JAERI-M
90-135

ベクトル化 KENO Ⅳ コードの性能評価

１９９０年８月

折居　茂夫*・樋口　健二・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　日立高速印刷株式会社

JAERI－M 90－135

ベクトル化KENO Ⅳコードの性能評価

日本原子力研究所東海研究所情報システムセンター
折居 茂夫*・樋口 健二・浅井 清

（1990年7月25日受理）

中性子輸送モンテカルロコードKENO Ⅳをベクトル化し，そのコードのベクトルとスカラ性能を，富士通のベクトル計算機ＶＰ－100において評価した。

ベクトル化KENO Ⅳコードには，ベクトル化アルゴリズムとしてスタックベースアルゴリズムを採用した。プログラミングテクニックとして，間接データアクセス法をこのアルゴリズムに採用した。

粒子数1800個を用いた臨界計算に対して，オリジナルKENO Ⅳコードに対するベクトル化コードの速度向上率は，ＶＰ－100において1.9倍であった。

この低い性能向上には4つの原因がある。第一に，オリジナルKENO Ⅳコードは，スカラーキャッシュメモリーの有効利用が高速データアクセスを可能にしているため，高性能である。第二に，スタックベース法を採用したためプログラムの命令が増すこと。第三に，間接データアクセス法を使用したため，ＶＰ－100のロード／ストアパイプラインがビジーとなり，従って四則演算パイプがしばしばデータを待つこと。最後に，この間接データアクセスが他のベクトル処理に比べて遅いことである。

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，富士通株式会社

JAERI-M 90-135

Performance Evaluation of a Vectorized KENO IV Code

Shigeo ORII*, Kenji HIGUCHI and Kiyoshi ASAI

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

(Received July 25, 1990)

A neutron transport Monte Carlo code KENO IV has been vectorized and its performance of vector and scalar was evaluated on a vector processor FACOM VP-100.

A stack driven algorithm and the indirect addressing for variables have been used as the vectorization algorithm.

On the VP-100, the performance improvement of the vectorized KENO IV was 1.9 times compared to the original scalar code. As the sample input, a tank type critical assembly problem was employed with 1800 particles in a batch.

There are four reasons for the low performance ratio. First, the original KENO IV code achieves high performance because the efficient use of a scalar cache memory makes it possible to access data quickly in the original scalar code. Second, instructions of the vectorized code have increased due to the vectrization. Third, by the use of indirect addressing for variables, the load/store pipelines of the VP-100 have been busy and the arithmetic pipelines have been forced to be idle so often. Last, the indirect data access is slower than other vector operations on the processor.

Keywords: Monte Carlo Method, Nuclear Code, Vector Processor, Vectorization, KENO IV, Performance

---

* On leave from FUJITSU, Ltd.

目　　次

1. はじめに …… 1
2. コード概要 …… 2
2.1 基礎方程式 …… 2
2.2 プログラム構成 …… 3
3. ベクトル化 …… 9
3.1 入力データ …… 9
3.2 コスト分布 …… 10
3.3 スタックドリブン法 …… 10
3.4 各処理のベクトル化 …… 11
3.5 ベクトル化率評価 …… 12
4. 性能評価 …… 21
4.1 ベクトル化コードのスカラ性能 …… 21
4.2 ベクトル化コードのベクトル性能 …… 21
5. 性能評価における問題点 …… 27
6. まとめと議論 …… 29
謝辞 …… 30
参考文献 …… 31
付録A 各カーネルのベクトル化方法 …… 33
付録B エネルギ群遷移処理のベクトル化 …… 43
付録C 時間測定法 …… 46
付録D 多重ＩＦ文の性能 …… 47

Contents

1. Intoroduction 1
2. Code outline 2
2.1 Basic equation 2
2.2 Scheme of program 3
3. Vectorization 9
3.1 Input data 9
3.2 Cost distribution 10
3.3 Stack-driven method 10
3.4 Vectorization of each kernel 11
3.5 Evaluation of vectrization ratio 12
4. Performance evaluation 21
4.1 Scalar performance of vectorized code 21
4.2 Vector performance of vectorized code 21
5. Problem of performance evaluation 27
6. Concluding remark 29
Acknowledgements 30
References 31
Appendix A Vectorization technique for each kernel 33
Appendix B Vectorization for determining out-going energy group 43
Appendix C Technique of CPU time measurement 46
Appendix D Vector performance of nested "IF" statement 47

1. はじめに

モンテカルロ法は，原子炉の中性子輸送方程式を計算する方法の一つとして用いられる。特に原子炉内の複雑な幾何形状を扱う場合，有限差分法のように差分メッシュを用いるもの，例えばSn法に比べ簡単に計算できる特長を持つ。日本原子力研究所（以後，原研と呼ぶ）においても，古くから，KENO[1]，MORSE[2]，VIM[3]，MCNP[4]等が用いられてきた。原研では，これらのコードによる計算量が多く，1985年ベクトル計算機VP－100の導入以来ベクトル計算による計算時間の短縮が図られており，適用方法の研究が継続的に行われている[5]。

これらの研究で用いられた基礎となるベクトル計算機向けアルゴリズム（イベントベース法）は，1973年にパラレル計算機の試作機Illiac－Ⅳ に適用され，1981年には商用ベクトル計算機CYBER 205に適用された[6]。日本では，1984年に原研にVP－100が導入されると同時に適用研究が開始され[7]，現在ではプロダクションコードの開発が行われている[8]。

しかしながら，ベクトル計算機への適用評価はそれぞれ異なっている。Illiac－Ⅳ プロジェクトに参加したMalvin H. Kaiosは，ベクトル計算機に見切りをつけ，パラレル計算機上への適用を研究している[9]。Forrest B. Brownは，CYBER 205を使ったベクトル計算により顕著な性能向上が得られたと報告し，プロダクションコードの作成，ベクトルパラレル計算機への適用を研究している[6]。浅井は，VP－100を使ったベクトル計算にはモンテカルロ専用のハードウェアが必要だとして，研究を行っている[10]。

本報告では，モンテカルロ法によるベクトル性能を詳細に分析することにより，何故このようなばらばらな結果が出てきたかを考察することが，メインテーマである。これまでに行われてきた報告との主な違いは，個々のDOループのベクトル化率，ベクトル性能を実際のVU時間（ベクトルユニットを使用している時間）の測定によって調べたこと，特定DOループのベクトル命令の実行時間を詳細に考察したこと，ベクトル／スカラの相対性能による評価方法について考察した点である。本文の2章では，中性子輸送モンテカルロコード KENO Ⅳの「コード概要」を述べる。3章では，オリジナルコードをベクトル計算機FUJITSU VP－100向けに再構成する「ベクトル化」について述べる。4章では，ベクトル化コードのVP－100における「性能評価」を行う。5章では，CYBER 205におけるベクトル性能を推測し，「性能評価における問題点」を示す。6章では，「まとめと議論」を行う。

尚モンテカルロ法は，多様な分野に使われているが，本論文では中性子輸送モンテカルロコードに話を限定する。

2. コード概要

中性子輸送モンテカルロコードのベクトル化では，そのコードがテストコードかプロダクションコードにより異なる。それは，プロダクションコードはテストコードと比較して色々な計算処理を行うため，そのために必要な計算のベクトル化を考慮する必要があるからである。そこでこの章では，KENO Ⅳで行われている主な計算処理について言及する。

既に引用したKENO Ⅳは，原力炉の多群モンテカルロ臨界計算プログラムで，実効増倍係数，中性子の寿命，リーク，吸収，フラックス及び核分裂密度を計算する。原子炉の形状入力方法は，球，直方体，円柱などの3次元形状とその大きさを指定し，これらの形状を組み合わせボックスという単位にまとめる。更にボックスを重ねて3次元の幾何形状を作る。

2.1 基礎方程式

KENO Ⅳのマニュアル[1]を参考に，計算される輸送方程式を簡単に説明する。

KENO Ⅳで計算されるフラックスの方程式は，次のBoltzmanの方程式から導かれる。左辺第一項は中性子のフラックスの時間変化，第二項は移流，第三項は吸収に対応する項である。また右辺はソース項である。

$$\frac{1}{v}\frac{\partial \Omega}{\partial t}(X,E,\Omega,t)+\Omega\cdot\nabla\Phi(X,E,\Omega,t)+\Sigma_t(X,E,\Omega,t)\Phi(X,E,\Omega,t)$$
$$=q(X,E,\Omega,t) \qquad (2.1)$$

ここに，　Φ：中性子束，　Σ_t：巨視的全断面積，　q：ソース項，　v：中性子の平均速度，

X：位置，　E：エネルギー，　Ω方向，　t：時刻である。

次に媒体が等方であると仮定し，多群近似をすると次式となる。

$$\frac{1}{v_g}\frac{\partial \Phi_g}{\partial t}(X,\Omega,t)+\Omega\cdot\nabla\Phi_g(X,\Omega,t)+\Sigma_{tg}(X)\Phi_g(X,\Omega,t)$$
$$=q_g(X,\Omega,t) \qquad (2.2)$$

ここに，　g：エネルギー群である。

ここでソース項qは次のように書ける。第二項は核分裂による生成項，第一項は散乱項である。

$$q_g(X,\Omega)=\sum_{g'}\int d\Omega'\Phi_{g'}(X,\Omega')\Sigma_s(X,g'\rightarrow g,\Omega\cdot\Omega')$$
$$+\frac{1}{4\pi}\sum_{g'}\int_{\Omega} d\Omega'\Phi_{g'}(X,\Omega')\chi_g\nu_{g'}\Sigma_{tg'}(X) \qquad (2.3)$$

ここに，　$\nu_{g'}$：核分裂生成中性子数，χ：核分裂生成スペクトル，Σ_s：g′・Ω′からg,Ωへの散乱断面積である。

（2.2）式において，中性子束が時間に依存していないとすると次式を得る。

$$\Omega \cdot \nabla \Phi_g(X,\Omega) + \Sigma_{tg}(X)\Phi_g(X,\Omega) = q_g(X,\Omega) \qquad (2.4)$$

次ぎに（2.4）式を積分形にするために，積分因子 $e^{-\int_0^R \Sigma_{tg}(X-R'\Omega)dR'}$ を導入し，$\Phi_g(X',\Omega)$ にこれを掛けてRで微分し，次の関係を得る。

$$-\frac{d}{dR}\left[\Phi_g(X',\Omega)\, e^{\int_0^R \Sigma_{tg}(X-R'\Omega)dR'}\right]$$

$$\triangleq -\frac{d\Phi_g(X',\Omega)}{dR} e^{-\int_0^R \Sigma_{tg}(X-R'\Omega)dR'} + \Sigma_{tg}(X')\Phi_g(X',\Omega) e^{-\int_0^R \Sigma_{tg}(X-R'\Omega)dR'} \qquad (2.5)$$

ここで，$X' = X - R\Omega$ とすれば，

$$\frac{d\Phi_g(X',\Omega)}{dR} = \frac{\partial \Phi_g}{\partial x'}\frac{dx'}{dR} + \frac{\partial \Phi_g}{\partial y'}\frac{dy'}{dR} + \frac{\partial \Phi_g}{\partial z'}\frac{dz'}{dR}$$

$$= -\Omega \cdot \nabla \Phi_g(X',\Omega)$$

これを（2.5）式の右辺の第一項に代入して次式を得る。

$$(2.5) \quad = \left[\Omega \nabla \Phi_g(X',\Omega) + \Sigma_{tg}(X')\Phi_g(X',\Omega)\right] e^{-\int_0^R \Sigma_{tg}(X-R'\Omega)dR'}$$

更に，右辺に（2.4）式を代入して次式を得る。

$$-\frac{d}{dR}\left[\Phi_g(X',\Omega) e^{-\int_0^R \Sigma_{tg}(X-R'\Omega)dR'}\right] = q_g(X',\Omega) e^{-\int_0^R \Sigma_{tg}(X-R'\Omega)dR'} \qquad (2.6)$$

（2.6）式をRについて0から∞まで積分し，R→∞の項を0とすると，次の中性子束の方程式を得る。

$$\Phi_g(X,\Omega) = \int_0^\infty q_g(X-R\Omega,\Omega) e^{-\int_0^R \Sigma_{tg}(X-R'\Omega)dR'} dR \qquad (2.7)$$

KENO Ⅳでは，（2.7）式をモンテカルロ法によって計算する。

2.2　プログラム構成

ここでは，中性子束の方程式（2.7）を計算する主要サブルーチンBEGINのプログラム構成を示す。ベクトル化のためには処理単位でプログラム構成を知ることが重要である。これにより次の2つのことが容易となる。

① 並列性の発見
② 処理単位でのベクトル化（処理単位でのアルゴリズムの変更も可能）

サブルーチンＢＥＧＩＮのフローチャートをFig.2.1に，処理の表をTable 2.1に示す。計算は，Fig.2.1に示したように世代ループの内側に粒子ループが入っており，粒子が死ぬまで一つの粒子に対して行われる（ヒストリ・ベース法，後述）。ＦＳＴＡＲＴにおいて初期値が設定され，ＰＡＴＨにおいて粒子の飛程と新しい位置が計算される。Inward Crossing において，領域の内側に向かっている粒子が境界を通過したか否かを調べ，飛程と位置の調節が行われる。ＡＲＲＡＹにおいて境界を横切ったか否かがボックス単位にチェックされ，横切った場合，粒子の位置は新しい座標系に変換される。ＡＬＢＥＤＯにおいて，アルベド反射体からの反射を計算する。反射体からの粒子がコア中のどのボックスに入ったかはＦＩＮＢＯＸによって調べられ，ボックス固有の座標系に従ってその座標が変換される。粒子の散乱，核分裂反応はＸＳＥＣによって計算される。Fig.2.2に幾何形状の概念図を示す。粒子は核分裂反応を起こす毎にその位置を記録され，次の世代の初期値として使われる。Table 2.1に示すように，粒子ループが終わる毎に，実効増倍係数とその標準偏差が計算される。乱数を使用する処理は，１－２初期エネルギー順位の決定，１－３初期方向余弦の決定，２－１飛程の計算，９－４ルシアンルーレット，９－５散乱によるエネルギー順位の決定，９－６異方性散乱による方向余弦の決定，９－７散乱による方向余弦の決定，10－２次世代中性子源位置決定である。

これらの処理を（2.7）(2.3）式に対応させたものをTable 2.2に示す。この表より次の２つのことがわかる。

① 異なった物質，形状の集合体中の中性子輸送を扱うため，飛程と位置の修正処理に計算時間（コスト）を必要とする。
② ＢＥＧＩＮは，式に対応する処理，分散低減処理に分類できる。

この内，プログラムの大部分を占める，飛程と位置の修正処理に対するアルゴリズムは，炉形状データ入力方法，座標系の取りかた等々によって決まり，色々なアルゴリズムを考えることができる。従って，これらの処理は他の中性子輸送モンテカルロコードと同一に議論することはできない。しかしながら，アルゴリズム的に意味のある処理別あるいは物理的に意味のある処理別に分類することによって，他のコードのベクトル化の指針とすることができる。

