Monte Carlo photon transport on a vector supercomputer

by William R. Martin
Paul F. Nowak
James A. Rathkopf

The use of "event-based" algorithms for particle transport Monte Carlo methods has allowed the successful adaptation of these methods to vector supercomputers. An alternative algorithm for the specific application of photon transport in an axisymmetric inertially confined fusion plasma has been developed and implemented on a vector supercomputer. The new algorithm is described; its unique features are discussed and compared with existing vectorized algorithms for Monte Carlo. Numerical results are presented illustrating its efficiency on a vector supercomputer, relative to an optimized scalar Monte Carlo algorithm that was developed for this purpose.

Introduction

The Monte Carlo method is a general approach to solving complex problems in science and engineering, wherein the physical phenomenon is simulated statistically by sampling from known probability distributions that describe the specific physical process. Thus, the Monte Carlo method has been used to simulate traffic flow, fluid flow, radiation transport, and quantum-mechanical systems. This paper discusses one specific type of Monte Carlo, particle transport Monte Carlo, which is used to simulate the transport of particles in a medium. Typical particle transport examples are electron transport in a semiconductor device and photon transport in an inertially confined fusion (ICF) plasma; the latter is the specific application discussed in this paper. There are several other examples of Monte Carlo methods, but this paper is restricted to particle transport Monte Carlo.

The basic idea of the Monte Carlo method for solving particle transport problems is to simulate the actual behavior of the particle by drawing random samples from appropriate probability distributions describing the actual physical process. Thus a source particle is randomly sampled from a known distribution by sampling an initial position $\vec{r}$ and velocity $\vec{v}$ for the particle (for a photon, one would sample the frequency ν and the direction of travel $\hat{\Omega}$).

Given the initial position and velocity of the particle and the properties of the medium in which the particle is traveling, the particle transport process is simulated by sampling from known probability distributions describing the interaction of the particle with the medium. The particle moves on straight lines between interactions and the simulation proceeds until the particle is "killed," such as by escaping from the system or being absorbed by the medium. The specific application of photon transport is described in more detail later.

Each of the particle simulations is called a "history," and by keeping track of the various outcomes ("tallies" or "scores"), one can obtain estimates of the various quantities of interest to the analyst, such as the escape probability or the absorption probability for particles in a given energy range and in a given region. Because the physical process is faithfully represented by the computer simulation, this is known as "analog" Monte Carlo. Modern-day Monte Carlo methods for particle transport are generally "non-analog"

because they contain variance-reduction schemes which allow the estimation of the quantities of interest with fewer histories. The vectorization approach is the same for either analog or non-analog Monte Carlo, and no distinctions are made between these types in the following discussion.

The advantage of Monte Carlo is its capability to treat any geometry or physical process, no matter how complex, assuming the geometry can be represented by mathematical equations and the physical process can be described by a probability distribution (which may have been experimentally or theoretically determined). The disadvantage is typically computation time—a substantial number of histories may be needed to yield reasonable statistics and thus may lead to excessive computing times. In addition, the inherent scalar nature of the conventional Monte Carlo algorithm has made it difficult to implement Monte Carlo methods on high-performance vector supercomputers such as the Cray-1 or CDC Cyber-205. However, recent advances in developing vectorized particle transport Monte Carlo algorithms have had a substantial impact on the cost of performing a Monte Carlo simulation. This paper discusses an alternative vectorized algorithm for a specific particle transport application which has been implemented on the Cray-1 and single-processor Cray-XMP vector supercomputer.

Vectorized Monte Carlo

The independence of the particle histories makes it impossible to vectorize the conventional Monte Carlo algorithm. Trying to follow a "vector" of particles would be fruitless because each component of the vector (i.e., each particle) follows a different path since the histories are all different. Hence, the "history-based" algorithm is not vectorizable. However, if instead of following "histories" the algorithm follows "events," then a vectorized algorithm can be formulated. In particular, an event is defined as a portion of a history that begins with the particle emerging at a position $\vec{r}$ with a velocity $\vec{v}$. This particle may have been the result of a scattering collision, it may have been emitted by a source, or it may be sitting on a boundary ready to be transported into the adjacent zone. As the event proceeds, the particle is tracked either to its next collision or to the next boundary, whichever is closer. The event terminates at the boundary or collision site, where the position and velocity of the continuing particle (if any) are known. These can then be used to initiate the next event. For time-dependent Monte Carlo, the event may be terminated by reaching the end of the current time step. Whereas all histories are different, all events are similar. The key to vectorizing Monte Carlo is to construct an "event-based" algorithm and process the particle vector for many events, continually updating the particle vector by eliminating deleted particles and adding new particles until the requisite number of simulations is performed.

Previous work

Brown and Martin [1] have vectorized multigroup neutron transport Monte Carlo in a general geometry. The term "multigroup" describes the cross-section database as well as the method for treating collisions. (The alternative to multigroup is "continuous-energy," wherein the energy of the particle is used directly to sample from probability distributions representing the collision physics.) Because of the difficulties in tracking particles in a general geometry with many dissimilar zones, the algorithm treated particles in only one zone at a time. This avoided the necessity to "gather" zonal data for the particle vector because all particles were in the same zone, but it did necessitate an outer loop over zones to process all of the particles. This "one-zone" algorithm yielded speedups of a factor of 30–35 on the CDC Cyber 205 versus an optimized scalar code on the CDC 7600. Brown [1, 2] subsequently developed a vectorized algorithm for continuous-energy Monte Carlo for reactor lattice geometry, where symmetry allowed the tracking of all particles simultaneously, irrespective of zone. This "all-zone" algorithm resulted in speedups of a factor of 20–85 on the Cyber 205, compared with the old production code on the CDC 7600.

