MONTE CARLO PHOTON TRANSPORT ON SHARED MEMORY AND DISTRIBUTED MEMORY PARALLEL PROCESSORS

William R. Martin, Tzu-Chiang Wan, Tarek S. Abdel-Rahman, and Trevor N. Mudge

UNIVERSITY OF MICHIGAN
ANN ARBOR, MICHIGAN 48109

Kenichi Miura

SUBJECT AREA EDITOR

Summary

Parallelized Monte Carlo algorithms for analyzing photon transport in an inertially confined fusion (ICF) plasma are considered. Algorithms were developed for shared memory (vector and scalar) and distributed memory (scalar) parallel processors. The shared memory algorithm was implemented on the IBM 3090/400, and timing results are presented for dedicated runs with two, three, and four processors. Two alternative distributed memory algorithms (replication and dispatching) were implemented on a hypercube parallel processor (1 through 64 nodes). The replication algorithm yields essentially full efficiency for all cube sizes; with the 64-node configuration, the absolute performance is nearly the same as with the CRAY X-MP. The dispatching algorithm also yields efficiencies above 80% in a large simulation for the 64-processor configuration.

The International Journal of Supercomputer Applications, Vol. 1, No. 3, Fall 1987, pp. 57–74.

Introduction

The Monte Carlo method has been utilized for years to solve complex problems in particle transport applications, such as the transport of neutrons in a nuclear reactor shield or the transport of photons in an inertially confined fusion (ICF) plasma. The unique advantage of the Monte Carlo method is its capability to simulate almost any particle transport problem, independent of the complexity of the geometry or the transport process, as long as the geometrical configuration can be described mathematically and the physical process can be represented by probability distributions, which may be determined by theory and/or measurement. The usual complaint against the Monte Carlo method is that the time required to simulate the requisite number of particle histories may be exorbitant on conventional computers. Modern-day computer architectures provide some relief —the natural parallelism of Monte Carlo would seem tailor-made for parallel processors, including massively parallel architectures such as hypercube parallel processors. This paper investigates the development and implementation of photon transport Monte Carlo algorithms on both shared memory and distributed memory parallel processors, including a vector/parallel architecture. Actual timing results for both the IBM 3090/400 and the NCUBE hypercube parallel processor are discussed along with predicted results for a hypothetical CRAY X-MP configuration with up to eight processors.

Results and Discussion

1. BACKGROUND

The vectorized photon transport Monte Carlo code VPHOT (Martin, Nowak, and Rathkopf, 1986) was originally developed to assess whether or not this application could be successfully adapted to a vector supercomputer. To provide a comparative tool, the SPHOT code was created, which solves the identical problem as VPHOT but with the conventional (inherently scalar) Monte Carlo algorithm. The VPHOT and SPHOT Monte Carlo codes solve the specific problem of photon transport in an ICF plasma, as depicted in Figure 1. The geometry is constrained to be a two-dimensional z-r mesh with azimuthal symmetry, and individual zones are in general quadrilaterals of revolution, as illustrated in Figure 2. The article by Martin et al. (1987) presents a detailed description of the capabilities of these codes as

"The unique advantage of the Monte Carlo method is its capability to simulate almost any particle transport problem, independent of the complexity of the geometry or the transport process, as long as the geometrical configuration can be described mathematically and the physical process can be represented by probability distributions."

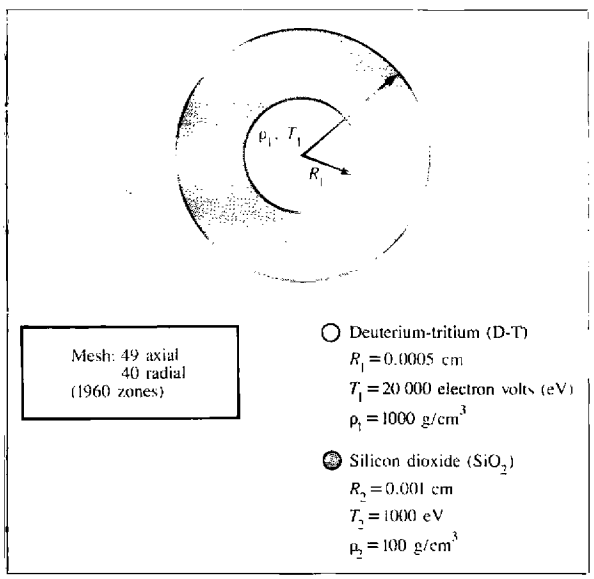

Fig. 1 Configuration for ICF application Copyright 1986 by International Business Machines Corporation; reprinted with permission.

well as a description of the physics, the Monte Carlo methodology, and the data base which describes the interaction of photons with the ICF plasma. The VPHOT results have demonstrated that this Monte Carlo application can be successfully vectorized, with speedups in the range of 5 to 6 on the CRAY X-MP versus the optimal scalar algorithm on the same computer. There has been substantial progress in recent years in vectorizing particle transport Monte Carlo applications other than photon transport, and the interested reader is referred to the recent review paper on this topic (Martin and Brown, 1987).

2. CONVENTIONAL MONTE CARLO ALGORITHM

The scalar algorithm contained in SPHOT is depicted in Figure 3. It is similar to the conventional Monte Carlo algorithm found in production-level codes such as MCNP (Los Alamos, 1981) or MORSE (Straker, Scott, and Byrn, 1970), where particles are emitted one at a time from a known source distribution and followed until "terminated" by, for example, absorption, escape from the geometry, or the end of the current time step (census). For the specific case of photon transport in an ICF plasma, the photons are created within a specific zone via Planckian emission and are followed throughout the plasma, which can either absorb the photons or scatter them (Thomson scattering).

