Rcpp by Examples

Dr. Dirk Eddelbuettel
edd@debian.org
dirk.eddelbuettel@R-Project.org

Workshop preceding R/Finance 2013
University of Illinois at Chicago
17 May 2013
Outline

1. Introduction
2. Usage
3. Sugar
4. Examples
5. RInside
6. More
A “vision” from Bell Labs from 1976

Source: John Chambers’ talk at Stanford in October 2010; personal correspondence.
An Introductory Example

Consider a function defined as

\[ f(n) \text{ such that } \begin{cases} 
   n & \text{when } n < 2 \\
   f(n - 1) + f(n - 2) & \text{when } n \geq 2 
\end{cases} \]
An Introductory Example: Simple R Implementation

R implementation and use:

```r
f <- function(n) {
  if (n < 2) return(n)
  return(f(n-1) + f(n-2))
}

## Using it on first 11 arguments
sapply(0:10, f)

## [1]  0  1  1  2  3  5  8 13 21 34 55
```
Timing:

```r
library(rbenchmark)
benchmark(f(10), f(15), f(20))[,1:4]
```

```
# test     replications elapsed relative
## 1 f(10) 100 0.034 1.00
## 2 f(15) 100 0.376 11.06
## 3 f(20) 100 4.280 125.88
```
An Introductory Example: C++ Implementation

```c
int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2));
}
```

Deployed as:

```r
library(Rcpp)
cppFunction('int g(int n) { if (n < 2) return(n); return(g(n-1) + g(n-2)); }')
## Using it on first 11 arguments
sapply(0:10, g)
## [1] 0 1 1 2 3 5 8 13 21 34 55
```
Timing:

```r
library(rbenchmark)
benchmark(f(20), g(20))[,1:4]
```

```r
##    test  replications elapsed relative
## 1 f(20)       100   4.126    589.4
## 2 g(20)       100   0.007     1.0
```

A nice 600-fold gain.
Well-know packages using Rcpp

**Amelia** by Gary King et al: Multiple Imputation from cross-section, time-series or both; uses Rcpp and RcppArmadillo

**forecast** by Rob Hyndman et al: Time-series forecasting including state space and automated ARIMA modeling; uses Rcpp and Armadillo

**RStan** by Andrew Gelman et al: Rcpp helps with automatic model parsing / generation for MCMC / Bayesian modeling

**rugarch** by Alexios Ghalanos: Sophisticated financial time series models using Rcpp and RcppArmadillo

**bigviz** by Hadley Wickham: High-performance visualization of datasets in the 10-100 million observations range
Type mapping

Standard R types (integer, numeric, list, function, ... and compound objects) are mapped to corresponding C++ types using extensive template meta-programming – it just works:

```r
library(Rcpp)
cppFunction("
    NumericVector logabs(NumericVector x) {
        return log(abs(x));
    }
"

logabs(seq(-5, 5, by=2))
```

```
## [1] 1.609 1.099 0.000 0.000 1.099 1.609
```

Also note: vectorized C++!
Type mapping also with C++ STL types

Use of `std::vector<double>` and STL algorithms:

```cpp
#include <Rcpp.h>
using namespace Rcpp;

inline double f(double x) { return ::log(::fabs(x)); }

// [[Rcpp::export]]
std::vector<double> logabs2(std::vector<double> x) {
    std::transform(x.begin(), x.end(), x.begin(), f);
    return x;
}
```
Type mapping also with C++ STL types

Used via

```r
library(Rcpp)
sourceCpp("code/logabs2.cpp")
logabs2(seq(-5, 5, by=2))
```

```r
## [1] 1.609 1.099 0.000 0.000 1.099 1.609
```
Simple outer product of a column vector (using Armadillo / RcppArmadillo):

```r
cppFunction("arma::mat v(arma::colvec a) {return a*a.t();}", depends="RcppArmadillo")
v(1:5)
```

```
## [1,]  1  2  3  4  5
## [2,]  2  4  6  8 10
## [3,]  3  6  9 12 15
## [4,]  4  8 12 16 20
## [5,]  5 10 15 20 25
```

