The Scalable Parallel Random Number Generators (SPRNG) Library

Prof. Michael Mascagni

Departments of Computer Science, Mathematics & Scientific Computing
Florida State University, Tallahassee, FL 32306 USA
E-mail: mascagni@fsu.edu
URL: http://www.cs.fsu.edu/~mascagni
Outline of the Talk

Where to Get SPRNG

How to Build SPRNG
  Testing SPRNG

How SPRNG is Structured
  Specific Generator Details

How to Use SPRNG
  Class Structure and Simple modes
  Random Number Parameters
  Usage Examples
    Usage Example - Default Interface
    Usage Example - Simple Interface

Other Parts of Interest in SPRNG
  Other Parts - Examples Folder
  Other Parts - Tests Folder

SPRNG’s Future

References
The main web site for SPRNG is located at URLs: http://sprng.cs.fsu.edu or http://www.sprng.org
Where to Get SPRNG

- The main web site for SPRNG is located at URLs: http://sprng.cs.fsu.edu or http://www.sprng.org
- Many versions available.
Where to Get **SPRNG**

- The main web site for **SPRNG** is located at URLs: [http://sprng.cs.fsu.edu](http://sprng.cs.fsu.edu) or [http://www.sprng.org](http://www.sprng.org)
- Many versions available.
- Latest version 4.0 which is C++
Where to Get SPRNG

- The main web site for SPRNG is located at URLs: http://sprng.cs.fsu.edu or http://www.sprng.org
- Many versions available.
- Latest version 4.0 which is C++
- The 4.0 page gives info pages to 4.0 page info
Where to Get SPRNG

- The main web site for SPRNG is located at URLs: http://sprng.cs.fsu.edu or http://www.sprng.org
- Many versions available.
- Latest version 4.0 which is C++
- The 4.0 page gives info pages to 4.0 page info
  - Quick Start
Where to Get SPRNG

▶ The main web site for SPRNG is located at
  URLs: http://sprng.cs.fsu.edu or
  http://www.sprng.org

▶ Many versions available.

▶ Latest version 4.0 which is C++

▶ The 4.0 page gives info pages to 4.0 page info
  ▶ Quick Start
  ▶ Quick Reference
Where to Get SPRNG

The main web site for SPRNG is located at URLs: http://sprng.cs.fsu.edu or http://www.sprng.org

Many versions available.

Latest version 4.0 which is C++

The 4.0 page gives info pages to 4.0 page info
  - Quick Start
  - Quick Reference
  - User’s Guide
Where to Get SPRNG

- The main web site for SPRNG is located at
  URLs: http://sprng.cs.fsu.edu or
  http://www.sprng.org
- Many versions available.
- Latest version 4.0 which is C++
- The 4.0 page gives info pages to 4.0 page info
  - Quick Start
  - Quick Reference
  - User’s Guide
  - Reference Manual
Where to Get SPRNG

- The main web site for SPRNG is located at URLs: http://sprng.cs.fsu.edu or http://www.sprng.org
- Many versions available.
- Latest version 4.0 which is C++
- The 4.0 page gives info pages to 4.0 page info
  - Quick Start
  - Quick Reference
  - User’s Guide
  - Reference Manual
  - Examples
How to Build SPRNG

▶ zcat sprng4.tar.gz | tar xovf -
How to Build SPRNG

- `zcat sprng4.tar.gz | tar xovf -`
- `cd sprng4`
How to Build SPRNG

- zcat sprng4.tar.gz | tar xofv -
- cd sprng4
- Run ./configure
How to Build SPRNG

How to Build SPRNG

- `zcat sprng4.tar.gz | tar xzvf -`
- `cd sprng4`
- `Run ./configure`
- `Run make`

NB: Sometimes 'make' has errors on some parts which can be ignored. In these cases, 'make -k' can be used to continue compiling even if there are errors.