The most recent work to vectorize Monte Carlo is that of Bobrowicz et al. [3], which considered the specific application of photon transport in a 2D axisymmetric geometry. In this case the zones are all similar, being bounded in general by four conic sections. This zonal symmetry allows an "all-zone" algorithm, but the overall algorithm is very different from the preceding one. Separate stacks (or "queues") of particles are constructed in accordance with what is to happen to the particles next (track to boundary, perform Thomson scatter, etc.). Eight stacks are utilized and a stack is executed if its length becomes 64—the length of the Cray vector registers. Hence this algorithm is optimized for the Cray architecture. Speedups in the range of 8–10 times on the Cray-1 versus the old scalar code on the Cray-1 are reported.

It is difficult to compare the results of the various efforts because the speedups are problem-sensitive as well as being sensitive to the efficiency (or inefficiency) of the original scalar code.

The present work is also devoted to the vectorization of photon transport in a 2D axisymmetric geometry. As will be seen, however, there are few similarities between the present approach and the algorithm of Bobrowicz et al. [3], although there are some common features with the earlier algorithms of Brown and Martin [1] and Brown [2].

Specific application

The Monte Carlo algorithm developed for this research is intended to analyze the transport of photons in a high-density, high-temperature plasma in an axisymmetric geometry. The material zones (each with its own

composition, temperature, and density) are simple volumes of revolution, bounded by at most four surfaces, each of which is a conic section. A typical mesh is illustrated in **Figure 1**. This relatively simple geometry allows the use of the "all-zone" algorithm in which all particles can be treated simultaneously, irrespective of zone. Although Fig. 1 depicts a uniform mesh, the algorithm assumes a general two-dimensional r-z mesh. Photons are sampled uniformly and isotropically within each zone from a Planckian energy spectrum ("blackbody" emission). The number of photons emitted within each zone is a function of the zone material properties. The photons which are emitted are then followed through the plasma until they are absorbed, have escaped, or reach census (end of time step). Besides absorption, the photons may undergo Thomson scattering. The photon transport calculation would normally alternate with a hydrodynamic calculation which updates the material properties of the zones, as well as the geometry of the zones. However, the present simulation includes only the within-time-step Monte Carlo calculation. A realistic database consisting of frequency-, temperature-, and density-dependent opacities for hydrogen, deuterium, tritium, silicon, and oxygen was obtained from Lawrence Livermore National Laboratory. This database returns an opacity, given the photon frequency and the zone composition, temperature, and density. The product of the opacity and density yields the macroscopic total cross section for the zone in question, which is the probability per unit length that the photon suffers a collision in the zone. The macroscopic cross section is needed to sample a distance to collision within a given zone. The macroscopic total cross section has units of cm^{-1}.

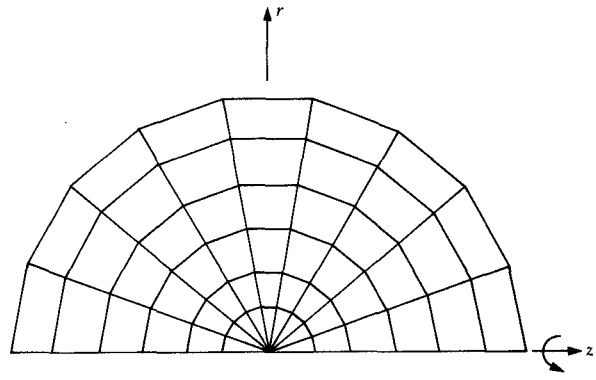

Figure 1

Typical 2D (r-z) axisymmetric mesh for a spherical configuration; r = radial coordinate (distance from z-axis), z = axial coordinate.

Conventional (scalar) Monte Carlo algorithm

The conventional Monte Carlo approach to solving this problem would be to have an outer loop over the zones to generate the photons, which are followed one at a time as soon as they are produced until they have escaped, have been absorbed, have reached the end of the current time step (census), or have otherwise been eliminated from the problem. This is known as a "history-based" algorithm, where a "history" corresponds to following a photon from birth (blackbody emission) to death (escape, absorption, or census). Within a given zone, a photon is sampled by determining its initial position $\vec{r}$, frequency ν, and direction of travel $\hat{\Omega}$. The position $\vec{r}$ is sampled uniformly within the zone, the frequency is sampled from the Planckian spectrum, and the direction of travel is sampled uniformly on the unit sphere (isotropic emission). A distance to collision d^C is sampled from the exponential distribution describing the penetration of photons of frequency ν in the specific material composition contained within the zone. Since zones can be of different composition, if the sampled distance to collision is greater than the distance to the boundary of the zone, the photon must be advanced to the boundary and a new distance to collision d^C sampled in the new zone. Therefore, the distance to the nearest zone boundary d^B along the photon trajectory must be calculated. In addition, for time-dependent problems, a distance to census d^{CEN} must be calculated, since the photon history must be terminated at the end of the time step (and saved for the next time step).

