JAERI-M
88-114

中性子輸送計算用モンテカルロ・コードのベクトル化技法

1988年6月

菅沼　正之*・樋口　健二・栗田　豊**・浅井　清

日本原子力研究所
Japan Atomic Energy Research Institute

JAERI-M レポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所技術情報部情報資料課（〒319—11 茨城県那珂郡東海村）あて，お申しこしください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11 茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

JAERI-M reports are issued irregularly.
Inquiries about availability of the reports should be addressed to Information Division, Department of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

編集兼発行　日本原子力研究所
印　　刷　山田軽印刷所

JAERI-M 88-114

中性子輸送計算用モンテカルロ・コードのベクトル化技法

日本原子力研究所東海研究所計算センター
菅沼　正之*・樋口　健二・栗田　豊**
浅井　清

（1988年5月30日受理）

計算速度の向上を目的として中性子輸送計算用モンテカルロ・コードKENO IV, MORSE－DD, MCNP, VIMのベクトル化がそれぞれ日本原子力研究所において既に試みられているが，ベクトル化によるオリジナル版に対する速度向上比はいづれも1.3から1.5倍程度に留まっている。本報告書は，これら4つの汎用大型モンテカルロ・コードのベクトル化について総合的に検討したものであり，モンテカルロ・コードのベクトル化のための分析手法，制御構造の変更法およびデバッグの手法について記している。なお，これらのコードのオリジナル版に対する速度向上比が低い原因についても検討を加えた。

東海研究所：〒319-11　茨城県那珂郡東海村白方字白根2-4
＊　外来研究員　富士通エフ・アイ・ピー株式会社
＊＊（財）原子力データ・センター

JAERI-M 88-114

Vectorization Techniques for Neuton Transport Monte Carlo Codes

Masayuki SUGANUMA*, Kenji HIGUCHI
Yutaka KURITA** and Kiyoshi ASAI

Computing Center
Tokai Research Establishment
Japan Atomic Energy Research Institute
Tokai-mura, Naka-gun, Ibaraki-ken

(Received May 30, 1988)

Four Monte Carlo codes, KENO IV, MORSE-DD, MCNP and VIM, have been vectorized already at JAERI Computing Center aiming at an increase in clculation performance, and speed-up ratios of vectorized codes to the original ones were found to be low values between 1.3 and 1.5. In this report the vectorization processes for these four codes are reviewed comprehensively, and methods of analysis for vectorization, modification of control structures of codes and debugging techniques are discussed. The reason for low speed-up ratios is also discussed.

Keywords: Monte Carlo, Vector, Supercomputer, Nuclear Code, KENO IV, MORSE, VIM, MCNP

---

* On leave from FUJITSU FACOM INFORMATION PROCESSING CORPORATION
** Nuclear Energy Data Center.

目　　次

Contents

1. はじめに

中性子，光子（以降，総称して粒子と呼ぶ）の輸送現象，核分裂物質の臨界現象の解析を行うためには体系中の粒子のふるまいを知ることが必要となる。これらの問題に対する解決法として大別すると次の２つの立場に立った手法がある。

① 粒子の個性を均一化して取り扱う手法

媒質中には非常に多くの粒子が存在する。これらの粒子の個性（ふるまい）を均質化して取り扱うことによって場を支配する方程式を導出し，導出された方程式を適当な方法（解析的な解法，数値的な解法)で解くことによって問題に対する近似解を得る。CITATION, ANISN, DOT などの拡散・輸送コードがこれにあたる。

② 粒子の個性を均一化しないで取り扱う手法

個々の粒子の発生から消滅までの過程を模擬し，粒子の寄与を累積することによって問題に対する近似解を得る。本報告書で述べるモンテカルロ・コードがこれに相当する。

①の立場に立ったコードの最も大きな欠点は問題の幾何形状の記述に対する制約が大きいため複雑形状を有した問題に対し適用が困難ということがあげられる。つまり，差分近似を使用して方程式の近似解を求めているために，空間メッシュ分割が必要であり，三次元の複雑な幾何形状を正確に表現することが出来ない。また，ダクトなどの空隙を含むような形状では，空隙を通りぬける粒子の微妙な速度方向の表現が重要となってくるが，角度方向に関する差分近似のため正確に取り扱えない。

これに対し，②の立場に立ったモンテカルロ・コードでは幾何形状の記述に対する制限が小さいため複雑幾何形状の問題にも適用が可能であるという長所を持つ反面，十分な計算精度を得るためには多量の粒子の追跡が必要となり，非常に多くのCPU時間を必要とするという短所がある。

問題の対象がシステムを構成するコンポーネントからシステム全体に拡がりつつある現在，複雑形状を正確に取り扱うことが可能なモンテカルロ・コードの需要はますます高まりつつあり，コードの高速化が望まれている。現時点で実現可能な高速化の手段としては浮動小数点演算能力の高いパイプライン制御方式のベクトル・プロセッサの利用を考えることができる。ベクトル・プロセッサの利用法は２種類あり，ひとつはベクトル・プロセッサのハードウエアを考慮してコードの新規開発を行う方法で Martin, Brown ら[1]によっていくつかのベクトル化モンテカルロ・コードが開発されている。もうひとつは既存のモンテカルロ・コードをベクトル・プロセッサ向けに改造する方法で日本原子力研究所（以降，原研と略称する）でいくつかの既存コード[2),3),4),5),6),7),8)]のベクトル化が行われている。

本報告書は，原研で過去に行われたいくつかの既存のモンテカルロ・コードのベクトル化の経験をもとにまとめられたものである。第２章ではベクトル版モンテカルロ・コードについての基本的な手法を解説し，第３章ではベクトル化のためのコードの分析技法について述べる。第４章ではベクトル化の基本項目について述べ，第５章ではベクトル版のための制御構造の変更法をラ

ンダム・ウオーク制御系と幾何形状処理系を例にとって解説する。第６章はデバッグの手法について解説し，第７章ではベクトル版モンテカルロ・コードのチューニング技法について解説する。第８章では問題点をあげる。

2．ベクトル版中性子輸送モンテカルロ・コードの手法

モンテカルロ法は確率によって支配される物理問題を擬似乱数を使用して直接計算機上で模擬し，問題の近似解を得る手法である。Fig．2.1に例として，粒子の輸送問題に対するモンテカルロ・コードの制御構造の例を示す。網かけされた部分が媒質中での粒子のふるまい，即ち拡散を模擬するために必要な処理系である。処理系は断面積処理系（粒子のエネルギーに対応する媒質の断面積を計算する），幾何形状処理系(粒子の空間座標と問題の幾何形状を構成する区画の間の対応をつける)，評価系(粒子の飛程から粒子束を求める)，衝突解析系(媒質を構成する核種と粒子の衝突後の状態をサンプリングする）及びランダム・ウォークさせる粒子を母集団からサンプリングするための粒子発生系から構成される。Fig. 2.2に示すように粒子を逐次的に処理するために，外側の粒子に関する陽的なループと内側の拡散過程を追跡するための陰的なループの二重ループの制御構造を持っている。

ベクトル化を行うためにはコードの並列性をみいだし，その並列性を利用してベクトル化できる様な形にコードの制御構造を変更することが必要である。輸送問題およびその他の多くの粒子モデルの問題では粒子間の相互作用を考慮する必要がない。このため粒子の挙動は互いに独立であると考えることができ，この独立性を利用すると粒子に対する並列処理が可能となる。粒子の独立性を利用するための制御構造の変更は，Fig. 2.3で示すように粒子のループと拡散のループを交換することである。DOループ１であらかじめ必要な粒子を発生させ，DOループ２で全ての粒子に対して拡散の処理を行なう。DOループ３で吸収，体系からの漏れ等によって消失した粒子を取り除き，生き残っている粒子がある場合にはDOループ２の拡散の処理にもどり，生き残っている粒子がない場合には処理を終了する。粒子が処理されてゆく様子をFig. 2.4に示す。ｎ回目の処理のｎは外側の粒子の拡散のループの繰り返しの回数を示す。拡散のループの繰り返し回数が増加するにしたがって並列に処理できる粒子数が減少してゆくことがわかる。

つぎの第３章ではベクトル化の事前調査として，ベクトル化ルーチンの選定法とベクトル化COMMON変数の抽出方法について述べる。

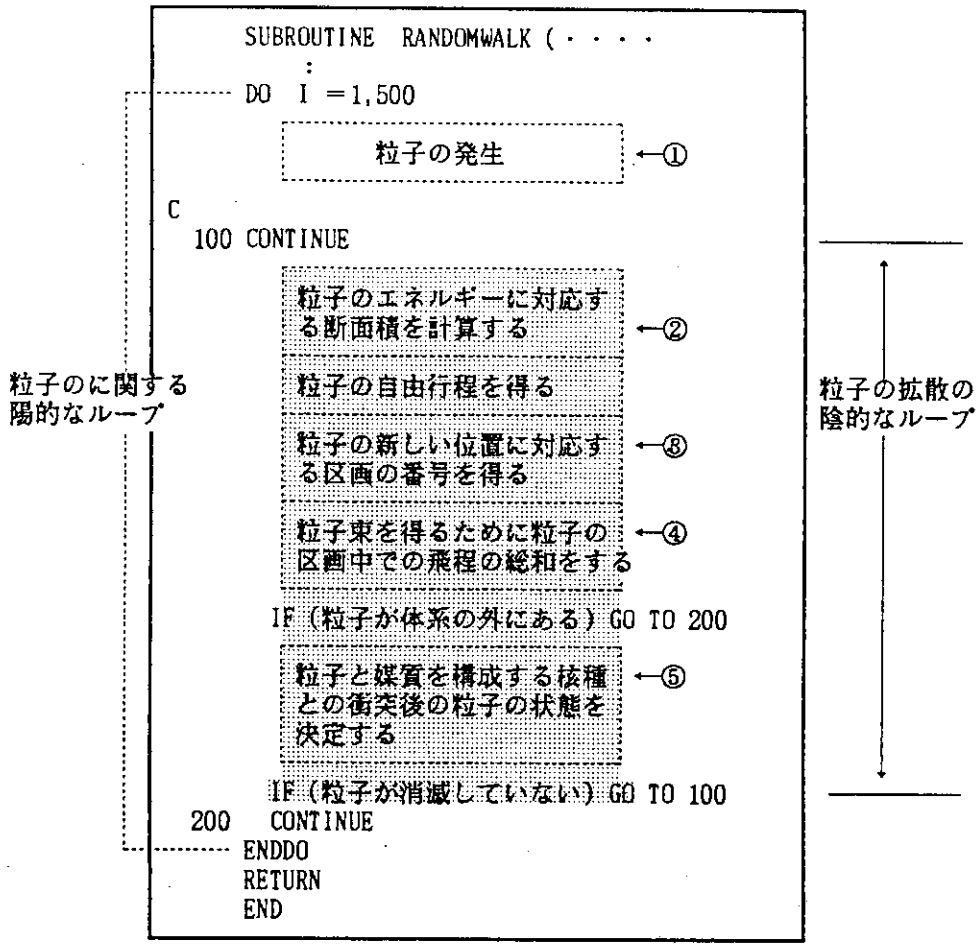
ンダム・ウオーク制御系と幾何形状処理系を例にとって解説する。第6章はデバッグの手法について解説し，第7章ではベクトル版モンテカルロ・コードのチューニング技法について解説する。第8章では問題点をあげる。

2．ベクトル版中性子輸送モンテカルロ・コードの手法

モンテカルロ法は確率によって支配される物理問題を擬似乱数を使用して直接計算機上で模擬し，問題の近似解を得る手法である。Fig．2.1に例として，粒子の輸送問題に対するモンテカルロ・コードの制御構造の例を示す。網かけされた部分が媒質中での粒子のふるまい，即ち拡散を模擬するために必要な処理系である。処理系は断面積処理系（粒子のエネルギーに対応する媒質の断面積を計算する），幾何形状処理系(粒子の空間座標と問題の幾何形状を構成する区画の間の対応をつける)，評価系(粒子の飛程から粒子束を求める)，衝突解析系(媒質を構成する核種と粒子の衝突後の状態をサンプリングする）及びランダム・ウオークさせる粒子を母集団からサンプリングするための粒子発生系から構成される。Fig. 2.2に示すように粒子を逐次的に処理するために，外側の粒子に関する陽的なループと内側の拡散過程を追跡するための陰的なループの二重ループの制御構造を持っている。

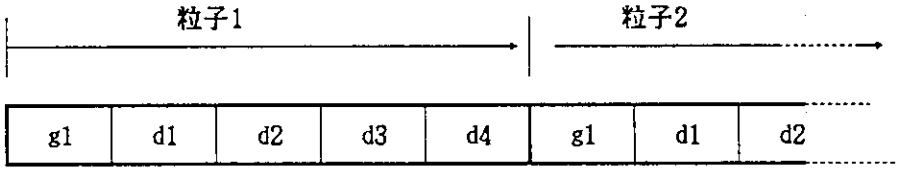
ベクトル化を行うためにはコードの並列性をみいだし，その並列性を利用してベクトル化できる様な形にコードの制御構造を変更することが必要である。輸送問題およびその他の多くの粒子モデルの問題では粒子間の相互作用を考慮する必要がない。このため粒子の挙動は互いに独立であると考えることができ，この独立性を利用すると粒子に対する並列処理が可能となる。粒子の独立性を利用するための制御構造の変更は，Fig. 2.3で示すように粒子のループと拡散のループを交換することである。DOループ1であらかじめ必要な粒子を発生させ，DOループ2で全ての粒子に対して拡散の処理を行なう。DOループ3で吸収，体系からの漏れ等によって消失した粒子を取り除き，生き残っている粒子がある場合にはDOループ2の拡散の処理にもどり，生き残っている粒子がない場合には処理を終了する。粒子が処理されてゆく様子をFig. 2.4に示す。n回目の処理のnは外側の粒子の拡散のループの繰り返しの回数を示す。拡散のループの繰り返し回数が増加するにしたがって並列に処理できる粒子数が減少してゆくことがわかる。

つぎの第3章ではベクトル化の事前調査として，ベクトル化ルーチンの選定法とベクトル化COMMON変数の抽出方法について述べる。



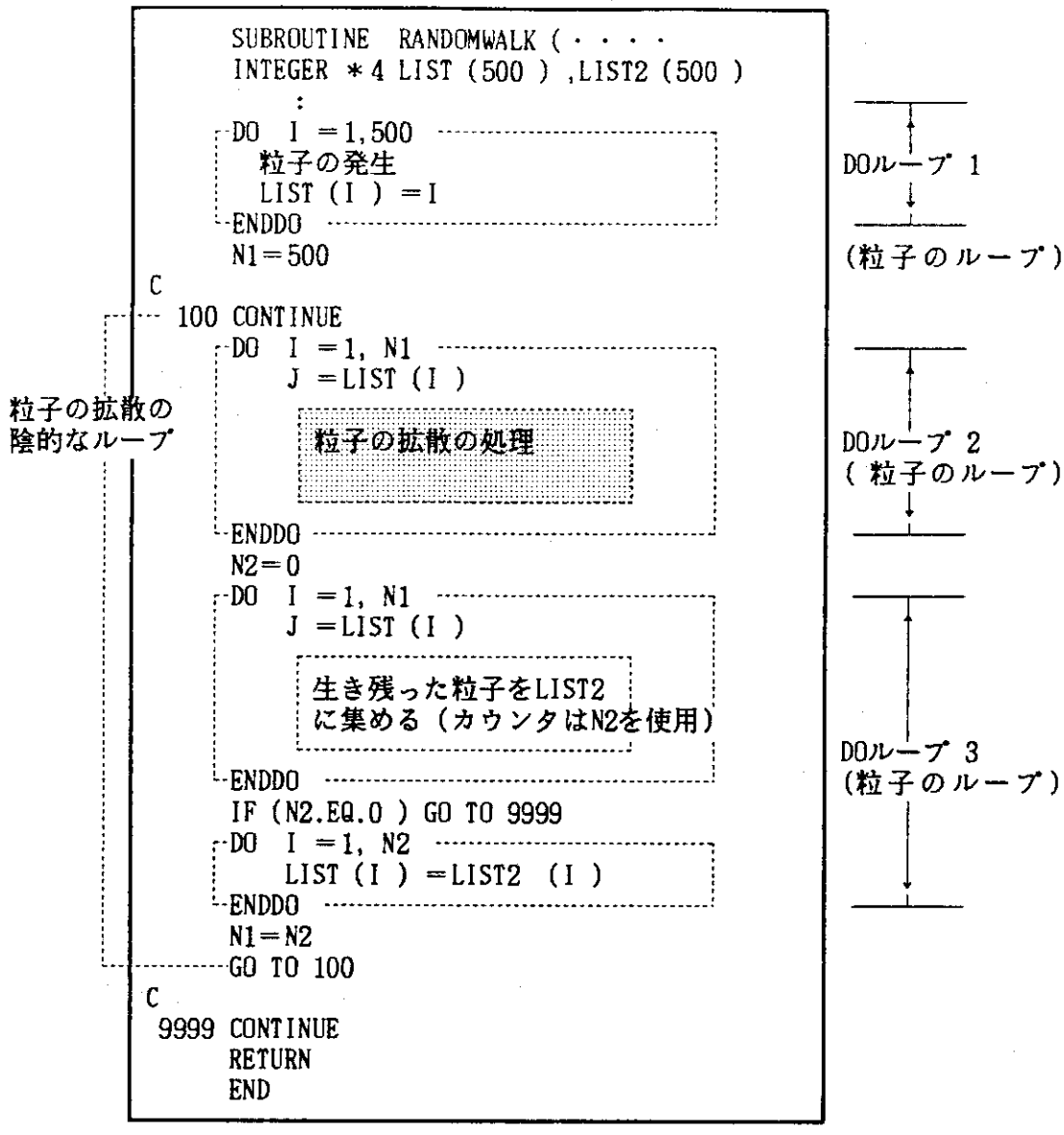
①粒子発生系　②断面積処理系　③幾何形状処理系　④評価系　⑤衝突解析系

Fig. 2.1 An example of control structure of a random walk control routine (original version).



gn：n 番目の粒子の発生，　dn：個々の粒子に対する n番目の拡散処理

Fig. 2.2 Sequence of events of the particles (original version).



