Nuclear Mathematical and Computational Sciences: A Century in Review, A Century Anew
Gatlinburg, Tennessee, April 6-11, 2003, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2003)

CELL BASED RANDOM NUMBER GENERATORS FOR REPRODUCIBLE MONTE CARLO SIMULATIONS

Thomas A. Brunner
Sandia National Laboratories*
PO Box 5800
Albuquerque, NM 87185-1186
tabrunn@sandia.gov

ABSTRACT

A new cell based random number generator approach for running Monte Carlo simulations is presented. Each cell in a finite element type mesh, typical of multi-physics calculations, has an independent random number generator, which ensures that parallel calculations are reproducible for any given processor topology. The cell based generator is in contrast to many current Monte Carlo codes that use an independent random number generator for each particle in the simulation. While the particle-based generators ensure that the results are also reproducible on any number of processors, storing the random number generator state for each particle can use a substantial amount of memory. The new cell-based random number generator approach also ensures reproducible results on any number of processors and can save considerable amounts of memory. The serial performance of the cell-based and particle-based methods are compared.

Key Words: Monte Carlo, Reproducible Parallel Simulations, Random Number Generators

1. INTRODUCTION

In Monte Carlo simulations it is desirable to exactly reproduce the result when the code is run multiple times using the same input; reproducibility is essential to ensure correct algorithms. Exact reproducibility is also needed when using different numbers of processors for the same simulation to guarantee that the parallel algorithms are working correctly [6]. In single processor calculations reproducibility is easily accomplished by assuring that the particles are simulated in the exact same order so that the stream of numbers from the pseudo random number generator (RNG) is used identically between runs. For parallel simulations, reproducibility is typically accomplished by storing the state of the RNG with the particle [2, 3, 5–7]. As the particle moves from processor to processor, the RNG state moves with it, assuring that the same stream of random numbers is used for each particle independently of the processor boundaries. Only extra storage is needed compared to the serial case for this method, and no significant algorithmic changes are needed. For time dependent calculations, it is necessary to store all the particles at the end of each time step, so the extra storage cost associated with the random number state on each particle can become significant.

In multi-physics computer codes where the physical system is typically described on a discrete mesh of some sort, each processor has a collection of cells. The cells on the processor could represents elements for a finite element mesh, for example. If each cell stores an RNG state, reproducibility for any number of processors can also be maintained. Storing the RNG state on the cell reduces the memory requirements, but

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Algorithm 1 Outline of a Single Time Step for the Particle Based RNG Method.

```
 1: for all time steps do
 2:    move census particle list to active particle list
 3:    while there are active particles on any processor do
 4:       while there are active particles on this processor do
 5:          pop a particle from the active list
 6:          get current cell properties
 7:          while particle is active do
 8:             move particle to event
 9:             if event was scatter then
10:                change particle direction
11:             else if  event was absorption or leak  then
12:                delete particle
13:             else if event was cell boundary crossing then
14:                if new cell is on another processor then
15:                   transfer particle to new processor
16:                get new cell properties
17:             else if  event was reached census  then
18:                push onto census list
19:       if other processors sent us particles then
20:          add new particles to active list
```

more bookkeeping must be done in order to assure that the particles are simulated in the exact same order within a cell. The simulation order of the cells is independent.

Details of the new cell based RNG method are presented below, along with performance comparisons to the standard particle based RNG method for the serial case only; no parallel calculations have yet been performed.

2. THE ALGORITHM

The algorithms for my implementations of the standard Monte Carlo method and the new cell based random number method are listed in Algorithm 1 and Algorithm 2, respectively. The algorithms list only the details within a time step. Initialization, tally computation, and other common parts of the code have been omitted from the algorithm listings. Link lists are used to store the particles, and several lists are used to store the particles in various states. In both the particle based RNG and the cell based RNG method, the census lists store particles that have finished the current time step. At the beginning of each time step, the census list is moved to the active particle list. The particles are removed from the active list one at a time and transported through the system until some terminating event occurs.