3. PARALLEL MONTE CARLO: GENERAL REMARKS

Monte Carlo particle transport is an inherently parallel application if the particles do not interact with one another or change the background medium with which they are interacting. This is known as a linear transport process (Duderstadt and Martin, 1979), and many transport applications fall in this category. For example, neutron transport in a nuclear reactor shield involves the simulation of neutrons moving in the shield, and each neutron's history is independent of the others'. Even in cases where the histories might interact, such as where the particles might change the medium (e.g., change its temperature), this nonlinearity can be handled in such a way as to allow the simulations to be performed independently. The photon transport application we discuss here is in this category, and time steps are used to remove the dependence of the medium on the photon

transport process. That is, within a time step, the photons are treated as independent particles, and the medium does not change during the time step. Between time steps the properties of the medium may be changed to account for photon interactions during the previous time step. The linking of time steps is of course essential for a real production code but is not important for assessing the efficiency of the Monte Carlo algorithms, since the bulk of the computational time is usually spent in the within-time-step problem—performing the particle transport simulation.

Given that the photon transport application considered here is inherently parallel because each photon history is independent, one would clearly design a parallel algorithm to partition the photons among the processors. There are many ways to perform this partitioning, some more suitable than others. Before we consider the specific partitioning used for our application, let us consider two alternative approaches, partitioning by geometry and partitioning by photon energy. Both of these approaches can be described as "domain decomposition" methods, since the partitioning involves splitting up the domain, either space or energy (or both), and assigning regions to specific processors.

Partitioning by geometry entails assigning geometric regions to each processor, which would follow only those photons inside the region, until they left the region, were absorbed, or reached the end of the time step. The problem with this approach is twofold: load-balancing and communications. It is difficult to ensure load-balancing without some a priori knowledge of the problem, because the photons may tend to congregate in certain regions of the domain. This can also lead to inefficient memory utilization, because each processor must have sufficient space to contain all particles that could conceivably be in its geometrical region. Communication between processors becomes an issue, because once a photon crosses a boundary of a region, it must be communicated to the neighboring processor, which may mean transmitting 10 to 12 words to describe the photon—position, velocity, weight, time to census, and any other attributes needed to perform the simulation. Thus, spatial domain decomposition may result in a substantial savings in memory due to the need for each processor to keep track of only one region, but this advantage may be negated by the adverse effect on memory utilization. Furthermore, spatial domain de-

"Monte Carlo particle transport is an inherently parallel application if the particles do not interact with one another or change the background medium with which they are interacting. Even in cases where the histories might interact, the nonlinearity can be handled to allow the simulations to be performed independently by using time steps to remove the dependence of the medium on the photon transport process. Within a time step, the photons are treated as independent particles. Between time steps the properties of the medium may be changed."

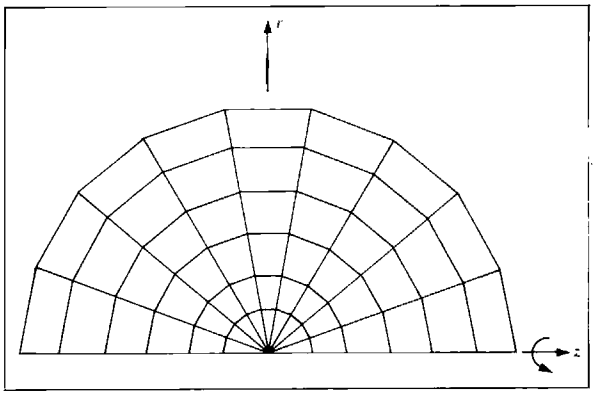

Fig. 2 Typical two-dimensional mesh (9 × 6)

- Read input
- Generate geometrical mesh
- Calculate zone properties (e.g., opacities)
- For zones **z** = 1,2,.....NZONES
  - For energy groups **g** = 1,2,..NGROUPS
    - Determine number of photons N_{zg} to be emitted in zone **z**, energy group **g**
    - For photons **n** = 1,2,...N_{zg}
      - Emit photon (Planckian emission)
      - Simulate photon history
      - Accumulate tallies
      - Next photon **n**
    - Next energy group **g**
  - Next zone **z**
- Report results
- End

Fig. 3 Sequential algorithm (SPHOT)

composition has the consequence of increased interprocessor communication (which may be overwhelming if the regions are transparent to the photons) and the potential of unequal workloads.

Domain decomposition by energy group involves partitioning the photons into groups depending on their energy (or frequency). These groups are chosen so that photons within them have constant interaction properties with the background medium. In this case, the processors would need to know the material properties for only a small portion of the energy scale, but would still need to know the entire geometrical mesh. However, photons can lose energy in collisions (Compton scattering) with the medium, and this would require communication between processors. For the application considered in this work, we allow only Thomson scattering, which does not change the photon energy; in general, both types of scattering would be considered. A second concern, however, is load-balancing, because the photons are emitted in different groups and interact with the medium at different rates depending on the energy group. Thus, it would be difficult to partition them in such a way as to ensure equal workloads.

Domain decomposition, either in space or in energy, does not seem to be a desirable approach for partitioning the problem. It does reduce the memory requirement, but leads to increased communications and increased difficulty in obtaining equal workloads. Instead, we have focused on methods where the entire domain (space and energy) is assigned to all processors and the particles are distributed among the processors in such a way as to equalize the workload and minimize communications. This partitioning of the photons is natural, and there are many specific methods to implement it. The next section describes the partitioning for the shared memory architecture.

