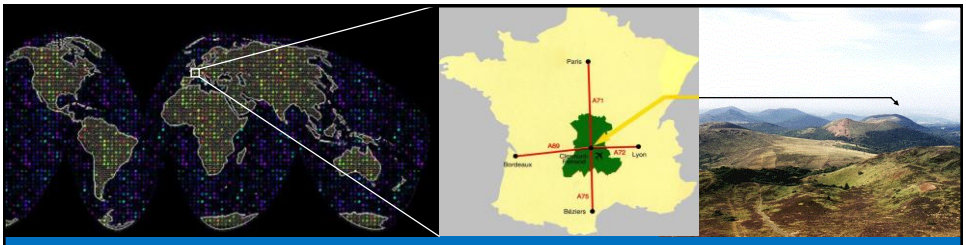


Ingénierie des Modèles et Simulation
(partie – SIMULATION – Rappels et Parallélisme)

David Hill – ISIMA

Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes

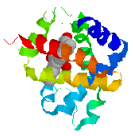
UMR CNRS 6158

Blaise Pascal University

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

ISIMA

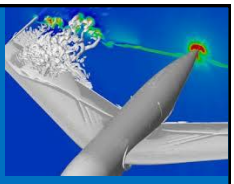
Expérience en simulation distribuée pour les Sciences de la Vie, la Santé, l'Environnement, la Biologie cellulaires et moléculaire



ISIMA - F2 (inside)

3

Part I.A
Reminder : what is Simulation...

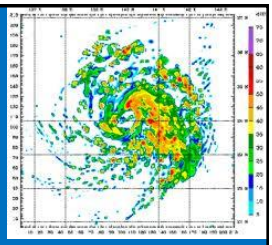


Aeronautic air tremble
© 2006 ONERA

- In Computer Science: Simulation is the imitation of the operation of a system or a real world process over time.
- The system is a collection of interacting objects (cf. dictionary definition)
- The system can be existing or not :
  - « A priori » modelling (non existing)
  - « A posteriori » modelling (existing)

4

And what about Models ?



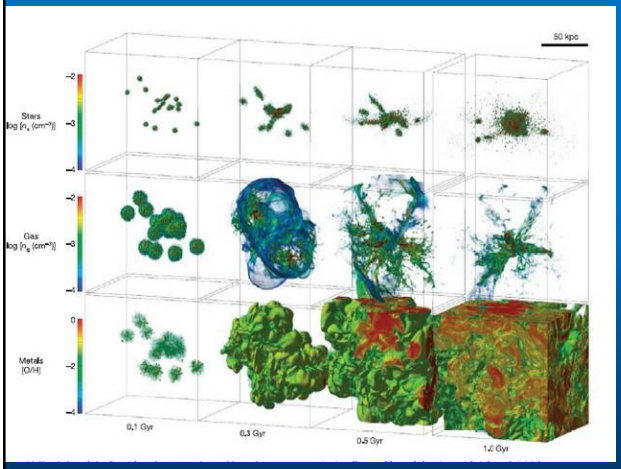
- A **model** is a representation of a **system**
- In Matter, Mind and Models (published by MIT Press in 1965) by Marvin L. Minsky we find the following definition :

> To an observer B,
> an object A* is a model of an object A
> to the extent that B can use A*
> to answer questions that interest him about A

- It implies that :
  - A model is built with an intended **goal** in mind.
  - A the model should be complex enough to answer the **questions** raised.

5

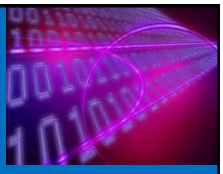
Ex: Simulations of galaxies



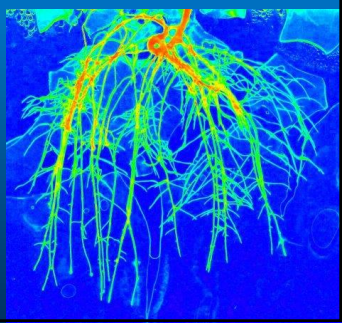
Sept. 27th 2012,
Two astronomers have made one of the largest ever conducted simulations in astrophysics to model the growth of galaxies.
Masao Mori (LA Univ.) and Masayuki Umemura (Tsukuba Univ.) are able to simulate galaxy evolution since 300 million years after the Big Bang until today.
Their results show that galaxies may have evolved much faster than previously thought.

6

Simulations
Deterministic vs. Stochastic



- **Deterministic** simulation : the output of the model is precisely determined by its structure and its parameter values
- **Stochastic** modeling : use of a random source
  - New scientific barriers to break are sometimes behind our best deterministic models
  - The assessment of stochastic parallel simulations is tough and this domain is less studied



Partie I.B
Rappels sur les Simulations stochastiques…

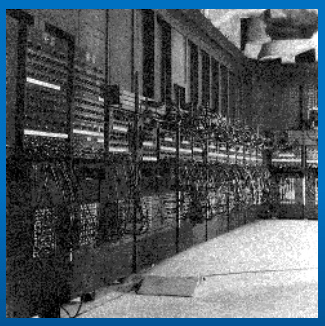
- Un seul appel à une source de hasard rend votre simulation stochastique
- Plus de 50% des temps de calcul sur « supercalculateurs » est utilisé pour des modèles stochastiques

- Des sources logicielles
  - ✓Pseudo-aléatoires
  - ✓Quasi-aléatoires
- Des sources matérielles: (vrai hasard - TRNG ?)

8

Simulation dites ‘Aléatoires’

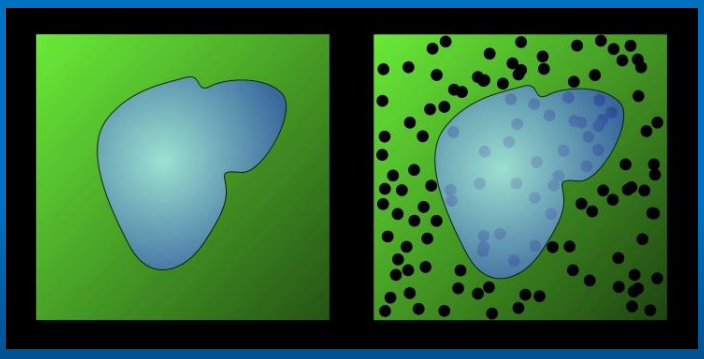
Nom de code WWII – Monte Carlo



Une méthode de calcul pour les intégrales multiples et complexes en dimensions élevées – utilisée dans de nombreux domaines dont la physique nucléaire

