

STATUS OF VECTORIZED MONTE CARLO FOR PARTICLE TRANSPORT ANALYSIS

William R. Martin

UNIVERSITY OF MICHIGAN
ANN ARBOR, MICHIGAN 48109-2104

Forrest B. Brown

KNOLLS ATOMIC POWER LABORATORY
SCHENECTADY, NEW YORK 12301-1072

Review Article

Summary

The conventional particle transport Monte Carlo algorithm is ill suited for modern vector supercomputers because the random nature of the particle transport process in the history based algorithm inhibits construction of vectors. An alternative, event-based algorithm is suitable for vectorization and has been used recently to achieve impressive gains in performance on vector supercomputers. This review describes the event-based algorithm and several variations of it. Implementations of this algorithm for applications in particle transport are described, and their relative merits are discussed. The implementation of Monte Carlo methods on multiple vector parallel processors is considered, as is the potential of massively parallel processors for Monte Carlo particle transport simulations.

The International Journal of Supercomputer Applications, Volume 1, Number 2, pp. 11-32.

Introduction

The Monte Carlo method is one of the principal computational techniques used for analyzing particle transport problems. The advantage of the Monte Carlo method is its generality—it can be used to simulate almost any particle transport problem, regardless of geometry or physical complexity, as long as the geometry can be described mathematically and the physical process can be represented by equations or probability distributions.

Historically, the principal drawback to the use of Monte Carlo methods has been its excessive demand on computational resources. Monte Carlo methods may require enormous amounts of central processor unit (CPU) time to allow the simulation of the necessary number of particle histories (perhaps millions) to achieve reasonable statistics, a consequence of the fact that the variance in the answer improves only as $1/\sqrt{N}$, where N is the number of histories.

Monte Carlo analysis places a premium on computational speed and memory, and the substantial increases in computer performance over the past decade would seem ideally suited to meet this demand. However, for some applications, including Monte Carlo, this increase in performance has been difficult to exploit because of the difficulty in developing algorithms that map efficiently onto vector architectures, which are employed in the largest and fastest computers at the current time—vector supercomputers. This review summarizes the status of efforts to develop and implement Monte Carlo algorithms for vector supercomputers. Additional details of vectorized Monte Carlo algorithms and those aspects of vector architectures that affect Monte Carlo algorithm development may be found in Brown and Martin (1985).

The overall gains in computational performance over the past several decades and the contribution of vector architectures to these gains are discussed. A summary of the conventional Monte Carlo algorithm and why it is not suitable for vector processing is presented, and an alternative, event-based algorithm, which is amenable to vector processing, is described. We also describe variations of this algorithm and specific implementations for several Monte Carlo applications. The issue of multi-tasking Monte Carlo on vector-parallel processors is discussed, and we conclude with a few remarks regarding

Monte Carlo analysis places a premium on computational speed and memory, and the substantial increases in computer performance over the past decade would seem ideally suited to meet this demand.

future directions for Monte Carlo methods on advanced computers.

Computer Performance

The unprecedented increase in computer performance over the past several decades is illustrated in Figure 1, which is a plot of performance measured in millions of floating point operations per second (MFLOPS) for a single CPU.

Figure 1 indicates that the impressive improvement in computer performance has been due not only to advances in hardware, but also to innovations in architecture—how the computer is designed and organized to carry out its computational tasks. This is seen by noting that after the CDC 7600, two speeds are plotted for each computer, scalar and vector. The scalar speed corresponds to the speed of the conventional (scalar) CPU, whereas the vector speed is indicative of the principal innovation for large-scale computation, vector (pipelined) architectures. The data in Figure 1 represent actual timing results rather than advertised peak rates, which are generally not reliable indicators for scientific computation. However, even these data should be taken carefully, especially the vector speeds, because they have been obtained from relatively simple kernels, which may not be representative of actual applications.

A vector, or pipelined, architecture is characterized by arithmetic units which are designed like an automobile assembly line; they are segmented into smaller tasks, each of which may take relatively little time to complete. Although the overall time for the total task (e.g., a multiplication) may be longer than for a conventional (scalar) arithmetic unit, the segmented arithmetic unit can accept vectors of operands, which stream through the unit in a lockstep fashion. Once the first result exits the arithmetic unit (in an amount of time denoted the startup time), the subsequent results are generated once every clock cycle (at the streaming rate).

Because one vector instruction executes the entire vector operation (which might correspond to 64 multiplications on the CRAY-1 or 65,535 multiplications on the CDC CYBER-205), it is necessary for the algorithm to construct vectors of operands and perform identical operations on every component of the vector, although of course the operands can be different. This task of developing or adapting an algorithm for a vector CPU is known as *vectorization*, and is essential for realizing the

potential performance of the vector CPU. Thus, for Monte Carlo codes to take advantage of the gains in computer performance of the past decade, and hence follow the upper curve in Figure 1, they must be vectorized.

Conventional Monte Carlo

THE HISTORY-BASED ALGORITHM

The conventional Monte Carlo algorithm is inherently scalar and cannot be vectorized. The culprit is its history-based structure. In a typical Monte Carlo code such as MCNP (Los Alamos, 1981) or MORSE (Straker, Scott, and Byrn, 1970), a particle will be emitted via a source routine, transported through the medium of interest (tracked), and processed through whatever collisions or interactions may occur (collision analysis). As the history unfolds, results of the simulation are accumulated (tallies), and the simulation continues until the particle is terminated, such as by absorption in the medium, by escape from the problem geometry, or, in a time-dependent simulation, by the end of the time step. The code will then loop over the requisite number of histories to achieve acceptable statistics (or unacceptable cost). (These remarks actually describe what is known as analog Monte Carlo, but our discussion regarding vectorization is also applicable to nonanalog Monte Carlo.) Figure 2 is a schematic of several particle histories in an idealized Monte Carlo simulation.

