Rcpp: Seamless R and C++

Romain François
romain@r-enthusiasts.com

Dirk Eddelbuettel
edd@debian.org

LondonR, Oct 05th, Counting House - 50 Cornhill, London.

Fine for Indiana Jones

Le viaduc de Millau

Plat du jour

1. Appetizers : Some background on R and C++
2. Main course : The Rcpp API
3. Desert : Rcpp sugar
4. Coffee : Rcpp modules

R support for C/C++

- R is a C program
- R supports C++ out of the box, just use a `.cpp` file extension
- R exposes a API based on low level C functions and MACROS.
- R provides several calling conventions to invoke compiled code.

```
SEXP foo( SEXP x1, SEXP x2 ){
   ...
}
```

```
> .Call( "foo", 1:10, rnorm(10) )
```

.Call example

```
#include <R.h>
#include <Rdefines.h>
extern "C"SEXP vectorfoo(SEXP a, SEXP b){
  int i, n;
  double *xa, *xb, *xab; SEXP ab;
  PROTECT(a = AS_NUMERIC(a));
  PROTECT(b = AS_NUMERIC(b));
  n = LENGTH(a);
  PROTECT(ab = NEW_NUMERIC(n));
  xa=NUMERIC_POINTER(a); xb=NUMERIC_POINTER(b);
  xab = NUMERIC_POINTER(ab);
  double x = 0.0, y = 0.0 ;
  for (i=0; i<n; i++) xab[i] = 0.0;
  for (i=0; i<n; i++) {
    x = xa[i]; y = xb[i];
    res[i] = (x < y) ? x*x : -(y*y);
  }
  UNPROTECT(3);
  return(ab);
}
```

.Call example: character vectors

```
> c( "foo", "bar" )
```

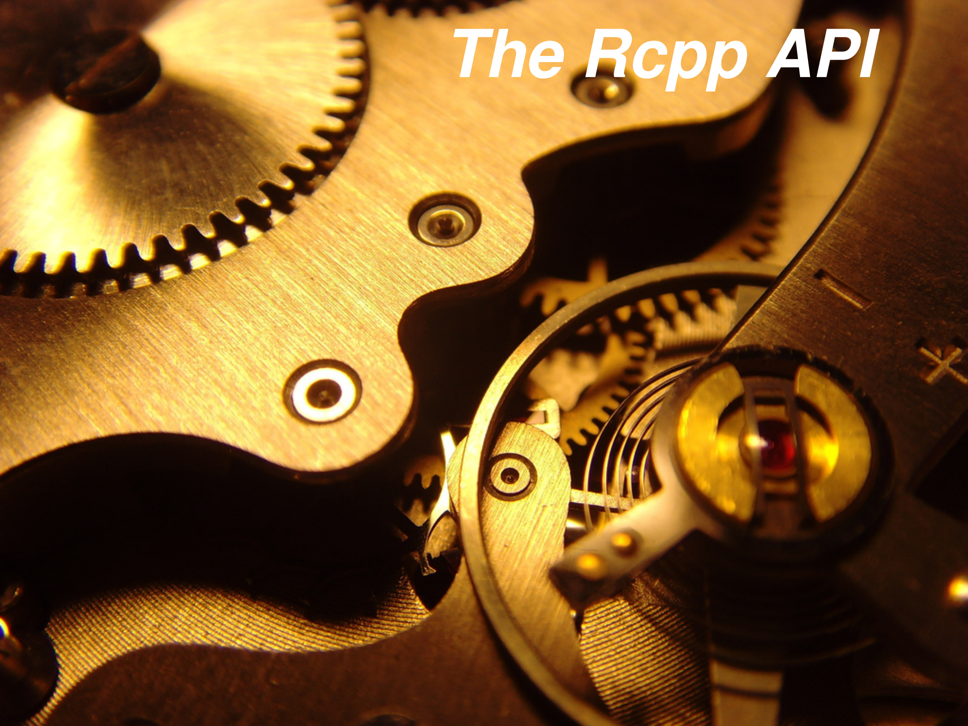
```
#include <R.h>
#include <Rdefines.h>
extern "C"SEXP foobar(){
  SEXP res = PROTECT(allocVector(STRSXP, 2));
  SET_STRING_ELT( res, 0, mkChar( "foo") ) ;
  SET_STRING_ELT( res, 1, mkChar( "bar") ) ;
  UNPROTECT(1) ;
  return res ;
}
```

.Call example: calling an R function

```
> eval( call( "rnorm", 3L, 10.0, 20.0 ) )
```

```
#include <R.h>
#include <Rdefines.h>
extern "C"SEXP callback(){
  SEXP call = PROTECT( LCONS( install("rnorm"),
    CONS( ScalarInteger( 3 ),
      CONS( ScalarReal( 10.0 ),
       CONS( ScalarReal( 20.0 ), R_NilValue )
      )
    )
  ) );
  SEXP res = PROTECT(eval(call, R_GlobalEnv)) ;
  UNPROTECT(2) ;
  return res ;
}
```