Table 2.1 Contents and time distribution in original soubroutine BEGIN

No.	CONTENTS	COST %
1	FSTART	1.05
1-1	Get starting condition for fission, box	0.02
1-2	Energy is determined using a random number FLTRN(~0%)	0.11
1-3	Direction is determined using a random number GTISO(~0%)	~0
1-4	Fix tracing condition for core, reflector, limit of region	0.04
1-5	Compute total weight as neutron flux	0.03
	Others(IF ~GOTO)	0.85
2	PATH	18.61
2-1	Determine path length using a random number EXPRN(6.01%)	7.29
2-2	Compute new position	11.32
3	Inward CROSsing	8.06
3-1	Set-up inner region information	1.06
3-2	Check geometry	1.00
3-3	Check inward crossing (Sub.CROS)	2.24
3-4	Replace x,y,z position if inward crossing (Sub.CROS)	
3-5	Replace transport distance if inward crossing (Sub.CROS)	
3-6	Compute neutron flux	0.90
	Others(Calling overhead, IF ~GOTO)	2.86
4	FINBOX	0.
4-1	Compute box No. if crossing a core from a reflector	
4-2	Compute new position if crossing a core from a reflector	
5	POSIT	6.39
5-1	Check geometry region at new position	6.39
6	Outward CROSsing	9.89
6-1	Set-up outer region information	0.08
6-2	Check geometry	1.08
6-3	Check outward crossing (Sub.CROS)	5.18
6-4	Replace x,y,z position if outward crossing (Sub.CROS)	
6-5	Replace transport distance if outward crossing (Sub.CROS)	
6-6	Compute neutron flux	2.03
	Others(Calling overhead, IF ~GOTO)	1.52
7	ARRAY	3.39
7-1	Check boundary crossing on box	2.68
7-2	Compute neighbor box number if boundary crossing	0.65
	Others(GO TO)	0.06
8	ALBEDO	~0.
8-1	Compute energy,weight,direction on albedo reflector	
9	XSEC	38.10
9-1	Compute neutron flux	1.86
9-2	Compute fission absorption and new weight	2.01
9-3	Save information with splitting particle	1.00
9-4	Russian roulette using FLTRN(~0%)	1.02
9-5	Determine out-going energy group using FLTRN(1.89%)	9.10
9-6	Determine anisotropic scattering using AZIRN(8.12%)	22.23
9-7	Determine out-going direction using GTISO(0.18%)	
	Others(Set-up data)	0.88
10-1	Compute flux sorted by energy group and region	4.99
10-2	Save informations with fission neutron with FLTRN(1.88%)	8.54
10-3	Get informations with splitted particle	0.
11-1	Compute Keff, fission density	~0.
11-2	Compute standard deviation	0.20
	TOTAL	99.22

Table 2.2 Contents and time distribution in original soubroutine BEGIN froma view point of transport equation

No.	CONTENTS	Corresponding contents of Table 2.1	COST %
a	Computing the equation		81.34
a-1	Compute free path, new position from the integrating factor	2-1, 2-2	18.61
a-2	Compute neutron flux	3-6, 6-6, 9-1, 10-1	7.92
a-3	Compute scattering term	9-5, 9-6, 9-7	31.33
a-4	Compute fission reaction term	9-2	2.01
a-5	Correct free path because of changing geometry and matter	3-1, 3-2, 3-3, 3-4, 3-5, 4-1, 4-2, 5-1, 6-1, 6-2, 6-3, 6-4, 6-5, 7-1, 7-2	20.42
a-6	Set initial condition	1-1, 1-2, 1-3, 1-4, 1-5	1.05
a-7	Compute albedo reflector	8-1	~ 0.
b	Variance reduction		10.76
b-1	Stack splitting particle	9-3	1.00
b-2	Russian roulette	9-4	1.02
b-3	Determine new reaction position	10-2, 10-3	8.54
b-4	Compute Keff, fission density	11-1	~ 0.
b-5	Compute standard deviation	11-2	0.20
		TOTAL	92.1

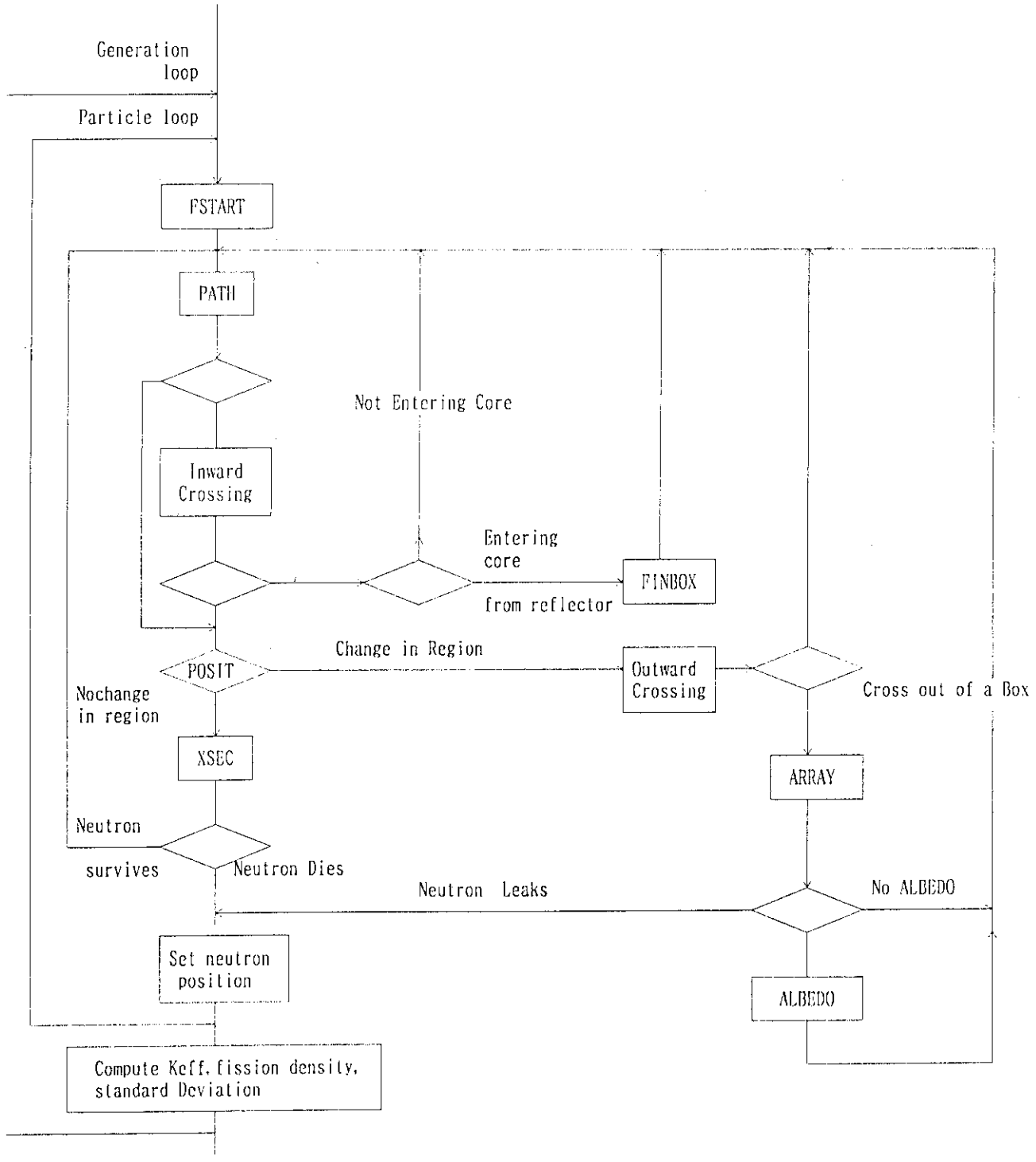


Fig. 2.1 Neutron tracking loop in subroutine BEGIN

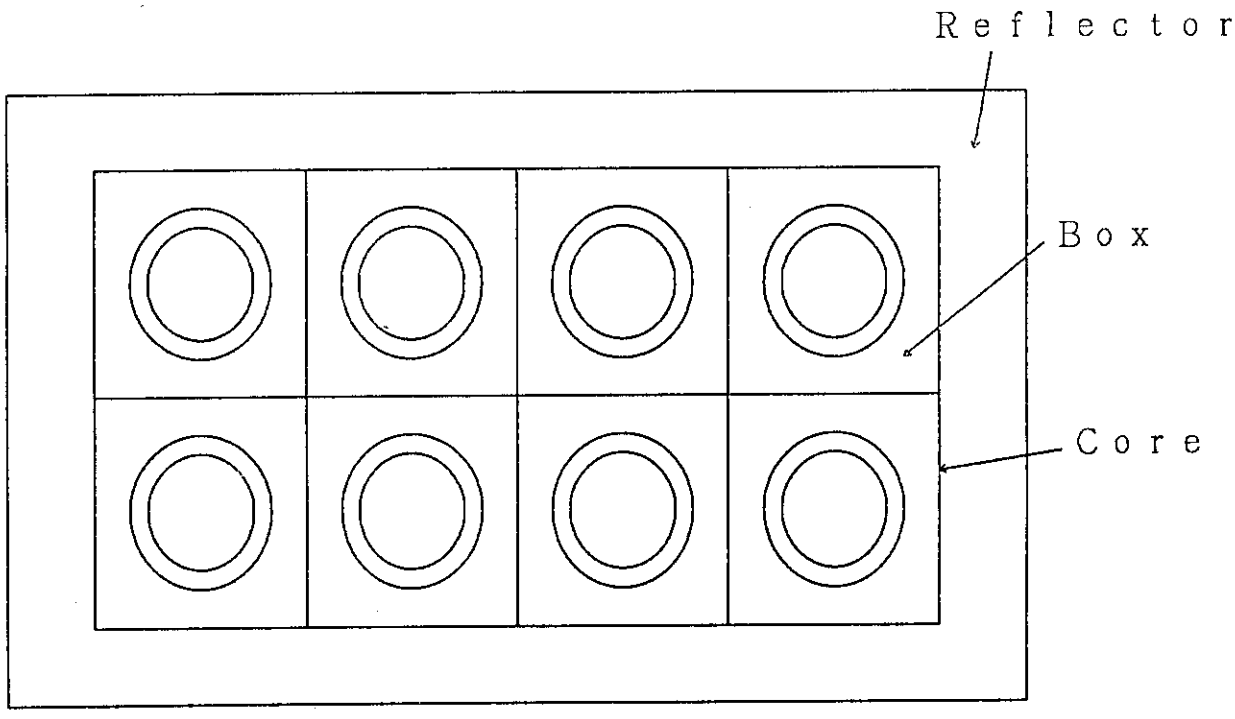


Fig. 2. 2 Example of geometrical assembly using geometry options

3. ベクトル化

ベクトル計算機によって計算時間の短縮を図るためには，時間がかかる計算をベクトル計算で行う必要がある。ベクトル計算は，Fig.3.1のベクトルユニット上で行われる。プログラムのある部分が，コンパイクすることによってベクトルユニット上で動く命令に置き換えられた時，その部分は自動ベクトル化されたと言う。ベクトル化によるスカラ／ベクトル相対性能は，次のAmdahlの法則によって決まる。

$$\text{相対性能} = \frac{\text{スカラ計算時間}}{\text{ベクトル計算時間}}$$

$$= \frac{1}{(1-V)+V/\alpha} \qquad (3.1)$$

ここに，V：ベクトル化率（＝ベクトル化部分のスカラ計算時間／全スカラ計算時間），α：ベクトル／スカラ速度比（＝ベクトル化部分のスカラ計算時間／ベクトル化部分のベクトル計算時間）

V＝90％と仮定すると相対性能は最大で10倍となる（$\alpha \to \infty$）。従って，ベクトル計算の十分な効果を得るためには，全スカラ計算時間の90％以上をベクトル計算するのが目安となる。

通常，科学技術計算のシミュレーションコードは多機能であるので，何を計算するかによってプログラム中の計算時間がかかる部分が変化する。また同種類の計算でも，計算の規模によって同様なことが起こる。例えば今回使用したデータはアルベド反射体を用いていない。これらのことを考慮してスカラ計算時間の90％以上をベクトル化するためには，プロダクションランで用いるデータを用いて計算時間がかかる部分を調べる必要がある。

本章では，プロダクションランで用いるデータ及びそれを用いた時の，各処理の計算時間分布（以後，コスト分布と呼ぶ）を示す。更に，自動ベクトル化を促進するためのプログラムの変更（これをベクトル化と呼ぶ）方法を述べる。また，使用したデータに対してのベクトル化率を示す。

3.1 入力データ

コスト分布を調査するための入力データとして，原研で行われたＴＣＡ（Tank type critical assembly）臨界試験のためのデータの一つを用いる。このデータは，19×19のアレイにU^{235}が23％のUO_2の燃料棒を入れ，Fig.3.2のようにコアタンクに配置し，臨界水位を計算する[11]。断面積ライブラリは，原研で作られたＭＧＣＬライブラリ[12]を用いて計算を行った。一世代の初期中性子数は300，世代数は203で中性子の位置決めのために最初の3世代を使う。この3世代の計算は実効増倍係数の計算に考慮されない。臨界計算ではこの問題より複雑な形

状を扱う場合もあるが，この計算は領域の境界を中性子が横切るという基本的な計算が含まれているので，データとして使用した。

3.2 コスト分布

サブルーチン，FORTRANで書かれたプログラムの各行毎の計算時間は，実行時のステートメント単位で推定するツールFORTUNE[13]によって知ることができる。FORTUNEは，翻訳されたアセンブラ命令の実行時間を考慮した計算時間を，和と差の計算時間を基準とした単位で解析する。FORTRANプログラム以外は，サブルーチン単位でFORTRANのライブラリの時間計測ルーチンを用いて測定し，全計算時間に対する割合から各処理にたいする計算時間の割合を推定する。

サブルーチンBEGIN，とCROS（以下，BEGIN，CROSと呼ぶ)は，コード全体のFORTUNEコストの99.7%を占める（Table 3.1参照）。CROSはBEGINから呼ばれる。アセンブラルーチンで作られている乱数ルーチンの殆どはBEGINで呼ばれる。2章で述べたとおり中性子輸送計算はBEGINによって行われているため，主要計算部分にコストが集中していることがわかる。従って，BEGINとBEGINが呼び出している上記サブルーチンをベクトル化すれば，ベクトル化率90%以上を達成することができる。

BEGINの各処理に対するコスト分布の割合はTable 2.1に示した通りである。アセンブラルーチンで作られている乱数ルーチンは，一回当たりの実行時間に実行回数を掛けて，FORTUNEコストに反映した。コストは少数の処理に集中せず，広い範囲に渡って均等に分布していることがわかる。FINBOX，ALBEDOのコストが殆ど0%なのは，このデータでは反射体を使用していないためである。Table 2.1からベクトル化率を90%以上にするためには，実行されている処理の殆どをベクトル化する必要があることがわかる。

つぎにTable 2.2を見ると，a－1の飛程と位置の計算，a－3 散乱項の計算，a－5 飛程と位置の修正にコストが集中していることがわかる。また，b－3 次世代中性子源位置決定処理が約8.5%のコストを占め，b分散低減法にもそのコストが無視できない処理があることがわかる。この計算においてa－6，a－7の初期・境界条件設定処理は計算時間を無視できることがわかる。

3.3 スタックドリブン法

ベクトル化を行うため，オリジナルコードのBEGINで用いられてきたヒストリベース法をスタックドリブン法に変更する必要がある。ここでは，ヒストリベース法，イベントベース法とその応用のスタックドリブン法について説明する。ヒストリベース法は，1粒子が吸収されて死ぬか系外に飛び出すか終了時間になるまで，乱歩を繰り替えす。Fig.3.3においては，例えば粒子番号1のⓈ，Ⓑ，Ⓔを計算し，次に例えば粒子番号2のⓈ，Ⓑ，Ⓒ，Ⓔの順で最後の番号粒子まで計算する。これに対しイベントベース法では，同じ処理のイベントの粒子を同じバンクに蓄え，バンク毎に計算を行う。従ってバンクに溜った粒子数がベクトル長になる。例

えば，Event 1では処理Ⓒの粒子番号3，4，5をまとめて計算し，次に処理Ⓑの粒子番号1，2を計算し，最後に処理Ⓔの粒子番号6を計算する。この方法では，次に起こる処理が変わる毎にバンクが書き直される。これはヒストリベース法には無かった計算で，計算時間の増加につながる。また，バンク中の粒子数が変化するため，ベクトル長が変化する。イベントベース法において，ベクトル長が一番長いバンクを優先して計算する方法をスタックドリブン法と呼ぶ。この方法では2回目の計算が同じバンクである場合がある。

イベントベース法あるいはスタックドリブン法を計算機にプログラミングする場合2つの方法がある。

① Brown等が用いた方法

バンクに属性（位置，方向余弦等々）を蓄え，各イベント毎に次に計算しない粒子の属性をバンクから取り除き，バンクの中に空きが無いようにデータを連続的に詰め替える。これにより，データは連続に取り出すことができる。