If the photon suffers a collision, the type of collision is sampled: For this application it may be either an absorption or a Thomson scattering collision. If absorption, the history is terminated, appropriate tallies are made, and a new history is initiated. If scattering, the angle of scattering is sampled and the new direction of travel is calculated. New distances to boundary and census are calculated, a distance to collision is sampled, and the history proceeds as before.

If the photon crosses a boundary, the photon is advanced to the boundary along its flight path and a test is made to see if the boundary is an outer boundary. If so, a tally for the escaping particles is incremented and a new history is initiated. If the boundary is an internal boundary, the zone index is updated and the photon is tracked through its next event (collision, boundary crossing, or census).

The history continues until the photon is gone. The overall simulation continues until the loop over zones has been completed and all of the photons have been emitted and followed. **Figure 2** is a flowchart for the scalar algorithm. A separate Monte Carlo code (SPHOT) based on this algorithm has been developed for comparison with the vectorized algorithm; that algorithm is discussed next.

Vectorized Monte Carlo algorithm

Since this is an "event-based" algorithm, a vector of particle "descriptors" must be constructed and then processed on an

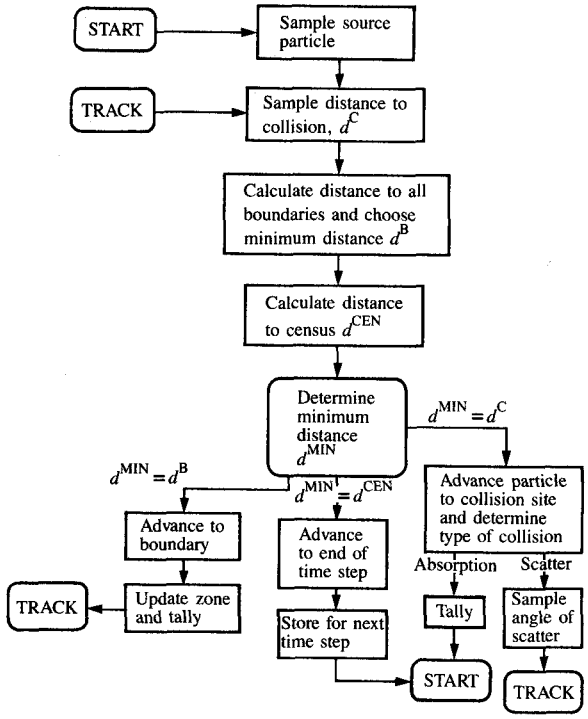

Figure 2

Flowchart for scalar Monte Carlo algorithm.

event basis. For the current application, twelve descriptors are used to describe each photon, as illustrated in **Figure 3**. The basic approach is then to process this "particle vector" until all of the photons are emitted and followed to termination. At this point the various tallies (or "scores") are accumulated and the problem is finished. A summary of the steps in the vectorized algorithm follows:

- *Initialization*

1. Read input data to set up geometrical mesh and specify zone compositions, temperatures, and densities. Also read in the maximum number of particles N to be included in the particle vector. Typically, $N = 2000$.
2. Using the zone compositions, temperatures, and densities, calculate the total cross section S_g^i for zone i and photon energy group g. This cross section is constant over the simulation time step. For the purpose of the following discussion, we denote this array as $\underline{S}$, which has length $M \times G$, where M = the number of zones and G = the number of photon energy groups. For our application, $G = 12$ and $M = 1960$.
3. Determine the (z, r) coordinates of the vertices of each surface S bounding zone i. For zones along the z-axis which have only two or three bounding surfaces, fictitious coordinates are included for the "missing" sides to allow the vectorization of the distance to boundary calculation. These fictitious vertices are specified to be outside the problem geometry, hence never leading to a valid intersection. This array of 8 coordinate pairs for each zone is denoted by $\underline{R}$, which has length $8M$.
4. Load the particle vector $\underline{\Gamma}^{(0)}$ with descriptors for N particles, determined by sampling in the source routine.
5. Load the source buffer $\underline{X\Gamma}$ with descriptors for N_X particles, where N_X is the number of particles stored in the source buffer, and the source routine is used to fill $\underline{X\Gamma}$.

- *Simulation*

For event $n = 1, 2, \cdots$

1. Fetch $\underline{\Gamma}^{(n-1)}$, the particle vector at the end of the previous event iteration. $\underline{\Gamma}^{(0)}$ is the initial particle vector.
2. Gather a vector of total cross sections

   $$\underline{\Sigma} = \text{col}[\Sigma_1, \Sigma_2, \cdots, \Sigma_N]$$

   from $\underline{S}$,

   $$\underline{\Sigma} \leftarrow \underline{S},$$

   where $\underline{S}$ is the cross-section array tabulated by zone-group index and $\underline{\Sigma}$ is the cross-section array tabulated by particle. The zone and group indexes for particle i are two of the twelve descriptors constituting the particle descriptor vector $\underline{\Gamma}^{(n-1)}$. Note that the cross-section data must be arranged by particle in order to vectorize the calculations that are to come.
3. Similarly, gather a vector of (z, r) vertices for each of the bounding surfaces,

   $$\underline{\gamma} \leftarrow \underline{R},$$

   where $\underline{R}$ contains the vertices tabulated by zone and $\underline{\gamma}$ contains the vertices tabulated by particle.
4. Using the vector $\underline{\Sigma}$, sample a distance-to-collision vector

   $$\underline{d}^C = \text{col}[d_1^C, d_2^C, d_3^C, \cdots, d_N^C],$$

   where d_i^C = distance to collision for particle i. It is easy to show that

   $$d_i^C = -\frac{1}{\Sigma_i} \ln \xi$$

   is a distance to collision sampled from the exponential distribution, where Σ_i is the macroscopic total cross section for the zone which contains particle i, and ξ is a random number on [0, 1].
5. Using the vector of vertices $\underline{\gamma}$, calculate the minimum distance-to-boundary vector $\underline{d}^B$, where d_i^B is the minimum distance to a boundary enclosing particle i (along the particle flight path). This calculation, while

straightforward (the intersection of a straight line with a conic section), may consume the bulk of the computational time in a typical Monte Carlo simulation.