9

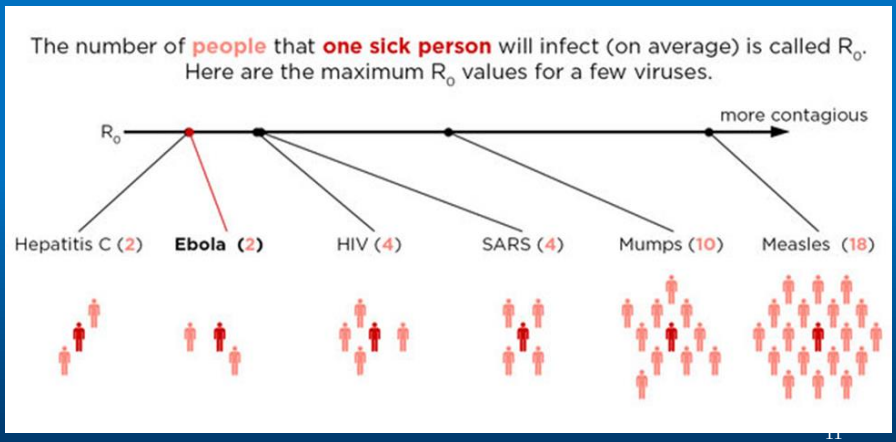
Vous jouez aux fléchettes ?



Estimation de surface grâce au tirage de points aléatoires : fonctionne avec bien plus de 2 dimensions.

10

Applications parfois d'actualité…

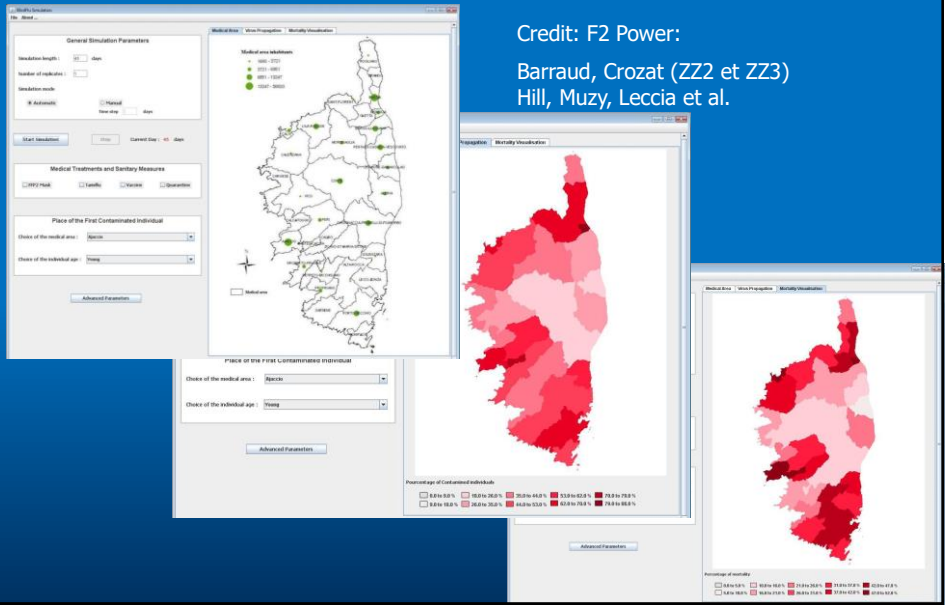


11

ISIMA

Egalement des applications « sociétales »

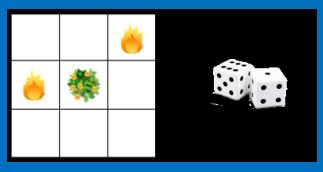
JBirdFlu Simulation

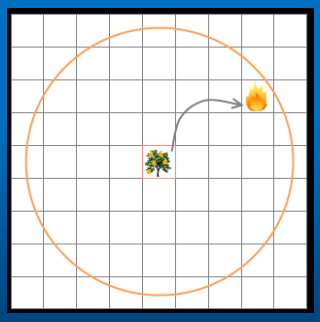


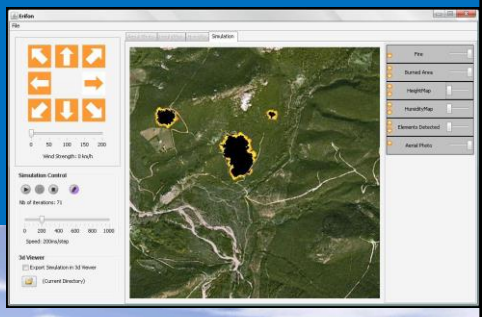
Credit: F2 Power:

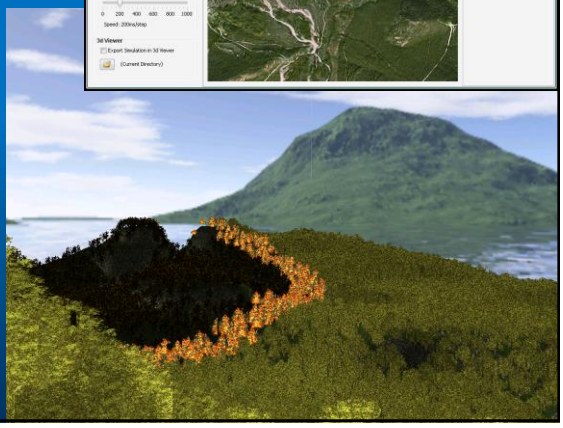
Barraud, Crozat (ZZ2 et ZZ3)
Hill, Muzy, Leccia et al.

