JAERI-M
85-143

原子力コードにおける数値解法とそのベクトル化

1985年9月

徳永　康男*・原田　裕夫・石黒美佐子

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　いばらき印刷㈱

JAERI－M 85－143

原子力コードにおける数値解法とそのベクトル化

日本原子力研究所東海研究所計算センター
徳永康男*・原田裕夫・石黒美佐子

（1985年8月19日受理）

ベクトル計算機において原子力コードを高速に実行するためには，ベクトル計算機の特性に合った数値解法の選択とベクトル計算機の特性を生かすプログラミング技術が必要である。本報告は，原研で使用されている代表的な原子力コードについてその計算内容を分析し，そこで使われている数値解法をベクトル化の観点から解説したものである。原子力コードのベクトル化方法を系統的に示し，合わせてベクトル化の事例を各数値解法毎に示した。

* 外来研究員（富士通）

JAERI-M 85-143

Vectorization of Nuclear Codes and Numerical Methods

Yasuo TOKUNAGA* , Hiroo HARADA and Misako ISHIGURO

Computing Center, Tokai Research Establishment, JAERI

(Received August 19, 1985)

In order to realize the fast execution of a nuclear code using a vector-processing computer, it is necessary to select an appropriate numerical method and to use efficient programming techniques for this computer. In this report, first the details of nuclear codes frequently used in JAERI are analyzed, then the numerical methods employed there are discussed from the viewpoint of vectorization. The vectorization method of a nuclear code is systematically described, and examples of vectorizations are shown for individual numerical methods.

Keywords; Computer Codes, Nuclear Codes, Numerical Methods, Vectorization, Supercomputer, VP-100

---

* On leave from Fujitsu Limited.

目　　次

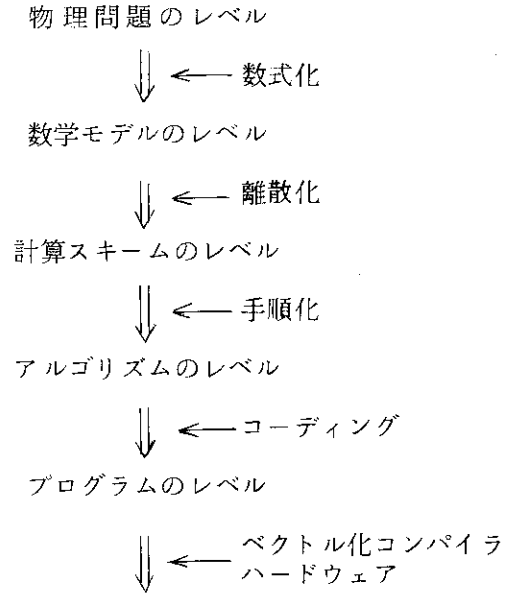
CONTENTS

1. はじめに

CRAY－1に代表されるベクトル演算型スーパーコンピュータ（以下ベクトル計算機と呼ぶ）は近年盛んに科学技術計算に使用されている。このベクトル計算機を効率良く使用し，プログラムを高速に実行するには，ベクトル計算機向きのプログラミングや数値解法が不可欠である。しかし，ベクトル計算機の歴史がまだ浅いため，原子力の分野における現存の計算プログラム（以下原子力コードと呼ぶ）はその多くが従来のタイプの計算機（以下スカラ計算機または逐次型計算機と呼ぶ）用に作られていることから，これらの原子力コードをベクトル計算機用に書き換える必要がある。またこれらの原子力コードに用いられている数値解法もベクトル計算向きでないことが多い。ベクトル計算向きの数値解法に関しては既に多くの報告がある。しかし実際のプログラミングでのそれらの具体的な使用方法やベクトル計算に対する効果は良く調べられているとは言えず，従ってこれらについてまとめた総合的な報告が必要となっている。

計算センターにおいては，59年末のFACOM VP－100の導入に対応して，ベクトル計算機向きの数値解法の研究やプログラミング技術の開発を行ってきた。また，これらの研究に基く種々の原子力コードのベクトル計算機向けの書き換え（以下これをプログラムのベクトル化と呼ぶ）に関する総合報告が浅井[1]並びに石黒[2]により提出されている。更にプログラミング上の細かな技術については奈良岡他[3]によって与えられている。本報告は，これらの報告を踏まえ，以下に示す通りプログラムのベクトル化に関する統一的な観点から，最新の成果を取り入れてまとめたものである。

ひとつの原子力コードのベクトル化を行う場合，対象とする物理問題をベクトル計算機向きのプログラムで表現するまでには以下のような様々なレベルで考慮すべき点がある。



本報告では物理問題，数学問題のレベルに関しては原子力コードの構造分析を行うにとどめ，主として計算スキーム，アルゴリズム，プログラムのレベルにおけるベクトル計算向けの工夫について述べる。

第二章では原研で使用されている代表的原子力コードについて，用いられている基礎方程式やコードの構造，使用されている数値解法，ベクトル化方法などを概説する。また，これまでに原研でベクトル化された原子力コードについて，M－380 と VP－100 による実行時間の比較を一覧表の形で示し，その中のいくつかの原子力コードについては計算ブロック単位での CPU時間の分布をベクトル化の前後で比較し図示した。これらは原子力コードのベクトル化効果を類推する上で参考になるであろう。

第三章では，原子力コードにおいてよく使用されている数値計算法と，そのベクトル化方法を解説する。最近注目されている共役傾斜法（Conjugate Gradient method，CG 法と略称）も解説している。

第四章ではベクトル化プログラミングの基本事項と，効果的なベクトル化プログラミング事例について示す。

2. 原子力コードの構造・計算内容とベクトル化方法

数値解法とそのベクトル化を考えるに当って，実際に原研で使用されているプログラムがどのように構成され，どの部分がベクトル化されるか，あるいはどの部分がベクトル化の障害になっているかを分析することは，ベクトル化作業に極めて有益であろう。ここでは各分野の代表的なプログラムとして6つの原子力コードを選び，その構成とプログラムの基礎となっている物理方程式を示す。更にプログラムをベクトル化した際に最も時間を費やした作業について簡潔に述べることにする。

2.1 原研における代表的な原子力コード

2.1.1 中性子拡散計算（CITATION）

原研において，JRR－3の改造や高速臨界実験装置の臨界計算などによく使用されている3次元多群中性子拡散コードCITATION[4]を例に，中性子拡散計算のベクトル化について述べる。

中性子拡散方程式は，有限差分法により離散化された空間メッシュとエネルギー群について，実効増倍係数k_eを固有値，中性子束$\boldsymbol{\Phi}$を固有ベクトルとした次のような固有値問題として解かれる。

$$A\boldsymbol{\Phi} = \frac{1}{k_e} F\boldsymbol{\Phi} \qquad (2.1)$$

空間のメッシュ数をN，エネルギー群数をGとしたとき，A，Fは，G・N次の行列となる。Aは，中性子の輸送，散乱，吸収に係わる係数であり，Fは，中性子源に係わる係数である。

(2.1)式は反復計算法によって，次のようになる。

$$\boldsymbol{\Phi}^{(n)} = \frac{1}{k_e^{(n)}} A^{-1} F \boldsymbol{\Phi}^{(n-1)} \qquad (2.2)$$

〔n：外側反復回数〕

(2.2)式において，$\boldsymbol{\Phi}^{(n)}$や$k_e^{(n)}$を求める過程は，内側・外側（inner－outer）の2重反復計算によって行われる。その構成をFig. 2.1に示す。

内側反復計算は，エネルギー群$\ell = 1, 2, \cdots\cdots, G$（$\ell = 1$は最も高いエネルギー群）の順に各エネルギー群のN元連立方程式を解く問題になる。反復計算では，ある空間のメッシュ点（x_i, y_j, z_k）における中性子束の値は，その点に隣接する，同一エネルギー群に属する点を使って計算される。3次元問題では，次の7点階差式となる。

$$\varphi_{i,j,k}^{(m)} = a + b\,\varphi_{i-1,j,k}^{(m)} + c\,\varphi_{i,j-1,k}^{(m)} + d\,\varphi_{i,j,k-1}^{(m)}$$
$$+ e\,\varphi_{i+1,j,k}^{(m-1)} + f\,\varphi_{i,j+1,k}^{(m-1)} + g\,\varphi_{i,j,k+1}^{(m-1)} \qquad (2.3)$$

a～g は(i,j,k)に依存した定数で，mは内側反復回数を表わす。

ベクトル化の観点からは，計算時間が集中している，この内側反復計算において中性子束を求める部分を高速化することが重要である。

差分法を用いた3次元の中性子拡散計算において，エネルギー群に関するループは逐次的に計算を進めなければならない。3次元の空間メッシュに関するループは，あるメッシュ点の計算に隣接するまわりの6点が必要ではあるが，各メッシュ点の計算する順序に関しては制限されない。従って，CITATIONコードの場合にはオリジナルコードで内側反復計算で用いられていたSLOR法（Successive Line Over－Relaxation method，逐次線過大緩和法）をSOR法（Successive Over－Relaxation method，逐次過大緩和法）に変更し，更に次章の3.1で述べる数値計算上の工夫により，空間メッシュに関してベクトル計算が適用できるように改良した。