6. Calculate the distance-to-census vector $\underline{d}^{CEN}$, where d_i^{CEN} = distance traveled by particle i at the end of the current time step.
7. Determine the minimum distance-to-event vector,

$$\underline{d}^{MIN} = \text{col}[d_1^{MIN}, d_2^{MIN}, \cdots, d_N^{MIN}],$$

where

$$d_i^{MIN} = \min[d_i^C, d_i^B, d_i^{CEN}].$$

8. Update the particle position coordinates in

$$\vec{r}_i \leftarrow \vec{r}_i + \hat{\Omega}_i d_i^{MIN},$$

where $\vec{r}_i$ = position of particle i and $\hat{\Omega}_i$ = direction of travel of particle i.

The above steps are performed in a strict sequential order and involve the entire particle vector $\underline{\Gamma}^{(n-1)}$. The following steps are performed on specific elements of the particle vector; they are controlled by logic vectors (bit vectors) which signal whether the indicated operation is to occur.

9. If the particle suffers a collision, sample the type of collision (either absorption or Thomson scatter). If the photon is absorbed, tag it for deletion from $\underline{\Gamma}^{(n-1)}$.
10. If the photon undergoes Thomson scattering, update the photon direction of travel:
    a. Gather up a vector of direction cosines

    $$\underline{\alpha} \leftarrow \underline{\Gamma},$$

    where $\underline{\alpha} = \text{col}[\alpha_1, \alpha_2, \cdots, \alpha_{NTHOM}]$ and $\underline{\alpha}_j = \text{col}[u_j, v_j, w_j]$. The terms u_j, v_j, w_j are the direction cosines for the jth particle to be Thomson-scattered, and *NTHOM* is the number of particles to be Thomson-scattered. Note that only the Thomson-scattered photons are to be analyzed, not all of the particles in $\underline{\Gamma}^{(n-1)}$.
    b. Sample angles of scattering $\underline{\Theta}$ from the Thomson angular distribution for the *NTHOM* particles and determine the new direction cosines $\underline{\alpha}'$ for the scattered particles.
    c. Update the particle vector by scattering the new direction cosines $\underline{\alpha}'$ back into the particle vector $\underline{\Gamma}^{(n-1)}$.
11. If the particle crosses a boundary, check to see if it is an outer boundary. If so, tag particle for deletion from $\underline{\Gamma}^{(n-1)}$. If not, update the zone index.
12. If the particle is at the end of the time step (census), tag it for deletion from $\underline{\Gamma}^{(n-1)}$.
13. Advance the particle vector to the beginning of the next event,

$$\underline{\Gamma}^{(n-1)} \rightarrow \underline{\Gamma}^{(n)},$$

by either

a. Scattering new particles from the source buffer into the tagged locations of $\underline{\Gamma}^{(n-1)}$ (see **Figure 4**), or
b. Compressing $\underline{\Gamma}^{(n-1)}$ by deleting the tagged particles and re-indexing $\underline{\Gamma}^{(n-1)}$. The compress step is performed if there are not enough particles in the

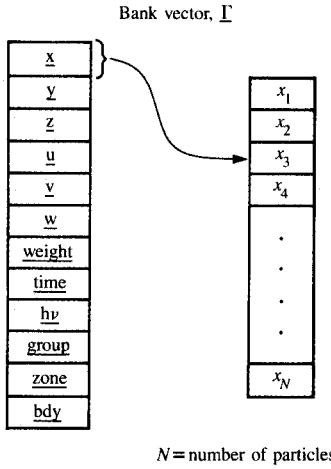

Figure 3

Particle bank vector.

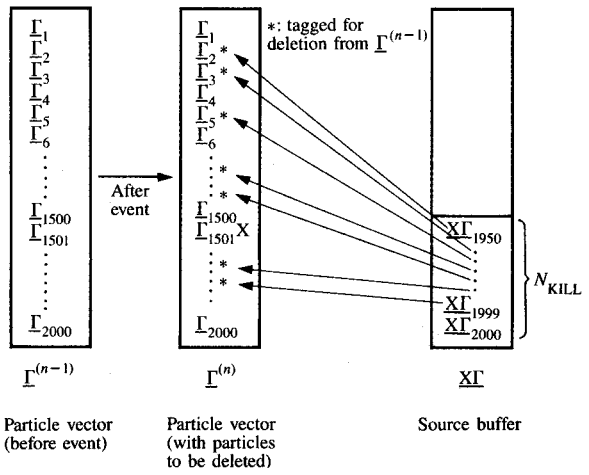

Figure 4

Use of source buffer to replenish particle vector via scatter operation.

source buffer $\underline{X}\underline{\Gamma}$ to replace the tagged particles in $\underline{\Gamma}^{(n-1)}$. In that case, $\underline{X}\underline{\Gamma}$ is refilled for the next event iteration. For the scatter fill, the length of the particle vector remains constant.