DOループ 1：粒子の発生
DOループ 2：粒子の拡散の処理
（断面積処理系，幾何形状系，評価系，衝突解析系を含む）
DOループ 3：生き残った粒子を集める，粒子の拡散の処理の再初期設定を行う

Fig. 2.3 A control structure of a random walk control routine (vector version).

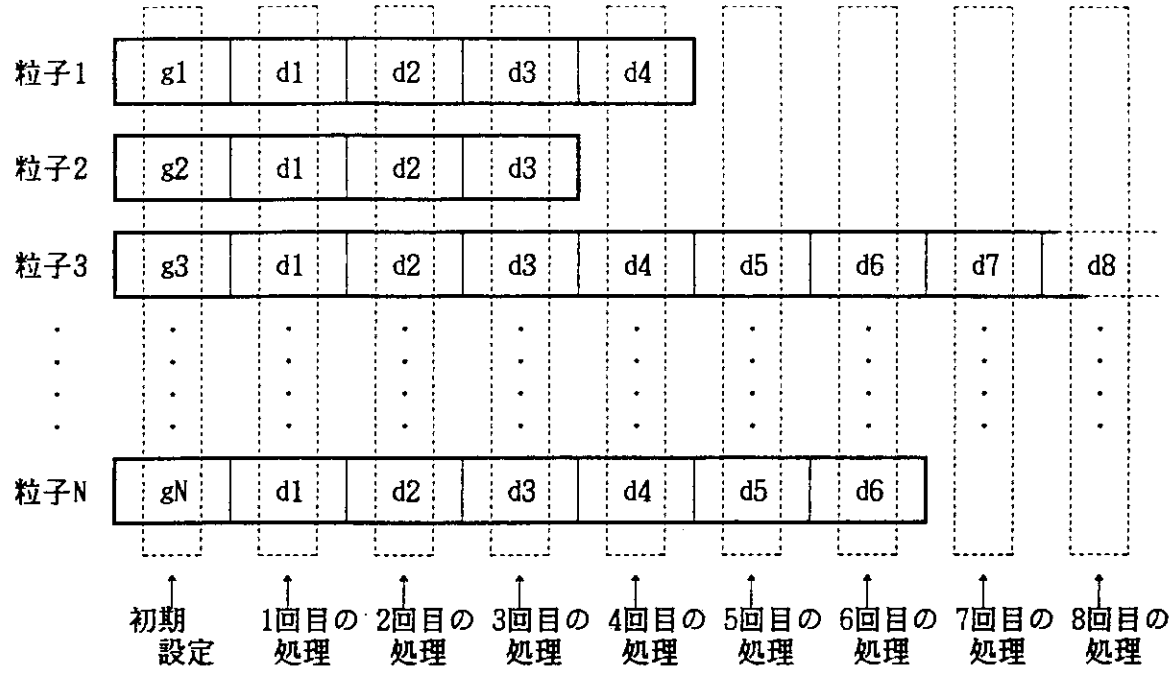


Fig. 2.4 Sequence of events of the particles (vector version).

3. ベクトル化のためのコードの調査

ベクトル化作業を始める前に決定しておかなければならない2つの項目がある。ひとつはベクトル化の作業範囲を決定するためのベクトル化対象ルーチンの選定であり，もうひとつはCOMMON変数によって取られているサブルーチン間のインターフェース改造のための調査である。

3.1 ベクトル化ルーチンの選定

ベクトル化によるコードの高速化はCPU時間の大部分を消費している処理部分のスカラ演算をベクトル演算で置き換えることによって行われる。このため，第一ステップとしてコード中のルーチンの消費しているCPU時間の分布を測定し，CPU時間の消費の集中しているルーチンを抽出することが必要になってくる。ここでは，直接CPU時間の消費率を見る代わりにプログラムの動的挙動解析ツールFORTUNE[9]とその出力解析ツールTOP 10[2]を使用してCPU時間に比例する量であるコストのサブルーチンに対する分布を測定した。これらのツールを使用することにより，サブルーチンのエントリ回数，DOループの平均ベクトル長などのベクトル化作業に有益な付加情報を入手することもできる。Table 3.1は連続エネルギー・モンテカルロ・コードVIMのある入力データに対する動的挙動解析ツールFORTUNEによるコスト分布の例である。この例においては，コスト第1位のサブルーチンCROSS1から第8位のサブルーチンNUTRNGまでと第10位のBIRTH1をベクトル化対象ルーチンとして（第9位のサブルーチンPRINTは出力編集用ルーチンであるため除外）した場合，消費しているCPU時間の約98％を消費しているルーチンがベクトル化の対象となったことになる。

第2ステップとして次の2つの項目を考慮して実際にベクトル化作業を行うルーチンを決定する必要がある。

① コストは低いがベクトル化対象ルーチンにしなければならない例。

Fig. 3.1(a)で示すように下位に高コストルーチンCを持っているルーチンBは上位ルーチンから粒子のループを引き込まないと下位ルーチンCのベクトル化が不可能なため，コストの如何にかかわらずベクトル化対象ルーチンとしなければならない。また，ランダム・ウオークの制御をしているトップ・レベルのルーチンもベクトル版の制御構造を形成するため必ずベクトル化しなければならない。

② 高コストのルーチンをベクトル化対象ルーチンから外さなければならない例。

Fig. 3.1(b)で示すサブルーチンEのようにいたる所で呼び出されているために高コスト・ルーチンになっている場合がある。このような場合，サブルーチンEの呼び出し回数の分布を調査して集中しているようであればその部分のみをベクトル化し，残りの部分はオリジナル版を使用することになる。呼び出し回数の分布がベクトル化非対象部分に分散している場合にはベクトル化対象ルーチンから外す。

3. ベクトル化のためのコードの調査

ベクトル化作業を始める前に決定しておかなければならない2つの項目がある。ひとつはベクトル化の作業範囲を決定するためのベクトル化対象ルーチンの選定であり，もうひとつはCOMMON変数によって取られているサブルーチン間のインターフェース改造のための調査である。

3.1 ベクトル化ルーチンの選定

ベクトル化によるコードの高速化はCPU時間の大部分を消費している処理部分のスカラ演算をベクトル演算で置き換えることによって行われる。このため，第一ステップとしてコード中のルーチンの消費しているCPU時間の分布を測定し，CPU時間の消費の集中しているルーチンを抽出することが必要になってくる。ここでは，直接CPU時間の消費率を見る代わりにプログラムの動的挙動解析ツールFORTUNE[9]とその出力解析ツールTOP10[2]を使用してCPU時間に比例する量であるコストのサブルーチンに対する分布を測定した。これらのツールを使用することにより，サブルーチンのエントリ回数，DOループの平均ベクトル長などのベクトル化作業に有益な付加情報を入手することもできる。Table 3.1は連続エネルギー・モンテカルロ・コードVIMのある入力データに対する動的挙動解析ツールFORTUNEによるコスト分布の例である。この例においては，コスト第1位のサブルーチンCROSS1から第8位のサブルーチンNUTRNまでと第10位のBIRTH1をベクトル化対象ルーチンとして（第9位のサブルーチンPRINTは出力編集用ルーチンであるため除外）した場合，消費しているCPU時間の約98％を消費しているルーチンがベクトル化の対象となったことになる。

第2ステップとして次の2つの項目を考慮して実際にベクトル化作業を行うルーチンを決定する必要がある。

① コストは低いがベクトル化対象ルーチンにしなければならない例。

Fig. 3.1(a)で示すように下位に高コストルーチンCを持っているルーチンBは上位ルーチンから粒子のループを引き込まないと下位ルーチンCのベクトル化が不可能なため，コストの如何にかかわらずベクトル化対象ルーチンとしなければならない。また，ランダム・ウオークの制御をしているトップ・レベルのルーチンもベクトル版の制御構造を形成するため必ずベクトル化しなければならない。

② 高コストのルーチンをベクトル化対象ルーチンから外さなければならない例。

Fig. 3.1(b)で示すサブルーチンEのようにいたる所で呼び出されているために高コスト・ルーチンになっている場合がある。このような場合，サブルーチンEの呼び出し回数の分布を調査して集中しているようであればその部分のみをベクトル化し，残りの部分はオリジナル版を使用することになる。呼び出し回数の分布がベクトル化非対象部分に分散している場合にはベクトル化対象ルーチンから外す。

コードのコスト分布は入力データに依存して変化するため，FORTUNEを使用してベクトル化対象ルーチンを抽出する場合，対抽となっているコードに対して一般的によく使用されている入力データを数ケース用意してコストの分布を調べ，ルーチンの抽出を行うことが必要となる。

3.2 ベクトル化COMMON変数の抽出

オリジナル版では粒子１個づつの逐次処理であったものがベクトル版では粒子に関する独立性を利用した複数粒子の並列処理となる。このための粒子の位置，飛行方向，エネルギーなどの粒子との間に１対１対応の関係がある変数は全て粒子に対してインデックスを付加し(配列化する)，粒子毎に個々に管理しなければならない。粒子に関するインデックスを付加した変数をここではベクトル変数と呼び，インデックスが増加して次元が上がることを昇格（promotion)と呼ぶ。粒子と１対１に対応する属性とは具体的にはつぎのようなものをあげることができる。

- 粒子の物理的属性を記述する変数または配列

  位置，エネルギー，飛行方向，etc

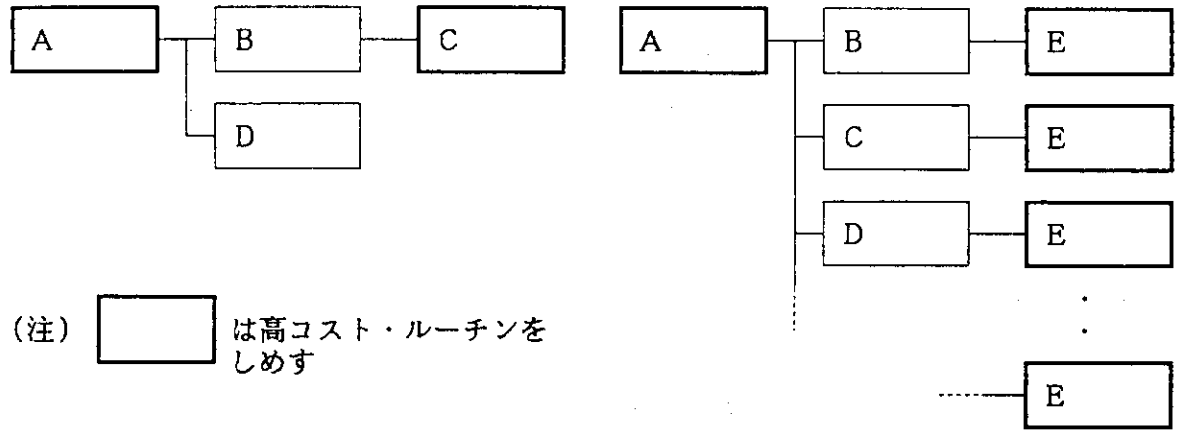
- 粒子の物理的属性ではないが，粒子と１対１の関係が認められる変数または配列

  粒子の重み，年齢，粒子の存在するZONEまたはCELL番号，粒子の状態フラグ，etc

FORTRANでは，変数はルーチン間で共通にアクセス可能なCOMMON変数とルーチン内にアクセス範囲が限定される局所変数の２つに分類される。モンテカルロ・コードのベクトル化ではまず最初にベクトル版のインターフェースの形成を行うためにベクトル変数として昇格されるべきCOMMON変数の抽出を全てのルーチンを対象にして行う。抽出作業の対象範囲を高コスト・ルーチンのみに限ると高コスト部分に存在しないベクトルCOMMON変数の抽出漏れとなりバグの原因とたるため注意する必要がある。ベクトル化COMMON変数の最小集合は一意に定まるのに対し，昇格されるべき局所変数はルーチンの改造の方法に依存する。このため，局所変数の昇格は個々のルーチンの改造方法がきまった段階で決定する。

Table 3.1 An example of execution cost distribution for Monte Carlo code.

NO.	ROUTINE	UNITS	LINES	ERR.	EXECUTIONS	COST	%	0.....1.....2.....3.....
0001	CROSS1	1	268	0	1	450008008	39.2	**********
	CROSS				402934			
0002	REACT1	1	984	0	1	153555486	13.4	**********
	REACT				381560			
0003	GGONE	1	948	0	1	150895303	13.1	**********
	GG				1237481			
0004	FLUXX1	1	209	0	1	143513448	12.5	**********
	FLUXX2				10			
	FLUXX				402934			
0005	ELAST1	1	273	0	1	98564411	8.6	**********
	ELAST				348342			
0006	G1ONE	1	360	0	1	49880109	4.3	*****
	G1				402934			
0007	UNRES1	1	177	0	1	44967095	3.9	****
	UNRES2				1			
	UNRES				248756			
0008	NUTRNG	1	411	0	1	35663250	3.1	***
0009	PRINT	1	580	0	10	7771743	0.7	
0010	BEGET	1	506	0	10	2671657	0.2	
0011	BIRTH1	1	313	0	1	2071579	0.2	
	BIRTH				9010			
0012	POINTR	1	164	0	1	647245	0.1	
0013	SOURCE	1	409	0	11	537495	0.0	
0014	GETIR1	1	205	0	1	344740	0.0	
	GETIR				900			
0015	STATS	1	301	0	1	330812	0.0	
0016	SITES	1	478	0	1	280581	0.0	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	
0106	PRTR2	1	24	0	0			
	PRTA2				0			
0107	CGCHEK	1	171	0	0			
0108	PRTI2	1	12	0	0			
0109	PRTI1	1	15	0	0			
0110	PRTECM	1	58	0	0			
0111	PRLAT	1	36	0	0			
0112	GETHX1	1	166	0	0			
	GETHEX				0			
0113	ALBERT	1	334	0	0			
0114	¥DUMPC	1	89	0	0			
			23158	0		1148790553		



(a) 下位にベクトル化対象高コスト・ルーチンCを持つサブルーチンB

(b) 引用か所の多いために高コスト・ルーチンになっているサブルーチンE

Fig. 3.1 High CPU weight routines in tree structures.