MULTIPLE HISTORIES AND VECTORS

Because the particle simulation is a random walk, or Markov, process, each step of a history is determined by statistical means (e.g., distance to the next collision, kind of collision, angle of scatter). Therefore, treating many histories simultaneously fails miserably, because the vector is destroyed after the first step in the simulation—some particles in the vector will reach a boundary, some may suffer a capture collision, others a scattering collision, some may reach the end of the current time step, etc. Thus, the vector of particles is no longer a vector from the standpoint of the vector CPU, because different operations will be performed on each component of the vector. This is apparent from an examination of Figure 2. The six particles can initially be processed as a vector because they are all being emitted by the source, but after this first step the vector becomes a

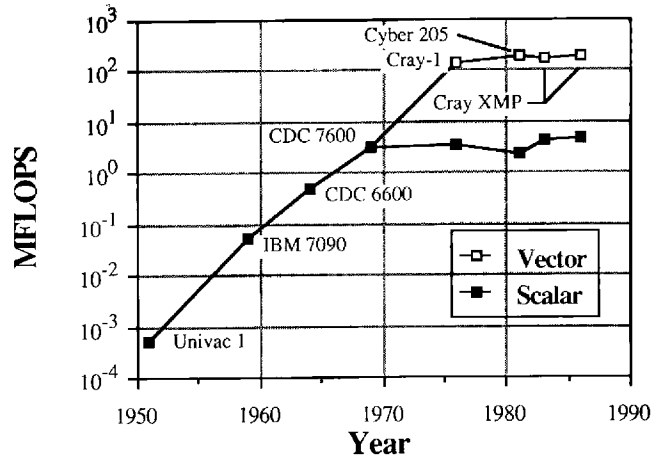


Fig. 1 Computer performance versus time
Sources:

- Univac 1 and IBM 7090-(Hockney and Jesshope, 1981)
- CDC 6600, 7600 and vector data for CRAY X-MP and CDC Cyber-205-(Dongarra, 1986)
- Scalar data for CRAY X-MP and CDC Cyber-205-(Bucher and Simmons, 1985)

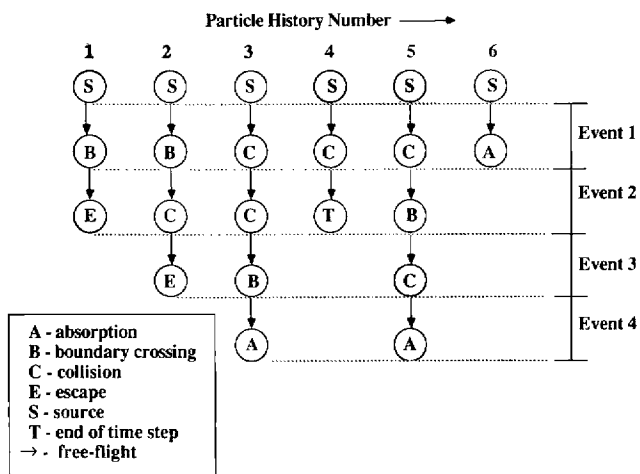


Fig. 2 The history-based algorithm

collection of disparate particles tracing their own paths through phase space.

Recompiling an old Monte Carlo source code on a vector supercomputer such as the CRAY X-MP will not result in efficient utilization of the vector architecture, because of the inherently scalar nature of the history-based algorithm. The potential loss in performance that results can approach a factor of 10 to 20. Recently a number of researchers have attained significant gains in performance by totally restructuring the Monte Carlo algorithm to be compatible with a vector architecture. Although there are substantial differences in the individual approaches, all of these vectorized algorithms have a common characteristic—they all are event-based algorithms versus the history-based algorithm of a conventional (scalar) Monte Carlo code.

Vectorized Monte Carlo

BACKGROUND

The first mention in the literature of the event-based approach was in 1973 (Troubetzkoy, Steinberg, and Kalos, 1973). This work was targeted for the Illiac-IV parallel processor with 64 processors, all controlled by one CPU. As with a vector architecture, the approach of assigning each history to a processor of the Illiac-IV would not work. Thus, an alternative approach was developed, which was based on breaking up the overall simulation of many histories into collections of events, such as tracking, collision analysis, boundary crossing, and tallying, which are similar and can be processed in a vectorized manner.

DEFINITION OF EVENT

We define an event as that portion of a history which is initiated with the appearance of an emerging particle in phase space at (\mathbf{r}, \mathbf{v}) and terminating at $(\mathbf{r}', \mathbf{v}')$, which is the beginning of the next event. Figure 3 illustrates the concept of an event. For example, an event might be initiated at (\mathbf{r}, \mathbf{v}) by sampling from a source distribution, emerging from a scattering collision, entering from census (time-dependent simulation), crossing a boundary, or emission from a nuclear reaction (e.g., fission).

Once the event is initiated, it proceeds by following the particle to the end of the event, which can be terminated by collision (any kind), boundary crossing (in-

cluding escape), census (end of time step), or killing (nonanalog Monte Carlo).

The important observation is that all events are similar—a vector of particles (the particle *bank*) can be processed for one event in a vector manner. The next section describes the basic manner in which the particle vector is processed for one event.

THE BASIC EVENT-BASED ALGORITHM

Assume that we have a reservoir (or bank) of particles at event iteration n , where each particle j is described by a number of attributes \mathbf{x}_j^n , where

$$\mathbf{x}_j^n = [x_{j1}^n, x_{j2}^n, x_{j3}^n, \dots, x_{jK}^n],$$

where K is the number of attributes. For example, $(x_{j1}^n, x_{j2}^n, x_{j3}^n)$ might be the position $\mathbf{r} = (x, y, z)$, and $(x_{j4}^n, x_{j5}^n, x_{j6}^n)$ might be the direction cosines $\boldsymbol{\Omega} = (u, v, w)$, and x_{j7}^n the particle time at event iteration n for particle j . Let us now define the particle bank vector $\boldsymbol{\Gamma}^n$ as the set of all particle vectors at event iteration n :

$$\boldsymbol{\Gamma}^n = [\mathbf{x}_1^n, \mathbf{x}_2^n, \mathbf{x}_3^n, \dots, \mathbf{x}_{L_n}^n],$$

where L_n is the number of particles in the particle vector at the beginning of event iteration n . In general, the bank vector $\boldsymbol{\Gamma}$ is ordered randomly and its order will change from one event iteration to the next—no attempt is made to keep track of individual particles in $\boldsymbol{\Gamma}$ from one event to the next. Given $\boldsymbol{\Gamma}^n$, the goal is to advance it to the next event iteration $n + 1$, as summarized in Figure 4.

