Workshop on Heterogeneous Computing,
16 - 20, July 2012

No Monte Carlo is safe Monte Carlo - more so parallel Monte Carlo

K. P. N. Murthy

School of Physics,
University of Hyderabad

July 19, 2012

Acknowledgement and Apologies

- Thanks to my friends V C V Rao, CDAC and Vinod, CMSD for the invitation
- Rao, Vinod and team mates : you are doing a great job; thanks
- Apologies for the pessimistic title
- I couldn't think of any other "catchy" title that captures the thoughts I want to express in the talk
- My intention is definitely not to dissuade you from practising Monte Carlo;
- On the contrary, I would like to encourage you to do Monte Carlo; to do parallel Monte Carlo;
- there are serious issues about random numbers for a serial machine and a parallel machine
- I just want to bring to your attention these issues

What is randomness ?

- There are several problems with random number generator (RNG)
- Let us start asking : what do we mean by the word random
- a simple example :
  - consider the three finite sequences of binary numbers

$$
\begin{aligned}
(I) &: 0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\\
(II) &: 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\\
(III) &: 1\,1\,0\,1\,1\,1\,0\,0\,1\,1\,0\,1\,1\,1\,1\,0\,1\,0
\end{aligned}
$$

  - Look at each sequence : tell me : is random or is it non-random ?
  - I am sure you will say : Sequence III is random ; I and II are non-random;
  - how did you come to this conclusion ?
  - Remember : all the three sequences have the same probability of occurrence in a coin-tossing experiment

blah ... blah ... on randomness - continued

- hence probability of occurrence of a sequence as calculated from random experiment - here coin tossing : $\{$ Heads - 0 and Tails - 1$\}$ - is not helpful in quantifying "randomness"
- when we say I and II are not random and III is random what exactly we have in mind ?
- sequence III does not have a pattern, whereas I and II do have a pattern
- can we quantify this intuitive notion of randomness ?
- yes: theory of complexity and algorithmic entropy - a hot topic in computer science
- if you actually toss a coin eighteen times you are unlikely to get precisely any one of the three sequences; but then you will most likely get a sequence which does not have any (discernible) pattern, like the sequence III

more blah ... blah

- think of an algorithm to communicate the binary messages I, II and III
- Number of bits required for the algorithms I and II shall be very small;
- I and II are simple
- bits required for III shall be large
- we need to write out explicitly the whole message
- the size of the algorithm is as large as the number of bits required to store the message
- III is complex
- : Algorithmic entropy is a measure of the size, measured in bits, of the algorithm required to communicate the message
- But then I have a surprise for you

Surprise

- message III is actually a binary representation of even (0) and odd (1) nature of the irrational number π in decimal representation.
- A smart scientist may discover this hidden order and communicate III by a very small algorithm;
- in the language of algorithmic randomness III is also simple !
- III is "algorithmically" as non-random as I and II are.
- thus algorithmic entropy is somewhat subjective
- a random number generator (RNG) is a deterministic algorithm
- the whole sequence is predictable
- hence the name pseudo random numbers!

What is it I want to say in this talk

- Of the several issues about random number generators, I want to highlight in this talk, problems arising due to the inevitable inner structure or inner order - the order may be strange (fractal) or simply Euclidean (Marsaglia planes) ;
- if the inner structure interferes (significantly) with the phenomenon you are investigating (in your simulation), then you have to be extremely careful in drawing conclusions - especially if they are profound or at least not run-of-the mill type
- there is no obvious way to detect the interaction between the inner order of RNG and the phenomenon under simulation
- the interaction will vary from problem to problem

- may be, when required, it is a good idea to simulate the same problem with different random number generators and see if the results are consistent with each other and within the calculated Monte Carlo error bars of each simulation
- but then you never know ....
- this problem of interference may get enhanced or suppressed in parallel streams of random numbers - obtained by independent parametrization or by splitting one single stream - *e.g.*, *via* leap frogging

Ideal Random Number Generator (RNG)

- Monte Carlo - a numerical method based on random sampling
- Random numbers fuel the Monte Carlo machine
- $\{\xi_i \ : \ i = 1, \ 2, \ \cdots\}$: a sequence of real random numbers, uniformly distributed between 0 and 1.
- What is an ideal Random Number Generator (RNG) :
  - uniform - we will transform the numbers to the desired probability distribution
  - independent - atleast uncorrelated
  - reproducible - required for debugging : Question : how can a sequence of numbers be reproducible and still be random ?
  - portable - a must atleast in modern times when the computing machines change from day to day, place to place, and person too person

- $\cdots$ continued
  - long period - one order of magnitude longer than what we require in a given program
  - inner structure should be analysable - a subtle point - has got to do with the physicist's way of defining determinism and predictability - I shall discuss this briefly in this talk
  - amenable to parallelisation - difficult
    - the RNG should be adaptable for any (reasonable) number of processors
    - the parallel streams of random numbers produced on different processors should not be correlated
    - the parallel streams should be generated independently for reasons of efficiency
  - in arbitrary stochastic simulation it should give right results - impossible
- Let me tell you : Even **Lord Krishna** can not give you such a "dream" random number generator
- to appreciate the issues raised, we must know how typically a sequence of random numbers is generated in a computer.