4. ベクトル化技法

モンテカルロ・コードのベクトル化でよく使用されるリスト・ベクトルによるDOループのベクトル化手法と上位ルーチンから引き込んだDOループのベクトル化促進のための分割に対する理論的整合性の保証法について解説する。

4.1 リスト・ベクトルによるDOループのベクトル化

粒子の挙動はたがいに独立であるため，個々の粒子に対して異なった処理が必要となる。このため，ベクトル版モンテカルロ・コードでは，同じ処理を行う粒子の識別番号をインデックス・リストに集め，この粒子の識別番号のインデックス・リストを使ってベクトル処理する方式をとる。例として，Fig. 4.1(a)にオリジナル版のモンテカルロ・コードMCNP[6]の粒子の輸送計算の部分を示し，Fig. 4.1(b)にそれに対応するベクトル版の輸送計算の部分を示す。オリジナル版が条件分岐で分岐してきた粒子に対し輸送計算の処理を行っているのに対し，ベクトル版では，あらかじめ作成しておいた輸送計算が必要な粒子の識別番号のインデックス・リストTRKBKを使用して輸送計算の処理をDOループ中で行っている。粒子の属性に対するアクセスは粒子のインデックス・リストを介した間接アドレス方式となる。ここで使用しているインデックス・リストを特にリスト・ベクトルと呼ぶ。

4.2 粒子のDOループの分割

粒子のDOループに対してベクトル化を行うためには，Fig. 2.3で示すように拡散のループと粒子のDOループを交換し最内ループが粒子のDOループになるようにすることが必要である。しかしループの単純交換のみであると粒子のDOループの中にベクトル化を阻害する要因である外部手続きの参照，複雑な条件分岐などがあってDOループのベクトル化が行われない場合が多い。Fig. 4.2(a)で示すようなベクトル化を阻害する要因を含むDOループのベクトル化を促進させるためにはこの大きなDOループの内部の制御構造を解析してベクトル化を妨げる要因をDOループを分割することによって分離し，Fig. 4.2(b)で示すようなベクトル演算が可能な制御構造の簡単な複数個のDOループを形成することである。一般的にDOループの分割のための情報を与えるFORTRANの文としてはTable 4.1で与えるようなものがある。Fig. 4.2(b)の例では条件分岐，CONTINUE文およびCALL文で処理を分割していることがわかる。処理の分割によって，個々のDOループの構造が簡素化されるため粒子に対するベクトル処理が可能となる。しかし，DOループの分割のために制御構造がオリジナル版と異なってしまうために論理的整合性を保つためにいくつかの付加的な処理の追加が必要となる。Fig. 4.2(b)の例では2つの文番号100への条件分岐およびサブルーチンLOTUSの呼び出し時の論理的整合性があげられる。

4.2.1　条件分岐に対する制御構造の変更

オリジナル版のモンテカルロ・コードでは条件分岐がコードの制御構造の中心的な役割を果たしていたが，ベクトル版モンテカルロ・コードではリスト・ベクトルによって作られたデータの流れがコードの制御構造の中心的役割を果たすことになる。このためDOループの分割時に必要となることは，今まで条件分岐を使って行っていたコードの制御をリスト・ベクトルを使ったデータの流れで実現するということである。条件分岐の多いモンテカルロ・コードのベクトル化ではベクトル版の条件分岐の論理的整合性をとるということが作業時の大きなウエイトをしめることになる。論理的整合のとりかたはＩＦ文の論理式が粒子の属性に依存するか，依存しないか等によって異なる。Fig. 4.3にその対処の例を示す。

①　ＩＦ文の論理式『条件１』が粒子の属性に依存しない場合

すべての粒子に対して論理式は真であるかまたは偽であるかのどちらかであるので，粒子のループを条件分岐の制御構造の中にいれることができる。Fig. 4.3(b)のようにすることで制御構造を保証できる。

②　ＩＦ文の論理式『条件１』が粒子の属性に依存する場合

ＩＦ文の論理式の真，偽は粒子の属性によって異なるので，条件１が偽である粒子に対しては処理２の実行をしなければならない。このためFig. 4.3(c)に示すように，条件１が偽であるような粒子のインデックス・リストＬＩＳＴ２を作り，それを使って処理２を実行するような制御構造をとらなければならない。

③　粒子の属性に依存する論理式を持つ上向きの条件分岐の場合

Fig. 4.4(a)で示すように処理２で設定される粒子の属性を判断基準とする上向きの条件分岐がある場合，ベクトル版ではFig. 4.4(b)に示すように処理２のDOループの処理が終了した後，条件１が真になる（処理２を再び行わなければならない）粒子があるかどうかのチェックとしてＬＩＳＴ２の作成を行う。処理２を再度行う必要のある粒子があれば，ＬＩＳＴ２の内容を処理２のDOループに対応するリスト・ベクトルＬＩＳＴに再セットして100番のCONTINUE文に分岐し処理を行う。このインプリシット・ループは条件１が真になる粒子が無くなるまでくりかえされる。Fig. 2.3もこのような変更がなされていることに注意されたい。

4.2.2　外部手続きの参照に対する制御構造の変更

DOループの分割にともなう外部手続きの参照に対する論理的整合性の保証をとるための方法は問題になっている外部手続きがベクトル化対象となっているか，そうでないかによって二通り考えられる。

①　ベクトル化対象の外部手続きの場合

粒子のループを外部手続きのなかにひきこみ，その外部手続きのベクトル化を行うことによって対処する。

②　ベクトル化対象外の外部手続きの場合

Fig. 4.5で示すように，外部手続きの参照の前後でその外部手続きおよび下位ルーチン群で定義，参照のあるCOMMON変数と対応するベクトルCOMMON変数間のデータの転送を行い，論理的整合性を保証する。

4.2.3　ローカル変数の定義・参照関係の修正

DOループの分割にしたがってローカル変数の定義・参照関係の修正が必要になる。Fig. 4.6の(a)で示すようにDOループの前半の部分でローカル変数Iが定義され，後半の部分でローカル変数Iが参照されていたとする。これをFig. 4.6(a)で示すような所で2つのDOループに分割すると，ローカル変数Iに対する定義・参照の順序関係が分割前と異なってしまうため矛盾をきたす。このため，ローカル変数に対しても粒子に関するインデックスを設けて定義・参照関係の修正をすることが必要となる。この調査では特にDOループの規模が大きい場合，不必要なローカル変数の次元の昇格，あるいは逆の昇格漏れをふせぐためにコンパイラの出力するクロス・リファレンスなどを使用して定義・参照の関係をチェックすることが望ましい。不必要なローカル変数の昇格はむだなメモリへのロード・ストアを発生するためにロード・ストア・ネックによるベクトル処理効率の低下をまねき，昇格漏れはバグを意味するからである。

Table 4.1　FORTRAN statements which give the timing of partitioning of DO loop.

文の種類	注意事項
IF文	論理式が粒子の属性に依存しない 論理式が粒子の属性に依存する
DO文と その制御範囲	粒子のDOループと可換である 粒子のDOループと可換でない
CONTINUE文	上向きの分岐に対応した 下向きの分岐に対応した
外部手続きの 引用	
GO TO 文	

(a)

```
D=MIN(PMF,DLS)
DT=D/VEL
TME=TME+DT
XXX=XXX+UUU*D
YYY=YYY+VVV*D
ZZZ=ZZZ+WWW*D
EGO=ERG
```

(b)

```
*VOCL LOOP,NOVREC
      DO 100 JJ=1,NTRK
         II=TRKBK(JJ)
         D¥(II)=MIN(PMF¥(II),DLS¥(II))
         DT¥(II)=D¥(II)/VEL¥(II)
         TME¥(II)=TME¥(II)+DT¥(II)
         XXX¥(II)=XXX¥(II)+UUU¥(II)*D¥(II)
         YYY¥(II)=YYY¥(II)+VVV¥(II)*D¥(II)
         ZZZ¥(II)=ZZZ¥(II)+WWW¥(II)*D¥(II)
         EGO¥(II)=ERG¥(II)
  100 CONTINUE
```

Fig. 4.1 An example of list vector method.

(a) 上位ルーチンから持ち込まれた大きな粒子に関してのDOループ

```
    SUBROUTINE    PROB1  (LIST,LNG)
    PARAMETER  (NP＝250,MX＝50)
    INTEGER ＊4   LIST (NP)
    DO：粒子のループ
        処理  1
        IF (条件1 )   GOTO 10
        処理  2
        IF (条件 2)   GOTO 10
        処理  3
10  CONTINUE
        処理  4
        CALL  LOTUS
        処理  5
    ENDDO
    RETURN
    END
```

(b) ベクトル演算可能なDOループに分割する

```
    SUBROUTINE    PROB1  (LIST,LNG)
    PARAMETER  (NP＝250,MX＝50)
    INTEGER ＊4   LIST (NP)
    DO粒子のループ
        処理  1
    ENDDO
    IF (条件1 ) GO TO 10 (注)
    DO粒子のループ
        処理  2
    ENDDO
    IF (条件 2) GO TO 10 (注)
    DO粒子のループ
        処理  3
    ENDDO
10  CONTINUE              (注)
    DO粒子のループ
        処理  4
    ENDDO
    DO粒子のループ
    CALL  LOTUS           (注)
    ENDDO
    DO粒子のループ
        処理  5
    ENDDO
    RETURN
    END
```

(注) 整合性をとる必要あり

Fig. 4.2 An example of partition of a particle DO loop.

(a) 大きなDOループのなかの条件分岐による制御構造

```
      SUBROUTINE    PROB2 (LIST,LNG)
      INTEGER *4    LIST (*)
            :
            :
    DO  I =1,LNG
        J =LIST (I )
        ┌──────────────────┐
        │    処理  1       │
        ├──────────────────┤
        │ IF (条件1 ) GO TO 10 │
        ├──────────────────┤
        │    処理  2       │
        ├──────────────────┤
10      │ CONTINUE         │
        ├──────────────────┤
        │    処理  3       │
        └──────────────────┘
    ENDDO
    RETURN
    END
```

(b) 条件分岐の論理式が粒子の属性に依存しない場合のベクトル化例

```
      SUBROUTINE    PROB2 (LIST,LNG)
      INTEGER *4    LIST (*)
            :
            :
            :
            :
        ┌ DO粒子のループ ──────────┐
        │   処理  1 (LIST使用)     │
        └ ENDDO ──────────────────┘
        IF (条件1 ) GO TO 10
        ┌ DO粒子のループ ──────────┐
        │   処理  2 (LIST使用)     │
        └ ENDDO ──────────────────┘
10      CONTINUE
        ┌ DO粒子のループ ──────────┐
        │   処理  3 (LIST使用)     │
        └ ENDDO ──────────────────┘

      RETURN
      END
```

(c) 条件分岐の論理式が粒子の属性に依存する場合のベクトル化例

```
      SUBROUTINE    PROB2 (LIST,LNG)
      PARAMETER ( NP=250 )
      INTEGER *4 LIST (*) ,LIST2 (NP)
            :
            :
            :
        ┌ DO粒子のループ ──────────┐
        │ 処理1                    │
        │ (リスト・ベクトルはLISTを使用) │
        └ ENDDO ──────────────────┘
        ┌ DO粒子のループ ──────────┐
        │ 処理2 に対する           │
        │   リスト・ベクトルLIST2の作成 │
        └ ENDDO ──────────────────┘
        ┌ DO粒子のループ ──────────┐
        │ 処理2                    │
        │ (リスト・ベクトルはLIST2を使用) │
        └ ENDDO ──────────────────┘
        ┌ DO粒子のループ ──────────┐
        │ 処理3                    │
        │ (リスト・ベクトルはLISTを使用) │
        └ ENDDO ──────────────────┘
      RETURN
      END
```

←条件1 が真とならない粒子のインデックス・リストを作成する

Fig. 4.3 An example of vectorization technique for a conditional branch (1).

(a) 粒子に依存する論理式を持つ上向きの条件分岐ある例

```
      SUBROUTINE    PROB3 (LIST,LNG)
      INTEGER *4    LIST (*)
            :
            :
  DO  I =1,LNG
      J =LIST (I )
        ┌──────────────┐
        │   処理  1    │
        ├──────────────┤
10      │ CONTINUE     │
        ├──────────────┤
        │   処理  2    │
        ├──────────────┤
        │ IF (条件1 ) GO TO 10 │
        ├──────────────┤
        │   処理  3    │
        └──────────────┘
  ENDDO
  RETURN
  END
```

(b) ベクトル化例

```
      SUBROUTINE     PROB3 (LIST,LNG)
      PARAMETER  (NP=250 )
      INTEGER *4 LIST (*) ,LIST2 (NP)
     +           LIST1  (NP)
        LNG1=LNG
        ┌ DO粒子のループ ──────┐
        │  処理1               │
        │ (リスト・ベクトルはLISTをリ │
        │  スト長はLNG を使用)  │
        └ ENDDO ──────────────┘
        ┌ DO粒子のループ ──────┐
        │ LISTをLIST1 COPYする │
        └ ENDDO ──────────────┘
10      CONTINUE
        ┌ DO粒子のループ ──────┐
        │  処理2               │
        │ (リスト・ベクトルはLIST1,リ │
        │  スト長はLNG1を使用)  │
        └ ENDDO ──────────────┘
        LNG2=0
        ┌ DO粒子のループ ──────┐
        │ リスト・ベLIST2 の作成 │
        │ (条件1 が真の粒子を集める, │
        │  LNG2をリスト長とする) │
        └ ENDDO ──────────────┘
      ┌ IF (LNG2.GT.0 ) THEN
      │   ┌ DO ──────────────┐
      │   │ LIST2 のデータをLIST1 に移す │
      │   └ ENDDO ──────────┘
      │   LNG1=LNG2
      │   GO TO 10
      └ ENDIF
C
        ┌ DO粒子のループ ──────┐
        │  処理3               │
        │ (リスト・ベクトルはLIST,  │
        │  リスト長はLNG を使用)  │
        │                      │
        └ ENDDO ──────────────┘
      RETURN
      END
```

Fig. 4.4 An example of vectorization technique for a conditional branch (2).

```
      SUBROUTINE    PROB4  (LIST,LNG, ・・
      INTEGER ＊4 LIST (LNG )
      COMMON／COM ／  D  ----------オリジナル版COMMON
      COMMON／COMV／  D2 (1000)  ----------ベクトル版COMMON

           :
           :
           :

     ┌--- DO  I ＝1, LNG
     :       J ＝LIST (I )
     :       D ＝D2 (J )  ---------- ①
     :       CALL   ORIGIN    ---- ②
     :       D2 (J ) ＝D ---------- ③
     └--- ENDDO

           :
           :
           :
           :
           :
      RETURN
      END
```

```
      SUBROUTINE   ORIGIN
      COMMON /COM/ D
           :
           :
            = D
           :
           :
           :
      D = ・・・・・・・
           :
      RETURN
      END
```

①ルーチンORIGIN　以下で参照しているオリジナル版のCOMMON変数にベクトル版の対応するCOMMON変数からデータのロードを行いオリジナル版ルーチンORIGINの動作環境を整える
②オリジナル版のルーチンORIGINを起動し，処理を行う
③ルーチンORIGIN以下で変更されたオリジナル版COMMON変数の値を対応するベクトル版の変数にセーブする
（注）ルーチンORIGIN　以下ではラベル名COM 上の変数D しか定義，参照を行っていないとする。

Fig. 4.5 An example of interface matching between vector common variables and original common variables.

