Novel PRNG schemes (EC-based and/or parallel) and their automated testing

M. Gonçalves, S. Konchenko, L. Trestioreanu

Parallel and Grid Computing Projects

Master in Information and Computer Sciences (MICS),

University of Luxembourg (UL), Luxembourg

Lecturers: *Dr S. Varrette*, *V. Plugaru* and *Prof. P. Bouvry*

Problem scope

What is random number

Flip coin sequences of Head and Tails P('heads')=1/2:

{ H T H T H T H T H T H T H T H T H T H T }

{ H T H **T T T** H T **H H H H H** T H **T T T** H T }

Which sequence is random?

- subjective randomness

Random Number Generator

Two categories of RNG:

- **deterministic approaches (Pseudo-Random Number Generator PRNG)**

  a formula where the input completely determines the output

- **non-deterministic approaches (Hardware-based Random Number Generator HRNG)**

  flipping coins, rolling dice, keyboard latency, white noise, level of radioactivity and motion of lava lamps

Is it enough to use PRNG in most application?

Pseudo-Random Number Generator

Characteristics of PRNG

- **Efficient:**
  many numbers in a short time

- **Deterministic:**
  sequence can be reproduced if the starting point is known.

- **Periodic:**
  sequence will eventually repeat itself

What about testing PRNG?

Testing of PRNG

Two distinct group:

- Theoretical Tests:

  require knowledge of the PRNG structure

- Empirical:

  use generated sequence

Which PRNG testing suites are known?

Testing suites of PRNG

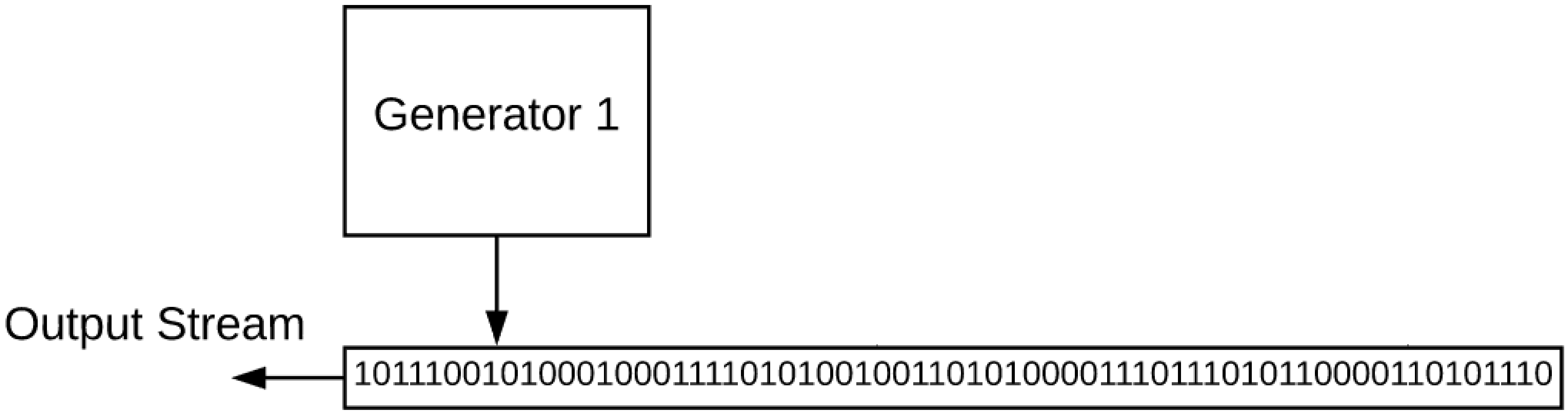
- Knuth’s tests (1969)
- Monte Carlo simulation of the 2D Ising model (1986)
- Diehard suite (1995)
- NIST Statistical Test Suite (2001)
- Dieharder suite (2003)
- TestU01 (2007)

Why we need parallel PRNG?

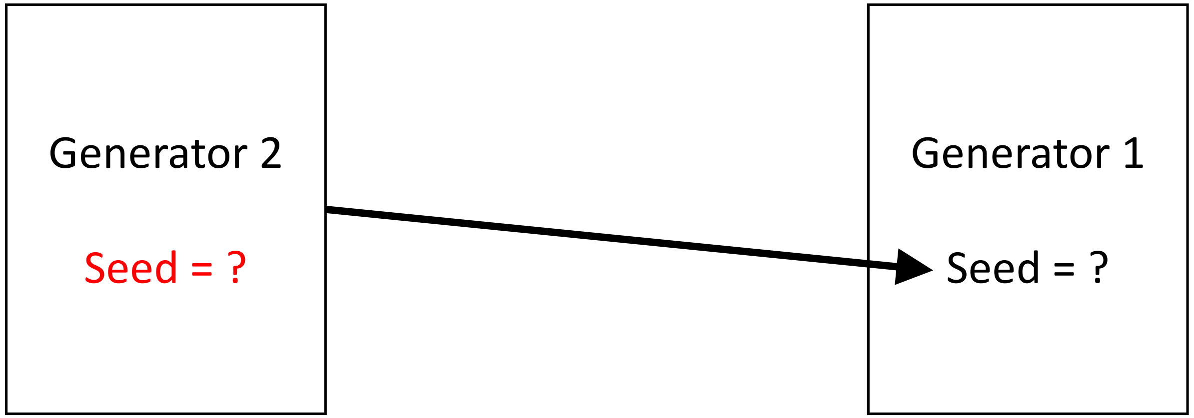
Sequential…

…to parallel execution!

