Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

VECTORIZATION AND PARALLELIZATION OF TRANSPORT MONTE CARLO SIMULATION CODES

Kenichi Miura

Fujitsu America, Inc.
3055 Orchard Drive
San Jose, California 95134-2017

ABSTRACT

This paper discusses the computational techniques, the coding methodology and the performance for the transport Monte Carlo simulation on the vector supercomputers and on the shared-memory parallel processors. A cascade shower simulation code EGS4 has been taken as an example. As for vector processing, more than 10 times of performance has been obtained by treating the problem in a different manner from the conventional sequential processing in such a way as to exploit the vector architecture of current supercomputers. As for parallel processing, more than 25 times performance has been obtained over sequential processing by using 29 processors. This paper also discusses a new analytical performance model for parallel processing, new issues in parallel processing the transport Monte Carlo codes, and comparisons between vector and parallel approaches.

1. INTRODUCTION

In recent years, the demand for solving large scale scientific and engineering problems has grown enormously. Since many programs for solving these problems inherently contain a very high degree of parallelism, they can be processed very efficiently if algorithms employed therein expose the parallelism to the architecture of a supercomputer.

Today's supercomputers such as CRAY X-MP,CRAY Y-MP, CRAY 2, FUJITSU Vector Processor Systems, Hitachi S-820 System and NEC SX System, mainly depend on the vector processing approach to boost their performances, with parallel processing capabilities besides vector processing in most cases [Fernbach 1986; Hockney and Jesshop 1988; Mendez and Orszag 1988; Uchida et al. 1990]. One example of the vector architecture is shown in Fig. 1 [Uchida et al 1990].

Particle transport problems are widely encountered in the neutron and radiation calculation in nuclear engineering, phase-space simulation and cascade shower simulation in high energy physics [Alcouffe et al. 1985; Nelson et al. 1985]. To this date, the Monte Carlo simulations constitute the only feasible means of solving many such problems involving complicated interactions and arbitrary geometrical structures, although they consume vast amount of computation time and are expensive.

While the supercomputers have successfully exhibited very high performance for applications such as solving the partial differential equations and signal processing, these machines usually give only the scalar performance for the existing Monte Carlo simulation codes. In order to fully utilize the architecture of a supercomputer for such applications, development of suitable algorithms is very important.

In this paper, we will discuss the algorithm issues as well as the software engineering issues involved in the high-performance computation of the transport Monte Carlo simulation, particularly how the parallelism in the application programs are to be matched with a given supercomputer architecture. We will also introduce a new law which is suitable for the performance characterization in the asynchronous parallel processing, as an alternative to Amdahl's law in the synchronous parallel processing. The techniques and issues discussed here are not solely for the Monte Carlo applications, but should be applicable to other applications as well.

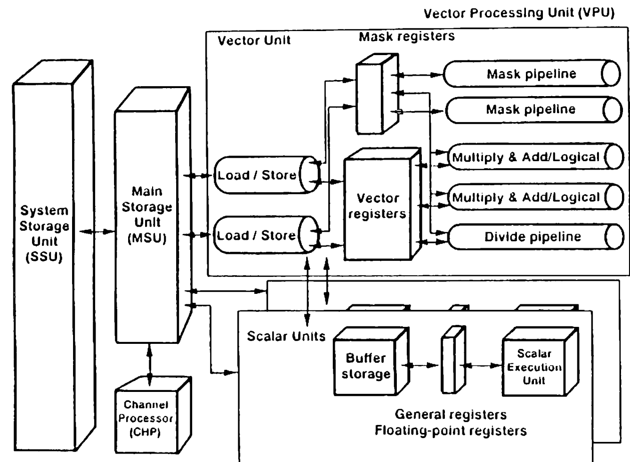


Figure 1. An Example of Vector Architecture (FUJITSU VP2000 Series)

2. CHARACTERISTICS OF TRANSPORT MONTE CARLO SIMULATION CODES

2.1 Scalar Nature of Code Structure

In a typical transport Monte Carlo simulation code, each particle is transported through the media, boundary crossing is checked in a given geometrical structure, then it encounters some interactions. This process is continued until the particle escapes from the structure, or until there is no further interest in this particle. Processing of each particle involves many data-dependent IF tests, due to the stochastic nature of the computational model of physical interactions, hence leaving very little parallelism within the particle loop.

It should be noted, however, that a very high degree of parallelism exists at the particle level, since there could be thousands of particles to be simulated, each of which can be treated independently of others. Due to the above-mentioned nature of the transport Monte Carlo simulation, it has generally been regarded as a perfect application for the parallel processing rather than for vector processing.

In this paper, we will take EGS4 code as an example, to dicuss how it can be effectively vectorized as well as parallelized.

2.2 Overview of Cascade Shower Simulation Code EGS4

EGS4 is the latest version of the EGS (Electron-Gamma Shower) Code System which has been developed at the Stanford Linear Accelerator Center [Nelson et al. 1985]. This code system is a general purpose package for the Monte Carlo simulation of the coupled transport of electrons and

photons on an arbitrary geometry. EGS4 is widely used in high energy physics (simulation of electromagnetic cascade showers) and in medical physics.

An electromagnetic cascade shower starts with one particle with very high energy (say, above 1 Gev) which subsequently creates many particles through radiation and collision (Fig. 2). Particles in a shower are transported through the media, and are eventually discarded as they lose their energy below a prescribed threshold through collision and radiation processes or as they escape from the geometrical structure. The analog Monte Carlo approach has been adopted in EGS4, and all the multiplicative processes are simulated.