4. PARALLEL ALGORITHM: SHARED MEMORY

The effort to develop a parallelized algorithm was targeted for the IBM 3090/400 shared memory parallel processor (Wan and Martin, 1986). A "dispatching" algorithm was developed that partitions the photons by the zone in which they were born (emitted) and assigns them to separate processes, which are in turn assigned to the next available processor. That is, a separate process is defined for all photons emitted in a given zone. The process emits the photons and follows them, re-

gardless of which energy group they are in or where they happen to travel. This algorithm was chosen for our initial examination because it was relatively easy to implement on the IBM 3090/400 with the Multitasking Facility (MTF). Since the MTF requires communication between the main task and slave tasks by argument list only (i.e., no "GLOBAL" COMMON), only the zone index must be transmitted to each process. The process then determines the total number of photons to be emitted (a function only of the zone volume and material properties) and processes the photons one at a time, similar to a conventional Monte Carlo algorithm. Since the photon histories are all independent (within the current time step), there is no need for interprocess communication until the simulation is completed. At that time the global tallies are accumulated from the partial tallies that have been accumulated by the slave processes. The algorithm is described in Figure 4.

This first approach is probably not optimal, because it is not load-balanced (the workload depends on the zone properties and location) and it is relatively fine-grained, with one process created for each zone. For the test problem that we have analyzed, there are 1,960 zones, hence 1,960 processes to be dispatched to at most four processors for the IBM 3090/400. We are currently investigating alternative algorithms that are load-balanced and coarser grained, with the number of processes equal to a small multiple (1,2,3,4, . . .) of the number of available processors.

Since the individual processes are independent statistical simulations, a method is needed to generate, for each process, independent random number sequences that will ensure statistically independent simulations as well as the reproducibility of the simulation. By reproducibility we mean that if the starting random seed is the same, then the overall simulation should always yield identical results regardless of the number of processors assigned to the simulation at run time (including the uniprocessor case). The "Lehmer tree" approach (Frederickson et al., 1984) is used to generate these random sequences for each process. The basic idea is to generate a "left" sequence of random numbers, each of which will be the starting random seed for the random number sequence ("right" sequence) used within each process. The linear congruential random number generator (Knuth, 1969) is used for the left and right sequences, with a careful choice of the multipliers and increments

MAIN → TASK
- Read input
- Generate geometrical mesh
- Calculate zone properties (e.g., opacities)
- Determine initial random seeds for each zone **z** ("left" sequence of Lehmer tree)
- Get processors (Call to MTF)
- For zones z = 1,2,.....NZONES
  - Create and assign process to processor (Call to MTF)

SLAVE → TASKS
- Determine total number of photons N_z to be emitted in zone **z**
- For photons **n** =1,2,...N_z
  - Emit photon
  - Simulate photon history
  - Accumulate partial tallies
  - Next photon **n**

MAIN → TASK
- Next zone **z**
- Synchronize (Call to MTF)
- Accumulate global tallies from partial tallies
- Report results
- End

Fig. 4 Dispatching algorithm (shared memory)

"Even the negligible sequential fraction of .03% has severe performance implications for a massively parallel architecture."

Table 1
Measured Versus Predicted Speedups for IBM 3090/400

N	Total CPU (min)	μsec/trk	Measured Speedup	Predicted—Eq (3) (f = 0)	Predicted—Eq (3) (f = .018)
1	3.14	45.2	—	—	—
2	1.63	23.4	1.93	1.999	1.93
3	1.11	16.0	2.83	2.998	2.84
4	0.84	12.1	3.74	3.996	3.73

N = number of processors.
Total CPU = total elapsed time (min) for simulation excluding I/O.
μsec/trk = total CPU time (μsec)/total number of tracks.
Speedup = total time (1 CPU)/total time (N CPUs).

(Frederickson et al., 1984) to ensure that the sequences are disjoint. In addition, since this approach guarantees that one can reproduce the random number sequences, reproducibility is ensured. However, the Lehmer tree approach, while convenient and easy to implement, has been shown to suffer from intersequence correlations (Bowman and Robinson, 1987). Since the purpose of our study was to assess the efficiency of our parallel algorithms (rather than obtain actual answers to a specific problem), and the correlations do not affect this study, we have continued to use the Lehmer tree approach. We have not noticed any anomalous results with the Lehmer tree, but this may be a consequence of our specific application.

In this algorithm the work in the loop over zones is completely parallel and the sequential portion is simply the input, initialization, and output. The fraction of work that is done in parallel depends on the number of photon histories. For the standard ICF simulation (Martin, Nowak, and Rathkopf, 1986), there are ~240,000 photons emitted in the plasma and the vast bulk of the work is done in the parallel section. The sequential fraction, denoted by α, was determined to be approximately .03%, by executing the parallel algorithm on a uniprocessor and explicitly taking synchronization delays into account by timing each process individually. However, as noted below, even this negligible sequential fraction (α = .0003) has severe performance implications for a massively parallel architecture.

This algorithm was implemented on the IBM 3090/400, and dedicated runs with one, two, three, and four processors yielded the measured speedups tabulated in Table 1, where the measured speedup S_N^m is defined

$$S_N^m = \frac{\text{elapsed time for single processor}}{\text{elapsed time for N processors}} \qquad (1)$$

Table 1 includes an absolute indicator of performance, the "microseconds per track" (μsec/trk), which is simply the total CPU time in microseconds divided by the total number of "tracks," where a track is the movement of a particle from one position in phase space to another. This measure has been found to be a useful absolute measure of Monte Carlo performance because it is relatively insensitive to the problem physics or geometrical mesh. For the test problem reported in Table 1, there were approximately 4,170,000 tracks, or more

than 17 tracks for each of the nearly 240,000 photons emitted in the plasma.

