Computer Physics Communications 57 (1989) 68–77
North-Holland

SUCCESSFUL VECTORIZATION – REACTOR PHYSICS MONTE CARLO CODE

William R. MARTIN [1]

Department of Nuclear Engineering, University of Michigan, Ann Arbor, MI 48109-2104, USA

Most particle transport Monte Carlo codes in use today are based on the "history-based" algorithm, wherein one particle history at a time is simulated. Unfortunately, the "history-based" approach (present in all Monte Carlo codes until recent years) is inherently scalar and cannot be vectorized. In particular, the history-based algorithm cannot take advantage of vector architectures, which characterize the largest and fastest computers at the current time, vector supercomputers such as the Cray X/MP or IBM 3090/600. However, substantial progress has been made in recent years in developing and implementing a vectorized Monte Carlo algorithm. This algorithm follows portions of many particle histories at the same time and forms the basis for all successful vectorized Monte Carlo codes that are in use today. This paper describes the basic vectorized algorithm along with descriptions of several variations that have been developed by different researchers for specific applications. These applications have been mainly in the areas of neutron transport in nuclear reactor and shielding analysis and photon transport in fusion plasmas. The relative merits of the various approach schemes will be discussed and the present status of known vectorization efforts will be summarized along with available timing results, including results from the successful vectorization of 3-D general geometry, continuous energy Monte Carlo.

1. Introduction

The Monte Carlo method is one of the principal computational techniques used for analyzing particle transport problems. Particle transport problems are characterized by high dimensionality and physical complexity and in many cases the Monte Carlo method is the only feasible approach to obtain reasonable solutions without resorting to drastic simplifications of either the problem geometry or the problem physics or both. Historically, the principal drawback to the use of Monte Carlo methods has been its excessive demand on computational resources – Monte Carlo methods require enormous amounts of central processor unit (CPU) time as well as a large, fast memory to contain the geometry information and cross section data base needed to describe the complicated physics.

[1] Royal Society Visiting Fellow, Department of Mechanical Engineering, Nuclear Power Section, Imperial College, University of London, Exhibition Road, London SW7 2BX, UK

2. Computer performance

The dramatic increase in computer performance (and substantial decrease in the cost per CPU second) over the past two decades would seem to be working in favor of Monte Carlo methods. However, this impressive improvement in computer performance has not been due only to advances in hardware – a significant portion of the gains has been due to innovations in *architecture* – how the computer is designed and organized to carry out its computational tasks. The principal innovations for large-scale computation have been vector (pipelined) architectures and parallel architectures. This paper will consider only vector architectures and the development of Monte Carlo methods for such architectures. As is well-recognized, particle transport Monte Carlo is especially well-suited for parallel architectures due to the independent nature of the individual particle histories, but this topic will not be considered in any detail although a few remarks regarding parallel

Monte Carlo are presented in the concluding section of this paper.

Since one vector instruction executes the entire vector operation (which might correspond to 64 multiplications on the Cray-1 or 65535 multiplications on the CDC Cyber-205), it is necessary that the algorithm is *vectorized* – vectors of operands need to be constructed and identical arithmetic operations have to be performed on each component of the vector. Properly constructed, a vectorized algorithm can take advantage of the vector architecture and realize the potential gains in performance offered by this particular type of architecture. As will be discussed below, the conventional Monte Carlo algorithm is inherently scalar and cannot take advantage of a vector architecture. However, effort over the past several years by a number of researchers has led to an alternative algorithm, described as the *event-based* algorithm (an unfortunate description for the high energy physics community, since the term "event" bears no relation to a high energy physics "event"). This paper will summarize these efforts.

The next section describes the conventional Monte Carlo method as used for particle transport analysis. The vectorized algorithms are then described along with results for the different implementations.

3. Conventional Monte Carlo

The conventional particle transport Monte Carlo algorithm involves the statistical simulation of one particle at a time moving through a given medium. Each simulation is termed a "history" and a realistic simulation might consist of 100 000 or more histories. In a typical Monte Carlo code such as MCNP [1], MORSE [2], or GEANT3 [3], a particle will be emitted via a source routine, transported through the medium of interest (*tracked*), and processed through whatever collisions or interactions may occur (*collision analysis*). As the history unfolds, results of the simulation are accumulated (*tallies*), and the simulation continues until the particle is terminated, such as by absorption in the medium, by escape from the problem geometry, or, in a time-dependent simulation, at the end of the time step. The code will then loop over the requisite number of histories to achieve acceptable statistics, which may easily lead to unacceptable computational cost for realistic simulations.