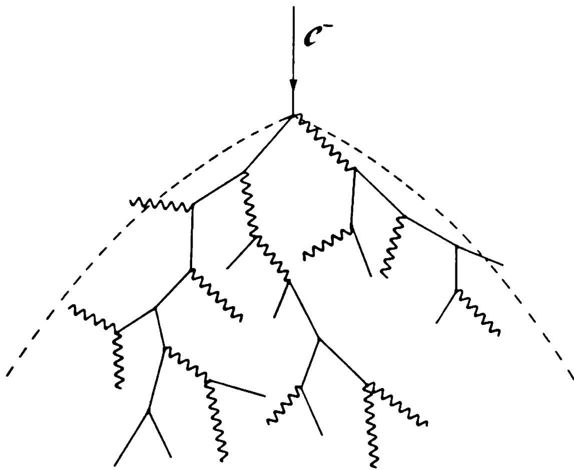


Figure 2. An Electromagnetic Cascade Shower

2.2.1 General Code Structure

The general control flow of the major subroutine SHOWER is illustrated in Fig. 3. In EGS4, only one particle is processed at a time, and there are no explicit DO loops at all in the main body of the code.

2.2.2 Data Structure

A push-down stack is used for storing the particle data with a pointer which points to the top of the stack. At the start of a simulation, the stack is loaded with one incident particle (usually an electron). Since only one particle can be processed at a time in the scalar processing, there is no need to develop a shower to its full extent. Instead, as the simulation proceeds, the newly created particles are placed in the stack, the particle with the lowest energy always being on the top. This is equivalent to tracing the shower tree in Fig. 2 toward the shortest branch, thus keeping the stack depth to the minimum. The push-down stack is obviously the optimal choice for scalar processing from the viewpoint of the memory size requirement.

2.2.3 Control Scheme

The control scheme employed in EGS4 is very simple; the particle on the top of the stack is always to be processed in the next simulation step. When the stack is empty, simulation is completed.

3. VECTOR PROCESSING OF TRANSPORT MONTE CARLO SIMULATION CODES

As the vector supercomputers become widely available, quite a few reports have been recently made regarding the efforts in vectorizing the transport Monte Carlo simulation

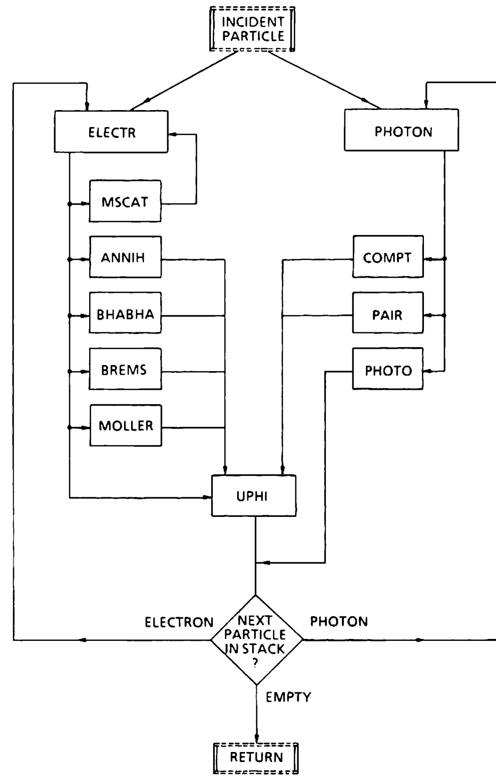


Figure 3. General Control Flow of EGS4

codes [Bobrowicz et al. 1984; Chauvet 1984; Asai et al. 1986; Martin and Brown 1987; Miura 1986,1987; Martin 1988], most of which strongly indicate that vector approach is indeed worthwhile. Vectorization is the act of tuning an application code to take advantage of the vector architecture [Fernbach 1987]. Vectorization may be done by the compiler alone, with the aid of the so-called the compiler directives, with explicit function calls to some special library, or through code restructuring, depending on the capabilities of the compiler as well as on the complexity of an application code.

3.1 Basic Strategy for Vector Processing

The basic strategy for vectorization is similar in all the reported works, namely to pool particles in a common data structure called stack or queue, and to form vectors with particles possessing identical characteristics by gathering them from the common data structure. In this way, many particles can be processed in one pass in vector mode. In order for this strategy to be successful, it is very important to carefully design the data structure so that the vectorized algorithms can exploit the parallelism contained in the problem; the scalar Monte Carlo codes in many cases adopt inherently sequential data structure (typically a last-in first-out buffer or a push-down stack).

The efficiency of vector processing also depends on the varieties of interaction patterns and/or complexities of geometry, both of which strongly influence the complexity of the codes as well as the effective vector length at each step of simulation. In the actual vectorized codes, most of the loops are heavily populated with the nested IF-THEN-ELSE structures, and the compiler's capability to vectorize such complex loops is essential in obtaining a good vector performance. In practice, code-restructuring involves a deep understanding of the codes and takes considerable amount of

effort, since the Monte Carlo codes are usually very large in size.

The reported vector performance improvement over the scalar ranges anywhere from 1.4 to to 85, most of them falling between 5 and 10 [Martin 1988].

3.2 Vectorization of Cascade Shower Simulation Code EGS4

All the works in vectorizing the transport Monte Carlo simulation codes, as mentioned in 3.1, are concerned with the transport of the neutral particles such as neutrons and photons, but no research work has been reported in the area of the charged particle transport, especially the cascade shower simulations. This section describes the approach for vectorizing the electromagnetic cascade shower Monte Carlo code EGS4, and shows that the vector supercomputer with powerful vector data handling capabilities can achieve good vector performance.

The remainder of this section describes EGS4-V, a vector version of the electromagnetic cascade shower Monte Carlo code EGS4, developed by us. The vector supercomputer used for this research is FUJITSU VP200 Vector Processor System with FORTRAN77/VP Vectorizing Compiler [Fernbach 1986].

As stated in 2.2, the code structure of EGS4 is highly sequential and seems unvectorizable at first sight. The following subsections will describe how EGS4 code has been restructured to yield a high vector performance.

3.2.1 Independence of Particles and Degree of Parallelism

In a cascade shower simulation, once a particle is created, it is completely independent of other existing particles and can be processed in any order. Therefore, if the shower is fully developed at the earliest possible stage, a very high degree of parallelism is expected. Furthermore, if a sufficient number of particles have been accumulated for one type of interaction, they can be efficiently processed in one pass in the vector mode. This observation leads to an entirely different control scheme and data structure from those in the original scalar code as described in 2.2.