The particle based RNG method listed in Algorithm 1 is an extension of the serial implementation of a particle Monte Carlo code. The particles store their position, direction, speed, time, and a pointer to the next particle in the list. In the particle based method, the particle also stores the state of the RNG. When a particle crosses a processor boundary, it is sent to the new processor and the simulation continues. The particles can be simulated in any order, which provides great flexibility in deriving efficient parallel

Algorithm 2 Outline of a Single Time Step for the Cell Based RNG Method.

```
 1: for all time steps do
 2:   for all cells do
 3:     move cell census particle list to cell active particle list
 4:   while there are active particles do
 5:     for all cells on this processor do
 6:       get cell properties
 7:       while there are active particles in the cell do
 8:         pop a particle from the cell active list
 9:         while particle is active do
10:           move particle to event
11:           if event was scatter then
12:             change particle direction
13:           else if event was absorption or leak then
14:             delete particle
15:           else if event was cell boundary crossing then
16:             push onto face crossing list
17:           else if event was reached census then
18:             push onto cell census list
19:     transfer particles on processor boundary faces to new processors
20:     for all cells do
21:       for all faces do
22:         move particles from face crossing list to cell active list
```

communication algorithms to move the particles between processors. The parallel transfer steps listed in Algorithm 1 can be modified to improve the parallel efficiency.

The new cell based RNG method listed in Algorithm 2 stores the RNG state with the cell instead of on the processors. There is a cost associated with this memory savings. Particle independence is lost, so all the particles must run in exactly the same order within each cell; extra bookkeeping is needed to ensure the particles are run in the same order. This bookkeeping consists of two lists on each face to temporarily store particles that are crossing the face in the positive and negative directions. Once all particles in the simulation have reached a face, census, or are absorbed, the particles in the face crossing lists are moved back to the cell active lists on Line 22 of Algorithm 2, being careful to move the particles off the faces in a consistent order that is independent of processor topology. Because the particles are already stored on the faces, an efficient parallel communication step is naturally implemented in the algorithm. The particles have been grouped on the faces, making it easy to package the particles in order to send one long message instead of many short messages for each particle. While particle independence has been lost in the cell based RNG method, each cell can now be run independently. Indeed, the main goal of this new algorithm has been to ensure cell independence so that each cell performs the same computations independent of the processor on which it is found.

In this study, only single processor calculations were performed; the parallel communication steps in Algorithms 1 and 2 were not implemented in the actual code. The primary focus of this study is to determine the cost of the extra bookkeeping required for the cell based RNG method.

Table I. Variables used in two methods and their centerings. Variables used only in the particle based RNG method, only in the cell based RNG method, or in both are labeled with *"(particle),"* *"(cell),"* and *"(both),"* respectively.

Particle Values	Cell Values	Face Values
x, y, z *(both)*	σ_a *(both)*	Position *(both)*
$\Omega_x, \Omega_y, \Omega_z$ *(both)*	σ_a *(both)*	Positive List *(cell)*
ν *(both)*	Faces *(both)*	Negative List *(cell)*
t *(both)*	Particle List Head *(both)*	
Next Particle *(both)*	RNG state *(cell)*	
RNG state *(particle)*		

3. MEMORY USAGE

Table I lists the variables used in the two methods and to which entity they belong. The physical variables listed are those for three dimensional simulations, but the analysis below takes into account that fewer variables are needed in one and two dimensions. The theoretical memory usage shown here is close to the minimum amount of memory needed for a particle transport simulation. In the implementations, some extra memory was used in oder to save computational effort. One example of this is keeping track of in which cell a particle is located instead of searching for the current cell each time step in the particle based method. Common parts of the methods were implemented in identical ways so that any extra overhead is incurred by both methods; these identical costs cancel in the discussion below.

Defining f, p, and r to be the amount of memory used by floats, pointers, and the RNG state respectively, we can write down some equations for the memory usage. Typically a float is eight bytes, a pointer four bytes, and the RNG state anywhere between four and two hundred bytes. We will also assume that there are n^d cells in the problem, where n is the number of spacial cells in one dimension and d is the dimensionality of the problem. For a problem with n^d cells, there are $d(n+1)n^{d-1}$ faces. There are P particles in the simulation. With these definitions, the memory usage for the particle-based RNG method, m_p , is

$$m_p = [(2d+2)f + p + r_p]P + [2f + (2d+1)p]n^d + [f]d(n+1)n^{d-1}, \tag{1}$$

where r_p is the RNG state size of the particle based method, and the memory usage for the cell-based RNG method, m_c, is