(OpenMPI)



Seed choice - Random



Only moving the problem

Seed choice - Incremental

Generator 1	Generator 2	Generator 3
Seed = 1000	Seed = 1001	Seed = 1002

Output: 97 94 95

Similar outputs…

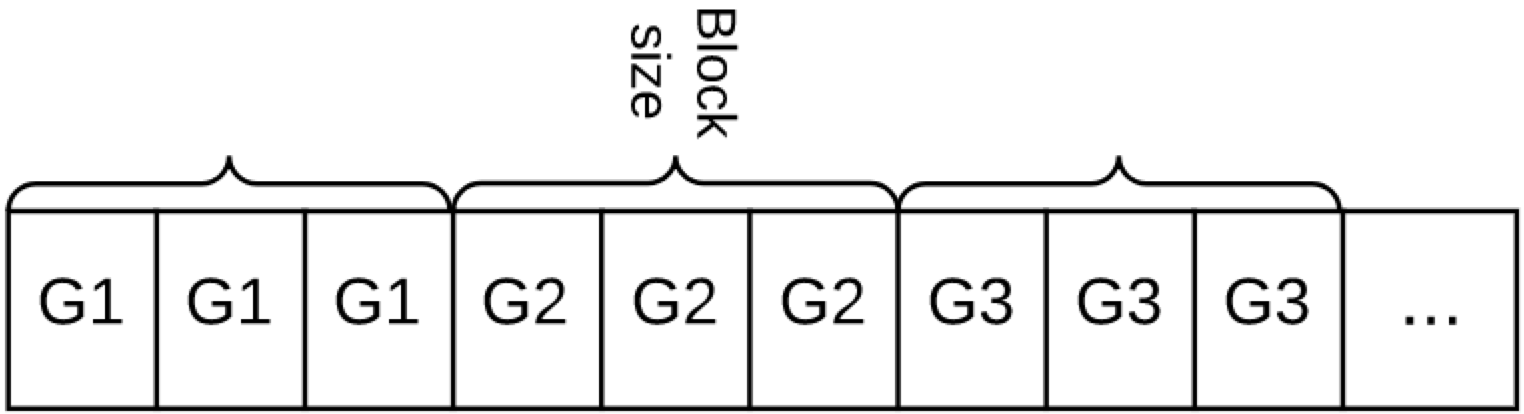
Seed choice - Multiplicative

Generator 1	Generator 2	Generator 3
Seed = 1000	Seed = 2000	Seed = 3000

Output:	5	42	11

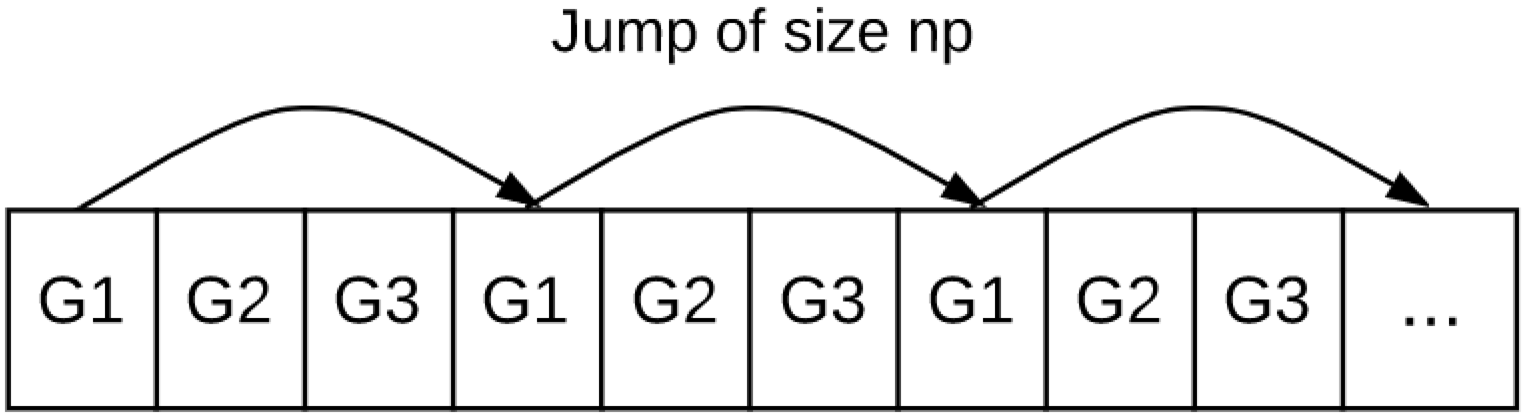
✓ Works!