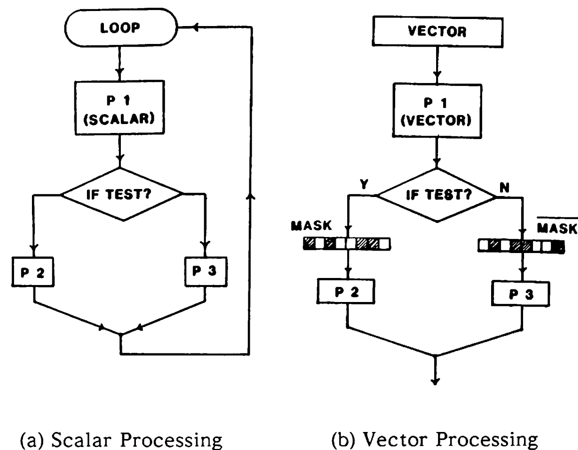
An experimental vector version, named EGS4-V, has been developed along this line. Neither the physics models nor the sampling algorithms have been modified in this effort; only the order in which the particles are to be processed are different from the original EGS4 code.

3.2.2 Vectorization of Loops Containing IF Tests

One of the most important issues in vectorizing a Monte Carlo simulation code is how to vectorize the DO loops containing IF tests. In a typical Monte Carlo simulation code, there are two types of IF structure, namely the feed-forward type IF test and the feed-backward type IF test [Miura 1986]. These two types of IF structure must be treated separately in vector coding.

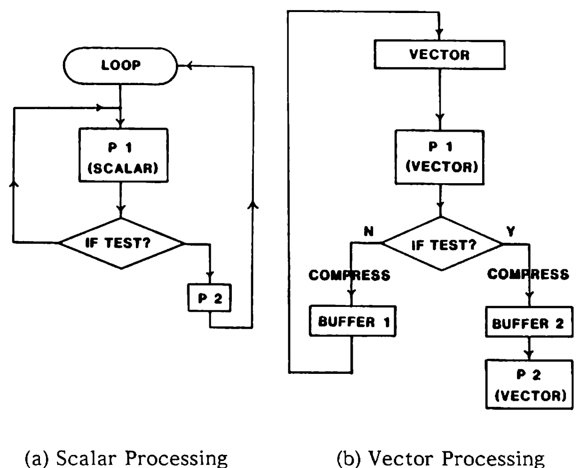
The first type is encountered when different computations are to be performed depending on the result of an IF test (Fig.4a). For example, positrons may be treated differently from electrons, or the particles in high energy range may be treated differently from those in low energy range, etc. The feed-forward type IF test can usually be vectorized by the vectorizing compilers. Fig. 4b depicts a simple case when the mask bits are used to process the two branches separately. Note that the feedback path has been eliminated in the vectorized code.

The second type is usually encounterd in the rejection sampling routines [Knuth 1981; Kalos and Whitlock 1986], where the data dependent values of the trial (rejection) functions are compared with the random numbers and the trials are repeated until the sampling is accepted (Fig. 5a). The vectorization of the feed-backward type IF requires some semantic modification of the original scalar code. The common approach is to define two temporary buffers, the Accept Buffer (Buffer 2 in Fig. 5b) and the Reject Buffer (Buffer 1 in Fig. 5b). The accepted samples are compressed into Buffer 2, while the rejected samples are compressed into Buffer 1 at the end of each iteration. In the subsequent iterations, Buffer 2 becomes the input to the loop. This process is repeated until Buffer 2 becomes empty. In the vectorized code, the feedback path still exists, but it has been moved to the outside of the loop, hence reducing the IF test to the feed-forward type. As is clear in the above description, the vector data handling capabilities such as the vector compress/expand and the hardware gather/scatter are essential for this operation. In many practical cases, these two types of IF tests are used together and heavily nested to construct a very complex IF structure.



(a) Scalar Processing (b) Vector Processing

Figure 4. Feed-forward Type IF Test



(a) Scalar Processing (b) Vector Processing

Figure 5. Feed-backward Type IF Test

3.2.3 Random Number Generation

The most commonly used technique for random number generation in the transport Monte Carlo codes is the

congruential method due to its simplicity [Knuth 1981; Kalos and Whitlock 1986]. In EGS4 code, for example, the following multiplicative congruential method is used:

```
Loop over i
  Iseed = A * Iseed modulo 2^32
         (random seed in integer format)
  Ran   = Iseed * 2^-32
         (floating-point random number),
         where A = 663608941.
```

Although this algorithm may seem recursive, it can be easily vectorized if the multiplicative coefficients (A, A^2, A^3, ...A^N) modulo(2^{32}) are pre-calculated and stored in an array. At the time of random number generation, each element of this array (up to the desired number of random numbers not exceeding N) is multiplied by the current value of Iseed and the resulting integer random numbers are normalized to obtain the floating-point random numbers, all in the vector mode. Only the last integer random number needs to be stored as the seed for future use. Similar technique is also applicable to the linear congruential method [Matsuura et al. 1985].

While the vectorized random number generator generates the identical random number sequence as the original scalar algorithm, the vectorized Monte Carlo code and the scalar Monte Carlo code do not necessarily produce the identical simulation results since the order in which the random numbers are used may be different in two cases.

3.2.4 Vector Algorithms for Scoring Computations

In the Monte Carlo simulations, the simulation results are usually represented by accumulating the physical quantities in the regions of interest or, equivalently, in the form of the histograms. Typical examples are the energy deposition in the detectors, or the angular distribution of the radiation. This type of computation may be generically called scoring. A simple model of the scoring computation may be depicted as follows:

```
Loop over i
  k = Ireg(i)
  Esum(k) = Esum(k) + Edep(i),
```

where i is the particle index ($1 \leq i \leq N$), Edep(i) is the energy to be deposited by the i-th particle, k = Ireg(i) is the index of the region where the i-th particle is located ($1 \leq k \leq K_{max}$), and Esum(k) is the accumulated energy in the k-th region.