This uses implicit conversion via `as<>` and `wrap` – cf package vignette Rcpp-extending.
**Basic Usage:** `evalCpp`

`evalCpp()` evaluates a single C++ expression. Includes and dependencies can be declared. This allows us to quickly check C++ constructs.

```r
evalCpp("std::numeric_limits<double>::max()")
```

```r
## [1] 1.798e+308
```
`cppFunction()` creates, compiles and links a C++ file, and creates an R function to access it.

```r
cppFunction("
    int useCpp11() {
        auto x = 10;
        return x;
    }
")
```

```r
useCpp11()  # same identifier as C++ function
```

```r
## [1] 10
```

Dirk Eddelbuettel

Rcpp by Examples
**Basic Usage:** `sourceCpp()`

`sourceCpp()` is the actual workhorse behind `evalCpp()` and `cppFunction()`. It is described in more detail in the package vignette `Rcpp-attributes`.

`sourceCpp()` builds on and extends `cxxfunction()` from package `inline`, but provides even more ease-of-use, control and helpers – freeing us from boilerplate scaffolding.

A key feature are the plugins and dependency options: other packages can provide a plugin to supply require compile-time parameters (cf `RcppArmadillo`, `RcppEigen`, `RcppGSL`).
Package are *the* standard unit of R code organization.

Creating packages with Rcpp is easy; an empty one to work from can be created by `Rcpp.package.skeleton()`.

The vignette *Rcpp-package* has fuller details.

As of April 2013, there are 110 packages on CRAN which use Rcpp, and a further 10 on BioConductor — with working, tested, and reviewed examples.
Outline

1. Introduction
2. Usage
3. Sugar
4. Examples
5. RInside
6. More
Syntactic 'sugar': Simulating $\pi$ in R

Basic idea: for point $(x, y)$, compute distance to origin. Do so repeatedly, and ratio of points below one to number N of simulations will approach $\pi/4$ as we fill the area of one quarter of the unit circle.

```r
piR <- function(N) {
  x <- runif(N)
  y <- runif(N)
  d <- sqrt(x^2 + y^2)
  return(4 * sum(d <= 1.0) / N)
}

set.seed(5)
sapply(10^(3:6), piR)
```

```r
```
The neat thing about Rcpp sugar enables us to write C++ code that looks almost as compact.

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double piSugar(const int N) {
    RNGScope scope;  // ensure RNG gets set/reset
    NumericVector x = runif(N);
    NumericVector y = runif(N);
    NumericVector d = sqrt(x*x + y*y);
    return 4.0 * sum(d <= 1.0) / N;
}
```

Apart from RNG set/reset, the code is essentially identical.
And by using the same RNG, so are the results.

```r
sourceCpp("code/piSugar.cpp")
set.seed(42); a <- piR(1.0e7)
set.seed(42); b <- piSugar(1.0e7)
identical(a,b)

## [1] TRUE

print(c(a,b), digits=7)

## [1] 3.140899 3.140899
```
The performance is close with a small gain for C++ as R is already vectorised:

```r
library(rbenchmark)
benchmark(piR(1.0e6), piSugar(1.0e6))[,1:4]
```

```
## test replications elapsed relative
## 1 piR(1e+06) 100 13.540 1.76
## 2 piSugar(1e+06) 100 7.695 1.00
```

More about Sugar is in the package vignette Rcpp-sugar.
A basic looped version:

```cpp
#include <Rcpp.h>
#include <numeric> // for std::partial_sum
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector cumsum1(NumericVector x){
    // initialize an accumulator variable
    double acc = 0;

    // initialize the result vector
    NumericVector res(x.size());

    for(int i = 0; i < x.size(); i++){
        acc += x[i];
        res[i] = acc;
    }
    return res;
}
```
An STL variant:

```cpp
// [[Rcpp::export]]
NumericVector cumsum2(NumericVector x){
    // initialize the result vector
    NumericVector res(x.size());
    std::partial_sum(x.begin(), x.end(),
                     res.begin());
    return res;
}
```
Or just sugar:

```cpp
// [[Rcpp::export]]
NumericVector cumsum_sug(NumericVector x){
    return cumsum(x); // compute + return result vector
}
```

Of course, all results are the same.