4. Vectorized Monte Carlo

The conventional Monte Carlo algorithm is *inherently scalar* in nature and cannot be vectorized. Since the particle simulation is a random walk, or Markov, process, each step of a history is determined by statistical means (e.g., distance to the next collision, what kind of collision, what angle of scatter, etc.). Therefore, treating many histories simultaneously fails immediately, because the vector is destroyed after the first step in the simulation – some particles in the vector will reach a boundary, some may suffer a capture collision, others a scattering collision, some may reach the end of the current time step, etc. Thus the vector of particles is no longer a vector from the standpoint of the vector CPU because different operations will be performed on each component of the vector.

Thus recompiling an old Monte Carlo source code on a vector supercomputer such as the Cray-X/MP will not result in an efficient utilization of the vector architecture, hence a resultant loss in performance that can approach a factor of 10–20 compared with that potentially attainable. However, recent progress by a number of researchers has shown that significant gains in performance can be attained by totally restructuring the conventional Monte Carlo algorithm to be compatible with the vector architecture. Although there are substantial differences in the individual approaches, all of these vectorized algorithms have a common characteristic – they all are *event-based* algorithms versus the *history-based* algorithm of a conventional (scalar) Monte Carlo code.

4.1. The event-based algorithm

The first mention in the literature of the event-based approach is that of Troubetzkoy et al. in 1973 [4], who adapted a version of the Monte

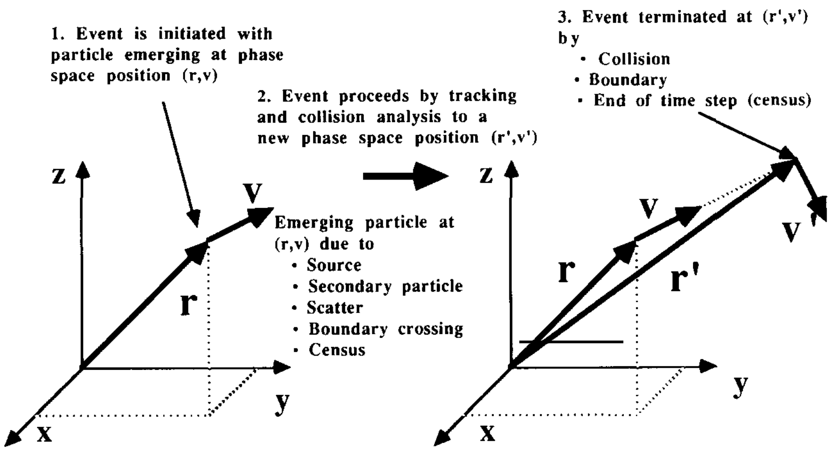
1. Event is initiated with particle emerging at phase space position (r,v)
2. Event proceeds by tracking and collision analysis to a new phase space position (r',v')
3. Event terminated at (r',v') by
• Collision
• Boundary
• End of time step (census)
Emerging particle at (r,v) due to
• Source
• Secondary particle
• Scatter
• Boundary crossing
• Census
z
y
x
r
v
r'
v

Fig. 1. Event initiation and progression.

Carlo code SAM-CE for use on the Illiac-IV, which was an experimental parallel processor with 64 processors.

Although the Illiac-IV was not a vector processor, all 64 processors performed the same operation at each clock step. Therefore it was a "single instruction–multiple data" architecture, or SIMD architecture, similar to a vector processor. The algorithm developed for the Illiac-IV was based on breaking up the overall simulation of many histories into queues of tasks, such as tracking to a boundary, processing an elastic scattering collision, crossing a boundary, etc. The significance of this work was the concept of splitting the history into *events*, which are similar and which can be processed in a vectorized manner.