14. Repeat steps 1 through 13 until the length of $\underline{\Gamma}$ is zero and the source particles are depleted.

Comparison with existing algorithms

The present algorithm is similar in many respects to the algorithm of Brown [1, 2] for the analysis of nuclear reactor lattice problems on a CDC Cyber 205. Both algorithms are "event-based" (as are all vectorized Monte Carlo algorithms for particle transport) and "all-zone," because all particles (in the current particle vector) are tracked simultaneously, regardless of their zone. However, Brown utilizes three separate particle stacks: one for particles being tracked to a boundary, another for particles undergoing collision analysis, and the third for particles outside the current energy group. Thus the particle data must be moved from one stack to another during the simulation. Our approach employs one main stack (the particle vector) which is processed in a manner similar to the way a scalar algorithm processes a single particle history. That is, a strict sequence of decisions and calculations is performed, but in a vector manner, and logical control vectors are used to determine the final states of the individual particles in the main stack. For those calculations which are performed for only a subset of the particles, such as the Thomson scattering transformation of the velocity vector, a substack is created using gather statements. These substacks contain the minimum amount of data needed for the particular calculation. For example, the Thomson scattering transformation requires only the direction cosines of the affected particles; therefore, the equivalent of three gather statements is performed to create a vector of direction cosines which can be transformed via the Thomson transformations to a new set of direction cosines. These are then scattered back into the main stack with a scatter operation, hence replacing the old direction cosines.

There is very little similarity to the algorithm of Bobrowicz et al. [3] (other than the actual physical application) because they employ many stacks, partitioning the computation into relatively small parcels. In some sense the present approach constructs these stacks implicitly as opposed to the explicit construction (and execution) of the stacks in the Bobrowicz scheme. In our algorithm, for example, the particle vector $\underline{\Gamma}$ follows a predetermined path through the various calculational kernels (distance to boundary, distance to collision, collision analysis, boundary crossing analysis, etc.), whereas the Bobrowicz scheme would have partitioned $\underline{\Gamma}$ into subvectors which would be dispatched to their appropriate queues. These queues would be processed if their length reached 64. There is one similarity, however, in the two algorithms. In particular, the treatment of Thomson scattering (or any other special physics option) in the present scheme is not unlike the Bobrowicz approach, because an explicit stack is constructed (or gathered) from the main stack. However, in our approach this substack is executed in its turn once every event iteration, regardless of its length. It is conceivable that some benefit might be realized if the substack were held until it became reasonably long. This would result in longer vector lengths, but at the penalty of having to save all of the particle descriptors in a separate queue and having to remove them from the main stack. This would complicate the algorithm somewhat, but more importantly, would necessitate moving all twelve of the particle descriptors rather than just the three descriptors needed to perform the Thomson analysis. This trade-off has not been evaluated but is currently under investigation.

Another unique feature of the present approach is the use of the source buffer to fill the particle vector after every event iteration. It is true that using a scatter operation to replace deleted particles in $\underline{\Gamma}$ is in general no more efficient (and perhaps less efficient) than compressing the main stack to eliminate the particles tagged for deletion; however, the logic is somewhat simpler and the source buffer can be used to accommodate other sources in the problem, such as those due to fission or splitting, because particles generated within the simulation can be stored in the source buffer, no matter where they come from.

Results

The vectorized algorithm described here has been incorporated into the FORTRAN code VPHOT, the present version of which is intended for the Cray-1 and Cray-XMP computers. The library STACKLIBE [4] is utilized extensively throughout the code due to the necessity to perform numerous gather and scatter operations. Because a significant portion of the gather/scatter operations involve multiple calls with the same index list, the "array" gather/scatter routines in STACKLIBE were quite useful and resulted in significant improvements in overall run time. For example, to obtain the zonal vertices by particle requires 16 gather operations, but each of these utilizes the same index list. The one STACKLIBE routine QARGATH performs these with one call, utilizing the ability of the Cray to fetch a vector with a constant stride.

The specific problem analyzed with the vectorized code VPHOT is an ICF plasma consisting of a 50%–50% mixture of deuterium and tritium (D-T) at elevated temperature and density, surrounded by a SiO_2 region, also at elevated temperature and density. Photons are emitted throughout both regions via Planckian emission, and the problem parameters are such that approximately 240000 photons (actually photon "bundles") are emitted during the simulation. The problem configuration is illustrated in **Figure 5**. In addition to keeping track of the number of absorptions and the number of photons escaping the plasma,

Table 1 Numerical results—ICF test problem.

	LLNL	*VPHOT*	*SPHOT*
Histories	240,000[3]	238,274	238,269
Escaped	62,163	62,719	62,757
Census	3,033	3,189	3,153
Lost[1]	248	11	7
Tracks	4,200,000[3]	4,192,757	4,195,966
CPU time[2] (s)	336.8	30.5	139.9
Timing (μs/track)	80.3	6.97	33.3

[1] A particle is considered "lost" if it is not in the correct zone (zone index is inconsistent with its x, y, z location).
[2] Cray XMP/2 (single processor only)—no I/O.
[3] These quantities are estimates, since they were not available from the LLNL code.