The following observations can be made regarding the event-based algorithm illustrated in Figure 4:

1. The arrays \mathbf{S} and \mathbf{R} are the usual cross-section and geometry arrays. In order to perform the vector calculations for the distances to collisions and boundaries, cross-section data and geometry data tabulated by particle index must be gathered from \mathbf{S} and \mathbf{R} into the arrays $\boldsymbol{\Sigma}$ and $\boldsymbol{\rho}$:

$$\boldsymbol{\Sigma} = [\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Sigma}_3, \dots, \boldsymbol{\Sigma}_{L_n}]$$

$$\boldsymbol{\rho} = [\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, \boldsymbol{\rho}_3, \dots, \boldsymbol{\rho}_{L_n}],$$

- where $\boldsymbol{\Sigma}_j$ represents the cross sections and $\boldsymbol{\rho}_j$ the geometry data for the zone that contains particle j .
2. Figure 4 is only intended to be illustrative of the event-based algorithm, which may differ significantly from one implementation to the next, as will be seen in the next section.
3. The compression step to eliminate terminated

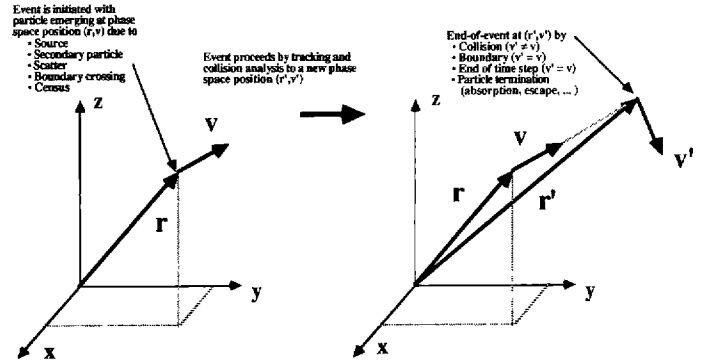
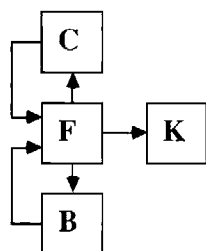


Fig. 3 Event initiation and progression

For event $n = 0, 1, 2, \dots$

- Fetch $\boldsymbol{\Gamma}^n$ ($\boldsymbol{\Gamma}^0$ is the initial particle vector)
 - Process the free-flight portion of the event
-
- ↑
- FREE FLIGHT ANALYSIS**
- Gather the cross section data and geometry data tabulated by particle,
 - $\boldsymbol{\Sigma} \leftarrow \mathbf{S}$
 - $\boldsymbol{\rho} \leftarrow \mathbf{R}$
 - Using $\boldsymbol{\Sigma}$, sample a vector \mathbf{d}_c of distances to collision
 - Using $\boldsymbol{\rho}$, determine the vector of minimum distances to a boundary, \mathbf{d}_b
 - Determine the minimum distances to end of event, $d_{\min} = \min\{d_c, d_b\}$
 - Update the particle coordinates, $\mathbf{r}^{n+1} = \mathbf{r}^n + \boldsymbol{\Omega}^n \cdot d_{\min}$
-
- ↓
- Process collisions
 - Gather particle attributes from bank, $\boldsymbol{\Omega} \leftarrow \boldsymbol{\Gamma}^n$ and $\mathbf{E} \leftarrow \boldsymbol{\Gamma}^n$
-
- ↑
- COLLISION ANALYSIS**
- Evaluate collision physics equations, determine new direction cosines $\boldsymbol{\Omega}' \leftarrow \boldsymbol{\Omega}$ and energies $\mathbf{E}' \leftarrow \mathbf{E}$
-
- ↓
- Scatter new particle attributes back into bank,
 - $\boldsymbol{\Omega}' \rightarrow \boldsymbol{\Gamma}^n$, $\mathbf{E}' \rightarrow \boldsymbol{\Gamma}^n$
-
- Process boundary crossings
 - Gather particle zone indices \mathbf{Z} from bank, $\mathbf{Z} \leftarrow \boldsymbol{\Gamma}^n$
-
- ↑
- BOUNDARY ANALYSIS**
- Determine new zone indices for particles, $\mathbf{Z}' \leftarrow \mathbf{Z}$
-
- ↓
- Scatter new zone indices back into particle bank, $\mathbf{Z}' \rightarrow \boldsymbol{\Gamma}^n$
-
- Compress the particle bank $\boldsymbol{\Gamma}^n$ to eliminate the particles which have been terminated, forming the updated particle vector $\boldsymbol{\Gamma}^{n+1}$ (with L_{n+1} particles),
 - $\boldsymbol{\Gamma}^n \Rightarrow \boldsymbol{\Gamma}^{n+1}$
 - If $L_{n+1} \neq 0$, continue.

Fig. 4 The event-based Monte Carlo algorithm



Task F - process free flights
 Task C - process collisions
 Task B - process boundary crossings
 Task K - process terminations (due to absorption, escape, or end of time step)

Fig. 5 Connectivity of computational tasks in an event

particles effectively scrambles the order of the particles in the bank vector Γ and leads to one consequence of the event-based algorithm—it is difficult to piece together individual particle histories from the event-based simulation.

4. There is a premium on efficient data-handling operations, resulting from the need to gather, scatter, compress, etc., the particle vector and other data arrays during every event iteration.
5. Most of the steps are vector operations, except for the data handling operations, which are generally custom-coded routines (sometimes in hardware), and the tally operations, which cannot be vectorized. The collision analysis, boundary-crossing analysis (perhaps with Russian roulette and splitting), and other operations to determine the outcome of the event can also be performed in a vector fashion, but subvectors would need to be defined (e.g., gathered from the main particle vector) for each distinct outcome and then processed.

THE STACK-DRIVEN VARIATION

In the basic event-driven algorithm, the particle vector is processed in a manner similar to the conventional history-based algorithm. One cycles through the free-flight analysis, then the collision analysis routine, then the boundary analysis routine, etc., on an event-by-event basis. At any time during the simulation, all particles will be in the same event iteration. A variation on this basic approach, called the stack-driven algorithm, arises when events are further subdivided into smaller computational tasks, which are then processed independently. The simplified event-based algorithm in Figure 4, for example, may be logically subdivided into the four separate computational tasks of free-flight analysis, collision analysis, boundary analysis, and particle termination analysis. Rather than cycling through these four tasks in a fixed order, the calculation may proceed by selecting the task involving the greatest number of particles and then performing the analysis for that task. According to the outcome of this analysis, the affected particles are then queued for the next appropriate task.