We define an *event* as that portion of a history which is initiated with the appearance of an emerging particle in phase space at $(\boldsymbol{r}, \boldsymbol{v})$ and terminating at $(\boldsymbol{r}', \boldsymbol{v}')$, which is the beginning of the next event. Figure 1 illustrates the concept of an event, and it is clear that it differs substantially from its definition in the high energy physics community.

For example, an event might be initiated at phase space position $(\boldsymbol{r}, \boldsymbol{v})$ by:
- sampling from a source distribution,
- emerging from a scattering collision,
- entering from census (time-dependent),
- crossing a boundary,
- emission from a nuclear reaction.

Once the event is initiated, it continues until terminated by:
- collision (any kind),
- boundary crossing (including escape),
- census (end of time step),
- killing (non-analog Monte Carlo).

The important observation is that all events are similar – a vector of particles (the particle *bank*) can be processed for one event in a vectorized fashion.

4.2. The event iteration

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

$$\boldsymbol{x}_j^n = \left[x_{j1}^n, x_{j2}^n, x_{j3}^n, \ldots, x_{jK}^n\right],$$

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

$$\Gamma^n = \left[\boldsymbol{x}_1^n, \boldsymbol{x}_2^n, \boldsymbol{x}_3^n, \ldots, \boldsymbol{x}_{L_n}^n\right],$$

where L_n is the number of particles in the particle vector at the beginning of event iteration n. In general, the particles in the bank vector Γ are

```
For event $n = 0, 1, 2, \ldots$:
• Fetch $\Gamma^n$.
• Perform the free flight analysis:
    • gather the cross section data and geometry data tabulated by particle,
        • $\Sigma \leftarrow \boldsymbol{S}$,
        • $\rho \leftarrow \boldsymbol{R}$;
    • using $\Sigma$, sample a vector of distances to collision, $\boldsymbol{d}_c$;
    • using $\rho$, determine vector of minimum distances to boundary, $\boldsymbol{d}_b$;
    • determine the minimum distances to end of event,
        $\boldsymbol{d}_{\min} = \min[\boldsymbol{d}_c, \boldsymbol{d}_b]$;
    • updata the particle coordinates,
        $\boldsymbol{r}^{n+1} = \boldsymbol{r}^n + \boldsymbol{\Omega}^n \cdot \boldsymbol{r}_{\min}$.
• Perform the collision analysis:
    • gather particle attributes,
        $\boldsymbol{\Omega} \leftarrow \boldsymbol{\Gamma}^n$, $\boldsymbol{E} \leftarrow \boldsymbol{\Gamma}^n$;
    • evaluate collision physics for new direction cosines and energies,
        $\boldsymbol{\Omega}' \leftarrow \boldsymbol{\Omega}$, $\boldsymbol{E}' \leftarrow \boldsymbol{E}$;
    • scatter new particle attributes back into bank,
        $\boldsymbol{\Omega}' \rightarrow \boldsymbol{\Gamma}^n$, $\boldsymbol{E}' \rightarrow \boldsymbol{\Gamma}^n$.
• Perform the boundary analysis:
    • gather particle zone indices $\boldsymbol{Z}$,
        $\boldsymbol{Z} \leftarrow \boldsymbol{\Gamma}^n$;
    • determine new zone indices,
        $\boldsymbol{Z}' \leftarrow \boldsymbol{Z}$;
    • scatter new zone indices back into particle bank,
        $\boldsymbol{Z}' \rightarrow \boldsymbol{\Gamma}^n$.
• Update the particle bank,
    $\boldsymbol{\Gamma}^n \Rightarrow \boldsymbol{\Gamma}^{n+1}$ (with $L_{n+1}$ particles)
    (e.g. compress out terminated particles).
• If $L_{n+1} \neq 0$, continue.
```

Fig. 2. The basic event iteration.

ordered randomly and this order will change from one event iteration to the next – no attempt is made to keep track of individual particles in $\boldsymbol{\Gamma}$ from one event iteration to the next. Given $\boldsymbol{\Gamma}^n$, the goal is to advance it to the next event iteration $n + 1$, as summarized in fig. 2.