$$m_c = [(2d+2)f + p]P + [2f + (2d+1)p + r_c]n^d + [f + 2p]d(n+1)n^{d-1}, \tag{2}$$

where r_c is the RNG state size of the cell based method. The memory difference between the two methods is

$$m_p - m_c = Pr_p - n^d r_c - 2dn^{d-1}(n+1)p, \tag{3}$$

where Pr_p is the total memory used by the RNG state in the particle method, $n^d r_c$ is the total memory used by the RNG state int the cell method, and $2d(n+1)n^{d-1}p$ is the memory needed for the extra bookkeeping of the cell method. When $m_p = m_c$, both methods use the same amount of memory; it is informative to find the number of particles per cell at this break even point. Setting $m_p - m_c = 0$ in Eq. 3 yields

$$\frac{P}{n^d} = \frac{r_c}{r_p} + 2d\left(1 + \frac{1}{n}\right)\frac{p}{r_p}. \tag{4}$$

By assuming that n is large, we can approximate Eq. 4 as

$$\frac{P}{n^d} = \frac{r_c}{r_p} + \frac{2dp}{r_p}, \tag{5}$$

which is now independent of the number of cells along a direction, n. The first term, r_c/r_p, in Eq. 5 represents the relative cost of the RNG state sizes in the two methods. The second term, $2dp/r_p$, is the storage needed for the bookkeeping of the cell based RNG method.

Many different random number generators are used. Linear congruential generators (LCGs) can use as few as four bytes of memory for the state. Multiple recursive generators (MRGs) typically use about twenty bytes for the state [1]. More memory will be need for each of these to guarantee that individual instances of the generators will have statistically independent streams, otherwise different particles could have the exact same histories. For some parallel RNG implementations, this cost is relatively modest and can even be zero. For other implementations, such as the SPRNG library [4], over forty bytes are used for the LCG. Linear congruential generators seem to suffice for most particle based RNG simulations. However, more events will be sampled using the same RNG in the cell based method, so the new method may require a higher quality RNG with a longer period for the cells.

Inserting some typical values for LCG state sizes into the simplified Eq. 5 shows that not very many particles per cell are needed for the cell based RNG to save memory. If both methods use an LCG, then $r_c = r_p = 1p$, and we get

$$\frac{P}{n^d} = 1 + 2d = (3, 4, 5), \tag{6}$$

where $(3, 4, 5)$ means that we need three, four, and five particles per cell in one, two, and three dimensions, respectively. If the particle based method uses the standard LCG, $r_p = 1p$, and the cell based method uses the more robust MRG, $r_c = 5p$, we get

$$\frac{P}{n^d} = 5 + 2d = (7, 9, 11). \tag{7}$$

The fraction of memory saved can also be estimated. Dividing the memory saved by the cell based RNG method, Eq. 3, by the amount of memory used in the particle based RNG method, Eq. 1, and assuming that n is large yields

$$m_s = \frac{P'r_p - 2dp - r_c}{P'r_p + ([2d+2]P' + 2 + d)f + (P' + 2d + 1)p}, \tag{8}$$

where $P' = P/n^d$ is the number of particles per cell and m_s is the fraction of memory saved by using the cell based RNG method instead of the particle based RNG method. If we let $P' \to \infty$, then

$$\text{Fraction of memory saved} \approx \frac{r_p}{r_p + [2d+2]f + p}, \tag{9}$$

which is simply the fraction of memory used by the RNG state in the particle. Figure 1 plots Eq. 8 for three cases, and assumes that $p = 4$ and $f = 8$. Using a very lean implementation of the LCG (storing just one integer) for the particle based RNG method, there is limited savings available for the cell based RNG method to realize. However, if there is a modest increase in the storage cost for the RNG, such as there is for the SPRNG LCG implementation, the savings of the cell based method can be substantial.