The performance with the optimal sequential algorithm was essentially the same as the parallel algorithm on a single processor, because the overall algorithms are similar and the MTF is not utilized for one processor. Therefore, the definition of speedup in Eq (1) is consistent with the usual definition—the ratio of the optimal sequential elapsed time to the elapsed time for multiple processors. Elapsed times were used because the IBM 3090/400 was run in dedicated mode; they are included in Table 1, with the resultant speedups calculated with Eq (1). It should be noted that all of the IBM 3090/400 timing results discussed here are for the parallel-scalar algorithm. We are currently implementing the vector code VPHOT and the parallel-vector code PVPHOT (discussed below) on the IBM 3090 and hope to obtain results shortly.

A few remarks regarding the MTF might be of interest. The MTF has limited capability, but we found that this made it relatively easy to use and difficult to abuse—it can only create processes, assign them to processors, and terminate processes. There is no way to communicate between processes, such as posting an event or setting a barrier. There is also no "GLOBAL" COMMON, which means that to update global variables one must first exit the slave task and return to the MAIN task. This also constrains one to communicate all information via argument list, although subordinate subroutines in a slave task can be called with shared COMMON blocks (tantamount to "TASK" COMMON). But these constraints were relatively easy to deal with (perhaps a consequence of our inherently parallel application), and we encountered no difficulties in executing the multitasked code.

In addition to the measured speedups, Table 1 includes predicted speedups using Amdahl's law (Amdahl, 1967),

$$S_N^A = \frac{1}{\alpha + (1 - \alpha)/N} \qquad (2)$$

where α is the sequential fraction and N is the number of processors. It is clear that the measured and predicted speedups do not agree for any of the runs. In an attempt to explain this discrepancy, we have introduced a "multitasking fractional overhead," denoted by f, which accounts for the overhead introduced by multitasking

Table 2
Effect of Multitasking Overhead

N	Predicted Speedups	
	$f = 0$	$f = .018$
2	1.999	1.93
3	2.998	2.84
4	3.996	3.73
8	7.98	6.98
16	15.9	12.4
32	31.7	20.2
64	62.8	29.5
128	123	38.3
256	238	45.0
512	444	49.4
1,024	784	51.9
∞	3,333	54.6

the sequential algorithm. For example, there will be some overhead due to the MTF creating and destroying processes and assigning them to the next available processor. There may also be some overhead due to memory conflicts. Defining the additional workload due to the postulated overhead as f times the total workload in the sequential algorithm, one obtains a revised version of Eq (2) which accounts for the postulated fractional overhead,

$$S_N = \frac{1}{\alpha + (1 - \alpha)/N + f} \qquad (3)$$

Thus, Eq (3) with $f = 0$ yields the usual Amdahl's law prediction for the speedup. (Note that we have postulated a fractional overhead that is independent of N.) A reasonable value for f may be determined by forcing the predicted speedup S_N to equal the measured speedup S_N^m for $N = 2$. This results in $f = .018$, which implies that there is an additional workload imposed that is equal to 1.8% of the total workload on a uniprocessor. If this value is then used for the runs with $N = 3$ and $N = 4$ processors, the predicted speedups are very close to the measured speedups, as noted in Table 1. Thus this simple model appears to explain the measured results quite well. The origin of this overhead is not clear, however, and we are examining alternative algorithms in an attempt to verify this simple model and explain the source of the overhead. We emphasize that this simple model is only tentative and should be subjected to further testing with alternative partitioning schemes that are more equally load-balanced and coarser grained.

It is interesting to project what the speedup would be for N processors assuming that the 1.8% overhead held for any number of processors. Table 2 illustrates the sensitivity of the predicted speedups to the presence of the seemingly modest 1.8% overhead. Thus, for an inherently parallel application such as Monte Carlo, where the degree of parallelization is close to 100%, the impact of a relatively small overhead due to the multitasking implementation will still be significant. Even for a modestly parallel configuration, the effect of a 1.8% overhead is significant—the change in speedup from eight processors to 16 processors is 5.42, an effective loss of more than two and a half processors. Similarly, adding 16 processors to a configuration with 16 pro-

cessors will result in an effective gain of less than eight processors. Thus, less than half of the added capability can actually be utilized. These results, however, are for only one possible partitioning of the simulation, by zone of emission. Figure 5 plots the results contained in Table 2 for a graphic display of the degradation in performance due to the postulated multitasking overhead.

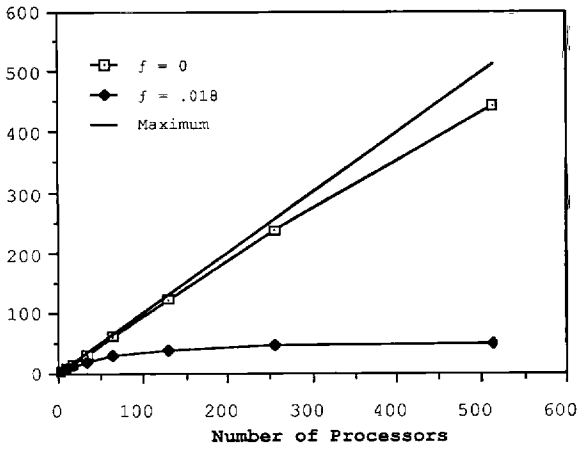

Fig. 5 Effect of multitasking overhead

5. PARALLEL ALGORITHM: DISTRIBUTED MEMORY

A parallelized algorithm for the 64-node NCUBE hypercube parallel processor at the University of Michigan (Hayes et al., 1987) has been developed. Three approaches were examined (Martin et al., 1987), stemming from the relatively modest memory (128 Kbytes) per node that made it impossible for the SPHOT code to be replicated on each node. (It should be noted that the NCUBE is now available with 512 Kbytes per node.) These versions are denoted A, B, and C.