The MPI programs sometimes need special configuring.
How to Build SPRNG

How to Build SPRNG

- zcat sprng4.tar.gz | tar xofv -
- cd sprng4
- Run ./configure
- Run make
- NB: Sometimes 'make' has errors on some parts which can be ignored. In these cases, 'make -k' can be used to continue compiling even if there are errors.
How to Build SPRNG

How to Build SPRNG

- `zcat sprng4.tar.gz | tar xovf -`
- `cd sprng4`
- Run `./configure`
- Run `make`
- NB: Sometimes `make` has errors on some parts which can be ignored. In these cases, `make -k` can be used to continue compiling even if there are errors.
- The MPI programs sometimes need special configuring.
Testing SPRNG

How to check the build

▶ Go to directory check, and run `.\checksprng`.
How to check the build

- Go to directory check, and run `./checksprng`.
- This program checks to see if SPRNG has been correctly installed.
Testing SPRNG

How to check the build

▶ Go to directory check, and run `./checksprng`.
▶ This program checks to see if SPRNG has been correctly installed.
▶ The check folder contains a single program which generates known sequences and checks this against a data file.
How SPRNG is Structured

- Directories in SPRNG

- SRC - Source code for SPRNG.
- EXAMPLES - Examples of SPRNG usage. All MPI examples are placed in subdirectory mpisprng. If MPI is installed on your machine, then all MPI examples will be automatically installed.
- TESTS - Empirical and physical tests for SPRNG generators. All MPI tests are stored in subdirectory mpitests. If MPI is installed on your machine, then all MPI tests will be automatically installed.
- check - contains executables ./checksprng and ./timesprng.
- lib - contains SPRNG library sprng after successful installation.
- include - SPRNG header files.
How SPRNG is Structured

- Directories in SPRNG
  - SRC - Source code for SPRNG.
How SPRNG is Structured

▶ Directories in SPRNG
  ▶ SRC - Source code for SPRNG.
  ▶ EXAMPLES - Examples of SPRNG usage. All MPI examples are placed in subdirectory mpisprng. If MPI is installed on your machine, then all MPI examples will be automatically installed.
How SPRNG is Structured

Directories in SPRNG

- **SRC** - Source code for SPRNG.
- **EXAMPLES** - Examples of SPRNG usage. All MPI examples are placed in subdirectory `mpisprng`. If MPI is installed on your machine, then all MPI examples will be automatically installed.
- **TESTS** - Empirical and physical tests for SPRNG generators. All MPI tests are stored in subdirectory `mpitests`. If MPI is installed on your machine, then all MPI tests will be automatically installed.
How SPRNG is Structured

Directories in SPRNG

- **SRC** - Source code for SPRNG.
- **EXAMPLES** - Examples of SPRNG usage. All MPI examples are placed in subdirectory `mpisprng`. If MPI is installed on your machine, then all MPI examples will be automatically installed.
- **TESTS** - Empirical and physical tests for SPRNG generators. All MPI tests are stored in subdirectory `mpitests`. If MPI is installed on your machine, then all MPI tests will be automatically installed.
- **check** - contains executables `./checksprng` and `./timesprng`.

lib - contains SPRNG library `libsprng` after successful installation.

include - SPRNG header files.
How SPRNG is Structured

Directories in SPRNG

- **SRC** - Source code for SPRNG.
- **EXAMPLES** - Examples of SPRNG usage. All MPI examples are placed in subdirectory `mpisprng`. If MPI is installed on your machine, then all MPI examples will be automatically installed.
- **TESTS** - Empirical and physical tests for SPRNG generators. All MPI tests are stored in subdirectory `mpitests`. If MPI is installed on your machine, then all MPI tests will be automatically installed.
- **check** - contains executables `./checksprng` and `./timesprng`.
- **lib** - contains SPRNG library `libsprng` after successful installation.
How SPRNG is Structured

- **Directories in SPRNG**
  - **SRC** - Source code for SPRNG.
  - **EXAMPLES** - Examples of SPRNG usage. All MPI examples are placed in subdirectory mpisprng. If MPI is installed on your machine, then all MPI examples will be automatically installed.
  - **TESTS** - Empirical and physical tests for SPRNG generators. All MPI tests are stored in subdirectory mpitests. If MPI is installed on your machine, then all MPI tests will be automatically installed.
  - **check** - contains executables ./checksprng and ./timesprng.
  - **lib** - contains SPRNG library libsprng after successful installation.
  - **include** - SPRNG header files.
Predefined Generators

