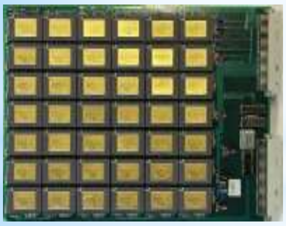
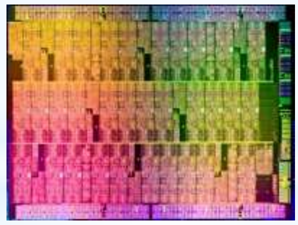
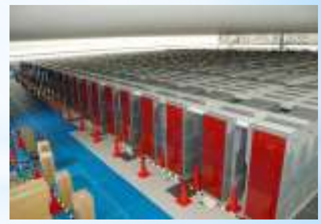
Reproducible High Performance computing for stochastic models and simulations

Hill David
Université Blaise Pascal
ISIMA/LIMOS UMR CNRS 6158


REPRODUCIBILITY BEGINNER's

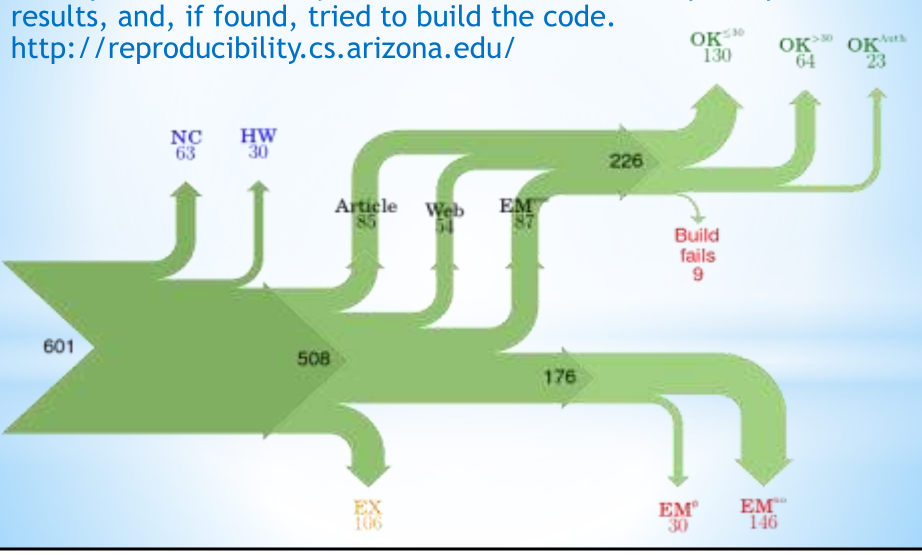
Reproducibility ? (defn.)

- In Fomel and Claerbout 2009:
  - ✓ Reproducibility **often means replication** depending on scientists
- In Drummond 2009[1]:
  - ✓ "*Reproducibility **requires changes; replicability avoids them***"
- In Demmel and Nguyen 2013
  - ✓ "*Reproducibility, i.e. **getting bitwise identical results from run to run***" > means in fact : **"repeatability"**
- In Revol and Théveny 2013.
  - ✓ "*What is called **numerical reproducibility** is the problem of getting the same result when the scientific computation is run several times, either on the same machine or on different machines, with different numbers of processing units, types, execution environments, computational loads, etc.*"

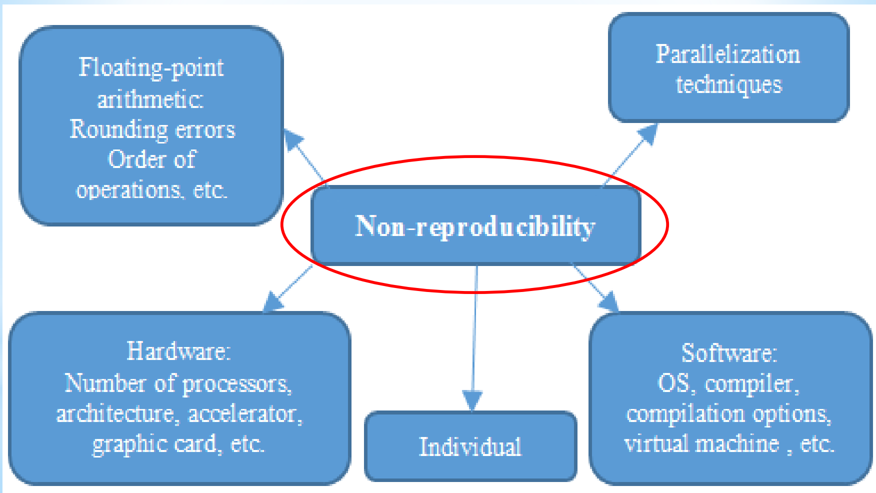
1: http://www.site.uottawa.ca/ICML09WS/papers/w2.pdf