Scoring computation is inherently sequential and not vectorizable in its present form, since more than one particle may deposit energy in the same region (to be called recursion). If the CPU time for the scoring computation is insignificant, it may be left scalar. If the CPU time is significant, on the other hand, there are several vectorizable algorithms which can avoid the recursion. Some of such algorithms are briefly described in the following.

Algorithm 1: Running Sum Method
(1) Sort k = Ireg(i) in the ascending order.
(2) Rearrange Edep(i) accordingly to obtain Edep(i').
(3) Count the runs for all distinguishable k's in (1) (say, R(k').
(4) For each k', take the summation of R(k') elements of Edep(i') and store it in Esum(k').

Algorithm 2: Sort and Stride Method [Parady 1988]
(1) Sort k = Ireg(i) in the ascending order.
(2) Rearrange Edep(i) accordingly to obtain Edep(i').
(3) Find the maximum run of k in (1) (say, Rmax).
(4) Accumulate Edep(i') in the corresponding Esum with the stride of Rmax.

Algorithm 3: Two-dimensional Work Buffer Method [Orii 1983]
(1) Define and clear a two-dimensional work buffer Wbuf(K_{max}, L).
(2) Accumulate Edep(i) in Wbuf(Ireg(i),i), L particles at a time.
(3) After all Edep(i)'s have been accumulated in Wbuf, accumulate W buf(k,*) in Esum(k) for each k.

Each algorithm has its own merits and demerits, and should be selectively used, depending on the N, Kmax and available work buffer area in the memory. Typical vector vs. scalar speedup factor is from 5 to 10, if an appropriate vector algorithm is used.

3.3 Global Code Structure of EGS4–V

Fig. 6 illustrates the general control flow structure of the subroutine SHOWER, the main part of EGS4–V. It consists of the dataflow control section and a multi-way jump to the slave subroutines. The dataflow control section monitors the particle data at each simulation step and initiates the execution of the next subroutine. The slave subroutines include the particle transport, the interactions and the garbage collections. Most of the slave subroutines are the vectorized version of the original subroutines, but some are newly defined for this vector version.

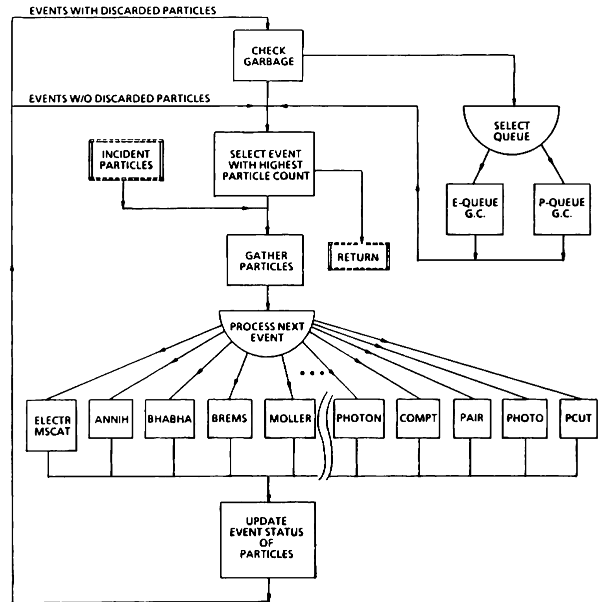


Figure 6. General Control Flow of EGS4–V

3.3.1 Data Structure

The EGS4–V uses the queues instead of the push-down stack in order to fully expose the parallelism in the particle data. Here, a queue means a collection of particles which are ready for the next step of computations. Queues are defined in the main memory as one-dimensional arrays, but the order of the elements is unimportant for this application. It is obvious that a shower can be most quickly developed by traversing the shower tree in Fig. 2 in the horizontal direction; the newly created particles are immediately stored in the queue.

In the current implementation of EGS4–V, there are two separate queues in EGS4–V, namely, E–Queue for electrons/positrons, and P–Queue for photons. This is not the only choice; one common queue may be used for all types of particles. The event status ID is also assigned to each particle in the queue besides the physical variables. The said ID can take one of the 14 values corresponding to the 14 slave subroutines. 14 event status counters are also provided,

each keeping track of the number of particles in the corresponding status.

3.3.2 Control Scheme

The dataflow control section in subroutine SHOWER serves as a global event monitor by constantly scanning the event status counters, so that the subroutine with the highest particle count is always to be executed in the next simulation step for the maximum vector efficiency.

3.3.3 Garbage Collection

In any vectorized shower simulation code, many particles are created and/or discarded in each simulation step, and the queue can easily overflow. Therefore, it is necessary to reclaim the unused portion of the queues. This process is commonly called garbage collection. There are two methods for implementing the garbage collection. The first method is to compress the queue whenever the number of discarded particles in the queue exceeds a certain threshold. In this method, the newly created particles can be stored in the contiguous locations of the queue. The second method is to use a so-called source buffer which holds indices pointing to all the available locations in the queue. When particles are discarded, their indices are added to the source buffer, while when particles are newly created, their locations are provided from the source buffer. In this method, the queue is always accessed via indirect addressing.

In the current implementation of EGS4-V, the first method has been adopted for ease of debugging. The dataflow control section checks the amount of garbage after executing the subroutines which involve discarded particles.

3.4 Timing Measurement of a Sample Problem with EGS4-V

A sample problem has been run on FUJITSU VP200 System to measure the vector performance of EGS4-V against the original scalar EGS4. In this problem, 1 Gev electrons are injected into a lead block of infinite size, so that no boundary crossing takes place. The number of the incident electrons has been varied from 10 to 200.

The results of the timing measurement are shown in Fig. 7. Since the publication of the early results [Miura 1987], the code has been improved, and an asymptotic vector vs. scalar performance ratio of 11.6 has been obtained for this measurement.

