Seamless R and C++
Integration with Rcpp:
Part 3 – Detailed Example

Dirk Eddelbuettel
dirk.eddelbuettel@R-Project.org

Center for Research Methods and Data Analysis
University of Kansas
November 16, 2013
Outline

1. Simulation
Where does our computation spend its time?
Profiling is a tool which may tell us

Having used Rcpp and the C++ compiler to speed up\(^1\)

- loops,
- function calls,
- element access,
- algebraic expressions
- and more,

what remains as a (sometimes considerable, but often also marginal) bottleneck?

\(^1\)List is neither exhaustive nor sorted but for illustration.
Where does out computation spend its time?
Unless of course you are doing real work . . .

The random number generator (RNG).

R provides two whole sets, one each for

- **uniform** Witchmann-Hill, Marsaglia-Multicarry, Super-Duper, Mersenne-Twister (the default), Knuth-TAOCP-2002, Knuth-TAOCP, L’Ecuyer-CMRG and user-supplied; and

- **normal** Kinderman-Ramage (and “Buggy Kinderman-Ramage”), Ahrens-Dieter, Box-Muller, Inversion (the default), and user-supplied

How do they perform?
RNG Speed varies
Uniform RNGs

Time for 100 times $1e6$ uniform draws

- WH
- MM
- SD
- MT
- KT
- LE

Time in msec

Dirk Eddelbuettel  
Rcpp Intro & Examples Part 3
RNG Speed varies
Normal RNGs

Time for 100 times 1e6 normal draws

Time in msec

KR  AH  BM  Inv

Dirk Eddelbuettel  Rcpp Intro & Examples Part 3
2 Ziggurat
The Ziggurat Generator
Marsaglia and Tsang, JSS, 2000

#include <math.h>
static unsigned long jz, jsr=123456789;

#define SHR3 (jz=jsr, jsr^=(jsr<<13), jsr^=(jsr>>17), jsr^=(jsr<<5), jz+jsr)
#define UNI (.5 + (signed) SHR3*.2328306e-9)
#define IUNI SHR3

static long hz;
static unsigned long iz, kn[128], ke[256];
static float wn[128], fn[128], we[256], fe[256];

#define RNOR (hz=SHR3, iz=hz&127, (fabs(hz)<kn[iz])? hz*wn[iz] : nfix())

/* nfix() generates variates from the residue when rejection in RNOR occurs. */
float nfix(void) { /* ... */ }

/* This procedure sets the seed and creates the tables */
void zigset(unsigned long jsrseed) { /* ... */ }

Can you spot the generator?
Can you spot another issue?
Using SHR3 in Ziggurat has issues and too short a cycle, use KISS (also by Marsaglia) instead.

```
#define MWC ((znew<<16)+wnew )
#define SHR3 (jz=jsr, jsr^=(jsr<<13), jsr^=(jsr>>17), \ 
             jsr^=(jsr<<5),jz+jsr)
#define CONG (jcong=69069*jcong+1234567)
#define KISS ((MWC^CONG)+SHR3)
#define RNOR (hz=KISS, iz=hz&127, \ 
          (fabs(hz)<kn[iz]) ? hz*wn[iz] : nfix())
```

Fixes one important issue, but another problem remains.
Ziggurat is used in a number of environments that care about speed (eg Julia, Matlab) as well as other open source environments (GNU GSL, GNU Gretl, QuantLib).

It has been reviewed well: Survey by Thomas et al (2007):

[W]hen maintaining extremely high statistical quality is the first priority, and subject to that constraint, speed is also desired, the Ziggurat method will often be the most appropriate choice.

So why no R implementation (apart from a 32-bit only implementation also lacking the Leong et al correction)?
Outline

3 C++ Design
- Base
- Ziggurat
- Factory
A Simple (Virtual) Base Class

Derived classes then implement the virtual functions

```cpp
#include <cmath>
#include <stdint.h> // not cstdint as it needs C++11

namespace Ziggurat {

class Zigg {

public:

    virtual ~Zigg() {};
    virtual void setSeed(const uint32_t s) = 0;
    // no getSeed() as GSL has none
    virtual double norm() = 0;

};

}```
#include <Zigg.h>

namespace Ziggurat { 
namespace Ziggurat { 

class Ziggurat : public Zigg { 

class Ziggurat : public Zigg { 

class Ziggurat : public Zigg { 

class Ziggurat : public Zigg { 

class Ziggurat : public Zigg { 

public:

Ziggurat(uint32_t seed=123456789) : /* ... */ { 

init(); 

setSeed(seed); 

} 

~Ziggurat() {}; 

void setSeed(const uint32_t s) { /* ... */} 

uint32_t getSeed() { return jsr; } 

inline double norm() { return RNOR; } 

} 

private:

private:

private:

private:

private:

private:

void init() { /* ... */ } 

inline float nfix(void) { /* ... */ } 

/* ... */

};

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

} 

}
A Simple GSL Variant
Implemented by Voss (2005), uses Mersenne Twister for uniforms