② 浅井等が用いた方法

バンクに粒子番号を蓄え，各イベント毎に次に計算しない粒子番号をバンクから取り除き，番号を連続的に詰める。各処理ではこの番号を参照して，データをアクセスする。この方法では，バンクから取り出した粒子番号を手掛かりにデータをアクセスする。

Brown等が用いた方法は，データが連続的に並んでいるため計算時のデータアクセス時間が速いが，イベント毎に全てのバンクの属性変更が必要になる。浅井等の方法では，バンクの属性が粒子番号のみなのでバンクの変更時間が少ないが，各計算毎に連続アクセスに比べて遅い間接アクセスが使われる。

本論文では，ベクトル化の向上を検討するため，ＶＰ－100向けに再構成したスタックドリブン法に書き直したKENO Ⅳコードを用いた。粒子ループを引き込んでＤＯループを作るだけではベクトル化できない処理について，改めてベクトル化を行った。

このプログラムは8つのバンクを持ち（Fig.3.4参照），粒子ループの終わりにはバンクのベクトル長を調べ一番長いバンクの処理へ振り分けるＩＦ～ＧＯＴＯ文がある。属性を記述する変数及びその計算のための変数は，ベクトル化のために配列変数に書き換えられている。バンクの変更及び利用はFig.3.5のように②の方法で行った。

3.4 各処理のベクトル化

この節では，ベクトル化のためアルゴリズムを変更した処理に対する，ベクトル化方法を述べる。括弧内にTable 2.1に対応する番号を示す。詳細は付録Ａを参照されたい。

① ＰＯＳＩＴのベクトル化（5－1）

テストデータが使用している2つの形状（スラブ，シリンダ）をベクトル化した。

② スプリット処理のベクトル化（9－3）
使用されない処理を分離して，部分ベクトル化を図った。

③ 散乱によるエネルギ群遷移処理のベクトル化（9－5）
2重D0ループを入れ換えベクトル化し，ループアンローリングを行いロード／ストアの回数の低減を図った（付録B参照）。

④ 幾何形状番号による分類処理のベクトル化（3－2，6－2）
IF～THENを用いてベクトル化した。

⑤ バンク変更計算のベクトル化
複数のバンク計算をベクトル計算で行った。

⑥ 乱数のベクトル化
処理（2－1）のEXPRN，（9－5）のFLTRN，（9－6）のAZIRNを富士通の科学用サブルーチンライブラリSSLⅡ／VPの一様乱数ルーチンRANU2を使用してベクトル化した。なお，コストが低い部分で呼ばれている乱数ルーチンは，オリジナルのまま使用した。

⑦ 次世代中性子源位置設定処理のベクトル化（10－2）
ここで使われている乱数FLTRNは，あらかじめ計算しておいた乱数を引用するよりオリジナルのFLTRNを使用したほうが速いので，オリジナルを使用した。

3.5 ベクトル化率評価

全体には，粒子数の増加に従って82～89%のベクトル化率となる（Table 3.2参照）。粒子数の増加に従ってベクトル化率が増加する理由は，可能な場合スカラ計算がベクトル計算と同時に行われるVPアーキテクチャによる。

本章の冒頭で述べたαをコード全体に対して求めると，粒子数300に対して3.4倍である。しかし倍率は2.6倍である。これはベクトル化率が十分でないことに起因する。今回，FORTRAN77／VPのコンパイラのレベルがV10L20からV10L30になり，VU時間（ベクトルユニットを使用している時間）を各処理毎に測定することが可能となったため，これを用いて各処理毎のベクトル化率を算出してTable 3.3を得た。次に個々のカーネルについて述べる。尚，括弧内はTable 2.1の処理番号である。

① サブルーチンCROS（3－2，3－3，3－4，3－5，6－2，6－3，6－4，6－5）は，コンパイラの表示とFORTUNEの解析結果によれば99.0%のベクトル化率であるのに対し，実測値は85.6%，90.5%とこれより低い。この理由としては，CROSの呼び出しオーバヘ

ッド（約730万回呼ばれている）と，アドレス計算等のスカラ計算がベクトル命令と同時に実行できなかった場合の二つが考えられる。

② 中性子束計算（3－6, 6－6)は，ボックス，領域単位に総和を求める計算である。これらの中には，一つ以上の粒子が存在する可能性があるため回帰計算となり，ベクトル化率が62.8%，60.4%と低下する。また，核分裂荷重と吸収荷重及び核分裂密度の計算（10－1）の所でも同様な計算が行われるため，殆どベクトル化率できない。これは，X(L(I))＝X(L(I))＋A(I)，I＝1，Imaxのタイプの計算である。このタイプの計算はベクトル化できる[14]が，このデータではループの平均ベクトル長が33と短いため，性能向上が望めない。

③ ルシアンルーレット（9－4）は，乱数ＦＬＴＲＮを使用する確率が低く実際のベクトル長が短くなるため，オリジナルのアセンブラルーチンを用いた。

④ 異方性散乱処理（9－6）は，乱数ＧＴＩＳＯを使用する確率が低いためオリジナルアセンブラルーチンを用いたが，処理自体はベクトル化が可能である。

処理（10－2）はアルゴリズムを変更することによってベクトル化可能と推測できる。この処理をベクトル化すれば，粒子数300でもベクトル化率は90%を越えると予測できる。

Table 3.1 FORTUNE estimated time distribution in KENO IV code

NO.	ROUTINE	UNITS	LINES	ERR.	EXECUTIONS	COST	%
0001	BEGIN	1	780	0	1	0.419778E+10	86.4
0002	CROS	1	502	0	7275222	0.644059E+09	13.3
0003	FINALE	1	308	0	1	0.394609E+07	0.1
0004	START	1	430	0	1	0.161575E+07	0.0
0005	CORSIZ	1	141	0	1	0.123659E+07	0.0
0006	POSIT	1	69	0	13546	0.116979E+07	0.0
0007	INPUT	1	177	0	1	0.114893E+07	0.0
0008	XSTAPE	1	242	0	1	0.112753E+07	0.0
0009	BOX	1	50	0	1	0.107380E+07	0.0
0010	NSTART	1	47	0	203	0.947043E+06	0.0
0011	CLEAR	1	10	0	3	0.526239E+06	0.0
0012	KEDIT	1	125	0	1	0.465927E+06	0.0
0013	KENOG	1	404	0	1	0.352211E+06	0.0
0014	VOLUME	1	152	0	1	0.337533E+06	0.0
0015	AREAD	1	252	0	42	0.333718E+06	0.0
	IREAD				91		
	FREAD				1428		
0016	FREAK	1	88	0	1	0.326871E+06	0.0
0017	KENO	1	567	0	1	0.373420E+05	0.0
0018	FILBOX	1	98	0	1	0.261180E+05	0.0
0019	MESAGE	1	17	0	1	0.200390E+05	0.0
0020	FHLPR	1	31	0	3	0.200070E+05	0.0
0021	MAIN	1	71	0	1	0.132720E+05	0.0
0022	RDREF	1	112	0	1	0.519000E+04	0.0
0023	SAVE	1	19	0	5	0.315000E+04	0.0
0024	PULL	1	5	0	204	0.224400E+04	0.0
0025	DATIM	1	43	0	1	0.107800E+04	0.0
0026	JOMCHK	1	483	0	1	0.467000E+03	0.0
0027	ALOCAT	1	7	0	1	0.150000E+02	0.0
0028	TIMFAC	1	6	0	1	0.140000E+02	0.0
0029	STORE	1	31	0	0		
	STORE1				0		
0030	JOM7	1	69	0	0		
	JM7				0		
0031	XXMOD	1	457	0	0		
0032	JOM6	1	121	0	0		
	JM6				0		
0033	ST1D	1	39	0	0		
0034	JOM9	1	57	0	0		
	JM9				0		
0035	JOM12	1	52	0	0		
0036	WARR	1	45	0	0		
0037	LOOKZ	1	61	0	0		
	LOKSET				0		
0038	JOMIN	1	196	0	0		
0039	JOM11	1	46	0	0		
0040	SFLRAN	1	9	0	0		
0041	GEOM	1	80	0	0		
0042	RESTRT	1	115	0	0		
	WRTRST				0		
	FINRST				0		
0043	READSG	1	27	0	0		
0044	REA	1	31	0	0		
0045	JOM5	1	83	0	0		
	JM5				0		
0046	JOM4	1	91	0	0		
	JM4				0		
0047	JOM17	1	39	0	0		
0048	JOM16	1	37	0	0		
0049	JOM13	1	59	0	0		
0050	MATK	1	31	0	0		
0051	MAKREF	1	242	0	0		
0052	JOM10	1	64	0	0		
	JM10				0		
0053	ARAMOD	1	159	0	0		
0054	LABL	1	39	0	0		
0055	ALBIN	1	195	0	0		
	ALBEDO				0		
0056	AJOINT	1	79	0	0		
			7790	0		0.485657E+10	

Table 3.2 Performance at 300, 900 and 1800 particles for vectorized KENO IV code

VP-100, unit:sec

The Number of particles	Orig. code	Vectorized code					
	Scalar[1] CPU time using OPT3	Scalar[2] CPU time using OPT3	Vector[3] CPU time (VU time)	T.A.V.L.	V.R. %	1)/3)	2)/3)
300	332.28	557.49	226.18 (118.24)	42, 54, 33	81.3	1.47	2.55
900	329.98	610.22	185.36 (103.59)	103, 139, 80	86.6	1.78	3.29
1800	332.63	670.52	175.34 (99.87)	190, 257, 148	88.7	1.90	3.82

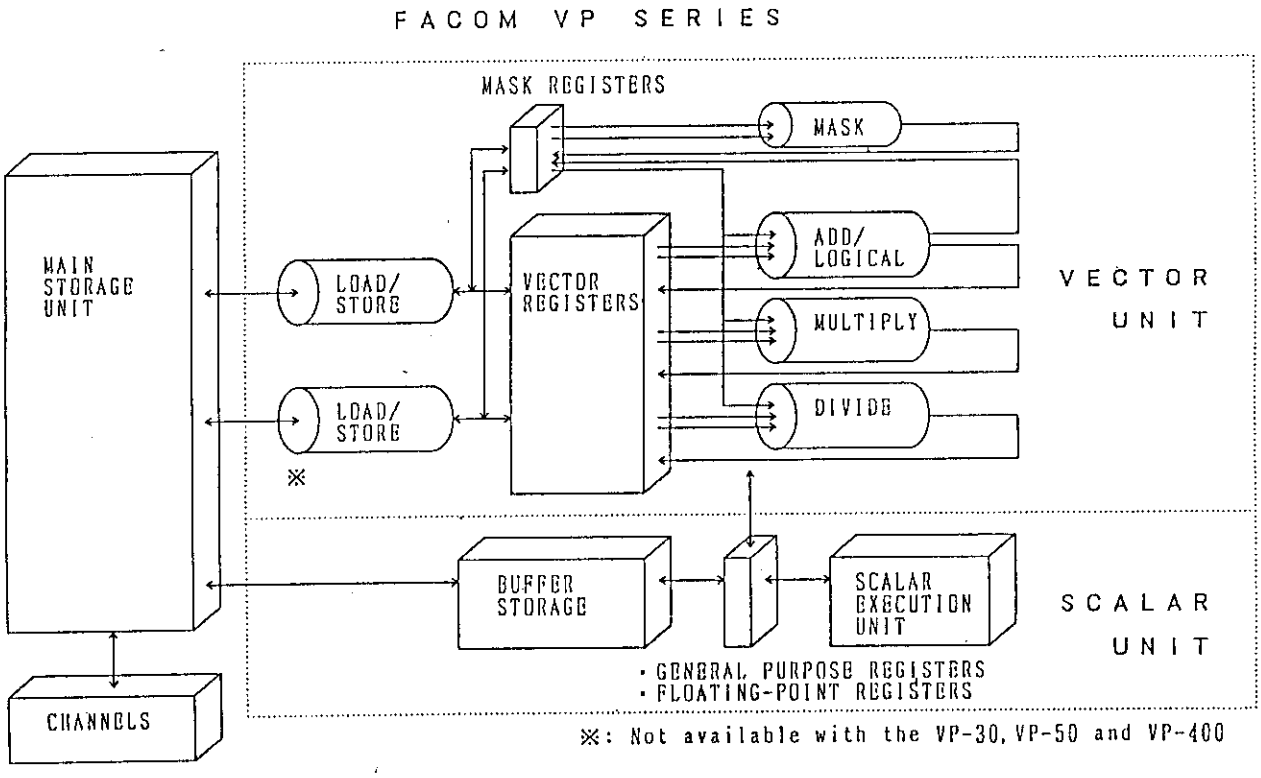
T.A.V.L.: Typical averaged vector length
V.R.: Vectorization ratio

Table 3.3 Performance of time consuming kernels in vectorized BEGIN

VP-100, unit:sec

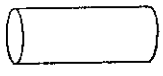
Corresponding contents of table 2.1	Loop cnd.	Scalar[1] CPU time	Vector[2] CPU time	(VU time)	V-Ratio (%)	α	1)/2)	Ave.V-length
PATH								
2-1 (EXPRN)	V	47.96	6.520	(5.273)	97.4	8.86	7.36	1E4
2-2	V	40.27	10.87	(10.70)	100.	3.75	3.70	42
compute bank	V	14.56	6.708	(6.131)	96.0	2.28	2.17	42
Inward cross								
3-1	V	4.089	3.882	(3.684)	95.2	1.06	1.05	26
3-2,3,4,5	V	51.68	18.48	(11.05)	85.6	4.00	2.80	--
3-6	M	7.980	4.684	(1.715)	62.8	2.92	1.70	26
POSIT								
5-1	V	37.00	22.20	(18.86)	91.0	1.78	1.67	39
Outward cross								
6-2,3,4,5	V	47.19	15.98	(11.48)	90.5	3.69	2.95	--
6-6	M	11.70	5.979	(1.347)	60.4	5.25	1.96	54
compute bank	V	7.443	2.883	(2.103)	96.0	2.77	2.58	54
ARRAY		29.54	11.30	(9.946)	95.4	2.83	2.61	--
7-2	V	4.979	2.297	(2.103)	96.2	2.28	2.17	40
XSEC								
9-1,2	V,S	28.72	--	--	--	--	--	33
9-3	V,S	7.514	1.911	(1.648)	96.5	4.40	3.93	33
9-4	V	13.88	7.516	(3.417)	70.5	2.86	1.85	33
9-5	V,S	4.953	1.280	(1.270)	99.8	3.89	3.87	33
(FLTRN)	V	7.313	1.204	(0.791)	94.4	8.72	6.07	33
	V,S	37.14	11.07	(5.820)	85.9	5.48	3.36	17,33
9-6 (AZIRN,GTISO)	V,S	41.82	10.48	(3.981)	84.5	8.87	3.99	33,1E4
9-7	V	60.22	14.90	(13.99)	98.5			
10-1	V,S	25.80	--	--	--	--	--	
10-2,3 (FLTRN)	S	18.92	25.18	(0.05458)	--	--	.751	299

Number of particles : 300, scalar computation if vector length < 10 : 3-2, 3-3, 3-4, 3-5, 3-6, 6-2, 6-3, 6-4, 6-5, 6-6, 9-1, 9-5, 9-6