Types of generators

- Types of generators
Predefined Generators

Types of generators

- 0: Modified Lagged-Fibonacci Generator (lfg)
Predefined Generators

Types of generators

- 0: Modified Lagged-Fibonacci Generator (lfg)
- 1: 48-Bit Linear Congruential Generator w/Prime Addend (lcg)
- 2: 64-Bit Linear Congruential Generator w/Prime Addend (lcg64)
- 3: Combined Multiple Recursive Generator (cmrg)
- 4: Multiplicative Lagged-Fibonacci Generator (mlfg)
- 5: Prime Modulus Linear Congruential Generator (pmlcg)
Predefined Generators

Types of generators

- 0: Modified Lagged-Fibonacci Generator (lfg)
- 1: 48-Bit Linear Congruential Generator w/Prime Addend (lcg)
- 2: 64-Bit Linear Congruential Generator w/Prime Addend (lcg64)
Predefined Generators

Types of generators

- 0: Modified Lagged-Fibonacci Generator (lfg)
- 1: 48-Bit Linear Congruential Generator w/Prime Addend (lcg)
- 2: 64-Bit Linear Congruential Generator w/Prime Addend (lcg64)
- 3: Combined Multiple Recursive Generator (cmrg)
Predefined Generators

Types of generators

- 0: Modified Lagged-Fibonacci Generator (lfg)
- 1: 48-Bit Linear Congruential Generator w/Prime Addend (lcg)
- 2: 64-Bit Linear Congruential Generator w/Prime Addend (lcg64)
- 3: Combined Multiple Recursive Generator (cmrg)
- 4: Multiplicative Lagged-Fibonacci Generator (mlfg)
Predefined Generators

Types of generators

- 0: Modified Lagged-Fibonacci Generator (lfg)
- 1: 48-Bit Linear Congruential Generator w/Prime Addend (lcg)
- 2: 64-Bit Linear Congruential Generator w/Prime Addend (lcg64)
- 3: Combined Multiple Recursive Generator (cmrg)
- 4: Multiplicative Lagged-Fibonacci Generator (mlfg)
- 5: Prime Modulus Linear Congruential Generator (pmlcg)
Predefined Generators

Types of generators

- 0: Modified Lagged-Fibonacci Generator (lfg)
- 1: 48-Bit Linear Congruential Generator w/Prime Addend (lcg)
- 2: 64-Bit Linear Congruential Generator w/Prime Addend (lcg64)
- 3: Combined Multiple Recursive Generator (cmrg)
- 4: Multiplicative Lagged-Fibonacci Generator (mlfg)
- 5: Prime Modulus Linear Congruential Generator (pmlcg)

The number represents the type of generator in the Class interface
Specific Generator Details

1. lfg: Modified-Lagged Fibonacci Generator (the default generator)
Specific Generator Details

1. \texttt{lfg}: Modified-Lagged Fibonacci Generator (the default generator)
   \[ z_n = x_n \text{ XOR } y_n \]
Specific Generator Details

1. \texttt{lfg}: Modified-Lagged Fibonacci Generator (the default generator)
   - \( z_n = x_n \ XOR \ y_n \)
   - \( x_n = x_{n-k} + x_{n-l} \mod M \)
Specific Generator Details

1. **lfg**: Modified-Lagged Fibonacci Generator (the default generator)
   - \( z_n = x_n \ XOR \ y_n \)
   - \( x_n = x_{n-k} + x_{n-l}(\ mod \ M) \)
   - \( y_n = y_{n-k} + y_{n-l}(\ mod \ M) \)
Specific Generator Details

1. \texttt{lfg}: Modified-Lagged Fibonacci Generator (the default generator)
   - \[ z_n = x_n \text{ XOR } y_n \]
   - \[ x_n = x_{n-k} + x_{n-l} \mod M \]
   - \[ y_n = y_{n-k} + y_{n-l} \mod M \]