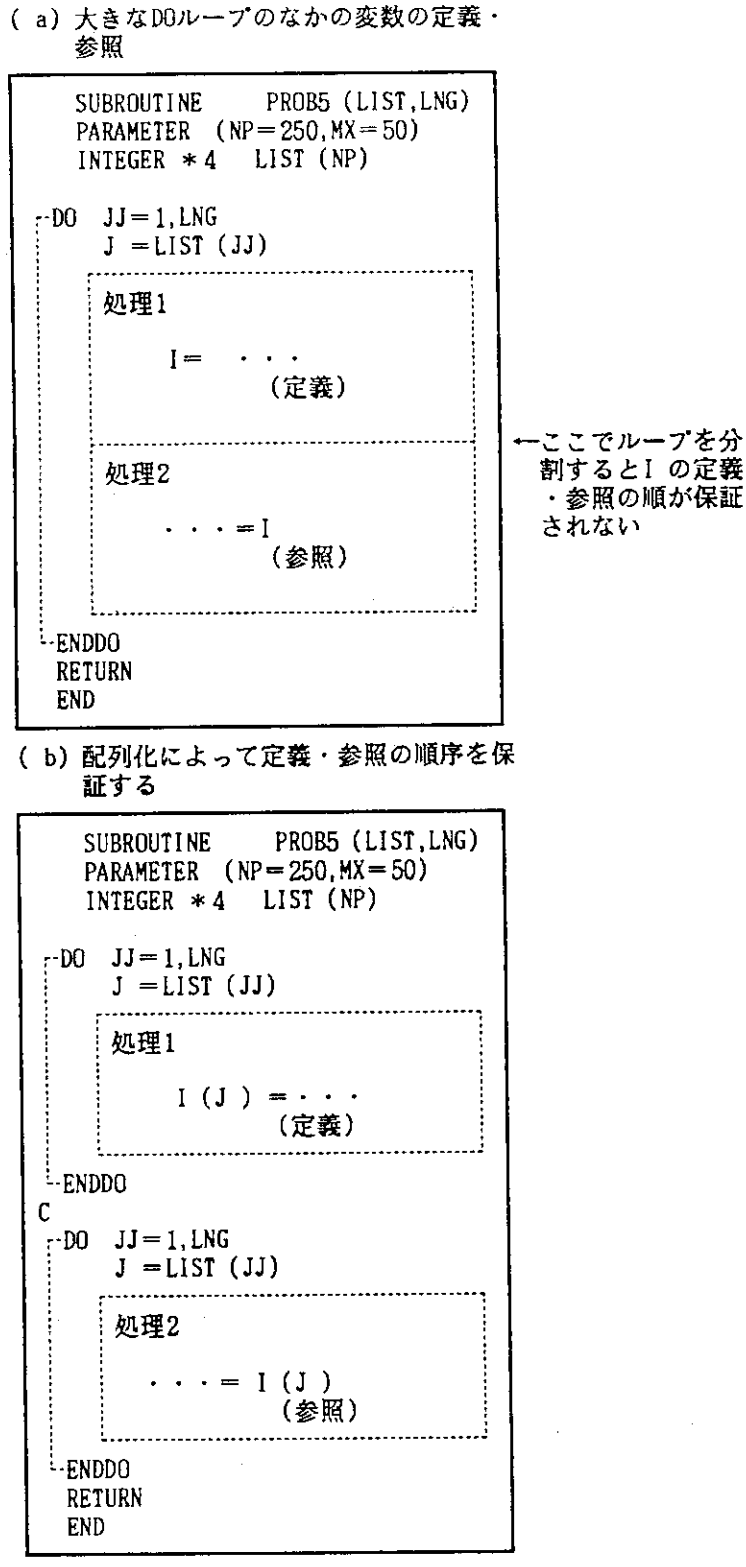


Fig. 4.6 Keep the sequence of definition and reference of variable by promoting the variable to the one dimensional array.

5．ベクトル化のための各処理系の制御構造変更の考え方

第2章で示したように中性子輸送問題の解決のために使用されるモンテカルロ・コードは粒子発生系，断面積処理系，幾何形状処理系，評価系，衝突解析系の5つの処理系とそれを制御するランダム・ウオーク制御系から構成されている。ここではベクトル版モンテカルロ・コードの制御構造を形成するのに必要なランダム・ウオーク制御系の制御構造の変更の考え方と複雑形状下でのベクトル版コードの性能に重大な影響を及ぼす幾何形状処理系の制御構造の変更の考え方について述べる。

5.1 ランダム・ウオーク制御系の制御構造変更の考え方

ランダム・ウオークしてゆく粒子に対してZONEの検索，断面積の計算と衝突後の状態のサンプリングなど，さまざまな処理が必要になる。これらの粒子がランダム・ウオークしてゆく過程に必要となる処理を総称してイベントと呼ぶ。イベントの種類はコードによって様々であるが，ここではFig. 5.1のようなランダム・ウオーク制御系の制御構造を例にとって考える。この例ではイベント1の処理が終了すると粒子は90％の確率でイベント2へ分岐するか，10％の確率でイベント3に分岐するものとする。Fig. 5.1の制御構造をそのままの形でベクトル化した例をFig. 5.2に示す。処理の順番はオリジナル版と同様にイベント1から始まる。次に，イベント2の処理が必要な粒子とイベント3の処理が必要な粒子をリスト・ベクトルに分離した後，イベント2，およびイベント3の処理を行う。

ベクトル処理が効率よく行われるためには各イベント処理部分のDOループのベクトル長が十分大きくなければならない。このような観点からFig. 5.2の例をみた場合，イベント2の処理での平均ベクトル長は分岐率から判断してイベント1の平均ベクトル長の90％前後であるため，イベント1でのベクトル長が大きければベクトル長の極端な低下による処理効率の低下は生じない。しかし，イベント1からイベント3への分岐率は10％であるためイベント3での平均ベクトル長はイベント1の場合の10％程度になってしまいベクトル処理効率の低下が問題になる場合がある。このように条件分岐の多いモンテカルロ・コードのオリジナル版の制御構造をそのままベクトル化した場合は短ベクトル長のもとでのベクトル処理が多発することが考えられ，ベクトル処理効率の低下を引き起こしてしまう場合が多い。

条件分岐によるベクトル長の低下が引き起こすベクトル処理効率の低下を防ぐ手法としてFig. 5.3に示すような粒子数の多いイベントを優先処理する方法がある。この方法では1つのイベントの処理が終了すると全イベントを対象としてイベント・バンク中の粒子数がチェックされ，最も粒子数の多いイベントの処理が起動される。このような制御構造をとることでFig. 5.2のようなオリジナル版の制御構造をそのままベクトル化した場合に問題になる短ベクトル長のもとでのベクトル処理の多発による処理効率の低下を防ぐことができる。

5.2 幾何形状処理系の制御構造変更の考え方

幾何形状を定義するためのデータ構造をMORSE[7]を例に解説する。Fig. 5.4に示すような球に円柱を埋め込んだ体系を考える。体系は各部の材質の違いにより3つの区画に分けて考える必要があると仮定する。MORSEではこの区画をzoneと呼ぶ。ここでは，円柱に抉られた球をzone 1，円柱と球の共通部分をzone 2，球にえぐられた円柱をzone 3とする。zoneはあらかじめ何種類か用意されているbodyと呼ばれる幾何形状の構成要素を使って定義される。この場合，bodyとして球（body 1）と円柱（body 2）を選択すると，各zoneは次のように定義される。

zone 1：body 1の内側かつbody 2の外側
zone 2：body 1の内側かつbody 2の内側
zone 3：body 1の外側かつbody 2の内側

モンテカルロ・コードの幾何形状処理系の役目は粒子の属するzoneを検索することである。このため，粒子があるzoneの内側に存在するのか外側に存在するのかということを判断することが主要な処理内容となる。粒子がzoneに属しているかどうかの判断はzoneを構成している複数個のbodyに対して粒子の座標が条件を満たしているかどうかをチェックしそれを総合して判断を行う。ベクトル版では複数粒子を同時に処理する必要が生じる。各zoneの定義は一般的には異なっているため，粒子に対するベクトル処理を行うためには処理条件の同一な粒子をあらかじめ分類しておく必要が生じる。分類の方法としては次の2種類があるがこれをFig. 5.5に示すようなもうすこし複雑な幾何形状の定義の例を使って説明する。

① 粒子の属しているzoneで粒子を分類する方法

粒子の属しているzoneが同じであればチェック対象となるbodyも同じであるため，同一zone番号を持つ粒子に関しては処理内容が同じとなりベクトル処理が可能となる。Fig. 5.6の例で示すように，まずzone 1に属する粒子に対してbody 1とbody 2に関するチェックが行われ，次にzone 2に属する粒子に関してbody 1，body 2，body 3およびbody 4に関するチェックを行う。このようにしてzone 1からzone Nまで処理して全体の処理をおえる。すなわち，テーブルの縦方向のスイープを行っていることになる。この分類法の欠点はあらかじめzone番号によって粒子を分類してしまうためにbodyの処理のベクトル長の期待値が全体の1/Nになってしまうことである。このためzoneの数の多い体系の場合はベクトル長の低下による処理性能の低下をひきおこすことになる。

② 粒子に属するzoneを構成するbodyで粒子を分類する方法

粒子を問題の体系を定義するために使用されているbodyで分類する方法である。各粒子について，粒子の属するzoneを構成する最初のbodyのbody番号で粒子を分類しbodyごとにまとめてベクトル処理を行う。Fig. 5.7の①の部分のbodyに関する処理を行ったことになる。次に②の部分にポインタを移動し各粒子の属するzoneを構成する2番目のbodyに関して粒子の分類を行いbodyごとにまとめてベクトル処理を行う。このように，zoneを構成するbodyがなくなるまで処理が繰り返される。Fig. 5.7では①から④までの横方向のスイープの処理の繰り返しを行って全体の処理をおえることになる。

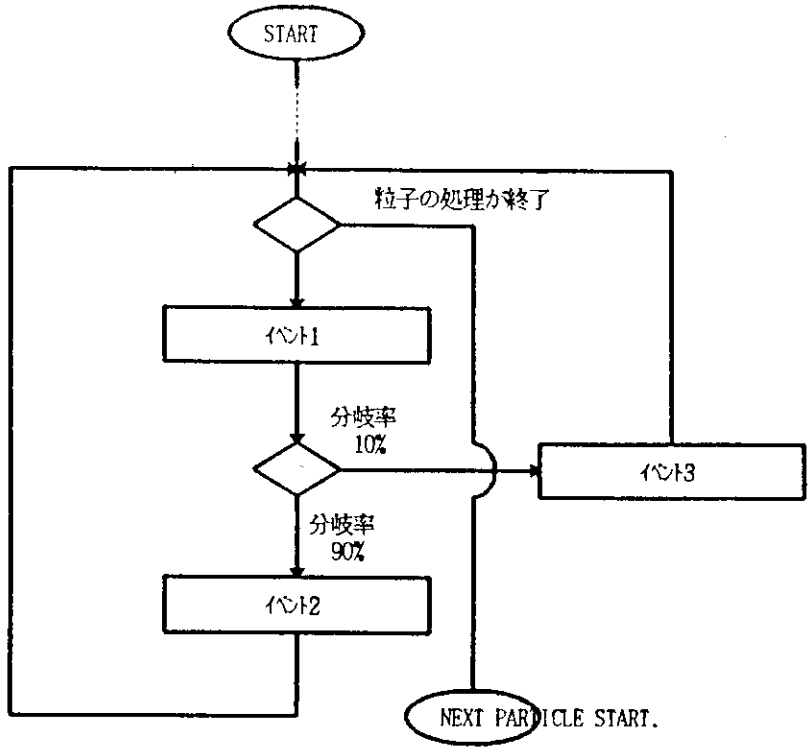


Fig. 5.1 An example of a simple random walk control routine.

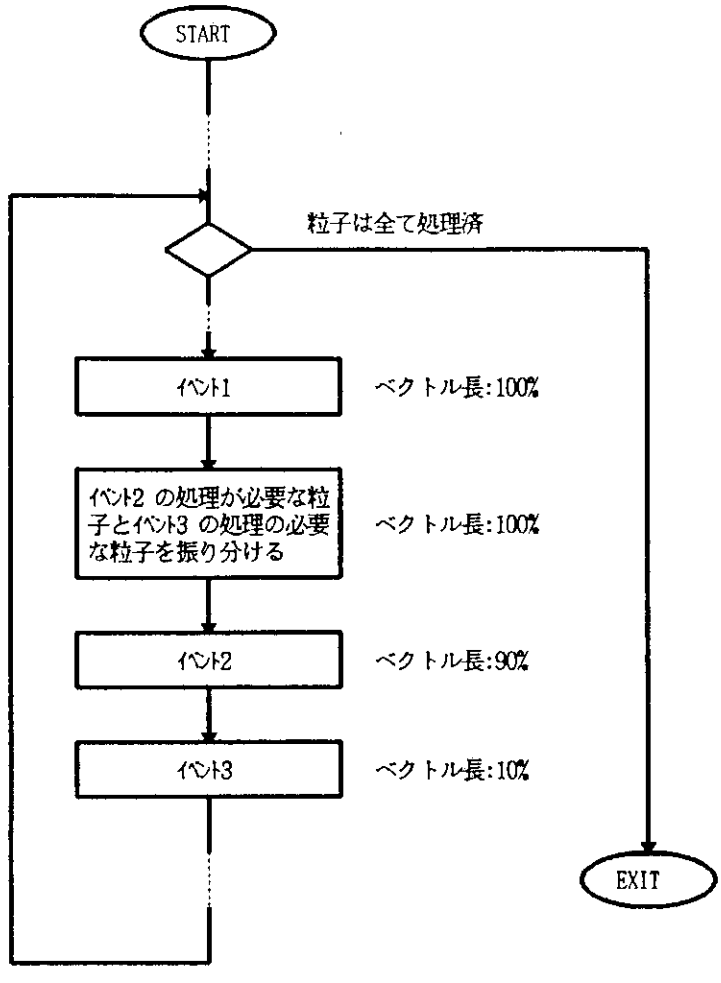


Fig. 5.2 An example of vectorization by using event driven aproach.

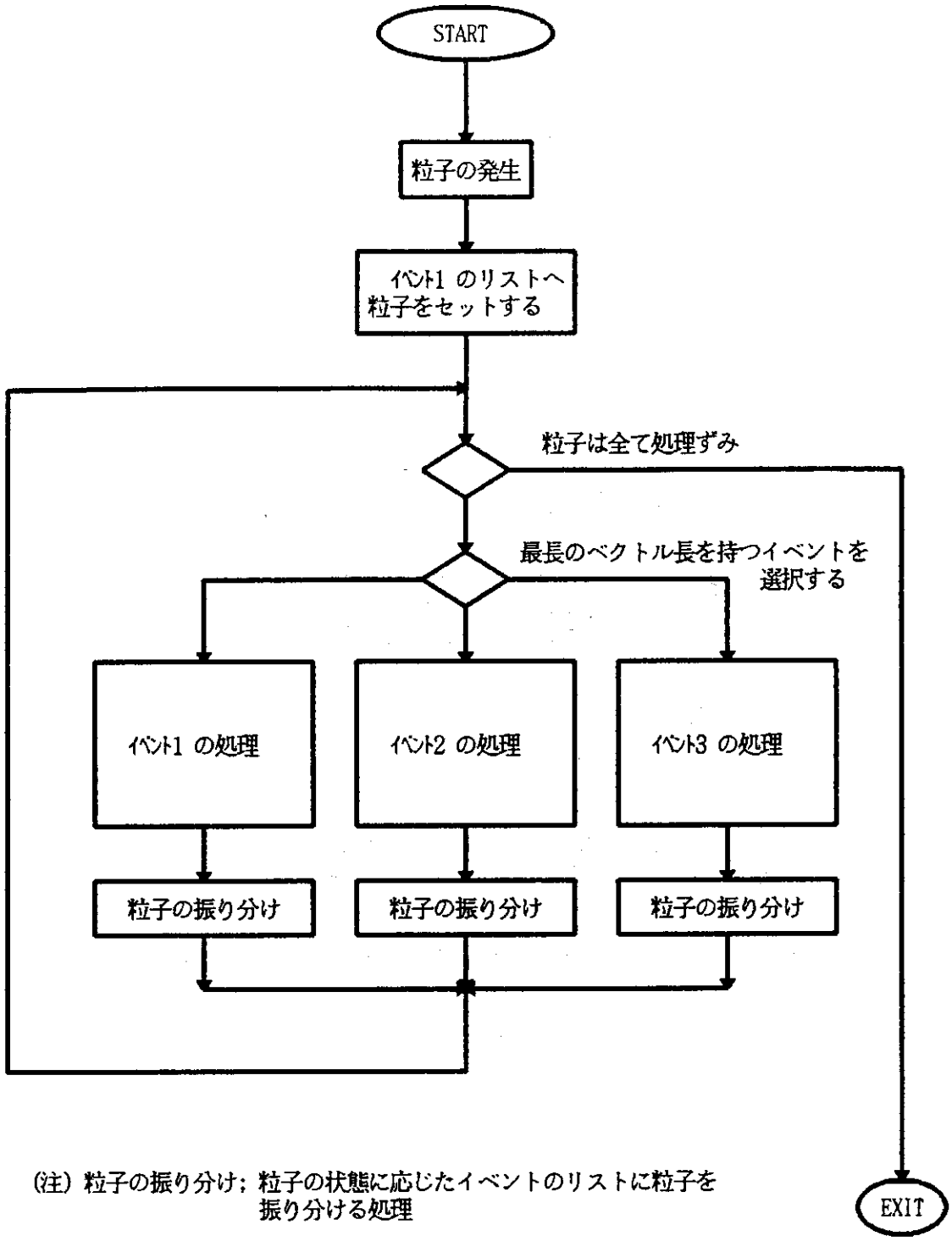
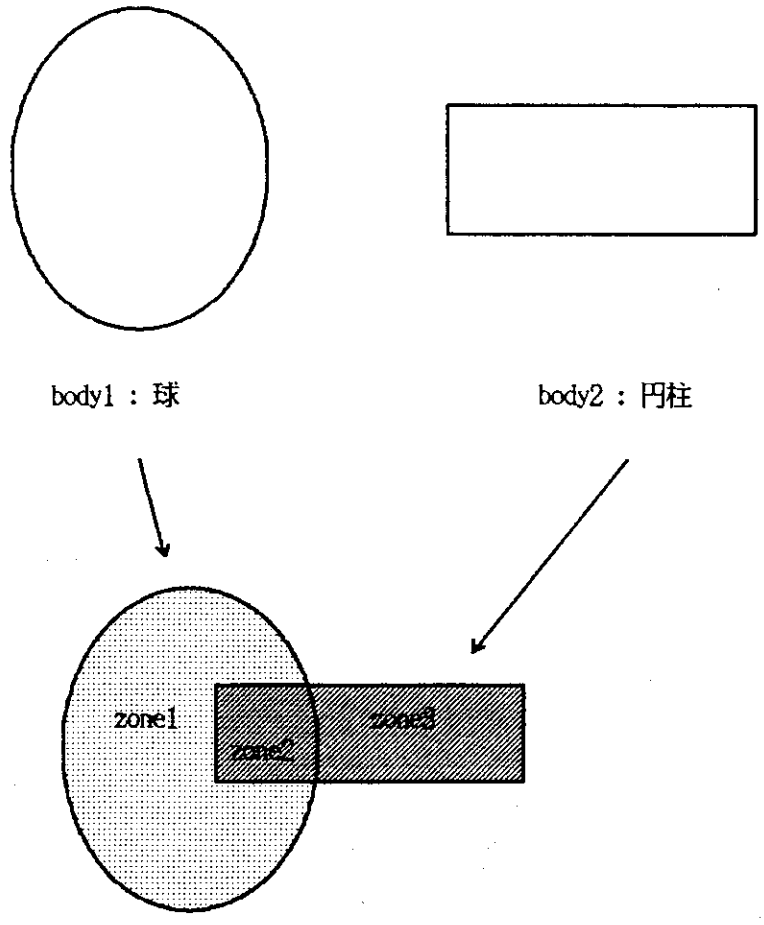


Fig. 5.3 An example of vectorization by using stack driven aproach.



zone1 = (body1 の内側) AND (body2 の外側)
zone2 = (body1 の内側) AND (body2 の内側)
zone3 = (body1 の外側) AND (body2 の内側)

(注) 実際のMORSE における内側, 外側の定義はそれぞれ+,- の演算子を用いる。

Fig. 5.4 Definition of geometric structure in the MORSE code.