```cpp
#include <Zigg.h>
namespace Ziggurat { 
namespace GSL { 
    class ZigguratGSL : public Zigg { 
    public:
        ZigguratGSL(uint32_t seed=12345678) { 
            gsl_rng_env_setup();
            r = gsl_rng_alloc (gsl_rng_default);
            gsl_rng_set(r, seed);
        }
    ~ZigguratGSL() { gsl_rng_free(r); }
    double norm() { 
        return gsl_ran_gaussian_ziggurat(r, 1.0);
    }
    void setSeed(const uint32_t seed) { 
        gsl_rng_set(r, seed);
    }
    private: 
        gsl_rng *r;
    }
};
```
A Ziggurat Generator Factory

Based on generator name seed, we get a Ziggurat instance

Now we can instantiate based on user-supplied id:

```cpp
Zigg* getZiggurat(const std::string generator, const int seed) {
    if (generator == "MT") {
        return new ZigguratMT(seed);
    } else if (generator == "LZLLV") {
        return new ZigguratLZLLV(seed);
    } else if (generator == "Ziggurat") {
        return new Ziggurat(seed);
    } else if (generator == "GSL") {
        return new ZigguratGSL(seed);
    } else if (generator == "QL") {
        return new ZigguratQL(seed);
    } else if (generator == "Gretl") {
        return new ZigguratGretl(seed);
    }

    Rcpp::Rcout << "Unrecognised generator: " << generator << std::endl;
    return NULL;
}
```
Outline

4. R Usage
   - Rcpp
   - R access
   - Speed
#include <Rcpp.h>

#include <ZigguratMT.h>

// Version 1 -- Derived from Marsaglia and Tsang, JSS, 2000

static Ziggurat::MT::ZigguratMT ziggmt;

// Marsaglia and Tsang (JSS,2000)

[[Rcpp::export]]
Rcpp::NumericVector zrnormMT(int n) {
    Rcpp::NumericVector x(n);
    for (int i=0; i<n; i++) {
        x[i] = ziggmt.norm();
    }
    return x;
}

// [[Rcpp::export]]
void zsetseedMT(int s) {
    ziggmt.setSeed(s);
}
R usage
As Rcpp Attributes makes it a breeze

```r
zsetseedMT(123456)
zrnormMT(5)
```

```r
## [1] 0.079374  -1.231947  -0.007092  -0.332614  -1.024207
zrnormMT(5)
```

```r
## [1] 0.6725  0.3251  1.2661  -0.6573  0.2544
zsetseedMT(123456)
zrnormMT(5)
```