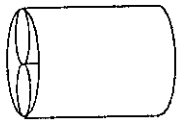


PIPELINE THROUGHPUT

(1) VP－30／50／100



(2) VP－200



(3) VP－400

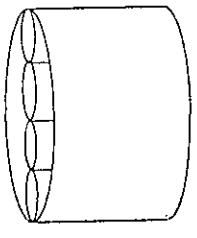
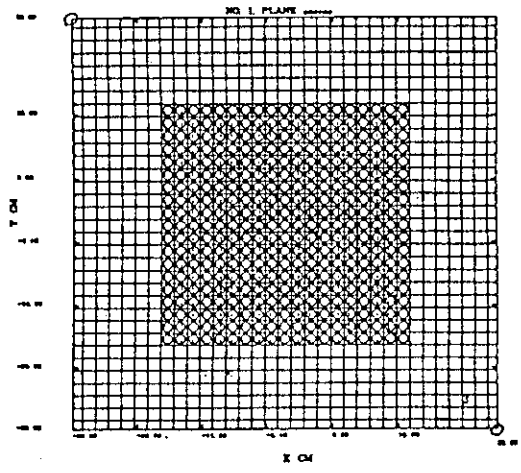
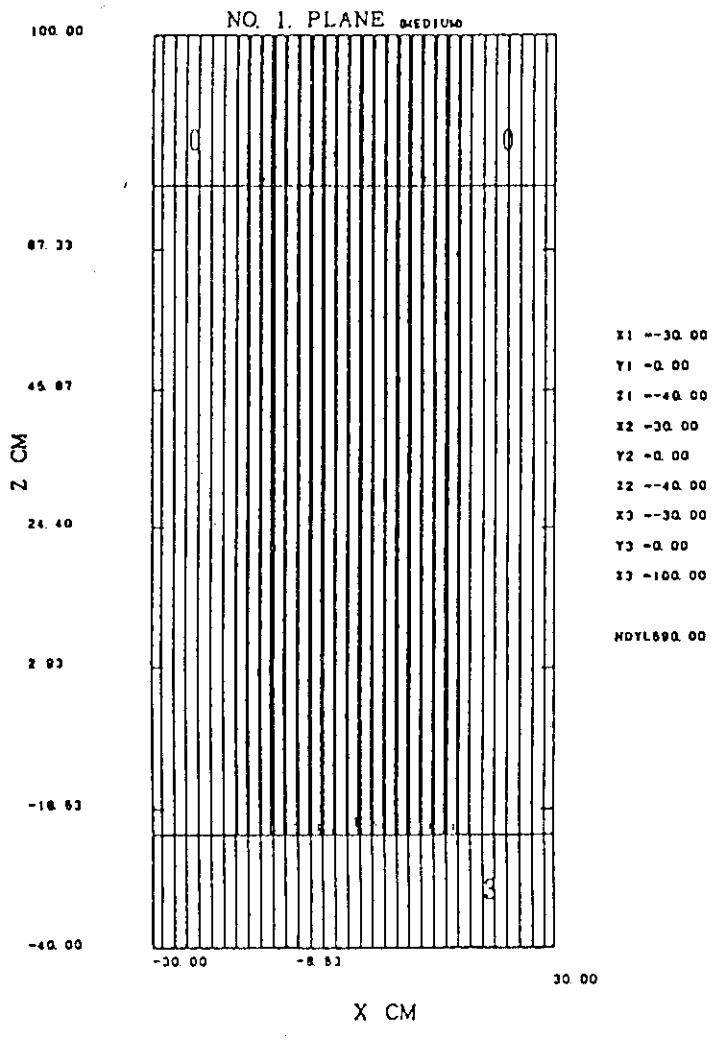


Fig. 3.1 FUJITSU vector processor brock diagram

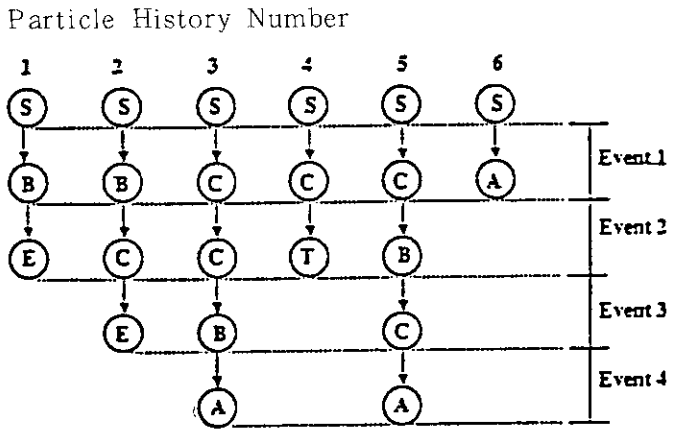


top view



side view

Fig. 3.2 Geometrical scheme of the experimment



A : absorption, B : boundary crossing, C : collision, E : escape,
S : saurce, T : end of time step, → free − flight

Fig. 3. 3 Brief illustration for an image of particle history and event

a. Set"NBANK1"

```
      DO 1 N=1,NPMAX
      IF (~) THEN
        L =L+1
        NBANK1 (L ) =NBANK1 (N )
      END IF
    1 CONTINUE
        NB1MAX=L
```

b. Use"NBANK1"

```
      DO 2 N=1,NB1MAX
      L =NBANK1 (N )
        =X  (L )
        =Y  (L )
        .
        .
        .
    2 CONTINUE
```

Fig. 3. 5 Event based algorithm using particle number sorting

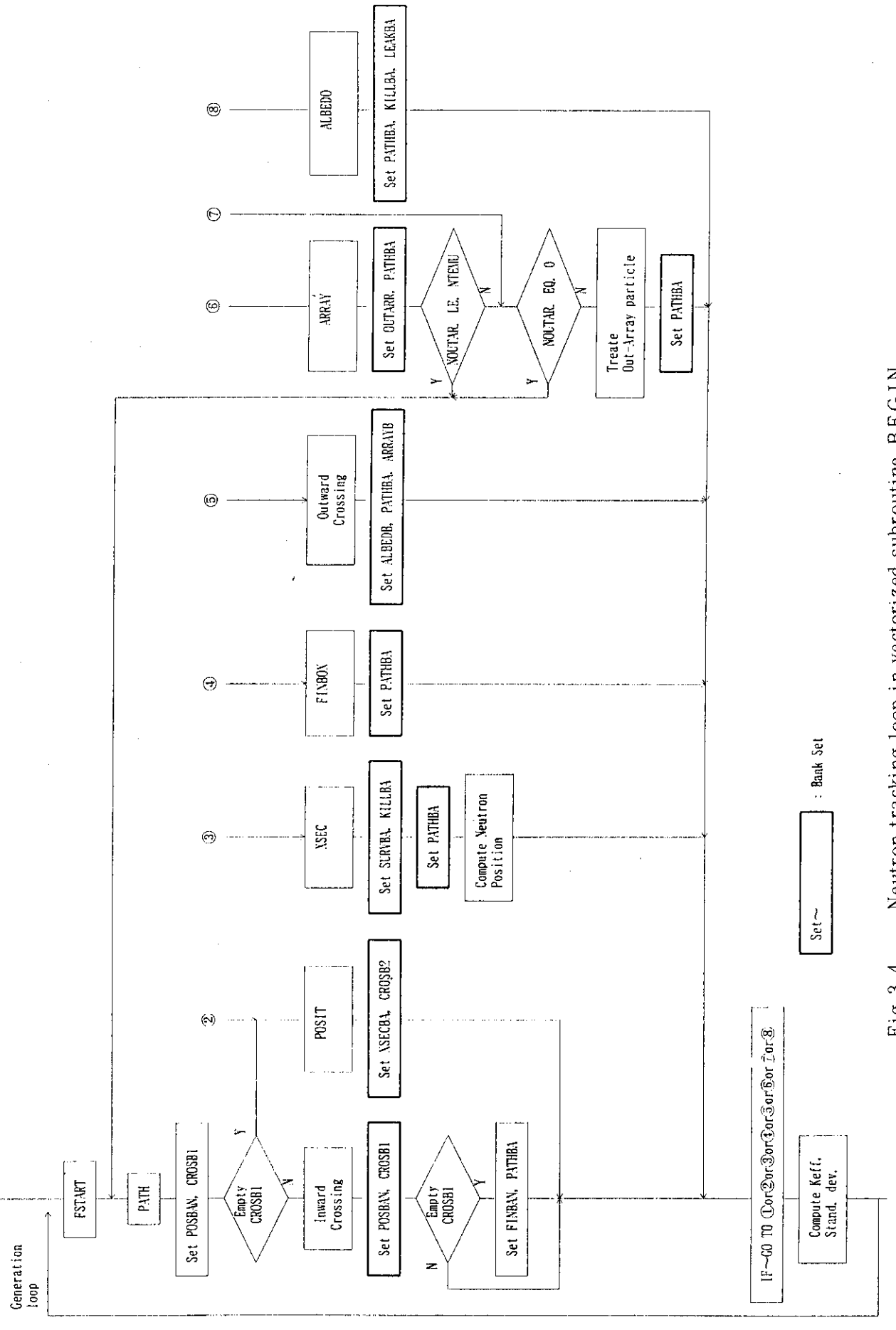


Fig. 3.4 Neutron tracking loop in vectorized subroutine BEGIN

4. 性 能 評 価

4.1 ベクトル化コードのスカラ性能

中性子輸送モンテカルロコードをベクトル化すると，ベクトル化コードのスカラ計算時間は，オリジナルコードのそれに比べて増加する。原研においてこれまでベクトル化した他のコードも同様である[15]。ＫＥＮＯⅣコードでは，粒子数300の場合，スカラ計算時間が1.78倍増加する（Table 4.1参照）。この原因は次の２つである。

① 変数を配列に変えたことによる番地計算等の命令の増加

オブジェクト命令の実行時間を考慮したＦＯＲＴＵＮＥのコストをオリジナルコードとベクトル化コードのスカラ計算で比較したものをTable 4.2に示す。この表は，スカラ版とベクトル版の命令の増加が1.7倍であることを示している。粒子数と世代数を掛けた値が一定になるように粒子数を変えた時，主要コスト部分の処理の実効回数はほぼ一致する。従って，全計算量を一定にして粒子数（ベクトル長）を変化させることができる。

② スタックドリブン法を採用したため変数が配列変数となり，異なった配列要素のアクセスが頻発することによるキャッシュの使用効率の低下

ベクトル版のスカラ計算実行時間は，粒子数が増えるに従って遅くなる（Table 4.1参照）。実行時間は命令の増加とキャッシュメモリの効率の両方の情報を含む。Table 4.2のオリジナルコードとベクトル化コードのＦＯＲＴＵＮＥコストを比較すると，粒子数の増加が命令数の増加に依存していないことがわかる。これより，実行時間の増加はキャッシュの影響と推測できる。言い換えると，粒子数の増加と共にキャッシュの使用効率が低下していることが推測できる。一方，オリジナル版の実行時間は粒子数に依存しない。これは，粒子の増加によって使用メモリがそれほど増加しないためと推定できる。

ベクトル版のみが行うバンク計算による計算量の増加も原因として考えられるが，この部分のＦＯＲＴＵＮＥコストはコード全体の３％，実測で４％であり，ベクトル化版を作るための書き替えによって増えた計算の中では，小さい量である。

4.2 ベクトル化コードのベクトル性能

計算時間を３章(1)式に代入すると，オリジナルコードと比べたベクトル化コードの相対性能は，1.47，1.78，1.90と低い値となる（Table 3.2参照）。この原因を調べるため，個々の処理の時間を粒子数300に対して測定した。時間測定方法は，付録Ｃを参照されたい。

各処理の性能をTable 3.3に示す。$\alpha \leq 4$の処理において，ベクトル性能を低くする原因は次の４つである。

① 多重ＩＦ文の使用

ＰＯＳＩＴ（５－１）は計算型ＧＯＴＯ文と多重ＩＦ文を使用している。この内，計算型ＧＯＴＯ文はInward／outward crossingで幾何形状分類を行う計算型ＧＯＴＯ文〔付録Ａの④〕において，3.11，5.14，6.17倍と十分速くなることが確認できる（Table 4.3参照）。一方ＰＯＳＩＴで使用している多重ＩＦ文を取り出してベクトル化した所，最大性能が2.6～2.7倍と低いことがわかった〔付録Ｄ〕。ＰＯＳＩＴの計算は，全計算時間の10％である。従って阻害原因の一つは，多重ＩＦ文をベクトル計算した時の性能が低いためである。この他，（3－6，6－6，Bank計算）もＩＦ文を含んでいる。

② 間接アクセスの使用

粒子数を増してベクトル長を長くしてもコード全体の性能向上が図れない原因を調査するため，典型的なモンテカルロ計算であるＰＡＴＨの位置計算部（Fig.4.1参照）のＣＰＵ時間を測定した所，ベクトル長が短い段階でベクトル性能（ＣＰＵ時間）が飽和していることがわかった（Table 4.4参照）。この計算は，位置の計算がVector = Vector + Vector * Vectorの形をしており，ＶＰアーキテクチャの中で高い性能を出せる可能性を持つ。この矛盾点を調べるため，各パイプラインのタイミングチャートを解析して実効時間比としてまとめたものをFig.4.2に示す。タイミングチャートを解析するに当たっては，パイプラインのリンケージ，コンカレンシー，立ち上がり，立ち下がりを考慮した。なお，ＩＦ文の真率は１％である。この図は，この計算がベクトル長が短いうちから間接アクセスのためロード／ストアパイプがビジーとなり，ベクトル長を長くして立ち上がり時間を無視できるようにしても性能が向上しないことを示す。

全てのタイミングチャートを調べる代わりに，各処理の間接アクセス命令の割合から間接アクセスのためロード／ストアパイプラインの状態を推定した（Table 4.5参照）。その結果，殆どの処理が同様な状態になっていることがわかる。

③ バンクコンフリクト

Table 3.3は，処理（３－１）のαが1.06と特に小さいことを示す。これは，同じデータをアクセスするためにバンクコンフリクトが生じているためである。

④ ベクトル長の減少

ＰＡＴＨの乱数計算を含まない部分は粒子数を増してベクトル長を長くしてもＣＰＵ時間を見るとあまり性能向上しないことがわかる（Table 4.4参照）。以前よりその原因はベクトル長の減少であると指摘されてきた[5]。そこでベクトル長10以下をスカラ計算とし，ＣＰＵ時間を推定した。

全てベクトル化した場合ＣＰＵ時間＝ 11.96 秒，ＶＵ時間＝ 10.86 秒である。ベクトル長10以下のものをスカラ化した場合ＣＰＵ時間＝ 11.13 秒，ＶＵ時間＝ 8.607 秒である。これよりベクトル長が10以下のスカラ計算時間を算出できる。

（ベクトル長が10以下のスカラＣＰＵ時間処理）

＝（ベクトル長10以下をスカラ処理した場合のスカラＣＰＵ時間（＝ 11.13－8.607）秒）

－（ベクトル化Ｄ０ループのスカラＣＰＵ時間（＝ 11.96－10.86）秒）

＝ 1.42 秒

スカラＣＰＵ時間は47.3秒ゆえ，ベクトル長が10以下の計算時間はＰＡＴＨ全体のスカラ計算時間の３％である。従って，ベクトル長の短い計算時間は高速化の阻害原因でない。

またTable 3.2より，ベクトル化コードの性能が粒子数の増加と供に上昇するのは，粒子数の増加と供にベクトルＣＰＵ時間が飽和しているにもかかわらずスカラＣＰＵ時間が増えるためであり，見掛け上のものであることがわかる。

Table 4.1 Scalar computation time of varying the number of particles for original and vectorized KENO IV code

VP-100, unit:sec

The number of particles	Original code	Vectorized code	
	Scalar CPU time[1] using "OPT3" option	Scalar CPU time[2] using "SCALAR" option	2)/1)
300	332.28	589.92	1.78
900	329.98	638.48	1.93
1800	332.63	694.83	2.09

Table 4.2 Analysis of dependency between the number of particles and the number of instructions using FORTUNE cost

The number of particles	FORTUNE cost		2)/1)
	Original code [1]	Vectorized code [2]	
300	4.842 E9	8.211 E9	1.70
900	——	8.070 E9	——
1800	4.901 E9	8.054 E9	1.64

Note :Cost of the random number generator is excluded.