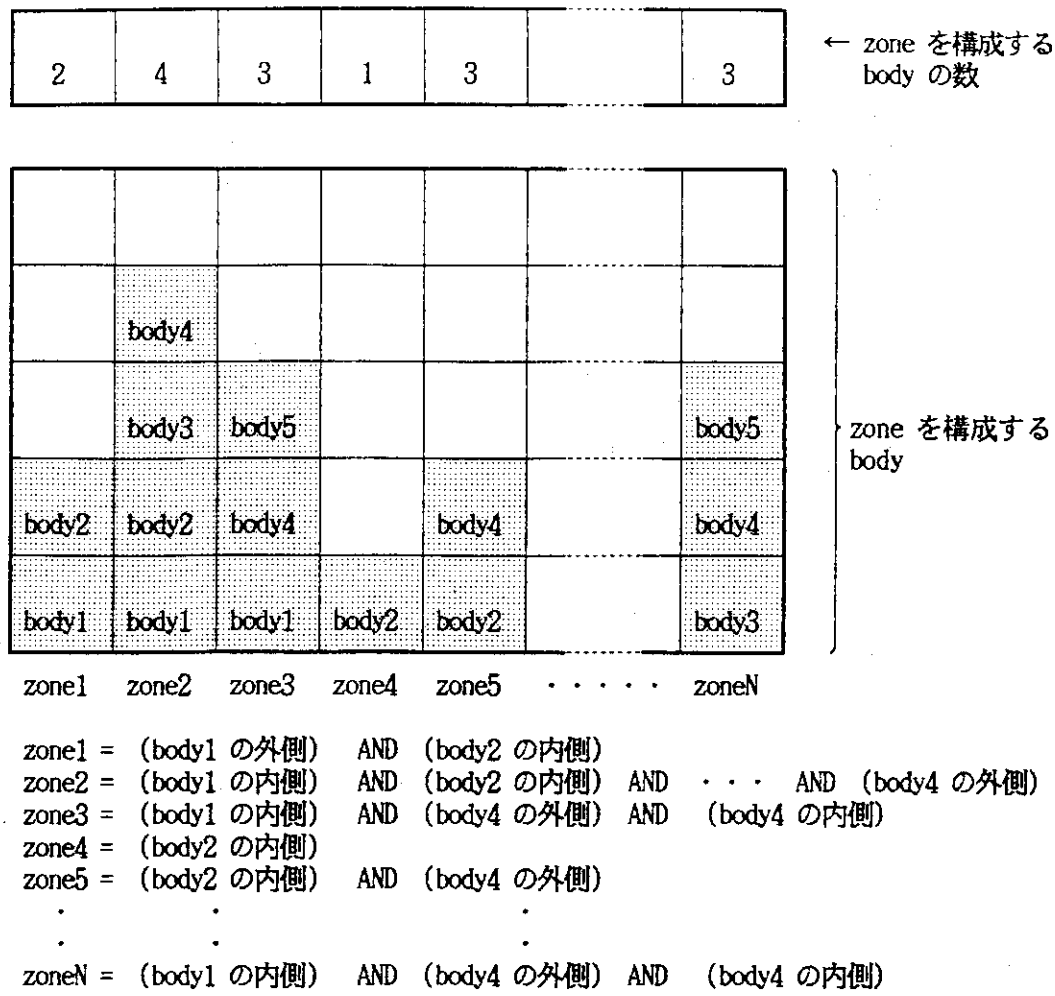
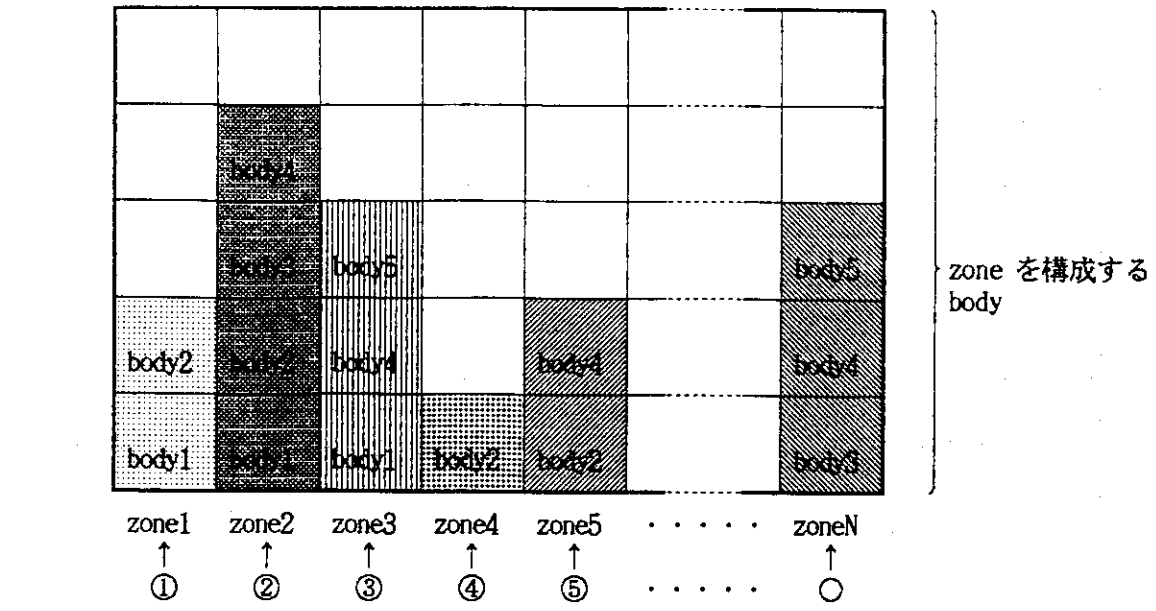
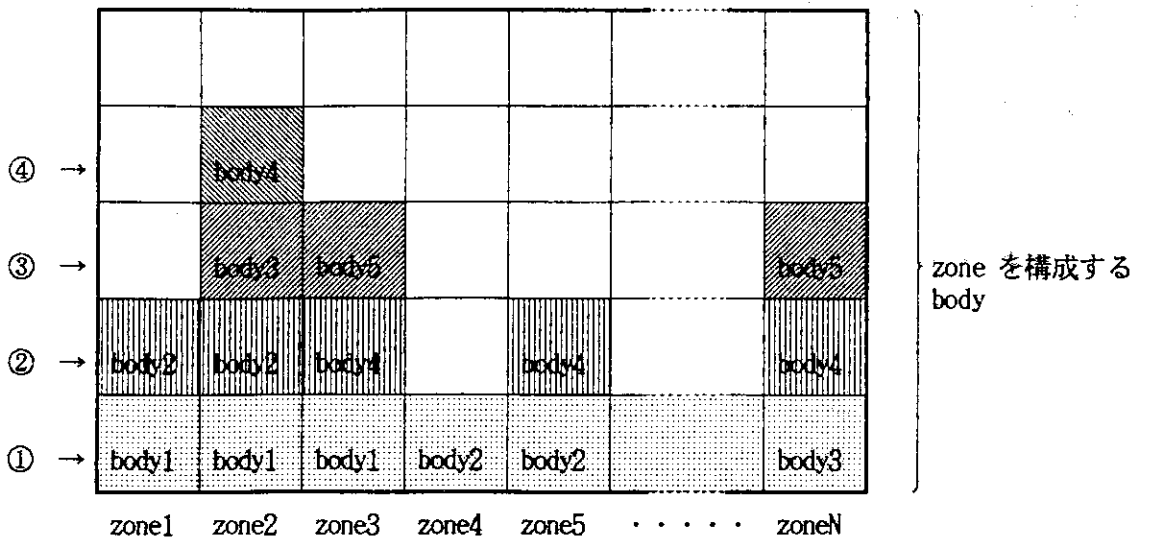


Fig. 5.5 An example of definition of complex geometric structure.



・zoneを定めてそのzoneに含まれる粒子に対しzoneを構成するbodyに関する処理を行う。
・処理は①から○へ向かって行われる（縦方向のスイープ）。

Fig. 5.6 Vertical body sweeping in the vector version geometric routine.



・各粒子に対して粒子の属するzoneを構成するbodyのタイプ毎に粒子リストを作成し処理する。
・zoneは複数個のbodyから構成されるため①から④の方向（横方向）にスイープされる。

Fig. 5.7 Horizontal body sweeping in the vector version geometric routine.

6．デバッグの手法

ベクトル版はオリジナル版との論理的正当性を確認する必要がある。論理的正当性を確認するためのデバッグ・テストは単一粒子・スカラモード，複数粒子スカラ・モード,複数粒子ベクトル・モードの順番でテストを行う。その際に採用した効果的なデバッグ・テストの手順についても述べる。

6.1 ベクトル化対象ルーチンのデバッグの手順

通常のコードのベクトル化作業はルーチン中の既存のDOループを対象としてDOループ単位で行われることが多いためデバッグもDOループ単位に行うことができ，作業範囲は比較的狭い。これに対し，モンテカルロ・コードでは第4章で示したように粒子のループを上位ルーチンから引き込んでベクトル化を行うために，ベクトル化対象ルーチンの大幅な制御構造の変更が必要となる。このためにデバッグは必然的にルーチン単位で行わなければならず作業範囲はかなり広くなる。

ベクトル版コードのデバッグの手順をFig. 6. 1に示す。デバッグはまず，Fig. 6. 1(a)で示すように粒子のランダム・ウオークを制御する最上位のランダム・ウオーク制御ルーチンからはじめる。このとき下位ルーチンはすべてオリジナル版を使用しベクトル化COMMON変数とオリジナル版COMMON変数間のインターフェース整合はFig. 4. 5で示すようなインターフェース整合用サブルーチンを作って行う。次に，ランダム・ウオーク制御ルーチンのデバッグが終了した時点で下位ルーチンのインターフェース整合用ルーチンをはずしてベクトル版の下位ルーチンを組み込んでデバッグを行うことになる。

Fig. 6. 1(b)ではランダム・ウオーク制御ルーチンの下位ルーチンの1つである粒子発生ルーチンのインターフェース整合用ルーチンをはずしベクトル版ルーチンに置き換えてデバッグを行っている例を示している。このようにモンテカルロ・コードのベクトル化ではルーチン単位でインターフェース整合用ルーチンを使ってトップ・ダウン方式でデバッグを進めてゆく形式をとることになる。

6.2 ベクトル化対象ルーチンの論理的正当性の確認法

個々のベクトル版ルーチンに対して次の3項目の動作確認をしなければならない。

① 単一粒子ランダム・ウオーク時のデバッグ・テスト（スカラ・モード）

まず第1段階としてベクトル版ルーチンを使ってオリジナル版と同じ様に粒子を1つづつ逐次的にランダム・ウオークさせ，粒子がオリジナル版と同じ動きをし，計算結果がオリジナル版と一致するかどうかを確認する。ベクトル版ルーチンを使った場合でも単一粒子モードでのランダム・ウオークでは粒子に与えられる乱数列，Estimatorにおける総和の順序等の環境がオリジナル版と同じであるためベクトル版ルーチンの論理的正当性の確認をすることができる。

② 複数粒子ランダム・ウオーク時のデバッグテスト（スカラ・モード）

複数粒子のランダム・ウオークを同時に管理しなければならないベクトル版の論理的正当性の確認は単一粒子モードでの実行結果のみで判断することはできないため複数粒子モードでの動作確認を必要とする。複数粒子モードのデバッグではローカル変数の粒子のインデックスの切り忘れ，ベクトル変数の抽出漏れ（粒子のインデックスの昇格漏れ）等をチェックすることができる。自明ではあるが，起こり得る可能性の高い誤りの例として，Fig. 6. 2に粒子に対するインデックスの昇格漏れの例を示す（粒子のｚ座標を表すzzzの昇格漏れ）。複数粒子モードでは粒子に対するインデックスがないために粒子のｚ座標はループの最後に処理されたものしか保存されていないため，すべての粒子がおなじｚ座標を持ってしまうことになるために論理的な矛盾を生じることになる。単一粒子モードのデバッグではこのようなバグを洗い出すことは出来ないことに注意しなければならない。ベクトルCOMMON変数の抽出漏れに関する問題はローカル変数に対する粒子のインデックス昇格漏れがルーチン内部の局所的な問題であるのに対し，コード全体の広域的な問題になるためソース・コード・レベルで数万枚におよぶようなものでは，再調査が必要となった場合に非常に大きな問題となる。

複数粒子モードでは粒子に対する乱数の割りつけや，Estimator での総和の順序がオリジナル版と異なってしまうため，オリジナル版・ベクトル版の間の比較を単純に行うことは出来なくなってしまう。Fig. 6. 3(a)で示すように単一粒子モードではベクトル版・オリジナル版双方とも横方向に乱数の割りつけ，Estimator での総和が行われるのに対し，複数粒子モードのベクトル版ではFig. 6. 3(b)で示すように縦方向に行われる。ベクトル版の複数粒子モードでの論理的整合性の確認を行うためには粒子に与える乱数列，総和の順番をベクトル版・オリジナル版共に同じにしなければならない。このように，ベクトル版・オリジナル版双方に次のような乱数の管理方法，総和の方法の変更を行うことによって複数粒子モードでのデバッグが可能になる。