The stack-driven variation involves greater complexity than the basic event-based algorithm in two areas: control of the calculational sequence and management of the particle attribute data. Considering first the control aspects, the stack-driven approach further

scrambles the order in which particle histories are processed, because a computational task is selected according to the number of particles queued for each task. The order of task execution becomes random, and the execution of a particular task may involve particles from different event iterations. Upon completion of a task, the affected particles must be queued for their next respective tasks. Each individual task must therefore be aware of its connectivity to the other tasks, and must disperse the particles accordingly. Figure 5 illustrates the connectivity between the four tasks used in the simplified example.

Considering next the management of particle attribute data, the stack-driven approach leads to a proliferation of particle banks, because a separate bank or stack is needed for each computational task. Particles awaiting a particular task are stacked up until that task is processed. Upon completion of the task, the particle attributes must be dispersed into the appropriate stacks for their next tasks. Figure 6 illustrates the stack-driven variation derived from the basic event-based algorithm of Figure 4.

Each of the four computational tasks includes its own stack for holding particle attributes and its own control logic for dispersing particles upon completion to their next tasks. The algorithm proceeds by selecting the task with the largest stack, executing that task, and then dispersing the particle attributes to other stacks. This procedure is followed until all stacks are empty, signifying the end of the calculation. The principal difference between the event-based algorithm and the stack-driven variation is the order in which computational tasks are executed. The fixed sequence of tasks in the event-based algorithm leads to simpler control logic and management of particle attribute data, at the cost of shorter vector lengths for each individual task. The stack-driven variation selects tasks in a sequence that maximizes the vector lengths, but involves additional logic for managing the particle attribute data. The vector computations performed in each task are the same in the two approaches, however, and each approach has been used successfully in a number of practical applications described below.

OTHER VARIATIONS

Although all of the vectorized Monte Carlo algorithms are based on the event-based approach, either the basic

- Fetch initial particle vector, $\Gamma_F \leftarrow \Gamma_0$
- While tasks are pending, select and execute task with longest vector $[\Gamma_F, \Gamma_C, \Gamma_B, \Gamma_K]$:

Task F - Process free flight portion of event

- Fetch Γ_F
- **Free flight analysis** (see Figure 4)
- Disperse Γ_F according to next task :
 - Γ_F (collisions) $\rightarrow \Gamma_C$
 - Γ_F (boundary crossings) $\rightarrow \Gamma_B$
 - Γ_F (terminations) $\rightarrow \Gamma_K$


Task C - Process collisions

- Fetch Γ_C
- **Collision analysis** (see Figure 4)
- Disperse Γ_C according to next task, $\Gamma_C \rightarrow \Gamma_F$

Task B - Process boundary crossings

- Fetch Γ_B
- **Boundary analysis** (see Figure 4)
- Disperse Γ_B according to next task, $\Gamma_B \rightarrow \Gamma_F$

Fig. 6 The stack-driven algorithm for event analysis



algorithm or the stack-driven variation, there are significant differences in specific implementations. The principal variations among these approaches depend on the manner in which the particle vectors are organized and treated. One characteristic is whether or not particles from more than one geometric zone are treated at the same time. If the particle bank Γ consists only of particles located within a single geometric zone, we denote this as a one-zone algorithm. An all-zone algorithm would then employ a particle bank consisting of particles from any zone in the problem geometry.

Another characteristic is the manner in which the particle banks are managed in stack-driven algorithms. In a "tagged-particle" scheme, there is only one particle bank but an additional particle attribute (i.e., the "tag") is used to keep track of the next task to be performed on each particle. The particle tags are examined to determine the next task to be processed. In an "explicit stack" scheme, a separate particle stack is explicitly reserved for each computational task. Upon completion of a task, particle data must be dispersed to the appropriate stacks for further analysis. In an "implicit stack" scheme, all particle data reside in one large bank and pointers to the particles are queued up for each task. A task uses its pointer list to gather particle data from the bank, performs the task analysis, scatters updated data back to the particle bank, and then disperses the particle pointer list to queues for other tasks.

Specific Vectorization Efforts

PERFORMANCE ASSESSMENT

The following sections describe several different implementations of the event-based algorithm for Monte Carlo analysis. It would be interesting to ascertain the relative performance of these different approaches, but this is not an easy task. In general, vectorization speedups are reported for each of these efforts, but these can be misleading and are difficult to assess because they are sensitive to the efficiency (or inefficiency) of the original scalar code. Nevertheless, these speedups are real from the standpoint of a user of the original scalar code and are useful measures of performance. It would be desirable, however, to have an absolute measure of computational efficiency, which might allow a comparison of the different approaches. One candidate

for this metric is the average CPU time required to process a particle for one event, or "track." This measure, defined as the $\mu\text{sec}/\text{track}$ (Bobrowicz et al., 1984), appears to be an excellent indicator of the absolute performance of a Monte Carlo code, whether scalar or vector, because it is remarkably insensitive to the physics or geometry. For example, for photon transport work described below, it was in the range of 6–15 $\mu\text{sec}/\text{track}$ for the vector code, even with 10-fold changes in the absorption cross sections and the geometric mesh. Typically, scalar Monte Carlo codes may process particles at the rate of 100–300 $\mu\text{sec}/\text{track}$.

PRELIMINARY STUDIES

The initial effort to develop a vectorized Monte Carlo algorithm (Brown and Martin, 1985; Calahan and Martin, 1980; Brown, Calahan, and Martin, 1981; Brown, Martin, and Calahan, 1981) was based on a prototype vectorized code developed by T. L. Jordan of Los Alamos National Laboratory (LANL). This was a simple (<300 lines) FORTRAN code that only analyzed gamma transport in a carbon cylinder, but it served well as a starting point for basic algorithmic studies. Both one-zone and all-zone algorithms (event-driven) were investigated, and the following speedups were obtained:

Speedups for simple gamma transport application

Version	Computer	Speedup
Scalar	Amdahl 470V/8	1
Scalar	CDC 7600	1
Scalar	CRAY-1	2
Vector	CRAY-1	7

Although these speedups were considered satisfactory, the simplicity of the codes did not allow one to draw conclusions with respect to production-level Monte Carlo codes. However, this effort stimulated the development of a more realistic vectorized Monte Carlo demonstration code, which is discussed in the next section.