Un autre exemple : environnemental *Fire Simulation*

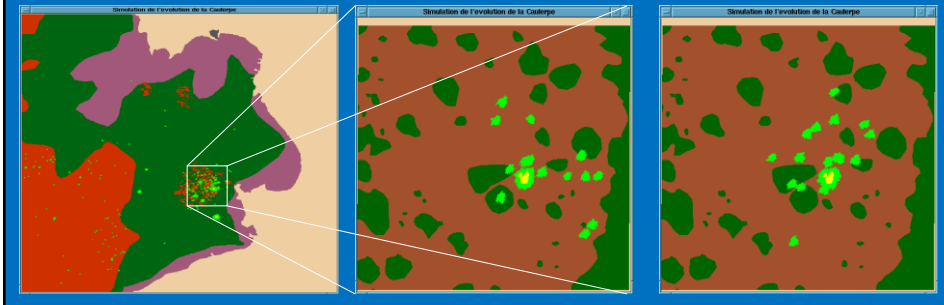


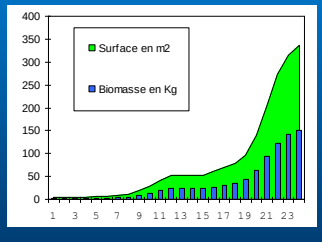


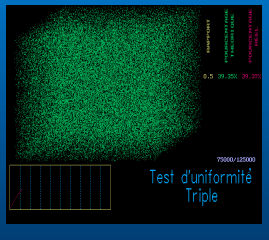


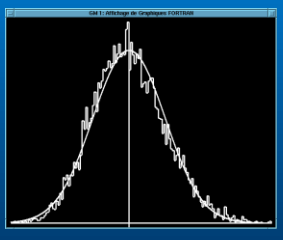


Rappel : Effectuer des réplications…

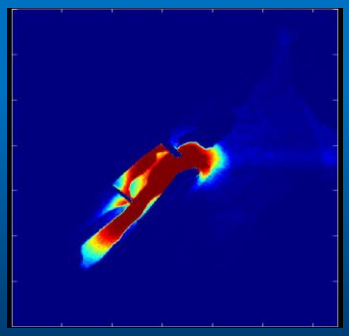






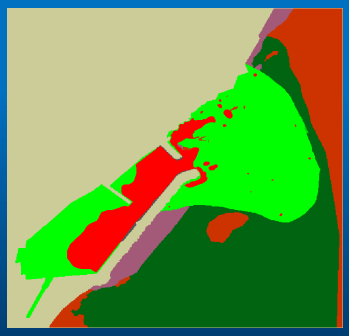


Rappel: Analyse spatiale et spectrale de résultats de simulations stochastiques (en 2 et 3 dimensions)



< Résultat d 'analyse spectrale

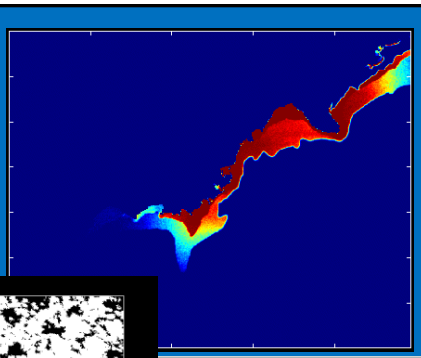
Etude > comparative

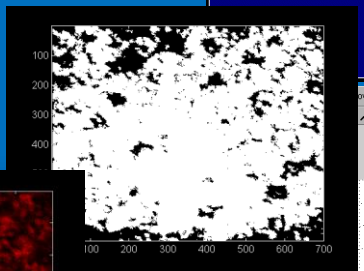


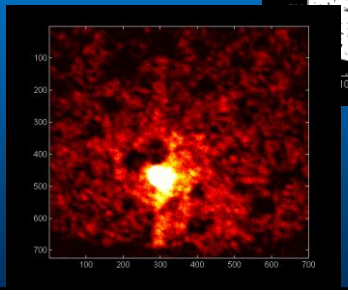
15

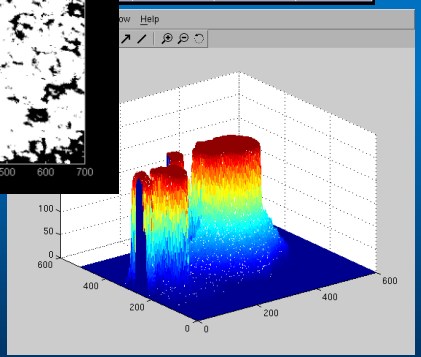
Etude des résultats par analyse spectrale

Parmi les méthodes de validation de modèles, l'analyse spectrale présente un intérêt non négligeable lorsque les résultats des simulations stochastiques montrent de fortes corrélations spatiales









Les limites de cette technique

- Les temps de calculs peuvent être lourds pour produire des cartes intéressantes sur des sites de grande taille => paralléliser, accélérer les techniques de génération de nombres pseudo-aléatoires
- L 'étude des résultats d'analyse spectrale suppose qu'il y ait eu enregistrement des répartitions spatiales pour chaque réplication.
- Confrontation aux résultats numériques correspondants, variance et homogénéité, par exemple, afin de détecter éventuellement plusieurs attracteurs spatiaux dont la « moyenne » ne serait guère informative.
- Etude approfondie pour déterminer les effets possibles d'éventuels attracteurs qui pourraient être exclusifs d'une réplication à l'autre.
- La mise en animation des images successives de l'écosystème peut nécessiter un temps de développement lourd mais il est alors possible de mettre à jour des phases transitoires fugaces et difficilement décelables par d'autres moyens.

17

Rappel:
Les méthodes et les simulations de Monte Carlo...

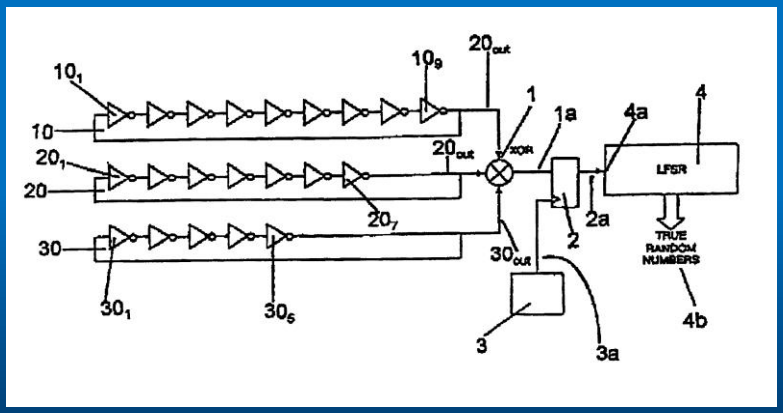
- Font des expériences d'échantillonage statistique en se servant de manière habile du hasard : "stokos" (stochastique)
- Basée sur un échantillon aléatoire
- Vitesse de convergence lente,
  - Approximativement $O(N^{-1/2})$

Accélération des méthodes de Monte Carlo

- Technique de réduction de Variance
- Méthodes "Quasi-Monte Carlo"
- Calcul parallèle : Cours de HPC // Master R

18

Partie II
Quelques rappel sur la génération de nombres « aléatoires… »



TRNG : sources de “vrai” hasard ? Encore de nombreuses questions ?

- Le coût de nombreux dispositifs sur des architectures parallèles massives
- Comment appréhender les pannes matérielles
- Les pannes réseau si il s’agit d’un serveur en ligne
- Comment vérifier et corriger les biais (correction intégrée &
- problème de la qualité des tests (problèmes récents détectés)
- Pour reproduire les calculs : faut-il archiver ?
  - ✓ se limiter à des petites séquences (pas adapté au calcul intensif) and
  - ✓ se limiter à ce qui peut être utilisé avec du “Memory mapping” ?

Au final, on note des problèmes de (1) fiabilité et (2) de répétabilité pour des applications scientifiques et pour le débogage :

- ✓ Reproductibilité des séquences ?
- ✓ Reproductibilité des expériences (méthode scientifique) ?