Fixed seed with Block-splitting



UNIVERSITÉ DU LUXEMBOURG

Fixed seed with Leap-frogging



Skipping ahead?

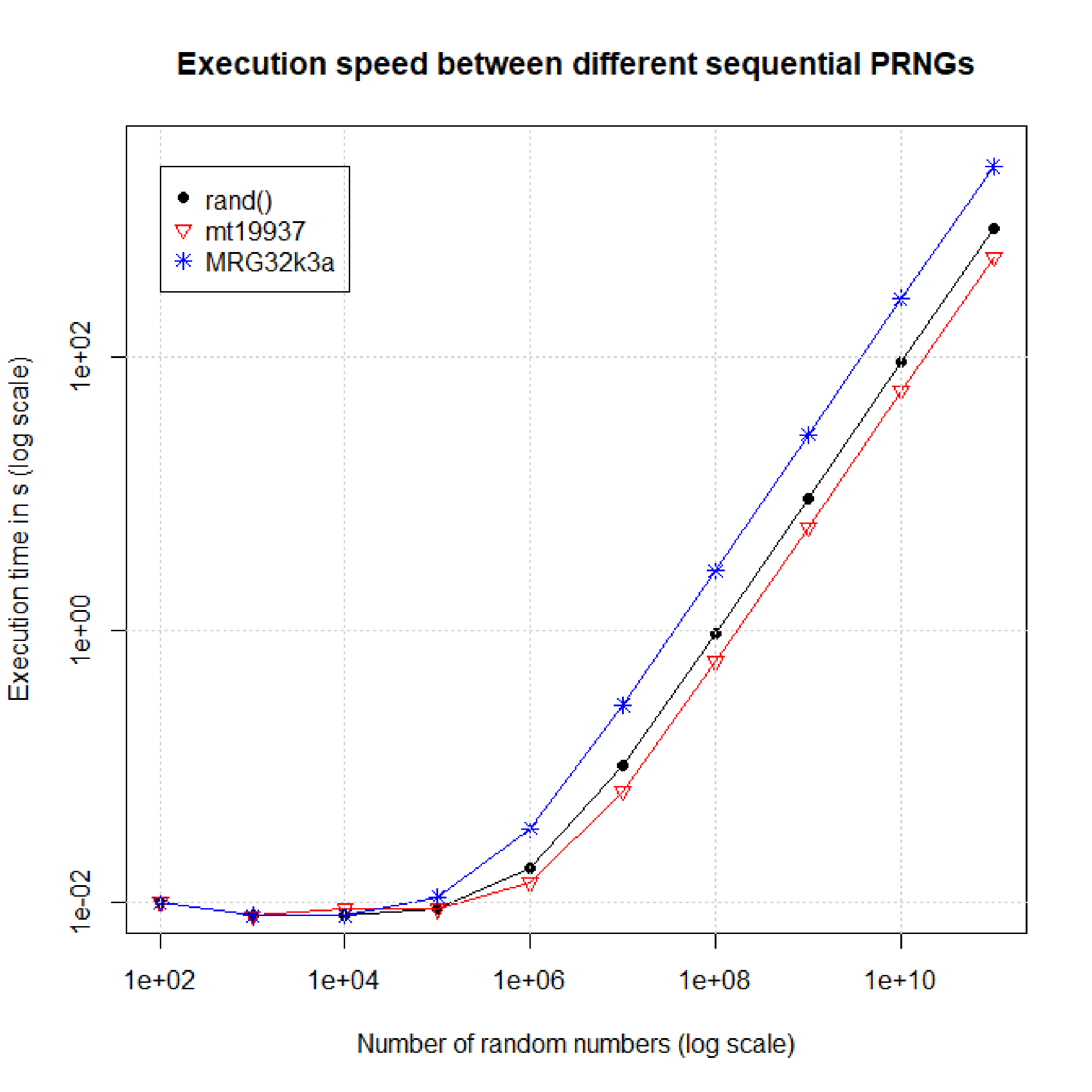
Seed →

p1 = 51	102	12
p2 = 17	53	222
p3 = 180	1	3
…		
pn = 2	95	76