4.3. Comments on the basic event-based approach

The following comments can be made regarding the event-based algorithm illustrated in fig. 2:

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

$$\Sigma = \left[\Sigma_1, \Sigma_2, \Sigma_3, \ldots, \Sigma_{L_n}\right],$$

$$\rho = \left[\rho_1, \rho_2, \rho_3, \ldots, \rho_{L_n}\right],$$

where Σ_j represents the cross sections and ρ_j the geometry data for the zone that contains particle j.

(2) Figure 2 is only intended to be illustrative of the basic event-based algorithm, since several different variations have been developed, as noted in sections 4.4 and 4.5.

(3) The *compression* step to eliminate terminated particles effectively scrambles the order of the particles in the bank vector $\boldsymbol{\Gamma}$ and leads to one consequence of the event-based algorithm – it is difficult to piece together *individual* particle histories from the event-based simulation.

(4) There is a premium on efficient data handling operations, due to the need to *gather, scatter, compress*, etc. the particle vector and other data arrays before, during, and after every event iteration.

(5) Most of the steps are vector operations, except for the data handling operations, which may be custom-coded (sometimes in hardware) routines and the tally operations, which cannot be vectorized. The collision analysis, boundary crossing analysis (perhaps with Russian roulette and splitting), and other operations to determine the outcome of the event can also be performed in a vector fashion, but now sub-vectors need to be defined (e.g. gathered from the main particle vector) for each distinct outcome and then processed.

4.4. The stack-driven variation

In the basic event-driven algorithm, the particle vector is processed in a manner similar to the conventional history-based algorithm – one cycles through the free-flight analysis, then the collisions analysis routine, then the boundary analysis routine, etc., on an event-by-event basis. Thus at any time during the simulation, all particles will be in the same event iteration. A variation on this basic approach, called the "stack-driven" algorithm, arises when events are further subdivided into smaller computational tasks which are then processed independently. The simplified event-based algorithm in fig. 2, for example, may be logically subdivided into the four separate computational tasks of free-flight analysis, collision analysis, boundary analysis, and particle termination analysis. Rather than cycling through these four tasks in a fixed order, the calculation may proceed by selecting the task involving the greatest number of particles and then performing the analysis for that task. Based on the outcome of this analysis, the affected particles are then queued for the next appropriate tasks.

In comparing the event-based algorithm and the stack-driven variation, the principal difference is the order in which computational tasks are executed. The fixed sequence of tasks in the event-based algorithm leads to simpler control logic and management of particle attribute data at the expense of shorter vector lengths for each individual task. The stack-driven variation selects tasks in a sequence which maximizes the vector lengths, but involves additional logic for managing the particle attribute data. However, it should be noted that the vector computations performed in each task are the same in either approach. Additional details regarding the stack-driven approach and its various implementations may be found in the recent review article by Martin and Brown [5].

4.5. Other variations

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

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

With these definitions, let us now consider the work up to the current time to vectorize Monte Carlo for particle transport. Only a brief summary of the various efforts will be included here – the interested reader is referred to a comprehensive discussion of the topic of vectorized Monte Carlo methodology [6] or the above-referenced review paper [5] which discuss the various vectorization efforts in more detail.

5. Vectorization results

This section is a survey of results obtained by various authors in the area of vectorized Monte Carlo. It is organized by application area, and it it noted that most of the applications relate to reactor physics and shielding problems.

5.1. Gamma transport

The initial effort to develop a vectorized algorithm for a vector supercomputer was that of Calahan, Martin and Brown [6–8] and was based on a simple vectorized code from Los Alamos National Laboratory (LANL) that analyzed only gamma transport in a single, homogeneous carbon cylinder. Speedups in the range of 5–10 were reported. These codes were mainly intended for basic algorithmic studies and not for the purpose of drawing conclusions with respect to production-level Monte Carlo codes, although they did provide valuable guidance for determining which approaches should be taken for the production codes.