A recent study at Arizona University

This study examined 601 papers from ACM conferences and journals, attempted to locate any source code that backed up the published results, and, if found, tried to build the code.
http://reproducibility.cs.arizona.edu/



Some Reasons for numerical reproducibility failures



Towards Exascale Computing…

- The goal of Exascale computing is to multiply by **10x** the performance of the fastest machine on operation.
- We can anticipate that Exascale systems will have around around 10^9 computing cores.
- This also means that at the same time each standard nodes will be able to deliver tenths of teraflops.
- This will help to generate much faster, more precise and more complex simulations, higher quality medical imaging will yield faster and personalized medicine with smarter medical diagnostic and treatment.
- Parallel Stochastic simulations are useful at this scale, particularly because they are "fault" tolerant.

Some scalability problems

1. Energy questions
2. Reliability (**hardware errors will be the rule…**)
   - ✓ Software & Hardware (including « soft » errors)
3. Performances: the need for « **disruptive technologies** »
   - ✓ Processors, InterConnect, IO (at affordable energy cost)
4. Really '**Big**' data & output Results interpretability
5. Software in many area:
   - ✓ **Focus : optimization speed while keeping Numerical reproducibility and repeatability (ability to debug !)**

Programmability

- Exascale application will involve approximately around O(10^9) logical cores (hardware threads).
- No human being can program, debug or optimize directly this many threads.
- Hope: High-level languages and DSL will allow us to express that parallelism more effectively
- Positive: data-parallel applications, can use the same kind of automation that has proved successful in areas like geometry and meshing and then map them onto complex graphical representations.
- **Task-parallel applications**: we can give a new focus on **statistical methods and Monte Carlo approaches** to develop more resilient software.

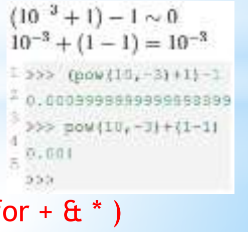
Reliablity

- Mandatory co-design (Hard. & Soft.) for HPC
- They are currently separated (eg. Introduction of- **Out of order** intructions…) is it a real option for "Exascale Comp" ?
- Hardware designers have been struggling with how to make systems a thousand times more reliable per bit-operation to keep us at the same level we are at in today's best systems.
- The only reason to do Exascale computing is to address increasingly more complex issues. This will require even more complex software.
- **Software complexity** is the **N°1** **cause of unreliability** in computation today… far exceeding hardware's worst efforts!

9

Zoom in: « Out of Order Execution » of floating point instructions

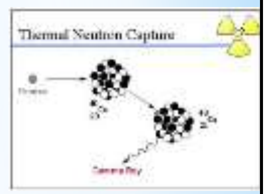
- Out-of-order execution is also known as **dynamic execution**. Most modern high-performance microprocessors optimize the execution of instructions based on the availability of input data to avoid delays.
- The original order of instructions in a program is no more respected.
- The micro-processor avoids having parts of its internal computing units being idle by processing the next instructions which are able to run immediately and "independently".
- It is the equivalent of the software dynamic recompilation (or just-in-time compilation) which improves instruction scheduling.



- Remember:
  floating point arithmetic is not associative (for + & *)
  ex: a+(b+c) != (a+b)+c.

Reliability & HPC…
…Silent & Soft errors…

1. Change the system state (external forces)
   - ✓ Alpha particles
   - ✓ Cosmic rays (High Energy Particles from space)
   - ✓ Thermal neutrons
   - ✓ Variation in voltage, temperature, etc.



2. They are at the origin of ECC…
   1. To avoids bits flips in memory cells
   2. There is also a rising of soft errors in arithmetic units !!!
   3. The more we size down the more this problem increases.
   4. Chip manufacturers spend money and silicon space to avoid this kind of errors
3. Soft errors are difficult to detect and reproduce – use spare time of SuperMachines ?

Silent Data & Result Corruption

- The integrity and the accreditation of the Science discoveries we want to make with computers is threatened (electrons speed above light speed…?!)
- Soft errors are not only corrupting data, **but they now affect calculations**. (**1 per month currently, up to one per hour at Exascale !**)
- ECC is essential for memory, but is does not solve this problem. We also have to face this with O.S. systems, middleware, and programming models.
- Indeed, soft errors will increase with the machine size and they also increase within modern arithmetic units.