The Rcpp API



The Rcpp API

- Encapsulation of R objects (`SEXP`) into C++ classes: `NumericVector`, `IntegerVector`, ..., `Function`, `Environment`, `Language`, ...
- Conversion from R to C++ : `as`
- Conversion from C++ to R : `wrap`
- Interoperability with the Standard Template Library (STL)

The Rcpp API : classes

Rcpp class	R `typeof`
`Integer(Vector|Matrix)`	`integer` vectors and matrices
`Numeric(Vector|Matrix)`	`numeric` ...
`Logical(Vector|Matrix)`	`logical` ...
`Character(Vector|Matrix)`	`character` ...
`Raw(Vector|Matrix)`	`raw` ...
`Complex(Vector|Matrix)`	`complex` ...
`List`	`list` (aka generic vectors) ...
`Expression(Vector|Matrix)`	`expression` ...
`Environment`	`environment`
`Function`	`function`
`XPtr`	`externalptr`
`Language`	`language`
`S4`	`S4`
...	...

The Rcpp API : example

```
SEXP foo( SEXP xs, SEXP ys ){
    Rcpp::NumericVector xx(xs), yy(ys) ;
    int n = xx.size() ;
    Rcpp::NumericVector res( n ) ;
    double x = 0.0, y = 0.0 ;
    for (int i=0; i<n; i++) {
        x = xx[i];
        y = yy[i];
        res[i] = (x < y) ?  x*x :  -(y*y);
    }
    return res ;
}
```

The Rcpp API : example

```
using namespace Rcpp ;
SEXP bar(){
    std::vector<double> z(10) ;
    List res = List::create(
      _["foo"] = NumericVector::create(1,2),
      _["bar"] = 3,
      _["bla"] = "yada yada",
      _["blo"] = z
      ) ;
    res.attr("class") = "myclass";
    return res ;
}
```

The Rcpp API : conversion from R to C++

`Rcpp::as<T>` handles conversion from `SEXP` to `T`.

```
template <typename T> T as( SEXP m_sexp)
    throw(not_compatible) ;
```

`T` can be:

- primitive type : `int`, `double`, `bool`, `long`, `std::string`
- any type that has a constructor taking a `SEXP`
- ... that specializes the `as` template
- ... that specializes the `Exporter` class template
- containers from the STL

more details in the `Rcpp-extending` vignette.

The Rcpp API : conversion from C++ to R

`Rcpp::wrap<T>` handles conversion from `T` to `SEXP`.

```
template <typename T>
SEXP wrap( const T& object ) ;
```

`T` can be:

- primitive type : `int`, `double`, `bool`, `long`, `std::string`
- any type that has a an operator `SEXP`
- ... that specializes the `wrap` template
- ... that has a nested type called `iterator` and member functions `begin` and `end`
- containers from the STL `vector<T>`, `list<T>`, `map<string,T>`, etc ... (where `T` is itself wrappable)

more details in the `Rcpp-extending` vignette.

The Rcpp API : conversion examples

```
typedef std::vector<double> Vec ;
int x_= as<int>( x ) ;
double y_= as<double>( y_) ;
VEC z_= as<VEC>( z_) ;

wrap( 1 ) ; // INTSXP
wrap( "foo") ; // STRSXP

typedef std::map<std::string,Vec> Map ;
Map foo( 10 ) ;
Vec f1(4) ;
Vec f2(10) ;
foo.insert( "x", f1 ) ;
foo.insert( "y", f2 ) ;
wrap( foo ) ; // named list of numeric vectors
```

The Rcpp API : *implicit* conversion examples

```
Environment env = ...  ;
List list = ...  ;
Function rnorm( "rnorm") ;

// implicit calls to as
int x = env["x"] ;
double y = list["y"];

// implicit calls to wrap
rnorm( 100, _["mean"] = 10 ) ;
env["x"] = 3;
env["y"] = "foo";
List::create( 1, "foo", 10.0, false ) ;
```

Rcpp sugar

Sugar : motivation

```
int n = x.size() ;
NumericVector res1( n ) ;
double x_= 0.0, y_= 0.0 ;
for( int i=0; i<n; i++){
        x_= x[i] ;y_= y[i] ;
        if( R_IsNA(x_) ||R_IsNA(y_) ){
            res1[i] = NA_REAL;
        } else if( x_< y_){
            res1[i] = x_* x_;
        } else {
            res1[i] = -( y_* y_)  ;
        }
    }
```

Sugar : motivation

We missed the R syntax :

```
> ifelse( x < y, x*x, -(y*y) )
```

Sugar : motivation

We missed the R syntax :

```
> ifelse( x < y, x*x, -(y*y) )
```

sugar brings it into C++

```
SEXP foo( SEXP xx, SEXP yy){
    NumericVector x(xx), y(yy) ;
    return ifelse( x < y, x*x, -(y*y) ) ;
}
```

Sugar : another example

```
double square( double x){
  return x*x ;
}

SEXP foo( SEXP xx ){
  NumericVector x(xx) ;
  return sapply( x, square ) ;
}
```

Sugar : contents

- logical operators: <, >, <=, >=, ==, !=
- arithmetic operators: +, −, *, /
- functions on vectors: `abs`, `all`, `any`, `ceiling`, `diag`, `diff`, `exp`, `head`, `ifelse`, `is_na`, `lapply`, `pmin`, `pmax`, `pow`, `rep`, `rep_each`, `rep_len`, `rev`, `sapply`, `seq_along`, `seq_len`, `sign`, `tail`
- functions on matrices: `outer`, `col`, `row`, `lower_tri`, `upper_tri`, `diag`
- statistical functions (dpqr) : `rnorm`, `dpois`, `qlogis`, etc ...