MULTIGROUP NEUTRON TRANSPORT (ONE-ZONE)

Subsequent effort was devoted to the development of a vectorized Monte Carlo demonstration code that incorporated most of the significant physics options in standard production-level Monte Carlo codes such as MCNP and MORSE but did not have many of the user conve-

A one-zone algorithm necessitates an outer loop over zones, and it was determined empirically that the optimum algorithm was to process the zone with the most particles for only one event and then to proceed to the next zone with the most particles and process it.

niences typical of a true production code (Brown, 1981; Brown, Martin, and Calahan, 1981; Brown and Martin, 1985). This code, named MCVMG, was a multigroup code, as was a companion scalar code (MCS) that was developed to allow a thorough assessment of the efficiency of the vector code.

The vectorized code MCVMG was developed for the CDC CYBER-205 supercomputer and utilized a one-zone event-driven algorithm. Separate stacks were constructed for each zone, and particles leaving one zone were added to the particle stack for the neighboring zone. A one-zone algorithm necessitates an outer loop over zones, and it was determined empirically that the optimum algorithm was to process the zone with the most particles for only one event and then proceed to the next zone with the most particles and process it.

The initial results (Brown, Martin, and Calahan, 1981) were obtained by emulating the CYBER-205 instructions on a conventional computer and then using published timing data to calculate the speedups:

Emulated speedups for one-zone algorithm

Version	Computer	Speedup
Scalar	Amdahl 470V/8	1
Vector	CDC CYBER-205	40

Overall, the results indicated a speedup in the range 25–40 with MCVMG relative to the optimized scalar code MCS for several realistic problems. Subsequent implementation of the vector code on the CDC CYBER-205 verified these results to within 10% (Martin, 1983).

The advantage of the one-zone algorithm is the ease of tracking particles in a general geometry. Because all particles in the bank vector Γ have the same zone index, there is no need to gather zonal quantities, such as the cross-section array Σ and the geometry array ρ described above. There are several distinct disadvantages, however. The vector length is determined by the number of particles in the zone, and if there are many zones, the vector length may be prohibitively small. Another disadvantage is the need to have storage arrays (buffers) to contain the particles that cross a boundary from one zone to an adjacent zone. Because one does not know a priori how many particles will cross a boundary in any event iteration, these storage arrays may be quite long. In addition, if the problem is relatively transparent, there will be many boundary

crossings, hence a substantial computational effort just to move particles between zones.

CONTINUOUS-ENERGY NEUTRON TRANSPORT

Excellent results have been reported for a vectorized, continuous-energy Monte Carlo code for two-dimensional reactor lattice analysis on the CDC CYBER-205 (Brown, 1983; Brown and Mendelson, 1984). The approach utilizes an all-zone stack-driven algorithm with three particle stacks, two for the tracking and collision analysis, and the third (the "total" bank) to act as a particle reservoir. Because of the size of the cross-section data base, particles were grouped into "supergroups," and the current bank contained only particles from a particular supergroup. Only data for that particular supergroup are stored in memory, and particles which have scattered into a different supergroup are transferred out of the current bank into the total particle bank. Particle data are transferred into the appropriate bank as needed. The following speedups (versus the scalar code on the CDC 7600) were observed for a range of problems from "small" to "large" (Brown and Martin, 1985):

Speedups for 2-D continuous energy neutron transport

Application	Speedup
Fuel cell (small)	75-85
Fuel assembly (medium)	20-60
Fuel assembly with depletion (large)	20-40

The absolute timings for the vector code are approximately 6-10 μ sec/track for the above problems (Brown, 1986).

The advantage of this approach is that all zones are treated simultaneously, which will tend to keep vector lengths high and minimize data movement. However, zonal information must be gathered before each event iteration as in the generic algorithm discussed previously. A disadvantage of the multiple stacks is that particle data must be moved from one stack to another.

PHOTON TRANSPORT (STACK-DRIVEN)

A vectorized Monte Carlo code for the analysis of photon transport in a two-dimensional Lagrangian mesh has been developed (Bobrowicz et al., 1984). The approach is all-zone and stack-driven, with nine working

The vector length is determined by the number of particles in the zone, and if there are many zones, the vector length may be prohibitively small.

stacks corresponding to the distinct computational tasks in the Monte Carlo simulation. For example, separate particle stacks are constructed to determine the distance to boundary and to perform the collision analysis, Thomson scattering, and other tasks. Particles are inserted into the appropriate stack from other stacks. A stack is executed when its length reaches 64, which is the length of the Cray vector registers. Results were reported for the original scalar code and the vectorized code for the CRAY-1, and are tabulated below (denoted as A), along with more recent results (Fisher, 1986) for the CRAY X-MP/4 (B). The latter results are for a modified version of the original code which employed 12 working stacks.

Speedups for stack-driven photon transport algorithm

Version	Computer	μ sec/track	Speedup
Scalar (A)	CRAY-1	100	1
Vector (A)	CRAY-1	24	4
Scalar (B)	CRAY X-MP/4	83	1
Vector (B)	CRAY X-MP/4	15	6

The advantage of this approach is its optimization of the Cray architecture, because all stacks (until the end of the simulation) will have 64 particles when they are executed. A disadvantage of this approach is the need to transfer particle data between stacks.

NEUTRON TRANSPORT (STACK-DRIVEN)

Timing results for a vectorized Monte Carlo algorithm for neutron transport in a two-dimensional Lagrangian mesh have been reported (Chauvet, 1984; Chauvet, 1985). The algorithm is similar to the approach of Bobrowicz et al. (1984), except there are fewer stacks and data movement between stacks is minimized where possible by transferring particle indexes, rather than particle data, between the stacks. The reported speedups relative to the scalar code on the CRAY-1 were

Speedups for stack-driven neutron transport algorithm

Version	Computer	Speedup
Vector	CRAY-1	7
Vector	CYBER-205	7
Vector	CRAY X-MP/4	13

The vectorized versions for the CRAY-1 and CRAY X-MP/4 employed assembly coding for determining dis-

tance to boundary, which is generally the most computationally intensive portion of a Monte Carlo code. The CYBER-205 version used the "q8" calls for the data-handling operations (gather/scatter/compress, etc.) while the CRAY X-MP/4 version took advantage of the hardware gather/scatter capabilities of that computer. Chauvet also reports results with multitasking on the CRAY X-MP/4, which is discussed in the next section.