12

Protecting state & logic (Reliablity)

- We can effectively protect correctness of state but correctness of logic poses special challenges.
- **State can be protected at about a 10% energy overhead.**
- Logic correctness requires more invasive approaches with some degree of redundancy **that could well exceed the 10% overheads**
- Current R&D focuses on residue checking (self checking FPU) and redundant multi-threading. This approach has a significant energy overheads;
- Due to the energy issues, we are going to be more limited than we should have been in protecting logic paths.
- This will require a significant degree of cooperation between software and hardware engineers.

13

HW/SW Codesign (for Reliablity)

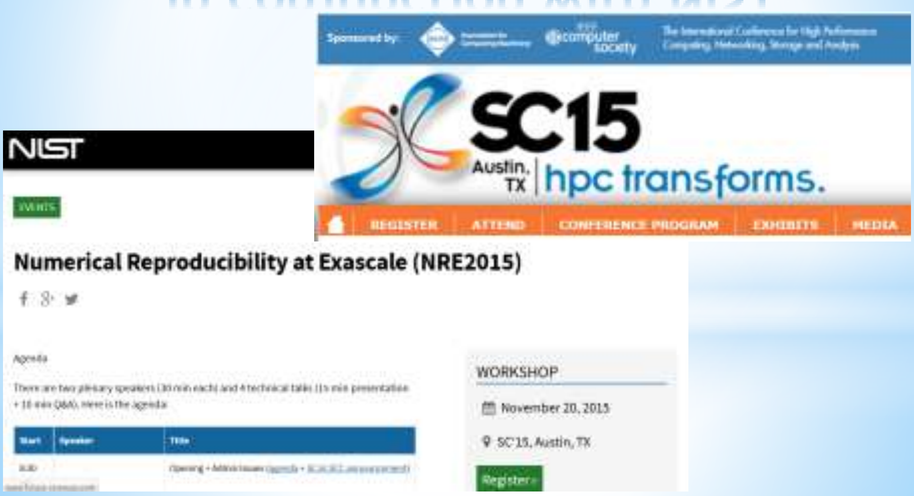
- Can we identify at compile time certain critical regions which need stronger correctness guarantees?
- We are already generating terabytes to petabytes of state per second. At exascale we will be generating exabytes of state each second.
- A single wrong bit can vitiate the entire calculation.
- For many scientific calculations: we should be able to gracefully tolerate many kinds of bit errors, and also the loss of many kinds of local resources.
- For example: in many Monte Carlo simulations, the loss of a processor does not imply the inherent failure of the simulation.

14

Checkpointing (Reliablity)

- Limits of classical checkpointing will be reached : a fault every hours (or less) with current MBTF – but an Exascale checkpoint could last 30 min. at 1 Tb/s without the use of exepensive disruptive technologies (Ultra Fast SSD, PCM memories)
- **Without a radical change we are going to be much worse than we are today…**
- We have to build a much higher level of local check-pointing capability into our software and hardware systems.
- **Parallel Stochastic Simulations could checkpoint must faster with only intermediate results and all the pseudo-random number generator statuses.**
- Using **raided non-volatile memory**, we could checkpoint state very often by moving copies of needed application state to nearest neighbor nodes (they only draw power when in use, this would have minimal energy implications).

Reproducible // Stochas. Sim Results presented at an SC Workshop in conjunction with NIST



Approach : Application Driven Parallel Stochastic Simulations

- Easier if they fit with the independent bag-of-work paradigm.
  - Such stochastic simulations can easily tolerate a loss of jobs, if hopefully enough jobs finish for the final statistics..
- Must use "independent" Parallel random streams.
  - Statuses should be small and fast to store at Exascale (Original MT – 6Kb status – MRG32K3a 6 integers)
- Should fit with **different distributed computing platforms**
  - Using regular processors
  - Using hardware accelerators (GP-GPUs, Intel Phi…)

17

A method: Repeatability of parallel stochastic simulations

Remember that a stochastic program is « deterministic » if we use (initialize and parallelize) correctly the pseudo-random number.