5.2. *Multigroup neutron transport*

Subsequent effort by Brown et al. [6,8] was devoted to the development of a vectorized Monte Carlo demonstration code that incorporated most of the significant physics options in standard production-level Monte Carlo codes, but did not have many of the user conveniences typical of a true production code. This code, named MCVMG, was a multigroup (discrete energy bins) code, and had a companion scalar code (MCS) that was developed to allow a fair evaluation of the efficiency of the vector code. The vectorized code was developed for the CDC Cyber-205 supercomputer and utilized a one-zone algorithm, in which only particles in the same zone (geometrical region) are processed at the same time (hence requiring an "outer iteration" over zones to process all the particles). The initial results were actually obtained by emulating the Cyber-205 instructions on a conventional computer and then using published timing data to estimate the speedups [8]. Subsequent implementation of the vector code on a Cyber-205 at Colorado State University verified these results to within 10 % [9]. The results indicated a speedup in the range 25–40 with MCVMG relative to the optimized scalar code MCS for several realistic problems.

5.3. *Continuous energy neutron transport – lattice geometry*

Brown [10] and Brown and Mendelson [11] have reported excellent results for a vectorized, continuous-energy Monte Carlo code for nuclear reactor lattice analysis. They have obtained speedups in the range of 20–85 compared with the previously used scalar production code. This approach utilized an all-zone algorithm with three fixed stacks to handle the tracking, collision analysis, and the total particle bank. The code was optimized for the CDC Cyber-205 vector supercomputer.

5.4. *Photon transport – 2-D*

In a separate application area, Bobrowicz [12] developed a vectorized Monte Carlo code for the analysis of photon transport in a 2-D Lagrangian mesh. The approach was all-zone, but with fixed stacks for most of the distinct computational tasks needed for the Monte Carlo simulation. For example, separate particle stacks were constructed to determine the distance to boundary, to perform the collision analysis, to perform Thomson scattering, and other tasks. When a particle was to undergo one of these tasks, it was put into the corresponding stack and the stack was executed when its length reached 64, which is the length of the Cray vector registers. As such, this algorithm was optimized for the Cray architecture. Reported results indicate that speedups were in the range of 10–20 although this was relative to an old version of the scalar code. Recently, improved speedups have been reported by Fisher [13] for this approach.

5.5. *Neutron transport – 2-D*

Chauvet [14,15] has developed a vectorized Monte Carlo algorithm for neutron transport in a 2-D Lagrangian mesh. The algorithm is similar to the approach of Bobrowicz, except there are fewer stacks and data movement between stacks is minimized where possible by transferring particle indexes rather than particle data between the stacks. The reported speedups relative to the scalar code on the Cray-1 were in the range of 7–13. The vectorized versions for the Cray-1 and Cray X-MP/48 employed assembly coding for determining distance to boundary, which is generally the most computationally-intensive portion of a Monte Carlo code. The Cyber-205 version used vendor supplied software ("q8" calls) for the data handling operations (gather/scatter/compress, etc.) while the Cray X-MP/48 version took advantage of the hardware gather/scatter capabilities of that computer.

5.6. Photon transport – 2-D

In addition to Bobrowicz, Martin et al. [16–18] have developed a vectorized Monte Carlo code for photon transport in a 2-D Lagrangian mesh, typical of an inertial confinement fusion plasma calculation. This code, named VPHOT, employed an all-zone algorithm with two major fixed stacks of particle and several dynamic stacks that are created upon demand. A scalar version of VPHOT, named SPHOT, was developed to allow a meaningful comparison of the vector and scalar algorithms. In addition, the VPHOT and SPHOT results have been compared with a reference Monte Carlo code at Lawrence Livermore National Laboratory (LLNL) for a typical ICF test problem. Results obtained to date indicate that VPHOT is approximately a factor of 5 faster than SPHOT and nearly a factor of 12 faster than the reference LLNL code for the Cray-X/MP. This code has also been adapted to the IBM 3090/400 with vector facilities and speedups consistent with the vector/scalar speed ratio of the IBM 3090 have been obtained.

5.7. KENO-IV vectorization

Asai et al. [19] have attempted to vectorize the production-level Monte Carlo code KENO-IV [20] with disappointing results. They employed an all-zone algorithm with dynamic stacks for the various tasks which are processed in order of length. The resulting vectorized code was tested on two relatively simple configurations and yielded speedups of 1.4 with respect to the original (scalar) version of KENO-IV. They attribute these relatively poor results (compared to results obtained by others) to deficiencies in the compiler, slow indirect addressing (gather/scatter), and the large number of sorting operations (constructing queues). However, given the excellent results obtained by others (including recent results by Brown with a production-level code discussion below), there may be some algorithmic changes that could be incorporated into the vectorized version of KENO-IV to obtain improved results.