2. \texttt{lcg}: 48-Bit Linear Congruential Generator w/Prime Addend
Specific Generator Details

1. **lfg**: Modified-Lagged Fibonacci Generator (the default generator)
   - $z_n = x_n \ XOR \ y_n$
   - $x_n = x_{n-k} + x_{n-l} \ (mod \ M)$
   - $y_n = y_{n-k} + y_{n-l} \ (mod \ M)$

2. **lcg**: 48-Bit Linear Congruential Generator w/Prime Addend
   - $x_n = a x_{n-1} + p \ (mod \ M)$
Specific Generator Details

1. \texttt{lfg}: Modified-Lagged Fibonacci Generator (the default generator)
   - \( z_n = x_n \text{ XOR } y_n \)
   - \( x_n = x_{n-k} + x_{n-l} \mod M \)
   - \( y_n = y_{n-k} + y_{n-l} \mod M \)

2. \texttt{lcg}: 48-Bit Linear Congruential Generator w/Prime Addend
   - \( x_n = ax_{n-1} + p \mod M \)
   - \( p \) is a prime addend
Specific Generator Details

1. **lfg**: Modified-Lagged Fibonacci Generator (the default generator)
   - \( z_n = x_n \ XOR \ y_n \)
   - \( x_n = x_{n-k} + x_{n-l}(mod \ M) \)
   - \( y_n = y_{n-k} + y_{n-l}(mod \ M) \)

2. **lcg**: 48-Bit Linear Congruential Generator w/Prime Addend
   - \( x_n = ax_{n-1} + p(mod \ M) \)
   - \( p \) is a prime addend
   - \( a \) is the multiplier

   

   The 48-bit LCG, except that the arithmetic is modulo \( 2^{48} \)

   The multipliers and prime addends for this generator are different from those for the 48-bit generator
Specific Generator Details

1. lfg: Modified-Lagged Fibonacci Generator (the default generator)
   - \( z_n = x_n \ XOR \ y_n \)
   - \( x_n = x_{n-k} + x_{n-l} (mod \ M) \)
   - \( y_n = y_{n-k} + y_{n-l} (mod \ M) \)

2. lcg: 48-Bit Linear Congruential Generator w/Prime Addend
   - \( x_n = ax_{n-1} + p (mod \ M) \)
   - \( p \) is a prime addend
   - \( a \) is the multiplier
   - \( M \) for this generator is \( 2^{48} \)
Specific Generator Details

1. \texttt{lfg}: Modified-Lagged Fibonacci Generator (the default generator)
   - \( z_n = x_n \text{ XOR } y_n \)
   - \( x_n = x_{n-k} + x_{n-l} \mod M \)
   - \( y_n = y_{n-k} + y_{n-l} \mod M \)

2. \texttt{lcg}: 48-Bit Linear Congruential Generator w/Prime Addend
   - \( x_n = ax_{n-1} + p \mod M \)
   - \( p \) is a prime addend
   - \( a \) is the multiplier
   - \( M \) for this generator is \( 2^{48} \)

3. \texttt{lcg64}: 64-Bit Linear Congruential Generator w/Prime Addend
Specific Generator Details

1. **lfg**: Modified-Lagged Fibonacci Generator (the default generator)
   - \( z_n = x_n \text{ XOR } y_n \)
   - \( x_n = x_{n-k} + x_{n-l} \mod M \)
   - \( y_n = y_{n-k} + y_{n-l} \mod M \)

2. **lcg**: 48-Bit Linear Congruential Generator w/Prime Addend
   - \( x_n = ax_{n-1} + p \mod M \)
   - \( p \) is a prime addend
   - \( a \) is the multiplier
   - \( M \) for this generator is \( 2^{48} \)

3. **lcg64**: 64-Bit Linear Congruential Generator w/Prime Addend
   - The 48-bit LCG, except that the arithmetic is modulo \( 2^{64} \)
Specific Generator Details

1. lfg: Modified-Lagged Fibonacci Generator (the default generator)
   - $z_n = x_n \ XOR \ y_n$
   - $x_n = x_{n-k} + x_{n-l} (mod \ M)$
   - $y_n = y_{n-k} + y_{n-l} (mod \ M)$