1. A process or object oriented approach has to be chosen for **every stochastic objects which has its own random stream**.
2. Select a **modern and statistically sound generators** according to the most stringent testing battery (TestU01);
3. Select a **fine parallelization technique adapted to the selected generator**,
4. The simulation must first be designed as **a sequential program which uses a parallel design**. The sequential execution – with a compiler disabling of "out of order" execution will be the reference to compare parallel and sequential execution at small scales on the same node.
5. Externalize, sort or **give IDs to the results for reduction** in order to keep the execution order or use compensated algorithms

[Hill 2015] : Hill D., "Parallel Random Numbers, Simulation, Science and reproducibility". IEEE/AIP - Computing in Science and Engineering, vol. 17, no 4, 2015, pp. 66-71.

An object-oriented approach?

A system being of collection of interacting "objects" (dictionary definition) – a simulation will make all those objects evolve during the simulation time with a precise modeling goal.

- Assign an « independent » random stream to each stochastic object of the simulation.
- Each object (for instance a particle) must have its own reproducible random stream.
- An object could also encapsulate a random variate used at some points of the simulation. Every random variate could also have their own random stream.

[Hill 1996] : HILL D., "Object-oriented Analysis and Simulation", Addison-Wesley, 1996, 291 p.

Back to basics for stochastic simulations
Repeatable Par.Rand.Num.Generators

Quick check with some **top PRNGs** used with different execution context (hardware, operating systems, compilers…

1. Use exactly the same inputs
2. Execute on various environments
3. Compare our outputs with author's outputs (from publications or given files)

Reproducing results - portability 1/4

- Errors found:
  - for different hardware,
  - different operating systems,
  - different compilers.

Table 3: Testing of reproducibility for 7 different PRNGs (MT19937 with 2 versions, TinyMT with 2 versions, MRG32k3a, WELL512, MLFG64) performed on 5 different processors (Intel E5-2650v2, Intel E5-2687W, Core 2 Duo T7100, AMD 6272 Opteron, Core i7-4800MQ) with different compilers (gcc, icc, lcc, open64, MinGW, Cygwin) were tested.

Generator	E5-2650v2		E5-2687W		Core 2 Duo T7100		AMD Opteron (TM) 6272		Core i7-4800MQ			
	gcc	icc	gcc	icc	gcc	open64	gcc	open64	Cygwin	MinGW	lcc	
											lc	lc64
MT19937	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
MT19937_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TinyMT_32	Yes	Yes	Yes	Yes	Yes	NO	Yes	Yes	Yes	Yes	Yes	Yes
TinyMT_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	NO	NO	Yes
MRG32K3a	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
WELL512a	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
MLFG_64	Yes	Yes	Yes	Yes	N/a	N/a	Yes	Yes	Yes	Yes	Yes	Yes

Reproducing results - portability 2/4

- Errors found:
  - Different Compilers (2 cases)
  - With Identical Hardware (2 cases)
  - With different operating Systems (2 cases)

Table 4: Results for TinyMT_32 PRNG on Core 2 Duo T7100 running Ubuntu-13.04 with open64-i386

Expected results CHECK32.OUT.TXT	Results obtained with Open64 i386
0.5714423	0.5714422
0.7421532	0.7421533
0.6638085	0.6638086
0.4334422	0.4334421
0.1254190	0.1254189
0.4688578	0.4688579
0.2675911	0.2675910
0.1784127	0.1784128

Table 5: Results for TinyMT_64 PRNG on Core i7-4800MQ running Windows 7 with MinGW

Expected results CHECK64.OUT.TXT	Results obtained with MinGW gcc
1.152012609994736	1.152012609994737
1.363201836673650	1.363201836673651
1.218170930629463	1.218170930629464

Reproducing results - portability 3/4

- Errors found :
  Problems Encountered With 32 And 64 Bits Architecture For The Same Compiler (lcc compiler 32 bits - ok for 64 bits)

Table 6: Results for TinyMT_64 PRNG on Core i7-4800MQ running Windows 7 with lc 32 bits

Expected results CHECK64.OUT.TXT	Results obtained with lc 32 bits compiler
0.125567123229521	0.514472427354387
1.437679237017648	1.386730269781771
0.231189305675805	0.112526841009551
0.777528512172794	0.197121666699821

Reproducing results - portability 4/4

- Errors found :
  when comparing between **real and virtual machines**
  a “Real” Core 2 Duo T7100 and a “Virtual Machine” (Virtual Box on top of Windows 7 with Intel(R) Core™ i7-4800MQ)