Randomness ?

- I am talking of "randomness of a sequence of numbers" and not of a single number.
  - Given a single number you can not tell whether it is random or not;
  - you need to know of its origin
    - Analogy : Given a microstate, you can not tell to which ensemble it belongs
    - you must tell of its probability or of its physical origin
  - randomness is not a private property of a single number it is a collective property of a sequence of numbers

- Analogy : entropy is not a private property of a single microstate;
- it is a property of an ensemble of microstates
- conventional Markov chain Monte Carlo can only calculate average of a private property over a Monte Carlo ensemble of microstates; it can not calculate entropy since it is not a private property of a microstate;
- Ensemble is given; you determine the probability distribution : example: in statistical mechanics we first construct a canonical ensemble and then extract the Boltzmann distribution
- or
- probability distribution is given; you generate an ensemble of realizations; example : in Monte Carlo simulation we take as input the Boltzmann distribution; employing Metropolis algorithm we generate a canonical ensemble of microstates.

Linear Congruential Generator (LCG)

- LCG - Lehmer (1951)

$$\begin{aligned} R_0 &\in (0, m-1); \quad R_{i+1} = a\, R_i + b \pmod{m} \\ a &= \text{generator or multiplier}; \quad b = \text{increment} \\ m &= \text{modulus}; \quad \xi = R_i/m \end{aligned}$$

- In the above $\{R_i : i = 0, 1, \cdots\}$, a, b, and m are integers between 0 and $m-1$;
- ξ is real; $0 \leq \xi < 1$; note: $(m-1)/m$ is nearly unity for m large.
- a, b and m are to be chosen properly for 'good' random numbers.
- the period is m;
- Thumb-rule for linear congruential recursion : in any single simulation do not use more than $\sqrt{\text{period}}$ of the random numbers.

- Minimal conditions on a, b and m
  - m and b should be relatively prime to each other
  - $a \equiv 1 \pmod{p}\ \forall$ prime factor p of m
  - $a \equiv 1 \pmod{4}$ if $m \equiv 0 \bmod 4$
- The above conditions ensure full period of integers
- m is usually chosen as $2^{t-1} - 1$ in a t-bit machine; *i.e.* t is the number of bits used to store an integer. Note: one of the t bits is used to store the sign of the integer.
- Some good RNG
  - Ahren generator :

$$a = 2^{t-2}\left(\frac{\sqrt{5}-1}{2}\right); b = 0$$

  - $a = 7^5 = 16801$, $b = 0$, $m = 2^{31} - 1$ for a 32 bit machine.

- Determinism, unpredictability, and randomness
- determinism does not necessarily imply predictability :
  - differential equations - first order in time :

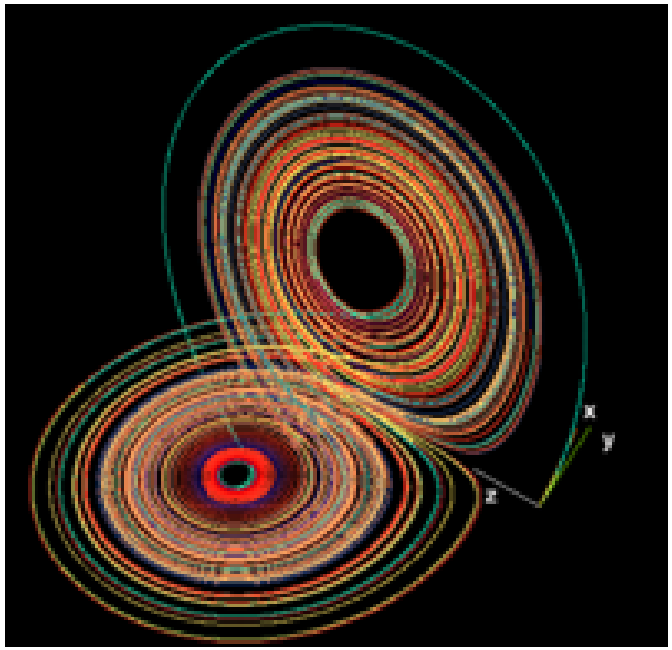
$$\frac{dx_i}{dt} = f_i(x_1, x_2, \cdots); \quad i = 1, 2, \cdots, \ n$$

- $|x(t)\rangle = (x_1(t), \ x_2(t), \ \cdots, \ x_n(t))'$ is an n - dimensional vector evolving in time;
- $|x_0\rangle = |x(t = 0)\rangle$ denotes the initial condition; *i.e.* the initial vector
- the solution of the differential equation is a trajectory in the n dimensional phase space, starting from $|x_0\rangle$.