20

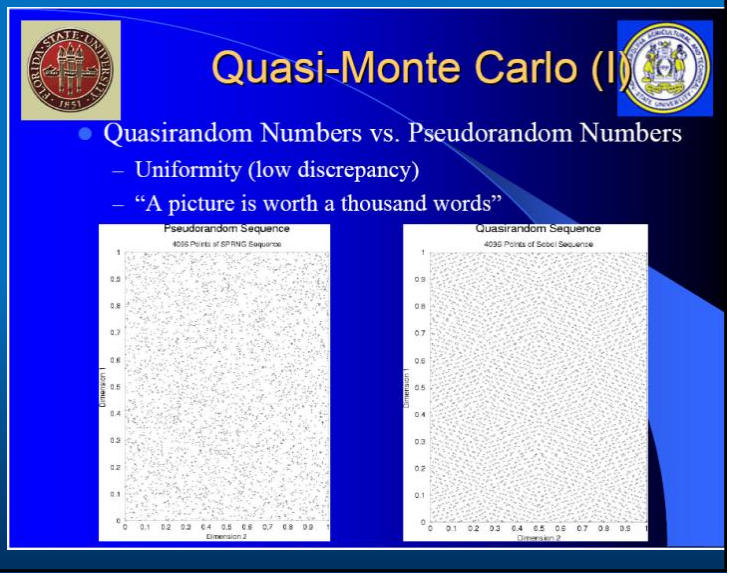
Une autre option : les nombres quasi-aléatoires

Prof. Mascagni & Dr. Li

Methodes

- Van der Corput
- Halton
- Faure
- Sobol
- Niederreiter

Convergence *proche de* $O(N^{-1})$
Pas de réplications
Mais cadre applicatif très limité



Au final : utiliser des générateurs pseudo-aléatoires

Rappel des principaux types de générateurs

Fuir ceux en rouge pour des application scientifiques…

- **LCG (Linear Congruential Generator)**
  $x_i = (a*x_{i-1} + c) \bmod m$
- **LCGPM (Linear Congruential Generator with Prime Modulus)**
- **MRG (Multiple Recursive Generator)**
  $x_i = (a_1*x_{i-1} + a_2*x_{i-2} + \ldots + a_k*x_{i-k} + c) \bmod m$ avec $k > 1$
- **LFG (Lagged Fibonacci Generator)**
  $x_i = x_{i-p} \otimes x_{i-q}$
- **SRG (Shift Register Generator = LFG avec operator XOR)**
- **Générateurs brassés qui combinent les autres.**

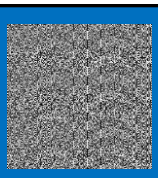
22

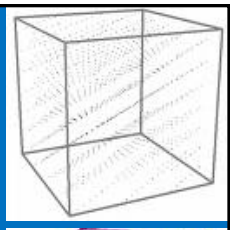
Les pré-requis pour un générateur séquentiel

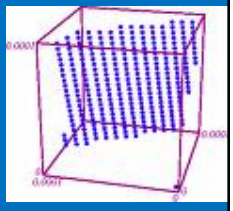
- *prop. 1 :* les nombres sont générés uniformément
- *prop. 2 :* la séquence n'est pas corrélée
- *prop. 3 :* la séquence est reproductible
- *prop. 4 :* le générateur est portable
- *prop. 5 :* la séquence peut être changée (germe, table,…)
- *prop. 6 :* la période est la plus grande possible
- *prop. 7 :* le générateur passe les tests statistiques courants
- *prop. 8 :* les nombres sont obtenus rapidement
- *prop. 9 :* le générateur utilise peu de mémoire

23

Batteries of tests for "random" numbers







- Some tests initially proposed by Knuth
- DieHard (G. Marsaglia) – 1990s
- NIST – STS (mainly for cryptography…)
- Tests form L'Ecuyer, Matsumoto, Kurita…
- DieHarder… (R.G. Brown)
- The most complete test battery is currently: TestU01
  - L'Ecuyer et al. 2002-2006
  - Small crush, crush and big crush with more than a hundred statistical tests

24

Partie II.1
«Paralléliser » les nombres aléatoires…

Parallelism in Stochastic & Monte Carlo (MC) Applications

MC simulations

- are computationally intensive
- most of the time applications appear to be **naturally parallel - sometimes said "embarrassingly parallel"**
- Appropriate for the independent *bag-of-work* paradigm
- Perfect fit with the distributed computing paradigm

But remember that we want rigorous simulations !

Main requirement for parallel stochastic simulations :

Independence of the parallel random number streams used by the different logical processors (CPUs or cores…)

26

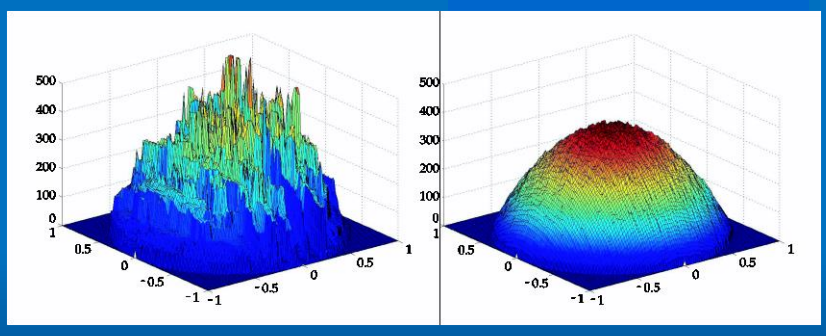
« Random number generators, particularly for parallel computers, should not be trusted. »

By Paul Coddington

In [Traore & Hill 2001]

- **It is necessary that RNs can be generated in parallel (ie each process may have an autonomous access to a sub-sequence issued from a common global sequence**
- **If such an autonomy is not guaranteed, the potential parallelism of the application is affected (if for instance processes access to a central RNG, even if this generator is also run in //)**
- **In addition, as stated by Paul Coddington: *"It is strongly recommended that all simulations be done with two or more different generators, and the result compared to check whether the random number generator is introducing a bias "***
- **The main problem is to find partitioning techniques which preserves the good properties required to guarantee not only the efficiency of the simulation but mainly the credibility of the results.**

Let's deal with the real impact of the generator quality…

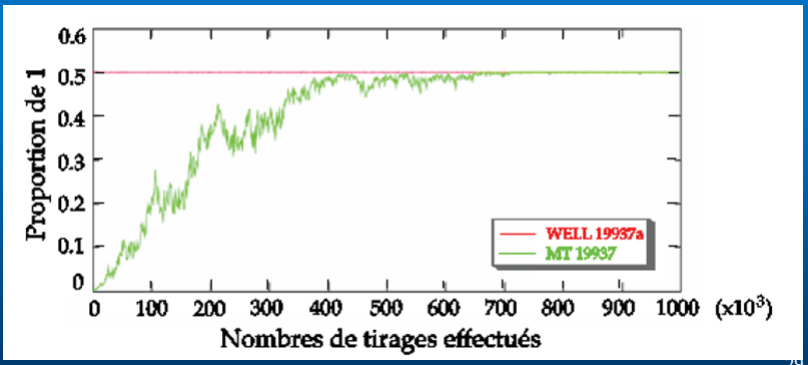


Here we see two results of the same PDE simulation (sequential).
On the left the image is obtained with Linux default rand in 2007 (which is already far better than the old std UNIX rand on 15bits)
On the right – same simulation with Mastumoto Mersenne Twister