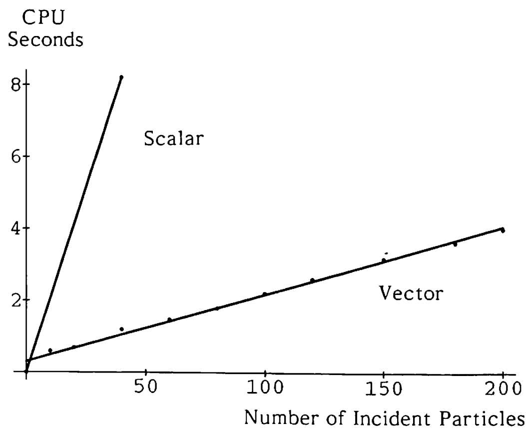


Figure 7. Timing Measurement of EGS4-V

Fig. 8 and Fig. 9 depict dynamic behaviors of the active vector length. Fig. 8 shows a case of 100 incident electrons with 1 Gev energy each, while Fig. 9 shows a case of one incident electron with 100 Gev energy. The vector lengths have been averaged over 10 simulation steps in these graphs. In both cases the vector length varies considerably in spite of the above-mentioned smoothing process. Table 1 summarizes the results of two cases.

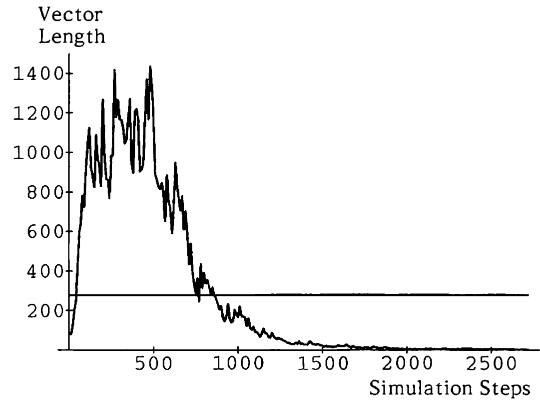


Figure 8. Typical Dynamic Behavior of Vector Length (1Gev, 100 electrons)

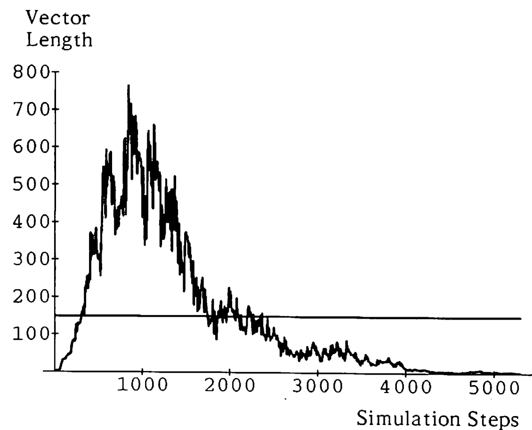


Figure 9. Typical Dynamic Behavior of Vector Length (100Gev, 1 electron)

Table 1. Simulation Results in Two Cases

Incident Energy (Gev)	Incident Particles	CPU Time (Sec.)	Simulation Steps	Peak Vector Length	Average Vector Length
1	100	2.2105	2720	2113	277.8
100	1	2.6552	5298	1245	148.8

It should be noted that the total energy involved in these two simulations is the same, and the total number of operations should be nearly equal. In vector processing, however, the total CPU time is shorter when there are more incident particles, which is due to longer vector length.

4. PARALLEL PROCESSING OF TRANSPORT MONTE CARLO SIMULATION CODES

4.1 Issues in Parallelizing Transport Monte Carlo Codes

In this paper, we will focus our discussions on the parallel processing based on the shared-memory architecture. Although the parallelization may seem conceptually more natural and straightforward than the vectorization for the transport Monte Carlo simulations, there are new issues in parallel programming at the same time. These issues are not solely confined to the Monte Carlo simulation, but of a more general nature. Some of them are addressed in the following.

4.1.1 Identifying Global and Private Variables

In a shared-memory architecture, all the variables in the COMMON blocks must be carefully examined whether they should be shared among the processors (global variables), or privately copied for each processor (private variables). In the scalar codes, each COMMON block may contain both types of variables, in which case it has to be split into two blocks for parallelization. This is a very time-consuming task, if done manually. Definitely, good software tools are needed in this area. Another issue to be noted is that the notion of the COMMON block for parallel processing is not well established, and some systems do not support both types of COMMON blocks, hence the portability problem.

4.1.2 Machine-dependent Library Functions of Synchronization Primitives for Parallel Programming

Even within the category of the parallel processing systems with the shared-memory, each system has its own library functions, compiler directives or FORTRAN extensions to describe and/or control parallel processing. There is no standard in this area. This, again makes the porting of the parallel codes very difficult.

4.1.3 Parallel Random Number Generation

Unless great care is taken that each particle uses the same sequence of random numbers in the parallel code as in the scalar code, results are not guaranteed to be the same. Worse yet, it is quite possible to construct a parallel code which does not produce the same results from run to run due to the effect of race conditions in obtaining random numbers. One practical solution to this problem is the concept called Lehmer-tree, based on two sets of the linear congruential random number generators [Frederickson et al. 1984]. By adopting this concept and by assigning a random seed to each particle, the same simulation results can be obtained regardless of the order in which the particles are processed and with any number of processors. This is a new area, and further research will be needed to establish algorithms for generating good parallel random numbers.

4.2 Two Approaches for Parallelizing EGS4 Code

In the remainder of this section, we describe our experiences in parallelizing EGS4 on Sequent B21000 Parallel Processing System [Osterhaug 1987]. There are two basic approaches in parallelizing the EGS4 code: one is to parallelize the original scalar code in such a way as to process many independent particles in parallel (to be called fine-grain approach), and the other approach is to start with the vectorized version and either to process each loop in parallel (so called microtasking), or to process the independently executable vectorized subroutines in parallel (so called macrotasking, or large-grain approach).