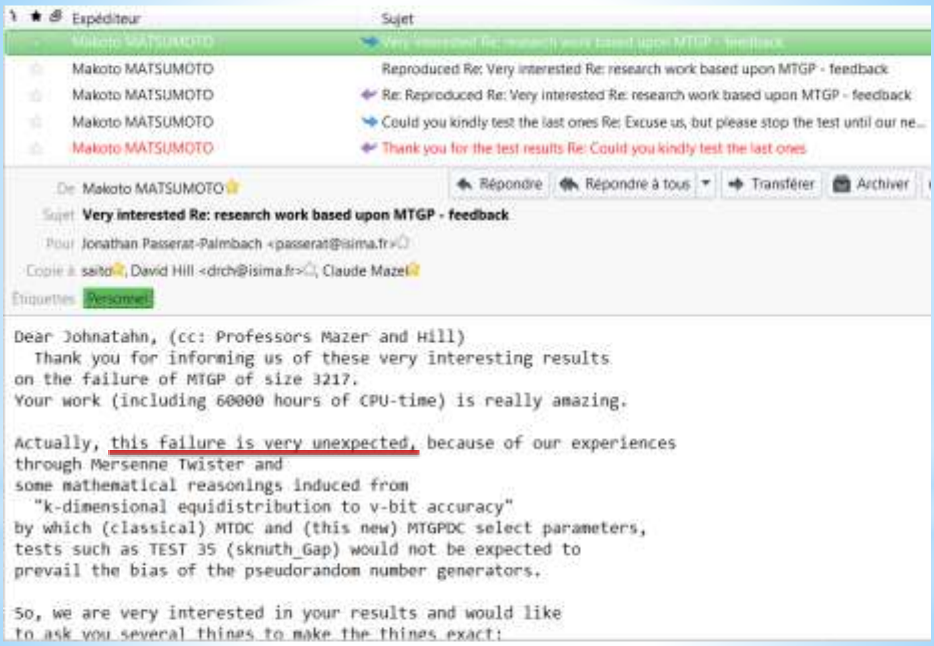
Table 4: Results for TinyMT_32 PRNG on Core 2 Duo T7100 running Ubuntu-13.04 with open64-i386

Expected results CHECK32.OUT.TXT	Results obtained with Open64 i386
0.5714423	0.5714422
0.7421532	0.7421533
0.6638085	0.6638086
0.4334422	0.4334421
0.1254190	0.1254189
0.4688578	0.4688579
0.2675911	0.2675910
0.1784127	0.1784128

Table 7: Results for TinyMT_32 PRNG with open64-i386 on virtual machines of Ubuntu-13.04 and 14.04

Expected results CHECK32.OUT.TXT	Results obtained with Ubuntu 13 on Virtual Box	Results obtained of Ubuntu 14 on Virtual Box
0.6455914	0.6455913	0.6455913
0.9415597	0.9415598	0.9415598
0.9034473	0.9034472	0.9034472
0.9348063	0.9348064	0.9348064
0.7581965	0.7581964	0.7581964

- **Will this impact Docker for Windows since it works on top of virtual Box ?**


Expéditeur	Sujet
Makoto MATSUMOTO	Very interested Re: research work based upon MTGP - feedback
Makoto MATSUMOTO	Reproduced Re: Very interested Re: research work based upon MTGP - feedback
Makoto MATSUMOTO	Re: Reproduced Re: Very interested Re: research work based upon MTGP - feedback
Makoto MATSUMOTO	Could you kindly test the last ones Re: Excuse us, but please stop the test until our ne...
Makoto MATSUMOTO	Thank you for the test results Re: Could you kindly test the last ones

Répondre | Répondre à tous | Transférer | Archiver

De Makoto MATSUMOTO
Sujet **Very interested Re: research work based upon MTGP - feedback**
Pour Jonathan Passerat-Palmbach <passerat@isima.fr>
Copie à saito, David Hill <drch@isima.fr>, Claude Mazel
Étiquettes Personnel

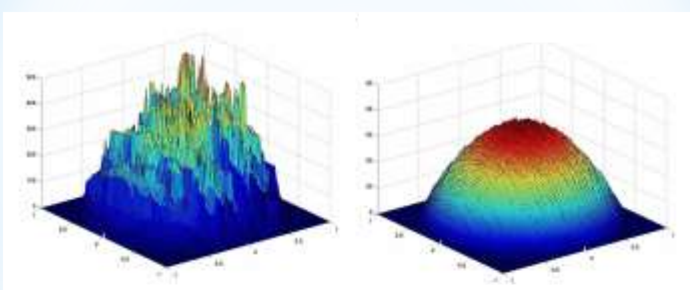
```
Dear Johnatahn, (cc: Professors Mazer and Hill)
  Thank you for informing us of these very interesting results
on the failure of MTGP of size 3217.
Your work (including 60000 hours of CPU-time) is really amazing.

Actually, this failure is very unexpected, because of our experiences
through Mersenne Twister and
some mathematical reasonings induced from
  "k-dimensional equidistribution to v-bit accuracy"
by which (classical) MTDC and (this new) MTGPDC select parameters,
tests such as TEST 35 (sknuth_Gap) would not be expected to
prevail the bias of the pseudorandom number generators.

So, we are very interested in your results and would like
to ask you several things to make the things exact:
```