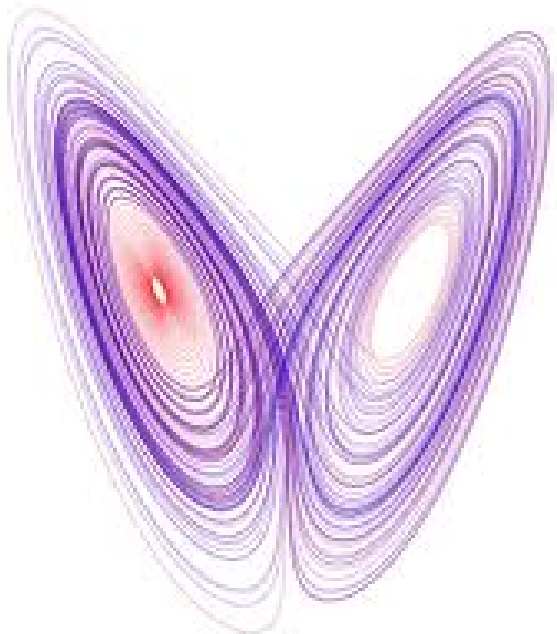
- consider two trajectories starting from initial points infinitesimally close to each other;
- let $d(t)$ denote the Euclidean distance between the two trajectories, as a function of time.
- if $d(t)$ is linear in time, we say the (deterministic) dynamics is predictable
- if $d(t)$ diverges exponentially, the predictability is lost; the dynamics is chaotic : deterministic non-linear dynamics sensitive initial conditions
- thus determinism does not imply predictability

- a random number generator employs nonlinear dynamical rule
- we test the sequence for its randomness ;
  - A test consists of constructing a function $\psi(r_1,\ r_2,\ \cdots)$ where $r_1,\ r_2,\ \cdots$ are independent random variables.
  - Calculate ψ for a sequence of numbers generated by your pet RNG
  - find what is the value that ψ is expected to have if $r_1,\ r_2,\ \cdots$ were to be truly random numbers.
  - the difference will give an idea how good is your RNG
- randomness tests can never be exhaustive
- there is an underlying order - strange order - in chaos;
- Marsaglia's planes in linear congruential random number generators
- the presence of such an inner structure in the sequence can interfere with the phenomenon you are investigating

Lorentz attractor - 1



Lorentz attractor - 2



- Let $\{x_1,\ x_2,\ \cdots\}$ denote a scalar time series obtained from a nonlinear dynamical equation exhibiting chaos
- $x_i\ :\ i = 1,\ 2,\ \cdots$ is stochastic;
- it would resemble noise; or in other words, you can not distinguish the time series from noise.
- embed the time series in a m dimensional phase space by constructing L-delay vectors :
- $\{\vec{a_i} = (x_i,\ x_{i+L},\ x_{i+2L},\ \cdots,\ x_{i+(m-1)L})'\ : i = 1,\ 2,\ \cdots$
- for a proper choice of embedding dimension and delay time, the sequence of vectors describe the dynamics of the system under study.

Parallelisation Methods

- Two parallelisation methods
- Method I : assigns different random number generators to different processors
- Method II : assigns different substreams of one large RNG to different processors
- Dangers of Method I
  - correlations between different RNGs employed
  - this is true even when we employ the same RNG but with different parameters
  - We need to investigate this problem, before we implement Method I
  - there is one family of RNG where theoretical support is available : Explicit-inversive congruential generator

- METHOD II is better, though there is still a risk
  - There are two variants to Method II
  - Method II(a) : leap-frog : Let $\{\xi_0, \xi_1, \xi_2, \cdots\}$ denote a sequence of random numbers generator by an RNG. If there are say 5 processors, the substreams are formed as below : processor 1 : $(\xi_0, \xi_5, \xi_{10}, \cdots$
    processor 2 : $(\xi_1, \xi_6, \xi_{11}, \cdots$
    processor 3 : $(\xi_2, \xi_7, \xi_{12}, \cdots$
    processor 4 : $(\xi_3, \xi_8, \xi_{13}, \cdots$
    processor 5 : $(\xi_4, \xi_9, \xi_{14}, \cdots$
  - Method II(b) : Splitting
  - consider again five processors
  - partition the original sequence into 5 long consecutive blocks
  - each block is assigned to a processors
  - nothing much is known about the correlations between disjoint substreams of consecutive random numbers
  - For sure, this subject is a dangerous territory

- We need to check the randomness of each substream fed into the processors
- we need to check if there are cross correlations
- we need to investigate how the hidden order propagates through parallelization and how does it affect the results of the simulation
- these are open and challenging problems
- and **THANKS**