Table 4.3 Vector performance of geometrical sorting in CROS

VP-100, unit:sec

The number of particles	Orig. code	Vectorized code	1)/2)
	Scalar[1]	Vector[2]	
	CPU time	CPU time (VU time)	
300	12.40	3.984 (2.120)	3.11
900	12.61	2.452 (1.578)	5.14
1800	12.78	2.070 (1.448)	6.17

Table 4.4 Vector performance of a part of PATH (No. 2-2 on Table 2.1)

VP-100, unit:sec

The number of particles	Orig. PATH	Vectorized PATH	1)/2)
	Scalar[1]	Vector[2]	
	CPU time	CPU time (VU time)	
300	47.03	11.13 (8.602)	4.23
900	61.19	8.845 (7.929)	6.92
1800	91.12	8.374 (7.791)	10.88

Table 4.5 Ratio of indirect load/store instructions for each kernel in vectorized BEGIN

Corresponding contents of table 2.1	The number of vector instructions		2)/1)
	total[1]	Indirect load/store[2]	(%)
2 − 2 *	62	21	34
Compute Bank	55	6	11
3 − 1	16	13	81
3 − 6	65	15	23
7 − 2	119	35	29
9 − 1	19	15	79
9 − 2 *	20	12	60
	29	9	31
	20	6	30
9 − 3 *	14	4	29
9 − 4 *	17	4	24
	11	1	9
9 − 5 *	4	1	25
	40	10	25
	6	2	33
9 − 6 *	23	7	30
9 − 7 *	95	27	28
10 − 1	10	3	30
	16	8	50
	7	4	57

* : A portion of the kernel

```
      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
```

Fig. 4.1 Statement of a part of PATH (No.2−2 on Table 2.1)

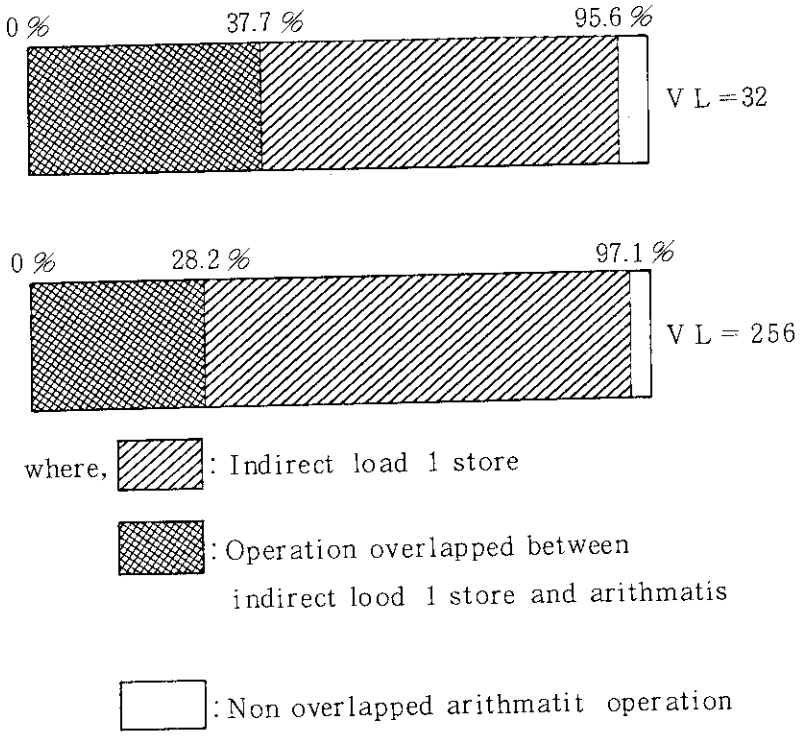


Fig. 4.2 Timing ratio of indirect load/store and arithmatic operation in a part of PATH (No.2−2 on Table 2.1)

5. 性能評価における問題点

ベクトル化されたＫＥＮＯⅣコードのスカラ／ベクトル相対性能は，粒子数1800の時，3.8倍であった。一方，Brown等によって行われたＭＣＶコードのベクトル化版は，ＣＤＣ 7600とＣＹＢＥＲ 205を比較して45倍である[16]。両者のスカラ性能比を２倍と仮定して，ベクトル化による性能向上は23倍と考えられる。この違いの主な原因として，ＶＰ－100はキャッシュメモリを持ちＣＹＢＥＲ 205は持たないことを揚げることができる。

ＣＹＢＥＲ 205（64bit，２パイプで）の命令実行時間をサイクルタイムで表示したものをTable 5.1に示す。

Brownらの計算は，１バッチ当たり16000個の中性子を用いたため，立ち上がり時間は無視できると考えられる。例えばランダム分散のスカラ／ベクトル相対性能は，次のようになる。

$$\lim_{N\to\infty} \frac{15\,N}{83 + 1.25\,N} = 12\text{倍}$$

同様に，加算，乗算は10倍である。従ってＣＹＢＥＲ 205上では，ベクトル化版ＫＥＮＯⅣのようにデータアクセスのためロード／ストアパイプラインがビジーになっていてもいなくても，スカラ／ベクトル相対性能は10倍を超えると推定できる。

一方，Ｍ380ではメインメモリアクセスタイムは20ＣＰＵクロックとＣＰＵに比べて遅いため，キャッシュメモリ方式を採用している[18]。ＶＰ－100はＭ380のアーキテクチャをスカラユニットに採用しており，64Ｋバイトのキャッシュメモリを持つ二階層メモリシステムである。スカラ計算においてデータがメインメモリからアクセスされた場合とデータがキャッシュメモリにある場合のアクセス時間比は，大体10倍である。従って，スカラ／ベクトル相対性能はキャッシュメモリの利用率によって異なる。キャッシュメモリの使用効率が高いほどスカラ／ベクトル相対性能は低くなる。

キャッシュメモリの利用率の例を，Table 4.4におけるＫＥＮＯⅣのベクトル化コードのＰＡＴＨ（Fig.4.1参照）に見ることができる。この表では世代数を調節して全演算量は同じにしてある。この時のＦＯＲＴＵＮＥコストは，ほぼ一定であり，オブジェクト命令の実行回数が一定であることを確認できる。一方，時間を測定すると，ベクトル版のスカラＣＰＣ時間は粒子数が増えるに従って増加する。このことより，粒子数300の時に最もキャッシュメモリが効率良く使用されていることがわかる。

これらの考察は，パイプラインのコンカレンシー，リンケージを考慮していない荒いものではあるが，次の二つのことが言える。

① スカラ性能がベクトル性能に比べて低い場合，相対性能は向上する（ハードウェアのスペック）。

② キャッシュメモリがある場合，キャッシュメモリの利用率の向上と共に相対性能は低下する。データ量の増加に対し，ベクトル性能が一定でキャッシュメモリの利用効率が低くなる場合，相対性能は向上する（アーキテクチャの違い）。

これらのことから，相対性能のみでコードの性能を評価することは危険であることがわかる。相対性能が高い原因として，スカラ性能がベクトル性能に比べて低く設定されている場合も十分考えられる。

この問題を解決する手段の一例として，F. W. Bobrowicz 等のように〔μsec／track〕[19] を用いることもできるが，全ての処理のＣＰＵ時間が込みのため，アルゴリズムの性能が向上して時間短縮が行われたか，ベクトル計算によって時間短縮が行われたかを判別することができない。ベクトル計算による時間短縮を議論する場合，各処理単位のＣＰＵ時間をスカラ計算とベクトル計算に対して示す必要があると考える。

Table 5.1 Expected scalar and vector preformance of a two-pipe CYBER205[17]

Functional scalar unit	Unit time (clock periods)
Load/Store	15
Addition/Subtraction	5
Multiplication	5

Vector Instruction	Execution time
Random scatter	83+1.25N
Random gather	67+1.25N
Full vector addition	51+0.5 N
Full vector multiplication	52+0.5 N

N : Number of particles

6. まとめと議論

KENO ⅣをＶＰ－100向けにスタックドリブン法でベクトル化し，ＶＰ－100で性能測定をしたところ，次のことがわかった。

① オリジナルコードに対するベクトル化コードの速度向上率は，ベクトル長が長くなる粒子数1800で1.9倍と低い。これは次の理由による；

- オリジナルコードが，キャッシュメモリの効率的使用という形で，スカラ計算機に対して最適化されているためである。
- オリジナルコードに対してベクトル化コードの命令数が増えたため計算時間が増加したためである。
- イベントドリブン法を粒子番号のバンクを使ってプログラミングをした結果，間接データアクセスを多用することになり，ＶＰ－100のロード／ストアパイプラインがビジーとなり，四則演算パイプがしばしばデータを待つこと。更に，この間接データアクセス命令が他のベクトル命令に比べて遅いためである。

② 上記の結果対し，ベクトル化コードのスカラ／ベクトル相対性能は，粒子数1800で3.8倍であった。これは次の理由による；

- ベクトル化コードのベクトル性能が粒子数の増加と供に飽和するのに対し，ベクトル化コードのスカラ性能は，粒子数の増加と供に低下するためである。

KENO Ⅳの式とコードを対応させて処理を分類すると，飛程と位置の修正のためにプログラムの多くを費やしており，計算時間をも費やしていることがわかる（Table 2.2参照）。この部分に計算時間がかかる理由は，原子炉の幾何形状を円筒，四角柱，球等々を使って組立られるようにプログラミングされているためである。これらマンマシンインターフェースの部分は，プログラムによって異なる場合が考えられ，一般論として論じることはできない。これゆえ処理単位毎に計算内容を調べ性能を比較することが，性能評価では必要であると考える。

ＣＰＵキャッシュメモリがあるＶＰ－100と無いＣＹＢＥＲ 205で調べた所，ＣＰＵキャッシュメモリがある方がメモリアクセスのスカラ／ベクトル相対性能が低くなる。従って，性能評価にはスカラ計算とベクトル計算のＣＰＵ時間を知ることが不可欠である。なぜなら，基準となるスカラ性能が低くても，相対性能は向上するからである。今までの適応評価が分かれた原因の一つは，ここにあると考える。

今までの研究では，ＰＯＳＩＴ等の特定カーネルを高速化し，かつアドレス計算を他のベクトル計算と同時に行ってコードの高速化を図る方法を検討した[10]。これに加え今回の性能評価によって，ベクトルロード・ストアの高速化が重要なポイントであることがわかった。これは，Brownの方法（3.3節参照）を用い，間接アクセスより期待値で2倍速い[20]連続アクセスを使うことによって，現在のＶＰシリーズ上で可能となる。この場合の全体性能は，バンクの属性データ分類のための計算量の増加とベクトル性能のトレードオフで決まる。

謝　　辞

本論文をまとめるに当たり，関連資料について御教示いただきました，情報システムセンター石黒美佐子氏に感謝いたします。

KENO Ⅳの内容について御教授いただきました，富士通株式会社科学システム部南多善氏に感謝いたします。

多重ＩＦ文の性能測定をお願いいたしました，情報システムセンター外来研究員平塚篤氏に感謝いたします。

論文全体に渡る校正をお願いいたしました，情報システムセンター外来研究員野々宮厳氏に感謝いたします。

フローチャートの作図をお願いした富士通株式会社山田圭子氏に感謝いたします。

参 考 文 献

1) L.M. Petrie et al. KENO Ⅳ－An Improved Monte Carlo Criticality Program, ORNL－4938 (1975)

2) E.A. Straker et al. : The MORSE Code－A Multigroup Neutron and Gamma－Ray Monte Carlo Transport Cord, ORNL－4585 (1970)

3) L.B. Levitt et al. : "VIM－1, A Non multigroup Monte Carlo Code for Analysis of Fast Cristical Assemblies", AI－AEC－12951, Atomics International (1970)

4) MCNP－A General Monte Carlo Code for Neutron and Photon Transport, LA－7996－M (1978)

5) 浅井，他：ベクトル・パラレル計算処理の原子力コードへの適用性 JAERI－M 87-136 (1987)

6) Martin et al. : Status of Vectorized Monte Carlo for Particle Transport Analysis, The International Journal of Supercomputer Applications, Vol. 1, No. 2, pp. 11－32 (1987)

7) 浅井：スーパーコンピュータの動向と原子力分野におけるベクトル計算処理，日本原子力学会 夏の学校テキスト (1984)

8) 中川，森，佐々木：私信

9) M.H. Kalos：輸送モンテカルロと将来の計算機，原子力におけるソフトウェア開発研究会報告書，JAERI－M 85-017 (1985)

10) 浅井，他：モンテカルロ計算装置の概念検討，原子力知能化システム技術の研究，JAERI－M 89-023, pp. 108－165 (1989)

11) H. Tsuruta et al.: Critical Sizes of Light－Water Moderated UO_2 and PuO_2 Lattices, JAERI 1254 (1978)

12) Y. Naito et al.: MGCL－PROCESSOR: A Computer Code System for Processing Multigroup Constants Library MGCL, JAERI－M 9396 (1981)

13) FACOM OS IV FORTUNE 使用手引書 V 10用 70 S P － 5730 (1987)

14) 折居：ベクトル計算機における粒子コードの高速化手法，情報処理 Vol. 27, No. 11, pp. 1257－1263 (1986)

15) 菅沼，他：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，JAERI－M 88-114 (1988)

16) F.B. Brown: Vectorized Monte Carlo Methods for Reactor Lattice Analysis, ANS, pp. 108－123 (1983)

17) F.W. Hockney et al. : Parallel Conputers 2 Adam Hilgen (1988)

18) M. Takamura, et al. : A Three－Level Memory System for the FACOM M－380, FUJITSU Sci. Tech. J., Vol. 22, No. 1, pp. 86－92 (1986)

19) F.W. Bobrowicz, et al. : Vectorized Monte Carlo Photon Transport, Parallel Computing, Vol. 1, pp. 295－305 (1984)

20) 伊藤，他：ベクトルプロセッサにおけるメモリ・アクセスの制御と性能，情報処理学会第27回（昭和58年度）全国大会 5 P － 2, PP. 163－164

付録A　各カーネルのベクトル化方法

①POSIT のベクトル化 (5-1)

オリジナルループ

```
    DO 9431 IV =1,NPOS
    JV =POSBAN (IV)
430 IGEO  (JV) =IGEOM  (K  (JV) )
    GO TO (460,450,470,460,462,464,466,467 ) ,IGEO  (JV)
450 RSQ=X1 (JV) *X1 (JV) +Y1  (JV) *Y1 (JV)
    IF  (Z1 (JV) .LE. XX  (K  (JV) ,2) .AND.Z1  (JV) .GE. XX  (K  (JV) ,3)
   1     .AND.RSQ.LE. XX  (K  (JV) ,4) )      GO TO 739
    GO TO 479
460 IF  (X1 (JV) .LE. XX  (J  (JV) ,1)
   +     .AND.X1  (JV) .GE. XX  (K  (JV) ,2)   .AND.Y1  (JV) .LE
   1    XX  (K  (JV) ,3)  .AND.Y1 (JV) .GE. XX  (K  (JV) ,4
   +                                          .AND.Z1  (JV) .LE.XX (K  (JV) ,5)
   2  .AND.Z1 (JV) .GE. XX  (K  (JV) ,6) )       GO TO 739
470 GO TO 479
    RSQ  =X1 (JV) *X1 (JV) +Y1  (JV) *Y1 (JV) +Z1 JV) *Z1 (JV)
    IF  (RSQ.LE. XX (K  (JV) ,2) )         GO TO 739
    GO TO 479
462 RSQX =Y1 (JV) *Y1 (JV) +Z1  (JV) *Z1 (JV)
    IF  (X1 (JV) .LE.XX (K  (JV) ,2) ) .AND.X1  (JV) .GE.XX (K  (JV) ,3) .AND.
   +     RSQX.LE.XX (K  (JV) ,4) )  GO TO 739
    GO TO 479
464 RSQY =X1 (JV) *X1 (JV) +Z1  (JV) *Z1 (JV)
    IF  (Y1 (JV) .LE.XX (K  (JV) ,2) ) .AND.Y1  (JV) .GE.XX (K  (JV) ,3) .AND.
   +     RSQY.LE.XX (K  (JV) ,4) )  GO TO 739
    GO TO 479
466 TZ1=Y1 (JV) *XX (K  (JV) ,3)
    IF  (ABS  (XX (K  (JV) ,3) ) .GE.5.0 ) TZ1 =Z1 (JV) *XX (K  (JV) ,3)
    IF  (ABS  (XX (K  (JV) ,3) ) .LE.2.0 ) TZ1 =X1 (JV) *XX (K  (JV) ,3)
    IF  ( ( X1  (JV) *X1 (JV) +Y1  (JV) *Y1 (JV) +Z1  (JV) *Z1 (JV) ) .LE.XX (K  (JV) ,2)
   +       .AND.TZ1.GE.0.0 )  GO TO 739
    GO TO 479
467 NHCYL=ABS  (XX (K  (JV) ,5) )
    SGN=SIGN (1.0,XX (K  (JV) ,5) )
    GO TO (471,472,473,474,475,476 ) ,NHCYL
471 IF  (SGN *X1 (JV) .LT.0.0 ) GO TO 479
    GO TO 450
472 IF  (SGN *Y1 (JV) .LT.0.0 ) GO TO 479
    GO TO 450
473 IF  (SGN *Z1 (JV) .LT.0.0 ) GO TO 479
    GO TO 462
474 IF  (SGN *X1 (JV) .LT.0.0 ) GO TO 479
    GO TO 462
475 IF  (SGN *Z1 (JV) .LT.0.0 ) GO TO 479
    GO TO 464
476 IF  (SGN *X1 (JV) .LT.0.0 ) GO TO 479
    GO TO 464
739 NXSEC=NXSEC+1
    XSECBA  (NXSEC ) =JV
    GO TO 9431
479 NCROS2 =NCROS2+1
    CROSB2  (NCROS2) =JV
9431 CONTINUE
```

