Rcpp

Making R Applications Go Faster and Further

Dirk Eddelbuettel

EARL 2015 Keynote Address
September 16, 2015

Released under a Creative Commons Attribution-ShareAlike 4.0 International License.
INTRODUCTION
A Very Kind Tweet

Research Consulting
@iqssrtc

Using #Rcpp to leverage the speed of c++ with the ease and clarity of R. Thanks, @eddelbuettel

Reply  Retweet  Favorited  More

Retweet  Favorite
1  1

10:29 AM - 19 Mar 2012
Love that my reaction almost every time I rewrite R code in Rcpp is "holy shit that's fast" thanks @eddelbuettel & @romain_francois rstats
And Yet Another Tweet

Pat Schloss
@PatSchloss

Thanks to @eddelbuettel's Rcpp and @hadleywickham AdvancedR Rcpp chapter I just sped things up 750x. You both rock.

RETWEETS  FAVORITES
3      5

11:55 AM - 29 May 2015
Rich FitzJohn
@rgfitzjohn

Writing some code using #rstats plain C API and realising/remembering quite how much work Rcpp saves - thanks @eddelbuettel
And Last But Not Least

"Rcpp is one of the 3 things that changed how I write rstats code". @hadleywickham at #EARL2014
EXTENDING R


Thanks to John Chambers for sending me high-resolution scans of the covers of his books.
xx <- faithful[, "eruptions"]
fit <- density(xx)
plot(fit)
A Simple Example

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density

1 2 3 4 5 6
0.0 0.1 0.2 0.3 0.4 0.5

density.default(x = xx)

N = 272
Bandwidth = 0.3348

Density
xx <- faithful[, "eruptions"]
fit1 <- density(xx)
fit2 <- replicate(10000, {
    x <- sample(xx, replace=TRUE);
    density(x, from=min(fit1$x), to=max(fit1$x))$y
})
fit3 <- apply(fit2, 1, quantile, c(0.025, 0.975))
plot(fit1, ylim=range(fit3))
polygon(c(fit1$x, rev(fit1$x)), c(fit3[1], rev(fit3[2])), col='grey', border=F)
lines(fit1)
A Simple Example - Refined

density.default(x = xx)

N = 272  Bandwidth = 0.3348
So Why R?

R enables us to

- work interactively
- explore and visualize data
- access, retrieve and/or generate data
- summarize and report into pdf, html, ...

making it the key language for statistical computing, and a preferred environment for many data analysts.
So Why R?

R has always been extensible via

- C via a bare-bones interface described in *Writing R Extensions*
- Fortran which is also used internally by R
- Java via rJava by Simon Urbanek
- C++ but essentially at the bare-bones level of C

So while *in theory* this always worked – it was tedious *in practice*
Chambers (2008), opens Chapter 11 *Interfaces I: Using C and Fortran*:

*Since the core of R is in fact a program written in the C language, it’s not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.*
Chambers (2008), opens Chapter 11 *Interfaces I: Using C and Fortran*:

*Since the core of R is in fact a program written in the C language, it’s not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.*
Chambers proceeds with this rough map of the road ahead:

- **Against:**
  - It’s more work
  - Bugs will bite
  - Potential platform dependency
  - Less readable software

- **In Favor:**
  - New and trusted computations
  - Speed
  - Object references
The *Why?* boils down to:

- **speed**: Often a good enough reason for us ... and a focus for us in this workshop.
- **new things**: We can bind to libraries and tools that would otherwise be unavailable in R.
- **references**: Chambers quote from 2008 foreshadowed the work on *Reference Classes* now in R and built upon via Rcpp Modules, Rcpp Classes (and also RcppR6)
And Why C++?

- Asking Google leads to about ~ 50 million hits.
- Wikipedia: C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose, powerful programming language
- C++ is industrial-strength, vendor-independent, widely-used, and still evolving
- In science & research, one of the most frequently-used languages: If there is something you want to use / connect to, it probably has a C/C++ API
- As a widely used language it also has good tool support (debuggers, profilers, code analysis)
Scott Meyers: View C++ as a federation of languages

- C provides a rich inheritance and interoperability as Unix, Windows, … are all built on C.
- Object-Oriented C++ (maybe just to provide endless discussions about exactly what OO is or should be)
- Templated C++ which is mighty powerful; template meta programming unequalled in other languages.
- The Standard Template Library (STL) is a specific template library which is powerful but has its own conventions.
- C++11 (and C++14 and beyond) add enough to be called a fifth language.

NB: Meyers original list of four languages appeared years before C++11.
Why C++?

