC(seq(1:10))
```

- Need to initialize your variables with data type.
- for loops of structure for(initialization; condition; increment).
- conditionals are the same as R.
- End every statement with a semicolon.
- Vectors and arrays are 0-indexed.
- size() is a member function on the vector class - x.size() returns the size of x.
- While C++ can be a very complex language, just knowing these will enable you to write faster R functions.

Typical bottlenecks in R

- Loops that depend on previous iterations, eg MCMC methods.
- Function calls in R slow, but very little overhead in C++. Recursive functions are very inefficient in R.
- Not having access to advanced data structures algorithms in R but available in C++.

When to use Rcpp

- Sometimes the solution is to become a better R coder.
- Before writing C++ code, you should first ask if it's necessary.
- Take advantage of vectorization when possible.
- Most base R functions already call C functions. Make sure there isn't already an efficient implementation of what you are trying to do.

Data Structures

- All R objects are internally represented by a SEXP: a pointer to an S expression object.
- Any R object can be passed down to C++ code: vectors, matrices lists. Even functions and environments.
- A large number of user-visible classes for R objects, which contain pointers the the SEXP object.
  - IntegerVector
  - NumericVector
  - LogicalVector
  - CharacterVector
  - NumericMatrix
  - S4
  - and many more

Rcpp Sugar

- Rcpp sugar brings a higher level of abstraction to C++ code written in Rcpp.
- Avoid C++ loops with code that strongly resembles R.
- Takes advantage of operator overloading.
- Despite the similar syntax, peformance is much faster in C++, though not quite as fast as manually optimized C++ code.

Example

```
pdistR <- function(x, ys) {
    (x - ys)^2
}

cppFunction('NumericVector pdistC2(double x, NumericVector ys) {
            return pow((x-ys), 2);
}'
)
pdistR(5.0, c(4.1,-9.3,0, 13.7))

## [1]   0.81 204.49  25.00  75.69

pdistC2(5.0, c(4.1,-9.3,0, 13.7))

## [1]   0.81 204.49  25.00  75.69
```

Logical Operators

```
// two integer vectors of the same size
NumericVector x;
NumericVector y;

// expressions involving two vectors
LogicalVector res = x < y;
LogicalVector res = x != y;

// one vector, one single value
LogicalVector res = x < 2;

// two expressions
LogicalVector res = (x + y) == (x*x);

// functions producing single boolean result
all(x*x < 3);
any(x*x < 3);
```

Logical Operators

There are many functions similar to what exists inside R

```
is_na(x);
seq_along(x);
sapply( seq_len(10), square<int>() );
ifelse( x < y, x, (x+y)*y );
pmin( x, x*x);
diff( xx );
intersect( xx, yy); //returns interserct of two vectors
unique( xx ); // subset of unique values in input vecto

// math functions
abs(x); exp(x); log(x); ceil(x);
sqrt(x); sin(x); gamma(x);
range(x);
mean(x); sd(x); var(x);
which_min(x); which_max(x);
// A bunch more
```

Density and random number generation functions

Rcpp has access to the same density, distribution, and RNG functions used by R itself. For example, you can draw from a gamma distribution with scale and shape parameters equal to 1 with:

```
cppFunction('NumericVector getRGamma() {
            RNGScope scope;
            NumericVector x = rgamma( 10, 1, 1 );
            return x;
}'
)
getRGamma()

##  [1] 0.6720068 0.3489705 0.3887167 2.1880637 0.4933245 1.8656011 2.3
##  [8] 4.8920221 1.0502233 0.8761420
```

RcppArmadillo

- Armadillo is a high level and easy to use C++ linear algebra library with syntax similar to Matlab.
- RcppArmadillo is an Rcpp interface allowing access to the Armadillo library.

Be careful with pointers!

```
library(inline, quietly=TRUE)
src <- '
    Rcpp::NumericVector invec(vx);
    Rcpp::NumericVector outvec(vx);
    for(int i=0; i<invec.size(); i++) {
        outvec[i] = log(invec[i]);
    }
    return outvec;
'
fun <- cxxfunction(signature(vx="numeric"), src, plugin="Rcpp")
x <- seq(1.0, 3.0, by=1)
cbind(x, fun(x))

##               x
## [1,] 0.0000000 0.0000000
## [2,] 0.6931472 0.6931472
## [3,] 1.0986123 1.0986123
```

Note: outvec and invec point to the same underlying R object.

Use clone to not modify original vector.

```
src <- '
    Rcpp::NumericVector invec(vx);
    Rcpp::NumericVector outvec = Rcpp::clone(vx);
    for(int i=0; i<invec.size(); i++) {
        outvec[i] = log(invec[i]);
    }
    return outvec;
'
fun <- cxxfunction(signature(vx="numeric"), src, plugin="Rcpp")
x <- seq(1.0, 3.0, by=1)
cbind(x, fun(x))

##       x
## [1,] 1 0.0000000
## [2,] 2 0.6931472
## [3,] 3 1.0986123
```

Creating R packages

Inspection of R source code for any R package will reveal the directories::

- R: for R functions
- vignettes: LaTeXpapers weaving R code and indicating the intended workflow of an analysis.
- man: documentation for exported R functions.
- src: compiled code

The file `DESCRIPTION` provides a brief description of the project, a version number, and any packages for which your package depends.

Creating R packages

- All compiled code coes in package/src directory.
- Code in src/ will be automatically compiled and shared libraries created when building the package.
- Instantiate an Rcpp package: Rcpp.package.skeleton

S4 objects with Rcpp

```
src <- '
S4 foo(x) ; foo.slot(".Data") = "bar" ; foo.slot("x")=100; return(foo);
'
fun <- cxxfunction(signature(x="any"), src,
                   plugin="Rcpp")
setClass( "S4ex", contains = "character",
         representation( x = "numeric" ) )
x <- new( "S4ex", "bla", x = 10 )
fun(x)

## An object of class "S4ex"
## [1] "bar"
## Slot "x":
## [1] 100

str(fun(x))

## Formal class 'S4ex' [package ".GlobalEnv"] with 2 slots
##   ..@ .Data: chr "bar"
##   ..@ x    : int 100
```

Case study

Example: Gibbs sampler to find posterior distributions for parameters in mixture of Skew Normal distributions of the form:

$$\sum_{k=1}^{K} \pi_k f_{SN}(y; \xi_k, \omega_k^2, \alpha_k) \tag{1}$$

where

$$f_{SN}(y; \xi, \omega^2, \alpha) = \frac{2}{\omega} \phi\left(\frac{y - \xi}{\omega}\right) \Phi(\alpha\omega^{-1}(y - \xi)) \tag{2}$$

See Früwirth-Schnatter, Pyne (2010) for details on how to derive the full conditionals.

`github.com/scristia/ComputingClubRcpp` for Rcpp implementation.

Resources

- `vignette("Rcpp-quickref")`
- 'Seamless R and C++ integration with Rcpp' by Dirk Eddelbuettel. Excellent book for learning Rcpp. Available for free through JHU library.
- Hadley Wickham's Rcpp tutorial: http://adv-r.had.co.nz/Rcpp.html
- A huge number of examples at http://gallery.rcpp.org
- Stack exchange.