Higher-Performance R Programming with C++ Extensions

Part 5: Rcpp Packaging and Debugging

Dirk Eddelbuettel

June 28 and 29, 2017

University of Zürich & ETH Zürich

Overview

Overview

- R Packaging basics
- Rcpp Examples covering
  - just Rcpp, or included sources
  - Rcpp with libraries
- We will then look at debugging with
  - basics of `gdb` debugging
  - Docker use of r-devel

R Packaging

R Packages

- This is an important topic in R programming
- Organising code in packages maybe *the* single most helpful step
- Core topic in R Programming / Advanced R courses
- Penn 2014 workshop had 90 minutes on this (slides)

R Packages

- `package.skeleton()` not helpful as it creates a stanza that does not pass `R CMD check` cleanly
- `pkgKitten::kitten()` creates *packages that purr*
- Rcpp and friends have their own versions of `package.skeleton()`: `Rcpp.package.skeleton()` etc.
- They can use `kitten()` if pkgKitten is installed
- Alternative: `devtools::create()` if you don't mind Hadleyverse dependency
- Also: RStudio File -> New Project -> New Directory -> R Package; and toggle 'R Package' to 'R Package w/ Rcpp' – in newest version with support for RcppArmadillo and RcppEigen

Case Studies

RcppAnnoy

One core C++ header, plus one support header and one header for Windows:

```
edd@max:~/git/rcppannoy$ tree inst/include/
inst/include/
├── annoylib.h
├── kissrandom.h
└── mman.h

0 directories, 3 files
edd@max:~/git/rcppannoy$
```

RcppAnnoy

Only one source file, plus one generated interface file (per R 3.4.0)

```
edd@max:~/git/rcppannoy$ tree src/
src/
├── annoy.cpp
├── Makevars
└── RcppExports.cpp

0 directories, 3 files
edd@max:~/git/rcppannoy$
```

RcppAnnoy: `src/Makevars`

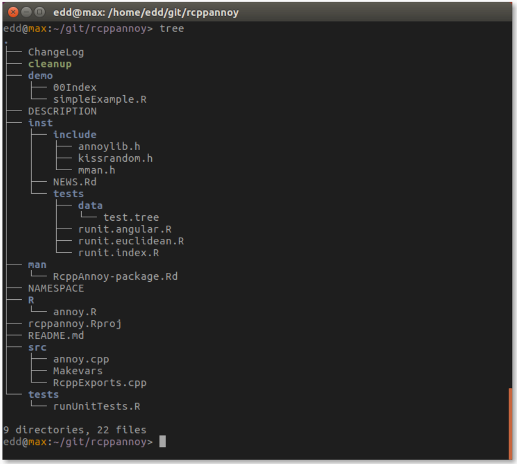
One include indirection to the header file

```
## We want C++11 as it gets us 'long long' as well
## If we comment this we fall back to C++98 and it is all
CXX_STD = CXX11

PKG_CPPFLAGS = -I../inst/include/
```

RcppAnnoy: `src/annoy.cpp`

- Implemented as Rcpp Modules (not discussed today)
- Wraps around templated C++ class for either
    - Angular distance, or
    - Euclidian distance
- Package interesting as upstream C++ core used with Python by upstream

RcppAnnoy



edd@max: /home/edd/git/rcppannoy

```
edd@max:~/git/rcppannoy> tree
.
├── ChangeLog
├── cleanup
├── demo
│   ├── 00Index
│   └── simpleExample.R
├── DESCRIPTION
├── inst
│   ├── include
│   │   ├── annoylib.h
│   │   ├── kissrandom.h
│   │   └── mman.h
│   ├── NEWS.Rd
│   └── tests
│       ├── data
│       │   └── test.tree
│       ├── runit.angular.R
│       ├── runit.euclidean.R
│       └── runit.index.R
├── man
│   └── RcppAnnoy-package.Rd
├── NAMESPACE
├── R
│   └── annoy.R
├── rcppannoy.Rproj
├── README.md
├── src
│   ├── annoy.cpp
│   ├── Makevars
│   └── RcppExports.cpp
└── tests
    └── runUnitTests.R

9 directories, 22 files
edd@max:~/git/rcppannoy>
```

Plus a few additional files for tests and documentation.