PHOTON TRANSPORT (EVENT-DRIVEN)

In a second application concerning photon transport, an alternative vectorized Monte Carlo algorithm has been developed to analyze photon transport in a two-dimensional Lagrangian mesh (Martin and Calahan, 1982; Martin, Rathkopf, and Nowak, 1985; Martin, Nowak, and Rathkopf, 1986). This code, named VPHOT, employs an all-zone, event-driven algorithm with two principal stacks of particles—a main stack and a buffer stack—and several substacks that are created upon demand. The overall structure of the algorithm is very similar to that described in Figure 4, except for the buffer stack, which holds source particles and particles created during the simulation, such as particles created by splitting or secondary particles emitted by a nuclear reaction. The buffer stack is used to replenish the main particle stack after each event iteration, which helps to keep the vector lengths high.

The substacks are utilized for those operations in which the entire bank of particles would not participate. For example, if Thomson scattering is relatively rare, it would be wasteful to process the entire particle vector Γ through the Thomson analysis and then sort out afterward which results (i.e., the new direction cosines) to accept. Rather, the necessary attributes for the affected particles (which for Thomson scattering are only the three direction cosines) are gathered from Γ into a temporary vector and processed vectorially. The new direction cosines are then scattered back into Γ . This approach of gathering up only those particle data needed to perform the necessary operation has the advantage of minimizing data movement between stacks; it is also used extensively in another algorithm (Brown, 1986) discussed below. However, it does place a premium on a fast gather/scatter capability. It should be noted that a definite order is followed as the event iteration is processed; for example, a Thomson substack is constructed every event iteration and processed, even if it contains

VPHOT employs an all-zone, event-driven algorithm with two principal stacks of particles—a main stack and a buffer stack—and several substacks are created upon demand. The buffer stack is used to replenish the main particle stack after each event iteration, which helps to keep the vector lengths high.

only one particle. This should be compared with the stack-driven approach, where processing of the Thomson stack would be delayed until it was full.

A scalar version of VPHOT, named SPHOT, has been developed to allow a meaningful comparison of the vector algorithm. The VPHOT and SPHOT results have been compared with a reference Monte Carlo code at Lawrence Livermore National Laboratory (LLNL) for a typical ICF test problem. The principal results are summarized below:

Speedups for event-driven photon transport algorithm

Code	Computer	$\mu\text{sec}/\text{track}$	Speedup
SPHOT	CRAY-1	45	1
VPHOT	CRAY-1	12	4
VPHOT	CRAY X-MP/4	7	6
VPHOT	Fujitsu VP-200	5	9
VPHOT	Convex C-1	120	1/3

The speedups are relative to the optimum scalar algorithm (SPHOT) on the CRAY-1. (The Fujitsu VP-200 is marketed as the Amdahl 1200 in the U.S.) For comparison, the reference LLNL code yielded a speed of 80 $\mu\text{sec}/\text{track}$ for the same test problem, hence a speedup of 12 for VPHOT versus the LLNL code.

Additional details regarding the specific algorithm and the detailed timing results may be found in Martin, Nowak, and Rathkopf (1986). One of the interesting conclusions is that one need not worry about the "endgame"—that portion of the Monte Carlo simulation where there are only a few particles (the "stragglers") and hence short vector lengths. Concern about this has led to suggestions to maintain separate scalar and vector copies, where one might switch to the scalar version when the vector length is short enough. However, it is shown that the "endgame" has a negligible effect on the overall performance because the bulk of the simulation is performed with relatively long vectors.

KENO-IV

An attempt to vectorize the production-level Monte Carlo code KENO-IV was made with disappointing results (Asai, Higuchi, and Katakura, 1986). These researchers employed an all-zone, stack-driven algorithm with stacks for the various tasks, which are processed in order of length. The resulting vectorized code was tested on two relatively simple problems and yielded

speedups of 1.4 with respect to the original (scalar) version of KENO-IV. They attribute these relatively poor results (compared with results obtained by others) to deficiencies in the compiler, slow indirect addressing (gather/scatter), and the large number of sorting operations.

Given the excellent results obtained by others (including recent results by Brown with a production-level code discussed below), there may be some algorithmic changes that could be incorporated into the vectorized version of KENO-IV to obtain improved results.

CONTINUOUS-ENERGY MONTE CARLO

Except for the KENO-IV work, none of the above approaches treated the case of a general-geometry Monte Carlo code with a general-physics package. That is, the seminal work by Brown, Martin, and Calahan with MCVMG (Brown, 1981; Brown, Martin, and Calahan, 1981; Brown and Martin, 1985) was constrained to multi-group Monte Carlo with a modest number of geometric zones. The subsequent work by Brown (Brown, 1983; Brown and Mendelson, 1984) was constrained to two-dimensional geometries. In addition, the works of Bobrowicz et al. (1984), Chauvet (1984; 1985), and Martin et al. (Martin and Calahan, 1982; Martin, Rathkopf, and Nowak, 1985; Martin, Nowak, and Rathkopf, 1986) were constrained to particle transport in a two-dimensional Lagrangian mesh. Therefore, until 1986, general-geometry, continuous-energy Monte Carlo had resisted vectorization attempts. This challenge appears to have been addressed in the recent work by Brown (1986). This effort has resulted in a three-dimensional, general-geometry, continuous-energy Monte Carlo production code called RACER3D that has essentially no restrictions on problem geometry or problem physics (for reactor analysis), and hence is capable of analyzing configurations typically treated by production codes such as MCNP, MORSE, or KENO-IV.

Brown's method utilizes an all-zone, stack-driven approach, with one large stack to hold particle data between events. Rather than shuffling particle data among the stacks, Brown constructs queues of pointers for each task; a pointer refers to the appropriate particle in the main stack. A task is processed if it contains the most particle pointers, by gathering up the affected particle attributes (perhaps only a fraction of the total), performing the indicated operations (vectorized), and then

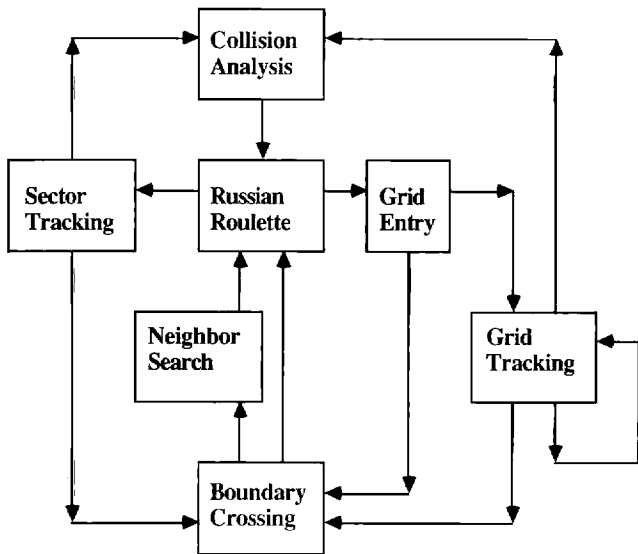


Fig. 7 Computational tasks and connectivity for RACER3D

scattering the affected attributes back into the main stack. Thus, the particle pointers rather than the particle attributes are "shuffled." The method of shuffling pointers rather than particle data was employed to a lesser extent by Martin et al. and Chauvet, as noted previously.