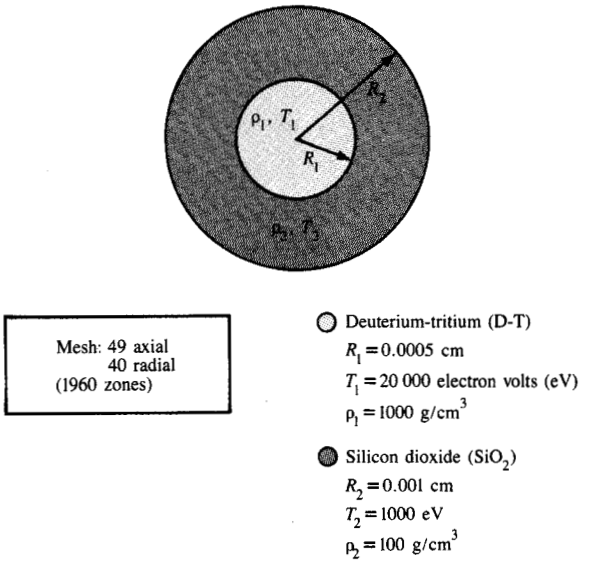

Figure 5

Configuration for ICF test problem (two concentric spheres).

the code tabulates the number of photons escaping within each energy group (the escaping "spectrum") as well as the total number of events (or "tracks") processed during the simulation. This latter quantity is useful for arriving at an absolute measure of efficiency—the average CPU time (in μs) needed to process one track. In addition, a scalar version (SPHOT) of the vectorized code VPHOT was created to allow a fair comparison of the efficiency of the vectorized algorithm. In addition to the need for a meaningful comparison to show the efficiency of VPHOT, there was a need to compare the accuracy of VPHOT to ensure that the physics was being predicted correctly. This was done by comparing the VPHOT and SPHOT results with results from a reference (scalar) Monte Carlo code at Lawrence Livermore National Laboratory (LLNL). This code was run on the same problem (identical mesh, database, source, etc.), and the results compared with the prediction of VPHOT and SPHOT.

Table 1 contains a summary of the overall tallies and timings for the reference LLNL Monte Carlo code as well as VPHOT and SPHOT. As can be seen, the overall tallies are predicted well within statistical error for all three codes. **Table 2** presents the escaping photon spectrum for the three codes, and again the agreement is excellent. From these results we have concluded that the scalar and vector codes SPHOT and VPHOT are correct to the extent that they predict essentially the same physics as does the reference LLNL code.

The efficiency of the vectorized code is evident in Table 1, where the vector code is seen to be faster by a factor of 4–5 than the scalar code SPHOT and nearly a factor of 12 faster than the reference LLNL scalar code. Although one must treat the comparison of VPHOT and the LLNL code somewhat carefully, the comparison with SPHOT is significant because SPHOT was developed to include the same physics as VPHOT and the algorithm was optimized for a scalar processor. Thus one can expect a speedup on the order of 4–5 for the Cray-1 and single-processor Cray-XMP supercomputers. This would correspond to speedups in excess of 10 when compared with a CDC 7600, which was an earlier measure of performance [1, 2]. This should be compared with the factor of 2–3 that would be expected for a scalar algorithm, such as conventional Monte Carlo, when converting from the CDC-7600 to the Cray-XMP (single processor). This performance gain is simply due to the increase in the scalar processing speed.

Table 2 Escaping photon energy spectrum.

Photon energy group[1]	*Escaping photon energy*[2]	
	LLNL	*VPHOT*
1	0.0345	0.0371
2	0.2398	0.2207
3	1.065	1.223
4	2.383	2.365
5	1.783	1.773
6	1.978	2.015
7	5.381	5.593
8	9.716	9.867
9	26.02	25.96
10	62.45	63.09
11	80.07	80.09
12	3.439	3.446
All groups	194.56	195.67

[1] The photons range in energy from 1 electron volt (eV) to 200 keV (200 000 eV). This range is divided into twelve energy groups.
[2] Total escaping photon energy is in units of 10^{15} keV (by energy group).

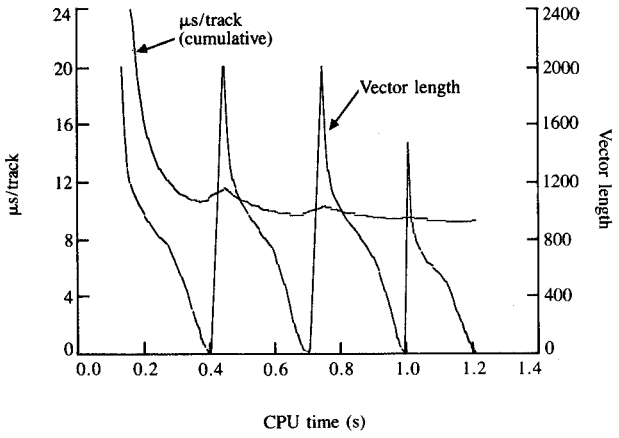

Figure 6

Timing curves for step-refill scheme.

