RCPP AT 1000 REVERSE DEPENDS:
SOME OBSERVATIONS

Dirk Eddelbuettel

DSC 2017

July 3, 2017

Ketchum Trading; Debian and R Projects
Some Notes ...

- about Rcpp, ever so briefly
- about testing
- about APIs

More a stream of consciousness
A few points

- 1000 depends is a nice milestone to summarize
- Rcpp is a fairly widely used package (over 1k direct depends)
- Rcpp affects a number of packages (over 7k recursive depends)
- We try to take testing somewhat seriously
Rcpp
Team Effort

- Dominic had the early vision
- Romain turned the dial to 11, and again, and again
- Doug and John provided early adult oversight
- JJ gave us Rcpp Attributes and much wisdom
- Kevin, KK, and Nathan are keeping the wheels on
Data current as of July 1, 2017.
library(pagerank)  # github.com/andrie/pagerank

cran <- "http://cloud.r-project.org"

pr <- compute_pagerank(cran)

##
## Attaching package: 'utils'

## The following objects are masked from 'package:Rcpp':
##
## .DollarNames, prompt

round(100*pr[1:5], 3)
Top 30 of Page Rank as of July 2017
Top 30 packages by page rank
CRAN Proportion

```r
db <- tools::CRAN_package_db()  # R 3.4.0 or later
dim(db)

## [1] 10958 65

## all Rcpp reverse depends
(c(n_rcpp <- length(tools::dependsOnPkgs("Rcpp", recursive=FALSE, installed=db)),
  n_compiled <- table(db[, "NeedsCompilation"])[["yes"]]))

## [1] 1074 2928

## Rcpp percentage of packages with compiled code
n_rcpp / n_compiled

## [1] 0.3668033
```
One Example
```c
#include <R.h>
#include <Rinternals.h>

SEXP convolve2(SEXP a, SEXP b) {

    int na, nb, nab;
    double *xa, *xb, *xab;
    SEXP ab;

    a = PROTECT(coerceVector(a, REALSXP));
    b = PROTECT(coerceVector(b, REALSXP));
    na = length(a);
    nb = length(b);
    nab = na + nb - 1;
    ab = PROTECT(allocVector(REALSXP, nab));
    xa = REAL(a);
    xb = REAL(b);
    xab = REAL(ab);
    for (int i = 0; i < nab; i++)
        xab[i] = 0.0;
    for (int i = 0; i < na; i++)
        for (int j = 0; j < nb; j++)
            xab[i + j] += xa[i] * xb[j];
    UNPROTECT(3);
    return ab;
}
```
Example: Convolution

```cpp
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::NumericVector convolve2cpp(Rcpp::NumericVector a,
                                 Rcpp::NumericVector b) {
    int na = a.length(), nb = b.length();
    Rcpp::NumericVector ab(na + nb - 1);
    for (int i = 0; i < na; i++)
        for (int j = 0; j < nb; j++)
            ab[i + j] += a[i] * b[j];
    return(ab);
}
```
cppFunction("Rcpp::NumericVector
cppFunction("Rcpp::NumericVector
convolve2cpp(Rcpp::NumericVector a,
Rcpp::NumericVector b) {
    int na = a.length(), nb = b.length();
    Rcpp::NumericVector ab(na + nb - 1);
    for (int i = 0; i < na; i++)
        for (int j = 0; j < nb; j++)
            ab[i + j] += a[i] * b[j];
    return(ab);
}"
convolve2cpp(1:4, 4:1)

## [1]  4 11 20 30 20 11  4
Testing
No Free Lunch

- Single run on a decent machine now takes more than a workday
- Should be easy-ish to parallelize (given resources)
- But that has not yet happened.
- Is testing support a community thing? R Hub?
No Free Lunch

- Do we need to rethink testing?
  - only packages which themselves are impactful? (*maybe*)
  - only packages which were updated recently? (*maybe not*)
  - only packages which may have failed in the past? (*possibly*)
  - other ways to subsample?

- This both an engineering and a statistics questions so ...
Still No Free Lunch

- Tests really only run the code they cover
- Rcpp has e.g. code generators, we generally do not regenerate *in client packages*
- The *one minute cap* via CRAN Policy means we *suppress tests*
API
That worked well

- Package system and design work as plan
- Access of C API of R now easier to access
- Good division of labour
Should Rcpp be promoted into Base R?

Question I get asked sometime

- Probably not
- If “you” take it “you” get to work on it
- Smaller base good design principle
API Re-Use?

- RApiSerialize
- RApiDatetime
- There could potentially be much more
- How can “we” (R users) get better (programmatic) access to what is already in R?
- Does the (relatively) wide use of Rcpp mean the core API is too hard to use?
Next Steps?

- Possible room for improvement on testing
- Possible need for better testing support
- Possible to open the API a little more