RcppCNPy

Uses one C++ header and one C++ source file from CNPy

```
edd@max:~/git/rcppcnpy$ tree src
src/
├── cnpy.cpp          # from CNPy
├── cnpy.h            # from CNPy
├── cnpyMod.cpp       # our wrapper
├── Makevars          # add -lz (from R) and C++11
└── Makevars.win      # ditto
└── RcppExports.cpp   # R 3.4.0

0 directories, 6 files
edd@max:~/git/rcppcnpy$
```

RcppCNPy

Needed Here:

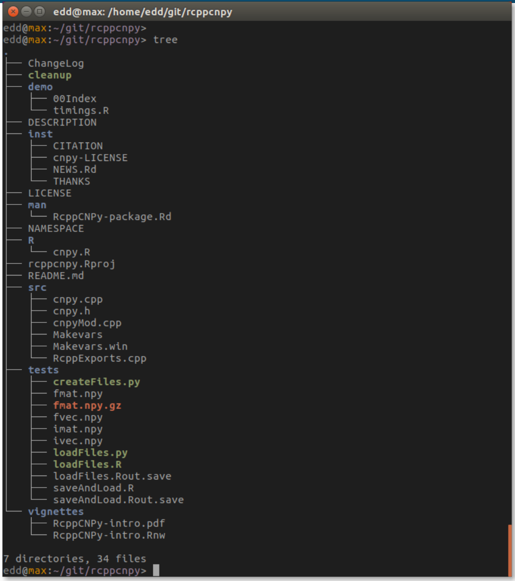
- For this package no other customization is needed
- Simply add the two source files
- Code integration done via Rcpp Modules (which we won't cover today)
- Here we just need one linker flag (supplied by R)

RcppCNPy: `src/Makevars`

One linker flag (and a compiler option for `long long`)

```
## We need the compression library
PKG_LIBS = -lz

## We want C++11 as it gets us 'long long' as well
CXX_STD = CXX11
```

RcppCNPy


edd@max: /home/edd/git/rcppcnpy

```
edd@max:~/git/rcppcnpy>
edd@max:~/git/rcppcnpy> tree
.
├── ChangeLog
├── cleanup
├── demo
│   ├── 00Index
│   └── timings.R
├── DESCRIPTION
├── inst
│   ├── CITATION
│   ├── cnpy-LICENSE
│   ├── NEWS.Rd
│   └── THANKS
├── LICENSE
├── man
│   └── RcppCNPy-package.Rd
├── NAMESPACE
├── R
│   └── cnpy.R
├── rcppcnpy.Rproj
├── README.md
├── src
│   ├── cnpy.cpp
│   ├── cnpy.h
│   ├── cnpyMod.cpp
│   ├── Makevars
│   ├── Makevars.win
│   └── RcppExports.cpp
├── tests
│   ├── createFiles.py
│   ├── fmat.npy
│   ├── fmat.npy.gz
│   ├── fvec.npy
│   ├── imat.npy
│   ├── ivec.npy
│   ├── loadFiles.py
│   ├── loadFiles.R
│   ├── loadFiles.Rout.save
│   ├── saveAndLoad.R
│   └── saveAndLoad.Rout.save
└── vignettes
    ├── RcppCNPy-intro.pdf
    └── RcppCNPy-intro.Rnw

7 directories, 34 files
edd@max:~/git/rcppcnpy>
```

More tests and documentation make this look more ”busy” – but still a simple package.

RcppAPT

Overview

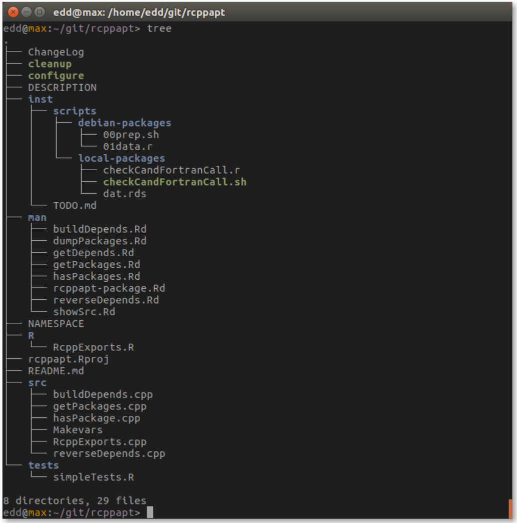
- A somewhat still ‘raw’ package which only builds on Ubuntu or Debian
- Interfaces a system library we can assume to be present on those systems – but not on OS X, Windows or even other Linux systems