- Mature yet current
- Strong performance focus:
  - *You don’t pay for what you don’t use*
  - *Leave no room for another language between the machine level and C++*
- Yet also powerfully abstract and high-level
- C++11 is a big deal giving us new language features
- While there are complexities, Rcpp users are mostly shielded
INTERFACE VISION
Algorithm Interface

ABC: general
(FORTRAN)
algorithm

XABC: FORTRAN
subroutine to
provide interface
between ABC &
language and/or
utility programs

XABC (INSTR, OUTSTR)

Input INSTR →

"X"
"Y"

Pointers/Values
Argument Names or
Blank
R offers us the best of both worlds:

- **Compiled** code with
  - Access to proven libraries and algorithms in C/C++/Fortran
  - Extremely high performance (in both serial and parallel modes)

- **Interpreted** code with
  - An accessible high-level language made for *Programming with Data*
  - An interactive workflow for data analysis
  - Support for rapid prototyping, research, and experimentation
Why Rcpp?

- **Easy to learn** as it really does not have to be that complicated – we will see numerous few examples
- **Easy to use** as it avoids build and OS system complexities thanks to the R infrastructure
- **Expressive** as it allows for *vectorised* C++ using *Rcpp Sugar*
- **Seamless** access to all R objects: vector, matrix, list, S3/S4/RefClass, Environment, Function, ...
- **Speed gains** for a variety of tasks Rcpp excels precisely where R struggles: loops, function calls, ...
- **Extensions** greatly facilitates access to external libraries using eg *Rcpp modules*
Speed
Consider a function defined as

\[
f(n) \text{ such that } \begin{cases} 
  n & \text{when } n < 2 \\
  f(n - 1) + f(n - 2) & \text{when } n \geq 2 
\end{cases}
\]
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)
```

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

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

```markdown
## test replications elapsed relative
## 1 f(10) 100 0.026 1.000
## 2 f(15) 100 0.327 12.577
## 3 f(20) 100 3.796 146.000
```
A C or C++ solution can be equally simple

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

But how do we call it from R?
```c
#include <R.h>
#include <Rinternals.h>

int fibonacci_c_impl(int n) {
    if (n < 2) return n;
    return fibonacci_c_impl(n - 1) + fibonacci_c_impl(n - 2);
}

SEXP fibonacci_c(SEXP n) {
    SEXP result = PROTECT(allocVector(INTSXP, 1));
    INTEGER(result)[0] = fibonacci_c_impl(asInteger(n));
    UNPROTECT(1);
    return result;
}

/*
## need to compile, link, load, ...
fibonacci <- function(n) .Call("fibonacci_c", n)
sapply(0:10, fibonacci)
*/
#include <R.h>
#include <Rinternals.h>

int fibonacci_c_impl(int n) {
    if (n < 2) return n;
    return fibonacci_c_impl(n - 1) + fibonacci_c_impl(n - 2);
}

// [[Rcpp::export]]
SEXP fibonacci_c(SEXP n) {
    SEXP result = PROTECT(allocVector(INTSXP, 1));
    INTEGER(result)[0] = fibonacci_c_impl(asInteger(n));
    UNPROTECT(1);
    return result;
}

/*** R
sapply(0:10, fibonacci_c)
*/
But Rcpp makes this *much* easier:

```cpp
Rcpp::cppFunction("int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2)); }")
sapply(0:10, g)
```

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

Rcpp::cppFunction("int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2)); }")
library(rbenchmark)
benchmark(f(25), g(25), order="relative")[,1:4]

## test replications elapsed relative
## 2 g(25) 100 0.099 1.000
## 1 f(25) 100 47.787 482.697

A nice gain of a few orders of magnitude.
Run-time performance is just one example.

*Time to code* is another metric.

We feel quite strongly that helps you code more succinctly, leading to fewer bugs and faster development.

A good environment helps. RStudio integrates R and C++ development quite nicely (eg the compiler error message parsing is very helpful) and also helps with package building.
#include <Rcpp.h>

// [[Rcpp::plugins("cpp11")]]

constexpr int fibonacci_recursive_constexpr(const int n) {
    return n < 2 ? n : (fibonacci_recursive_constexpr(n - 1) +
                        fibonacci_recursive_constexpr(n - 2));
}

// [[Rcpp::export]]
int constexprFib() {
    const int N = 42;
    constexpr int result = fibonacci_recursive_constexpr(N);
    return result;
}
POPULARITY
Used by 462 CRAN Packages as of Last Weekend
Page Rank One (According to Andrie de Vries)

Dirk Eddelbuettel
@eddelbuettel

Achievement unlocked: @revoandrie says #Rcpp has page rank 1 on CRAN! #useR2015
APPLICATION SPOTLIGHT: RBLPAPI
Programming with financial data: Connecting R to Lim and Bloomberg

Dirk Eddelbuettel
B of A and Debian
edd@debian.org

Presentation at the
useR!
2004

conference, Vienna, May 20-22, 2004
History: Key package using Java (Ana, then John)

R[name redacted]

R[name redacted] handles fetching data from the [name redacted] financial data application from within R. Both the application and the R package are named after the mayor of New York City, whose name has been stricken from this site following a misguided request from the data service provider.