4.3 Fine-grain Parallel Programming

In this approach, each processor fetches a particle from a shared stack and executes the scalar simulation code. The synchronization is done by locking and unlocking the stack pointer to the shared particle stack, thus allowing dynamic load balancing. We have developed the fine-grain parallel version of EGS4 called EGS4-P [Miura and Babb 1988]. The Lehmer-tree technique [Frederickson et al. 1984] was incorporated in this code. A sample problem was run with one 50 Gev electron injected into a lead block of infinite size, and a parallel speedup factor of more than 25 was obtained with 29 Sequent B21000 processors. (Fig. 10)

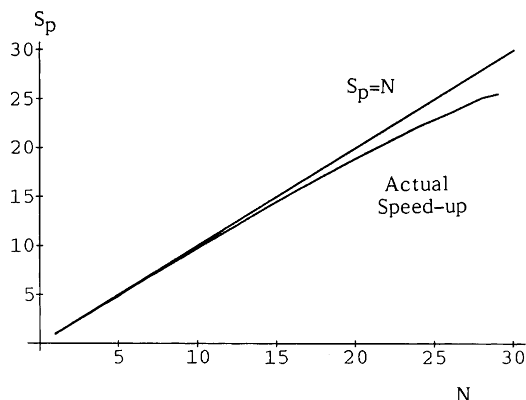


Figure 10. Speedup of EGS4-P on Sequent B21000 System

The sub-linear characteristics of the speedup curve as indicated in Fig. 10 can be ascribed to the following factors:

(1) Initiation of the parallel tasks
When a new task is spawned, all the private common blocks have to be replicated. This task replication time is proportional to the number of tasks, while the total execution time itself is almost inversely proportional to the number of the tasks. Therefore, this overhead becomes more significant as the number of the processors increases, unless the problem size is sufficiently large.

(2) Asymmetric development of a cascade shower
The development of a shower is asymmetric in the initial stage, so that the degree of parallelism is low until a sufficiently large number of particles have been generated.

(3) Critical section of the code
There are inherently sequential operations in the child tasks due to the exclusive accesses to the particle stack (the critical section).

(4) Physical resource contentions
In a shared-memory architecture, the common physical resources such as the memory bus and the memory banks also limit accesses by only one processor at a time. The amount of the overhead due to this factor heavily depends on the system parameters such as the cache size, the memory bus throughput, the memory configuration etc., and are not under the user's control.

While the first two factors may become less significant as the size of a shower increases, the third and the fourth factors always take a certain percentage of the CPU time, regardless the size of a shower. In our numerical experiments on the Sequent Parallel Processing System, the fraction of the CPU time spent in the critical section is estimated to be somewhere between 1 and 2 percent. The effects of the physical resource contentions are not included in this estimate. We further analyze the effect of the critical section to the speedup factor in Section 5.

4.4 Microtasking and Large-grain Multitasking Approaches

We have investigated the possibilities of parallel processing the vectorized version of the EGS4 Monte Carlo code. So far, our experiments revealed that the microtasking approach did not turn out to be attractive due to the complexity of the DO loop structure and a lack of a software tool at the time of this study. On the other hand, the large-grain approach is more promising since the code structure of the vectorized version of the EGS4 already incorporates independently executable slave subroutines. With the advent of the vector multiprocessor systems, this approach seems to be the right one, and deserves further research. We are now in the process of implementing a large-grain vector-parallel version of EGS4 code based on the Large Grain Data Flow concept [Babb and DiNucci 1989].

5. ANALYTICAL PERFORMANCE MODEL FOR FINE-GRAIN PARALLEL MONTE CARLO CODES

In this section, we introduce a simple analytical model which is suitable for interpreting the sub-linear characteristics of the speed-up factor which was observed in the fine-grain parallel version of EGS4 code.

5.1 Synchronous Model vs Asynchronous Model for Parallel Processing

It should be emphasized that the commonly known Amdahl's law is applicable only to the cases when the parallelizable part of the code is processed simultaneously by all the processors before entering the sequential part of the code (Barrier Model). Throughout this paper, we will call this model synchronous parallel processing. Vector processing can be regarded as a special case of the synchronous parallel processing.

The Amdahl's law can be stated as follows [Amdahl 1988]:

$$S_p = 1 / (1 - \alpha + \alpha / N)$$

where S_p is the speedup factor, α is the fraction of the CPU time to be spent in the parallelizable part of the code, and N is the number of the processors. Fig. 11 illustrates typical speedup as a function of α for various values of N for the synchronous parallel processing while Fig. 12 illustrates the speedup as a function of N for various values of α.

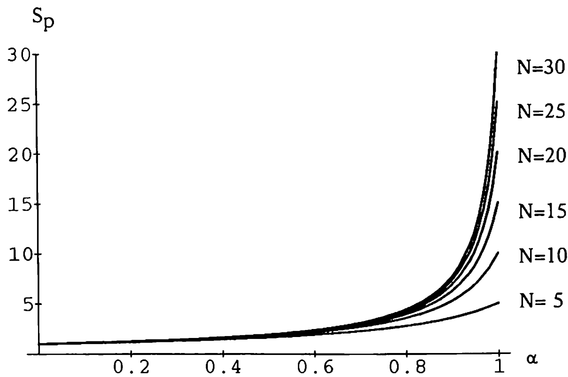


Figure 11. Speedup Factor for Synchronous Model of Parallel Processing (S_p vs α)

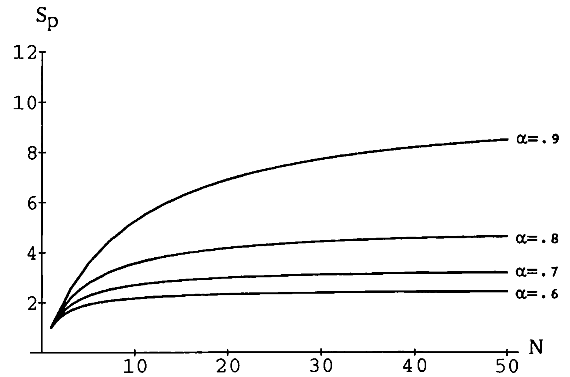


Figure 12. Speedup Factor for Synchronous Model of Parallel Processing (S_p vs N)

5.2 Analytical Speedup Factor for Asynchronous Model of Parallel Processing

In the fine-grain parallel version of the EGS4 code, on the other hand, processors can run independently of others as long as they do not access the particle stack. The access to the particle stack is exclusive, constituting a critical section, but only those processors which access this critical section while it is busy are kept waiting, not all the processors are kept waiting (Critical Section Model). In this sense, the parallel processing in this case may be dubbed asynchronous parallel processing.