28

There is no perfect Generator (1/2)
First Mersenne Twister (1997): a known default now corrected…

Between 1997 and 2002: very long recovery of **zero-excess initial state** for MT19237 (700 000 drawings…)

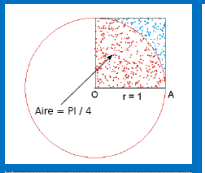


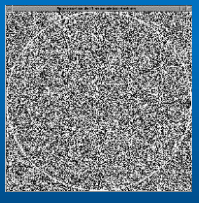
29

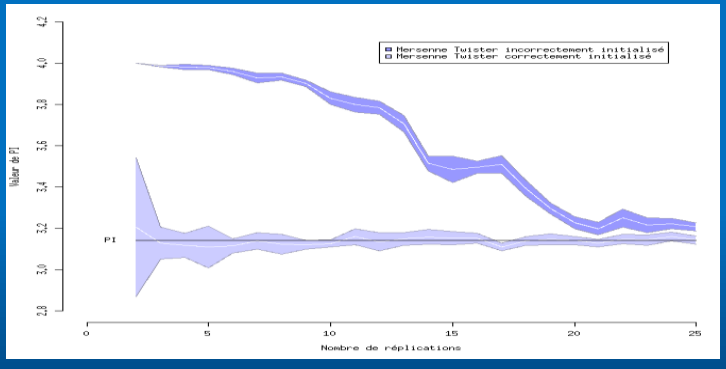
There is no perfect Generator… (1/2)

"We are here dealing with mere cooking recipes for making digits"

(1951 Von Neuman)







If it is not correctly initialized, even a very good generator can lead to biased results. Here, we obtain 3.25 for π after 25 replications each drawing 50 000 points (100000 pseudo-random numbers). Darker blue : 3.14 is not even in the confidence interval after the 25 replicates… Lighter blue : a correct simulation.

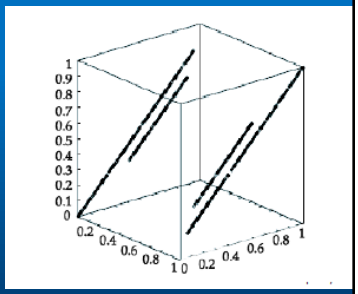
Even some not so old and well distributed software, were showing strong weaknesses, but were used for distributed simulations…

- **NS 2 – Famous Network Simulation Software**
  - **Over 50% of ACM and IEEE network simulation papers from 2000-2004 cite the use of *ns-2***
  - **8000 download / month**
  - **Was based on a 1969 PRNG with 2^{31} period and used for // simulation.**
- **CLHEP – Particle Physics Library still has many fairly obsolete generators (used in GEANT 4 for nuclear & particle physics)**
- **GATE : nuclear medicine software, based by default on James random PRNG (1997) – using CLHEP and based on GEANT 4**
  - **They recently changed their default generator to Mersenne Twister (2009)**

"not so old" simulation software still widely used…

- **Even after improvements a famous Open Source Network simulator – developed and financed by DARPA (Defense Advanced Research Project Agency) was still showing great weaknesses**
- **It proposed an integer seed for its parallel PRNG initialization**

BUT: simple tests with 1,2 & 3 were showing strong linear correlations.



(Shown by Entacher et al., 2002)

32

NS 2 again with a "very good – generator" from L'Ecuyer…

But a bad seeding API…

- Due to bad documentation and re-use of old scripts many people still use the old API functions to explicitly set seeds.
- Unfortunately, this corrupts the correct functioning of the new generator (MRG32k3a) and can lead to correlated simulation results.
- This might affect the ns-2 simulation results
  - how many currently published results ?

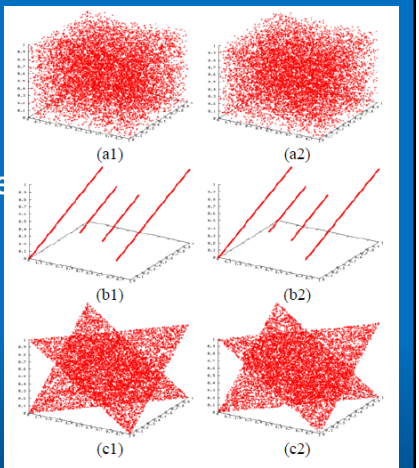


Figure 1. Correlation between three random variables for 10,000 values drawn. Left: MRG32k3a, right: old Park/Miller RNG. (a1) new (correct) seeding method, (a2) Seedset 1 (known "good" seeds), (b1/2) Seedset 2, (c1/2) Seedset 4.

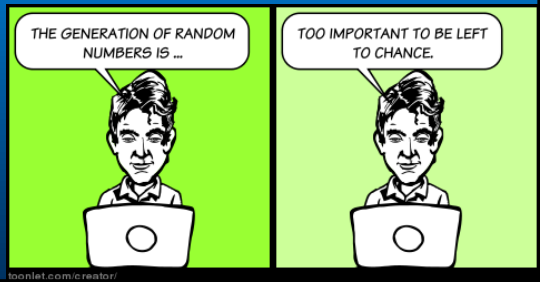
By Martina Umlauft & Peter Reichl

"Don't Trust Parallel Monte Carlo…"

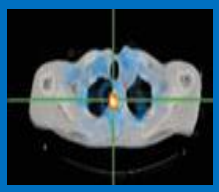
(Hellekalek PADS 1998)

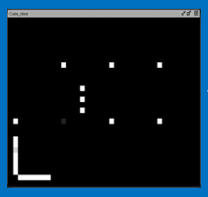
- There are many examples were bad parallelization of random streams lead to erroneous results
- In 2003 Entacher and Hechenleitner, showed the kind of pitfalls obtained when using parallel streams in OMNET++ simulations

But let's also talk about OUR errors…

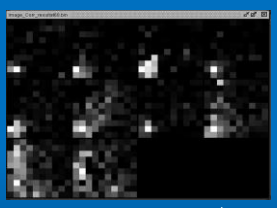


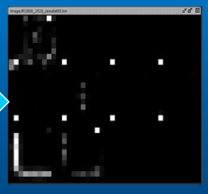
Impact of bad parallelization of random streams on *small tumor detection* (2004-2006)

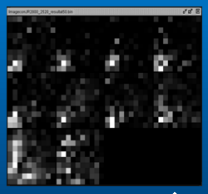




Reference Targeted Image







Correct simulated reconstruction

Image reconstructed with correlations between parallel random streams

Image we obtained > involuntary correlations and overlapping between parallel random streams

Manual scripting

35

So… what are the requirements for Parallel RNGs ?