5.8. Continuous energy neutron transport – general geometry

Except for the KENO-IV work, none of the above approaches treated the case of a general geometry Monte Carlo code with a general physics package. However, this deficiency appears to have been addressed in recent work reported by Brown [21,22]. This effort has resulted in a three-dimensional, general-geometry, continuous-energy Monte Carlo production code that has essentially no restrictions on problem geometry or problem physics (for reactor analysis), hence is capable of analyzing configurations typically treated by production codes such as GEANT3, MCNP, or KENO-IV.

Brown's method utilizes an all-zone approach with one large stack to hold particle data between events. However, rather than shuffling particle data among the stacks, Brown constructs queues of pointers for each separate task, where the pointer refers to the appropriate particle in the main stack. An event queue is processed if it contains the most particles (indexes) by gather up the affected particle particle attributes (perhaps only a fraction of the total), performing the corresponding (vectorized) operations, and then scattering the affected attributes back into the main stack. Thus the particle pointers become the index list and are "shuffled" among the stacks rather than the particle attributes. Shuffling pointers rather than particle data was also employed to a lesser extent by Martin et al. [16–18] and Chauvet [14,15].

Brown has reported speedups in excess of 10 for a full-core 3-D pressurized water reactor (PWR) model consisting of a pressure vessel containing 137 fuel assemblies, with each assembly containing a 14×14 array of fuel pins, poison pins, control rods, waterholes, etc. The absolute performance for this code is in the range of 9–15 μs/track on the CDC Cyber-205 for a typical mix of applications and less than 8 μs/track on the Cray X/MP4. These results are the most impressive results reported to date and indicate that vectorized Monte Carlo has finally matured – the vectorized code developed by Brown is now the principal production-level Monte Carlo code at

Knolls Atomic Power Laboratory (KAPL) and is used on a daily basis.

5.9. Electron transport - 3-D

Miura [23] has recently reported a factor of 8–10 speed-up for the vectorization of the EGS4 (electron gamma shower) coupled electron and photon transport code, for infinite medium geometries. His algorithm utilizes the stack-driven approach wherein queues of particles are constructed and the queue with the largest number of particles is processed. While this version of EGS4 contains vectorized collision analysis routines, no tracking to boundaries is permitted thus far.

5.10. Particle tracking

Youssef has recently applied a vectorized ray tracing algorithm [24] to Monte Carlo simulations of electromagnetic showers. Based on preliminary results for benchmark cases involving pure particle tracking (no collision analysis) he reports a factor of 20 speed-up from vectorization [25].

5.11. GEANT3 vectorization effort

A description of the initial vectorized algorithm for the geometrical routines in GEANT3 has been reported by Dekeyser [26,27]. No performance results were given.

5.12. Recent work reported at CHEP89

Several papers presented at the CHEP89 conference related to ongoing efforts to vectorize Monte Carlo codes (including GEANT3) for simulation of high energy physics experiments [28–32], and these proceedings should be consulted for further details.

5.13. Summary of Monte Carlo vectorization efforts

Table 1 summarizes the above discussion regarding the vectorization of Monte Carlo codes. As can be seen, impressive speedups can be obtained but it has been found in all successful cases that substantial modifications and revisions were needed to the original Monte Carlo codes (in

Table 1
Summary of Monte Carlo vectorization efforts

Author	Application/Type	Speedup	Computer	vs.	Ref.
Brown, Martin and Calahan	gamma transport, cylinders	7	Cray-1	CDC-7600	[6]
Brown, Martin and Calahan	neutron transport, multigroup, 3-D	30–40	CDC Cyber-205	CDC-7600	[6]
Brown, Mendelson	neutron transport, continuous energy, 2-D	20–85	CDC Cyber-205	CDC-7600	[11]
Bobrowicz et. al., Fisher	photon transport, 2-D	4–6	Cray-1, XMP	Cray-1	[12,13]
Chauvet	neutron transport, 2-D	7–13	Cray-1, XMP	Cray-1	[14,15]
Martin, Nowak and Rathkopf	photon transport, 2-D	4–9	Cray-1, XMP, Fujitsu VP-200	Cray-1	[18]
Asai, Higuchi and Katakura	neutron transport, multigroup, 3-D	1.4	Fujitsu VP-200	same	[19]
Brown	neutron transport, continuous energy, 3-D	> 10	Cyber-205	CDC-7600	[21,22]
Miura	high energy physics (EGS4)	8	Amdahl 1200	same	[23]
Youssef	ray tracing	20	ETA-10	same	[25]
Dekeyser	high energy physics (GEANT)	–	Cray, ETA-10	same	[26,27]

