RUTGERS, THE STATE UNIVERSITY OF NEW JERSEY

DATA STRUCTURES & ALGORITHMS TERM PAPER

Monte Carlo Methods for Large Scale Circuit Simulation

Elie Rosen

June 1, 2014

Contents

Abstract

This paper will discuss the Gibbs Sampling method for pseudo-random, fully conditional, parameter selection for multiple-run circuit analysis. A Monte Carlo method is a means to strategically select values for simulation such that the results cover the entire range of values for which contain interesting data while also minimizing the runs necessary to attain this data from a typically time consuming process. We will conclude that that the impact of such simulations saves engineers not only time but also companies hundreds of thousands of dollars compared to other distributed methods.

1 Introduction

In the field of microelectronics where digital circuit transistor counts reach into the billions and high frequency analog circuits face effects of higher order harmonic distortion, the ability to fully test a circuits performance under a multitude of conditions has become ever more important. The inherent problem of these circuits is that due to the exponentially large number of possible parameter variations and test cases, it has become impossible in modern times to test every possible parameter value or test case. This issue was originally combated by selecting values pseudo-randomly over a range to get a large enough sample size of simulation results, however this is inefficient in that values which may have an important significance may be skipped over or not all of the important test cases can be covered. Another issue is the time required to complete these simulations, a trade off is formed where either more simulated trials can be ran versus the incremental time added for each additional test.

So how does one solve these complex and challenging issues that modern circuit designers face every day? The answer, Monte Carlo methods for pseudo-random sampling! A Monte Carlo algorithm for circuit analysis is a method that uses probability mixed with random selection to provide a series of parameter values for simulation of which will typically have a more significant impact for following simulation runs due to its slight dependence on the previous chain of iterations.

Monte Carlo methods typically break down into a similar pattern depending on the application:

1. Define a domain of possible inputs.
2. Generate inputs randomly from a probability distribution of the domain of inputs.
3. Perform a deterministic calculation on the distributed inputs
4. Finally, aggregate the results.

The most commonly used example to show how Monte Carlo works is to estimate π by throwing points onto a unit square that contains a unit circle within. Next, we mark randomly selected points into the square where the number of points used can be increased to achieve a better estimation. Then we count the number of marks within the circle and we take the ratio of this to the total number of points selected. Since the area of square compared to the area of a circle is known to be $\frac{\pi}{4}$ we can multiply this ratio by 4 to get an estimate of π. Only a few thousand points are needed to get a probabilistically close to exact estimation of π. This can be seen in Figure 1.

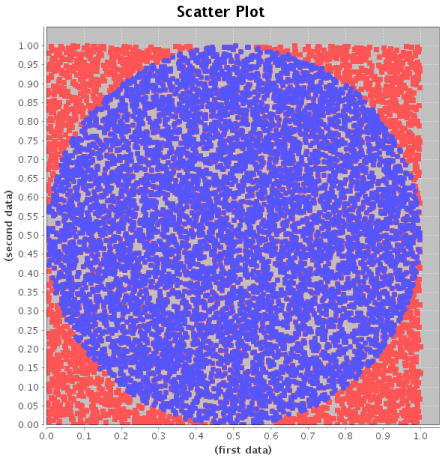

Figure 1: Monte Carlo point selection for the estimation of π

While this is a process that can be done in mere milliseconds on modern computers, the domain of circuit analysis contains many magnitudes of orders of parameters more compared to this simple example. Therefore, we need to also use methods to probabilistically limit the playing field for the device under test. One simple way of doing this is known as Gibbs Sampling where we can set bounds to each of the parameters which can always be known since the scope of values required to be tested usually only contain a limited range of useful values.

2 Definitions

2.1 Markov Chains

Gibbs Sampling is a Markov Chain Monte Carlo algorithm, a Markov Chain is a stochastic process (a consecutive set of non-deterministic quantities defined for some known set θ) where future states of a chain are independent of past states given its current state. With this algorithm, it is possible to select values of interest across a distribution, θ that will probabilistically ensure that almost all values of interest can be included across all of the simulations to be processed. To satisfy the Markov property we can consider $\theta^{(t)}$ as the set of parameters at time t, the next set of parameters, $\theta^{(t+1)}$, will only rely on $\theta^{(t)}$ and not any of the other previous parameters. This can be shown in the equation below.

$$p(\theta^{t+1}|\theta^{(1)},\theta^{(2)},\ldots,\theta^{(t)} = p(\theta^{(t+1)}|\theta^{(t)})$$

Therefore, the Markov chain is built upon the parameter space of θ which is only slightly related to the previous state. The chain at its current state only remembers where it was at the last time period.