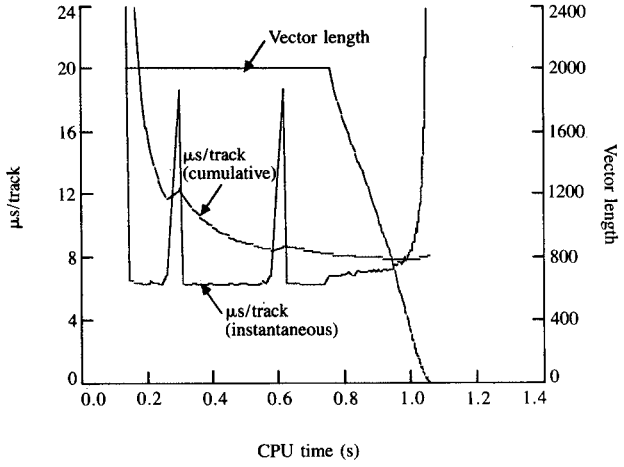

Figure 7

Timing curves for continuous-refill scheme.

In addition to keeping track of the overall timing statistics, VPHOT is instrumented to allow output of the instantaneous timing (in μs/track) as well as the cumulative timing at each event iteration. The instantaneous timing is simply the CPU time per track during the current event iteration, and the cumulative timing is the average CPU time per track up to that point in the overall simulation. This time is calculated by determining the number of tracks and dividing this number into the total CPU time for the simulation, excluding input/output operations. Since some event iterations involve calling the source routine to replenish the source buffer, the instantaneous timing will show a marked increase during these event iterations. In **Figures 6** and **7** we have plotted the timing curves for a small test problem (7461 source photons) for two different replenishment schemes:

1. Step refill—the particle vector is depleted to zero before being refilled. Thus the particle vector is initially 2000 long and it decreases to zero before being refilled to 2000. This continues until the 7461 photons are processed.
2. Continuous refill—the particle vector is refilled after each event iteration by scattering from the source buffer (scatter fill). This is the current approach in VPHOT.

Figures 6 and 7 plot the cumulative and instantaneous timing as well as the instantaneous vector length as a function of event iteration. As expected, the timing improves with increasing vector length. In **Figure 8** we have plotted the cumulative timing curves for the two replenishment schemes on the same scale. It is evident that the continuous-refill scheme is to be preferred. It results in longer vector lengths on the average since there is only one reduction to a zero vector length. A variation on the step-refill scheme was examined wherein the particle vector was not allowed to go below some minimum length specified by the user. If it does fall below this minimum, it is refilled from the source routine. It was found that the optimal minimum particle vector length was 1750 for a maximum vector length of 2000, which is the usual case. However, for this optimal minimum length, the overall timing was about the same as with the continuous-refill scheme. Since the logic was much simpler with the continuous scheme, it was chosen for the final algorithm.

Figure 7 illustrates another point regarding the overall performance. Since there is no provision in VPHOT to switch to a scalar algorithm when the vector length becomes small (e.g., less than 5), there is a substantial increase in the instantaneous CPU time per track towards the end of the overall simulation. During this "end game" the vector length may be 1 or 2 for many event iterations, resulting in inefficient code due to the vector constructs that are utilized: for example, the instantaneous timing peaks at 300–400 μs/track, which is nearly ten times slower than the scalar code. However, the interesting observation is that this relatively slow portion of the simulation has a negligible effect on the overall timing, as can be seen by the slight increase in the cumulative timing over this portion of the simulation. The reason for this is that the bulk of the simulation is performed with relatively long vectors and the effect of the short-vector iterations is not important. This leads one to conclude that the "end game" is not important for the overall performance and that one should not worry about schemes to avoid the short vectors, such as switching to a scalar code when the vector length becomes short.

Besides complicating the algorithm, the addition of separate scalar code requires one to maintain two versions of the same code, which is not advisable, especially if the code is to be modified frequently.

A more difficult problem is the comparison with other results, such as those of Bobrowicz et al. [3]. One possible measure is the absolute indicator of efficiency, given by the average CPU time to process one track. Table 1 gives approximately 7 μs/track for VPHOT, versus 33 μs/track for SPHOT and in excess of 80 μs/track for the reference LLNL code. Bobrowicz [3] reports 25 μs/track on the Cray-1, which corresponds to 16–18 μs/track on the Cray-XMP. However, his Monte Carlo algorithm includes different physics options; hence this comparison must be taken with some caution. Brown [5] reports approximately 4 μs/track for the reactor lattice problems on the CDC Cyber-205, but again the different application and different computer make it difficult to make a fair comparison of the relative efficiencies of the various algorithms.

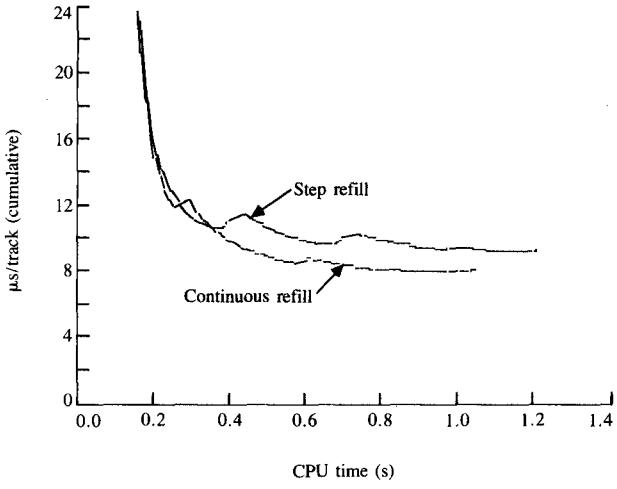

Figure 8

Cumulative timing curves for step-refill vs. continuous-refill schemes.