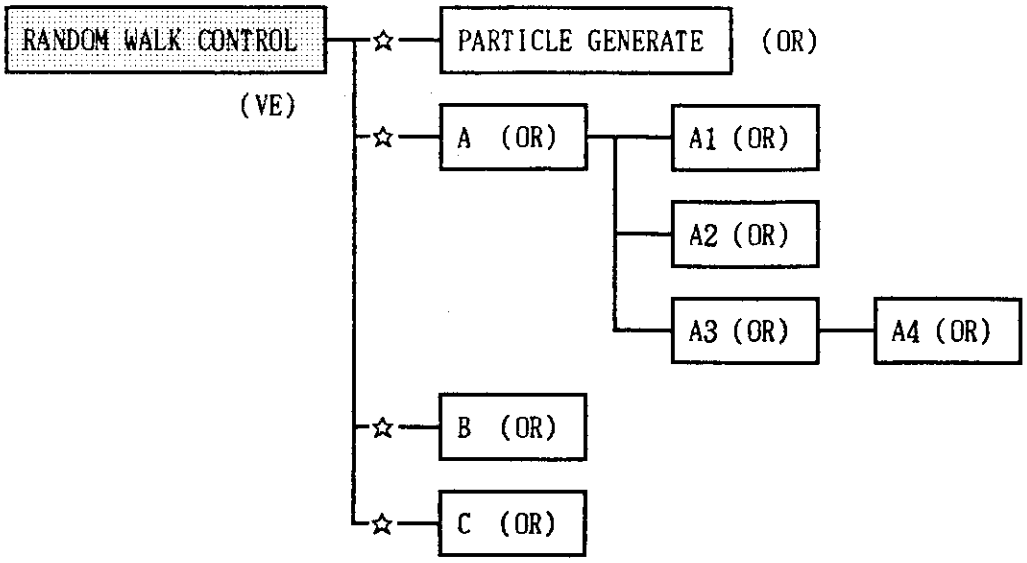
乱数の管理に対してはFig. 6. 4で示したような方法をとる。この方法では個々の粒子に対して１つの乱数列からSeed乱数を個々の粒子に割当て，粒子はこのSeed乱数の初期値としてランダム・ウオークを始める。またEstimator での総和は総和用のエリアに粒子のインデックスを設け各粒子別に総和をとり，全ての粒子のランダム・ウオークの追跡の処理が終了した時点で粒子に対する総和をとることで総和の順序を保証する。さらに臨界問題を解くようなモンテカルロ・コードの場合には次世代のFission source のサンプリングの順序の保証が必要になってくることに注意しなければならない。

③ 複数粒子ランダム・ウオーク時のデバッグ・テスト（ベクトル・モード）

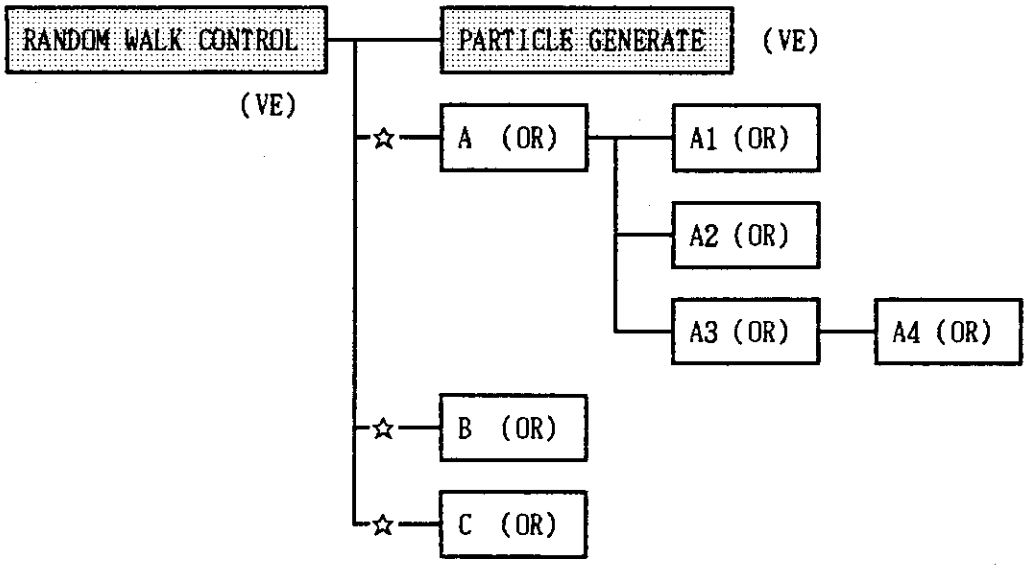
ベクトル・モードでの複数粒子ランダム・ウオークの計算結果とスカラ・モードでの複数粒子ランダム・ウオークの計算結果を比較する。ベクトル・モードでは，スカラ・モードでの場合と演算順序や組み込み関数の精度が違うために計算結果が異なる場合がある。双方の計算結果が異なる場合その原因がベクトル版のバグによるものなのか，あるいは演算順序，組み込み関数の精度の違いに起因するものなのかをチェックする必要がある。

ベクトル版モンテカルロ・コードのデバッグでは上に示したような３つのモードに対する動作確認が必要となるため，各々のモードに対するデバッグ環境を作り，デバッグ対象ルーチンを組み込んでデバッグを進めることが望ましい。

(a) 第一段階



(b) 第二段階



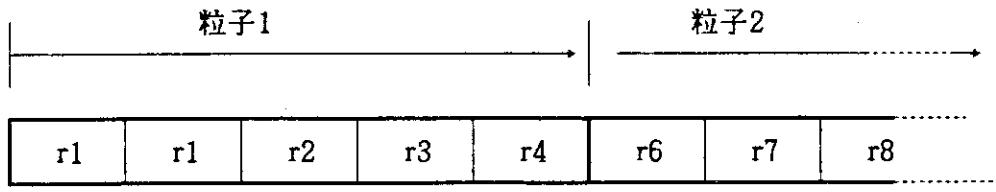
(OR) オリジナル版ルーチン
(VE) ベクトル版ルーチン
☆ インターフェース整合用ルーチン

Fig. 6.1 An example of debugging and testing sequences.

```
      DO 100 JJ=1,NTRK
         II=TRKBK(JJ)
         D¥(II)=MIN(PMF¥(II),DLS¥(II))
         DT¥(II)=D¥(II)/VEL¥(II)
         TME¥(II)=TME¥(II)+DT¥(II)
         XXX¥(II)=XXX¥(II)+UUU¥(II)*D¥(II)
         YYY¥(II)=YYY¥(II)+VVV¥(II)*D¥(II)
         ZZZ     =ZZZ     +WWW¥(II)*D¥(II)
  100 CONTINUE
```

Fig. 6.2 An example of bug caused by the lack of promotion of variables.

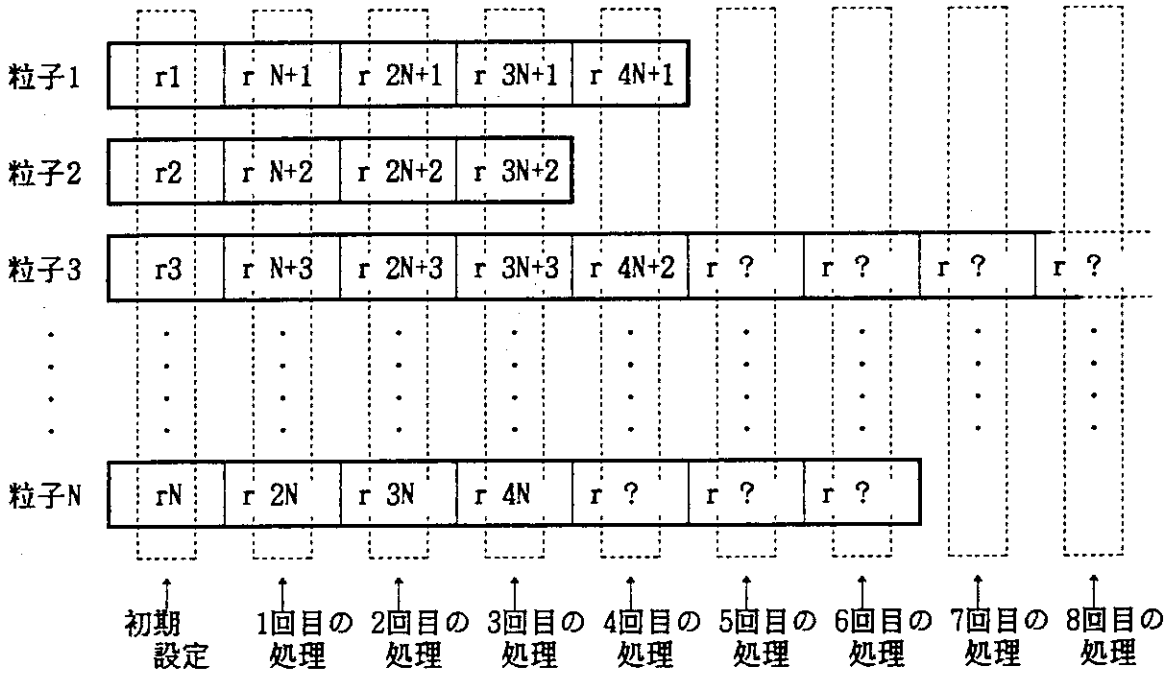
(a) オリジナル版



rn : n 番目の乱数

粒子に対する乱数の割当およびEstimator における粒子のトラック長等の総和は横方向に行われる。

(b) ベクトル版



rn : n 番目の乱数

粒子に対する乱数の割当およびEstimator における粒子のトラック長等の総和は縦方向に行われる。

Fig. 6.3 An assignment method of random numbers to the particles.

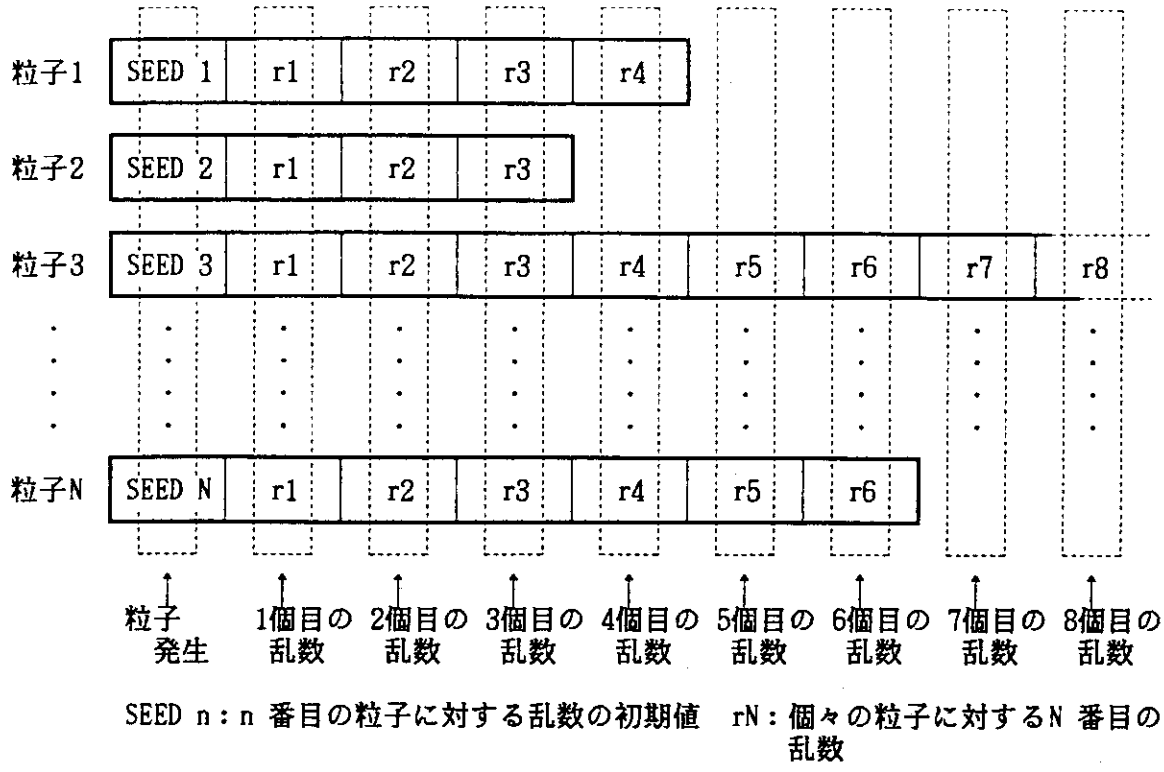


Fig. 6.4 Modification of random number assignment methods.

7. ベクトル版コードのチューニング法

一般にコードのベクトル・モードでの性能を高めるためには，ベクトル化率の向上，DOループの平均ベクトル長の増大，メモリ・アクセス効率の向上，ベクトル化改造によってもたらされたオーバーヘッドの低減を考慮したチューニングをコードにほどこす必要がある。ここではベクトル版モンテカルロ・コードに限定してチューニングの手法を解説する。

7.1 世代の並列処理による平均ベクトル長の増大

第2章で解説したようにベクトル版モンテカルロ・コードの概念はバッチ単位でサンプリングした粒子を同時にランダム・ウオークさせることである。バッチ単位で粒子を処理した場合，体系外への漏れ，媒質を構成する核種との相互作用の結果の吸収による粒子の消滅等の原因によりランダム・ウオークが進行するにつれ並列に処理できる粒子が減少する。このため当然ベクトル長も減少することになり，バッチ単位でのベクトル処理では平均ベクトル長を大きく保つことが難しいということが言える。バッチ単位の処理方式を採用した場合のモンテカルロ・コードMCNP[6]におけるベクトル長の推移のようすをFig. 7. 1 (a)に示す。この例では1度に250個の粒子をサンプリングし，同時にランダム・ウオークさせているが平均ベクトル長はわずか15である。このような状態の場合，短ベクトル長のもとでのベクトル処理が多発することになりパイプライン演算器等のハードウエアを有効利用することができない。

ベクトル長の減少を防ぐためにはバッチ単位の処理をやめ，処理の終了した粒子が発生した場合ただちに新しい粒子をサンプリングして減少したベクトル長を補う方式に変更しなければならない。この方式を世代並列処理方式と呼ぶ。Fig. 7. 1 (b)にMONPコードにおける世代並列処理方式をとった場合のベクトル長の推移を示す。250個の粒子を1度にサンプリングしてランダム・ウオークを始めるという点ではFig. 7. 1 (a)同様であるが，世代並列処理を採用しているためにベクトル長の低下はランダム・ウオークの終りの部分以外ではほとんどなく，平均ベクトル長は141である。ベクトル版モンテカルロ・コードにとって世代並列処理方式は平均ベクトル長を大きく保つための非常に有効な手段といえる。

ベクトル版モンテカルロ・コードへの世代並列処理の適応時には次のような点に注意しなければならない。まず第1にEstimatorでの総和に関する問題がある。バッチ単位でのベクトル処理方式をとった場合の例をFig. 7. 2 (a)に示す。Estimator等での総和は処理中のバッチの最後の粒子の処理の終了を待って中間結果の編集・出力が行われる。世代並列処理方式をとった場合の例をFig. 7. 2 (b)に示す。バッチ内の最後の粒子の終了を待って中間結果の出力の処理に移る点ではFig. 7. 2 (a)と同様であるが，中間結果の出力時には次のバッチの粒子もランダム・ウオークしていることに注意しなければならない。このためバッチ毎の総和を区別するために総和の領域にバッチに対するインデックスを追加し，バッチ間の総和の値を分離しなければならない。

もうひとつの問題は臨界問題を解くような場合に生じる。オリジナル版およびバッチ処理方式

のベクトル版では現在のバッチの処理中に次のバッチのFission sourceの抽出を行っている。世代並列処理の場合には次世代の粒子の処理がフライング・スタートの形式をとるためFission sourceの抽出が不完全の状態で次の世代の粒子をサンプリングしてしまう可能性があげられる。

7.2 ベクトル化率の向上

7.2.1 ベクトル化制御行によるベクトル化の促進

DOループのベクトル化の手法としてリスト・ベクトルを使用した場合，コンパイラはあらかじめリスト・ベクトルの内容を知ることができないため回帰的なデータの定義・参照関係になることを恐れてDOループをベクトル化しない。回帰的なデータの定義・参照関係を生じないようなリストの内容であることが保証できる場合はFACOM FORTRAN77 VPコンパイラの場合にはFig. 4.1(b)で示したようにDOループの直前に*VOCLで始まるベクトル化制御行を挿入することでDOループのベクトル化の促進をはかる。

ただし注意しなければならないのはEstimatorでの粒子トラック長等の総和のDOループは粒子に対して行うのではなく幾何形状の構成要素であるzoneまたはcellに対しての総和であるため回帰的なデータの定義・参照関係となりベクトル演算することができないということである。Fig. 7.3の例で示すようにTALに足し込みを行うような場合セルは複数個の粒子を含んでいることが考えられるため回帰的なデータの定義・参照の関係が生じることになる。データの回帰的な定義・参照を回避してベクトル処理をするためにはTALに粒子にのインデックスを追加することが考えられるが，連続世代処理のために世代に関するインデックスも必要になることを考慮するとEstimatorの総和の領域が莫大な大きさとなるためコードの実用性を加味してEstimatorのスカラ処理を選択すべきである。

