VortexRNG

Random Number Generation for Computer Simulations

Random number have been used in computer simulations for about as long as people have been doing computer simulations. When a simulation is run using the same set of conditions it will generate the same results every time it is run. Using random numbers is an easy way to introduce variability into a simulation which is vital for the simulation to represent real-world situations.

For example, a simulation could be run to model crop production to determine how far into the future a region can sustain its population. Without getting too deep into the details, consider three levels of variability that would be required for accurate results from this simulation:

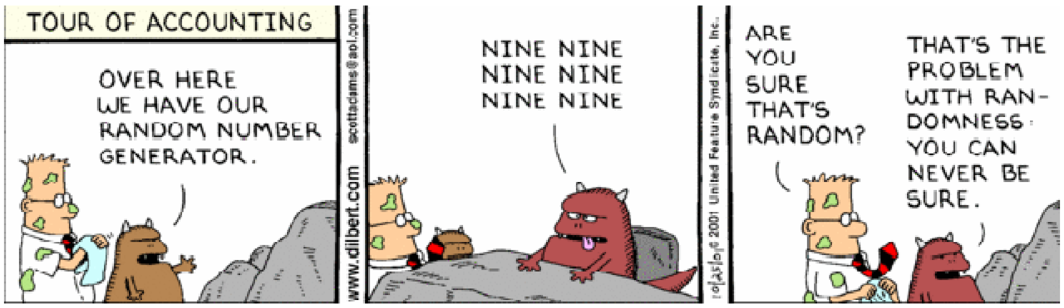
- Initial conditions – these are high level macro parameters such as the amount of cropland in production, population growth rates and so on that won’t change drastically over the course of a year.
- Coarse adjustments – these parameters could be seasonal weather patterns, impact of plant diseases or insect infestations and such that would not change significantly even at a monthly level.
- Fine adjustments – these parameters would represent short term variations such weekly or even daily changes in weather conditions.

Nearly all natural processes have some degree of variability, some of it random and some of it due to conditions that influence the process. All of this variability must be taken into account in a simulation for the results to be representative of the real world.

Random numbers are used to derive all of this variability in a simulation and the quality of those numbers can have a big impact on the quality of the simulation results.

Random Numbers 101

When used in regards to simulations, “random number” is really a misnomer. In simulations and nearly all applications that use random numbers what is important is that a sequence of numbers isn’t predictable. In a security application if someone can predict the next number based on the previous numbers then there is no security. In simulations if you can predict the number sequence then there will likely be unwanted patterns in the simulation results. To work with truly random numbers you have to be ale to tolerate sequences of predictable numbers. Truly random is not what we’re after for simulations. Dilbert’s take on random numbers is spot-on.


TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
scottadams@aol.com
www.dilbert.com
NINE NINE NINE NINE NINE NINE
© 2001 United Feature Syndicate, Inc.
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.

© United Feature Syndicate INC./Dist. By PIB Copenhagen 2010

For simulation work there are characteristics of the random numbers that are far more important than being truly random. In his paper "Random Number Generators for Parallel Computers" (1997), Paul Coddington at the Northeast Parallel Architecture Center at Syracuse University discussed nine important characteristics of an ideal random number generator. The four main characteristics to consider for simulations are:

