An Introduction to Rcpp

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

Bay Area userR Group
San Jose, 27 January 2015
Outline

1 Motivation
   • C++
   • Vision
   • Features
   • Interface
A tweet about Rcpp from IQSS / Harvard

Research Consulting
@iqssrtc

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

10:29 AM - 19 Mar 2012
Another tweet about Rcpp

Peter Hickey
@PeteHaitch

Love that my reaction almost every time I rewrite R code in Rcpp is "holy shit that’s fast" thanks @eddelbuettel & @romain_francois #rstats

9:08 PM - 18 Oct 2013
Rcpp: The Good Parts

- Ease of use
- R -> C++
- C++ as Glue
Why R?
Interactive

R enables us to

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

making it a preferred environment for many data analysts.
Why R?
Extensible

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 S Urbanek

**C++** but essentially at the bare-bones level of C

So ’in theory’ this worked – yet tedious ’in practice’.
Why C++?

- Asking Google [currently] leads to about 42 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).
Why C++?
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++ 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 STL which is a specific template library which is powerful but has its own conventions.

C++11 adds enough to be called a fifth language.

NB: Meyers original list of four language 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 a language between the machine level and C++”
- Yet also powerfully abstract and high-level
- C++11 and beyond are a big deal giving us new language features
- While there are complexities, Rcpp users are mostly shielded
Source: John Chambers, personal communication.
Interface Vision

- Use trusted numerical libraries (mostly/exclusively written in Fortran)
- Provide environment which statistician could use more easily
- Enable interactive and iterative data exploration
- Make it extensibility for research into statistical methods
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  it really does not have to be that complicated – we will look at a few examples

Easy to use  as it avoids build and OS system complexities thanks to the R infrastructure

Expressive  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
Rcpp Playground: Defined by `.Call`

R side

```r
res <- `.Call("nameOfFunction", a, b, c,
            ..., PACKAGE="name")`
```

C++ side

```cpp
extern "C"
SEXP `nameOfFunction`(SEXP a, SEXP b,
             SEXP c, ...)
```
Outline

2

Rcpp

• How
• Example: VAR(1)
• Packages
How do we use Rcpp?
RStudio makes it very easy: Single File
How do we use Rcpp?

RStudio example cont’ed

The following file used to get created:

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

// Below is a simple example of exporting a C++ function to R. You can source this function into an R session using the Rcpp::sourceCpp function (or via the Source button on the editor toolbar)

// For more on using Rcpp click the Help button on the editor toolbar

// [[Rcpp::export]]
int timesTwo(int x) {
    return x * 2;
}
```
The following file gets created now:

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

// Below is a simple example of exporting a C++ function to R. 
// You can source this function into an R session using the
// Rcpp::sourceCpp function (or via the Source button on the
// editor toolbar)

// For more on using Rcpp click the Help button on the editor toolbar

// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
    return x * 2;
}
```
How do we use Rcpp?

Rcpp Attributes: evalCpp, cppFunction, sourceCpp

```r
## evaluate a C++ expression, retrieve result
evalCpp("2 + 2")
```

```r
## [1] 4
```

```r
## create ad-hoc R function 'square'
cppFunction('int square(int x) { return x*x; }')
square(7L)
```

```r
## [1] 49
```

```r
## or source an entire file (including R code)
#sourceCpp("code/squareWithRCall.cpp")
```
When would we use Rcpp?

Easy speed gain: VAR(1) Simulation

Let’s consider a simple possible VAR(1) system of $k$ variables.

For $k = 2$:

$$X_t = X_{t-1}B + E_t$$

where $X_t$ is a row vector of length 2, $B$ is a 2 by 2 matrix and $E_t$ is a row of the error matrix of 2 columns.
When do we use Rcpp?
Easy speedup:: VAR(1) Simulation

In R code, given both the coefficient and error matrices (revealing $k$ and $n$):

```r
rSim <- function(B,E) {
  n <- nrow(E); k <- ncol(E)
  X <- matrix(0, n, k)
  for (r in 2:n) {
    X[r,] = X[r-1, ] %*% B + E[r, ]
  }
  return(X)
}
```
When do we use Rcpp?

Easy speed gain: VAR(1) Simulation

```cpp
cppFunction('arma::mat cppSim(const arma::mat& B,
                           const arma::mat& E) {
    int n = E.n_rows; int k = E.n_cols;
    arma::mat X = arma::zeros<arma::mat>(n,k);
    for (int r=1; r < n; r++) {
        X.row(r) = X.row(r-1) * B + E.row(r);
    }
    return X;
}', depends="RcppArmadillo")
```
When do we use Rcpp?
Easy speed gain: VAR(1) Simulation

```r
a <- matrix(c(0.5,0.1,0.1,0.5),nrow=2)
e <- matrix(rnorm(10000),nrow=2)
all.equal(cppSim(a,e), rSim(a,e))
```

```r
## [1] TRUE
```

```r
benchmark(cppSim(a,e), rSim(a,e),
          order="relative")[,1:4]
```

```r
## test replications elapsed relative
## 1 cppSim(a, e) 100 0.044 1.000
## 2 rSim(a, e) 100 2.633 59.841
```
How do we use Rcpp?

RStudio makes it very easy (using Rcpp.package.skeleton())
How do we use Rcpp?

Key Features

As we just saw in the small example:

- No build system issues: We just use R
- No operating system dependencies
- No (manual or explicit) memory management
- No complicated C++ code (though we could if we wanted to)
- Easy transition from exploration ("one-liners") to deployment ("packages")
3 Growth
Rcpp on CRAN: Now at 323 packages

Growth of Rcpp usage on CRAN

- Number of CRAN packages using Rcpp
- Percentage of CRAN packages using Rcpp

May 02 2012
Dec 01 2012
Aug 03 2013
Feb 03 2014
Aug 04 2014
Jan 23 2015

Growth of Rcpp usage on CRAN

Dirk Eddelbuettel
Introduction to Rcpp
Outline

4 Doc
  • Basics
  • Gallery
  • Book
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.

**StackOverflow** has over 670 posts too.

Several blog posts introduce/discuss features.
What Else?

Rcpp Gallery: 90+ working and detailed examples

**Featured Articles**

- Quick conversion of a list of lists into a data frame — John Merrill
- Passing user-supplied C++ functions — Dirk Eddelbuettel
- This example shows how to select user-supplied C++ functions
- Using Rcpp to access the R 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++

**Recently Published**

- Apr 12, 2013 » Using the RcppArmadillo-based Implementation of R's sample() — Christian Gunning and Jonathan Ormiston
- Apr 8, 2013 » Dynamic Wrapping and Recursion with Rcpp — Kevin Ushey
- Mar 14, 2013 » Using bigmemory with Rcpp — Michael Kame
- 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
What Else?
The Rcpp book

Seamless R and C++ Integration with Rcpp

In print since June 2013