```r
## [1] 0.079374  -1.231947  -0.007092  -0.332614  -1.024207
```
Ziggurat Speed
R versus different Ziggurat implementations

Time for 100 times 1e6 normal RNGs

Time in msec:

- R Inv
- Zigg
- ZiggGSL
- ZiggQL
- ZiggGretl

Dirk Eddelbuettel  Rcpp Intro & Examples Part 3
### Ziggurat Speed - Table

R versus different Ziggurat implementations

<table>
<thead>
<tr>
<th>#</th>
<th>Test</th>
<th>Replications</th>
<th>Elapsed</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>zrnorm(N)</td>
<td>100</td>
<td>2.573</td>
<td>1.000</td>
</tr>
<tr>
<td>3</td>
<td>zrnormGSL(N)</td>
<td>100</td>
<td>2.863</td>
<td>1.113</td>
</tr>
<tr>
<td>4</td>
<td>zrnormQL(N)</td>
<td>100</td>
<td>4.964</td>
<td>1.929</td>
</tr>
<tr>
<td>5</td>
<td>zrnormGl(N)</td>
<td>100</td>
<td>5.667</td>
<td>2.202</td>
</tr>
<tr>
<td>1</td>
<td>rnorm(N)</td>
<td>100</td>
<td>33.382</td>
<td>12.974</td>
</tr>
</tbody>
</table>

A ten to twelve-fold gain without trying all that hard ...
5 Tests
- Chi-squared (cf Leong et al, JSS, 2005)
- Standard Test
- Normal Test
Count draws in evenly spaced grid
Standard chi-square test given normal dist. cdf – 5% crit. value below
Given *uniform* generator, take \( n \) draws from a \( U(0, 1) \). Compute sum of these \( n \) values. Repeat this \( m \) times.

With \( n \) large enough, \( m \) results converge towards a \( N(n/2, \sqrt{n/12}) \) (ie. the Irwin-Hall dist. of sum of uniforms).

Given the asymptotics, we can construct \( p_i \) for each of the \( m \) values using the inverse of Normal using the known mean and standard deviation.

We now have \( m \) uniformly distributed values.

Use a standard Kolmogorov-Smirnov or Wilcoxon test against departures from the uniform.
Standard Test
Based on 5e10 draws
Basic Design for Normal RNG Test
Suggesting simple extension of Standard Test

- Take $n$ draws from $N(0, 1)$ distribution. Compute sum of these $n$ values. Repeat this $M$ times.
- With $n$ large enough, the $m$ results converge towards a $N(0, \sqrt{N})$.
- Given this result, we can construct $p_i$ for each of the $m$ values using the inverse of the Normal using the known mean mean and standard deviation.
- We now have $m$ uniformly distributed values.
- Use a standard Kolmogorov-Smirnoff or Wilcoxon test against departures from the uniform.
Normal Test
Based on 5e10 draws

Normal test results

Ziggurat

MT

LZLLV

GSL

QL

Gretl

pKS: 0.7199
pWil.: 0.3577

0.0 0.2 0.4 0.6 0.8 1.0

0.0 0.2 0.4 0.6 0.8 1.0

Ziggurat

x

Fn(x)

pKS: 0
pWil.: 0.0891

0.0 0.2 0.4 0.6 0.8 1.0

0.0 0.2 0.4 0.6 0.8 1.0

MT

x

Fn(x)

pKS: 0.8329
pWil.: 0.7635

0.0 0.2 0.4 0.6 0.8 1.0

0.0 0.2 0.4 0.6 0.8 1.0

LZLLV

x

Fn(x)

pKS: 0.3134
pWil.: 0.137

0.0 0.2 0.4 0.6 0.8 1.0

0.0 0.2 0.4 0.6 0.8 1.0

GSL

x

Fn(x)

pKS: 0.706
pWil.: 0.8138

0.0 0.2 0.4 0.6 0.8 1.0

0.0 0.2 0.4 0.6 0.8 1.0

QL

x

Fn(x)

pKS: 0.91
pWil.: 0.7245

0.0 0.2 0.4 0.6 0.8 1.0

0.0 0.2 0.4 0.6 0.8 1.0

Gretl

x

Fn(x)

Draws:5e+09 Repeats: 100 Seed: 1234567890 Created at: 2013-11-13 17:51:09 Version: 0.0.2.20131112

Dirk Eddelbuettel
Rcpp Intro & Examples Part 3
Outline

Conclusion
  • Summary
  • Next Steps
Ziggurat Revisited

- Updated Ziggurat: portable code for 32 + 64 bit OSs
- Clean C++ wrapper facilitates comparison & testing
- Reconfirmed Leong et al of MT failing at 32 bit limit
- Applied standard random number test (for uniforms); confirming result
- Suggested new test based directly on normal random deviates; also confirms MT issue
- Tested + validated three other OS implementations
- Timings favourable: our variant fastest across other Ziggurat implementations and the R generators
TODOs and Extensions
Things to consider

- Use (R)DieHarder and Big Crush (TestU01) tests?
- R integration – straightforward via given interface
- Look into other $U(0, 1)$ generators, too bad R does not let us access its Mersenne Twister (deliberate, see Writing R Extensions)
- Look into more Open Source variants to test?
- Look at Doornik’s paper & code (not Open Source)
- Update package, complete vignette
- See the Github repo and CRAN versions
Outline

7 Rcpp Resources
Rcpp Resources

Site  http://www.rcpp.org
Book  http://www.rcpp.org/book
Gallery  http://gallery.rcpp.org
Code  http://github.org/RcppCore/Rcpp
Blog  http://dirk.eddelbuettel.com/blog/code/rcpp
List  http://lists.r-forge.r-project.org/mailman/listinfo/rcpp-devel