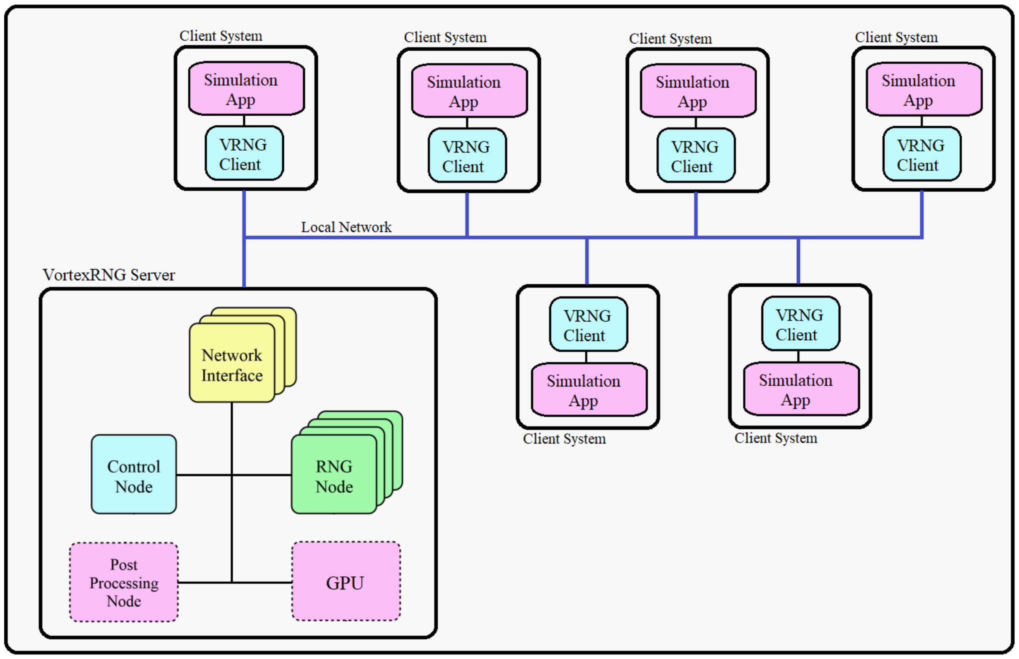
- The numbers should be uniformly distributed across their range. If you need 5,000 numbers between 10 and 100 you don't want 4,000 of them to be between 90 and 100 and the rest of them between 10 and 90. In the crop yield simulation a grossly uneven distribution would likely result in crop failure due to extremely cold/hot temperatures or drought/flooding. If the numbers are always in the sweet spot of ideal conditions then the simulation results would resemble a paradise that doesn't exist anywhere.
- The number sequence should never repeat itself. In a simulation a repeating sequence is essentially the same as not having any variation in the conditions. In the crop yield simulation a repeating sequence would generate cyclical patterns in the results that don't reflect reality.
- The number generation generally needs to be fast. This isn't as important for establishing initial conditions and for coarse adjustments. It is vitally important for the fine adjustments. Consider a simulation to analyze turbulent airflow over an aircraft wing. This could involve hundreds of thousands to millions of data points and each of them could require calculations with a degree of variability at every pass of the simulation. Depending on the complexity of the calculations, generating a random number may take more time than performing the primary calculations for the simulation.
- The numbers need to be deterministic. For any given seed value the same sequence of numbers must always be generated. The first reason for this is so that during development of the simulation model the developer knows that any significant change in the results isn't because of a change in the number sequence. The second reason is so multiple people working on the same project all get the same results for a simulation run. This is critical for peer-reviewed work where others must get the same simulation results as the researcher.

This last point brings us to the different categories of random number generator (RNG). Broadly speaking there are three categories of RNGs (and a multitude of types of RNGs within each category):

- True RNG (TRNG) - A "true RNG" implies a hardware based generator. The gold standard for a TRNG is based on measuring gamma rays produced by a decaying radio-isotope (typically the time interval between bursts of gamma rays). Besides being radioactive this RNG is rather sizable, slow and expensive. In recent years TRNGs have been implemented on computer plug-in circuit boards but they mainly address the radioactivity and physical size issues. A TRNG does satisfy the requirements of never repeating and they generally have a reasonably good uniformity of distribution. Their main downfall for simulation work is they are not deterministic and historically they have not been fast (hundreds to a few thousand numbers per second).
- Pseudo RNG (PRNG) - It would be hard to make a formal definition of a PRNG other than to say it is not a true RNG. A PRNG is more defined by certain characteristics that have been attributed to software based RNGs. A PRNG is generally considered fairly simple, generates number sequences with a poor level of "randomness", has questionable uniformity of distribution and has a short period, meaning it is likely to repeat before you want it to. The positive aspects of a PRNG reflect their simplicity - they tend to be fast and only require a small amount of memory.
- Deterministic RNG (DRNG) - In many ways a DRNG can be considered a PRNG on steroids. A DRNG is a pseudo-random generator, it is not a true RNG. The distinguishing features of a DRNG are due to the algorithm being designed to overcome the weaknesses of a typical PRNG. A good DRNG has good to excellent "randomness" and uniformity of distribution and it has a large amount of state data which gives it a long period before repeating. Good DRNGs typically incorporate multiple passes through a combination of logical and arithmetic operations to achieve these results so the penalty for improved performance is typically paid in slower operation.

Why VortexRNG?

The VortexRNG DRNG is different than other DRNGs commonly used for simulations. It was developed with the intent to integrate it into a random number server instead of being integrated into the simulation code. The benefit to the user for using a random number server is number generation is offloaded from the computing resources running the simulation so simulations run faster, users get their results sooner and wait queues to run simulations are shorter. The diagram below shows an example installation for the VortexRNG Server. It is running on a dedicated system and generating random numbers for a number of client systems on the local network. These clients could each be running different simulations or each one could be a processing node in a massively parallel simulation. Each client is assigned it's own virtual DRNG that is completely independent of the other virtual DRNGs.