5.1 VERSION A

The DIMENSION statements in SPHOT were reduced to allow a maximum of 20 zones versus a maximum of 2,000 zones for the original version of SPHOT. The entire code (except for input/output) was replicated in each processor, and the host merely reads in the input data, broadcasts it to the nodes, and then receives the results from each node and prints out separate results for each calculation. The Lehmer tree method was utilized to start each node calculation with a different random seed, allowing the separate Monte Carlo runs to be combined to yield results statistically equivalent to one large run with the same total number of photons. Table 3 summarizes the results for the reduced-dimension algorithm (Version A) as a function of N, the number of nodes.

Table 3 also includes the microseconds per track performance measure, which was 39.6 μsec/trk for the 64-node NCUBE/six. For comparison, the identical 5 × 4 mesh problem was run on the CRAY X-MP/22 (single CPU) with the scalar code SPHOT and yielded 37.5 μsec/trk. Thus, in scalar mode, the 64-node NCUBE/six was approximately as fast as a single CRAY X-MP processor for the 5 × 4 test problem. It should be noted that the vectorized code VPHOT is nearly six times faster than the scalar code SPHOT, and this comparison between the NCUBE and the CRAY is only to indicate

Table 3
Measured Timing Results on NCUBE

	Version A (5 × 4)			Version B (5 × 4)			Version B (49 × 40)		
N	Node	Total	μsec/trk	Node	Total	μsec/trk	Node	Total	μsec/trk
1	13.5	14	2,481	22.7	24	4,279	194.7	205	4,920
2	13.5	14	1,242	22.8	24	2,130	194.7	204	2,448
4	13.4	14	624	22.8	24	1,065	194.7	204	1,224
8	13.4	14	312	22.8	24	532	194.7	205	615
16	13.4	14	156	22.8	24	266	194.7	205	308
32	13.4	14	80.4	22.8	24	137	194.7	204	158
64	13.4	14	39.6	22.8	24	67.6	194.7	205	78.1

N = number of processors.
Node = average CPU time per node (sec).
Total = elapsed time from posting of first message to nodes to receipt of last result from nodes (sec).
μsec/trk = average CPU time to process one track (move a photon from one position in phase space to the next).

their relative scalar performances on this particular Monte Carlo simulation.

5.2 VERSION B

The uniform and homogeneous mesh (in the ICF test problem) was exploited by calculating the zone vertices and material composition as a function of the zone index. (The SPHOT and VPHOT codes allow an arbitrary two-dimensional mesh.) This eliminated the need to store the zone vertices and composition for each zone. This modified version allows the analysis of the "standard" test problem with the 49 × 40 mesh, which could not be fit into the NCUBE node if the zone vertices and compositions were to be stored. Other than this change to eliminate the stored arrays, Version B is identical to Version A and yields identical results. Table 3 summarizes the Version B results for the "small" mesh (5 × 4) and the "standard" ICF test problem with the 49 × 40 mesh. A comparison of the timing results for the 5 × 4 mesh with Versions A and B indicates the penalty (nearly doubling the computational time) associated with calculating the zone vertices rather than storing them.

5.3 VERSION C

The "dispatching" algorithm developed for the shared memory parallel processor was modified to be operational on the NCUBE. In this case, the algorithm is shown in Figure 6. Version C differs from Versions A

and B in that a single Monte Carlo simulation is being spread out over the N processors in the cube, rather than replicating the same calculation (with different random number sequences) in each of the N processors. This is the more conventional approach to "parallelizing" an algorithm. However, the replication approach (Version B) does take advantage of the unique feature of Monte Carlo wherein results of several calculations can be combined a posteriori to yield an improved result. As with Version B, the method to conserve memory by calculating zone vertices and compositions "on the fly" was used. Tables 4 and 5 summarize the performance results for Version C as a function of the number of nodes and the problem size (i.e., the number of photons simulated). The results reported in Tables 4 and 5 are for the 49 × 40 mesh.

As can be seen, the measured efficiencies are poor for the small problems, which is to be expected, since the parallel workload is proportional to the number of photon histories that are simulated. For the large problem, which is the standard ICF test problem discussed earlier with ~240,000 photon histories, the efficiencies are considerably better, approaching 80% for the 64-node configuration. Anomalous results are obtained for both the medium-size problem (Table 4) and the large problem (Table 5) in that the efficiency is greater for 64 nodes than for 16 or 32 nodes.