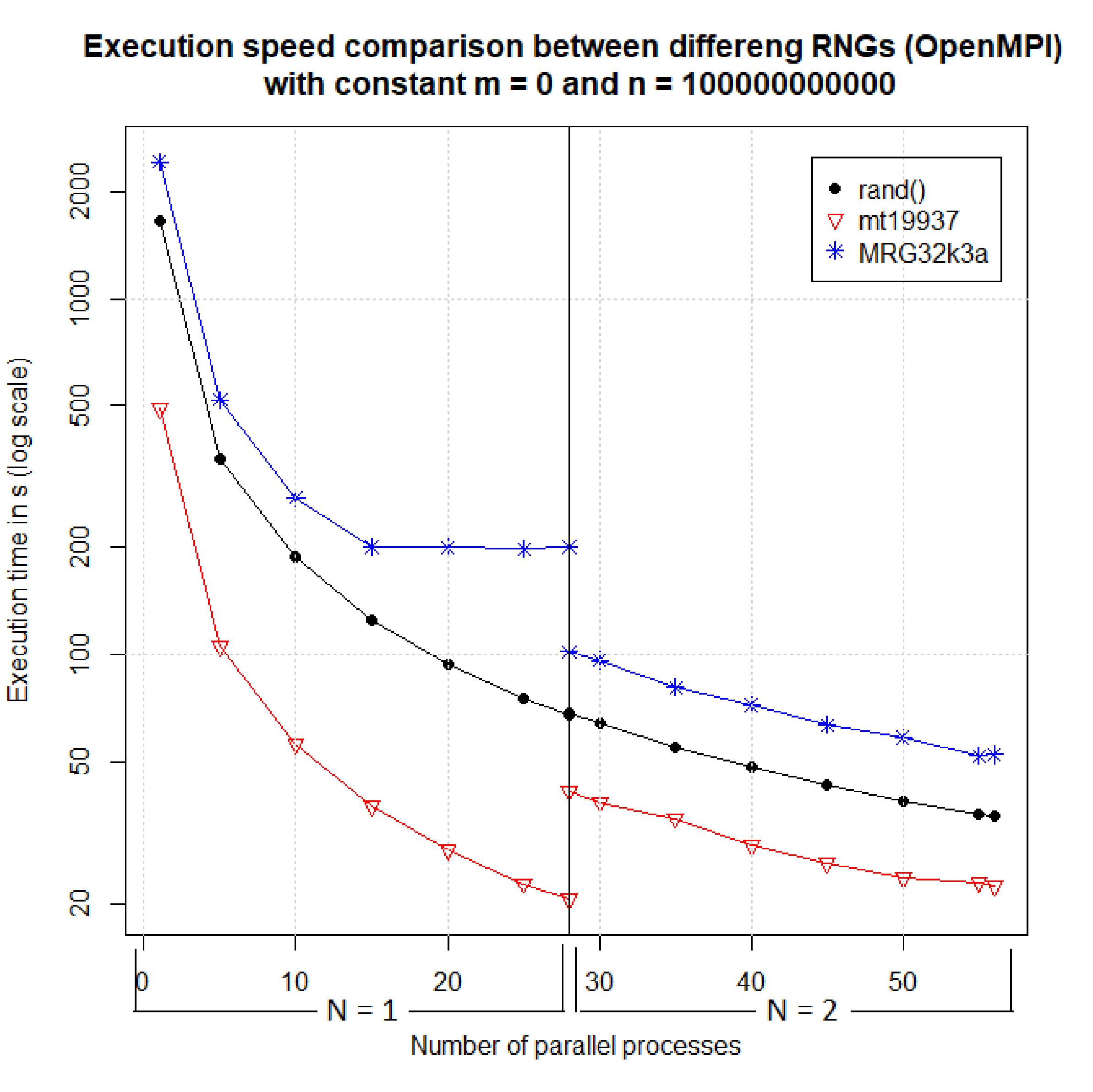
Output: 42 , 37

No direct formula for parameter prediction

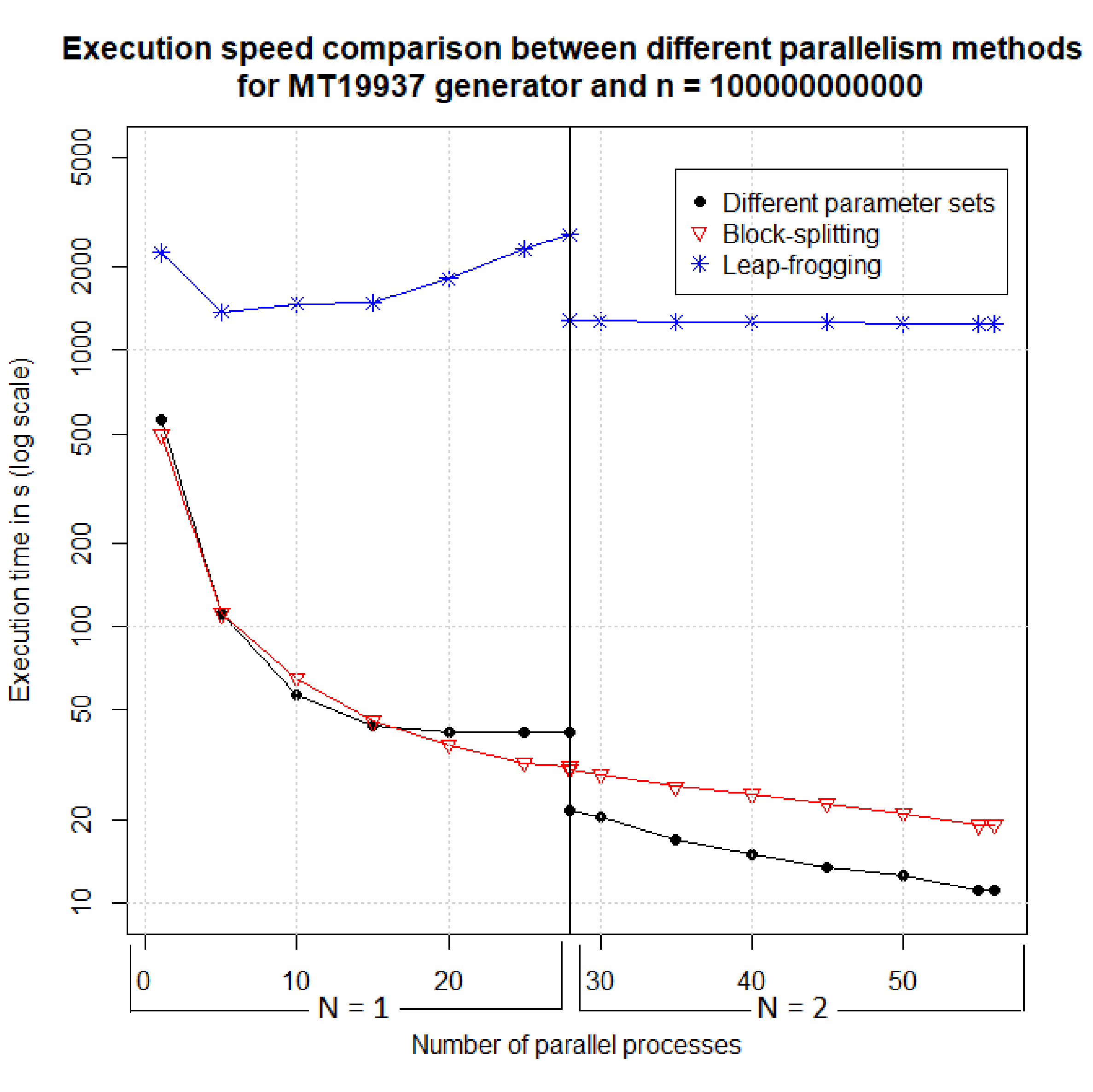
Experimental Results – Speed Benchmark



Experimental Results – Speed Benchmark