An Example VortexRNG Server Installation

The actual performance performance benefit of off-loading number generation depends largely on the nature of the simulation. The crop yield simulation has by far fewer data points and a much coarser time scale than the aircraft wing simulation - on the order of 100,000 data points analyzed on a daily or weekly scale for several years for the crop simulation verses on the order of a million data points analyzed on a millisecond scale for several seconds for the aircraft wing simulation. The crop yield simulation could still benefit from offloading the number generation since the simulations could be run thousands of times because of the number of variable parameters that need to be evaluated. Also, if a system is concurrently running several simulations like this the combined benefit could be substantial.

Offloading number generation has a number of other benefits. It means little emphasis needs to be placed on minimizing memory usage in the generation algorithm so memory can be more freely used to improve the quality of the generated numbers and to reduce the probability of a repeating number sequence (in this regard using more memory is better than using less memory). The VortexRNG number generation algorithm was designed with the intent to implement it in hardware in an FPGA for an order of magnitude or better increase in number generation rates. This makes the core number generation loop run very quickly on an X86 processor. Generation rates of 25 to 40 million numbers per second have been seen on a single core of a commodity-level 64-bit X86 processor. An important side benefit of this is the algorithm runs very efficiently on an NVIDIA GPU with 32 random numbers being simultaneously generated by a single CUDA block instead of numbers being generated one at a time on an X86 processor. Generation rates on the order of one billion numbers per second has been achieved with 4 CUDA blocks in parallel on a consumer graphics GPU (using 1/6 of the CUDA blocks on that particular GPU).

VortexRNG and the Competition

The VortexRNG DRNG compares very favorably in most ways to the popular Mersenne Twister DRNG. The results on common tests of “randomness” are indistinguishable. The primary difference between the two DRNGs is the Mersenne Twister state data is essentially an array of the previously generated numbers, thus the chances of a repeating string is simply based on the amount of state data. The entire contents of the state data memory has to match what it was at a previous point in the number generation in order to generate a repeating pattern. The amount of state data is comparable for both DRNGs, the probability for either of them generating a repeating sequence of numbers is 1 in a number many orders of magnitude larger than the age of the universe in seconds. The VortexRNG DRNG has a configuration option to use two orders of magnitude more state data. That takes the probability of a repeat sequence to over 1 in $2^{1,000,000}$. More importantly, the VortexRNG algorithm is designed to avoid recurring patterns in the state data in order to prevent repeating strings being generated. Using proprietary techniques, several large sections of the VortexRNG DRNG state data are managed to prevent this from happening by introducing variability into the way the state data is generated (in a repeatable way so the results are deterministic).

Users of the VortexRNG Server aren’t forced to use the VortexRNG DRNG. Mersenne Twister is offered as an alternative DRNG for customers who wish to use it. This provides for backwards compatibility with prior simulation work and for compatibility with users that don’t use the VortexRNG Server. Additional RNGs can be supported based on customer interest. Other RNG algorithms are supported through dynamically linked libraries. In this model of operation, the user provides the RNG algorithm code through a library that is loaded when the VortexRNG Server starts up. This is a flexible and fairly easy way to allow the use of any RNG algorithm that has source code in the public domain.

Unlike the VortexRNG DRNG algorithm, the Mersenne Twister algorithm is not a good candidate for acceleration. It would not see a significant performance improvement in FPGA hardware due to the sequential nature of its number generation loops. It would perform much worse than the VortexRNG DRNG in a GPU implementation. GPUs are based on a Single Instruction Multiple Data architecture. The NVIDIA GPUs have 32 execution units or threads operating in parallel in what is called a CUDA block. These threads simultaneously execute the same instruction but each thread has its own set of data that it is operating on. Mersenne Twister makes heavy use of data that would be stored in the main GPU memory so every time the code accesses main memory the execution becomes serialized, waiting for 32 transfers between the CUDA block and main memory to complete before the parallel code execution resumes. The Mersenne Twister algorithm reads a value from main memory, does several fairly simple manipulations on it and writes the result back to main memory. The time it would take to do the 32 reads and 32 writes would be much longer than the time it takes to do the manipulations. There have been implementations of Mersenne Twister in FPGAs and for GPUs but they don’t produce the same number sequences as the commonly used software versions so they are effectively different RNGs.