many cases, throwing away the original Monte Carlo code and starting over) to obtain these speedups.

6. Concluding remarks

• *Particle transport Monte Carlo has been successfully vectorized.* Although the conventional Monte Carlo algorithm is inherently scalar in nature due to its history-based structure, the event-based algorithm has been shown to be very effective at exploiting vector architectures. While the first reported results with the vectorized algorithms were with "restricted" Monte Carlo, it is now clear that general geometry, general-purpose, continuous-energy Monte Carlo can be vectorized with excellent speedups.

• *The vectorization of a Monte Carlo code is a significant undertaking.* It is clear from the successful vectorization efforts that global algorithmic changes are necessary – comprehensive changes to the data structures and possibly a complete rewrite of the code. This degree of effort may not be possible, or affordable, but is probably essential to achieve significant speedups. If this substantial investment is not taken, then one might not enjoy optimum results, as evidenced by the disappointing results with the KENO-IV vectorization, which was encumbered by the structure of the original KENO-IV code.

• *The next challenge is multitasking.* There is no question that Monte Carlo can be parallelized, the principal question is the ease of implementation. For example, the Winfrith Atomic Energy Establishment has recently implemented its production-level, general-purpose, general-geometry Monte Carlo code MONK6 to the Meiko Transputer-based parallel processor with excellent results [33]. In addition, Martin et al. [34] have reported results with photon transport on both shared-memory and distributed-memory parallel processors. But the case for multiple vector processors is not so apparent. The question that needs to be addressed is: How efficient will a vectorized Monte Carlo algorithm be when implemented on multiple vector processors? For the current generation of modestly-parallel vector processors, such as the Cray X-MP/48 and IBM 3090/400, this may not be a problem. The reason for this confidence is the inherent parallelism of particle transport Monte Carlo and the successful vectorization efforts as reported above. That is, once the vectorized algorithm is developed, implementing it on multiple vector processors is as straightforward as implementing conventional Monte Carlo on multiple scalar processors. However, as the number of processors grows, the efficiency may become unacceptable because of the trade-off between vector length and granularity – as the number of vector processors increases, the average vector length will decrease unless the problem size is also allowed to increase. If one accepts the fact that such machines will only be used for truly large simulations, then acceptable performance will be guaranteed.

7. Acknowledgements

This work was supported by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and IBM Corporation (Kingston Laboratory). Portions of this paper are based on two previous review papers by the author [5,35].

8. References

[1] Los Alamos Monte Group, MCNP – A General Monte Carlo Code for Neutron and Photon Transport, LA-7396-M (revised), Los Alamos National Laboratory (1981).
[2] E.A. Straker, W.H. Scott and N.R. Byrn, The MORSE General Purpose Monte Carlo Multigroup Neutron and Gamma Ray Transport Code with Combinatorial Geometry, USAEC Report ORNL-4585 (1970).
[3] R. Brun, F. Bruyant, M. Maire, A.C. McPherson and P. Zanarini, GEANT3 User's Guide, CERN DD/EE/84-1, CERN, Geneva, Switzerland (May 1986).
[4] E. Troubetzkoy, H. Steinberg and M. Kalos, Trans. Am. Nucl. Soc. 17 (1973) 260.
[5] W.R. Martin and F.B. Brown, Int. J. Supercomput. Appl. 1 (1987) 11.
[6] F.B. Brown and W.R. Martin, Prog. Nucl. Energy 14 (1985) 269.
[7] F.B. Brown, W.R. Martin and D.A. Calahan, Trans. Am. Nucl. Soc. 39 (1981) 755.
[8] F.B. Brown, Ph.D. thesis, The University of Michigan, Ann Arbor, Michigan (1981).