Summary and conclusions

A vectorized Monte Carlo code (VPHOT) has been developed for the analysis of photon transport in an ICF plasma with axisymmetric geometry. A companion scalar code (SPHOT) has also been developed for comparison with the vector code. Both VPHOT and SPHOT are capable of performing realistic simulations, representative of typical ICF fusion calculations, and they have been verified by comparison with a reference Monte Carlo code at LLNL. The vectorized code is approximately four to five times faster than the scalar code and nearly twelve times faster than the reference LLNL scalar code. The vectorized code also compares well with the alternative algorithm of Bobrowicz et al. [3], although the comparison is not conclusive. The optimal algorithm may well be a compromise between these two approaches. However, the present algorithm has several advantages over the Bobrowicz approach. The present algorithm is somewhat simpler and has many similarities to the scalar Monte Carlo algorithm, hence making it easier to make the transition to a vectorized code. Second, it is relatively straightforward to add additional physics options to the present code because there is no complicated synchronization among the various stacks, as in the Bobrowicz scheme. Finally, it minimizes the movement of data because the same particle stack is processed for most of the simulation. Although this may have the effect of a shorter vector length, it is our opinion that the minimization of the data movement is the more important issue. However, this issue is still under examination.

Future effort

Effort is now under way to implement a multiprocessed version of VPHOT on the Cray-XMP/48 at LLNL. In addition, a parallel effort is under way to implement a multiprocessed version of the scalar code SPHOT on the four-processor IBM 3084, and it is expected that this latter effort will be extended to massively parallel architectures in the near future.

Acknowledgments

The authors acknowledge helpful comments and suggestions from William Chandler, Rollin Harding, Frank McMahon, and David Hardin of Lawrence Livermore National Laboratory (LLNL) and Bernie Rudin of the IBM Kingston laboratory. This work was partially supported by grants from IBM (Kingston) and LLNL (under Contract W-7405-ENG-48 with the U.S. Department of Energy).

References

1. F. B. Brown and W. R. Martin, "Monte Carlo Methods for Radiation Transport Analysis on Vector Computers," *Prog. Nucl. Energy* **14**, 269–299 (1984).
2. F. B. Brown, "Vectorized Monte Carlo Methods for Reactor Lattice Problems," *Proceedings of the American Nuclear Society Topical Meeting on Advances in Reactor Computations*, Salt Lake City, UT, 1983, pp. 108–123.
3. F. W. Bobrowicz, J. E. Lynch, K. J. Fisher, and J. E. Tabor, "Vectorized Monte Carlo Photon Transport," *Parall. Comput.* **1**, 295–305 (1984).
4. F. H. McMahon, "STACKLIBE—A Library of Fast Vector Functions for Complete Vector Formulation of Program Logic on the Cray-1," draft version of manuscript, Lawrence Livermore National Laboratory, Livermore, CA, 1980.
5. F. B. Brown, Knolls Atomic Power Laboratory, Schenectady, NY, private communication, April 1985.

Received July 29, 1985; accepted for publication October 28, 1985

William R. Martin *University of Michigan, Department of Nuclear Engineering, Ann Arbor, Michigan 48109.* Dr. Martin is an associate professor of nuclear engineering at the University of Michigan. He received his Ph.D. in nuclear engineering from the University of Michigan in 1976. Upon graduation he joined Combustion Engineering Inc. and was responsible for developing advanced computational methods for nuclear reactor analysis. He returned to the University of Michigan in 1977 and has been active in computational methods development in several areas, including reactor physics, thermal/hydraulics, and particle transport. His current research interests include the development of algorithms for scientific computation on advanced computers, including Monte Carlo simulation on vector supercomputers and parallel processors, nuclear reactor plant simulation on parallel processors, and logic simulation on vector supercomputers. Dr. Martin is a member of the American Association for the Advancement of Science, the American Nuclear Society, the American Physical Society, the Association for Computing Machinery, Sigma Xi, and the Society for Industrial and Applied Mathematics.

Paul F. Nowak *University of Michigan, Ann Arbor, Michigan 48109.* Mr. Nowak is a graduate student in the Department of Nuclear Engineering at the University of Michigan, where he received his B.S.E. degree in 1984 and his M.S.E. degree in 1985. He is currently working on Monte Carlo algorithms for vector computers with single processors and multiprocessors, with application to photon transport in inertially confined plasmas. He is also interested in the development of nodal transport and diffusion methods for purposes of nuclear reactor design and analysis. Mr. Nowak is a member of Alpha Nu Sigma, the American Nuclear Society, and Tau Beta Pi.

James A. Rathkopf *Westinghouse Electric Corporation, Pittsburgh, Pennsylvania 15235.* Dr. Rathkopf received his B.S.E. degree in nuclear engineering at the University of Florida, Gainesville, in 1979 and his Ph.D. in nuclear engineering from the University of Michigan in 1984. While a graduate student, he spent several summers at Lawrence Livermore National Laboratory studying the application of Monte Carlo methods on vector supercomputers. Since receiving his Ph.D., he has been with the Nuclear Fuels Division of Westinghouse Electric Corporation, where he is developing advanced computational methods for the analysis of nuclear reactor cores. His technical interests include the development of methods for scientific computation on vector and parallel processors, real-time simulation of nuclear reactor core behavior, and deterministic and probabilistic methods of nuclear analysis. Dr. Rathkopf is a member of the American Nuclear Society.