〔原因分析〕

計算型GOTO文は、ベクトル化できない。

GO TO 450,462,464 はインプリシトDOループを作るため、ループ全体がベクトル化対象にならない。

IF文の条件式内に2 つ以上の条件式を書くとベクトルコンパイラでは、スカラコンパイラより遅いオブジェクトを作る。

(スラブ・シリンダ判定計算のベクトル化)

再構成ループ

```
*VOCL  LOOP,NOVREC
*VOCL  LOOP,VDOPT
V      DO 99431 IV =1,NPOS
V      JV=POSBAN (IV)
V      MASK¥=.FALSE.
V      IGEO (JV) = IGEOM (K  (JV) )
V      IF (IGEO (JV) .EQ.1 ) THEN
V      IF (X1 (JV) .LE. XX  (K  (JV) ,1)  THEN
V        IF (X1 (JV) .GE. XX  (K  (JV) ,2) )  THEN
V          IF (Y1 (JV) .LE. XX  (K  (JV) ,3) )  THEN
V            IF (Y1 (JV) .GE. XX  (K  (JV) ,4) )  THEN
V              IF (Z1 (JV) .LE. XX  (K  (JV) ,5) )  THEN
V                IF (Z1 (JV) .GE. XX  (K  (JV) ,6) ) THEN
V                MASK¥=.TRUE.
V                ENDIF
V              END IF
V            END IF
V          END IF
V        END IF
V      END IF
V      IF (IGEO (JV) .EQ.2 ) THEN
V      RSQ =X1 (JV) *X1 (JV) +Y1  (JV) *Y1 (JV)
V      IF (Z1 (JV) .LE. XX  (K  (JV) ,2) ) THEN
V         IF  (Z1 (JV) .GE. XX  (K  (JV) ,3) ) THEN
V             IF  (RSQ.LE. XX (k  (JV) ,4) )     THEN
V               MASK¥=.TRUE.
V             ENDIF
V         END IF
V      END IF
V      MASK1 ¥ (JV) =MASK¥
V 99431 CONTINUE
*VOCL  LOOP,NOVREC
*VOCL  LOOP,VDOPT
V      DO 99432 IV =1,NPOS
V      JV=POSBAN (IV)
V      IF (MASK1 ¥ (JV) ) THEN
V        NXSEC =NXSEC+1
V        XSECBA (NXSEC ) =JV
V      ELSE
V        NCROS2=NCROS2+1
V        CROSB2 (NCROS2) =JV
V      END IF
V 99432 CONTINUE
```

②スプリット処理のベクトル化 (9-3)

オリジナルループ

```
      DO 9908 IV=1,NXSEC
      JV=XSECBA (IV)
  761 IF (WT (JV) .LE.WTHIGH (K  (JV) ,IG  (JV) ) )  GO TO 9908
      ISC =ISC+1
      IF (ISC.LE.500)  GO TO 770
      WRITE  (OUTPUT,5020 )
 5020 FORMAT (1H ,SPLITTING BINS FULL')
      ICOUNT=ICOUNT+1
      IF (ICOUNT.GE.10)  GO TO 1068
      ISC =ISC-1
      GO TO 9908
  770 WT (JV) =WT (JV) *0.5
      IF (ISC.GT.25 )  GO TO 771
      TIMSTR (ISC ) = TME (JV)
      WTSTOR (ISC ) =WT (JV)
      XSTOR  (ISC ) =X  (JV)
      YSTOR  (ISC ) =Y  (JV)
      ZSTOR  (ISC ) =Z  (JV)
      KSTOR  (ISC ) =K  (JV)
      IGSTOR (ISC ) =IG (JV)
      IXSTOR (ISC ) =NBX  (JV)
      IYSTOR (ISC ) =NBY  (JV)
      IZSTOR (ISC ) =NBZ  (JV)
      SPLITB (ISC ) =JV
      GO TO 761
  771 SUBS1  (ISC ) = TME (JV)
      SUBS2  (ISC ) =WT (JV)
      SUBS3  (ISC ) =X  (JV)
      SUBS4  (ISC ) =Y  (JV)
      SUBS5  (ISC ) =Z  (JV)
      ISUBS1 (ISC ) =K  (JV)
      ISUBS2 (ISC ) =IG (JV)
      ISUBS3 (ISC ) =NBX  (JV)
      ISUBS4 (ISC ) =NBY  (JV)
      ISUBS5 (ISC ) =NBZ  (JV)
      SPLITB (ISC ) =JV
      GO TO 761
 9908 CONTINUE
```

〔原因分析〕

GO TO 761 によるインプリシトループがあるため、ベクトル化の対象外のループである。しかるにCPU時間測定をすると、FORTRAN77 とFORTRAN77 /VPとでは、後者の方が遅くなる。 (24.75 → 30.09秒)
これは、命令番号761 のIF文の真率が高いにもかかわらず、FORTRAN77 /VPでは、真率が低い場合の計算を計算するためと考えられる。

命令番号761 までの処理と次からの処理を2 つのDOループに分け、はじめのDOループをベクトル化する判定結果は、マスクとしてたくわえ、2 番目のループでこれを参照する。

再構成ループ

```
      LC¥=0
 * VOCL LOOP,NOVREC
V       DO 99910 IV =1,NXSEC
V       JV=XSECBA (IV)
V       IF (WT (JV) .LE.WTHIGH (KJ (JV) ,IG  (JV) ) ) THEN
V       ELSE
V       LC¥=LC¥+1
V       LIST¥ (LC¥) =JV
V       END IF
  99910 CONTINUE
      IF (LC¥.EQ.0 )  GO TO 99909

      IV=0
  99908 CONTINUE
      IV=IV+1
      JV=LIST¥ (IV)
    761 IF (WT (JV) .LE.WTHIGH (KJ (JV) ,IG  (JV) ) )  GO TO 9908
      ISC =ISC+1
      IF (ISC.LE.500)  GO TO 770
      WRITE  (OUTPUT,5020 )
   5020 FORMAT (1H ,SPLITTING BINS FULL')
      ICOUNT=ICOUNT+1
      IF (ICOUNT.GE.10)  GO TO 1068
      ISC =ISC-1
      GO TO 9908
    770 WT (JV) =WT (JV) *0.5
      IF (ISC.GT.25 )  GO TO 771
      TIMSTR (ISC ) = TME (JV)
      WTSTOR (ISC ) =WT (JV)
      XSTOR  (ISC ) =X  (JV)
      YSTOR  (ISC ) =Y  (JV)
      ZSTOR  (ISC ) =Z  (JV)
      KSTOR  (ISC ) =K  (JV)
      IGSTOR (ISC ) =IG (JV)
      IXSTOR (ISC ) =NBX  (JV)
      IYSTOR (ISC ) =NBY  (JV)
      IZSTOR (ISC ) =NBZ  (JV)
      SPLITB (ISC ) =JV
      GO TO 761
    771 SUBS1  (ISC ) = TME (JV)
      SUBS2  (ISC ) =WT (JV)
      SUBS3  (ISC ) =X  (JV)
      SUBS4  (ISC ) =Y  (JV)
      SUBS5  (ISC ) =Z  (JV)
      ISUBS1 (ISC ) =K  (JV)
      ISUBS2 (ISC ) =IG (JV)
      ISUBS3 (ISC ) =NBX  (JV)
      ISUBS4 (ISC ) =NBY  (JV)
      ISUBS5 (ISC ) =NBZ  (JV)
      SPLITB (ISC ) =JV
      GO TO 761
   9908 CONTINUE
      IF (IV.LT.LC¥)  GO TO 99908
  99909 CONTINUE
```

③散乱によるエネルギ群遷移処理のベクトル化 (9-5)

オリジナルループ

```
      DO 9907 IV=1,NSURV
      JV=SURVBA (IV)
      DO 800 IZ =1,NDS
  800 IF (R  (JV) .LE.
     .FSP  (IZ+IGRK  (JV) ) )  GO TO 810
      STOP
  810 IZZ  (JV) =IZ
      IG (JV) =IZZ  (JV) +IG (JV) - 1
 9907 CONTINUE
```

〔原因分析〕
DO 800は、平均ベクトルが3 と短く、ベクトル性能が出ない。

〔チューニング方法〕
DO 800とDO 9907 のループをいれかえる。
判定と計算を別々のループにする。
(判定：DO 9907, 計算：DO 79907)
ループアンローリングを行い、外側のループの繰り返しによる、ロードメスト3 を減らす。
ループアンローリングで判定できなかった場合通常の処理を行う。 (DO 69907)

再構成ループ

```
      DO 99907 IV =1,NSURV
      LIST¥ (IV) =SURVBA (IV)
99907 CONTINUE
      NSERV ¥=NSURV
      DO 800 IZ =1,NDS,7
      ISUM¥=0
*VOCL LOOP,NOVREC
*VOCL LOOP,VDOPT
V     DO 9907 IV=1,NSURV ¥
V       JV=LIST¥ (IV)
V       IGKR¥=IZ+IGKR  (JV)
V       R ¥=R  (JV)
V       IZD ¥=-1
V       IF (R ¥.LE.FSP  (IGKR¥+6) )  IZD¥=6
V       IF (R ¥.LE.FSP  (IGKR¥+5) )  IZD¥=5
V       IF (R ¥.LE.FSP  (IGKR¥+4) )  IZD¥=4
V       IF (R ¥.LE.FSP  (IGKR¥+3) )  IZD¥=3
V       IF (R ¥.LE.FSP  (IGKR¥+2) )  IZD¥=2
V       IF (R ¥.LE.FSP  (IGKR¥+1) )  IZD¥=1
V       IF (R ¥.LE.FSP  (IGKR¥  ) )  IZD¥=0
V       IF (IZD ¥.EQ.-1)  THEN
V         ISUM¥=ISUM¥+1
V         LIST¥ (ISUM¥) =JV
V       END IF
V 9907 CONTINUE
      IF (ISUM¥.EQ.0 )  GO TO 89907
      NSURV ¥=ISUM¥
      NDS ¥=IZ
  800 CONTINUE
      DO 90800 IZ =NDS ¥+1,NDS
      ISUM¥=0
*VOCL LOOP,SCALAR
      DO 69907 IV =1,NSURV ¥
        JV=LIST¥ (IV)
        IGKR¥=IZ+IGKR  (JV)
        R ¥=R  (JV)
        IZD ¥=-1
        IF (R ¥.LE.FSP  (IGKR¥) )  IZZ (JV) =IZ
        IF (IZD ¥.EQ.-1) THEN
          ISUM¥=ISUM¥+1
          LIST¥ (ISUM¥) =JV
        END IF
69907 CONTINUE
      IF (ISUM¥.EQ.0 )  GO TO 89907
      NSURV ¥=ISUM¥
90800 CONTINUE
89907 CONTINUE
*VOCL LOOP,NOVREC
      DO 79907 IV =1,NSURV
        JV=SURVBA (IV)
        IG (JV) =IZZ  (JV) + IG (JV) - 1
79907 CONTINUE
```

④幾何形状番号による分類処理のベクトル化 (3-2,6-2)

オリジナルループ

```
*VOCL LOOP,SCALAR
      DO 9010 IV =1,NCROS
      JV =CROSBA (IV)
      IF  (IGEO (JV) .LE.8.AND.IGEO (JV) .GE.1 )  GO TO 10
      WRITE (OUTPUT,5000 )  IGEO  (JV) ,K (JV) ,X (JV) ,Y (JV) ,Z (JV) ,
     +                      X1  (JV) ,Y1  (JV) ,Z1  (JV)
 5000 FORMAT  (19H ***** CROSS ERROR ,3X,2I5,3X,1P6E13.5)
      MFLAG  = MFLAG + 1
      IF  ( MFLAG.GT.10  ) STOP
   10 GO TO (1020,1280,1380,1020,1430,1440,1007,1500 ) ,IGEO  (JV)
 1020 J20=J20+1
      CRO20 (J20 ) =JV
      GO TO 9010
 1280 J280 =J280+1
      CRO280  (J280) =JV
      GO TO 9010
 1380 J380 =J380+1
      CRO380  (J380) =JV
      GO TO 9010
 1430 J430 =J430+1
      CRO430  (J430) =JV
      GO TO 9010
 1440 J440 =J440+1
      CRO440  (J440) =JV
      GO TO 9010
 1007 J7 =J7+1
      CRO7  (J7) =JV
      GO TO 9010
 1500 J500 =J500+1
      CRO500  (J500) =JV
 9010 CONTINUE
```

〔原因分析〕
命令番号10の計算型GOTO文は、ベクトル処理することができない。
STOP文を含んだ文は、ベクトル化されにくい。
WRITE 文は、ベクトル化できない。

再構成ループ

```
*VOCL LOOP,NOVREC
V       DO 9010 IV =1,NCROS
        JV =CROSBA (JV)
        IF  (IGEO (JV) .LE.8.AND.IGEO (JV) .GE.1 ) THEN
V       IF  (IGEO (JV) .EQ.1 .OR. IGEO  (JV) .EQ.4 ) THEN
V           J20=J20+1
V           CRO20 (J20 ) =JV
V       END IF
V       IF  (IGEO (JV) .EQ.2 ) THEN
V           J280 =J280+1
V           CRO280  (J280) =JV
V       END IF
V       IF  (IGEO (JV) .EQ.3 ) THEN
V           J380 =J380+1
V           CRO380  (J380) =JV
        END IF
V       IF  (IGEO (JV) .EQ.5 ) THEN
V           J430 =J430+1
V           CRO430  (J430) =JV
        END IF
V       IF  (IGEO (JV) .EQ.6 ) THEN
V           J440 =J440+1
V           CRO440  (J440) =JV
        END IF
V       IF  (IGEO (JV) .EQ.7 ) THEN
V           J7 =J7+1
V           CRO7  (J7) =JV
        END IF
V       IF  (IGEO (JV) .EQ.8 ) THEN
V           J500 =J500+1
V           CRO500  (J500) =JV
V       END IF
V       END IF
V 9010 CONTINUE
```