[9] W.R. Martin, Vectorized Monte Carlo on the Cyber-205, Final Report for Control Data Corporation, University of Michigan (1983).

[10] F.B. Brown, in: Proc. Am. Nucl. Soc. Topical Meeting on Advances in Reactor Computations, Salt Lake City (1983), p. 108.

[11] F.B. Brown and M.R. Mendelson, Trans. Am. Nucl. Soc. 46 (1984) 727.

[12] F.W. Bobrowicz, J.E. Lynch, K.J. Fisher and J.E. Tabor, Parallel Comput. 1 (1984) 295.

[13] K.J. Fisher, Vectorized Monte Carlo Radiation Transport, LA-UR-86-3737, Los Alamos National Laboratory (1986).

[14] Y. Chauvet, Cray Channels 6 (1984) 3.

[15] Y. Chauvet, Vectorization and Multitasking with a Monte Carlo Code for Neutron Transport Problems, in: LANL-CEA Meeting on Recent Applications of the Monte Carlo Method, CEA-CONF 7902 (1985).

[16] W.R. Martin and D.A. Calahan, Trans. Am. Nucl. Soc. 43 (1982) 399.

[17] W.R. Martin, J.A. Rathkopf and P.F. Nowak, Trans. Am. Nucl. Soc. 50 (1985) 278.

[18] W.R. Martin, P.F. Nowak and J.A. Rathkopf, IBM J. Res. Dev. 30 (1986) 193.

[19] K. Asai, K. Higuchi and J. Katakura, Nucl. Sci. Eng. 92 (1986) 298.

[20] J.T. West, L.M. Petrie and J.K. Fraley, KENO-IV/CG. The Combinatorial Geometry Version of the KENO Monte Carlo Criticality Safety Program, ORNL /NUREG/CSD-7, Oak Ridge National Laboratory (1979).

[21] F.B. Brown, Trans. Am. Nucl. Soc. 53 (1986) 283.

[22] F.B. Brown and F.G. Bischoff, Computational Geometry for Reactor Applications, in: Proc. Am. Nucl. Soc. Winter Meeting, Washington, DC (1988).

[23] K. Miura, Comput. Phys. Commun. 45 (1987) 127.

[24] S. Youssef, Comput. Graphics Image Process. 18 (1982) 109.

[25] S. Youssef, Vectorized Simulation and Ray Tracing, in: Proc. Workshop on Detector Simulation for the SSC, Argonne National Laboratory (August 1987); Supercomputer Computations Research Institute Technical Report FSU-SCRI-87-63, Florida State University (1987).

[26] J.-L. Dekeyser, Nucl. Instrum. Methods A 264 (1987) 291.

[27] J.-L. Dekeyser, Technical Report on the Vectorization of GEANT3, Supercomputer Computations Research Center Report FSU-SCRI-87-63, Florida State University (1987).

[28] S. Youssef, W. Martin, T.C. Wan and S. Wilderman, Comput. Phys. Commun. 57 (1989) 251.

[29] C.H. Georgiopoulos and M.E. Mermikides, Comput. Phys. Commun. 57 (1989) 255.

[30] M.J. Corden, C.H. Georgiopoulos and M.E. Mermikides, Comput. Phys. Commun. 57 (1989) 260.

[31] U. Chandra, G. Riccardi, J. Vagi, J.L. Dekeyser and F. Hannedouche, Comput. Phys. Commun. 57 (1989) 263.

[32] M.J. Corden, C.H. Georgiopoulos, R. Brun, F. Bruyant and J.L. Dekeyser Comput. Phys. Commun. 57 (1989) 268.

[33] R.J. Brissenden, UKAEA Winfrith, private communication (March 1989).

[34] W.R. Martin, T.C. Wan, T. Abdel-Rahman, and T.N. Mudge, Int. J. Supercomput. Appl. 1, No. 3, (1987) 57.

[35] W.R. Martin, Particle Transport Monte Carlo on Vector and Parallel Architectures, in: Sixth IMACS Int. Symp. Computer Methods for Partial Differential Equations, Bethlehem, PA (June 1987).