RcppAPT: `src/Makevars`

- Very simple

```
PKG_LIBS = -lapt-pkg
```

RcppAPT



```
edd@max:~/git/rcppapt> tree
.
├── ChangeLog
├── cleanup
├── configure
├── DESCRIPTION
├── inst
│   ├── scripts
│   │   ├── debian-packages
│   │   │   ├── 00prep.sh
│   │   │   └── 01data.r
│   │   └── local-packages
│   │       ├── checkCandFortranCall.r
│   │       ├── checkCandFortranCall.sh
│   │       └── dat.rds
│   └── TODO.md
├── man
│   ├── buildDepends.Rd
│   ├── dumpPackages.Rd
│   ├── getDepends.Rd
│   ├── getPackages.Rd
│   ├── hasPackages.Rd
│   ├── rcppapt-package.Rd
│   ├── reverseDepends.Rd
│   └── showSrc.Rd
├── NAMESPACE
├── R
│   └── RcppExports.R
├── rcppapt.Rproj
├── README.md
├── src
│   ├── buildDepends.cpp
│   ├── getPackages.cpp
│   ├── hasPackage.cpp
│   ├── Makevars
│   ├── RcppExports.cpp
│   └── reverseDepends.cpp
└── tests
    └── simpleTests.R

8 directories, 29 files
edd@max:~/git/rcppapt>
```

Fairly simple: a few functions accessing a 'system' library.

RcppFaddeeva

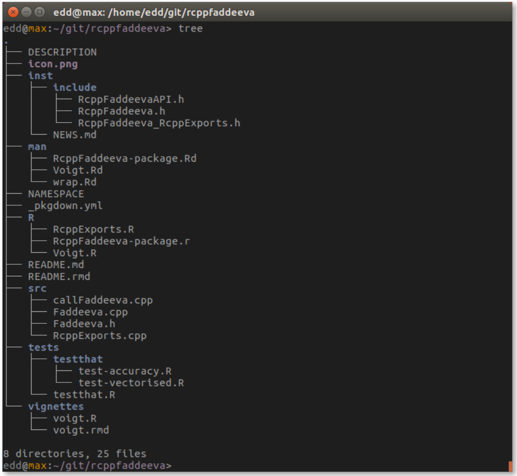
Overview

- A small package by Baptiste Auguie with some help from me
- Wrapper around some complex-valued error functions by Steven Johnson
- Upstream ships a single header and a single C++ file –> just place in `src/`
- Usage pretty easy: loop over elements of argument vector and call respective function to build return vector

RcppFaddeeva

```
//' @title Faddeeva family of error functions of the complex variable
//' @description the Faddeeva function
//' @param z complex vector
//' @param relerr double, requested error
//' @return complex vector
//' @describeIn wrap compute w(z) = exp(-z^2) erfc(-iz)
//' @family wrapper
//' @examples
//' Faddeeva_w(1:10 + 1i)
//' @export
// [[Rcpp::export]]
std::vector< std::complex<double> >
Faddeeva_w(const std::vector< std::complex<double> >& z, double relerr=0) {
  int N = z.size();
  std::vector< std::complex<double> > result(N);
  for(int i=0; i<N; i++) {
    result[i] = Faddeeva::w(z[i], relerr);
  }
  return result;
}
```

RcppFaddeeva



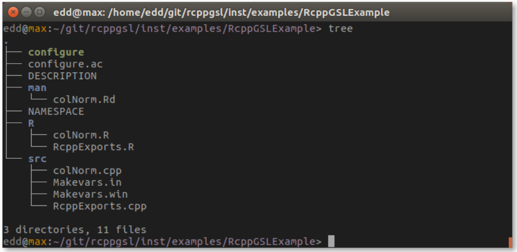
```
edd@max:~/git/rcppfaddeeva> tree
.
├── DESCRIPTION
├── icon.png
├── inst
│   ├── include
│   │   ├── RcppFaddeevaAPI.h
│   │   ├── RcppFaddeeva.h
│   │   └── RcppFaddeeva_RcppExports.h
│   └── NEWS.md
├── man
│   ├── RcppFaddeeva-package.Rd
│   ├── Voigt.Rd
│   └── wrap.Rd
├── NAMESPACE
├── _pkgdown.yml
├── R
│   ├── RcppExports.R
│   ├── RcppFaddeeva-package.r
│   └── Voigt.R
├── README.md
├── README.rmd
├── src
│   ├── callFaddeeva.cpp
│   ├── Faddeeva.cpp
│   ├── Faddeeva.h
│   └── RcppExports.cpp
├── tests
│   ├── testthat
│   │   ├── test-accuracy.R
│   │   └── test-vectorised.R
│   └── testthat.R
└── vignettes
    ├── voigt.R
    └── voigt.rmd

8 directories, 25 files
edd@max:~/git/rcppfaddeeva>
```