25

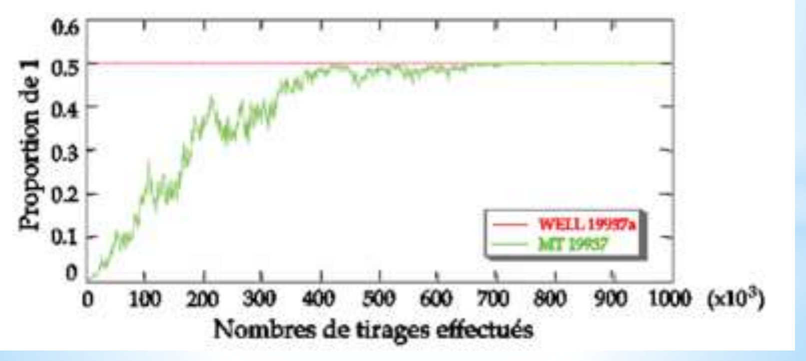
*Let's « see » the potential impact of the generator quality…



Two results of the same simulation (sequential) – PDE Harmonic solution computed with Brownian movements.
On the left the image is obtained with Linux rand (which is already far better than the old std UNIX rand on 15bits)
On the right – same simulation with Mastumoto Mersenne Twister (1997 version) – right solution elipsoid with a circular section.

* There is no perfect Generator…
Ex: First Mersenne Twister : a known default…

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



Some top PRNGs (Pseudo Random Number Generators)

Only Green PRNG are recommended:

- LCG (Linear Congruential Generator) - $x_i = (a*x_{i-1} + c) \bmod m$ forget them for Scientific Computing see [L'Ecuyer 2010]
- LCGPM (Linear Congruential Generator with Prime Modulus – could be Mersenne or Sophie Germain primes)
- MRG (Multiple Recursive Generator) $x_i = (a_1*x_{i-1} + a_2*x_{i-2} + \ldots + a_k*x_{i-k} + c) \bmod m$ – with k>1
- (Ex: **MRG32k3a & MRG32kp** – by L'Ecuyer and Panneton)
- LFG (Lagged Fibonacci Generator) $x_i = x_{i-p} \square x_{i-q}$
- **MLFG** (Multiple Lagged Fibonacci Generator) – Non linear by Michael Mascagni MLFG 6331_64
- L & GFSR (Generalised FeedBack Shift Register…) Mod 2
- Mersenne Twisters – by Matsumoto, Nishimura, Saito (MT, SFMT, MTGP, TinyMT) – **WELLs** Matsumoto, L'Ecuyer, Panneton

See [Hill et al 2013] for advices including hardware accelerators

Quick survey of random streams parallelization

(1) Using the same generator

- *The **Central Server** (CS) technique (avoid for flexible reproducibility)
- *The **Leap Frog** (LF) technique. Means partitioning a sequence $\{x_i$, i=0, 1, …} into 'n' sub-sequences, the j^{th} sub-sequence is $\{x_{kn+j-1}$, k=0, 1, …} - 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, …,} into 'n' sub-sequences, the j^{th} sub-sequence is $\{x_{k+(j-1)m}$, k=0, …, m1} where m is the length of each sub-sequence
  - *Jump Ahead technique (can be used for both Leap Frog or Sequence splitting)
- *The **Cycle Division** or **Jump ahead** approach. Analytical computing of the generator state in advance after a huge number of cycles (generations)
- *The **Indexed Sequences** (IS) - or random spacing. Means that the generator is initialized with 'n' different seeds/statuses