While the basic stack-driven approach used in this application is similar to that given in Figure 6, the complexity of the geometric treatment has resulted in seven large computational tasks that are interconnected in a complicated manner, as shown in Figure 7. The "sector-tracking" task handles the particle free-flight analysis in general three-dimensional geometry, while the "grid-tracking" tasks analyze free-flights in more detailed (but more regular and restricted) geometric regions, which may be embedded in the general geometry. The "neighbor search" task deals with the difficult problem of determining which neighboring regions will be entered after crossing a region boundary.

Each of the tasks shown in Figure 7 comprises a very large section of coding, with several levels of control logic, data management, and vector computation. Performance depends critically on the availability of fast hardware gather/scatter and compress/expand capabilities. The sector-tracking tasks, for example, involve nested iterations over the numbers and the types of surfaces that bound the geometric regions. At the highest level there is an iteration over the number of surfaces associated with the regions containing particles. This iteration requires temporary stacks of particle and surface pointers. On each iteration, the temporary particle and surface pointer lists are compressed and distances are calculated. The distance calculations, however, involve additional iterations over the types of surfaces (e.g., planes, spheres, cylinders, general quadratics). Additional temporary stacks are used to hold the selected particle and surface pointers for each iteration. Actual particle attributes and surface parameters are finally gathered from the main stack into temporary stacks and a vectorized distance calculation is performed. Minimum distances are retained and scattered back to the main particle stack, and all temporary stacks are discarded. Upon completion of all iterations in the sector-tracking task, particle pointers are dispersed to queues for either the collision analysis or the boundary analysis tasks. The other tasks shown in Figure 7 are similar in complexity

$$\text{Speedup } (N \text{ CPUs}) = S_N \equiv \frac{t_s}{t_N},$$

where both the single CPU and the multiple CPU calculations are performed on a dedicated computer. Now let us define the sequential fraction f_s as the fraction of the calculation that can be performed by only one CPU and the parallel fraction $f_N = 1 - f_s$ as that fraction that can be performed by all N CPUs. The speedup can then be obtained,

$$S_N = \frac{1}{f_s + f_N/N},$$

which is simply Amdahl's law (Amdahl, 1967). This is the maximum theoretical speedup for N processors, and it is easy to see that the sequential fraction is the critical parameter. For example, if $f_s = .1$, the maximum speedup is 10, no matter how many processors are utilized. The efficiency of a parallel algorithm may then be defined as the ratio of the observed speedup to the theoretical speedup,

$$\text{Efficiency } (N \text{ CPUs}) = \epsilon_N \equiv \frac{S_N^{\text{actual}}}{S_N^{\text{Amdahl}}}.$$

Chauvet (1984; 1985) has reported on a multitasked version of a stack-driven algorithm, similar to that of Bobrowicz et al. (1984), that replicates the overall simulation in each processor. For one case on the CRAY X-MP/2, Chauvet predicted a maximum speedup of 1.961 and observed a speedup of 1.957, for an efficiency of 99.8%. In addition, parallel-vector algorithms have been developed (Wan and Martin, 1986) for the CRAY X-MP and the IBM 3090 computers. No actual timing results have been reported yet, although predicted speedups are reported, which indicate a degradation in performance as the number of CPUs increases, because of the decrease in the vector length. However, it appears that there should not be major difficulties in adapting existing vectorized Monte Carlo codes to the modestly parallel (≤ 6 CPUs), shared memory multiprocessors typified by these computers.

There may be challenges with the new class of massively parallel vector processors such as the Intel iPSC-VX or the Floating Point Systems T-series computers. These are hypercube-connected multiprocessors of scalar and vector computers with a distributed memory, so that each CPU has its own memory and there is no

shared memory. If the memory assigned to each processor is not sufficient to contain the entire code and data base for the overall Monte Carlo simulation, then the problem may need to be partitioned and the individual nodes need to communicate (Martin et al., 1986), which will complicate the algorithm and introduce additional overhead for the communications. In addition, the large number of vector processors exacerbates the issue discussed above regarding vector length versus task granularity. Thus, substantial algorithm development may be needed to take advantage of this particular architecture.

One possible approach involves an extension of the stack-driven algorithm, with separate computational tasks placed on separate processors. Particular attention, however, must be given to the issues of load balancing between tasks and interprocessor communication requirements.

Concluding Remarks

The following remarks summarize the status of vectorized Monte Carlo:

Monte Carlo has been successfully vectorized. Although the conventional Monte Carlo algorithm is inherently scalar because of its history-based structure, the event-based algorithm has been shown to be very effective at exploiting vector architectures. Until 1986, only "restricted" Monte Carlo codes (restricted by geometry, physics, or relative simplicity) had been successfully vectorized. Brown's recent work indicates that general-geometry, general-purpose Monte Carlo is also vectorizable with excellent speedups.

The vectorization of a Monte Carlo code is a significant undertaking. One conclusion arising from the successful vectorization efforts is that global algorithmic changes are necessary—comprehensive changes to the data structures and possibly a complete rewrite of the code. This degree of effort may not be possible, or affordable, but is probably essential to achieve significant speedups. Thus, the disappointing results with the KENO-IV vectorization may be due to the fact that it was encumbered by the structure of the original KENO-IV code.

The next challenge is multitasking. The question is, How efficient will it be when implemented on multiple vector processors? For the current generation of modestly parallel vector processors, such as the CRAY X-MP/48 and the IBM 3090/600, this may not be a

problem. However, as the number of processors grows, the level of efficiency may become unacceptable. Moreover, the new class of distributed memory vector processors will pose its own challenges for algorithm designers. Additional research is needed to address these issues.

ACKNOWLEDGMENT

This work was partially supported by grants from IBM (Kingston) and Lawrence Livermore National Laboratory.