Fairly simple: an included cpp file and header

RcppGSLExample

- This package is included in the RcppGSL package and part of the test environment
- It implements the same column norm example we looked at earlier.

RcppGSLExample



```
edd@max: /home/edd/git/rcppgsl/inst/examples/RcppGSLExample
edd@max:~/git/rcppgsl/inst/examples/RcppGSLExample> tree
.
├── configure
├── configure.ac
├── DESCRIPTION
├── man
│   └── colNorm.Rd
├── NAMESPACE
├── R
│   ├── colNorm.R
│   └── RcppExports.R
└── src
    ├── colNorm.cpp
    ├── Makevars.in
    ├── Makevars.win
    └── RcppExports.cpp

3 directories, 11 files
edd@max:~/git/rcppgsl/inst/examples/RcppGSLExample>
```

Simple package against library which we test for (`configure`) and set environment variable for (`src/Makevars.win`)

R-only to R and Rcpp

Essentially:

- No full example here
- Easy to do manually:
    - Add `LinkingTo: Rcpp` to `DESCRIPTION`
    - Also add `Imports: Rcpp` to `DESCRIPTION`
    - Add `importFrom(Rcpp, ”evalCpp”)` to `NAMESPACE`
- Add some C++ code in `src/`
- Remember to run `compileAttributes()` each time you add (or change!) a C++ intrface

Packages and Rcpp

Summary of ways to link to external libraries

- *Full copies:* Do what RcppMLPACK (v1), RcppCNPy, … does and embed a full copy; larger build time, harder to update, self-contained
- *With linking of libraries:* Do what RcppGSL or RcppMLPACK (v2) do and use hooks in the package startup to store compiler and linker flags which are passed to environment variables
- *With C++ template headers only:* Do what RcppArmadillo and other do and just point to the headers

Debugging

Using gdb

Basics:

- Debugging is unfortunately platform-specific
- When the compiler is `g++`, the debugger is `gdb`
- When the compiler is `clang++`, the debugger is `lldb`.
- I use `g++` more often (under Linux) so we'll focus on `gdb`. However, `lldb` is very similar.

Using gdb

```cpp
#include <Rcpp.h>

// [[Rcpp::export]]
bool divbyzero(int x) {
    int res = x / 0L;
    Rcpp::Rcout << "res is now " << res << std::endl;
    return true;
}
```

Using gdb

```
R> sourceCpp(”debugEx.cpp”)
debugEx.cpp: In function ‘bool divbyzero(int)’:
debugEx.cpp:6:17: warning: division by zero [-Wdiv-by-zero]
     int res = x / 0L;
                 ^
R> divbyzero(10L)
res is now
Process R floating point exception (core dumped) at Wed Jun 24 20:19:12 2015
```

Using gdb

Start R with `-d gdb` switch, then type `run` to launch R.

```
R> Rcpp::sourceCpp("debugEx.cpp")
[Thread 0xb5cecb40 (LWP 25544) exited]
[Thread 0xb64edb40 (LWP 25543) exited]
[Thread 0xb44ebb40 (LWP 25545) exited]
debugEx.cpp: In function ‘bool divbyzero(int)’:
debugEx.cpp:6:17: warning: division by zero [-Wdiv-by-zero]
     int res = x / 0L;
                 ^
R> divbyzero(10L)
res is now
Program received signal SIGFPE, Arithmetic exception.
0xb0b94a5f in divbyzero (x=10) at debugEx.cpp:6
6       int res = x / 0L;
(gdb)
```

Now at line of floating point exception.