2. lcg: 48-Bit Linear Congruential Generator w/Prime Addend
   - $x_n = ax_{n-1} + p (mod \ M)$
   - $p$ is a prime addend
   - $a$ is the multiplier
   - $M$ for this generator is $2^{48}$

3. lcg64: 64-Bit Linear Congruential Generator w/Prime Addend
   - The 48-bit LCG, except that the arithmetic is modulo $2^{64}$
   - The multipliers and prime addends for this generator are different from those for the 48-bit generator
Specific Generator Details

4. cmrg: Combined Multiple Recursive Generator
Specific Generator Details

4. cmrg: Combined Multiple Recursive Generator
   ▶ \( z_n = x_n + y_n \times 2^{32} \mod 2^{64} \)
4. cmrg: Combined Multiple Recursive Generator
   - \( z_n = x_n + y_n \cdot 2^{32} \mod 2^{64} \)
   - \( x_n \) is the sequence generated by the 64 bit Linear Congruential Generator
Specific Generator Details

4. **cmrg**: Combined Multiple Recursive Generator
   - \[ z_n = x_n + y_n \cdot 2^{32} (\mod 2^{64}) \]
   - \( x_n \) is the sequence generated by the 64 bit Linear Congruential Generator
   - \( y_n \) is the sequence generated by the following prime modulus Multiple Recursive Generator
Specific Generator Details

4. **cmrg**: Combined Multiple Recursive Generator
   - \( z_n = x_n + y_n \times 2^{32} \mod 2^{64} \)
   - \( x_n \) is the sequence generated by the 64 bit Linear Congruential Generator
   - \( y_n \) is the sequence generated by the following prime modulus Multiple Recursive Generator

5. **mlfg**: Multiplicative Lagged-Fibonacci Generator
Specific Generator Details

4. **cmrg**: Combined Multiple Recursive Generator
   - $z_n = x_n + y_n \times 2^{32} (mod 2^{64})$
   - $x_n$ is the sequence generated by the 64 bit Linear Congruential Generator
   - $y_n$ is the sequence generated by the following prime modulus Multiple Recursive Generator

5. **mlfg**: Multiplicative Lagged-Fibonacci Generator
   - $x_n = x_{n-k} \times x_{n-l} (mod M)$
Specific Generator Details

4. **cmrg**: Combined Multiple Recursive Generator
   - \( z_n = x_n + y_n \times 2^{32} \pmod{2^{64}} \)
   - \( x_n \) is the sequence generated by the 64 bit Linear Congruential Generator
   - \( y_n \) is the sequence generated by the following prime modulus Multiple Recursive Generator

5. **mlfg**: Multiplicative Lagged-Fibonacci Generator
   - \( x_n = x_{n-k} \times x_{n-l} \pmod{M} \)
   - \( l \) and \( k \) are called the lags of the generator, with convention that \( l > k \).
Specific Generator Details

4. cmrg: Combined Multiple Recursive Generator
   - $z_n = x_n + y_n \times 2^{32} (mod \ 2^{64})$
   - $x_n$ is the sequence generated by the 64 bit Linear Congruential Generator
   - $y_n$ is the sequence generated by the following prime modulus Multiple Recursive Generator

5. mlfg: Multiplicative Lagged-Fibonacci Generator
   - $x_n = x_{n-k} \times x_{n-l} (mod \ M)$
   - $l$ and $k$ are called the lags of the generator, with convention that $l > k$.
   - $M$ is chosen to be $2^{64}$
Specific Generator Details

4. cmrg: Combined Multiple Recursive Generator
   ▶ $z_n = x_n + y_n \cdot 2^{32} (mod \ 2^{64})$
   ▶ $x_n$ is the sequence generated by the 64 bit Linear Congruential Generator
   ▶ $y_n$ is the sequence generated by the following prime modulus Multiple Recursive Generator