HOST →
- Read input
- Generate geometrical mesh
- Calculate zone properties (e.g., opacities)
- Determine initial random seeds for each zone **z** (Lehmer tree)
- For zones **z** = 1,2,.....NZONES
  - Dispatch index **z** and input data to next available processor (via message from **HOST** to **NODE**)

NODES →
- Determine total number of photons N_z to be emitted in zone **z**
- For photons **n**=1,2,...N_z
  - Emit photon
  - Simulate photon history
  - Accumulate partial tallies
  - Next photon **n**
- Send message "**ALL DONE**" back to **HOST**

HOST →
- Next zone **z**
- Accumulate global tallies from partial tallies
- Report results
- End

Fig. 6 Dispatching algorithm (local memory)

Table 4
Measured Timings for Version C as a Function of Problem Size

	Small (~2,700 photons)				Medium (~24,000 photons)			
N	**Total**	**μsec/trk**	**Speedup**	**Eff**	**Total**	**μsec/trk**	**Speedup**	**Eff**
1	207	4,674	1.00	1.0	1,773	4,428	1.00	1.0
2	142	3,206	1.46	.73	—	—	—	—
4	72	1,626	2.88	.71	563	1,406	3.15	.79
8	46	1,039	4.50	.56	337	841	5.26	.66
16	26	587	7.96	.49	167	417	10.60	.66
32	16	361	12.90	.40	85	212	20.90	.65
64	13	294	15.90	.25	39	97	45.50	.71

N = number of processors.
Total = total elapsed time for simulation excluding I/O (host clock resolution = 1 sec).
μsec/trk = total time (μsec)/number of tracks.
Speedup = total time for 1 CPU/total time for N CPUs.
Efficiency = speedup/N.

Table 5
Measuring Timings for Version C (Large Problem with ~240,000 Photons)

N	Total	μsec/trk	Speedup	Eff
1	18,124	4,395	1.0	1.0
16	1,596	387	11.4	.71
32	873	212	20.8	.64
64	348	84	52.1	.81

N = number of processors.
Total = total elapsed time for simulation excluding I/O (host clock resolution = 1 sec).
μsec/trk = total time (μsec)/number of tracks.
Speedup = total time (1 CPU)/total time (N CPUs).
Efficiency = speedup/N.

One possible explanation for the anomalous results is that better load-balancing is achieved with 64 processors because of a better distribution of the zones to the processors. The distribution depends on the number of processors, the number of zones, the order in which the zones are dispatched to the next available processor, as well as the message traffic through the I/O structure of the overall cube. Since there is a specific order in which the zones are dispatched to the next available processor, and the workload is a sensitive function of the zone properties (volume and temperature), it is probably fortuitous that the efficiency was higher for the 64-processor case. We have examined the CPU timing statistics (workload) for each node and have calculated the fractional standard deviation in the node workload (standard deviation/mean workload × 100%). This value, which is indicative of the uniformity (or nonuniformity) of the workload, was approximately 22% for both the 16-node and 64-node cases and 30% for the 32-node case given above in Table 5. Thus the 32-node case results in a significantly more nonuniform workload, consistent with the above postulated reason. Since this anomaly is repeatable, this explanation seems plausible, although we are still examining this issue. We also will be trying the alternative algorithm described above, which utilizes fewer tasks and is more load-balanced. This may help us to explain these anomalous results.

For Versions A and B, it is important to note that the problem size grows linearly with the number of nodes, because the same problem is simply run on more processors. However, since the random seeds for each problem are different, the overall simulation does represent a meaningful calculation because the results from each simulation can be combined to yield overall results with the standard deviation in the estimated results improved by the ratio $1/\sqrt{N}$, where N is the number of processors (simulations). The Version C algorithm, on the other hand, partitions the workload internally, dispatching work (batches of particles emitted within a specific zone) to the next available processor.

6. PARALLEL-VECTOR ALGORITHM: SHARED MEMORY

With the introduction of parallel and/or vector shared memory architectures such as the CRAY X-MP/48 and IBM 3090/400 computers, the development of multitasked vectorized algorithms gains importance. In gen-

eral, the first priority is vectorization to maximize the utilization of a single processor. Clearly, one important trade-off is vector length versus granularity—as the number of tasks increases, the vector length decreases given a constant problem size. Some effort has already been reported, for multitasking a vector Monte Carlo code (Chauvet, 1984, 1985). Chauvet reports results for the CRAY X-MP/22, which has two vector processors. His results are for a multitasked version of an alternative vectorized algorithm, which is "stack-based" (Martin and Brown, 1987), rather than the VPHOT algorithm considered below.

Since the VPHOT code is vectorized (Martin, Nowak, and Rathkopf, 1986), an attempt was made to develop a parallelized version for a multiple vector processor. The resultant parallelized and vectorized Monte Carlo algorithm is a simple extension of the original VPHOT algorithm illustrated in Figure 7.

Figure 8 plots the measured performance of VPHOT as a function of the problem size (i.e., the number of photons emitted within the plasma). It is apparent that the performance degrades with decreasing problem size, even for what appear to be reasonably large simulations (e.g., a 10% degradation in performance with 15,000 photons compared with 240,000 photons).

In order to address this issue, an algorithm was developed that distributes the workload "evenly" among the available processors. The disadvantage of this approach is that it relies on some a priori knowledge of the problem, in this case that the number of photons emitted was proportional to the volume of the zones. Figure 9 summarizes the parallel-vector algorithm, which is contained in the code PVPHOT.

This algorithm has not yet been implemented on the CRAY X-MP, but results have been obtained by simulating the parallel algorithm on a uniprocessor and using the data contained in Figure 8 to predict the resulting performance of the algorithm. Table 6 tabulates the results of these emulated runs, as a function of the number of slave processes (i.e., the number of independent "VPHOT" simulations) and the number of available processors. Since this is an emulation study, results were also obtained for more than four processors.