CITATIONコードのベクトル化については，文献(5)に述べられている。

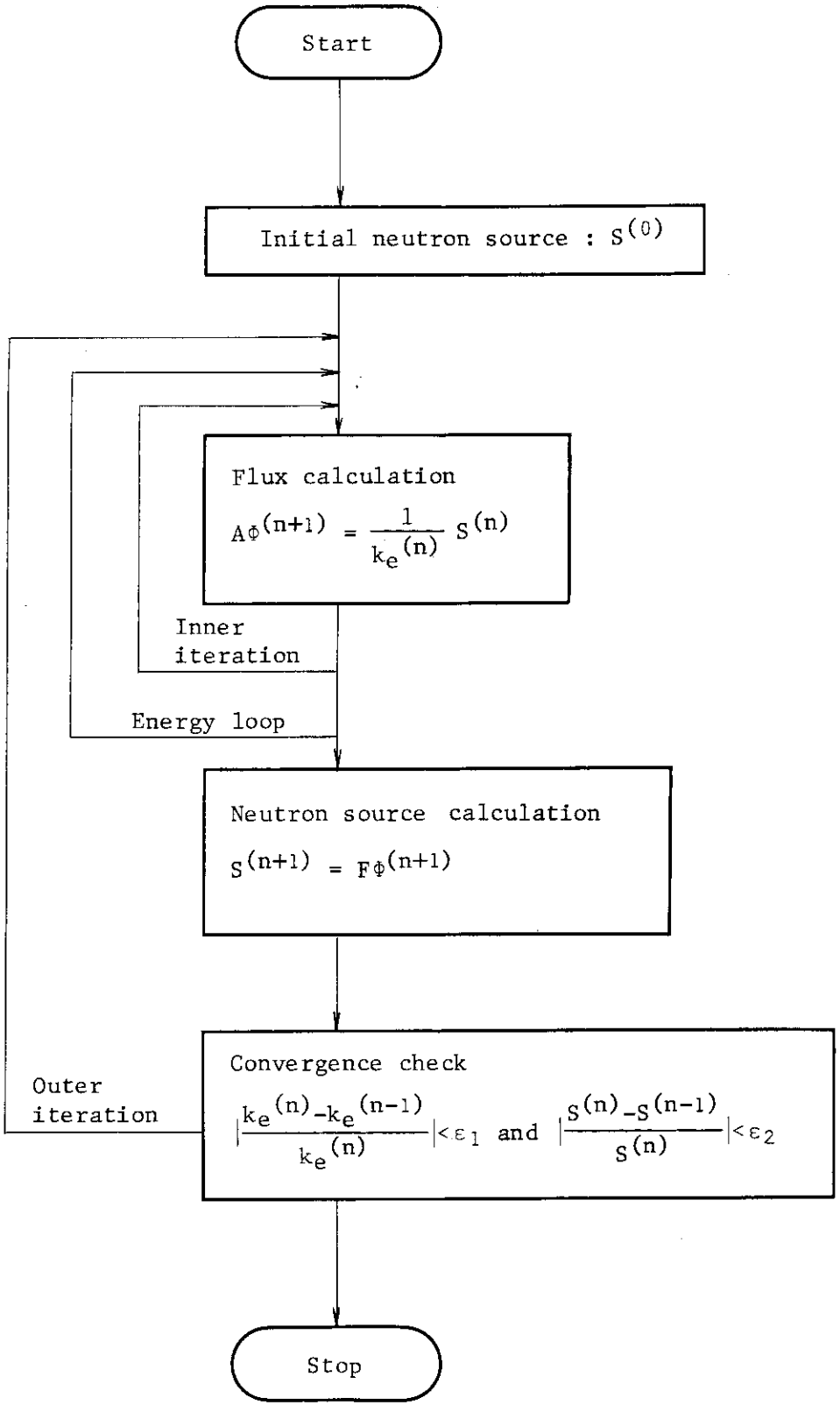


Fig. 2.1 Flow Diagram of the Neutron Diffusion Code CITATION.

2.1.2　中性子輸送計算

（DOT 3.5）

原研において，核融合炉物理用中性子源施設や核融合炉設計の遮蔽計算などによく使用されているDOT 3.5コード[6]を例に，中性子輸送計算におけるコードのベクトル化について述べる。なお本節中で示す式の詳しい導出の仕方については文献(6)を参照されたい。

DOT 3.5コードの構成をFig. 2.2に示す。

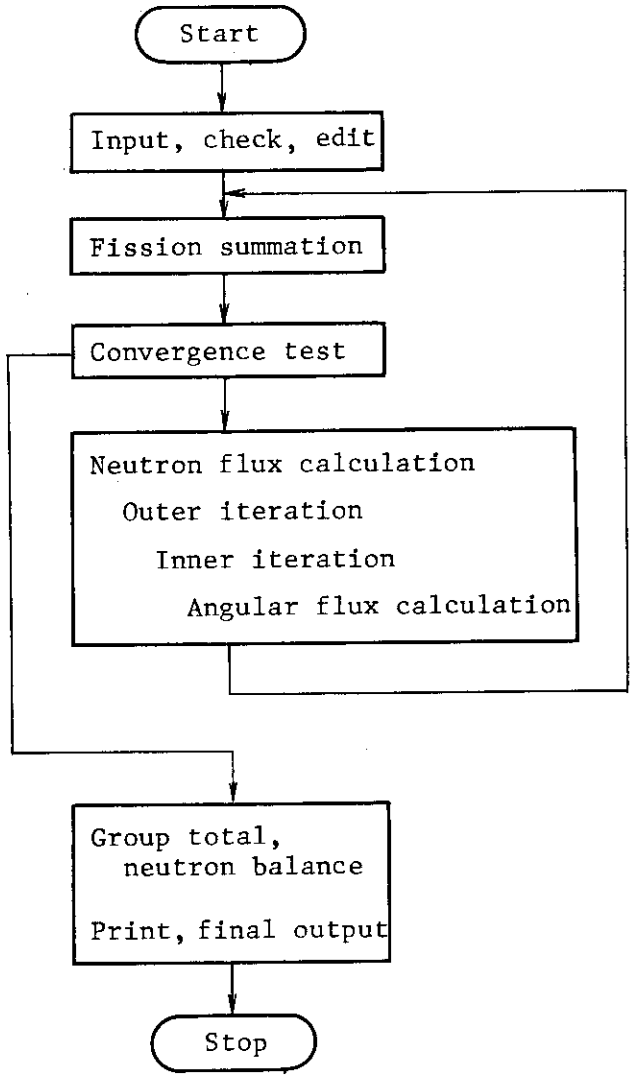


Fig. 2.2　Flow Diagram of the Neutron Transport Code DOT3.5.

シリンダ形状に対する座標系をFig. 2.3に示す。解かれるべき基礎方程式は

$$\frac{d\varphi(r,z,\theta,\eta,\psi,E)}{ds}+\Sigma^{T}(r,z,\theta,E)\varphi(r,z,\theta,\eta,\psi,E)$$

$$=S(r,z,\theta,\eta,\psi,E)+\int_{-1}^{+1}\int_{0}^{2\pi}\int_{0}^{\infty}\Sigma^{S}(r,z,\theta;E',\Omega'\rightarrow E,\Omega)\varphi(r,z,\theta,\eta',\psi',E')dE'd\psi'd\eta' \qquad (2.4)$$

ここで，φは中性子束，sは空間及び角度方向を表わし，r, z, θ, ηで与えられる。Sは中性子源を表わす。

離散化により，最終的に次の式が算出される。

$$\varphi_{G,I,J,D}=(|\bar{\mu}_D|2\pi\bar{r}_I\Delta z_J\left\{\begin{matrix}\text{or}\\ \end{matrix}\begin{matrix}\varphi_{G,i+1,J,D}\\ \varphi_{G,i,J,D}\end{matrix}\right\}+|\bar{\eta}_D|2\pi\bar{r}_I\Delta r_I\left\{\begin{matrix}\text{or}\\ \end{matrix}\begin{matrix}\varphi_{G,I,j+1,D}\\ \varphi_{G,I,J,D}\end{matrix}\right\}$$

$$+\frac{\Delta z_J}{w_D}\bar{r}_N\varphi_{G,I,J,n,K}+\frac{1}{2}V_{I,J}S'_{G,I,J,D})/(|\bar{\mu}_D|\Delta z_J\,2\pi\bar{r}_I+|\bar{\eta}_D|2\pi\bar{r}_I\Delta r_I$$

$$+\frac{\Delta z_J}{w_D}\bar{r}_N+\frac{1}{2}V_{I,J}\Sigma^{T}_{G}). \qquad (2.5)$$

ここで，I, J, Dは各々$(i-\frac{1}{2}, i+\frac{1}{2})$，$(j-\frac{1}{2}, j+\frac{1}{2})$及び$(n-\frac{1}{2}, n+\frac{1}{2}; K)$の区間を示す。また，散乱角の向きによって，(2.5)式の上段または下段の計算式が使用される。