2.2 Transition Kernel

Now that the Markov chain has been defined, we need to express how new parameters in θ are selected for subsequent simulations. This is referred to as the transition kernel, which is a system for detailing the probability of moving to another state based off of the current state of the parameters, θ. For a discrete state space which contains k possible states where k is defined as the number of parameters required for simulation, a kernel will be a kxk matrix of transition probabilities. For example if $k = 3$ the transition matrix would look like Figure 2.

$p(\theta_A^{(t+1)}\|\theta_A^{(t)})$	$p(\theta_B^{(t+1)}\|\theta_A^{(t)})$	$p(\theta_C^{(t+1)}\|\theta_A^{(t)})$
$p(\theta_A^{(t+1)}\|\theta_B^{(t)})$	$p(\theta_B^{(t+1)}\|\theta_B^{(t)})$	$p(\theta_C^{(t+1)}\|\theta_B^{(t)})$
$p(\theta_A^{(t+1)}\|\theta_C^{(t)})$	$p(\theta_B^{(t+1)}\|\theta_C^{(t)})$	$p(\theta_C^{(t+1)}\|\theta_C^{(t)})$

Figure 2: Table of possible kernel transitions when the number of parameters to choose from is 3.

The rows of this matrix can be summed to give a conditional probability mass function (PMF) based on the current state, while the columns are the marginal probabilities of being in a specific state at time $t+1$.

3 Building a Markov Chain

To bring this all together, a Markov Chain is built using the following:

1. First, an initial distribution must be chosen known as $\prod^{(0)}$, this is a $1xk$ vector of probabilities that can be summed to one. For some applications, a reasonable assumption may be made for these values however as will be shown in the next section, the selection of arbitrary values will work as well.

2. After the first iteration, $\prod^{(1)}$ can be found from the following equation, $\prod^{(1)} = \prod^{(0)} \times \mathrm{P}$, where P is the probability matrix discovered from the initial conditions.

3. At time t, the distribution $\prod^{(} t)$ will take the form of: $\prod^{(t)} = \prod^{(t-1)} \times \mathrm{P} = \prod^{(0)} \times \mathrm{P}^t$.

This gives a definition for the structure of the Markov chain at time t throughout the simulation.

3.1 Convergence and Burn-in

Due to a feature of Bayesian statistics, Markov chains tend to converge to a particular distribution of parameters regardless of its starting point. As a result, given a range of parameters in a set y, it is possible to find a Markov chain which will have a posterior distribution of $p(\theta|y)$. By running the simulation process on this Markov chain, the parameters selected from this chain will become approximately $p(\theta|y)$ when the chain hits its convergence point.

Since the first few results may not be as close to the convergence point as required for a particular application, a practice known as burn-in is used to help diminish the original effects of the initial conditions of the Markov chain. By removing or throwing out the parameter results from a specified number of iterations, this makes the current chain less dependent from the initial condition and chances are that the future results will become much closer to the convergence point faster.

Overall, Monte Carlo Markov chain allows for the simulation of possible parameter sets that are slightly dependent from previous iterations that will approximate a probability distribution of the given parameters. From this, it is possible to calculate parameter values of interest to optimize the time required to fully simulate a circuit. The Gibbs Sampler as a method for calculating these values of interest.

4 Gibbs Sampling

Gibbs Sampling correlates well to the process of circuit analysis. The sampler works by using Monte Carlo Markov chains on the condition that the range of each parameter is known. This is very attributable to circuits since in a typical analysis, there will be a range of values for which components will be useful to the circuit under test. With

a joint distribution $p(\theta_1,\ldots,\theta_k)$ of which contains samples to calculate parameter values, along with the fact that we have the full conditional distribution for all of the parameters, $p(\theta_j,\theta_{-j},y)$. We can find parameter samples by the following steps:

1. Suppose we want to sample values from $p(\theta|y)$, where θ is a vector of k parameters.
2. We find initial values for the parameters and store them in $\theta^{(0)}$.
3. Then we pick any parameter θ_n where n is a randomly decided value and we take a value of $\theta_n^{(1)}$ from the full conditional $p(\theta_n|\theta_1^{(0)},\ldots,\theta_k^{(0)},y)$.
4. Next, we select a new random parameter that has not be chosen yet, $\theta_{n'}$ from the full conditional except now we include the result from the previous iteration $p(\theta_{n'}^{(1)}|\theta_1^{(0)},\ldots,\theta_n^{(1)},\ldots,\theta_k^{(0)},y)$.
5. This is done for k iterations until all parameters have been updated at least once based off of the results from the other changes in the previous iterations.
6. Repeat this process M times where each t will provide a vector $\theta^{(t)}$.
7. Finally, we adjust for burn-in to make the results better.