7.2.2 DOループからの乱数系の外部手続き参照の取り除き

ベクトル版の制御構造の正当性を確認するデバック・テストでは，乱数系のルーチンはオリジナル版と結果を合わせるためにオリジナル版に使用されていたものと同じものを使用していた。しかし，外部手続きの参照を含むDOループはベクトル化対象とならないためIn-line 展開その他の処置でDOループ中からの外部手続きの引用をとりのぞきDOループのベクトリ化を促進する必要がある。

Fig. 7.4にアセンブラで記述された3次元単位ベクトル生成ルーチンAZIRN，GTISOのFORTRANによるIn-line展開例[4),5)]を示す。AZIRN，GTISOに対応するIn-line 展開されたFORTRANによる処理部分は数学的には同等であるが組み込み関数等の精度のちがいにより数値的には完全に同等なものではないので注意を要する。またFig. 7.4ではIn-line展開不可能の乱数ルーチンをDOループから取り除いた例を示す。DOループの中で一様乱数ルーチンFLTRNFを参照するかわりに，DOループの外側でベクトル版一様乱数ルーチンRANU2でCOMMONエリアRR¥に書き込んだ乱数値を参照している。この場合，乱数ルーチンをとりかえたことによって乱数列が変化しモンテカルロ計算の結果に影響をおよぼすことに注意しなければならない。

7.3 ベクトル化改造によってもたらされたオーバーヘッドの低減

ベクトル化の改造によって制御構造が改善されるためにベクトル版のスカラ・モードでの速度向上が認められるコードもあるがモンテカルロ・コードの場合これを期待出来ない。Table 7.1に示すようにチューニング済みのベクトル版のスカラ・モードでのオリジナル版に対する処理速度比はMORSEを除いて0.6から0.7倍程度でありオリジナル版に対して性能が低下していることがわかる。これは，条件分岐によって形成されているオリジナル版の制御構造をリスト・ベクトルを使ったデータの流れで実現することによってベクトル演算可能な制御構造を構成する際の演算量の増加および，間接アドレスの多用によるデータ・アクセス時のアドレス計算量の増加とデータ・アクセスの非効率化などが大きな原因となっているためと考えられる。オリジナル版の制御構造をベクトル版の制御構造に変更した直後の状態ではベクトル版のスカラ・モードの処理性能比はさらに悪く，ＶＩＭの場合では約0.5であった。以下にベクトル版の処理性能比を高めるためにとった処置を示す。

7.3.1 例外処理の孤立化

ベクトル版の性能を高めるためには平均ベクトル長を大きくしたり，ベクトル化率を向上させたり，メモリ・アクセス効率の向上などのベクトル演算器の効率的利用を目的とするようなチューニングも必要であるが，モンテカルロ・コードのベクトル化の場合には改造によってもたらされたオーバーヘッドを減少させるような，すなわちコンパイラが生成するオブジェクト・コードがより効率的になるようなソースの修正も必要である。

Fig. 7.5に示した例ではDOループ中の例外処理を別なDOループに分離することによって通常処理の実行効率の向上をはかった例[8]である。改造前の例にある文番号9320を持つDOループの後半にある3つのＩＦ文は真になる確率が非常に少ないことがわかっている。例外処理として3つのＩＦ文を分離した改造後の例ではDOループ9320で通常処理と例外処理フラグＸＹＺの設定，DOループ9323で例外処理の必要な粒子の有無の検出，DOループ9322で例外処理をスカラ演算で行っている。このような処置によってDOループ9320から生成されるオブジェクト・コードから例外処理の粒子が存在するかどうかの確認に必要なマスク・ベクトルの生成とそのチェック等の処理に対応する機械命令が取り除かれるため効率をあげることができる。

7.3.2 多次元配列の一次元化

多次元配列のアクセス時のアドレス計算の量を低減させた例[6]がFig. 7.6である。Fig. 7.6(a)のように，第1次元目が粒子番号，第2次元目が粒子の属するセルの番号であるような多次元配列RTCR¥がルーチン内のDOループで頻繁にアクセスされているような場合はDOループごとに多次元配列RTCR¥に対するアドレス計算がDOループごとにされてむだが多い。Fig. 7.6(b)のように1次元化されたRTCR¥に対して①のDOループでインデックス計算を行いアドレス・インデックス・リストを処理のまえに設定しておき，②，③のDOループでこれを使ってRTCR¥にアクセスするように変更するとDOループ内での配列RTCR¥アクセス時のアドレスの計算量をへらすことができる。

7.4 メモリ・アクセス効率の向上

メモリ・コンフリクトを防いでメモリ・アクセス効率の向上をはかることはベクトル・チューニングの基本的項目のひとつである。リスト・ベクトルを使った単精度のコードのベクトル化においては粒子発生ルーチンでのリスト・ベクトルの設定によってランダム・ウオークの初期にメモリ・コンフリクトを生ずる恐れがあるため注意を要する。Fig. 7.7 (a)のようにリスト・ベクトルを初期設定した場合，Fig. 7.7 のサブルーチンPROB8の DO ループの①，②のようなアクセス法ではメモリ・コンフリクトを生じる。FACOM VPシリーズの場合，メモリ・レジスタ間のデータのアクセスは8バイトを単位として行われておりリストを使用しない連続アクセスでは8バイトの後半の4バイトのデータをすぐ使用するためデータの使用効率がよいが，リスト・ベクトルを使った場合はあくまでもランダム・アクセスを前提に考えているためロードされた8バイトのデータの片方しか使用せず，さらにリストの内容が連続的であるため後半の4バイトのデータ・アクセスのために同じデータ・バスに対しアクセスを行うことになりデータ・バス・コンフリクトを生ずることになる。③のような例ではFig. 7.7 の③のような例ではリスト・ベクトルの内容が連続的であってもメモリ上では連続アクセスとならないため最悪条件のメモリ・コンフリクトは生じない。しかし，第1次元目がメモリ・バンクの数（VP－100 の場合は4バンク）の倍数または約数であったような場合はメモリ・バンク・コンフリクトを生じるため注意する必要がある。Fig. 7.7 (a)のような例の場合，多次元配列では粒子のインデックスの切り方によってメモリ・バンク・コンフリクトを回避できるが，1次元配列の場合には回避できない。このため Fig. 7.7 (b)で示すように粒子発生時にリストの内容をランダムに設定することが望ましい。

Table 7.1 CPU time and performance ratio.

	オリジナル版	ベクトル版 スカラ (処理速度比)	ベクトル版 ベクトル (処理速度比)	ベクトル化率
KENO IV	380.0 sec	583.0 sec (0.65)	262.4 sec (1.45)	80%
MORSE	84.0 sec	84.0 sec (1.0)	53.9 cec (1.56)	71%
MCNP	47.9 sec	78.4 sec (0.61)	37.7 sec (1.27)	76%
VIM	103.1 sec	144.8 sec (0.71)	73.8 sec (1.4)	67%

(注)
・『ベクトル版ベクトル』はベクトル版のベクトル・モードにおける処理時間,
『ベクトル版スカラ』はベクトル版のスカラ・モードにおける処理時間を示す。
・処理速度比はオリジナル版の処理時間／ベクトル版の処理時間で定義する。
・処理時間の単位は『秒』である。

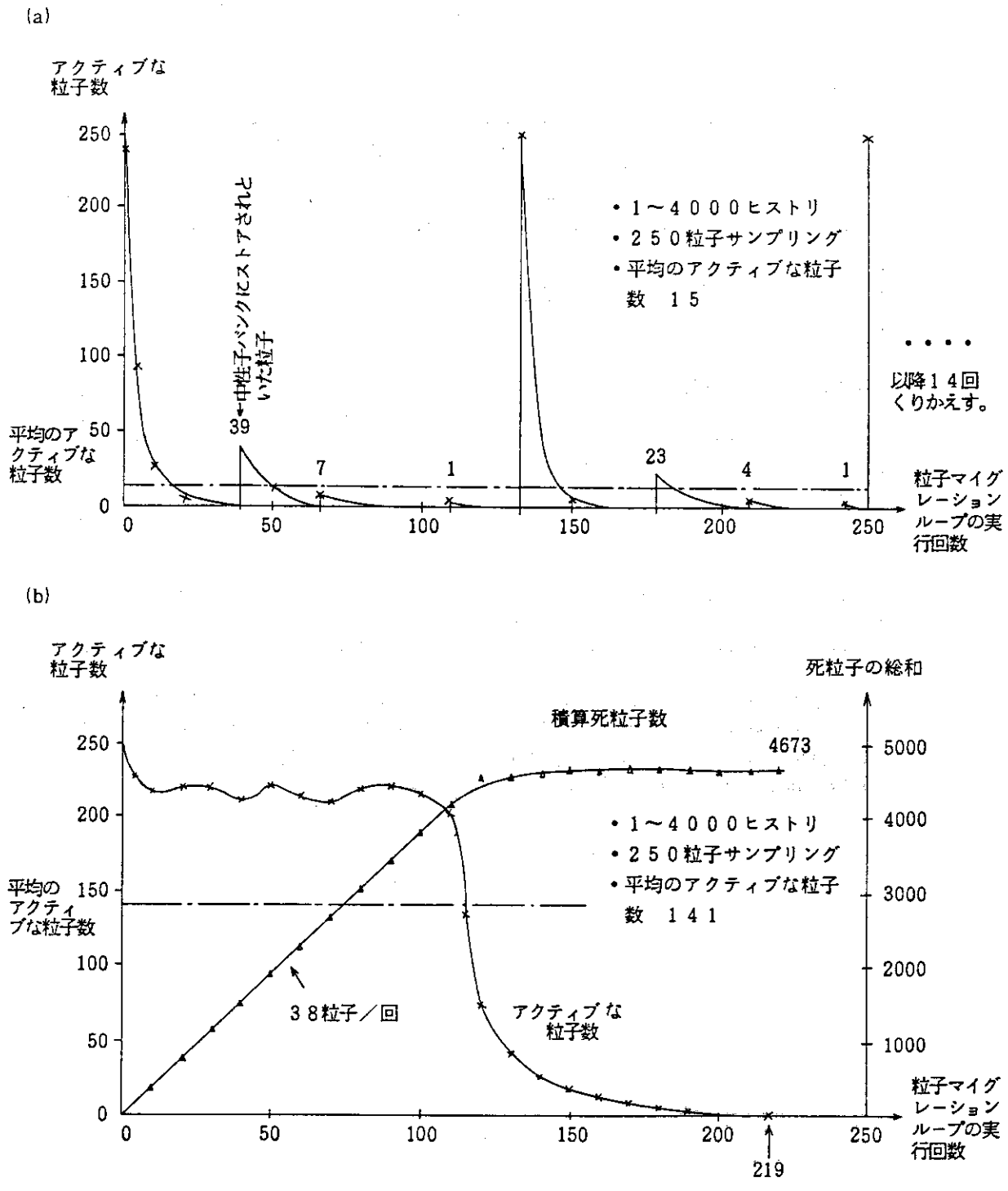


Fig. 7.1 Reduction of number of active particles during the random walk by the batch sampling method (a) and successive sampling method (b).

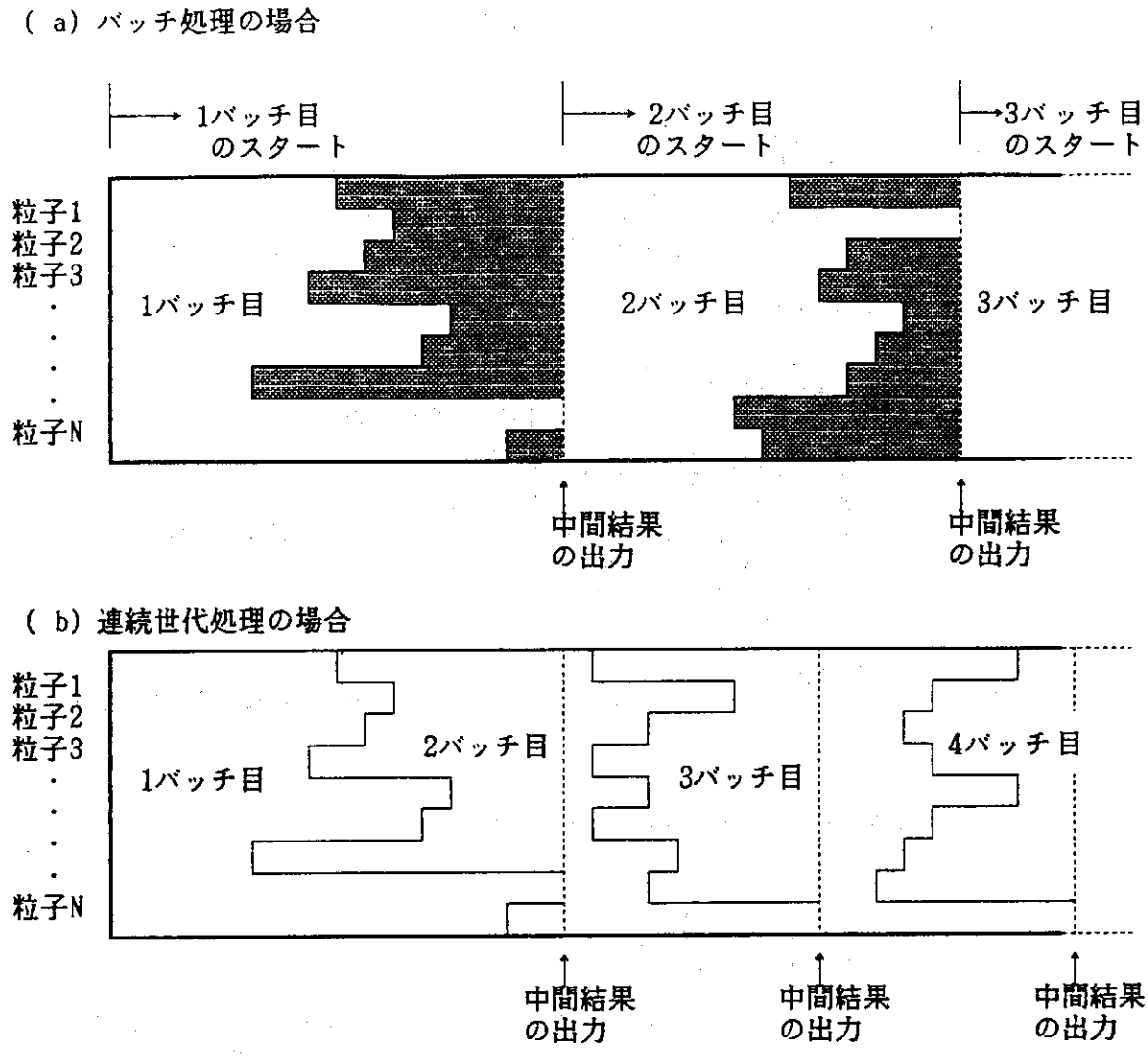


Fig. 7.2 The difference of start timing of next generation particles between the batch sampling and successive sampling methods.

```
      DO  I =1,  LENG
          J =LIST (I )
            :
          TAL  (ICL  (J ) ) =TAL  (ICL  (J ) ) + TRK  (J )
            :
            :
            :
      ENDDO
```

TAL (N) : セルN に対する総和の領域
ICL (J) : 粒子J の存在するセル番号
TRK (J) : 粒子J の属するセル内での飛程の長さ

Fig. 7.3 An example of in-line expansion of external procedure in a DO loop.

(a) DOループ中の外部手続きのIn-line 展開

```
      DO  100 K=1, NPART
      K ¥=LPART(K)
C .... CALL AZIRN(BSTAR¥(K¥), GSTAR¥(K¥))
      BSTAR ¥(K¥)=SIN(PI2 ¥*RR ¥(k+NRR¥))
      GSTAR ¥(K¥)=COS(PI2 ¥*RR ¥(K+NRR¥))      } AZIRN  の展開
        :
100   CONTINUE

      DO  200 K=1, NPART
      K ¥=LPART(K)
C .... CALL GTISO(GAMMA¥(K¥), ALPHA¥(K¥), BETAP¥(K¥))
      WKS1=SIN(PI2¥*RR ¥(NRR¥+K*2-1))
      WKC1=COS(PI2¥*RR ¥(NRR¥+K*2  ))
      WKS2=2.0D0*RR ¥(NRR¥+2*K)-1.0D0
      WKC2=SQRT(1.0D0-WKS2*WKS2)                } GTISO の展開
      GAMMA ¥(K¥)=WKC2*WKC1
      ALPHA ¥(K¥)=WKC2*WKS1
      BETAP ¥(K¥)=WKS2
        :
200   CONTINUE
```

★RR¥上には［0,1 ］一様乱数が書き込まれている。

(b) 乱数ルーチンのDOループ中からの取り除き

```
      DO 4120 KI¥=1, N350¥
      K ¥=L350 ¥(KI ¥)
      RANF=FLTRNF(0)
        :
      STMFP ¥(K¥)=-ALOG(RANF)          ★  FLTRNF: オリジナル版での
        :                                       一様乱数ルーチン
4120 CONTINUE

           ↓

      CALL  RANU2(IX ¥, RR¥(IREM+1), NRR ¥, ICON)
        :
      DO 4120 KI ¥=1, N350¥
        :
      STMFP¥(K¥)=-ALOG(RR¥(NRR¥+KI ¥))
        :
4120 CONTINUE                          ★ベクトル版一様乱数ルーチンRANU2
                                       を使ってCOMMON領域RR¥に乱数を書
                                       き込む。
```

Fig. 7.4 An example of removing a random number generator from a DO loop.