(2.5)式から$\varphi_{G,I,J,D}$を求め，次の(2.6)式からメッシュ(i, j, n, g)上のφを計算する。

$$\left\{\begin{matrix}\text{or}\\ \end{matrix}\begin{matrix}\varphi_{G,i,J,D}\\ \varphi_{G,i+1,J,D}\end{matrix}\right\}=2\varphi_{G,I,J,D}-\left\{\begin{matrix}\text{or}\\ \end{matrix}\begin{matrix}\varphi_{G,i+1,J,D}\\ \varphi_{G,i,J,D}\end{matrix}\right\}$$

$$\left\{\begin{matrix}\text{or}\\ \end{matrix}\begin{matrix}\varphi_{G,I,j,D}\\ \varphi_{G,I,j+1,D}\end{matrix}\right\}=2\varphi_{G,I,J,D}-\left\{\begin{matrix}\varphi_{G,I,j+1,D}\\ \varphi_{G,I,J,D}\end{matrix}\right\} \qquad (2.6)$$

$$\varphi_{G,I,J,n+1,K}=2\varphi_{G,I,J,D}-\varphi_{G,I,J,n,K}$$

(2.6)式の場合，得られる計算式が，φ_{i+1}の計算に対して，φ_iのみを参照して計算を進める形になり，並列計算には再帰演算となり不向きである。

また，反復計算において，odd－even法などを用いる方法なども数値実験の結果では，φ_{i+1}の計算に，φ_iのみを参照している点で，収束性がよくない。

従って，現ベクトル化版では係数計算や内積計算の部分に対してベクトル計算を行い，再帰演算（漸化式）の部分は逐次計算で行っている。

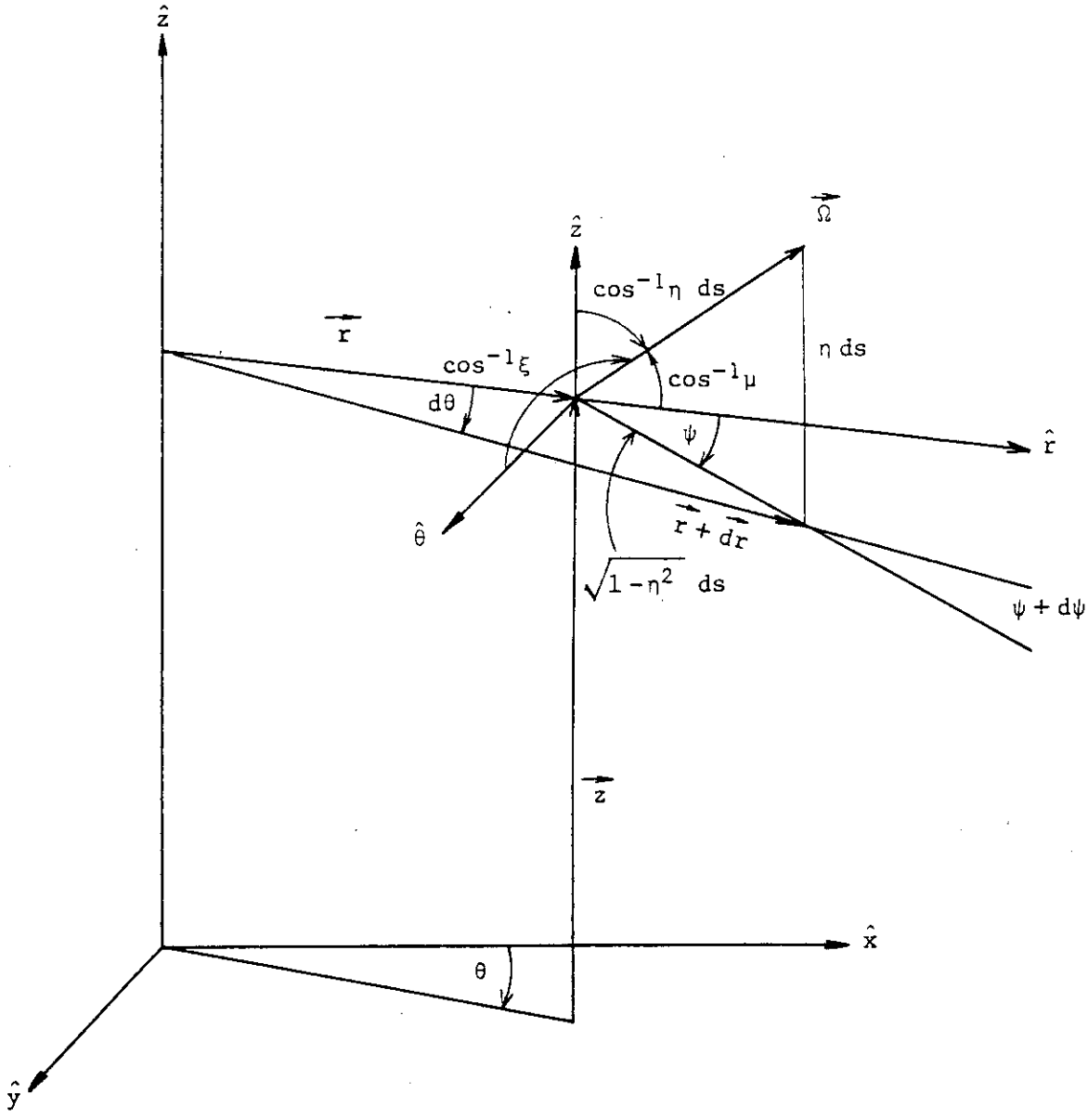


Fig. 2.3 The Coordinate System for Cylindrical Geometry.

2.1.3　非線型MHD不安定解析計算

（AEOLUS）

トカマクプラズマにおける非線型MHD現象に関する計算は非常に盛んで，原研においても計算機使用時間の順位の上位を占めている。現在は以下に示す2つのコード群で計算が頻繁に行われているが，これらのコードは原研におけるTRITON計画[7]の一環として開発されているものである。

AEOLUS－R：抵抗性MHD不安定性解析コード群

AEOLUS－E：自由境界トカマクプラズマの非線型不安定性解析コード群

これらのプログラムではまず以下に示す基礎MHD方程式から出発する。

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot \vec{\rho} v = 0 \qquad (2.7)$$

$$\rho \left(\frac{\partial}{\partial t} + \vec{v} \cdot \vec{\nabla}\right) \vec{v} = -\vec{\nabla} P + \vec{j} \times \vec{B} \qquad (2.8)$$

$$\frac{d}{dt} \left(P \rho^{-r}\right) = 0 \qquad (2.9)$$

$$\frac{\partial \vec{B}}{\partial t} = \vec{\nabla} \times (\vec{v} \times \vec{B} - \eta \vec{j}) \qquad (2.10)$$

$$\vec{j} = \vec{\nabla} \times \vec{B} \qquad (2.11)$$

$$\vec{\nabla} \cdot \vec{B} = 0. \qquad (2.12)$$

実際の計算に当っては円柱座標（R, Φ, Z）あるいはヘリカル座標を用いて，式（2.7）～（2.17）を書き下して簡約方程式（reduced set of equations）を得ることになる。どの座標を用いてもプログラムのベクトル化の観点からは同様なので，以下にAEOLUS－R群のプログラムで円柱座標（R, Φ, Z）用いた例[8]を示す。なおAEOLUS－E群のプログラムはすべて円柱座標のみを用いている。

$$\frac{\partial \psi}{\partial t} + v \cdot \nabla_\perp \psi = B_0 \frac{\partial \Phi}{\partial \zeta} + \eta J - E_w(t), \qquad (2.13)$$

$$\frac{\partial U}{\partial t} + v \cdot \nabla_\perp U = \nabla \zeta \cdot \nabla P \times \nabla_\perp \left(\frac{R}{R_0}\right)^2 + \left(\frac{R}{R_0}\right)^2 \nabla \zeta \cdot \nabla_\perp \psi \times \nabla_\perp J + B_0 \frac{\partial J}{\partial \zeta} \qquad (2.14)$$

$$\frac{\partial P^*}{\partial t} + v \cdot \nabla_\perp P^* = 0, \qquad (2.15)$$

$$B = B_0 \nabla \zeta + \nabla \zeta \times \nabla_\perp \psi,$$

$$v = \left(\frac{R}{R_0}\right)^2 \nabla \zeta \times \nabla_\perp \Phi,$$

$$J = \left(R^2 \nabla_\perp \cdot \frac{\nabla_\perp \psi}{R^2}\right) \nabla \zeta,$$

$$U = \left(\frac{R}{R_0}\right)^2 \nabla^2_\perp \Phi,$$

$$P^{*}=\left(\frac{R}{R_0}\right)^{2\gamma} P ,$$

$$\zeta = R_0 \Phi$$

$$\nabla_{\perp} = \left(\frac{\partial}{\partial R}\right) \nabla R + \left(\frac{\partial}{\partial Z}\right) \nabla Z$$

ただし，R_0はトーラスの主半径，U，ψ，Φはそれぞれ渦度，ポロイダル磁場フラックス，流れ関数（Stream function）を示す変数である。更に磁力線が直線で表わされる様に円柱座標（R，Φ，Z）新しい座標（r，θ，ζ），$r=\sqrt{g}\ R_0/R^2$（$\sqrt{g}$はヤコビアン）を用いて変換し，式(2.14)－(2.15)の変数ψ，J，Φ，P^{*}を式(2.16)のようにフーリエ展開する。

$$\psi(r,\ \theta,\ \zeta)=\sum_{m,n} \psi_{m,n}(r)\exp i\left(m\theta-\frac{n}{R_0}\zeta\right), \qquad (2.16)$$

式（2.16）のように展開すると式(2.13)－(2.15)の中の角度に関する微分は消滅しｒによる微分だけが残る。ｒによる微分を差分に直すと最終的にｒに関する三重対角行列を係数行列とする連立方程式がフーリエ展開の展開項の数だけ得られる。