- They have to be easy to split (partitioning the numbers into many independent sub-sequences that are allocated to different logical Processing Elements or Cores, without the need of communication or synchronization;
- Each sub-sequence has to be a good sequential RNG; it must possess good "random" qualities according to the current most stringent tests (TestU01)
- There should be no correlation between the sub-sequences on different PEs (crossed-correlation)
- No "mathematical proof" can really help…you have to run tests or to use "tested sequences".

36

Before going parallel : Optimizing for speed... What about Meta programming ?

37

Ex de Défi IDM : Optimisation et méta-programmation

Le compilateur est « embauché » pour faire des calculs ☺

```
template <int N, int P>
inline long puissance()
{
      return puissance<P==1 ? 0 : N, P-1>() * N;
}
inline long puissance<0,0>() { return 1; }

void main()
{
      const long k = puissance<2,10>();
}
```

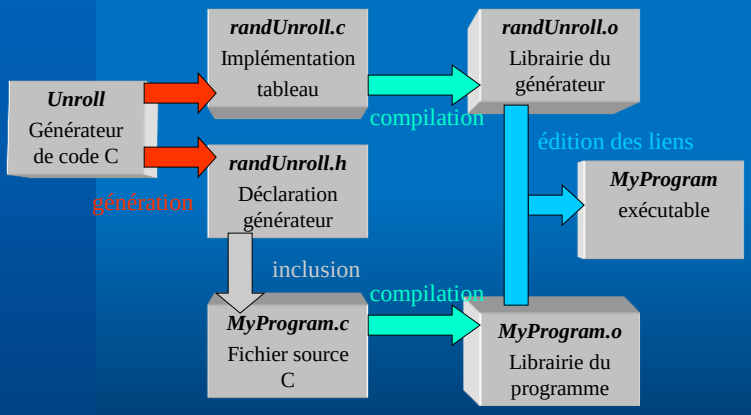
- Le calcul 2 à la puissance 10 est réalisé à la compilation. Le code qui sera généré est équivalent à ce qui suit :

```
void main()
{
   const long k = 1024; // !
}
```

- Cette technique peut-être combinée à du code C++ « normal » on obtient alors un programme hybride.
- Application : Optimisation par « unrolling »

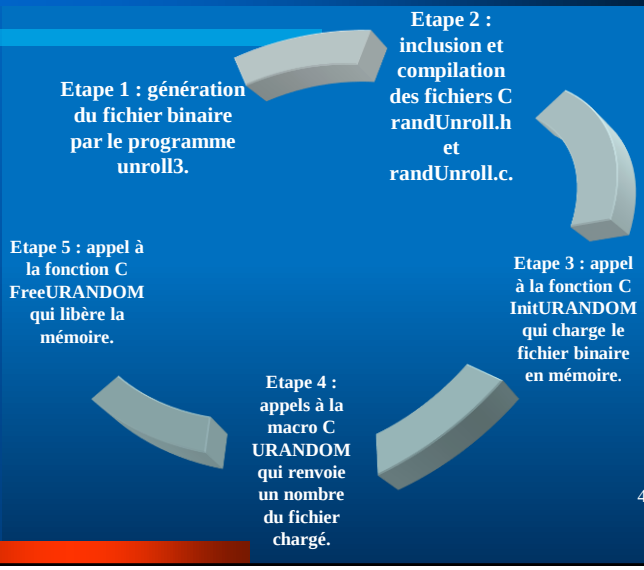
38

Autre piste d'utilisation de l'IDM : Accélération de la génération de nombres aléatoires La méthode dite du « unrolling »

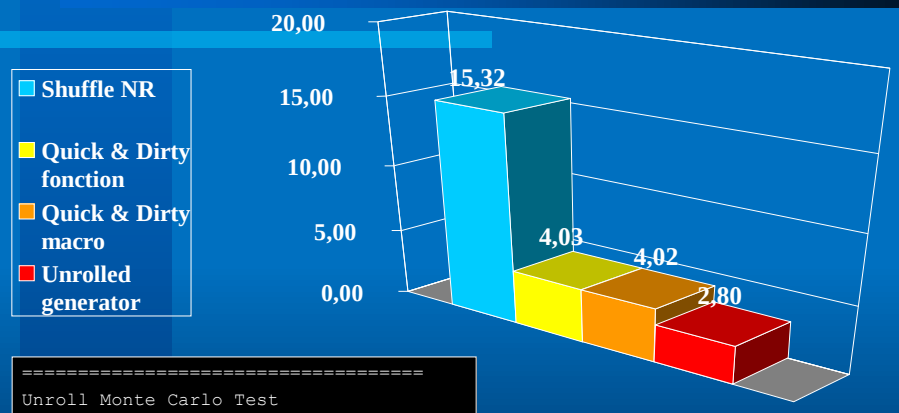


Ébauche du unrolling

Unrolling avec « Memory Mapping »



Tests sur sur 4 générations d'ordinateurs

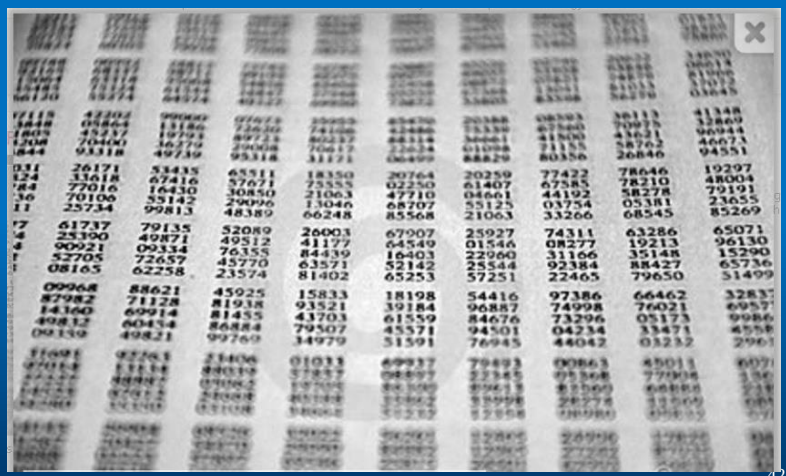



```
==================================
Unroll Monte Carlo Test
==================================
Testing unrolled Shuffle NR generator
Using 50000000 experiences
Computing time: 3710 ms
```

Temps de calcul des différents générateurs aléatoires, pour 100 millions de tirages (en secondes)