29

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**

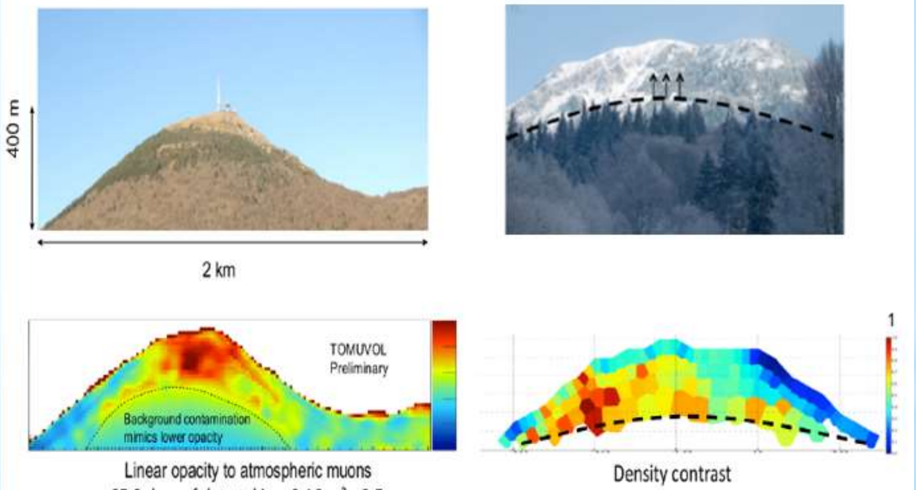
30

Application : Reproducible HPC for Muonic Tomography - billions of threads…

Labex Clervolc Tomuvol project with C. Cârloganu

P. Schweitzer thesis for HPC

2D Tomographic rendering



Linear opacity to atmospheric muons
65.8 days of data taking, 0.16 m² x 0.5 m

Density contrast
14 days of data taking, 0.66 m² x 1 m

Optimization for a single « hybrid » node (Intel E52650 & Xeon Phi 7120P)

Parallel stochastic simulation of muonic tomography

- Parallel programming model using p-threads
- On stochastic object for each Muon
- Multiple streams using MRG32k3a[1]
- A billion threads handled by a single node (queue & pooling)
- Compiling flags set to maximum reproducibility

Table 3: Performance of a billion event simulation when parallelized on 1 Phi, 1 CPU, 2 CPUs

	Intel Xeon Phi 7120P	Intel Xeon E5-2650v2	2x Intel Xeon E5-2650v2
Time	48 h 49 min	36 h 32 min	18 h 17 min
Speedup	1	1.34	2.67

(1) P. L'Ecuyer, R. Simard, E. J. Chen, and W. D. Kelton, ``An Objected-Oriented Random-Number Package with Many Long Streams and Substreams", Operations Research, Vol. 50, no. 6 (2002), pp. 1073-1075.

Bit for bit reproducibility

Do not expect bit for bit reproducibility when working on Intel Phi vs. regular Intel processors[1].

- We observed bit for bit reproducibility in single precision but not in double precision (and with the expected compiler flags)
- The relative difference between processors (E5 vs Phi) in double precision were analyzed and are shown below:

Table 1: Relative CPU-Phi differences between the results and number of altered bits

Difference ↓ \ Result →	Position X	Position Z	Direction X	Direction Y	Direction Z
0 bit: bit for bit reproducibility	4922	4934	4896	4975	4913
1 bit: $1.11E-16 \leq \Delta < 2.22E-16$	25	21	14	5	18
2 bits: $2.22E-16 \leq \Delta < 4.44E-16$	21	18	52	4	31
3 bits: $4.44E-16 \leq \Delta < 8.88E-16$	15	12	23	6	12
4 bits: $8.88E-16 \leq \Delta < 1.78E-15$	10	7	5	4	10
≥ 5 bits: $1.78E-15 \leq \Delta < 2.25E-11$	7	8	10	6	16

(1) Run-to-Run Reproducibility of Floating-Point Calculations for Applications on Intel® Xeon Phi™ Coprocessors (and Intel® Xeon® Processors) – by Martin Cordel https://software.intel.com/en-us/articles/run-to-run-reproducibility-of-floating-point-calculations-for-applications-on-intel-xeon

Relative difference (Phi vs E5)

With regular compiler flags – no hope of reproducibility

With a careful use of compiler flage the results on the two architectures are of the same order,

Both of them have the same sign and the same exponent (even if some exceptions would be theoretically possible, they would be very rare).

The only bits that can differ between these results are the least significant bits of the significand.

For a given exponent e, and a result $r1 = m \times 2e$, the closest value greater than r1 is $r2 = (m + \varepsilon d) \times 2e$, where εd is the value of the least significant bit of the significand: $\varepsilon d = 2^{-52} \approx 2.22\ 10^{-16}$.

Intel Compiler flags:

- ✓ “-fp-model precise -fp-model source -fimf-precision=high -no-fma” for the compilation on the Xeon Phi
- ✓ “-fp-model precise -fp-model source -fimf-precision=high” for the compilation on the Xeon CPU.