The PDF Manual has temporarily been removed.

Troubleshooting & Help

Please read the help page if you are having difficulty with R[name redacted].

Installing Latest (Development) R[name redacted]

To install the latest daily build of the development version of R[name redacted] from this site:

```
install.packages("R[name redacted]",repos="http://r.findata.org/")
```

Compiling R[name redacted] from Source

If you wish to compile R[name redacted] as an R package yourself, you will need to download and install R Tools, and read the instructions for compiling R packages on Windows which is slightly more involved than on other platforms.

The R[name redacted] source is available to download from the R-Forge project page, or by checking it out from the Bazaar repository:

```
bzr branch http://findata.org/code/blipwrapper
```
History: But the vendor API keeps improving
Present .. and Future (Whit, Dirk, and John)
Rblpapi: R Interface to BBG

An R Interface to the BBG service is provided via the Bllp API.

Version: 0.3.0
Imports: Rcpp (≥ 0.11.0), utils
LinkingTo: Rcpp, BH
Suggests: fts, xts, zoo, knitr
Published: 2015-08-14
Author: Whit Armstrong, Dirk Eddelbuettel and John Laing
Maintainer: Dirk Eddelbuettel <edd at debian.org>
License: file LICENSE
NeedsCompilation: yes
SystemRequirements: A valid BBG installation. API headers and dynamic library are downloaded from <https://github.com/Rblp/blp> during the build step.

Materials: README NEWS Changelog
CRAN checks: Rblpapi results

Downloads:

Reference manual: Rblpapi.pdf
Vignettes: Introducing Rblpapi
Package source: Rblpapi_0.3.0.tar.gz
Windows binaries: r-devel: Rblpapi_0.3.0.zip, r-release: Rblpapi_0.3.0.zip, r-oldrel: Rblpapi_0.3.0.zip
OS X Snow Leopard binaries: r-release: not available, r-oldrel: not available
OS X Mavericks binaries: r-release: Rblpapi_0.3.0.tar.gz
Third Time Lucky: The Rblpapi package

The new rewrite is different:

- Lighter – no longer uses or requires Java
- Simpler – leverages Rcpp
- More flexible – easy to add new functionality with C++
Where we are at now:

- Robust and fast
- Implements most widely-used features
- (Basic) documentation for everything
- Travis CI integration
- On GitHub and in the ghrr repository

NB: And now on CRAN too, see below.
Core Functions known from other API accessors:

- `bdp(c("ESA Index", "SPY US Equity"), c("PX_LAST", "VOLUME"))`
- `bds("GOOG US Equity", "TOP_20_HOLDERS_PUBLIC_FILINGS")`
- `bdh("SPY US Equity", c("PX_LAST", "VOLUME"),
  start.date=Sys.Date()-31)`
- `getBars("ESA Index", startTime=ISOdatetime(2015,1,1,0,0,0))`
- `getTicks("ESA Index", "TRADE", Sys.time()-60*60)`
- `fieldSearch("VWAP")`
Things we addressed

- Fixed-dimension retrieval very easy
- Now include shared library with \texttt{rpath}-encoded path
- Builds “everywhere” including on Travis CI
Things we [then] need[ed] to address:

- DataFrame class caused trouble, need something new
- Builds on “that other OS” very difficult while (vendor) API library built with VC++
- More features: subscriptions, screens, portfolios...
- Pull requests welcome!
Concluding:

- Bloomberg provides a first-rate API and infrastructure
- So the R Community came up with good packages
- Language/OS choice matter: some vendors still “different”
- We prefer Open Source; package may not go onto CRAN
- But we have alternatives in GitHub-hosted repositories

NB: That was then …
What we learned:

- Powerful APIs are compelling
- Providing working code is key
- Unexpectedly, we got a pull request for Windows support
- With some additional work, this got onto CRAN
- Supporting Linux, OS X and Windows “out of the box”
The End
The Rcpp package comes with nine pdf vignettes, and numerous help pages.

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

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

StackOverflow has almost 900 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 Rcpp — Dirk Eddelbuettel
This post shows how to use the exported API functions of Rcpp

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 Rcpp for output synchronized with R — Dirk Eddelbuettel
This post shows how to use Rcpp (and Rcpp) 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 Rs sample() — Christian Gunning and Jonathan Olsten
Apr 8, 2013 » Dynamic Wrapping and Recursion with Rcpp — Kevin Ushey
Mar 14, 2013 » Using bitmemory 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
Thank You!

dirk@eddelbuettel.com

http://dirk.eddelbuettel.com/presentations/