Another advantage of the VortexRNG DRNG is that Mersenn Twister and the other DRNGs commonly used for simulations only operate in deterministic mode. If a simulation model developer wants to test the robustness of their model using non-deterministic random numbers their best options are to use the DRNG with different seed values for each simulation run or use the operating system's random number generator (which is more likely to be optimized for cryptographically secure applications than for the characteristics important for use in simulations). By changing a single configuration bit in the client software the VortexRNG DRNG can be put into random generation mode. This mode offers the same level of performance and test scores as deterministic mode. By incorporating various system performance metrics in the number generation algorithm the result is essentially a true RNG without any of the drawbacks of a hardware RNG.

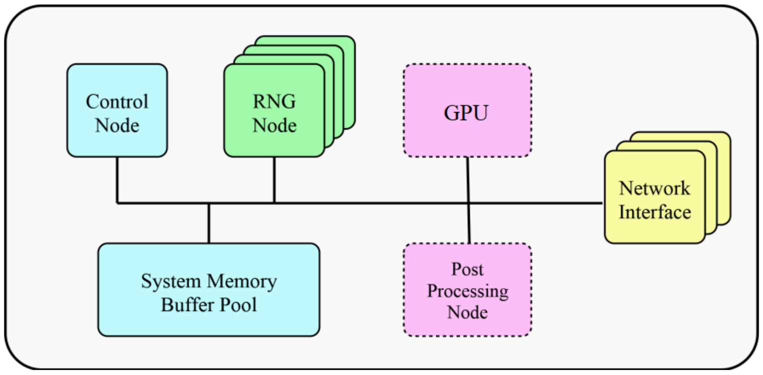
Lastly, it can be hard to ignore that the DRNGs commonly used for simulations are free while the VortexRNG Server must be purchased. However, as with many "free" things the hidden costs can be substantial. Months of research and model development can be wasted because the simulations don't produce the expected results due to low quality random numbers. The VortexRNG Server is also much more than just a DRNG. As noted there are substantial benefits to offloading the DRNG from the primary computing resources including increased productivity from the users. The VortexRNG Server also offers GPU acceleration for very high number generation rates without compromising the quality of the generated numbers or creating incompatibility between the software version and the accelerated version. The VortexRNG Server is offered in a number of versions in configurations suitable from small departmental use to large data center and supercomputer applications. For the non-GPU versions, at the maximum number of streams the cost is well under $50 per stream (a stream is essentially a user but a user can use multiple streams). Several of these configurations are priced below common discretionary spending limits for a university professor or mid-level manager in a corporate environment.

VortexRNG

VortexRNG Server ™

The VortexRNG Server is an innovative software package for researchers and scientists who require a large volume of high quality, deterministic random numbers in their simulation work. The VortexRNG Server off-loads random number generation from the compute resources running simulations for faster simulations and quicker results. The VortexRNG Server turns a 64-bit X86 based Linux computer into a high-performance, high quality random number generator that can be shared between many users.

The VortexRNG Server utilizes multiple CPU cores as RNG nodes as shown in the block diagram below. As well as being highly scalable (it can run on computers from laptop to supercomputer), the VortexRNG Server is also highly configurable. Options are provided for prioritizing users, dedicating resources for increased generation rates or sharing resources when run on a system used for other workloads and more.



RNG Features

- Supports two DRNGs, the popular Mersenne Twister DRNG and the proprietary VortexRNG DRNG.
- The Mersenne Twister DRNG provides compatibility with previous simulation work and with others not using the VortexRNG Server. Additional RNGs are supported through dynamically linked libraries.
- The VortexRNG DRNG is a very fast and strong DRNG. Generation rates in excess of 20 million numbers per-second per-core can be achieved with a single core on a 64-bit X86 processor. Performance and the quality of numbers generated are comparable to commonly used DRNGs.

VortexRNG Server

- Offloads random number generation from primary compute resources so simulations run faster.
- Multi-threaded architecture takes advantage of multi-core processors and multiple processors in a system.
- A C source code driver is provided for client use and easy implementation on a variety of host microprocessors.
- Multiple options for increasing number generation rates such as grouping streams for parallel generation.
- Post processing options for floating point results and specific distributions.
- Expected to run on any Linux distribution (tested on Ubuntu and CentOS, others to follow).
- Multiple product offerings suitable for running on systems ranging from laptop to supercomputer.

GPU Acceleration

- The VortexRNG DRNG can take advantage of an NVIDIA GPU for significantly higher number generation rates (over one billion numbers per second with a mid-range graphics oriented GPU)
- Supports NVIDIA architectures starting with Pascal (compute capabilities 6.0 or higher). Even the older architectures provide a significant performance boost over running on an X86 processor.
- Supports multiple GPUs and a mix of GPU types, each individually configurable. Provides an easy way for multiple users to share an expensive resource.