時間微分についても差分に直したのち，これらの方程式に対し以下の様な計算を行う。

① 非線形項を計算する為にフーリエ展開後の各モード間のカップリングを考慮しモードの重ね合わせ（コンボリューション）を行う。

② 時間発展に関しては陰解法を用いる（ただし実際は式（2.13）中のψの時間微分とＪに関してだけ行う）。

③ 三重対角方程式を解く。

①と③の部分はｒについてベクトル化され，殆んどの部分は何の手も加えなくとも自動的にベクトル化される。②の部分に関しては必然的に再帰計算が現われて，そのままではベクトル化できない。しかし，フーリエ展開の項数（AEOLUSプログラム群では約20）でベクトル化することができる。

この様にしてAEOLUSプログラム群では殆んど全部がベクトル化されており2.2節に示す様にベクトル計算の効率は極めて良い。

2.1.4 放射性粒拡散のシミュレーションと被曝放射線量の計算（PRWDA）

放射性粒子の拡散から地上での放射線被曝量を推定するプログラムには拡散方程式を解いているものが多い。ここではこの拡散方程式をＰＩＣ法及びモンテカルロ法を用いてシミュレーションしているプログラムPRWDAを例にとってその概要を説明する。

プログラムPRWDA[9]は最近原研において開発された緊急時被曝放射線情報予測システムSPEEDI[10]の一部であり，ソースライン数は約3900，サブルーチン数は40である。参考の為SPEEDI中でのPRWDAの位置をFig. 2.4に簡単に示す。

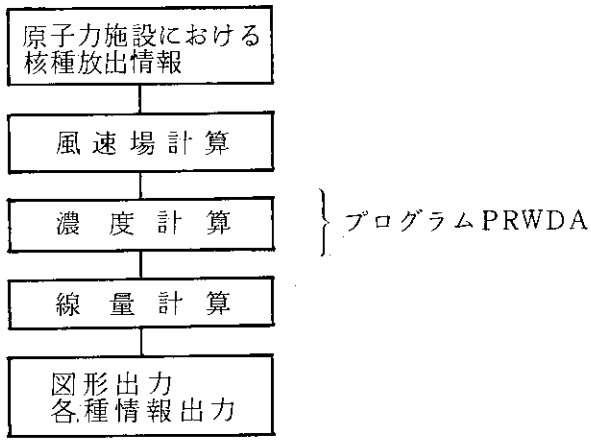


Fig. 2.4　Program PRWDA and the SPEEDI System.

プログラムPRWDAでシミュレートされている3次元拡散方程式は式（2.17）で示すようなものである。

$$\frac{\partial X}{\partial t} + \mathbf{u}_A \nabla X - \nabla(\mathbf{K} \nabla X) = 0 \qquad (2.17)$$

X：濃度

$\mathbf{u}_A$：移流速度

モンテカルロ法で式（2.17）をシミュレーションする場合には時刻tを基準にした時の時刻 $t+\Delta t$ での粒子の位置を次のように定める。

$$x_{t+\Delta t} = x_t + u_x \Delta t + x_{p'} \qquad (2.18)$$

u_x：x方向の風速

$x_{p'} = \sqrt{24 K \Delta t}\,(0.5-R)$

K：拡散係数　R：〔0, 1〕間の一様乱数

またPIC法では

$$x_{t+\Delta t} = x_t + u_{p,x} \Delta t \qquad (2.19)$$

$u_{p,x}$ は移流速度 $\mathbf{u}_A$ と拡散速度 $\mathbf{u}_d = -\mathbf{K}(\nabla X/X)$ の合成で与えられる。

プログラムPRWDAではZ方向については基本的にPIC法を，x，y方向に対してはモンテカルロ法を用いている。式（2.17）のシミュレーションは空間をセルに分割して行われるが粒子が1つのセルに何個も同時に存在したり，あるいは全くないというような場合が起る。また計算領域外に出て行く粒子も生じてくる。これらはモンテカルロ法による粒子シミュレーションに見られる特徴的な現象である。

プログラムのベクトル化に当ってはこのモンテカルロ法に特有な現象をうまく処理しないと計算効率を高めることができない。この問題は有効粒子のみを集めてパッキングすることにより解決できる。詳しくは，3.3節を参考にされたい。

更に，この様にして放射性粒子の空間濃度分布が定まると，これから各地点における放射線被曝線量を積算して求めることができる。（プログラムGAMRW，文献(11)）この積算に関しても，影響を与える粒子与えない粒子があるのでその選択を行う部分のベクトル化はモンテカルロ計算

のベクトル化と同様な注意を必要とする場合がある。

2.1.5 有限要素法による構造解析計算（SAP 5）

構造解析計算や強度計算は近年非常に大規模な構造物にも適用される様になっている。SAP 5[12]は，原研においては，多目的高温ガス炉やNBI 加熱装置の設計などによく使用されている。このような状況で計算機側では高速演算機能や大容量記憶装置が必要となっており，プログラム側では高速な演算アルゴリズムや解法，またメモリ使用を少なくする方法などが試みられている。ここで概説するプログラムSAP 5にもこれらの観点から特色のある工夫が施されている。

SAP 5では以下の3種類の計算を行うことができる。

- 静的解析（Static Analyses）
- 固有振動のモード解析（Calculation of Frequencies and Mode Shapes）
- 動的解析（Dynamic Analyses）

静的解析は通常よく行われる釣り合い状態の計算を行うもので最終的には次の多元連立方程式を解くことに帰着する。

$$K\vec{u} = \vec{R} \qquad (2.20)$$

K：剛性マトリクス

$\vec{u}$：変位ベクトル

$\vec{R}$：荷重ベクトル

（2.20）式を解く場合には$K = LDL^t$と分解し前進消去後退代入を行う。

動的解析では

$$M\ddot{u} + C\dot{u} + Ku = R(t) \qquad (2.21)$$

M, C, K：係数マトリクス

R(t)：時間依存成分を持つベクトル

のような形式の式を解くことになるが，この場合には2つの方法が用意されている。1つはここで主に説明する振動モード解析後の各振動モードを重ね合わせて，時間をtからt＋ΔtとΔtのステップで（2.21）式を解いてゆく方法である。もう1つは（2.21）式をΔt ステップで直接解いてゆく方法である。これらは有限要素法でよく行われる時間発展の方程式を解く場合と同様であるのでここでの説明は避ける。

固有振動モードの解析では次の固有値方程式を解くことになる。

$$K\phi = \omega^2 M\phi \qquad (2.22)$$

K：剛性マトリクス

ω：固有振動数

M：質量マトリクス

ϕ：固有ベクトル

固有振動モードはKがn次元マトリクスであれば最大n個存在するが，全てのモードを解析するのは非常に時間がかかるので物理現象的に最も重要だと思われる低い振動数のモードに着目して解析を行うことがよく行われる。SAP 5においても低い振動数のモード P 個を調べる様になっ

ている。

SAP5においてはp個の固有ベクトルを求めるのに部分空間反復法（subspace iteration）と呼ばれる方法を用いている。部分空間反復法は対象とする固有ベクトルが張る空間内で(2.22)式を扱おうとするものであり，固有ベクトル全てを同時に反復法で求めるという点で同時反復法（simultaneous iteration，例えば文献⑬参照）に似ていると言える。またSAP 5 ではp個の固有ベクトルを精度良く少ない反復回数で求める為に保護ベクトル(guard vector，文献⑬，p. 305）を追加し合計q個の固有ベクトルを対象とする様になっており，ここでは$q = \min$（2p，p＋8）が採用されている。

ここで実際に行われている計算は以下に示す通りである。

(2.22) 式でϕがn個の要素を持つベクトル，K，Mがそれぞれn×nの行列であるとする。適当なn次元ベクトルq個を用いて初期固有ベクトル行列V_0を定義する。

$$V_0 = (p_1, p_2, \cdots\cdots, p_q) \quad ; \quad n \times q \text{ 行列} \tag{2.23}$$

ベクトル行列$\bar{V}_k$を求める。

$$K\bar{V}_k = MV_{k-1} \quad ; \quad k \text{ は計算全体の反復回数} \tag{2.24}$$

$\bar{V}_k$を求める時は$K = LDL^t$と分解し前進消去・後退代入を行う。またKの分解は最初に1回行うだけでよい。

次にq個の独立したベクトルで張る部分空間にKとMを射影する。

$$K_k = \bar{V}_k^t \, K \, \bar{V}_k \tag{2.25}$$

$$M_k = \bar{V}_k^t \, K \, \bar{V}_k \tag{2.26}$$

(2.25)，(2.26) 式のように変数を縮約して次の固有値方程式を解く

$$K_k Q_k = M_k Q_k \Omega_k^2 \tag{2.27}$$

Ω_k^2 ： 固有値対角行列

Q_kはq×q 行列で各列が(2.27) 式を満たす固有ベクトルである。(2.27) 式はJacobiの方法（文献⑬，p. 250参照）により解きΩ_k^2とQ_kを求めている。この際Ω_k^2の要素は固有値の小さい順に並ぶようにしておく。

これで部分空間での固有値問題が解けたので，部分空間の固有ベクトルを元の空間の固有ベクトルに変換する。

$$V_k = \bar{V}_k Q_k \tag{2.28}$$

このV_kを使って再び（2.24）式以下の計算をV_kが収束するまで繰り返す。

p個の固有ベクトルと固有値が求まったら，これらの固有値が本当に小さい振動数からp個採用されているかどうかSturm列と逆反復法（文献⑬，p. 295参照）を用いて確かめる。

部分空間反復法の収束の速さは初期固有ベクトル行列V_0に強く依存するのでV_0の選び方には注意を要する。Jennings（文献⑬p. 315参照）はこの点を指摘すること及び計算量から見ても大差がないことなどから同時反復法の方が有利であると主張している。色々な計算方法を試してみることは我々の今後の課題である。