The number of photons simulated by each slave process is approximately equal to $240{,}000/N_S$, where N_S is the number of slave processes. For the PVPHOT al-

- Read input
- Generate geometrical mesh
- Calculate zone properties (e.g., opacities)
- Determine number of photons to be emitted in each zone and energy group
- Loop over event iterations n = 1,2,...
  - Fill particle bank (length ≤ 2000)
  - Advance particle bank to end of event (random walk)
  - Accumulate tallies
  - Replenish particle bank
  - Next event iteration n
- Report results
- End

Fig. 7 Vector algorithm (VPHOT)

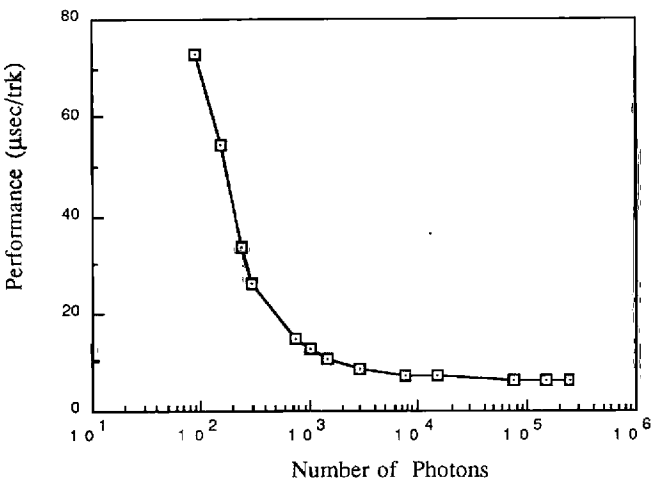

Fig. 8 Effect of problem size on VPHOT performance

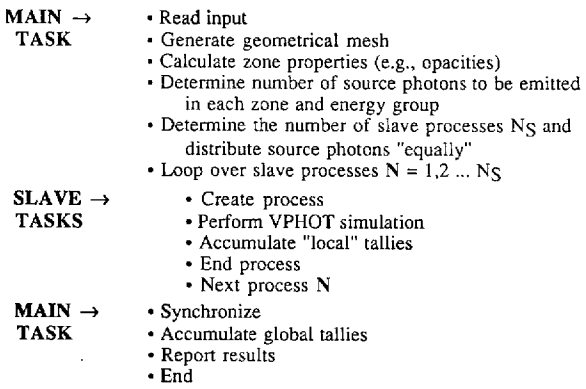

Fig. 9 Parallel-vector algorithm (PVPHOT)

Table 6
Emulated Performance (Speedup) of PVPHOT on CRAY X-MP

No. of CPUs	No. of Slave Processes					
	2	4	6	8	12	16
2	1.91	1.88	1.85	1.82	1.78	1.74
4	—	3.49	—	3.39	3.31	3.23
6	—	—	4.76	—	4.58	—
8	—	—	—	5.88	—	5.70

gorithm, the sequential fraction has been measured to be approximately 4%, which is somewhat higher than the parallel-scalar algorithm because of the need to determine the number of emitted photons for each zone beforehand and to partition these "equally" to the slave processes. The following two observations are made based on the results in Table 6: (1) Increasing the number of slave processes, while keeping the number of CPUs the same, results in a degradation in performance, a consequence of the decrease in vector length; (2) increasing the number of available CPUs, while keeping the number of slave processes constant (hence the vector length constant), also results in a degradation in performance, due to the increasing number of processors waiting during the sequential portion of the algorithm (synchronization delay).

The two-processor results seem comparable to those reported by Chauvet (1984), where a speedup of 1.94 was measured for a sequential fraction of approximately 2%, compared with the emulated speedup of 1.91 reported in Table 6 for a sequential fraction of 4%.

On the basis of these results, it is clear that one would like to keep the number of tasks equal to the number of processors, at least for our case where the load-balancing could be ensured. The results reported by Chauvet (1984) also suggest this. Furthermore, there is a noticeable degradation in performance for even a modest number of processors, even when the number of processors is equal to the number of tasks. This might suggest that it is going to be difficult to obtain high parallelization efficiency for a configuration of multiple vector processors, unless the problem is extremely large.

The emulated performance cannot take into account operating system overhead or memory bank conflicts, which will further degrade the speedups tabulated in Table 6, and this effect should be more pronounced as the number of processors increases. As such, these emulated speedups are probably optimistic, and we are currently implementing PVPHOT on the CRAY X-MP/48 as well as the IBM 3090/400 to obtain measured timing results.

7. CONCLUSIONS

Parallel versions of the Monte Carlo codes SPHOT and VPHOT have been developed for shared memory and distributed memory parallel processors. Measured timing data have been obtained for the IBM 3090/400

for the parallel-scalar algorithm, and emulated data have been predicted for the CRAY X-MP for the parallel-vector algorithm. The overhead to implement multitasking is seen to be an important consideration for the shared memory algorithm, at least for the specific Monte Carlo algorithm studied. For the parallel-vector algorithm, the trade-off between vector length and granularity is the important issue. With respect to the distributed memory parallel processors, two different algorithms were examined, replication and dispatching, and extensive timing data for the NCUBE/six-hypercube parallel processor are reported. Linear performance as a function of the number of processors can be obtained for the replication algorithm, but this takes advantage of a unique feature of Monte Carlo and results in the problem size growing linearly with the number of processors. For the dispatching algorithm, where a single Monte Carlo simulation is spread across N processors, the speedups are almost linear with N, if the original problem is large enough.