The result is a Markov chain which contains a series of samples from θ that are close to converging to $p(\theta|y)$. Using a method such as Monte Carlo integration can be used to best estimate the final parameter values of interest.

4.1 Sample Code

The code listed below shows how a typical Gibbs Sampler may work, as can be seen it follow the process listed from the section before to return a series of parameters for use in simulation across each trial.

```
gibbs(n.sims, beta.start, alpha, gamma, delta,  y, t, burnin, thin) {
        beta.draws <- c()
        lambda.draws <- matrix(NA, nrow = n.sims, ncol = length(y))
        beta.cur <- beta.start
        lambda.update <- function(alpha, beta, y, t)    {
                rgamma(length(y), y + alpha, t + beta)
        }
        beta.update <- function(alpha, gamma, delta, lambda, y) {
                rgamma(1, length(y) * alpha + gamma, delta + sum(lambda))
        }
        for (i in 1:n.sims)  {
                lambda.cur <- lambda.update(alpha, beta.cur, y, t)
                beta.cur <- beta.update(alpha, gamma, delta, lambda.cur, y)
                if (i > burnin & (i - burnin))  {
                        lambda.draws[(i - burnin)/thin, ] <- lambda.cur
                        beta.draws[(i - burnin)/thin] <- beta.cur
                }
        }
        return(list(lambda.draws, beta.draws))
}
```

5 Analysis

There are a few ways to look at the algorithmic complexity of utilizing Gibbs Sampling versus the traditional approach of just selecting random parameter values for simulation. The sampling process is recursive throughout the building of the Markov chain, this process becomes exponential as the number of parameters required for simulation increases. However, since modern computers can preform these calculations very quickly, this time has almost no effect to the efficiency of the algorithm. What we really care about is how long it takes for the sampler to converge to a specific value over the traditional method.

Most often in circuit analysis, the goal is to minimize some characterization of a design. For example, a company may specify that a particular circuit may not exceed a specific maximum power consumption. Where Gibbs offers the most is in it's ability to meet given specifications while also minimizing the number of simulations required to come to convergence earlier than other methods. Figure 3 below shows the typical steps Gibbs sampling uses to converge to values which meet the specified criteria.

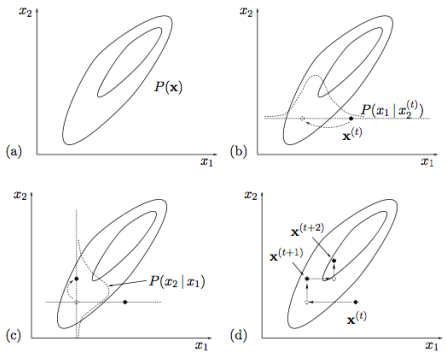

Figure 3: The entire probability space is shown as P(x) in a. Graph D shows the steps taken at each iteration to move into the area of distributed probably for which the parameters meet the specification.

Most often a single circuit simulation can take up to a few minutes or more. When you add into account the fact that the simulation will need to be ran many multiple times, it becomes clear that the entire simulation process can add up to take even days. Correct utilization of Gibbs Sampling will always minimize the number of simulations required by its property of convergence. This is far superior to the old practice of guess and check and randomly picking values for testing of which may never give a good result for weeks at a time. Figure 4 shows results from a Monte Carlo simulation from the popular circuit analysis software, Cadence.

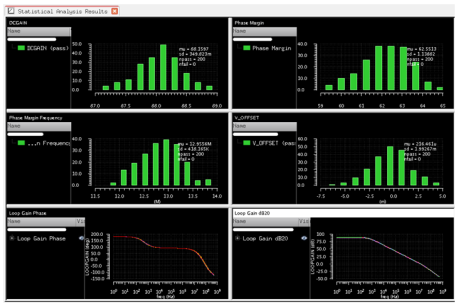

Figure 4: Various plots of Monte Carlo circuit simulations

6 Application Usage

While there are a vast number of possible application uses for Monte Carlo methods and Gibbs Sampling in engineering, such as wind energy analysis to robotic localization estimation. This paper focuses specifically on its use in the field of microelectronics and circuit analysis. An unfortunate realization has to be made that time is a precious commodity in the world of engineering and it is with this realization that it is just not possible to be able to create and test every facet of a design. At the scale of current digital chips, it would literally take longer than the existence of the known universe to fully test every logical switch of every transistor in integrated circuits and this would be for just one IC. So mathematicians have devised a method for testing applications not completely but in a way such that the probability distribution of the values cover enough of the scope of the logic such as to be reasonably sure to a few sigma values that the integrated circuit behaves and operates correctly.