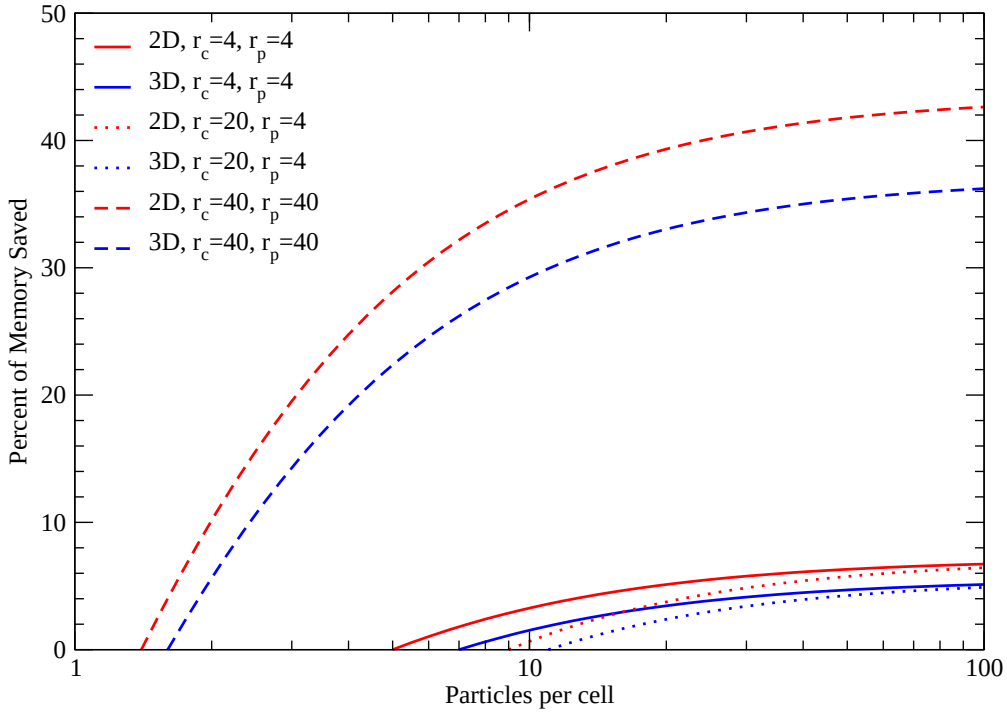

Figure 1. Percent of memory saved by using the cell based RNG method instead of the particle based RNG method using Eq. 8 with $p = 4$ and $f = 8$ and the other variables set as described in legend. Note that the fraction of memory saved as the number of particles goes to infinity is independent of the amount of memory used in the cell based method; the limit is the fraction of memory that the RNG uses in the particle based RNG method compared to all the memory used by the particle.

4. RESULTS

Two codes have been written to implement both the standard method and the new cell based method. Both implementations are meant to be fairly efficient, yet common parts of the code are shared between them, which ensures that run time comparisons are as fair as possible. All the comparisons are for serial only simulations; no parallel simulations have been performed.

The basic test problem is a pulsed source of isotropic particles at the center of the regular Cartesian mesh, which covers the coordinates from -1.1 to 1.1 in each dimension. In order to test the effective cost of these two methods, nearly two hundred simulations were run with the two codes with varying parameters. The number of particles, the scattering cross section, the number of mesh divisions in each dimension, and the simulation end time were all varied in order to test many different types of materials. The parameters used in the simulation are shown in Table II. Nearly all combinations of the parameters shown in Table II were used as inputs for the two codes, with the exception of 501 mesh divisions in three dimensions because these problems did not fit into the computer's memory. The absorption cross section is always set to zero. Only one time step is used to reach both the end times. It is assumed that the same amount of work per time step would be seen for any number of time steps. A linear congruential generator is used in all cases.

Figure 2 shows the histograms of the ratios of the cell based RNG method run times to the particle based RNG method run times for one, two, and three dimensional simulations. Also shown are all results from all

Table II. Parameters used in the test simulations.

Parameter	Values used in simulations
Particles	1.0e5, 1.0e6, 1.0e7
$\sigma_{\text{scattering}}$	0.0, 1.0, 10.0, 100.0
Mesh intervals	11, 101, 501
Simulation end time	1.0, 10.0

Table III. Averages of the run time ratio for cell-based method to particle based method. The filtered averages include only the simulations where the cell based method saved memory per Eq. 4.