More information in the `Rcpp-sugar` vignette.

Sugar : benchmarks

expression	sugar	R	R / sugar
`any(x*y<0)`	0.000447	4.86	10867
`ifelse(x<y,x*x,-(y*y))`	1.331	22.29	16.74
`ifelse(x<y,x*x,-(y*y))` (⋆)	0.832	21.59	24.19
`sapply(x,square)`	0.240	138.71	577.39

Benchmarks performed on OSX SL / R 2.12.0 alpha (64 bit) on a MacBook Pro (i5).

⋆ : version includes optimization related to the absence of missing values

Sugar : benchmarks

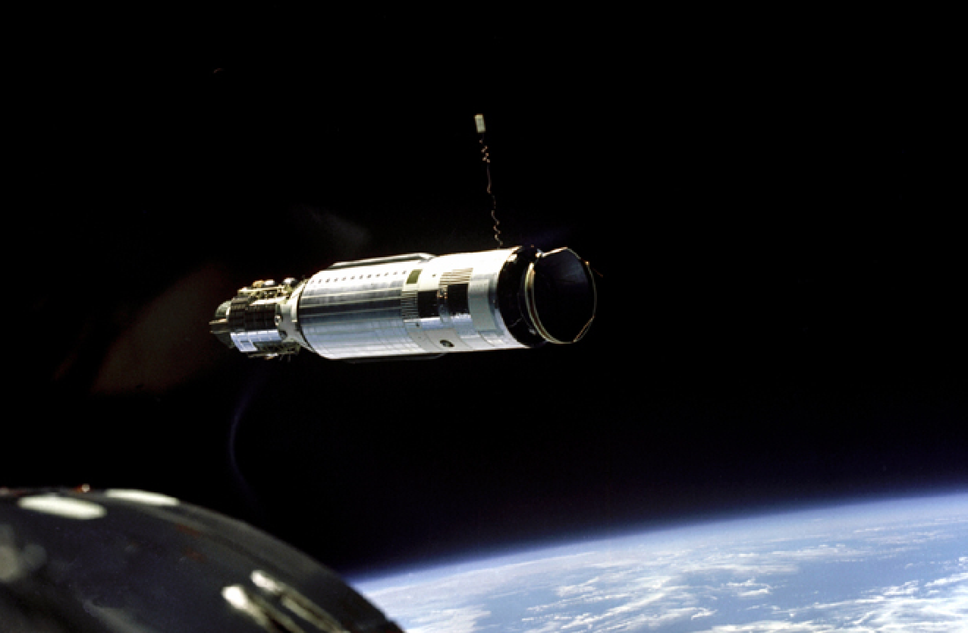
Benchmarks of the convolution example from Writing R Extensions.

Implementation	Time in millisec	Relative to R API
R API (as benchmark)	218	
Rcpp sugar	145	0.67
`NumericVector::iterator`	217	1.00
`NumericVector::operator[]`	282	1.29
`RcppVector<double>`	683	3.13

Table: Convolution of x and y (200 values), repeated 5000 times.

Extract from the article *Rcpp: Seamless R and C++ integration*, accepted for publication in the R Journal.

Rcpp modules



Modules: expose C++ to R

```
const char* hello( const std::string& who ){
    std::string result( "hello ") ;
    result += who ;
    return result.c_str() ;
}

RCPP_MODULE(yada){
    using namespace Rcpp ;
    function( "hello", &hello ) ;
}
```

```
> yada <- Module( "yada" )
> yada$hello( "world" )
```

Modules: expose C++ classes to R

```
class World {
public:
    World() :  msg("hello"){}
    void set(std::string msg) {
        this->msg = msg;
    }
    std::string greet() {
        return msg;
    }
private:
    std::string msg;
};

void clearWorld( World* w){
    w->set( "") ;
}
```

Modules: expose C++ classes to R

C++ side: declare *what* to expose

```
RCPP_MODULE(yada){
    using namespace Rcpp ;

    class_<World>( "World")
        .method( "greet", &World::greet )
        .method( "set", &World::set )
        .method( "clear", &clearWorld )
    ;

}
```

Modules: on the R side

R side: based on R 2.12.0 reference classes (see `?ReferenceClasses`)

```
> World <- yada$World
> w <- new( World )
> w$greet()
[1] "hello"

> w$set( "hello world")
> w$greet()
[1] "hello world"

> w$clear()
> w$greet()
[1] ""
```

Want to learn more ?

- Check the vignettes
- Questions on the `Rcpp-devel` mailing list
- Hands-on training courses
- Commercial support

Romain François **`romain@r-enthusiasts.com`**
Dirk Eddelbuettel **`edd@debian.org`**