〔ベクトル化方法〕
WRITE 文をとる。
STOP文のついたIF文をとる。
計算型GOTO文をIF文に書きかえる。

⑤バンク変更計算のベクトル化

オリジナルループ

```
      IF ( NPATH.LT.0 )   NPATH =0
      IF (NARRAY.LT.0 )  NARRAY =0
      IF (NALBED.LT.0 )  NALBED =0
*  VOCL LOOP,SCALAR
      DO 9482 IV=1,NCROS2
      JV=CROSB2 (IV)
    IF   ( LCOR  (JV) )  THEN
      IF (K  (JV) .GT.KREFM )  THEN
         NALBED =NALBED+1
         ALBEDB  (NALBED) =JV
      ELSE
        IC (JV) =.FALSE.
        NPATH =NPATH+1
        PATHBA (NPATH ) =JV
      ENDIF
    ELSE
      IF (K  (JV) .LE.K2 (JV) )  THEN
        IC (JV) =.FALSE.
        NPATH =NPATH+1
        PATHBA (NPATH ) =JV
      ELSE
        IF   ( LSGUN  ) THEN
          NALBED=NALBED+1
          ALBEDB= (NALBED) =JV
        ELSE
          K  (JV) =K  (JV) -1
          NARRAY=NARRAY+1
          ARRAYB (NARRAY) =JV
        ENDIF
      ENDIF
    ENDIF
 9482 CONTINUE
```

再構成ループ

```
      IF ( NPATH.LT.0 )   NPATH =0
      IF (NARRAY.LT.0 )  NARRAY =0
      IF (NALBED.LT.0 )  NALBED =0
V     DO 19482 IV =1,NCROS2
V     MASK1 ¥ (IV) =.FALSE.
V     MASK2 ¥ (IV) =.FALSE.
V19482 CONTINUE


* VOCL LOOP,NOVREC
      DO 9482 IV=1,NCROS2
V     JV=CROSB2 (IV)
V     IF   ( LCOR  (JV) )  THEN
* VOCL STMT,IFT (0 )
V       IF (K  (JV) .GT.KREFM )  THEN
V         MASK1 ¥ (IV) =.TRUE.
V       ELSE
V         IC (JV) =.FALSE.
V         MASK2 ¥ (IV) =.TRUE.
V       ENDIF
V     ELSE
V       IF (K  (JV) .LE.K2 (JV) )  THEN
V         IC (JV) =.FALSE.
V         MASK2 ¥ (IV) =.TRUE.
V       ELSE
V         IF   ( LSGUN  ) THEN
V           MASK1 ¥ (IV) =.TRUE.
V         ELSE
V           K  (JV) =K  (JV) - 1
V           NARRAY=NARRAY+1
V           ARRAYB (NARRAY) =JV
V         ENDIF
V       ENDIF
V     ENDIF
V 9482 CONTINUE

*VOCL LOOP,NOVREC
V     DO 29482 IV =1,NCROS2
V     JV=CROSB2 (IV)
*VOCL STMT,IF (0 )
V     IF (MASK1 ¥ (IV) )  THEN
V       NALBED=NALBED+1
V       ALBEDB (NALBED) =JV
V     ENDIF
V     IF (MASK2 ¥ (IV) )  THEN
V       NPATH =NPATH+1
V       PATHBA (NPATH ) =JV
V     ENDIF
V29482 CONTINUE
```

〔原因分析〕
同じインデクスを同一ループで2回コンプレスしているため、ベクトル化できない。

〔チューニング方法〕
2回コンプレスが必要な場合、別々のループで1回づつコンプレスを行い、1回目のコンプレスでマスクを作り、2回目のコンプレス時にそれを利用する。

⑥乱数のベクトル化

オリジナルループ

```
S         DO 9910 IV=1,NSURV
S         JV=SURVBA (IV)
S    9910 R  (JV) =FLTRN  (0 )
```

再構成ループ

```
          IF (NLIM¥.LT.NFLT¥)  THEN
            NFLT¥=0
            CALL RANU2 (IX¥,FLT¥,NRMAX¥,ICON ¥)
          END IF
V         DO 9910 IV=1,NSURV
V         JV=SURVBA (IV)
V    9910 R  (JV) =FLT ¥ (IV+NFLT ¥)

          NFLT¥=NFLT¥+NSURV
```

〔原因分析〕
外部関数FLTRN は、ベクトル化対象外。
〔チューニング方法〕
同じ機能を持つ SSLⅡ／VPルーチンRANU2 に置きかえる。

オリジナルループ

```
*VOCL LOOP,SCALAR
      DO 9321 IV =1,NPATH
      JV =PATHBA (IV)
 9321 IF  (RPTH (JV) .LE.0.0 )  RPTH  (JV) = EXPRN (0 )
```

再構成ループ

```
       IF  (NLIM¥.LT.NEXP¥)  THEN
           NEXP ¥=0
           CALL RANU2  (IX¥,EXP¥,NRMAX¥,ICON ¥)
V          DO 19321 IV=1,NRMAX ¥
V            EXP¥ (IV) =-LOG (EXP ¥ (IV) )
V19321 CONTINUE
       END IF
*VOCL LOOP,NOVREC
V      DO 9321 IV =1,NPATH
V      JV =PATHBA (IV)
V 9321 IF  (RPATH  (JV) .LE.0.0 )  RPTH  (JV) = EXP¥ (IV+NEXP ¥)
       NEXP ¥=NEXP¥+NPATH
```

〔原因分析〕
EXPRN は、指数乱数を計算する外部関数である。この関数は、アセンブラで書かれており、ベクトル化することが出来ない。

〔ベクトル化方法〕
指数乱数は、一様乱数の対数をとることによって作る。
一様乱数は、 SSLⅡ／VPを用いてベクトル化処理する。
粒子一つ毎に計算しないで、一括して計算する。（上記の場合、10000 個毎）
計算した乱数は配列にたくわえ、使用する時は、これを呼び出す。

付録B　エネルギ群遷移処理のベクトル化

〔1〕オリジナルループ

```
      DO 9907 IV=1,NSURV
      JV=SURVBA (IV)
      DO 800 IZ =1,NDS
  800 IF (R  (JV) .LE.FSP  (IZ+IGKR  (JV) ) ) GO TO 810
      STOP
  810 IZZ  (JV) =IZ
      IG (JV) =IZZ  (JV) +IG  (JV) -1
 9907 CONTINUE
```

内側の平均ループ長は4,外側は32。NDS は100 のオーダのため，STOP文を外しそのままベクトル化しても速くならない。

〔2〕ループアンローリングを使用する方法

a．コーディング

```
      DO 99907 IV =1,NSURV
      LIST¥ (IV) =SURVBA (IV)
 99907 CONTINUE
      NSURV ¥=NSURV
      DO 800 IZ =1,NDS,2
        ISUM¥=0
*VOCL  LOOP,NOVREC
*VOCL  LOOP,VDOPT
      DO 9907 IV=1,NSURV ¥
        JV=LIST¥ (IV)
        IGKR¥=IZ+IGKR  (JV)
        R ¥=R  (JV)
        IZD ¥=-1
          IF (R ¥.LE.FSP  (IGKR¥+1) ) IZD ¥=1
          IF (R ¥.LE.FSP  (IGKR¥  ) ) IZD ¥=0
          IZZ  (JV) =IZ+IZD¥
        IF (IZD ¥.EQ.-1)  THEN
          ISUM¥=ISUM¥+1
          LIST¥ (ISUM¥) =JV
C         STOP
        END IF
 9907 CONTINUE
      IF (ISUM¥.EQ.0 ) GO TO 89907
      NSURV ¥=ISUM¥
  800 CONTINUE
89907 CONTINUE
      DO 79907 IV =1,NSURV
      JV=SURVBA (IV)
      IG (JV) =IZZ  (JV) +IG  (JV) -1
79907 CONTINUE
```

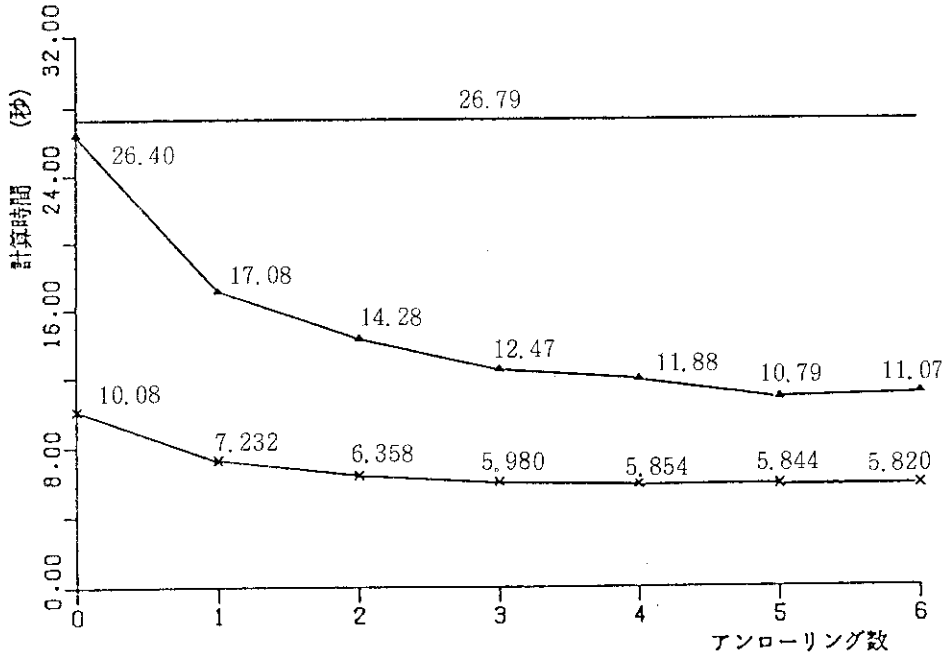
1回のループアンローリングを行っている部分

本コーディングは、STOPの処理が含まれていない。完全な形のものは、付録Aの⑤を参照

b．性能

・ループアンローリング版
FORTRAN77 /VP V10L30
VDOPT
・オリジナル版
FORTRAN77 V10L31

無印：オリジナル版CPU 時間
△印：ベクトル 版CPU 時間
×印：ベクトル 版VU 時間



エネルギ群探索処理（D09907）のループアンローリングによる性能
（6回のループアンローリングを行った）

付録C　時間測定法

時間測定は，FORTRAN77 /VPのライブラリ，CLOCKVを用いた。CLOCKVは従来のCPU 時間の他にVU時間をμsec 単位で測定することができ，個々の処理のベクトル化率を算出することができる。また測定では，ライブラリの呼び出し時間のオーバヘッドを除くため，

のように測定を行った。

```
CALL CLOCKV  (VU0,CPU0,2,2)
CALL CLOCKV  (VU1,CPU1,2,2)
        処理
CALL CLOCKV  (VU2,CPU2,2,2)
TCPU=TCPU+CPU2-CPU1-  (CPU1-CPU0 )
TVU  =TVU +VU2 -VU1
```

付録D　多重IF文の性能

カーネルPOSIT のスラブとシリンダ判定計算部分を取り出しベクトル化し，ドライバルーチンを設けて性能を測定した。

測定は，VP-100で FORTRAN77 (OPT3指定)，FORTRAN77 ／VP (VP (100) 指定) で行った。

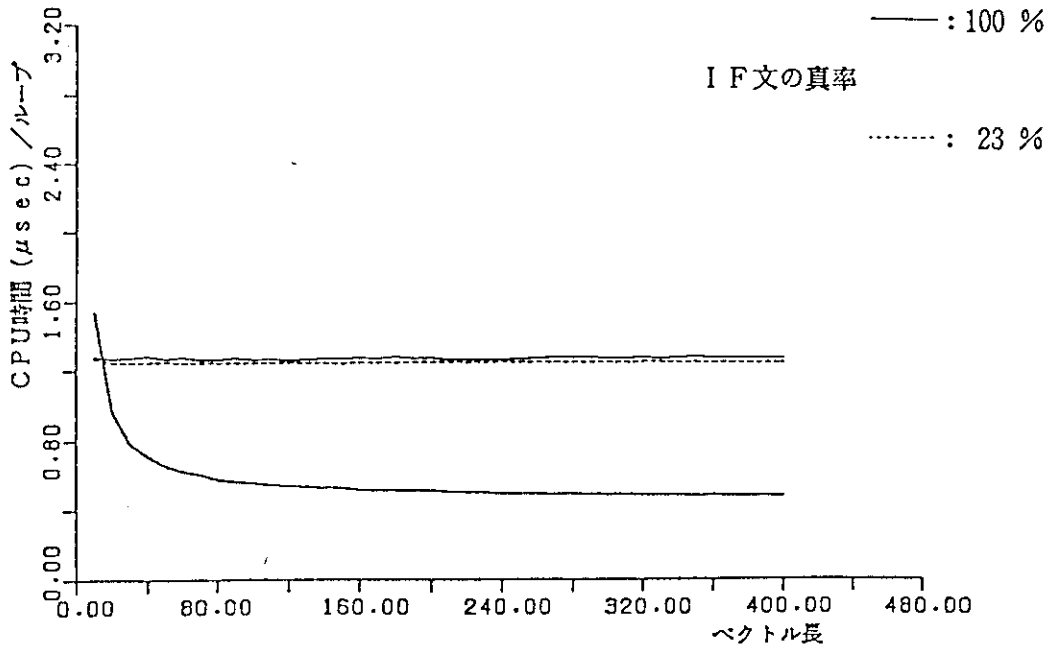
スカラ・ベクトルCPU 時間比の概略は次の通りである。

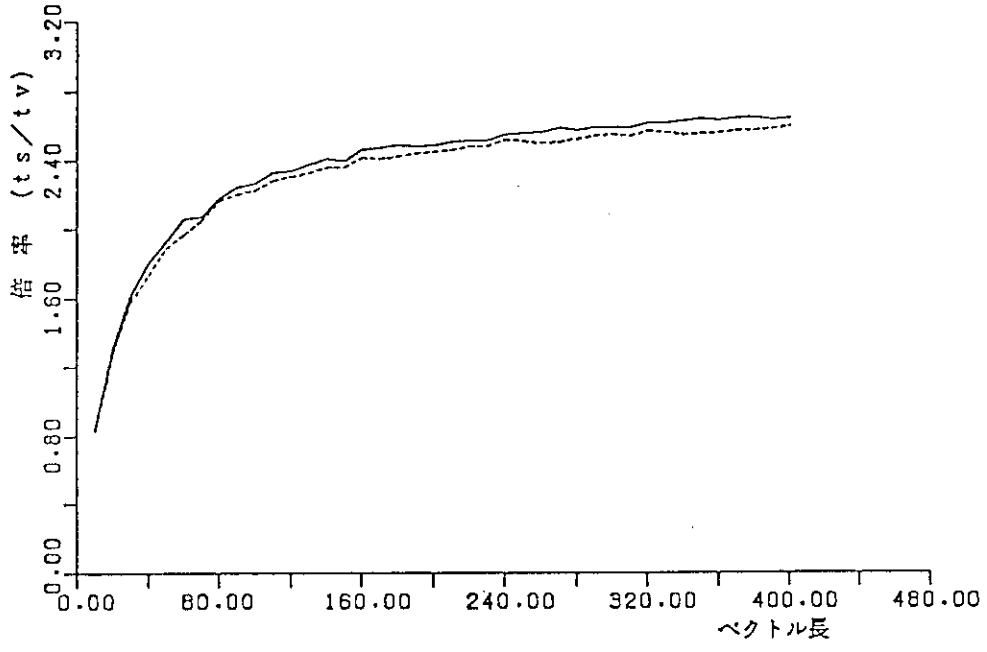
シリンダ判定計算：2.6 倍　(IF文真率100 %，ベクトル長400)

スラブ　判定計算：2.7 倍　(IF文真率100 %，ベクトル長400)