SAP5のベクトル化は極めてうまく行われている。前述の計算方法なども殆んどが行列演算などでベクトル化が容易なものである。プログラムがVP100／200でどの程度高速に実行されるかは対象とする構造物に強く依存するので一概には言えないがM 380に比べ4～10倍程度である。

SAP5は比較的小容量の主記憶を持つ計算機でも実行可能とする為に，剛性行列などを分割し，各部分のデータの入出力を繰り返しながら計算を進める様になっている。その為，大型の計算では入出力回数が数万から数十万に達することがあり，CPU時間はそれ程多くない場合でも最終的に結果を得るまでの経過時間が非常に長くなることがある。これを避ける為に，剛性行列全体を主記憶に常駐させる様にした特別なベクトル化版SAP 5が用意されている。ただし，原研で通常使用できる最大の主記憶領域8 MB（許可ジョブはこの限りにあらず）に対して，このベクトル化版コードでは扱える方程式の数が2000程度（即ち3次元の構造物ではノード数700程度）である。

2.1.6 軽水炉安全性解析計算（RELAP5／MOD1）

軽水炉安全性解析において，代表的なコードであるRELAP5／MOD 1[14]を例に述べる。このコードは，軽水炉LOCAおよびnon－LOCA過渡状態解析のために開発された1次元過渡解析コードであり，原研においてはROSA－ⅢおよびⅣ計画においてよく使用されている。

RELAP5／MOD 1の基本方程式とそれを差分化した式については，文献[15]に解説されているので詳しくはそちらを参照されたい。

二流体非平衡モデルの基本方程式は，次の2つの質量保存の式，2つの運動量の式，1つの混合エネルギーの式の合計5個の方程式で表わされる。

質量保存の式：

$$\frac{\partial}{\partial t}(\alpha_g\rho_g)+\frac{1}{A}\frac{\partial}{\partial x}(\alpha_g\rho_g v_g A)=\Gamma \ , \tag{2.29}$$

$$\frac{\partial}{\partial t}(\alpha_f\rho_f)+\frac{1}{A}\frac{\partial}{\partial x}(\alpha_f\rho_f v_f A)=-\Gamma \ . \tag{2.30}$$

運動量の式：

$$\begin{aligned}&\alpha_g\rho_g A\frac{\partial}{\partial t}v_g+\frac{1}{2}\alpha_g\rho_g A\frac{\partial}{\partial x}v_g^2\\&=-\alpha_g A\frac{\partial P}{\partial x}+\alpha_g\rho_g B_x A-(\alpha_g\rho_g A)FWG(v_g)+\Gamma_g A(v_{gI}-v_g)\\&-(\alpha_g\rho_g A)FIG(v_g-V_f)-C\alpha_g\alpha_f\rho A\left[\frac{\partial}{\partial t}(v_g-v_f)+v_f\frac{\partial}{\partial x}v_g-v_g\frac{\partial}{\partial x}v_f\right] \ .\end{aligned} \tag{2.31}$$

$$\alpha_f\rho_f A\frac{\partial}{\partial t}v_f+\frac{1}{2}\alpha_f\rho_f A\frac{\partial}{\partial x}v_f^2$$

$$=-\alpha_f A\frac{\partial P}{\partial x}+\alpha_f\rho_f B_x A-(\alpha_f\rho_f A)FWF(v_f)+\Gamma_f A(v_{fI}-v_f)$$

圧力勾配　体積力　壁との摩擦　二相間の運動量交換　　(2.32)

$$-(\alpha_f\rho_f A)FIF(v_f-V_g)-C\alpha_f\alpha_g\rho A\left[\frac{\partial}{\partial t}(v_f-v_g)+v_g\frac{\partial}{\partial x}v_f-v_f\frac{\partial}{\partial x}v_g\right]\ .$$

二相間の摩擦　仮想質量加速項

エネルギーの式：

$$\frac{\partial}{\partial t}(\alpha_g\rho_g U_g+\alpha_f\rho_f U_f)+\frac{1}{A}\frac{\partial}{\partial x}(\alpha_g\rho_g v_g U_g A+\alpha_f\rho_f v_f U_f A)$$

$$=-\frac{1}{A}\frac{\partial}{\partial x}(\alpha_g v_g PA+\alpha_f v_f PA)+(\alpha_g\rho_g v_g+\alpha_f\rho_f v_f)B_x+Q\ .\qquad(2.33)$$

混合流体を示す添字mは省略している。添字Iは二相境界を意味する。

A：流路断面積。

$\alpha_g=\alpha$：蒸気体積比（ボイド率）。

$\alpha_f=1-\alpha_g$。

$\Gamma=\Gamma_g=-\Gamma_f$，全体として質量の増減はないので，一般的にこの関係が成り立つ。

B_x：x軸方向の体積力。

FWG，FWF：蒸気および液体に対する壁摩擦係数（wall frictional drag coefficient）。

FIG，FIF：蒸気および液体に対する二相間摩擦係数（interphase drag coefficient）。

C：仮想質量係数。

$\rho=\alpha_g\rho_g+\alpha_f\rho_f$。

$U=e+\frac{1}{2}v^2$。

Q：熱源。

これらの式のうち，（2.29）と（2.30）や（2.31）と（2.32）は数値計算上の工夫により,各々対をなす和と差の式を基本式として採用している。

コードにおいては，差分化した式を最終的には圧力Pに関する方程式として解いている。

RELAP5／MOD1コードにおける計算の流れをFig. 2.5に示す。この中で,高速化のためには，方程式を解く部分よりもむしろ，係数計算の部分（Fig. 2.5のconstitutive relationsやspecial process modelの計算）に計算時間が多くかかっているので，この部分からベクトル化を進める。係数計算の部分においては，staggered mesh法とdonor cell法によって，ノード付けされた各ジャンクションとボリュームに対して独立に計算できるので，この並列性を最大限に利用してベクトル計算する。

RELAP5／MOD1コードのベクトル化については，文献(16)に詳しく述べられている。

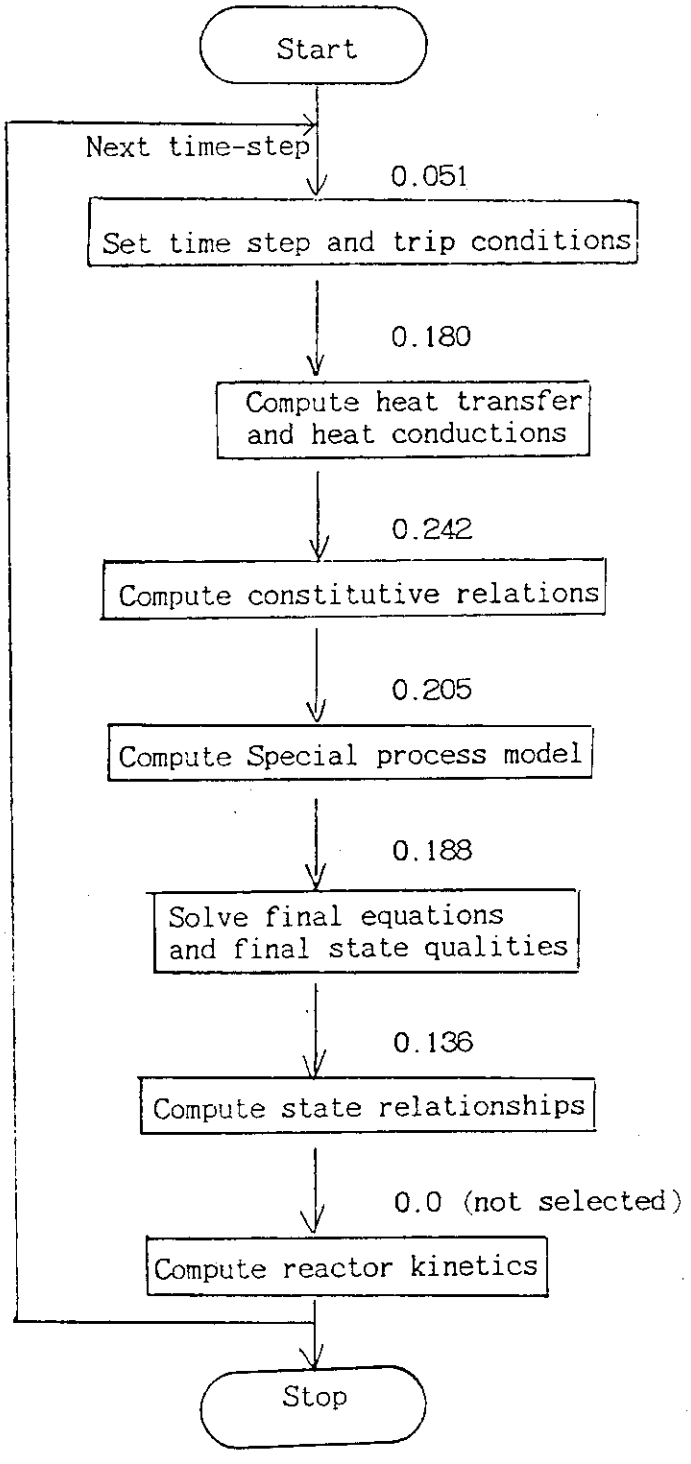


Fig. 2.5 Block Diagram of the RELAP5/MOD1 Code and Proportion of CPU-Time Consumption of Each Calculation Block in the Transient Analysis Calculation.