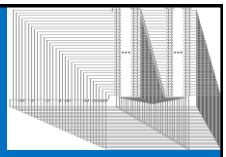
41

Part II.2 - Recent history of top generators & partitioning techniques



42

Before the beginning of the Millenium…



Matsumoto & Nishimura - Mersenne Twister 1997

- Large linear-feedback shift register
- The "Seed" is a register 19937 bits long, stored in a 624 array of 32 bits values
- Period: $2^{19937} - 1$ (Mersenne Prime #)
- Claimed to be free of long-term correlations and equidistributed in 623 dimensions
- Not cryptosecure

(2004) WELL… Panneton, L'Ecuyer & Matsumoto

- In fact the huge-period generators proposed so far are not quite optimal when assessed via their equidistribution properties.
- WELL corresponds to new generators, with better equidistribution and "bit-mixing" properties for equivalent period length and speed.
- Approximately half of the coefficients of the characteristic polynomial of these generators are nonzero.
- The state of this new kind of generators evolves in a more "random" way than for the original Mersenne twister.
- This can reduce the impact of persistent dependencies among successive output values, which can be observed in certain parts of the period of gigantic generators such as the MTs.

44

MT has been updated : 2002, 2004, 2006, 2009,2010, 2011,2012…

- MT: Improvement of the long recovery of zero-excess initial state (enhancements since 2002)
- SFMT (Saito & Matsumoto – 2006)
  - SIMD-oriented Fast Mersenne Twister (SFMT) using SSE instruction set of modern processors
  - SFMT is roughly twice faster than the original Mersenne Twister with periods ranging from 2^{607}-1 to 2^{216091}-1
  - Has a better equidistibution property,
  - And a quicker recovery from zero-excess initial state.
- MTGP… for GP-GPU (introduced in 2009 presented later)
- TinyMT – current last generator of the familly

45

NOW: A Quick survey of random streams parallelization (1) Using the same generator

- The Central Server (CS) technique
- The Leap Frog (LF) technique. Means partitioning a sequence $\{x_i, i=0, 1, \ldots\}$ into 'n' sub-sequences, the j^{th} sub-sequence is $\{x_{kn+j-1}, k=0, 1, \ldots\}$ - like a deck of cards dealt to card players.
- The Sequence Splitting (SS) – or blocking or regular/fixed spacing technique. Means partitioning a sequence $\{x_i, i=0, 1, \ldots,\}$ into 'n' sub-sequences, the j^{th} sub-sequence is $\{x_{k+(j-1)m}, k=0, \ldots, m1\}$ where m is the length of each sub-sequence
- The Indexed Sequences (IS) - or random spacing. Means that the generator is initialized with 'n' different seeds/statuses
- The Cycle Division or Jump ahead technique. It can be used for both Leap Frog or Sequence splitting. It deals with the Analytical computing of the generator state in advance after a huge number of cycles (generations)

46

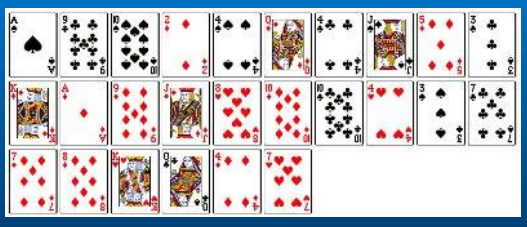
Quick survey of random streams parallelization
(2) Using different generators: parameterization

- The same type of generator is used with different parameters for each processor meaning that we produce different generators
- In the case of linear congruential generators (LCG), this can rapidly lead to poor results even when the parameters are very carefully checked. (Ex: Mascagni and Chi proposed that the modulus be Mersenne or Sophie Germain prime numbers)
- Explicit Inversive Congruential generator (EICG) with prime modulus has some very compelling properties for parallelizing via parameterizing. A recent paper describes an implementation of parallel random number sequences by varying a set of different parameters instead of splitting a single random sequence (Chi and Cao 2010).
- In 2000 Matsumoto et al proposed a dynamic creation technique

47

Part II.3

Details of the different parallelization techniques for pseudorandom numbers…



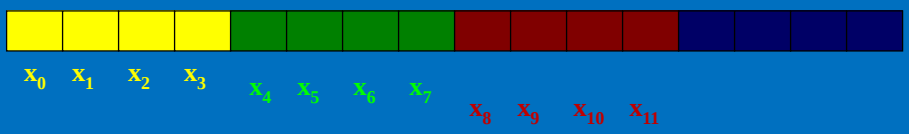
48

Central server approach

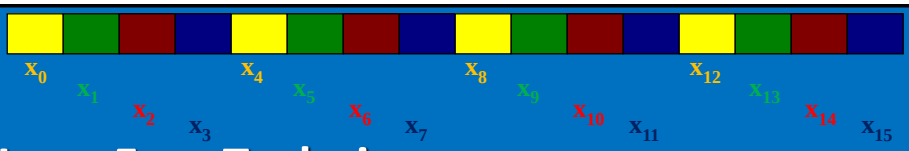
- Not truly parallel.
- A central server (CS), running a “good” sequential RNG and provides pseudorandom numbers on demand – feeding other processors (first demand, first served)
- The two major drawbacks for this approach are the following:
  - a simulation using this technique will not be reproducible (if the number of nodes / CPUs / cores is varying)
  - the central server will become a bottleneck when we have a large number of processors
- Suitable for serious games with limited parallelism but not for scientific applications.

49

Sequence splitting technique



- Split a sequence into non overlapping contiguous blocks.
- Hechenleitner showed that in the OMNeT++, the spacing between sequences was set to 1 million drawings led to biased results (due to inter-sequence correlations) for processes using more random numbers.
- Even if overlapping can be easily avoided, long range correlations in the initial RNG can lead to small range correlations between the potential substreams.

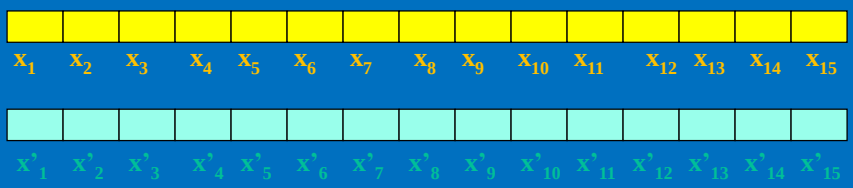


Leap Frog Technique

- Partition of random stream like a deck of cards dealt to card players.
- Given the period p of the global sequence, the period of each stream is p/N.
- Like with the splitting technique, the long range correlations in the initial RNG can lead to small range correlations between the potential substreams, particularly if we have a large number of processors.
- Wu and Huang 2006 showed that depending on the interval used (ie the number of processors) cross correlations can be observed.
- A case where the quality of the original RNG is seriously affected by the LF technique was shown by Hellekalek.
- When this technique is used without cycle division (explained in a forward slide), the performances are divided by the number of LPs and we find again the bottleneck problem exposed in the Central Server technique.