```r
cppFunction('NumericVector cumsum_sug(NumericVector x) { return cumsum(x); }')
x <- 1:10
all.equal(cumsum_sug(x), cumsum(x))
```
```
## [1] TRUE
```
#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]

using namespace Rcpp;
using namespace arma;

// [[Rcpp::export]]
mat matrixSubset(mat M) {
    // logical condition:
    // where is transpose larger?
    umat a = trans(M) > M;
    mat N = conv_to<mat>::from(a);
    return N;
}
M <- matrix(1:9, 3, 3)

M

## [,1] [,2] [,3]
## [1,] 1 4 7
## [2,] 2 5 8
## [3,] 3 6 9

matrixSubset(M)

## [,1] [,2] [,3]
## [1,] 0 0 0
## [2,] 1 0 0
## [3,] 1 1 0
Armadillo subsetting


```cpp
#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
arma::vec matrixSubset2(arma::mat M) {
    arma::mat Z = M * M.t();
    arma::vec v = Z.elem(arma::find(Z >= 100));
    return v;
}
```

**matrixSubset2(M)**

```r
matrixSubset2(M)
```

```r
## [,1]  
## [1,] 108 
## [2,] 108 
## [3,] 126 
```
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
NumericVector callFunction(NumericVector x, Function f) {
    NumericVector res = f(x);
    return res;
}

/*** R
callFunction(x, fivenum)
*/
#include <Rcpp.h>

using namespace Rcpp;

// Important: enable C++11 via plugin
// [[Rcpp::plugins("cpp11")]]

// [[Rcpp::export]]
std::vector<double>
transformEx(const std::vector<double>& x) {
    std::vector<double> y(x.size());
    std::transform(x.begin(), x.end(), y.begin(),
                   [](double x) { return x*x; } );
    return y;
}
/ **[[Rcpp::depends](BH)]**

```c++
#include <Rcpp.h>

// One include file from Boost
#include <boost/date_time/gregorian/gregorian_types.hpp>

using namespace boost::gregorian;

// **[[Rcpp::export]]**
Rcpp::Date getIMMDate(int mon, int year) {
    // compute third Wednesday of given month / year
    date d = nth_day_of_the_week_in_month(
        nth_day_of_the_week_in_month::third,
        Wednesday, mon).get_date(year);
    date::ymd_type ymd = d.year_month_day();
    return Rcpp::wrap(Rcpp::Date(ymd.year, ymd.month, ymd.day));
}
```
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
double takeLog(double val) {
    try {
        if (val <= 0.0) { // log() not defined here
            throw std::range_error("Inadmissible value");
        }
        return log(val);
    } catch (std::exception &ex) {
        forward_exception_to_r(ex);
    } catch (...) {
        ::Rf_error("c++ exception (unknown reason)");
    }
    return NA_REAL; // not reached
}
Using Exceptions

See http://gallery.rcpp.org/articles/intro-to-exceptions/

```r
takeLog(exp(1))     # works

## [1] 1

takeLog(-1.0)        # throws exception

## Error: Inadmissible value

takeLog(exp(2))      # but carries on

## [1] 2
```
#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
arma::vec getEigenValues(arma::mat M) {
    return arma::eig_sym(M);
}
```r
set.seed(42)
X <- matrix(rnorm(4*4), 4, 4)
Z <- X %*% t(X)
getEigenValues(Z)