Experimental Results – Speed Benchmark



Dieharder Tests & NIST Test Suite

std::rand

MT19937

MRG32k3a

Dieharder Tests & NIST Test Suite

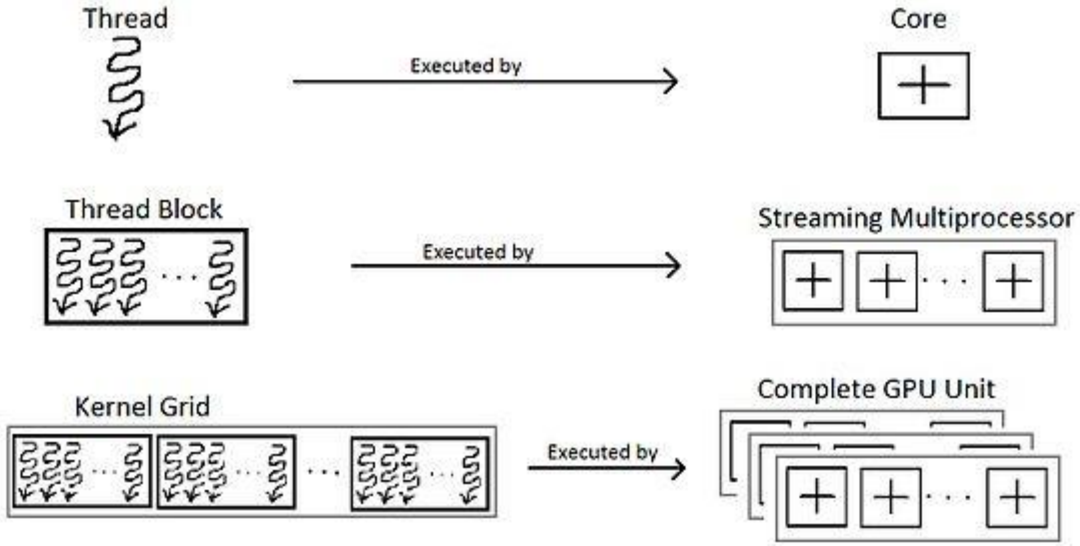
Seed choice (Multiplication)