Using gdb

- Worth learning more about gdb
- Some tutorials:
    - SO post of mine
    - Similar SO post for OS X
    - Seth Falcon (of R Core) video
    - BioConductor HOWTO on C debugging

ASAN Errors

ASAN Errors

- CRAN is now using recent g++ / clang++ features for
  - ASAN (“Address Sanitizer”)
  - UBSAN (“Undefined Behaviour Sanitizer”)
- These allow us to “instrument” R with compiler-dependent run-time diagnostics
- Problem: Needs R sources, recent compilers, knowledge of building R from source
- Solution: Docker! (but we’d need more time than we have today to properly introduce Docker)
- sanitizers package triggers ‘true positives’ validating toolchain setups so that errors can be replicated & fixed.
- See eg my sanitizers page and my worked UBSAN example

ASAN Errors

```
#include <R.h>
#include <Rdefines.h>

extern "C" {
    // https://code.google.com/p/address-sanitizer/wiki/ExampleHeapOutOfBounds
    SEXP heapAddressSanitize(SEXP xs) {
        int *array = new int[100];
        int x, y;
        SEXP res;
        int *pres;
        array[0] = 0;
        x = INTEGER_VALUE(xs);
        y = array[x + 100];         // BOOM
        delete [] array;
        PROTECT(res = NEW_INTEGER(1));  // Allocating storage space
        pres = INTEGER_POINTER(res);    // pointer to SEXP object
        pres[0] = y;
        UNPROTECT(1);
        return res;
    }
}
```

ASAN Errors

```
edd@max:~/git$ docker run --rm -ti -v $(pwd):/mnt  rocker/r-devel-san RD CMD check /mnt/sanitizers_0.1.0.1.tar.gz
* using log directory ‘//sanitizers.Rcheck’
* using R Under development (unstable) (2015-06-17 r68530)
[...]
* checking tests ...
  Running ‘testHeapAddressSanitize.R’
 ERROR
Running the tests in ‘tests/testHeapAddressSanitize.R’ failed.
Last 13 lines of output:
    Freed heap region:       fd
    Stack left redzone:      f1
    Stack mid redzone:       f2
    Stack right redzone:     f3
    Stack partial redzone:   f4
    Stack after return:      f5
    Stack use after scope:   f8
    Global redzone:          f9
    Global init order:       f6
    Poisoned by user:        f7
    Contiguous container OOB:fc
    ASan internal:           fe
  ==266==ABORTING
* checking PDF version of manual ... OK
* DONE

Status: 1 ERROR
See
  ‘//sanitizers.Rcheck/00check.log’
for details.
```

UBSAN Errors

- For UBSAN we use a different Docker image
- It includes a wrapper script `check.r` which makes deployment very easy.

UBSAN Errors

```
#include <R.h>
#include <Rdefines.h>

extern "C" {
    // with thanks to Greg Jefferis (https://github.com/eddelbuettel/docker-debian-r/i
    // call with a sufficiently large x such as 31
    SEXP intOverflow(SEXP xs) {
        int x, y;
        SEXP res;
        int *pres;

        x = INTEGER_VALUE(xs);
        y = (1 << x)  - 1;          // BOOM -- (signed) int overflow

        PROTECT(res = NEW_INTEGER(1));  // Allocating storage space
        pres = INTEGER_POINTER(res);    // pointer to SEXP object
        pres[0] = y;
        UNPROTECT(1);
        return res;
    }
}
```

UBSAN Errors

```
edd@max:~/git$ docker run --rm -ti -v $(pwd):/mnt  rocker/r-devel-ubsan-clang check.r -s /mnt sanitizers_0.1.0.1.t
* using log directory '/mnt/sanitizers.Rcheck'
[...]
* checking for unstated dependencies in 'tests' ... OK
* checking tests ...
  Running 'testHeapAddressSanitize.R'
  Running 'testIntOverflowSanitize.R'
 ERROR
Running the tests in 'tests/testIntOverflowSanitize.R' failed.
Last 13 lines of output:

  R is a collaborative project with many contributors.
  Type 'contributors()' for more information and
  'citation()' on how to cite R or R packages in publications.

  Type 'demo()' for some demos, 'help()' for on-line help, or
  'help.start()' for an HTML browser interface to help.
  Type 'q()' to quit R.

  >
  > library(sanitizers)
  > intOverflowSanitize(31)
  int_overflow.cpp:17:23: runtime error: signed integer overflow: -2147483648 - 1 cannot be represented in type 'i
* checking PDF version of manual ... OK
* DONE

Status: 1 ERROR
See
  '/mnt/sanitizers.Rcheck/00check.log'
for details
```