2.2 原子力コードのベクトル化の効果

原研でベクトル化された原子力コードのスカラ計算機M−380とベクトル計算機VP−100／200での実行時間の比較をTable 2.1に示す。表中に示されている内容は左欄より順に以下の通りである。

第一欄：プログラム名（適用分野）
第二欄：計算内容（物理基礎方程式）
第三欄：数値解法
第四欄：プログラムライン数と精度
第五欄：ベクトル長とベクトル化率
第六欄：M−380, VP−100, VP−200での実行CPU時間
　　α：オリジナルプログラムをM 380で実行した場合
　　β：ベクトル化したプログラムをM 380で実行した場合
　　γ：　〃　VP 100／200で実行した場合
第七欄：性能向上比
　　$U=\alpha/\gamma$　　$P=\beta/\gamma$
第八欄：ベクトル化により変更されたプログラムラインの割合とベクトル化後のメモリ増加の割合
第九欄：附　記

ベクトル化によるプログラムの性能向上はベクトル長とベクトル化率に強く依存し，一般にこの2つの値が大きい程性能向上比も大きくなる。従ってプログラム作成の時にはこの事を念頭に置いておく必要がある。詳しくは文献（3,24）を参照されたい。

なおVP 100は原研に設置してあるもの，VP 200は富士通沼津工場のものを使用している。

Table 2.1に示されている物の中で特に目立ったいくつかの事柄について以下に列挙する。

(1) TOROINDコード

ベクトル化したプログラムは，M 380で実行してもオリジナルのプログラムより高速に実行される物が多いが，これはベクトル化すると同時にプログラムの汎用最適化を行っているからである。この最も良い例が磁場計算を行うコードTOROINDである。これはベクトル化を含む汎用最適化のみで5倍以上高速化されている（$\alpha/\beta \gtrsim 5$）。この他のプログラムで，ベクトル化版をM−380で実行すると元のプログラムより遅くなっているものがあるが，これらはベクトル化の為に余計な計算が加わった為である。

(2) モンテカルロコード

Table 2.1中に見られる2つのモンテカルロコードはベクトル化による高速化の効果が，プログラムの修正率が大きいのに反しあまり見られないが，これは大規模モンテカルロ計算に見られる複雑なＩＦ文構造の為である。モンテカルロ計算には多くの形態があり，積分問題や最近の素粒子物理学でのゲージ理論の問題[17]などでは比較的単純な構造を持ったプログラムが多く，ベクトル計算機により極めて効率良く処理される。

(3) RELAP5／MOD1コード

モンテカルロプログラム以外でもこれに似た複雑なＩＦ文構造を持つものがあり，この代表的なものが軽水炉の冷却水喪失事故を模擬するRELAP5コードである。RELAP5は複雑なＩＦ文構造の他に，演算順序や組み込み関数の計算値の僅かな精度の差によりプログラムの挙動が大幅に変化するという問題がありベクトル化の障害になっている。

(4) 有限要素法のプログラム

有限要素法を用いたプログラムの中にあまり高速化されないものが幾つかあるが，これらは主に，計算対象の幾何形状に不規則格子を用いている為，容易に再帰計算が排除できないからである。

(5) 核融合関連のコード

核融合関係で用いられているコードでは再帰関係の少ない行列演算を行っている物が多く極めて高速に処理されることが多い。

Table 2.1のみではプログラムが実際どの様に高速化されるか明らかでない。そこで幾つかのコードの計算内容をブロック化し，各ブロックがどの様に高速化されるかをFig. 2.6～2.13に示す。

(6) DICONコード

Fig. 2.7で示すコードDICONの場合では，オリジナルのプログラムはブロックごとに計算内容を分けてＣＰＵ時間の測定をすることが難かしいのでモンテカルロ計算として1つにまとめてある。DICONではＣＰＵ時間が特定のサブルーチンに集中していないこと，ベクトル化が殆んどのサブルーチンに対し行われているので，全ての計算ブロックが平均的に高速化されている。

(7) EPIC／Ⅳコード

Fig. 2.9に示すEPIC／Ⅳコードのベクトル化ではCG法による計算の部分が顕著に高速化されておりCG法がベクトル計算機向きの数値解法であることが分る（3.2節参照）。

(8) CITATIONコード

オリジナルのCITATIONコードをスカラ計算機M－380で実行すると中性子束計算の部分が全ＣＰＵ時間の70％を占める。この部分は3.1項で述べるodd－even法を組み込んだSOR法を採用する（オリジナルコードはSLOR法）ことによりベクトル化されており，この部分だけの性能向上比は9.7倍となっている。この性能向上にはベクトル長が大きい（約1500）ことが大きな要因となっている。CITATIONは中性子束計算以外の部分もベクトル化され，全体で約95％のベクトル化率となっている。Fig. 2.12に示すようにコード全体としてもベクトル化により非常に良く高速化されている。

Table 2.1 Nuclear Codes Vectorized in JAERI.

Program name (Appl. field)	Calculated physical phenomena (Employed basic equations)	Adopted numerical methods for calculation	Program statement number and precision	Vector length and cost-weighted fraction of vectorized statements (%)	CPU time orig. prog. = α(M) rev. prog. = β(M) rev. prog. = γ(VP) M=M380, VP=VP100/200	Performance improvement factors U = α/γ P = β/γ	Fractions of modified statements and memory increase (%)	Comments
THIDA-ACT4 (Nuclear fusion)	Neutron-induced radioactivity and γ ray radiation dose	Matrix exponential method	5000 Double	10 - 80 98	α = 2m16s37 β = 1m57s94 γ = 35s41 (VP100)	U=3.85 P=3.38	5 0	
EDDYTORUS (Nuclear fusion)	Analysis of transient eddy-current on the JT-60 vacuum vessel surface	Finite circuit-element method Gauss-Legendre integration	4670 Single	81 95	α = 2m24s8 β = 2m 7s4 γ = 18s1 (VP100)	U = 8.0 P = 7.0	25 10	
TOROIND (Nuclear fusion)	Calculation of inductance and magnetic field from the methods of Neuman and Biot-Savarre	Gauss-Legendre integration	1100 Double	10752 or 7632 99.8	α = 7m39s5 β = 1m20s2 γ = 6s6 (VP100)	U = 69.4 P = 12.2	30 146	
EDDYARBT (Nuclear fusion)	Calculation of power dissipation from eddy-current, eigenvalue	FEM Numerical integration	2200 Double	? 95	α = 63m43s16 β = 67m56s83 γ = 8m37s03 (VP100)	U = 7.4 P = 7.9	12 300	
AEOLUS-R3 (Nuclear fusion)	Analysis of non-linear MHD instability. (Non-linear MHD equations)	FDM Convolution	4197 Double	200 95	α = ? β = ? γ = ? (VP200)	U = 8.9	10 0	
AEOLUS-E (Nuclear fusion)	Analysis of non-linear external kink mode. (Reduced MHD equations)	FDM Fourier expansion	2000 Double	? ?	α = ? β = ? γ = ? (VP100)	U = 8.0	? 0	
ALSCYL30 (Nuclear fusion)	Analysis of non-linear instability of free-boundary plasma (Reduced MHD equations)	FDM Fourier expansion	6500 Double	20 or 200 90	α = 8m37s β = 8m26s γ = 2m 9s (VP100)	P = 4.0 U = 3.9	2 0	vu=65.3%
DICON (Nuclear fusion)	Analysis of neutral particles in the divertor chamber	Monte Carlo method	5000 Single	1 - 500 80	α = 1m22s49 β = 1m51s24 γ = 40s18 (VP100)	U = 2.1 P = 2.8	50 250	
ECIS76 (Nuclear fusion)	Cross-section of particl induced nuclear reaction (DWBA)	Numerical integration	10000 Single	 96	α = 3m36s95 β = 3m06s22 γ = 56s69 (VP100)	U = 3.83 P = 3.28	0 0	

Table 2.1 Continued

Program name (Appl. field)	Calculated physical phenomena (Employed basic equations)	Adopted numerical methods for calculation	Program statement number and precision	Vector length and cost-weighted fraction of vectorized statements (%)	CPU time orig. prog. = α(M) rev. prog. = β(M) rev. prog. = γ(VP) M=M380, VP=VP100/200	Performance improvement factors U = α/γ P = β/γ	Fractions of modified statements and memory increase (%)	Comments
PRWDA (Env. safety)	Diffusion of radioactive particles and calculation of radiation dose	Monte Carlo method PIC method	3900 ? ?	? ?	α = 2m30s91 β = 1m52s86 γ = 36s79 (VP100)	U = 4.1 P = 3.1	? ?	
MATHEW (Env. safety)	3-dimensional wind field calculation (hydro-dynamic eq.)	FDM SOR with odd-even method	3000 Single	50 - 1260 98	α = 5m30s99 β = 2m21s41 γ = 8s59 (VP100)	U = 38.6 P = 15.9	? ?	
WIND04 (Env. safety)	3-dimensional wind field calculation (hydro-dynamic eq.)	FDM SOR with odd-even method	500 Single	50 - 1260 95	α = ? β = 1m36s60 γ = 8s81 (VP100)	P = 11.0	? ?	
GAMPUL (Env. safety)	γ-radiation dose	Gauss integration	2000 Single	30 - 100 90	α = ? β = 6m25s81 γ = 1m38s88 (VP100)	P = 3.9	20 170 (1MB)	
KENO4 (Fuel safety)	Reactor criticality analysis in 3-dimensional system and neutron multi-energy group system	Monte Carlo method	7600 Double	1 - 300 60	α = 6m20s β = 8m30s γ = 7m55s (VP100)	U = 0.9 P = 1.1	? ?	
FEMAXI-III (Fuel safety)	Analysis of dynamic behaviour of pellet and fuel element	FEM	12500 Double	20 60	α = 1m59s84 β = 1m28s90 γ = 1m 2s17 (VP100)	U = 1.93 P = 1.43	48 30	vu=41%
HEATING-6 (Fuel safety)	Calculation of heat-conductance of vessel of radioactive materials	FDM SOR with odd-even method	17000 Double	80-270 93	α = 11m50s5 β = 12m00s92 γ = 4m06s10 (VP100)	U = 2.83 P = 2.92	9 60 (1MB)	vu=69%