(a) 改造前

```
*VOCL LOOP,NOVREC
      DO 9320 IV=1,NPATH
      JV=PATHBA(IV)
      KR(JV)=MAT(K(JV))
*VOCL STMT,IF(99)
      IF(KR(JV).NE.0) THEN
      PTH(JV)=RPTH(JV)*RSIGT(KR(JV),IG(JV))
      ELSE
      PTH(JV)=BIG
      ENDIF
      X1(JV)=X(JV)+PTH(JV)*U(JV)
      Y1(JV)=Y(JV)+PTH(JV)*V(JV)
      Z1(JV)=Z(JV)+PTH(JV)*W(JV)
*VOCL STMT,IF( 0)
      IF(X(JV).EQ.X1(JV)) X1(JV)=X(JV)+SIGN(X(JV),U(JV))*(1.0E-6)
*VOCL STMT,IF( 0)
      IF(Y(JV).EQ.Y1(JV)) Y1(JV)=Y(JV)+SIGN(Y(JV),V(JV))*(1.0E-6)
*VOCL STMT,IF( 0)
      IF(Z(JV).EQ.Z1(JV)) Z1(JV)=Z(JV)+SIGN(Z(JV),W(JV))*(1.0E-6)
 9320 CONTINUE
```

(b) 改造後

```
*VOCL LOOP,NOVREC
      DO 9320 IV=1,NPATH
      JV=PATHBA(IV)
      KR(JV)=MAT(K(JV))
*VOCL STMT,IF(99)
      IF(KR(JV).NE.0) THEN
      PTH(JV)=RPTH(JV)*RSIGT(KR(JV),IG(JV))
      ELSE
      PTH(JV)=BIG
      ENDIF
      X1(JV)=X(JV)+PTH(JV)*U(JV)
      Y1(JV)=Y(JV)+PTH(JV)*V(JV)
      Z1(JV)=Z(JV)+PTH(JV)*W(JV)
      XYZ(IV)=(X(JV)-X1(JV))*(Y(JV)-Y1(JV))*(Z(JV)-Z1(JV))
 9320 CONTINUE
      IXYZ=0
*VOCL LOOP,NOVREC
      DO 9323 IV=1,NPATH
      IF(XYZ(IV).EQ.0) IXYZ=IXYZ+1
 9323 CONTINUE
      IF(IXYZ.EQ.0) GO TO 9324
*VOCL LOOP,SCALAR
      DO 9322 IV=1,NPATH
      JV=PATHBA(IV)
*VOCL STMT,IF( 0)
      IF(X(JV).EQ.X1(JV)) X1(JV)=X(JV)+SIGN(X(JV),U(JV))*(1.0E-6)
*VOCL STMT,IF( 0)
      IF(Y(JV).EQ.Y1(JV)) Y1(JV)=Y(JV)+SIGN(Y(JV),V(JV))*(1.0E-6)
*VOCL STMT,IF( 0)
      IF(Z(JV).EQ.Z1(JV)) Z1(JV)=Z(JV)+SIGN(Z(JV),W(JV))*(1.0E-6)
 9322 CONTINUE
 9324 CONTINUE
```

Fig. 7.5 An isolation of exceptional processing part in a DO loop.

(a) 三次元配列のアクセス

```
      SUBROUTINE   PROB6 (LIST,LNG)
      PARAMETER  (NP=250,MX=50)
      INTEGER *4
     +        LIST (*)
      REAL*8 RTCR¥ (NP,MX,10)
      COMMON  /SAMPLE/
     +        RTCR¥ (NP,MX,10) ,
     +        IEX  (NP)
              :
              :
              :
              :
              :
              :
              :
              :
      DO I=1,LNG
         J=LIST (I )
         R=RTCR¥ (J, IEX (J ) ,8)
              :
      ENDDO
              :
              :
              :
              :
      DO I=1,LNG
         J=LIST   (I )
         RTCR ¥ (J, IEX (J ) ,8)
     +      =・・・・・
              :
      ENDDO
              :
              :
              :
      RETURN
      END
```

・ループごとにアドレスを再計算する。

(b) 一次元化配列のアクセス

```
      SUBROUTINE   PROB6 (LIST,LNG)
      PARAMETER  (NP=250,MX=50)
      INTEGER *4
     +        LIST (*) ,L (NP)
      REAL*8 RTCR¥ (NP,MX,10)
      COMMON  /SAMPLE/
     +        RTCR¥ (NP,MX,10) ,
     +        IEX  (NP)
              :
              :
      DO I=1,LNG
         J=LIST (I )
①     L (J ) =10*MX* (J-1 ) +
     +        10* ( IEX (J ) -1) + 8
      ENDDO
              :
      DO I=1,LNG
         J=LIST (I )
②        K=L  (I )
         R=RTCR¥ (k)
              :
      ENDDO
              :
              :
      DO I=1,LNG
         J=LIST   (I )
③        K=L  (I )
         RTCR ¥ (k)
     +      =・・・・・
              :
      ENDDO
              :
              :
              :
      RETURN
      END
```

・①のループでまとめて計算

Fig. 7.6 Decrease of amount of adress computation for multi dimensional array accessing.

```
      SUBROUTINE  PROB8 (L1,LNG)
      PARAMETER  (NP=1000)
      INTEGER *4    L1  (LNG )
      REAL*4        X (NP) ,  D (NP,3) ,  T (3,NP)
        :
        :
      DO I=1,LNG
        J=L1 (I )
        X (J ) = . . .    ---------- ①
        D (J,1 ) = . . .  ---------- ②
        T (3,J ) = . . .  ---------- ③
      ENDDO
        :
        :
        :
      RETURN
      END
```

(a) リスト・ベクトルL1の内容が連続的である時

	1	2	3	4	5	6	LNG-1	LNG
L1	1	2	3	4	5	6	LNG-1	LNG

(b) リスト・ベクトルL1の内容がランダムである時

	1	2	3	4	5	6	LNG-1	LNG
L1	735	621	11	349	2	111	33	852

Fig. 7.7 Some examples of accessing pattern of array.

8. 問　題　点

8.1　低い実効性能と原因

ベクトル化の第1目的は処理速度の向上であるが中性子輸送計算のための大型汎用のモンテカルロ・コードの場合のオリジナル版に対するベクトル版の処理速度の向上比は1.3から1.5倍程度にとどまっている。これはベクトル化改造時にもたらされるオーバーヘッドによるベクトル版のスカラー性能の低下と，ベクトル化率が60％から80％程度と低いことの2つの原因が考えられる。ベクトル版のスカラー性能の低下は配列化される変数および次元の上がる配列が極端に多く，間接アドレスの頻度が非常に多くなるためにメモリ・アクセスのためのアドレス計算の量が増えること，オリジナル版の条件分岐をリスト・ベクトルを使用したデータの流れに変更する際にもたらされる演算量の増加などによるものである。またベクトル化率が低い点は条件分岐によるベクトル長の低下のためにベクトル化指示行でスカラ演算を指定しているループが多い。Estimatorの処理などのように回帰的なデータの定義・参照のためベクトル演算できない部分が多い，等の原因によってスカラー演算の割合を減らすことが難しいためである。このような理由から，パイプライン方式のベクトル・プロセッサを利用した既存の汎用大型モンテカルロ・コードの高速化は困難なものとなっている。しかし取り扱う幾何形状が限定されている一部の専用モンテカルロ・コードのベクトル化では10倍以上の高速化の例[10), 11)]もあり，パイプライン方式のベクトル・プロセッサを使用したモンテカルロ・コードの高速化も小規模で単純構造のモンテカルロ・コードに対しては有効かと思われる。

8.2　オリジナル版とベクトル版での計算結果の相違

乱数の割当，Estimatorにおける総和および次のバッチのためのFission sourceの保存などの順序がオリジナル版が横方向なのに対し，ベクトル版は縦方向となってしまうため結果がオリジナル版と異なる。第6章2節で述べたように配列にインデックスを増設してこれらの順序をオリジナル版と同じように保証することも可能であるが，メモリの使用率の増大によってJOBのターン・アラウンドを下げることになってしまうため実用に供する時には検討を要する。

また，連続世代処理方式を臨界計算に使用した時は次のバッチのFission sourceの分布が不完全のまま粒子をサンプリングして次のバッチをスタートさせてしまうためFission sourceの分布の収束にどの程度影響をおよぼすのかを検討する必要もある。

8.3　調査時における問題

モンテカルロ・コードのベクトル化作業のなかで重要な項目は2つあり，1つはオリジナル版のCOMMON変数のなかから粒子のインデックスを追加しなければならない変数をさがす―ベク

トル変数の抽出—作業である。ベクトル変数抽出の基準が『粒子と1対1対応のある変数』というような漠然としたものであるため抽出作業は大規模コードの場合には非常な困難をきわめるのが常であるが，解決策はないのが実状である。

もうひとつはサブルーチンの制御構造の解析に関するものである。個々のルーチンのベクトル化を行うためにはオリジナル版の条件分岐の制御構造を解析しそれをリスト・ベクトルを使ったデータの流れに置き換えることが必要である。中性子輸送計算用の大型モンテカルロ・コードではひとつのルーチンの規模が大きく，かつ条件分岐が多いため，条件分岐によって形成されている制御構造をリスト・ベクトルによるデータ・フローに置き換えるための制御構造の解析に非常に多くの時間を要する。DOループ内部の制御構造，データ・フロー，ループの分割等の解析を行うソフトウエア・ツールを開発し，制御構造の解析情報，ループ分割情報，ループ分割時のローカル変数の次元の昇格の情報を正確に把握することにより複雑な制御構造をもったコードのベクトル化に要する工数を消減することができるものと思われる。

9. おわりに

本報告書ではモンテカルロ・コードのベクトル化の際に必要なコードの調査および，デバックの手法を中心として述べた。モジュール相互間の結合が強く，またモジュール内の条件分岐が多いようなコードのベクトル化においては調査の部分に非常に多くの時間を費やさなければならないのが現状である。しかし，コードの調査項目のなかには，DOループ分割のためのモジュール内の制御構造の解析，DOループの分割時のローカルな変数および配列の次元の昇格の情報の提供などツール化できる部分もあるためツールによって調査の効率と信頼性をあげることが可能かと思われる。

現状のMISD (Multi Instruction Single Data stream) 方式のベクトル処理システムを使った既存の中性子輸送計算用モンテカルロ・コードの高速化の試みでは良い結果を得ることができなかった。これらの経験から，現状のシステムの機能強化のための分類処理用のパイプライン演算器等のハードウエアの提案もなされている[4]。また，近年MIMD(Multi Instruction Multi Data stream)方式のマルチ・プロセッサの開発がさかんに行われており，MIMD 方式とのマッチングの良さから中性子輸送計算用のモンテカルロ・コードの高速計算に対する道も開かれつつあるものと思われる[4), 12), 13)]。

トル変数の抽出—作業である。ベクトル変数抽出の基準が『粒子と1対1対応のある変数』というような漠然としたものであるため抽出作業は大規模コードの場合には非常な困難をきわめるのが常であるが，解決策はないのが実状である。

もうひとつはサブルーチンの制御構造の解析に関するものである。個々のルーチンのベクトル化を行うためにはオリジナル版の条件分岐の制御構造を解析しそれをリスト・ベクトルを使ったデータの流れに置き換えることが必要である。中性子輸送計算用の大型モンテカルロ・コードではひとつのルーチンの規模が大きく，かつ条件分岐が多いため，条件分岐によって形成されている制御構造をリスト・ベクトルによるデータ・フローに置き換えるための制御構造の解析に非常に多くの時間を要する。DOループ内部の制御構造，データ・フロー，ループの分割等の解析を行うソフトウエア・ツールを開発し，制御構造の解析情報，ループ分割情報，ループ分割時のローカル変数の次元の昇格の情報を正確に把握することにより複雑な制御構造をもったコードのベクトル化に要する工数を消減することができるものと思われる。

9. おわりに

本報告書ではモンテカルロ・コードのベクトル化の際に必要なコードの調査および，デバックの手法を中心として述べた。モジュール相互間の結合が強く，またモジュール内の条件分岐が多いようなコードのベクトル化においては調査の部分に非常に多くの時間を費やさなければならないのが現状である。しかし，コードの調査項目のなかには，DOループ分割のためのモジュール内の制御構造の解析，DOループの分割時のローカルな変数および配列の次元の昇格の情報の提供などツール化できる部分もあるためツールによって調査の効率と信頼性をあげることが可能かと思われる。

現状のMISD (Multi Instruction Single Data stream) 方式のベクトル処理システムを使った既存の中性子輸送計算用モンテカルロ・コードの高速化の試みでは良い結果を得ることができなかった。これらの経験から，現状のシステムの機能強化のための分類処理用のパイプライン演算器等のハードウエアの提案もなされている[4]。また，近年MIMD(Multi Instruction Multi Data stream)方式のマルチ・プロセッサの開発がさかんに行われており，MIMD 方式とのマッチングの良さから中性子輸送計算用のモンテカルロ・コードの高速計算に対する道も開かれつつあるものと思われる[4), 12), 13)]。

謝　　辞

中性子輸送モンテカルロ・コードのベクトル化にあたり，コードを提供およびコードに関する情報を提供してくださったプラント安全解析研究室片倉純一氏，原子炉システム研究室森貴正氏に感謝致します。

富士通株式会社科学システム部第2システム課長南多善氏，同第3システム課牧野光弘氏および富士通エフ・アイ・ピー株式会社技術開発部第4開発課今西史龍氏には記述法の御指導を戴きました。感謝致します。

参　考　文　献

1. William R. Martine, Forrest B. Brown,
"STATUS OF VECTORIZED MONTE CARLO FOR PARTICLE TRANSPORT ANALYSIS", The International Journal of Supercomputer Application, Volume 1, Number 2, pp. 11-32, (1987).
2. 奈良岡賢逸他,
"原子力コードのベクトル化プログラミング〔1〕", 私信
3. 徳永康男他,
"原子力コードにおける数値解析法とそのベクトル化", JAERI-M85-143, (1985).
4. Kiyoshi Asai et. al.,
"VECTORIZATION OF KENO IV CODE AND ESTIMATE OF VECTOR-PARALLE PROCESSING", JAERI-M86-151, (1986).
5. 菅沼正之他,
"連続エネルギー・モンテカルロ・コードVIMのベクトル化", JAERI-M86-190, (1986).
6. 栗田　豊他,
"モンテカルロ・コードMCNPのベクトル化", JAERI-M87-022, (1987).
7. 樋口健二他,
"モンテカルロ・コードMORSE-DDのベクトル化", JAERI-M87-032, (1987).
8. Kiyoshi Asai et. al.,
"Vectorization of the KENO-IV Code", NUCLEAR SCIENCE AND ENGINEERING 293-307 (1986).
9. 富士通株式会社,
"FACOM OS IV/F4 MSP FORTUNE 使用手引 V10用", 78SP-5360.
10. Y. Tokunaga et. al.,
"VECTORIZATION OF THE MONTE CARLO PROGRAM DICON", Computer Physice Communications, 38 (1985) 15-26, NORTH-HOLLAND.
11. A. Nishiguchi et. al.,
"Vector Calculation of Particle Code", Journal of Computational Physice, Vol. 61, №3, December 1985, Academic press, Inc.
12. 小原和博,
階層構造のMIMD型スーパー・コンピュータ,
—イリノイ大学のCedarプロジェクトを例として—, 情報処理, Vol. 25, May 1984.
13. 浅井　清他,
"ベクトル・パラレル計算処理の原子力コードへの適応性", JAERI-M87-136, (1987).