The Monte Carlo photon transport application considered here is an inherently parallel application, as are almost all Monte Carlo particle transport applications. In addition, it is completely vectorizable, which implies that it should be an ideal candidate for advanced architectures with multiple scalar and/or vector processors, including massively parallel distributed memory architectures as well as shared memory architectures. The results obtained to date seem to indicate that even for such an inherently parallel application, the extrapolation to massive numbers of processors may not result in a corresponding increase in performance, especially if the processors are vector processors. This conclusion is relaxed if the problem size is allowed to grow at the same rate as the number of processors, although we feel that the efficient utilization of a massively parallel configuration of vector processors will still be difficult.

Regarding the effort to convert production-level Monte Carlo codes to vector and/or parallel architectures, it is clear that the vectorization effort represents a substantial investment, because it necessitates changes to the global algorithm and attendant data bases (Martin and Brown, 1987). On the other hand, its inherently parallel nature makes it relatively straightforward to parallelize a typical Monte Carlo code, whether scalar or vector. It should be pointed out that in essence the "replication" algorithm discussed above could be imple-

mented on a multiple CPU configuration by the operating system—it could queue up several independent Monte Carlo simulations, and the user could accumulate the results afterward.

ACKNOWLEDGMENTS

This work was supported by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and IBM Corporation (Kingston Laboratory). The authors would also like to acknowledge the assistance of Kevin Jones (IBM, Kingston), who performed the dedicated runs on the IBM 3090/400.

BIOGRAPHIES

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 development of algorithms for scientific computation on advanced computer architectures.

Tzu-Chiang Wan was born in Kaohsiung, Taiwan, Republic of China, on May 24, 1956. He received his B.S. degree in nuclear engineering at the National Tsing-Hua University, Taiwan, in 1978, and his M.S. degree in nuclear engineering from the University of New Mexico in 1982. He is a Ph.D. candidate in the Department of Nuclear Engineering at the University of Michigan. His doctoral research is the development of a Monte Carlo response matrix method, which utilizes the Monte Carlo method in conjunction with the response matrix method to solve computationally intensive problems in nuclear reactor analysis. He is also interested in the development of algorithms for scientific computation on vector and parallel processors, as well as the application and development of computational methods in particle transport and reactor physics. Mr. Wan is a member of the American Nuclear Society, Alpha Nu Sigma, and Sigma Xi.

Tarek S. Abdel-Rahman received the B.Sc. degree in electrical engineering from Kuwait University, Kuwait, in 1980, and the M.S. degree in electrical engineering from the University of Michigan, Ann Arbor, in 1981. He is currently working toward his Ph.D. degree at the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, where he has been a research assistant since 1982. His current research interests include computer systems architecture and parallel processing. Mr. Abdel-Rahman is a member of the IEEE computer society and the Association for Computing Machinery.

Trevor N. Mudge received the B.Sc. degree in cybernetics from the University of Reading, England, in 1969, and the M.S. and Ph.D. degrees in computer science from the University of Illinois in 1973 and 1977, respectively. He is currently an associate professor in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, where he has been since 1977. His research interests include computer architecture, programming languages, VLSI design, and computer vision.

CORRESPONDING AUTHOR

William R. Martin, Department of Nuclear Engineering, 225 Cooley Building, North Campus, University of Michigan, Ann Arbor, Michigan 48109.

REFERENCES

Amdahl, G. 1967. Validity of the single processor approach to achieving large scale computing capabilities. *Proc. Spring Joint Conf. AFIPS* 30:483–485.

Bowman, K. O., and Robinson, M. T. 1987. Studies of random number generators for parallel processing. In *Hypercube Multiprocessors 1987*, edited by M. T. Heath. Philadelphia: SIAM, pp. 445–453.

Chauvet, Y. 1984. Multitasking a vectorized Monte Carlo algorithm on the CRAY X-MP/2. *Cray Channels* 6:6–9.

Chauvet, Y. 1985. Vectorization and multitasking with a Monte Carlo code for neutron transport problems. Presented at the LANL-CEA Meeting on Recent Applications of the Monte Carlo Method, April 22–26, Cardarache, France. CEA-CONF 7902.

Duderstadt, J. J., and Martin, W. R. 1979. *Transport theory.* New York: Wiley.

Frederickson, P. F., Hiromoto, R., Jordan, T. L., Smith, B., and Warnock, T. 1984. Pseudo-random trees in Monte Carlo. *Parallel Comput.* 1:175.

Hayes, J. P., Jain, R., Martin, W. R., Mudge, T. N., Scott, L. R., Shin, K. G., and Stout, Q. F. 1987. Hypercube computer research at the University of Michigan. In *Hypercube Multiprocessors 1987*, edited by M. T. Heath. Philadelphia: SIAM, pp. 382–394.

Knuth, D. E. 1969. *The art of computer programming.*

Vol. 2. Reading, Massachusetts: Addison-Wesley.

Los Alamos Monte Carlo Group. 1981. MCNP: A general Monte Carlo code for neutron and photon transport. LA-7396-M (revised). Los Alamos, New Mexico: Los Alamos National Laboratory, Group X-6.

Martin, W. R., and Brown, F. B. 1987. Status of vectorized Monte Carlo for particle transport analysis. *Internat. J. Supercomput. Appl.* 1(2):11–32.

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 *Hypercube Multiprocessors 1987*, edited by M. T. Heath. Philadelphia: SIAM, pp. 454–463.

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 combinational geometry. USAEC Rep. ORNL-4585. Oak Ridge, Tennessee: Oak Ridge National Laboratory.

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