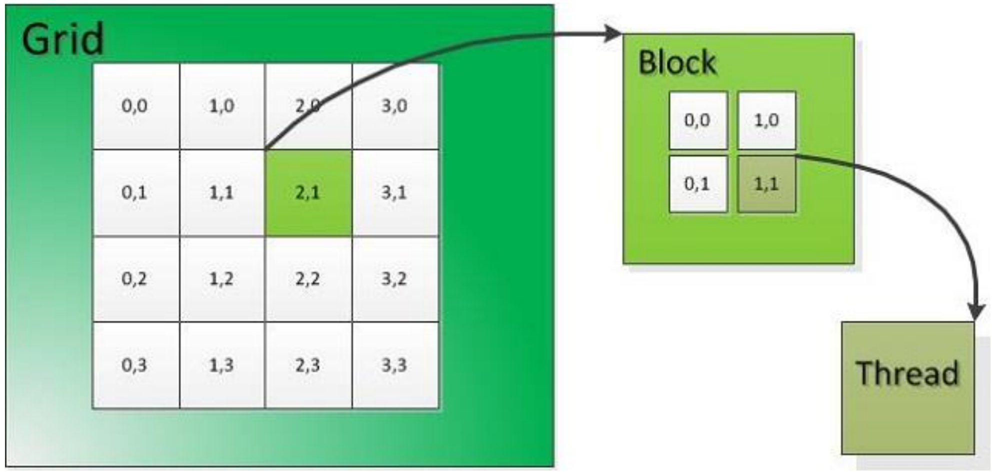
Block-splitting

Leap-frogging

GPU accelerated PRNGs

<u>Programmer vs hardware perspective [1]</u>

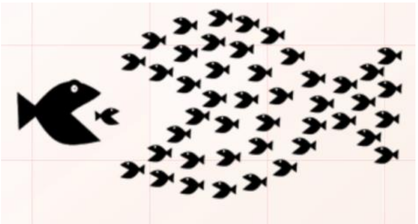




<u>Power of the crowd</u>

sheer computing power built for repetitive tasks

many times more cores/threads than CPUs, can run in parallel



GPU accelerated PRNGs

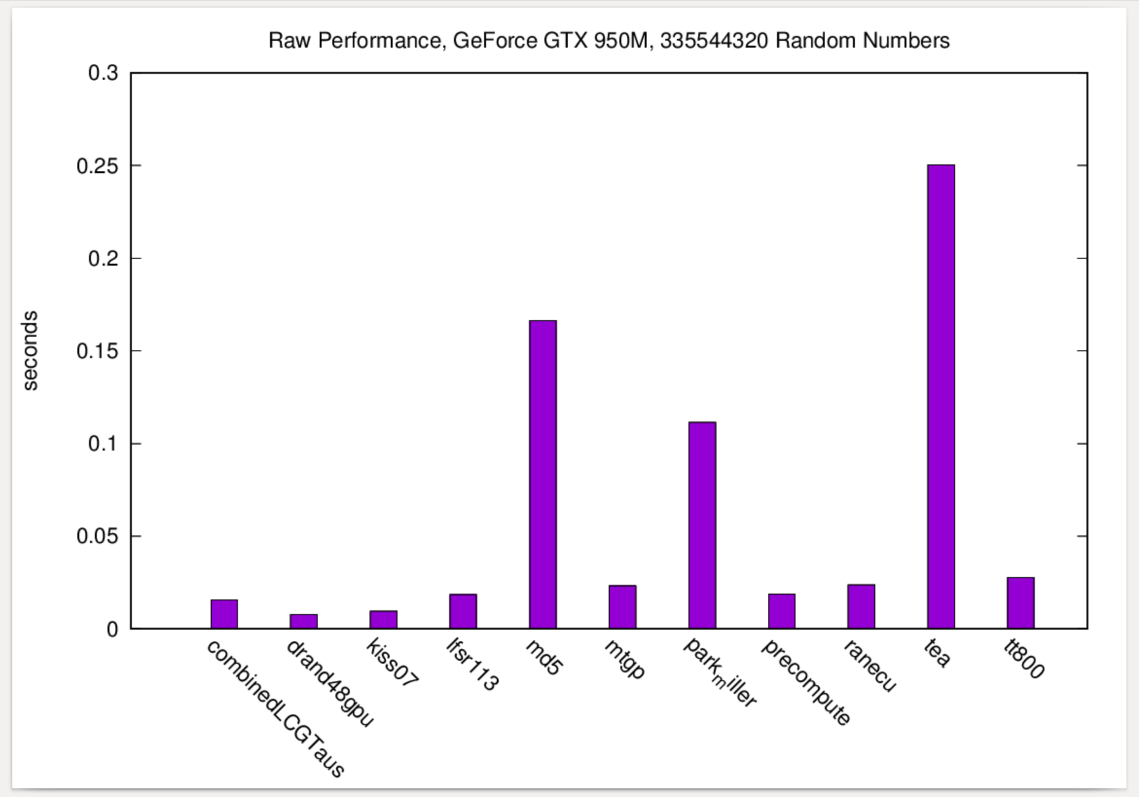
Raw performance

10 program calls, 10 kernel calls, 16384 threads, 2048 rand numbers computed in each thread in each kernel call