51

Independent sequences technique



- The Indexed Sequences (IS) method builds a partition of N streams by initializing the same generator with N random statuses.
- In the case of old LCGs it was named random seeding.
- For modern generators with a more complex status, the random statuses are generated with another RNG and this technique is interesting when generators have a huge period.
- The risk is of course to get a bad initialization. Recent history has showed that even some of the best RNG algorithm could fail when badly initialized.
- In 2008, Reuillon proposed 1 million statuses for the first Mersenne Twister (MT) with its $2^{19937} - 1$ period, he used a RNG with cryptographic qualities to propose independent and well balanced bit statuses.

Cycle division technique « jump ahead » (1/2)

- **This technique enables the analytical computing of the generator state in advance after a huge number of cycles (generations) and corresponds to a jump ahead in the random stream.**
- **Interesting with generators that have very large periods.**
- **Until recently, MT and the WELL generators, based on linear recurrences modulo 2, did not have an efficient companion software providing multiple disjoint streams and substreams with cycle division techniques.**
- **Matsumoto and L'Ecuyer teams joined to develop a viable jump-ahead algorithm taking some milliseconds on that time processors (2008).**
- **More efficient algorithms exist, like the MRG32k3a generator from L'Ecuyer which takes only a few microseconds for its jump ahead (but can be 2 times slower than MT for regular generation)**

… Independent streams and substreams (2/2)

- With cycle division a random stream can be controlled by having it jump to fixed checkpoints.
- Interesting if you may want to return to a previous state of a simulation.
- With substreams one can jump back and forth among multiple substreams.
- Initially available for two generator types supporting multiple independent streams and substreams:
  - The Combined Multiple Recursive **MRG32k3a – 2^{124}**
  - the Multiplicative Lagged Fibonacci **MLFG6331_64** generators (2^{127})
- Now possible with MTs & WELLs generators (less efficient)

(2) Different generators - detail : DC dynamic creation of generators

- **Each generator has a different ID going for instance from 1 to the number of generators needed.**
- **The ID of a generator is encoded in the characteristic polynomial of the MT RNG – then each generator is assumed highly independent from the other ones**
- **PNRGs based on linear recurrences with characteristic polynomials relatively prime to each other are supposed to be mutually independent**
- **As it is stated one quickly understands that there is no mathematical proof…**
- **Currently it has been found much safer than some other parameterization approaches.**

55

Summary of requirements for massive parallelism in Stochastic Applications

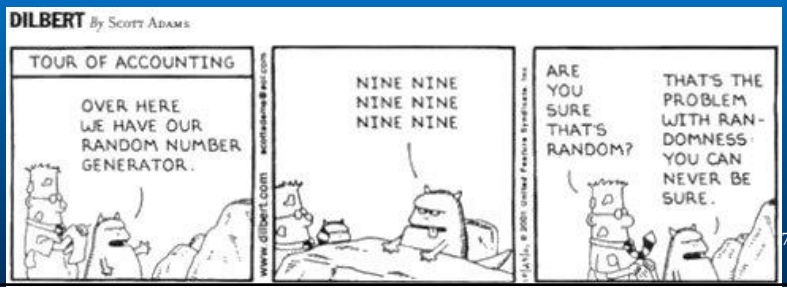
- Fits with the Independent bag-of-work paradigm for **different distributed computing platforms**
- Ensure as much as possible the independence of underlying random number streams
  - ✓ Use fine partitioning techniques
  - ✓ Use dedicated libraries
- If your application is critical: rigorous testing is mandatory

56

TESTING : “Random number generators should not be chosen at random” (D. Knuth)

- Even if this is not the case with D.C. Matsumoto et al. warn us since 2000
- Dynamic creator “is not tested well yet, so it may contain many bugs. We are not responsible for any damage caused by these codes.”

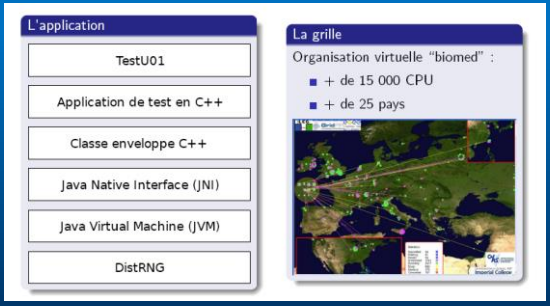
This means that we have to test the produced generators.



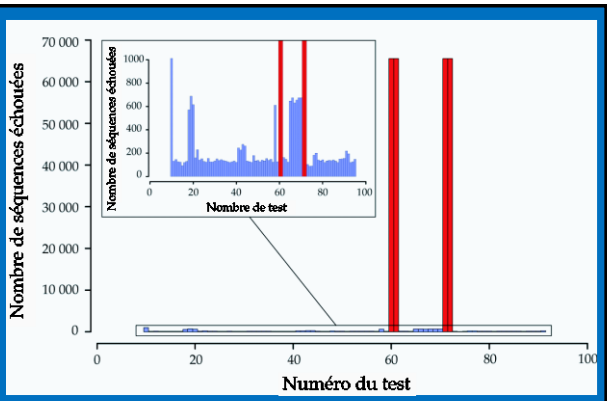
Production and test of MT generators

Credit : Reuillon - Hill

- Generation of 65536 generators
- Each generated MT has a period of 2^{512}-1
- Test U01 – « crush » battery with 96 tests
- Identification and archival of more than 63000 good « statuses »
- Presented at the 2008 EGEE Grid user forum
- 59 years of computing
- Executed in 13 days



Results



- Identification of some weak MTs failing some tests
- Archival of more than 63000 good « statuses » out 65536
- MT generators do not claim to be cryptosecure and as expected each generated MT failed to the 4 cryptographic tests

Credit : Reuillon - Hill

Survey of Libraries & software for PRNG and stoch. sim. paralellization

- **SPRNG (Scalable Parallel Random Number Generation) library – (C/C++) Mascagni et al.**
- **Parallel streams and substreams in C/C++, Fortran, R, Java and Matlab, R-streams (L'Ecuyer et al. 2002-2005)**
- **Evolution and maintenance mainly in Java for SSJ (Stochastic Simulation in Java)**
  - ✓ **A very good library from L'Ecuyer et al.**
- **JAPARA – A Java Parallel Random Number Library for High-Performance Computing – Coddington et al (2004)**
- **Grid-Computing Infrastructure for Monte Carlo Applications (GCIMCA) - (Mascagni et al 2003)**
- **DistMe / DistRNG – Reuillon et al. > OpenMole (2010)**
- **Dynamic creation of MT generators (for CPU & GPU)**