## [,1]
## [1,] 0.3319
## [2,] 1.6856
## [3,] 2.4099
## [4,] 14.2100

# R gets the same results (in reverse)
# and also returns the eigenvectors.
```
#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]

using namespace Rcpp;

// [[Rcpp::export]]
arma::mat mvrnormArma(int n, arma::vec mu, arma::mat sigma) {
  int ncols = sigma.n_cols;
  arma::mat Y = arma::randn(n, ncols);
  return arma::repmat(mu, 1, n).t() +
         Y * arma::chol(sigma);
}
// the embedded R via RInside
#include <RInside.h>

int main(int argc, char *argv[]) {

    // create an embedded R instance
    RInside R(argc, argv);

    // assign a char* (string) to 'txt'
    R["txt"] = "Hello, world!\n";

    // eval the init string, ignoring returns
    R.parseEvalQ("cat(txt)");  

    exit(0);
}
RInside in a nutshell

Key aspects:

- RInside uses the embedding API of R
- An instance of R is launched by the RInside constructor
- It behaves just like a regular R process
- We submit commands as C++ strings which are parsed and evaluated
- Rcpp is used to easily get data in and out from the enclosing C++ program.
The question is sometimes asked how to embed RInside in a larger program.

We have a nice example using Qt:

```c
#include <QApplication>
#include "qtdensity.h"

int main(int argc, char *argv[]) {

    RInside R(argc, argv);       // embedded R inst.
    QApplication app(argc, argv);
    QtDensity qtdensity(R);      // passess by ref.
    return app.exec();
}
```
Application example: Qt density slider

This uses standard Qt / GUI paradigms of
- radio buttons
- sliders
- textentry

all of which send values to the R process which provides a PNG image that is plotted.
Given the desktop application with **Qt**, the question arises how to deliver something similar “over the web” — and **Wt** helps.

**Wt** is similar to **Qt** so the code needs only a few changes. **Wt** takes care of all browser / app interactions and determines the most featureful deployment.
The package comes with eight pdf vignettes, and numerous help pages.

The introductory vignettes are now published (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp. Stat. & Data Anal.*).

The rcpp-devel list is *the* recommended resource, generally very helpful, and fairly low volume.

By now StackOverflow has a fair number of posts too.

And a number of blog posts introduce/discuss features.
Featured Articles

Quick conversion of a list of lists into a data frame — John Merrill
This post shows one method for creating a data frame quickly

Passing user-supplied C++ functions — Dirk Eddelbuettel
This example shows how to select user-supplied C++ functions

Using Rcpp to access the C API of xts — Dirk Eddelbuettel
This post shows how to use the exported API functions of xts

Timing normal RNGs — Dirk Eddelbuettel
This post compares drawing N(0,1) vectors from R, Boost and C++11

A first lambda function with C++11 and Rcpp — Dirk Eddelbuettel
This post shows how to play with lambda functions in C++11

First steps in using C++11 with Rcpp — Dirk Eddelbuettel
This post shows how to experiment with C++11 features

Using Rcout for output synchronised with R — Dirk Eddelbuettel
This post shows how to use Rcout (and Rcerr) for output

Using the Rcpp sugar function clamp — Dirk Eddelbuettel
This post illustrates the sugar function clamp

Using the Rcpp Timer — Dirk Eddelbuettel
This post shows how to use the Timer class in Rcpp

Calling R Functions from C++ — Dirk Eddelbuettel
This post discusses calling R functions from C++

More »

Recently Published

Apr 12, 2013 » Using the RcppArmadillo-based implementation of R's sample() — Christian Gunning and Jonathan Ooms

Apr 8, 2013 » Dynamic Wrapping and Recursion with Rcpp — Kevin Ushey

Mar 14, 2013 » Using bigmemory with Rcpp — Michael Kane

Mar 12, 2013 » Generating a multivariate gaussian distribution using RcppArmadillo — Ahmadou Dicko

Mar 1, 2013 » Using Rcpp with Boost/Regex for regular expression — Dirk Eddelbuettel

Feb 27, 2013 » Fast factor generation with Rcpp — Kevin Ushey
The Rcpp book

Seamless R and C++ Integration with Rcpp

UseR!

Dirk Eddelbuettel

Originally expected in May 2013.

Real Soon Now.