	All Simulations			Filtered Simulations		
Series	**Simulations**	**Average**	**Standard Deviation**	**Simulations**	**Average**	**Standard Deviation**
1D	73	1.23	0.24	72	1.22	0.24
2D	73	1.52	0.53	56	1.36	0.22
3D	49	2.79	3.53	32	1.77	1.88
All	193	1.73	1.91	160	1.38	0.88

simulations in a fourth histogram. Most of the ratios are above one, indicating that there is a modest run time performance hit for the new method. In a few cases, the new method was faster than the old method. The new method generally performed best when there were fewer cell boundary crossings compared to other events, such as scattering. However, in a few pure vacuum cases, the cell-based method also performed rather well. This might be explained by the fact that the cell based quantities stayed in the CPU's memory cache for all the particles, while in the particle based method, the particles are retrieving new cell information at each event. The more work there is to do per cell, the better the new method should perform relative to the standard method.

Table III shows the averages of the run time ratios for all simulations. The one and two dimensional simulations had only a modest run time performance hit. The three dimensional simulations show extremely poor performance, but many of these simulations had much fewer than one particle per cell. Not only is this well below the memory break even point given by Eq. 4, but there are not enough particles in the simulation to yield meaningful results. Because of this, Table III also includes a lists the statistics from the simulations with less memory usage for the cell based method per Eq. 4. There is still a large variation in the three dimensional simulations, but the run time penalty of the new method for all filtered simulations is about 1.38 slower than the particle based method.

5. SPECULATIONS ON PARALLEL PERFORMANCE

The work presented here was meant to determine the costs for the extra bookkeeping of the cell based RNG method for serial calculations only. However, it may be useful to speculate on some of the issues that will arise in the parallel implementation.

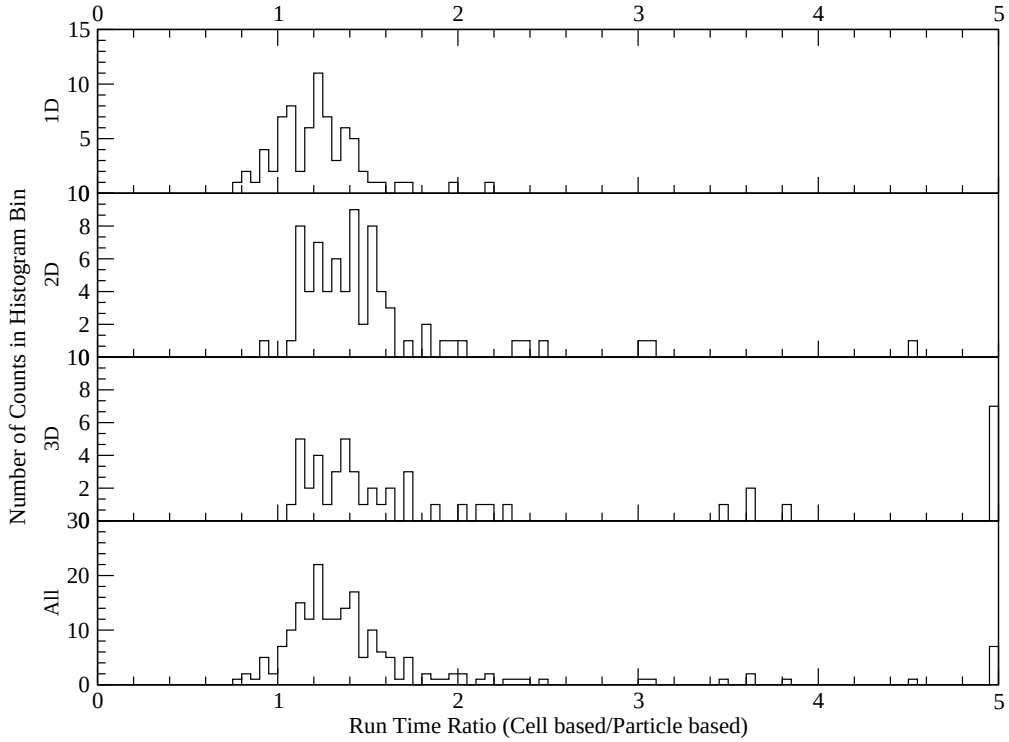

Figure 2. Histograms of the ratio of the cell based method run time to the particle based run time method for one, two, and three dimensional simulations, as well as all simulations tallied together in a fourth histogram. Simulations with a ratio above five have been tallied in the last bin just before five.