5. mlfg: Multiplicative Lagged-Fibonacci Generator
   ▶ $x_n = x_{n-k} \cdot x_{n-l} (mod \ M)$
   ▶ $l$ and $k$ are called the lags of the generator, with convention that $l > k$
   ▶ $M$ is chosen to be $2^{64}$

6. pmlcg: Prime Modulus Linear Congruential Generator
Specific Generator Details

4. **cmrg**: Combined Multiple Recursive Generator
   - \( z_n = x_n + y_n \times 2^{32} \mod 2^{64} \)
   - \( x_n \) is the sequence generated by the 64 bit Linear Congruential Generator
   - \( y_n \) is the sequence generated by the following prime modulus Multiple Recursive Generator

5. **mlfg**: Multiplicative Lagged-Fibonacci Generator
   - \( x_n = x_{n-k} \times x_{n-l} \mod M \)
   - \( l \) and \( k \) are called the lags of the generator, with convention that \( l > k \).
   - \( M \) is chosen to be \( 2^{64} \)

6. **pmlcg**: Prime Modulus Linear Congruential Generator
   - \( x_n = a \times x_{n-1} \mod 2^{61} - 1 \)
Default Interface

Sprng(int streamnum, int nstreams, int seed, int param) (Constructor)
Default Interface

- **Sprng(int streamnum, int nstreams, int seed, int param)** (Constructor)
- double sprng() - The next random number in [0,1) is returned
Default Interface

- Sprng(int streamnum, int nstreams, int seed, int param) (Constructor)
- double sprng() - The next random number in [0,1) is returned
- int isprng() - The next random number in [0,2^{31}) is returned
Simple Interface

- int * init_sprng(int seed, int param, int rng_type = 0)
- The next random number in \([0, 1)\) is returned
- int isprng() - The next random number in \([0, 2^{31})\) is returned
Simple Interface

- `int * init_sprng(int seed, int param, int rng_type = 0)`
- `double sprng()` - The next random number in \([0, 1)\) is returned
Simple Interface

- `int * init_sprng(int seed, int param, int rng_type = 0)`
- `double sprng()` - The next random number in $[0, 1)$ is returned
- `int isprng()` - The next random number in $[0, 2^{31})$ is returned
Random Number Parameter

Random Number Parameters

- Parameter is the number of predefined families defined
Random Number Parameter

Random Number Parameters

- Parameter is the number of predefined families defined
  - Modified Lagged Fibonacci Generator - 11
Random Number Parameter

Random Number Parameters

- Parameter is the number of predefined families defined
  - Modified Lagged Fibonacci Generator - 11
  - 48 Bit Linear Congruential Generator - 7
Random Number Parameters

- Parameter is the number of predefined families defined
  - Modified Lagged Fibonacci Generator - 11
  - 48 Bit Linear Congruential Generator - 7
  - 64 Bit Linear Congruential Generator - 3
Random Number Parameter

Random Number Parameters

- Parameter is the number of predefined families defined
  - Modified Lagged Fibonacci Generator - 11
  - 48 Bit Linear Congruential Generator - 7
  - 64 Bit Linear Congruential Generator - 3
  - Combined Multiple Recursive Generator - 3
Random Number Parameters

Parameter is the number of predefined families defined

- Modified Lagged Fibonacci Generator - 11
- 48 Bit Linear Congruential Generator - 7
- 64 Bit Linear Congruential Generator - 3
- Combined Multiple Recursive Generator - 3
- Multiplicative Lagged Fibonacci Generator - 11
Random Number Parameter

Random Number Parameters

- Parameter is the number of predefined families defined
  - Modified Lagged Fibonacci Generator - 11
  - 48 Bit Linear Congruential Generator - 7
  - 64 Bit Linear Congruential Generator - 3
  - Combined Multiple Recursive Generator - 3
  - Multiplicative Lagged Fibonacci Generator - 11
  - Prime Modulus Linear Congruential Generator - 1
Usage Example - Default Interface