The analytical formula for the speedup curve which is discussed here is based on the machine servicing model or quasi-random input model in the Queueing Theory [Feller 1968; Kobayashi 1978], where the parallelizable part of the code (to be called parallel section hereafter) corresponds to N sources, and the sequential part of the code (to be called critical section hereafter) corresponds to a single server. For the sake of the simplicity of the discussion, we assume that N sources and one server have exponential service time distributions with the parameters λ and μ, respectively. Due to the stochastic nature of the execution time of the parallel and critical sections of the code, the parallelization ratio α should be interpreted as the statistical average, and is related to λ and μ as follows:

$$\alpha = \mu / (\lambda + \mu)$$

Under the above assumptions, the analytical form of the speedup factor for the asynchronous model can be derived by utilizing the readily available results from the corresponding queueing model.

Since a detailed derivation of the analytical formula is given in [Miura 1990], only the results are presented here.

$$S_p = (1-\Pi_0)/(1-\alpha) \quad (0 < \alpha < 1)$$
$$= 1 \quad (\alpha = 0)$$
$$= N \quad (\alpha = 1)$$

Where Π_0 is the probability that no processor is in the critical section, which is a function of α and N as defined in the following:

$$\Pi_0 = 1/(1+N\rho+N(N-1)\rho^2+....+N!\rho^N),$$

and

$$\rho = \lambda/\mu = (1-\alpha)/\alpha$$

Fig. 13 illustrates the speedup (S_p) as a function of parallelization ratio (α) for various values of N, while Fig. 14 illustrates the speedup as a function of the number of processors (N) for various values of α.

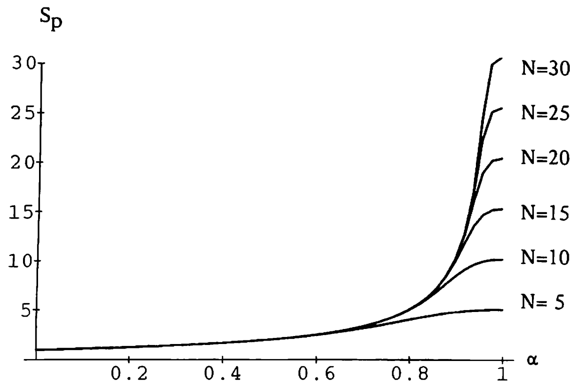


Figure 13. Speedup Factor for Asynchronous Model of Parallel Processing (S_p vs α)

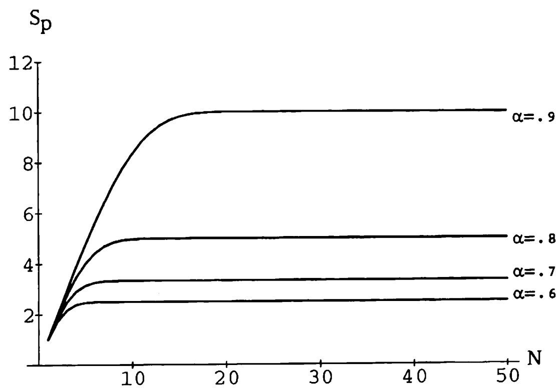


Figure 14. Speedup Factor for Asynchronous Model of Parallel Processing (S_p vs N)

5.3 Discussions

By comparing Fig. 11 with Fig. 13, or Fig. 12 with Fig. 14, several interesting observations can be made regarding the similarities and the dissimilarities between the synchronous and the asynchronous models of parallel processing.

(1) In both cases, speedup factor is N when $\alpha = 1$.

(2) For a given N and $0 < \alpha < 1$, the asynchronous speedup factor is always greater than the synchronous one, and both are less than $1/(1-\alpha)$.

(3) As α decreases from 1, the asynchronous speedup factor initially gives a slow decay while the synchronous speedup factor gives a very sharp decay.

(4) For a given α and with an increasing N, the asynchronous speedup factor saturates at the asymptotic value $1/(1-\alpha)$ faster than the synchronous one.

(5) As α approaches 0, the asynchronous speedup factor rapidly converges to the synchronous one, both yielding 1 at $\alpha = 0$.

While the asynchronous speedup factor may depend on the actual probability distribution of the CPU time spent the parallel and critical sections, the qualitative results as described above are believed to hold in general. With regard to our experimental data for the EGS4 code, our analytical model gives $\alpha = .97$ as the best fit. Since the critical section alone takes from 1 to 2 percent of the total CPU time, the bus contention and the memory bank conflict are supposed to take the rest of the 3 percent according to this model.

6. CONCLUSIONS AND FUTURE DIRECTIONS

This paper discussed vectorization and parallelization techniques for the transport Monte Carlo simulations. The fundamental differences between scalar coding and vector coding, and also between the scalar coding and parallel coding have been addressed, based on our own experiences. It has been pointed out that the transport Monte Carlo codes inherently contain a very high degree of parallelism, and that they can be either vectorized or parallelized efficiently.

As for the vectorization of the transport Monte Carlo codes, typical speedup factors of 5–10 have been reported in the literature. It should also be noted that parallelism at the higher level becomes visible through the vectorization process, which is quite suitable for vector–parallel processing. The vectorization techniques described in this paper are not solely confined to the transport Monte Carlo simulations, but should be applicable to other seemingly unvectorizable problems. The vector data handling capabilities which are accessible from FORTRAN language, are the key factors for implementing vector codes.