335544320 (amount of random numbers per launch) = kernel_calls * num_threads * num_randoms_per_thread

GPU

- multiprocessor count: 5
- stream processor count: 128 (total 640)
- warp size: 32
- max threads per block: 1024
- max block dimensions: 1024 x 1024 x 64
- max grid dimensions: 2147483647 x 65535 x 65535

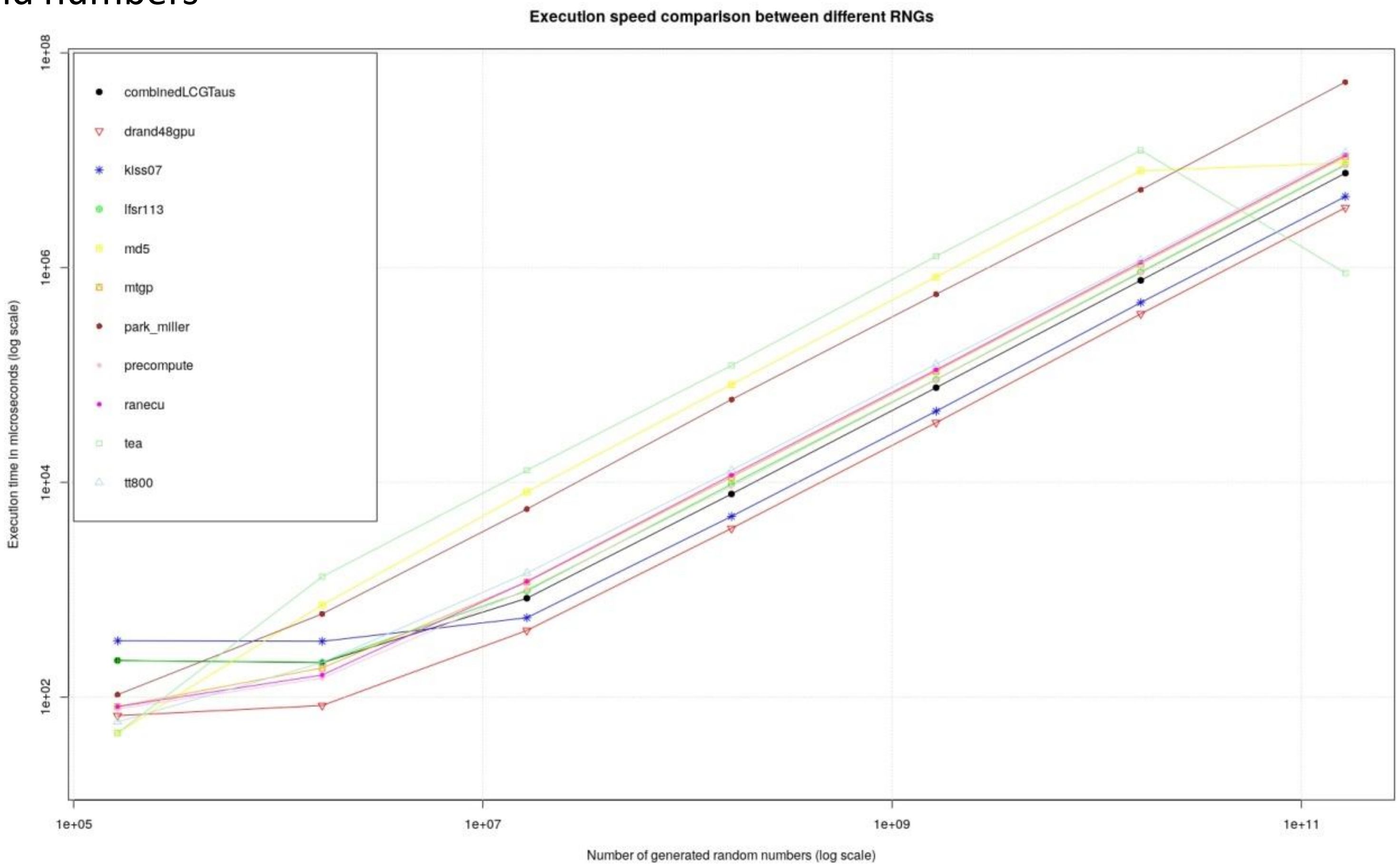


GPU accelerated PRNGs

Execution time versus the quantity of random numbers generated, for all studied RNGs.