本付録では，〔1〕シリンダ判定計算部の性能，〔2〕スラブ判定計算部の性能，〔3〕ドライバ・テストループプログラムを示す。

〔1〕シリンダ判定計算部の性能



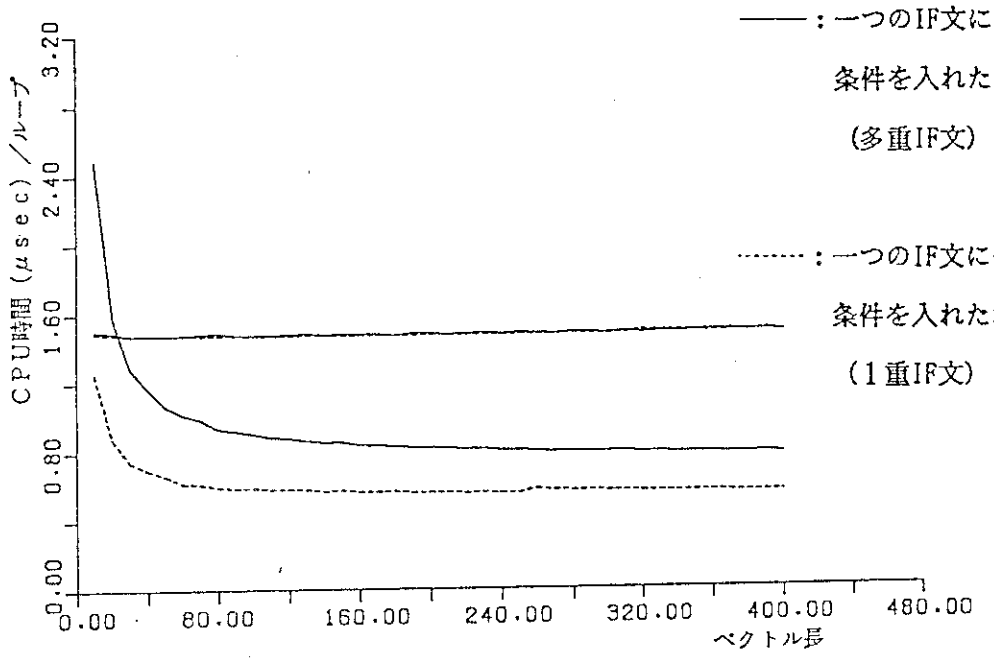


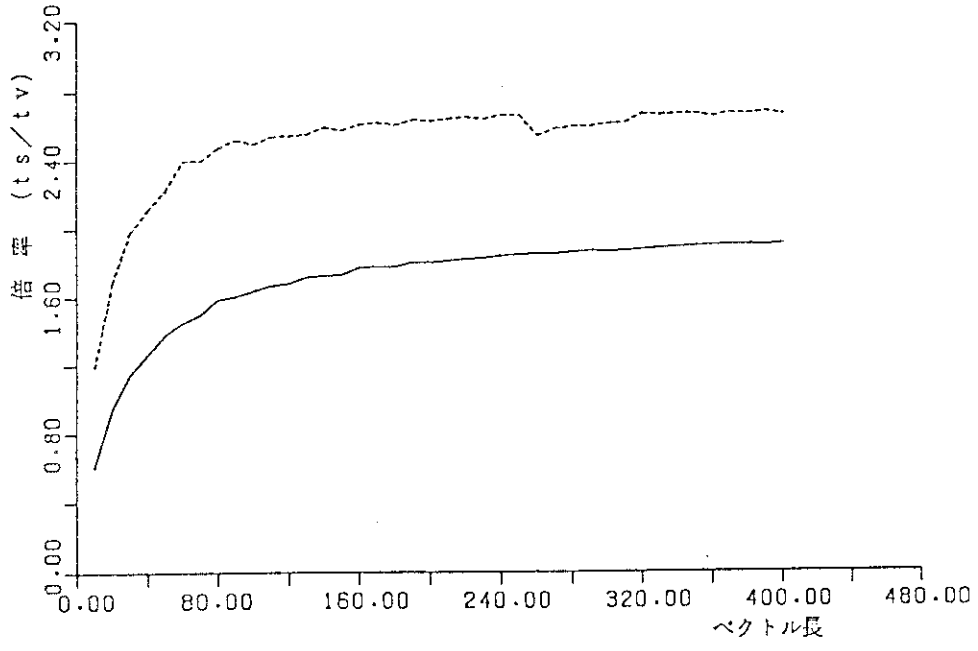
シリンダ判定計算部のベクトル長に対する1エレメント当たりの性能

ベクトル長400,真率　23%の時の性能：1.27μ秒（スカラ）,0.481μ秒（ベクトル）

ベクトル長400,真率 100%の時の性能：1.24μ秒（スカラ）,0.479μ秒（ベクトル）

〔2〕スラブ判定計算部の性能





スラブ判定計算部のベクトル長に対する1エレメント当たりの性能

1重IF文のベクトル長400 時の性能：1.48μ秒（スカラ），0.554μ秒（ベクトル）

多重IF文のベクトル長400 時の性能：1.48μ秒（スカラ），0.776μ秒（ベクトル）

（尚，再構成2では多重IF文を用いてベクトル化した。）

〔3〕ドライバ・テストループプログラム
a. シリンダ判定計算ドライバ

```
      PARAMETER  (NPSMX ¥=400,ITERM ¥=1000,PERTRU =100.)
      COMMON/CNTRL /ITERM,NPSMAX
      COMMON/TIMER /CPU ¥, VU¥
      LOGICAL *4     MASK¥
      INTEGER *4     POSBAN (NPSMX ¥) ,K (NPSMX ¥)
      REAL*4         X1 (NPSMX ) ,Y1  (NPSMX ) ,Z1  (NPSMX )
      REAL*4         XX (NPSMX ¥,6)
      REAL*8         CPU ¥, VU¥
      REAL*8         CPUSEC,VUSEC
      CHARACTER *45 MIDASI
C   SETTING INITIAL VALUE
      DATA MIDASI /'   V-LNGTH   CPU-TIME (SEC )   VU-TIME  (SEC ) ' /
      ITERM =ITERM ¥
      NPSMAX=NPSMX ¥
C
      DO 4000 I =1,NPSMAX
        POSBAN (I ) =I
        K      (I ) =I
        X1     (I ) =0.
        Y1     (I ) =0.
        Z1     (I ) =0.
 4000 CONTINUE
      DO 4010 I =1,NPSMAX
        XX (K  (I ) ,1) =0.
        XX (K  (I ) ,2) =0.
        XX (K  (I ) ,3) =0.
        XX (K  (I ) ,5) =0.
        XX (K  (I ) ,6) =0.
 4010 CONTINUE
C
      WRITE  (6,*) 'TRUE RATIO ',PERTRU,'%'
      WRITE  (6,6000)  MIDASI
      NPOS=0
      DO 4100 WHILE  (NPOS.LT.NPSMAX)
        NPOS  =NPOS+10
        CPU ¥=0.D0
        VP¥   =0.D0
        ITRN  =  (0.01*PERTRU)  *REAL (NPOS) + 0.5
        DO 4110 I =1,NPOS
          IF (I.LE.ITRN )  THEN
            XX (K  (I ) ,4) =0.
          ELSE
            XX (K  (I ) ,4) =-1.
          END IF
 4110   CONTINUE
        CALL PST23 (MASK¥,POSBAN,K,X1,Y1,Z1,XX,POS )
        CPUSEC=CPU ¥*1.D-6
        VUSEC =VU¥  *1.D-6
        WRITE  ( 6,6100 ) NPOS,CPUSEC,VUSEC
        WRITE  (66,6200 ) NPOS,CPU¥,VU ¥
 4100 CONTINUE
C
 6000 FORMAT (A45 )
 6100 FORMAT (/,5X,I4.4,6X,F9.6,6X,F9.6)
 6200 FORMAT (I4.4,1X,F16.10,1X,F16.10)
      STOP
      END
```

b．シリンダ判定計算テストループ

```
      SUBROUTINE PST23 (MASK¥,POSBAN,K,X1,Y1,Z1,XX,POS )
      COMMON/CNTRL /ITERM,NPSMAX
      COMMON/TIMER /CPU ¥, VU¥
      LOGICAL *4     MASK¥
      INTEGER *4     POSBAN (NPSMX ¥) ,K (NPSMX ¥)
      REAL*4         X1 (NPSMX ) ,Y1  (NPSMX ) ,Z1  (NPSMX )
      REAL*4         XX (NPSMX,6 )
      REAL*8         CPU ¥, CPU0¥,CPU1 ¥,CPU2 ¥,CPUINT
      REAL*8         VU¥,VU0¥,VU1¥,VU2¥,VUINT
C
      DO 4100 LOOP=1,ITERM
        MASK¥=.FALSE.
        CALL CLOCKV  (VU0 ¥,CPU0 ¥,2,2)
        CALL CLOCKV  (VU1 ¥,CPU1 ¥,2,2)
        DO 4000 IV=1,NPOS
          JV=POSBAN (IV)
          RSQ =X1 (JV) *X1 (JV) + Y1 (JV) *Y1 (JV)
          IF (Z1 (JV) .LE.XX (K  (JV) ,2) )  THEN
            IF (Z1 (JV) .GE.XX (K  (JV) ,3) )  THEN
              IF (RSQ .LE.XX (K  (JV) ,4) )  THEN
                MASK¥=.TRUE.
              END IF
            END IF
          END IF
 4000   CONTINUE
        CALL CLOCKV  (VU2 ¥,CPU2 ¥,2,2)
        CPUINT=CPU1¥-CPU0 ¥
        VUINT =VU1 ¥-VU0  ¥
        CPU ¥=CPU ¥+  ( CPU2 ¥- CPU1¥) - CPUINT
        VU  ¥=VU  ¥+  ( VU2¥  - VU1 ¥)
 4100 CONTINUE
      CPU ¥=CPU ¥/REAL (ITERM )
      VU¥  =VU¥  /REAL (ITERM )
      RETURN
      END
```

c．スラブ判定計算ドライバ

```
      PARAMETER  (NPSMX ¥=400,ITERM ¥=1000)
      COMMON／CNTRL ／ITERM,NPSMAX
      COMMON／TIMER ／CPU ¥, VU¥
      LOGICAL *4     MASK¥
      INTEGER *4     POSBAN (NPSMX ) ,K (NPSMX )
      REAL*4         X1 (NPSMX ) ,Y1  (NPSMX ) ,Z1  (NPSMX )
      REAL*4         XX (NPSMX ¥,6)
      REAL*8         CPU ¥, VU¥
      REAL*8         CPUSEC,VUSEC
      CHARACTER *45 `MIDASI
C   SETTING INITIAL VALUE
      DATA MIDASI ／'   V-LNGTH   CPU-TIME (SEC )   VU-TIME  (SEC ) ' ／
      ITERM =ITERM ¥
      NPSMAX=NPSMX ¥
C
      DO 4000 I =1,NPSMAX
        POSBAN (I ) =I
        K      (I ) =I
        X1     (I ) =0.
        Y1     (I ) =0.
        Z1     (I ) =0.
 4000 CONTINUE
      DO 4010 I =1,NPSMAX
        XX (K  (I ) ,1) =0.
        XX (K  (I ) ,2) =0.
        XX (K  (I ) ,3) =0.
        XX (K  (I ) ,4) =0.
        XX (K  (I ) ,5) =0.
        XX (K  (I ) ,6) =0.
 4010 CONTINUE
C
      WRITE  (6,6000)   MIDASI
      NPOS=0
      DO 4100 WHILE  (NPOS.LT.NPSMAX)
        NPOS  =NPOS+10
        CPU ¥=0.D0
        VP¥   =0.D0
      CALL PSTIT (MASK¥,POSBAN,K,X1,Y1,Z1,XX,POS )
      CPUSEC=CPU ¥*1.D-6
      VUSEC =VU¥   *1.D-6
      WRITE  ( 6,6100 ) NPOS,CPUSEC,VUSEC
      WRITE  (66,6200 ) NPOS,CPU¥,VU ¥
 4100 CONTINUE
C
 6000 FORMAT (A45 )
 6100 FORMAT (／,5X,I4.4,6X,F9.6,6X,F9.6)
 6200 FORMAT (I4.4,1X,F16.10,1X,F16.10)
      STOP
      END
```

d．スラブ判定計算テストループ（多重ＩＦ文）

```
      SUBROUTINE PSTIT (MASK¥,POSBAN,K,X1,Y1,Z1,XX,POS )
      COMMON/CNTRL /ITERM,NPSMAX
      COMMON/TIMER /CPU ¥, VU¥
      LOGICAL *4     MASK¥
      INTEGER *4     POSBAN (NPSMX ) ,K (NPSMX )
      REAL*4         X1 (NPSMX ) ,Y1  (NPSMX ) ,Z1  (NPSMX )
      REAL*4         XX (NPSMX,NPSMAX)
      REAL*8         CPU ¥, CPU0¥,CPU1 ¥,CPU2 ¥,CPUINT
      REAL*8         VU¥,VU0¥,VU1¥,VU2¥,VUINT
C
      DO 4100 LOOP=1,ITERM
        MASK¥=.FALSE.
        CALL CLOCKV  (VU0 ¥,CPU0 ¥,2,2)
        CALL CLOCKV  (VU1 ¥,CPU1 ¥,2,2)
        DO 4000 IV=1,NPOS
          JV=POSBAN (IV)
          IF (X1 (JV) .LE.XX (K  (JV) ,1) )  THEN
            IF (X1 (JV) .LE.XX (K  (JV) ,2) )  THEN
              IF (Y1 (JV) .LE.XX (K  (JV) ,3) )  THEN
                IF (Y1 (JV) .LE.XX (K  (JV) ,4) )  THEN
                  IF (Z1 (JV) .LE.XX (K  (JV) ,5) )  THEN
                    IF (Z1 (JV) .LE.XX (K  (JV) ,6) )  THEN
                      MASK¥=.TRUE.
                    END IF
                  END IF
                END IF
              END IF
            END IF
          END IF
 4000   CONTINUE
        CALL CLOCKV  (VU2 ¥,CPU2 ¥,2,2)
        CPUINT=CPU1¥-CPU0 ¥
        VUINT =VU1 ¥-VU0  ¥
        CPU ¥=CPU ¥+  ( CPU2 ¥- CPU1¥) - CPUINT
        VU  ¥=VU  ¥+  ( VU2¥  - VU1 ¥)
 4100 CONTINUE
      CPU ¥=CPU ¥/REAL (ITERM )
      VU¥  =VU¥  /REAL (ITERM )
      RETURN
      END
```

e. スラブ判定計算テストループ（1重ＩＦ文）

```
      SUBROUTINE PSTIT (MASK¥,POSBAN,K,X1,Y1,Z1,XX,POS )
      COMMON/CNTRL /ITERM,NPSMAX
      COMMON/TIMER /CPU ¥, VU¥
      LOGICAL *4     MASK¥
      INTEGER *4     POSBAN (NPSMX ) ,K (NPSMX )
      REAL*4         X1 (NPSMX ) ,Y1  (NPSMX ) ,Z1  (NPSMX )
      REAL*4         XX (NPSMX,NPSMAX)
      REAL*8         CPU ¥, CPU0¥,CPU1 ¥,CPU2 ¥,CPUINT
      REAL*8         VU¥,VU0¥,VU1¥,VU2¥,VUINT
C
      DO 4100 LOOP=1,ITERM
        MASK¥=.FALSE.
        CALL CLOCKV  (VU0 ¥,CPU0 ¥,2,2)
        CALL CLOCKV  (VU1 ¥,CPU1 ¥,2,2)
        DO 4000 IV=1,NPOS
          JV=POSBAN (IV)
          IF ( (X1 (JV) .LE.XX (K  (JV) ,1) )  .AND.
     .         (X1 (JV) .LE.XX (K  (JV) ,2) )  .AND.
     .         (Y1 (JV) .LE.XX (K  (JV) ,3) )  .AND.
     .         (Y1 (JV) .LE.XX (K  (JV) ,4) )  .AND.
     .         (Z1 (JV) .LE.XX (K  (JV) ,5) )  .AND.
     .         (Z1 (JV) .LE.XX (K  (JV) ,6) ) )  THEN
              MASK¥=.TRUE.
          END IF
 4000   CONTINUE
        CALL CLOCKV  (VU2 ¥,CPU2 ¥,2,2)
        CPUINT=CPU1¥-CPU0 ¥
        VUINT =VU1 ¥-VU0  ¥
        CPU ¥=CPU ¥+  ( CPU2 ¥- CPU1¥) - CPUINT
        VU  ¥=VU  ¥+  ( VU2¥  - VU1 ¥)
 4100 CONTINUE
      CPU ¥=CPU ¥/REAL (ITERM )
      VU¥  =VU¥  /REAL (ITERM )
      RETURN
      END
```