A domain decomposed mesh that is load balanced for the deterministic physics of the simulation, such as hydrodynamics, is seldom the best mesh for load balancing Monte Carlo simulations. Both the cell based RNG and the particle based RNG method will suffer the same inefficiencies from the poor load balancing that is forced upon the Monte Carlo simulation. Because particles are independent, it is possible to replicate some of the domains with many particles in the particle-based RNG method, thereby increasing the overall parallel efficiency. This is not possible with the cell-based RNG method; a cell is a unit and cannot be divided between processors; however, the cells on a processor, determined by load balancing issues for other types of physics, can be further subdivided onto more processors to improve the load balancing for the the Monte Carlo simulation.

In the particle based RNG method, the particles that cross processor boundaries during a time step can be packaged up and sent to the other processors whenever it is convenient. This allows quite a bit of flexibility to find efficient parallel communication algorithms for any type of parallel machine by overlapping work and communication. In the cell based RNG method, particles must be transfered from the cell faces once all the particles in the simulation have reached a cell face. This extra synchronization step (Step 19 in Algorithm 2) in the parallel algorithm that is not present in the particle based RNG method; synchronization points in parallel algorithms generally reduce performance. Compared to the longest and most infrequent messages that can be sent with the particle based RNG method, the cell based RNG method will have smaller and more frequent communications. However, because the cells within each processor are independent, communication can be initiated with the boundary cells, and then the cells interior on each processor can advance their particles, helping alleviate the synchronization problem. Once

the interiors cells are finished, the boundary cells can finalize communication and then advance their particles. It is possible, even with the shorter messages of the cell based RNG method, that the work to communication ratio will be sufficiently large.

Some parallel computer systems are a collection of nodes, and each node consists of several processors. For these systems, other parallel strategies are possible. In the particle based RNG method, each processor on the node works on advancing a subset of the particles that are found on the node. For the cell based RNG method, the cells are independent, and each processor within the node can work on a subset of the cells, advancing the particles within each cell.

While all this speculation addresses key aspects of the two methods, it doesn't actually show anything which method will work best in parallel of if the run time performance hit of the new method is small enough to justify the memory savings. Real parallel codes must be implemented and tested on a variety of test problems in order to determine if any of the issues raised in this section will be detrimental to the new cell-based RNG method or if the method will still be fairly efficient.

6. CONCLUSIONS

Time dependent Monte Carlo particle simulations can be memory intensive. For reproducibility on any number of processors, the random number state is typically stored with the particle, increasing the particle size. The new cell based RNG method presented here, which stores the random number state with each cell, can save memory, given enough particles in each cell. Extra bookkeeping is required in order to ensure reproducibility, and this bookkeeping reduces the performance of the code by a factor of 1.38 with a standard deviation of 0.88 for the serial calculations shown here. If the random number state stored on the particle in the standard method is the smallest possible, one four byte integer, the memory savings of the new method is limited to about six percent for three dimensional simulations. However, if the random number state is forty bytes, the memory savings is about thirty six percent for three dimensional simulations and even greater for two and one dimensional simulations.

REFERENCES

[1] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, F. Rossi, *GNU Scientific Library Reference Manual*, 1.2 ed. (2002), URL `http://www.gnu.org/software/gsl/gsl.html`.

[2] N. A. Gentile, "Private Communication", (2002).

[3] N. A. Gentile, N. Keen, J. Rathkopf, "The KULL IMC Package", Tech. Rep. UCRL-JC-132743, Lawrence Livermore National Laboratory, Livermore, CA (1998).

[4] M. Mascagni, A. Srinivasan, *The Scalable Parallel Random Number Generators Library (SPRNG) for ASCI Monte Carlo Computations*, Florida State University (1999), URL `http://sprng.cs.fsu.edu/`.

[5] O. E. Percus, M. H. Kalos, "Random Number Generators for MIMD Parallel Processors", *Journal of Distributed and Parallel Computing*, **vol. 6, no. 3**, pp. 447–497 (1989).

[6] T. J. Urbatsch, T. M. Evans, "Reproducibility in Parallel Monte Carlo Codes", Los Alamos National Laboratory Memorandum (1999).

[7] T. J. Urbatsch, T. M. Evans, “MILAGRO Implicit Monte Carlo: New Capabilites and Results”, Tech. Rep. LA-UR-00-6118, Los Alamos National Laboratory (2000), URL `http://lib-www.lanl.gov/la-pubs/00357106.pdf`.