163.840 to 163.840.000.000 rand numbers

1. drand48gpu
2. kiss07
3. combinedLCGTaus
4. precompute
5. lfsr113
6. mtgp (Mersenne twister)
7. ranecu
8. tt800
9. park_miller
10. md5
11. tea



GPU accelerated PRNGs

Overview of the Dieharder and Raw Performance tests

RNG	Test results [passed/failed/weak/total]			
	Passed	**Failed**	**Weak**	**Speed**
lfsr113	112	0	0	reference
mtgp	111	0	1	reference
kiss07	111	0	1	faster
md5	110	0	2	slower
combinedLCGTaus	110	0	2	faster
tea	109	0	3	slower
tt800	108	0	4	reference
ranecu	56	**50**	6	reference
drand48gpu	54	**49**	9	fastest
park_miller	26	**85**	1	slower
precompute	0	**112**	0	reference

Not necessarily the slower RNGs have the best quality and vice-versa

Conclusion - CPU

All three methods used yielded good results with “leap-frogging” performing slightly worse in terms of speed and scalability.

Using the same seeds for all generators will produce the same outputs. Thus, not using either block-splitting or leap-frogging will fail the statistical tests.

std::rand generator should not be used.

Conclusion - GPU

Best performers:

•“lfsr113“ – passed all tests

• "mtgp“

•"kiss07“ – second fastest

Should be avoided (systematically failed tests)

•"Ranecu“

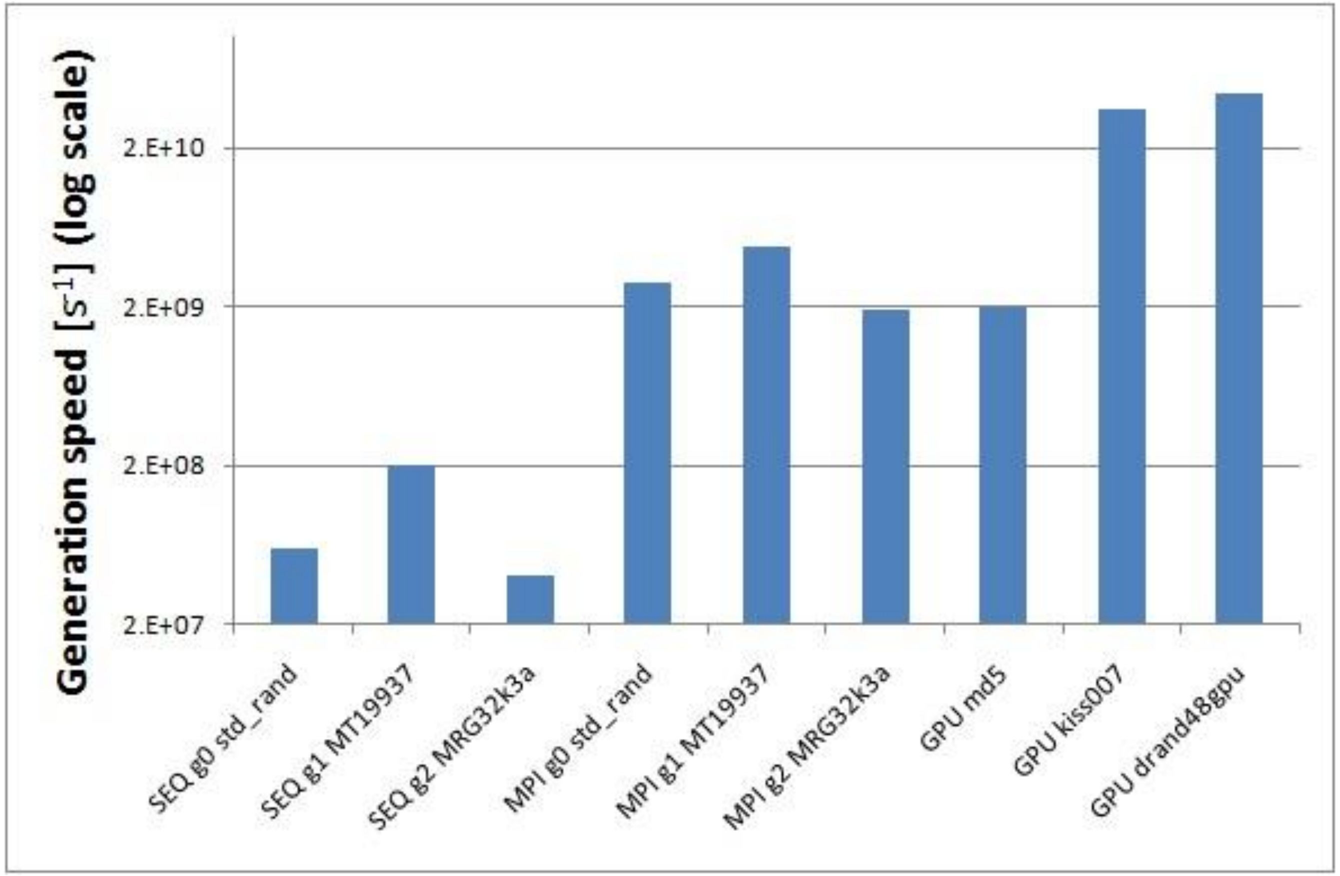
•"drand48gpu“

•"park_miller“

•"precompute”

Conclusion - Speeds

GPU execution was faster than parallel CPU, which was faster than sequential CPU execution.



Questions?