BIOGRAPHY

William R. Martin was born in Flint, Michigan, on June 2, 1945. He received a B.S.E. degree in engineering physics from the University of Michigan in 1967 and an M.S. degree in physics from the University of Wisconsin in 1968. Following a tour of duty with the U.S. Navy (Naval Reactors Division), he returned to the University of Michigan in 1973 and received his Ph.D. degree in nuclear engineering in 1976. He then joined Combustion Engineering, Inc., and was responsible for developing advanced methods for nuclear reactor analysis. He returned to the University of Michigan in 1977 and is currently an Associate Professor of Nuclear Engineering and has recently been named Director of the Laboratory for Scientific Computation in the College of Engineering. His research interests include computational methods development in several areas, including particle transport, reactor analysis, and thermal/hydraulics as well as the de-

velopment of algorithms for scientific computation on advanced computer architectures.

Forrest B. Brown received his Ph.D. degree in nuclear engineering from the University of Michigan in 1981, with a dissertation titled "Vectorized Monte Carlo."

In 1981 he joined the Knolls Atomic Power Laboratory in Schenectady, New York, as a physicist. He continued his work on vectorized Monte Carlo methods, developing new calculational techniques and applying them to the solution of large-scale reactor physics problems. In 1983, he received the Outstanding Technical Contribution award from the Power Systems division of the General Electric Company for this work. He is currently the Lead Physicist for the Nuclear Design Methods group at KAPL. His principal activities include the development of Monte Carlo methods, advanced computational techniques for large-scale reactor physics problems, and the effective utilization of vector/parallel computers in nuclear engineering.

REFERENCES

Amdahl, G. 1967. Validity of the single processor approach to achieving large

scale computing capabilities. *Proc. Spring Joint Conf. AFIPS* 30:483-485.

Asai, K., Higuchi, K., and Katakura, J. 1986. Vectorization of the KENO-IV code. *Nucl. Sci. Eng.* 92:298-302.

Bobrowicz, F. W., Lynch, J. E., Fisher, K. J., and Tabor, J. E. 1984. Vectorized Monte Carlo photon transport. *Parallel Comput.* 1:295-305.

Brown, F. B. 1981. Vectorized Monte Carlo. Ph.D. thesis. Ann Arbor: University of Michigan, Department of Nuclear Engineering.

Brown, F. B. 1983. Vectorized Monte Carlo methods for reactor lattice analysis. In *Proc. topical meeting on advances in reactor computations*. La-Grange Park, Illinois: American Nuclear Society, pp. 108-123.

Brown, F. B. 1986. Vectorization of 3-D general-geometry Monte Carlo. *Trans. Amer. Nucl. Soc.* 53:283-284.

Brown, F. B., and Martin, W. R. 1985. Monte Carlo methods for radiation transport analysis on vector computers. *Progr. Nuclear Energy* 14:269-299.

Brown, F. B., and Mendelson, M. R. 1984. Vectorized Monte Carlo applications in reactor physics analysis. *Trans. Amer. Nucl. Soc.* 46:727-728.

Brown, F. B., Calahan, D. A., and Martin, W. R. 1981. Investigation of vectorized Monte Carlo algorithms. Ann Arbor: University of Michigan, Department of Nuclear Engineering.

Brown, F. B., Martin, W. R., and Calahan, D. A. 1981. Investigation of vectorized Monte Carlo algo-

rithms. *Trans. Amer. Nucl. Soc.* 39:755-756.

Bucher, I. Y., and Simmons, M. L. 1985. Performance assessment of supercomputers. LA-UR-85-1505. Los Alamos: Los Alamos National Laboratory, Computing and Communication Division.

Calahan, D. A., and Martin, W. R. 1980. Final report for preliminary studies on vectorized Monte Carlo for Los Alamos National Laboratory. Ann Arbor: University of Michigan, Department of Nuclear Engineering.

Chauvet, Y. 1984. Multitasking a vectorized Monte Carlo algorithm on the Cray-X/MP2. *Cray Channels* 6(3).

Chauvet, Y. 1985. Vectorization and multitasking with a Monte Carlo code for neutron transport problems. In *Proc. LANL-CEA meeting on recent applications of the Monte Carlo method*. CEA-CONF 7902.

Dongarra, J. J. 1986. Performance of various computers using standard linear equations software in a Fortran environment. Tech. Memo. 23. Argonne, Illinois: Argonne National Laboratory.

Fisher, K. J. 1986. Vectorized Monte Carlo radiation transport. LA-UR-86-3737. Los Alamos: Los Alamos National Laboratory, Computing and Communication Division.

Hockney, R. W., and Jesshope, C. R. 1981. *Parallel computers*. Bristol, Great Britain: Adam Hilger Ltd.

Los Alamos Monte Carlo Group. 1981. MCNP—A general Monte Carlo code for neutron and photon transport. LA-7396-M (revised). Los Alamos: Los



Alamos National Laboratory, Group X-6.

Martin, W. R. 1983. Vectorized Monte Carlo on the Cyber-205. Ann Arbor: University of Michigan, Department of Nuclear Engineering.

Martin, W. R., and Callahan, D. A. 1982. Vectorized Monte Carlo for non-linear radiation transport. *Trans. Amer. Nucl. Soc.* 43:399-400.

Martin, W. R., Nowak, P. F., and Rathkopf, J. A. 1986. Monte Carlo photon transport on a vector supercomputer. *IBM J. Res. Develop.* 30:193-202.

Martin, W. R., Poland, D., Wan, T. C., Mudge, T. N., and Abdel-Rahman, T. S. 1987. Monte Carlo photon transport on the NCUBE. In *Proc. second conf. on hypercube multiprocessors*. New York: SIAM, in press.

Martin W. R., Rathkopf, J. A., and Nowak, P. F.

1985. Vectorized Monte Carlo photon transport on the Cray-XMP. *Trans. Amer. Nucl. Soc.* 50:278-279.

Straker, E. W., Scott, W. H., and Byrn, N. R. 1970. The MORSE general purpose Monte Carlo multigroup neutron and gamma ray transport code with combinatorial geometry. USAEC Report ORNL-4585. Oak Ridge, Tennessee: Oak Ridge National Laboratory.

Troubetzkoy, E., Steinberg, H., and Kalos, M. 1973. Monte Carlo radiation penetration calculations on a parallel computer. *Trans. Amer. Nucl. Soc.* 17:260.

Wan, T. C., and Martin, W. R. 1986. Parallel algorithms for photon transport Monte Carlo. *Trans. Amer. Nucl. Soc.* 53:285-286.