Although parallelization of the Monte Carlo codes may be more straight–forward than vectorization, a lot more research and development should be made in the programming environment in general, especially in compiler technology, in debugging tools, and in parallel software development tools which can provide useful information for efficient parallel programming. The necessity for a global scanning capability on the part of the compiler should be emphasized as the architectural trend moves toward various forms of parallel architecture, where the detection of so–called large granularity parallelism is required. Development of a fully automatic compiler may become impractical for such systems. Rather, user–friendly interactive software tools seem to be the right approach [Pacific Sierra Research Corporation 1989; Klappholtz and Kong 1989].

We have also introduced an analytical model of the asynchronous parallel processing for the cases where the commonly known Amdahl's law is not applicable. This model is suitable, for example, for describing the performance characteristics of the fine–grain parallel version of the EGS4 Monte Carlo code. With the advent of the asynchronous parallel algorithms in various application areas, the analytical performance models along the line of our approach should be examined for other cases.

Since supercomputing is an application–driven area, very close interactions between researchers in various applications and supercomputer manufacturers will be crucial in order to cope with the ever–increasing demands for large scale scientific and engineering computations.

ACKNOWLEDGEMENT

The author would like to thank Dr. W. R. Nelson of Stanford Linear Accelerator Center for providing EGS4 code and for valuable discussions, Dr. R. G. Babb II for valuable discussions on the fine–grain and the large–grain parallel processing techniques, and Sequent Computer Systems, Inc. for providing computational resources for the timing measurements of the fine–grain version of EGS4.

REFERENCES

Alcouffe, R., R. Dautray, A. Forster, G. Ledanois, and B. Mercier, Eds. (1985), "Monte Carlo Methods and Applications in Neutronics, Photonics, and Statistical Physics," *Lecture Notes in Physics 240*, Springer-Verlag, New York, NY.

Amdahl, G.M. (1988), "Limits of Expectation," *International Journal of Supercomputer Applications 2*, 1, 88-94.

Asai, K., K. Higuchi, and J. Natakura (1986), "Vectorization of KENO-IV Code," *Nuclear Science and Engineering 92*, 298-302.

Bobrowicz, F., J.E. Lynch, K.J. Fisher, and J.E. Tabor (1984), "Vectorized Monte Carlo Photon Transport," *Parallel Computing 1*, 3-4, 295-305.

Chauvet, Y. (1984), "Multitasking a Vectorized Monte Carlo Algorithm on the Cray X/MP2," *Cray Channels 6*, 3, 6-9.

Feller, W. (1968), *An Introduction to Probability Theory and Its Applications*, Volume 1, Wiley & Sons, New York, NY.

Fernbach, S., Ed. (1986), *Supercomputers*, North-Holland, Amsterdam, The Netherlands.

Frederickson, P., R. Hiromoto, T.L. Jordan, B. Smith, and T. Warnock (1984), "Pseudo-random Trees in Monte Carlo," *Parallel Computing 1*, 2, 175-180.

Hockney, R. and C. Jesshop (1988), *Parallel Computers 2*, Adam Hilger, Philadelphia, PA.

Kalos, M. and P.A. Whitlock (1986), *Monte Carlo Methods*, Volume 1, Wiley-Interscience, New York, NY.

Klappholtz, D. and X. Kong (1989), "CFTP: A Tool to Aid in Hand-Parallelizing Sequential Code,"*The Digest of Papers for COMPCON Spring '89*, IEEE Computer Society Press, Los Alamitos, CA, 92-97.

Knuth, D. (1981), *The Art of Computer Programming*, Volume 2, Second Edition, Addison-Wesley, Reading, MA.

Kobayashi, H. (1978), *Modeling and Analysis*, Addison Wesley, Reading, MA.

Matsuura, T., K. Miura, and M. Makino (1985), "Supervector Performance Without Toil," *Computer Physics Communications 37*, 101-107.

Martin, B. (1988), "Particle Transport Monte Carlo on Shared-Memory and Distributed-Memory Parallel Processors," In *Proceedings of the Third International Conference on Supercomputing 2*, 348-353.

Martin, W. and F. Brown (1987), "Status of Vectorized Monte Carlo for Particle Transport Analysis," *International Journal of Supercomputer Applications 1*, 2, 11-32.

Mendez, R. and S. Orszag, Eds. (1988), "Japanese Supercomputing," In *Lecture Notes in Engineering 36*, Springer-Verlag, New York, NY, 111-127.

Miura, K. (1986), "Vectorization of Phase Space Monte Carlo Code in FACOM Vector Processor VP-200," In *Computing in High Energy Physics '86*, L.O. Hertzberger and W. Hoagland, Eds. Elsevier Science Publishing, New York, NY.

Miura, K. (1987), "EGS4-V: Vectorization of the Monte Carlo Cascade Shower Simulation Code EGS4," *Computer Physics Communications 45*, 127-136.

Miura, K. (1990), "Vectorization and Parallelization of Transport Monte Carlo Codes," In *Supercomputing, NATO ASI Series F62*, J.S. Kowalik, Ed. Springer-Verlag, New York, NY.

Miura, K. and R. Babb II (1988), "Tradeoffs in Granularity and Parallelization for a Monte Carlo Shower Simulation Code," *Parallel Computing 8*, 1-3, 91-100.

Nelson, W.R., H. Hirayama, and D.W.O. Rogers (1985), "The EGS4 Code System," Research Report SLAC-265, Stanford Linear Accelerator Center Report, Stanford, CA.

Orii, S. (1983), Fujitsu Limited, Personal Communications.

Osterhaug, A. (1987), *Guide to Parallel Programming*, Sequent Computer Systems Incorporated, Beaverton, OR.

Pacific Sierra Research Corporation (1989), *The Forge Easy Reference Guide*, Placerville, CA.

Parady, B. (1988), Amdahl Corporation, Personal Communications.

Uchida, N., M. Hirai, M. Yoshida, and K. Hotta (1990), "Fujitsu VP2000 Series," In *Digest of Papers for COMPCON Spring '90*, IEEE Computer Society Press, Los Alamitos, CA, 4-11.