Table 2.1 Continued

Program name (Appl. field)	Calculated physical phenomena (Employed basic equations)	Adopted numerical methods for calculation	Program statement number and precision	Vector length and Cost-weighted fraction of vectorized statements (%)	CPU time orig. prog. = α(M) rev. prog. = β(M) rev. prog. = γ(VP) M=M380, VP=VP100/200	Performance improvement factors U = α/γ P = β/γ	Fractions of modified statements and memory increase (%)	Comments
GMSCOPE (Reactor engin.)	Simulation of electron micro tube	Fourier transformation	240 Single	101 99	α = ? β = 19s15 γ = 1s25 (VP100)	P = 15.3	? 10 (60KB)	
ANISN (Reactor engin.)	One-dimensional neutron transport calculation	FDM	3000 Single	106 66	α = ? β = 2m29s γ = 1m38s (VP100)	U = ? P = 1.5	5 10	
TWOTRAN (Reactor engin.)	Two-dimensional neutron transport calculation	FDM X-Y coord. system R-Z coord. system	7000 Single	36 95 (X-Y) 80 (R-Z)	α = ? (X-Y) β = 1m57s (X-Y) γ = 31s (X-Y, VP100) α = ? (R-Z) β = 58s (R-Z) γ = 22s (R-Z, VP100)	X-Y: P = 3.7 R-Z: P = 2.7	X-Y: 5 20 R-Z: 5 20	
CITATION (Reactor engin.)	Three-dimensional neutron diffusion calculation	FDM SOR with odd-even method	29000 Single	1483 - 68544 95	α = 4m34s11 β = 5m10s61 γ_1 = 43s15 (VP100) γ_2 = 37s40 (VP200)	U_1 = 6.4 P_1 = 7.2 U_2 = 7.3 P_2 = 8.3	4 80	I/Os are decreased to 1/10
FEM-BABEL (Reactor engin.)	Three-dimensional neutron transport calculation	FEM, SOR with the Jacobi's method in X-Y plane	5140 Single	223 95	α = 1m49s β = 2m48s γ = 26s (VP100)	U = 4.2 P = 6.5	10 50	
DOT3.5 (Reactor engin.)	Two-dimensional neutron transoport calculation	FDM Successive calcul. from the boundary	7000 Double	45 80	α = 7m 6s β = 5m34s γ = 3m39s (VP100)	U = 1.8 P = 2.0	5 10	I/Os are decreased to 1/4 vu=30%

Table 2.1 Continued

Program name (Appl. field)	Calculated physical phenomena (Employed basic equations)	Adopted numerical methods for calculation	Program statement number and precision	Vector length and cost-weighted fraction of vectorized statements (%)	CPU time orig. prog. = α(M) rev. prog. = β(M) rev. prog. = γ(VP) M=M380, VP=VP100/200	Performance improvement factors U = α/γ P = β/γ	Fractions of modified statements and memory increase (%)	Comments
RELAP5 /MOD1 (Reactor safety)	Safety analysis of light-water reactor (Transient analysis of LWR LOCA and non-LOCA)	FDM Hydro-dynamic eq. Node-junction model	80000 Double	20 - 265 70	α = 81m31s β = 48m09s γ = 28m58s (VP100)	U = 2.8 P = 1.7	15 10	Overlayed structure
THYDE-W (Reactor safety)	Safety analysis of light-water reactor (LWR LOCA)	FDM Hydro-dynamic eq. Node-junction model	45000 Double	1 - 200 70	α = 2m21s7 β = 2m34s7 γ = 1m17s0 (VP100)	U = 1.84 P = 2.00	? 37 (0.6MB)	
TRAC-PF1 (Reactor safety)	Safety analysis of light-water reactor (Transient analysis of LWR LOCA and non-LOCA)	FDM Hydro-dynamic eq. Node-junction model	102000 Double	- 	α = β = γ = (VP100)	U = P =	?	Now at work
SAP5 (High temp reactor)	Linear structural analysis of a construction	FEM Subspace iteration method	16000 Double	50 - 200 90	α = ? β = 46s γ = 12s (VP100)	P = 4.1	20 200 (3.5MB)	I/Os are decreased to 1/8
SOLA-ICE (High temp reactor)	Velocity distribution of fluid (Hydro dynamic eq.)	FDM SOR (Odd-even) SOR (Hyper plane)	500 Double	4 (OE) 1 - 8 (HP) 90	α = 1m10s β_1 = 1m20s (OE) γ_1 = 45s (OE, VP100) β_2 = 1m15s (HP) γ_2 = 42s (HP, VP100)	U_1 = 1.6 P_1 = 1.8 U_2 = 1.7 P_2 = 1.8	3 0	
SALE (High temp reactor)	Velocity distribution of fluid (Hydro dynamic eq.)	FDM SOR with odd-even method	3100 Double	25 95	α = 25s β = 24s γ = 5s (VP100)	U = 5.0 P = 4.8	20 ?	
TRUMP4 (High temp reactor)	Analysis of thermal behaviour of vessel of radioactive material	FEM SOR with odd even method	5300 Double	150 - 290 70	α = 18s6 β = γ = 8s5 (VP100)	U = P = 2.2	10 4 (0.1MB)	vu=33%
EPIC/IV (High temp reactor)	Analysis of elastic-plastic deformation	FEM Conjugate gradient method	1870 Single	3 - 782 70	α = 2m 8s86 β = 1m26s38 γ = 33s97 (VP100)	U = 3.79 P = 2.54	5 30	
SPIN (Physics)	Spin-spin interaction in the two-dimensional Ising model	Numerical summation based on the statistical mechanics	2600 Double	60 - 65000 97	α = 3m15s β = 2m 6s73 γ = 8s63 (VP100)	U=22.6 P=14.7	5 10	vu=100%

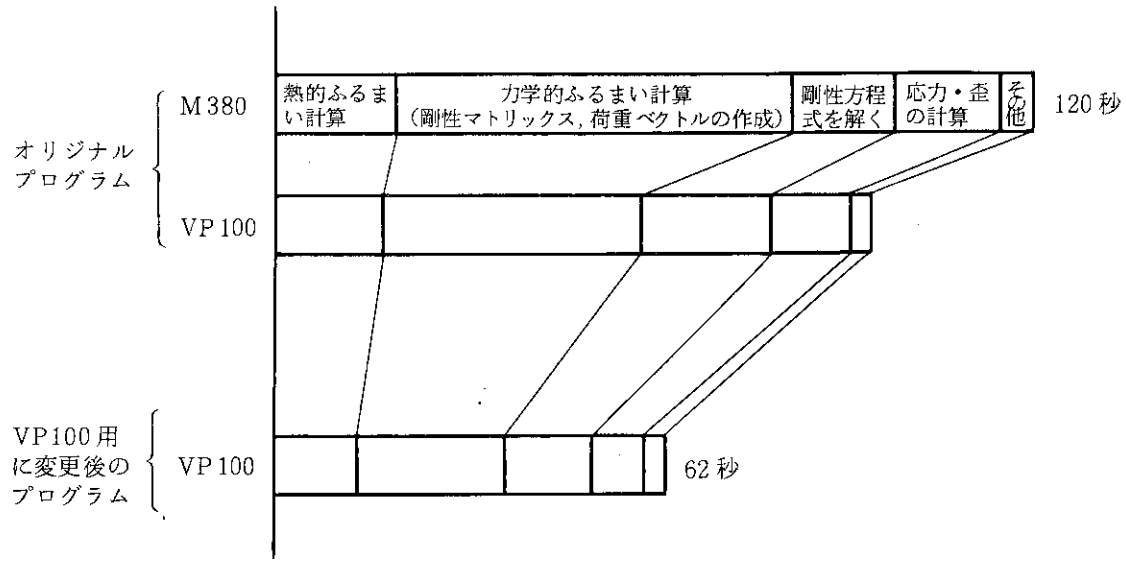


Fig. 2.6 Vectorization Effect on the Reduction of the CPU-Time of the FEMAXI-III Code.