Use Example - Default Interface

```c
#define PARAM SPRNG_LFG
int gtype = 1;
seed = make_sprng_seed();
Sprng *gen1;
gen1 = SelectType(gtype);
gen1->init_sprng(0,ngens,seed,PARAM);
int random_int = gen1->isprng();
double random_float = gen1->get_rn_flt_simple();
gen1->free_sprng();
```
Usage Example - Simple Interface

```c
#define PARAM SPRNG_LFG
int gtype = 1;
seed = make_sprng_seed();
gen = init_sprng(seed, PARAM, gtype);
int random_int = isprng();
double random_float = get_rn_flt_simple();
```
Other Parts - Examples Folder

- convert.cpp - Used to be an example of converting old code to new, but mostly empty
- pi-simple.cpp - Compute pi using Monte Carlo integration
- spawn.cpp - Small sample program to get you started
- Fortran versions as well
Other Parts - Examples Folder

- Examples Folder
  - convert.cpp - Used to be an example of converting old code to new, but mostly empty
  - pi-simple.cpp - Compute pi using Monte Carlo integration
  - spawn.cpp - Small sample program to get you started
Other Parts - Examples Folder

- convert.cpp - Used to be an example of converting old code to new, but mostly empty
- pi-simple.cpp - Compute pi using Monte Carlo integration
Other Parts - Examples Folder

- **convert.cpp**: Used to be an example of converting old code to new, but mostly empty
- **pi-simple.cpp**: Compute pi using Monte Carlo integration
- **spawn.cpp**: Small sample program to get you started

Fortran versions as well
Other Parts - Examples Folder

- **convert.cpp** - Used to be an example of converting old code to new, but mostly empty
- **pi-simple.cpp** - Compute pi using Monte Carlo integration
- **spawn.cpp** - Small sample program to get you started
- Fortran versions as well
Other Parts - Tests Folder

- Tests Folder
Other Parts - Tests Folder

- Tests Folder
  - Statistical Tests
Other Parts - Tests Folder

- Tests Folder
  - Statistical Tests
    - chisquare.cpp - Chi-Square and Kolmogorov-Smirnov Probability Functions
Other Parts - Tests Folder

- Tests Folder
  - Statistical Tests
    - chisquare.cpp - Chi-Square and Kolmogorov-Smirnov Probability Functions
    - collisions.cpp - Collision test
Other Parts - Tests Folder

Other Parts - Tests Folder
- Tests Folder
  - Statistical Tests
    - chisquare.cpp - Chi-Square and Kolmogorov-Smirnov Probability Functions
    - collisions.cpp - Collision test
    - coupon.cpp - Coupon test
Other Parts - Tests Folder

Tests Folder

Statistical Tests

- chisquare.cpp - Chi-Square and Kolmogorov-Smirnov Probability Functions
- collisions.cpp - Collision test
- coupon.cpp - Coupon test
- equidist.cpp - Equidistribution test
Other Parts - Tests Folder

- Tests Folder
  - Statistical Tests
    - chisquare.cpp - Chi-Square and Kolmogorov-Smirnov Probability Functions
    - collisions.cpp - Collision test
    - coupon.cpp - Coupon test
    - equidist.cpp - Equidistribution test
  - Other Tests
Other Parts - Tests Folder

- Tests Folder
  - Statistical Tests
    - `chisquare.cpp` - Chi-Square and Kolmogorov-Smirnov Probability Functions
    - `collisions.cpp` - Collision test
    - `coupon.cpp` - Coupon test
    - `equidist.cpp` - Equidistribution test
  - Other Tests
    - `fft.cpp` - FFT test
Other Parts - Tests Folder

- Tests Folder
  - Statistical Tests
    - chisquare.cpp - Chi-Square and Kolmogorov-Smirnov Probability Functions
    - collisions.cpp - Collision test
    - coupon.cpp - Coupon test
    - equidist.cpp - Equidistribution test
  - Other Tests
    - fft.cpp - FFT test
    - metropolis.cpp - Metropolis Algorithm
Other Parts - Tests Folder

- Tests Folder
  - Statistical Tests
    - chisquare.cpp - Chi-Square and Kolmogorov-Smirnov Probability Functions
    - collisions.cpp - Collision test
    - coupon.cpp - Coupon test
    - equidist.cpp - Equidistribution test
  - Other Tests
    - fft.cpp - FFT test
    - metropolis.cpp - Metropolis Algorithm
    - random_walk.cpp - Random Walk Algorithm
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer \$g++\$ versions