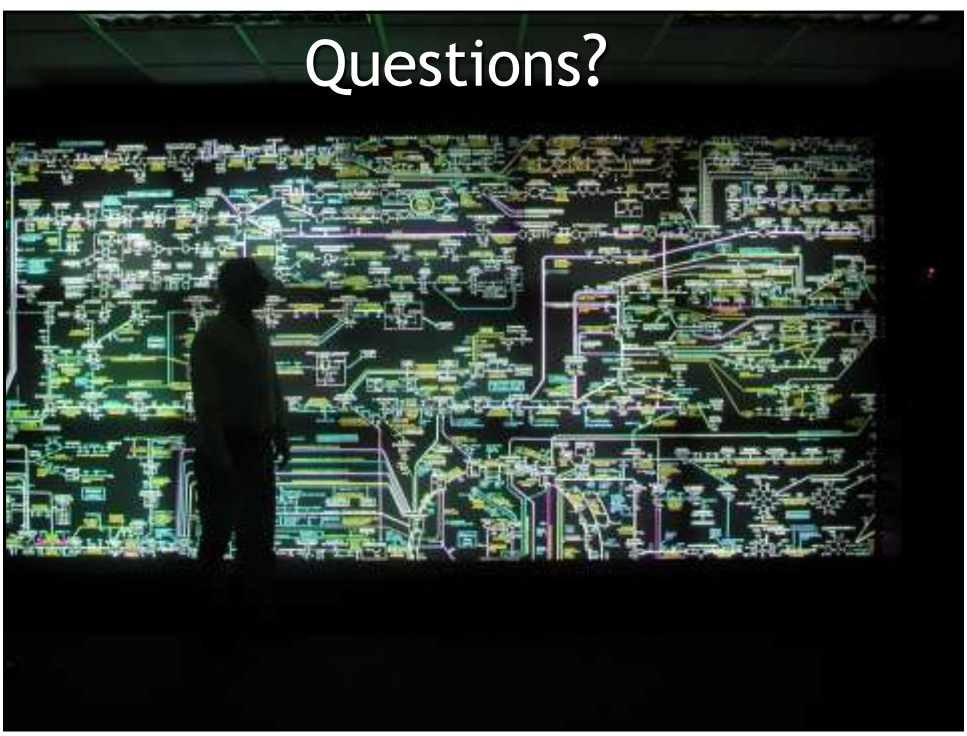
Conclusion

- **Repeatability achieved on identical execution platforms**
- **Numerical differences reduced between classical Xeon and Intel Xeon Phi.**
- **Numerical Reproducibility is possible for Parallel Stochastic applications with independent computing on homogeneous nodes.**
- **This approach can be used for low reliability supercomputers (with current MTTF below 1 day)**
- Key elements of a method have been presented to produced numerically reproducible results for parallel stochastic simulations comparable with a sequential implementation (before scaling on Petaflopic or future Exascale systems)
- **Numerical replications is very important for scientists in many sensitive areas, finance, nuclear safety, medicine…**

Perspectives

- Simulation of parallel independent processes can be now considered as “easy”,
- **BUT: simulating time-dependent entities or interacting entities, with numerical reproducibility across interactions and cross various heterogeneous communicating nodes will be tough.**
- Software simulation of co-routines within the simulation application and synchronous communications can be required in addition to the mandatory assignment of a different random streams to each stochastic object.
- Numerical replication is at least very important for debugging.
- Get prepared with Fault Injection frameworks (like SEFI – Los Alamos National Library, USA)

37



Euh…What's next in HPC ?

Top Future nodes (US CORAL program)

- Will be Hybrid with a shared memory between CPUs – GPUs and FPGAs (currently available)
- Will provide large memories
- Will provide fast storage (Ultra Fast SSD – PCM memories etc.)
- Data centric with computing even at memory and network level.
- Probabilistic approaches ("à la Watson")

Reproducibility for :

- Quantum accelerators (D-Wave, IBM,…) ?
- Neuromorphic chips (for deep learning) ?

Spring 2016 Perspectives

Reproducibility Seminar for Computer Scientists in Auvergne with the input of Philosophers and Lawyers

- ✓ Reproducible Research
- ✓ Numerical Reproducibility
- ✓ Epistemology – how do we build knowledge
- ✓ Ethics and more…

Définitions:

Accuracy :
nombre de chiffres corrects sur un calcul

Precision :
nombres de bits utilisés pour le calcul

Can have the same errors : but with reproducibility