This has recently become a very significant realization to my internship work at Teledyne LeCroy in circuit design and verification. Often times, it is required to run simulation that may last a whole week just to find the precise values for operating at the high frequencies of LeCroy's product line. Here we have a real world situation where the time it takes for simulation can have a huge impact to the total cost of design. Simulation software licenses can cost as much as $100,000 per user and when a simulation is running, no one else is allowed to operate on that license until it's done. This requires a company to have to maintain multiple expensive licenses at a time so that engineers can continue other work in the meantime. Other factors to consider include the near superclusters used to run these simulations often operating with many processing cores and tens of gigabytes of ram all just to speed up the simulations to get results faster.

Monte Carlo methods save time by minimizing the number of simulation iterations required to get accurate results. Without this, companies operating at the bleeding edge of technology would not be able to deliver products in any form of reasonable time.

7 Related Work

The field of Monte Carlo research is vast, with high rewards for even slight improvements in the time required to hit convergence. Two particular works stand out when one cares to best minimize the time required to analyze a circuit. That is in "Efficient Approach for Monte Carlo Simulation Experiments ant Its Applications to Circuit Systems Design" and "Using Lower and Upper Bounds to Increase the Computing Accuracy of Monte Carlo Method" . In the first paper, the authors attempt to efficiently solve for component values to be used for a double T filter circuit. They attempt a few methods for Monte Carlo pseudo-random sampling and conclude that their methods are able to come to convergence with a 92% time savings and an accuracy of 60% in the values selected. For them, this was a significant goal due to the available computing resources and time available to them. They also conclude that if the circuit contained more possibilities for variation, that they would be able to improve these results as well.

The second paper was based off of the example originally described in the introduction for estimating the value of π. Here, the authors propose adding bounds to the parameters in their Markov chain to achieve better accuracy in their estimation. They show that this method which is also known as Gibbs Sampling provides a much better accuracy due to the smaller scope of the allowed values in a distribution. They conclude that the bounded method can be repeated many times to provide additional time savings.

8 Conclusion

In this paper, it is shown that the use of Monte Carlo methods for circuit simulation has become an invaluable tool for engineers in the field of microelectronics. Where with many of today's circuits it would be physically impossible to test every possible variation of parameters in a given circuit due to the time it would take to simulate. It has been shown that the use of the Gibbs Sampling method for pseudo-random, fully conditional, parameter selection, that it is now possible to get relatively accurate simulation data hundreds of times faster due to its slight dependance on previous simulation results compared to just random sampling. This shows that engineers now have tools at their fingertips for which allow rapid prototyping of circuits that will save companies large sums of money in engineering time costs and licensing that would otherwise be prohibitive to continue their work.

9 References

1. Chun-Hung Chen; Donohue, K.; Jianwu Lin; Yucesan, E., "Efficient approach for Monte Carlo simulation experiments and its applications to circuit systems design," Simulation Symposium, 2001. Proceedings. 34th Annual , vol., no., pp.65,71, 2001 doi: 10.1109/SIMSYM.2001.922116

2. Jianwen Chen; Feng, L., "Using Lower and Upper Bounds to Increase the Computing Accuracy of Monte Carlo Method," Computational and Information Sciences (ICCIS), 2010 International Conference on , vol., no., pp.630,633, 17-19 Dec. 2010 doi: 10.1109/ICCIS.2010.159

3. http://web.mit.edu/ wingated/www/introductions/mcmc-gibbs-intro.pdf

4. http://www.cisl.columbia.edu/kinget_group/student_projects/montecarlotools/MonteCarloDeviceMismatch.pdf

5. http://pareto.uab.es/mcreel/IDEA2014/MCMC/mcmc.pdf

6. http://mathworld.wolfram.com/MonteCarloIntegration.html

7. http://en.wikipedia.org/wiki/Monte_Carlo_method

8. http://en.wikipedia.org/wiki/Gibbs_sampling