Support for cycle splitting

New generators for SPRNG
  1. Shift-register generators via splitting (MT/Well)
  2. New parameters for small-memory generators

Commercialization of SPRNG

New architectural support for (and maintaining reproducibility as an option)
  1. Multicore via OpenMP and eventually OpenCL
  2. GPGPU support via CUDA and eventually OpenCL

SPRNG Library
**SPRNG’s Future**

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer `g++` versions
- The class interface is not optimal for extension
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer g++ versions
- The class interface is not optimal for extension
- New architectural support for (and maintaining reproducibility as an option)

1. Multicore via Open/MP and eventually Open/CL
2. GPGPU support via CUDA and eventually Open/CL

- Support for cycle splitting

- New generators for SPRNG
  1. Shift-register generators via splitting (MT/Well)
  2. New parameters for small-memory generators

- Commercialization of SPRNG
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer g++ versions
- The class interface is not optimal for extension
- New architectural support for (and maintaining reproducibility as an option)
  1. Multicore via Open/MP and eventually Open/CL
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer g++ versions
- The class interface is not optimal for extension
- New architectural support for (and maintaining reproducibility as an option)
  1. Multicore via Open/MP and eventually Open/CL
  2. GPGPU support via CUDA and eventually Open/CL

Support for cycle splitting

New generators for SPRNG
  1. Shift-register generators via splitting (MT/Well)
  2. New parameters for small-memory generators

Commercialization of SPRNG
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer \$g++\$ versions
- The class interface is not optimal for extension
- New architectural support for (and maintaining reproducibility as an option)
  1. Multicore via \$Open/MP\$ and eventually \$Open/CL\$
  2. GPGPU support via \$CUDA\$ and eventually \$Open/CL\$
- Support for cycle splitting
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer g++ versions
- The class interface is not optimal for extension
- New architectural support for (and maintaining reproducibility as an option)
  1. Multicore via Open/MP and eventually Open/CL
  2. GPGPU support via CUDA and eventually Open/CL
- Support for cycle splitting
- New generators for SPRNG
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer g++ versions
- The class interface is not optimal for extension
- New architectural support for (and maintaining reproducibility as an option)
  1. Multicore via Open/MP and eventually Open/CL
  2. GPGPU support via CUDA and eventually Open/CL
- Support for cycle splitting
- New generators for SPRNG
  1. Shift-register generators via splitting (MT/Well)
**SPRNG’s Future**

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer g++ versions
- The class interface is not optimal for extension
- New architectural support for (and maintaining reproducibility as an option)
  1. Multicore via Open/MP and eventually Open/CL
  2. GPGPU support via CUDA and eventually Open/CL
- Support for cycle splitting
- New generators for SPRNG
  1. Shift-register generators via splitting (MT/Well)
  2. New parameters for small-memory generators
SPRNG’s Future

- A Visual Studio compile is under development (for Windows!?)
- There are several compiler warnings that need to be addressed with newer g++ versions
- The class interface is not optimal for extension
- New architectural support for (and maintaining reproducibility as an option)
  1. Multicore via Open/MP and eventually Open/CL
  2. GPGPU support via CUDA and eventually Open/CL
- Support for cycle splitting
- New generators for SPRNG
  1. Shift-register generators via splitting (MT/Well)
  2. New parameters for small-memory generators
- Commercialization of SPRNG
References

[M. Mascagni and H. Chi (2004)]
Parallel Linear Congruential Generators with Sophie-Germain Moduli,
*Parallel Computing*, **30**: 1217–1231.
References

[M. Mascagni and H. Chi (2004)]
Parallel Linear Congruential Generators with Sophie-Germain Moduli,
Parallel Computing, 30: 1217–1231.

[M. Mascagni and A. Srinivasan (2004)]
Parameterizing Parallel Multiplicative Lagged-Fibonacci Generators,
References


Questions?
© Michael Mascagni, 2010