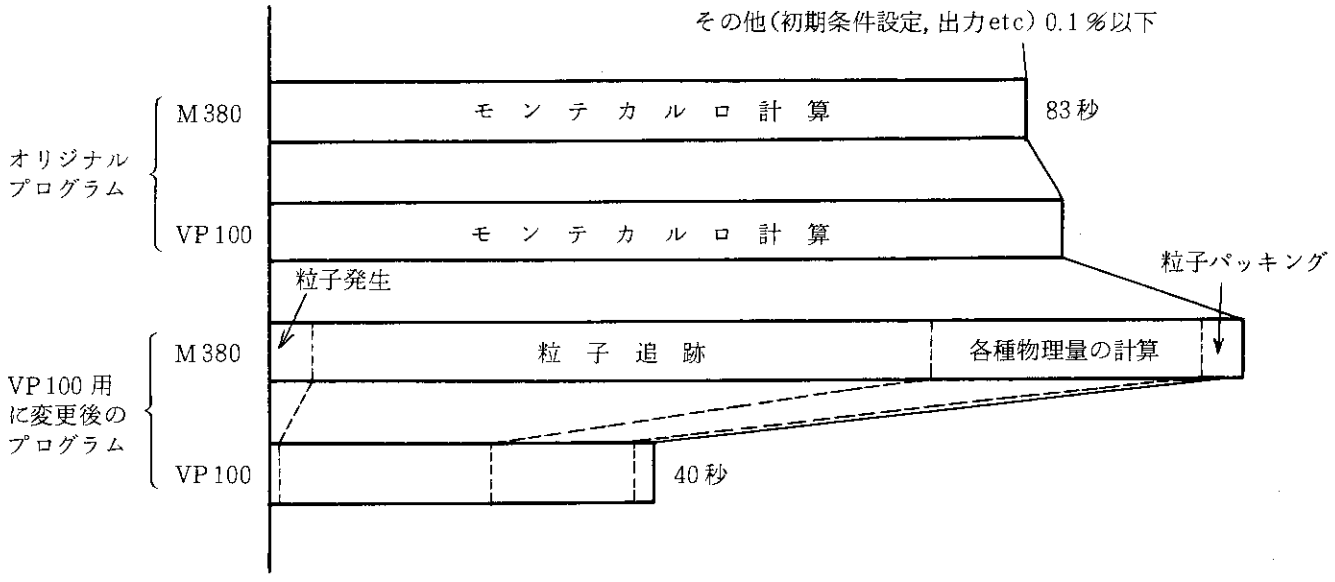


Fig. 2.7 Vectorization Effect on the Reduction of the CPU-Time of the DICON Code.

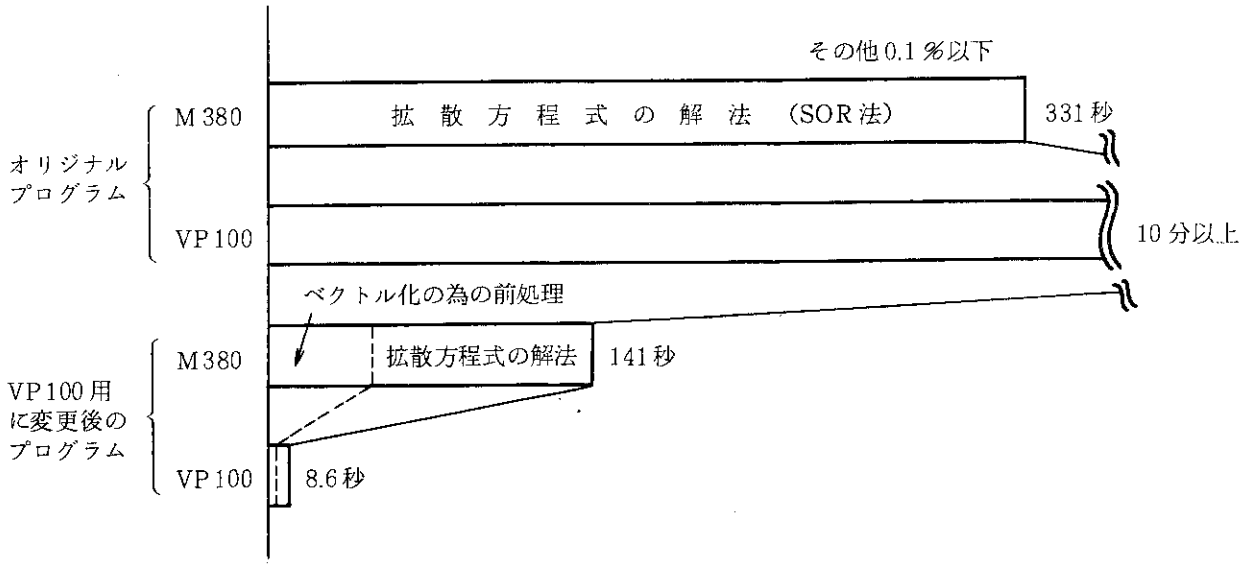


Fig. 2.8 Vectorization Effect on the Reduction of the CPU-Time of the MATHEW Code.

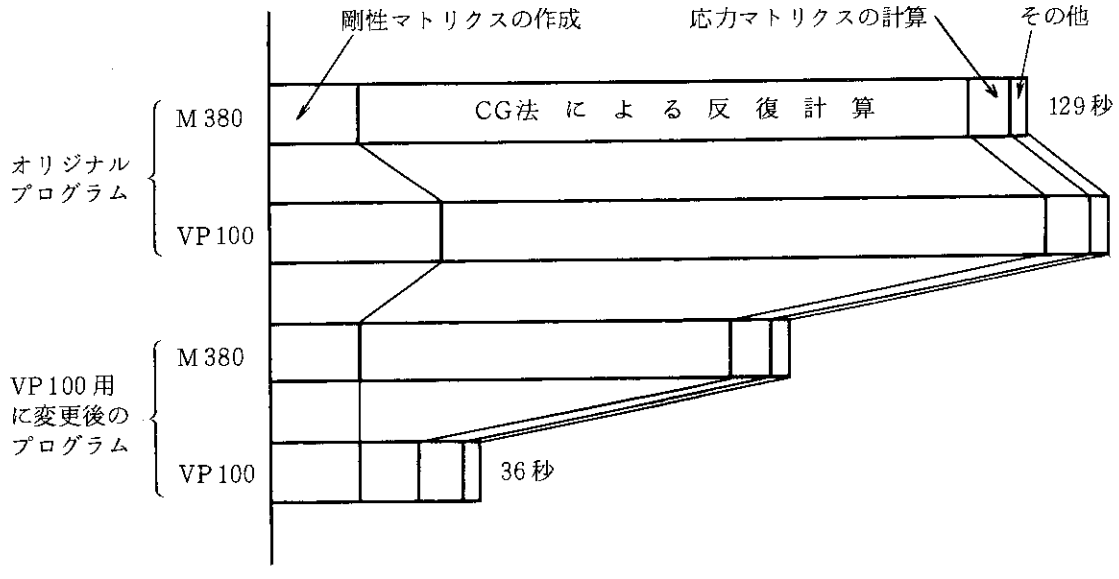


Fig. 2.9 Vectorization Effect on the Reduction of the CPU-Time of the EPIC/IV Code.

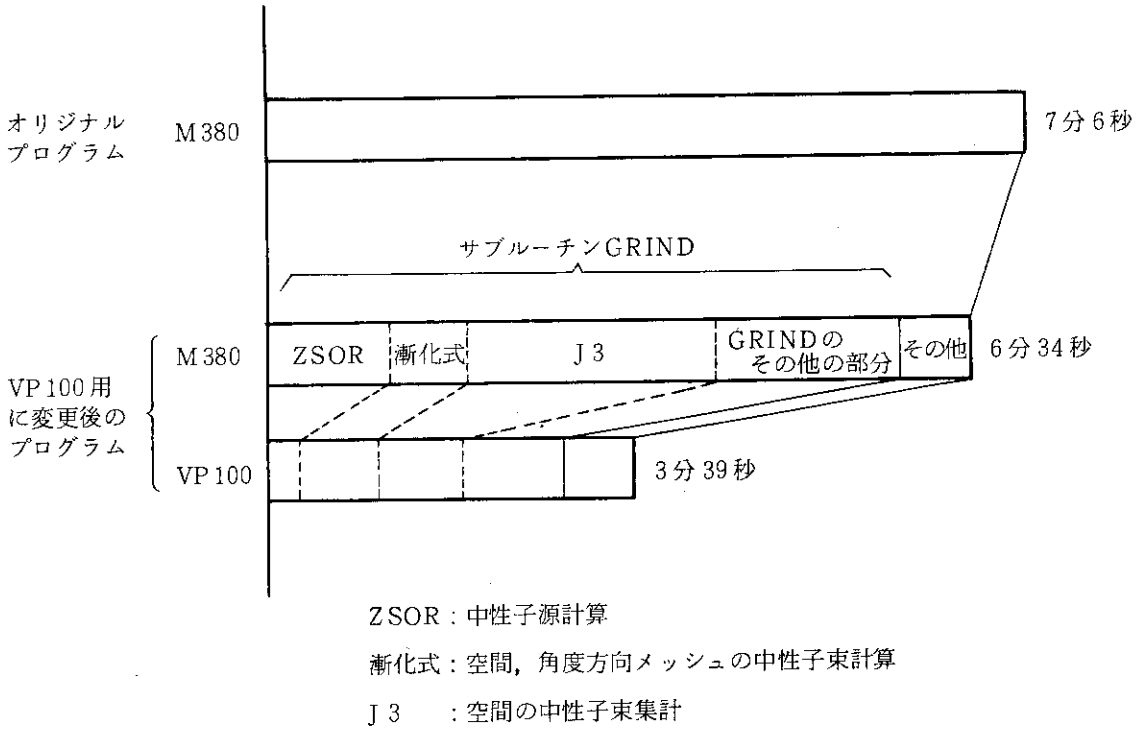


Fig. 2.10　Vectorization Effect on the Reduction of the CPU-Time of the DOT3.5 Code.

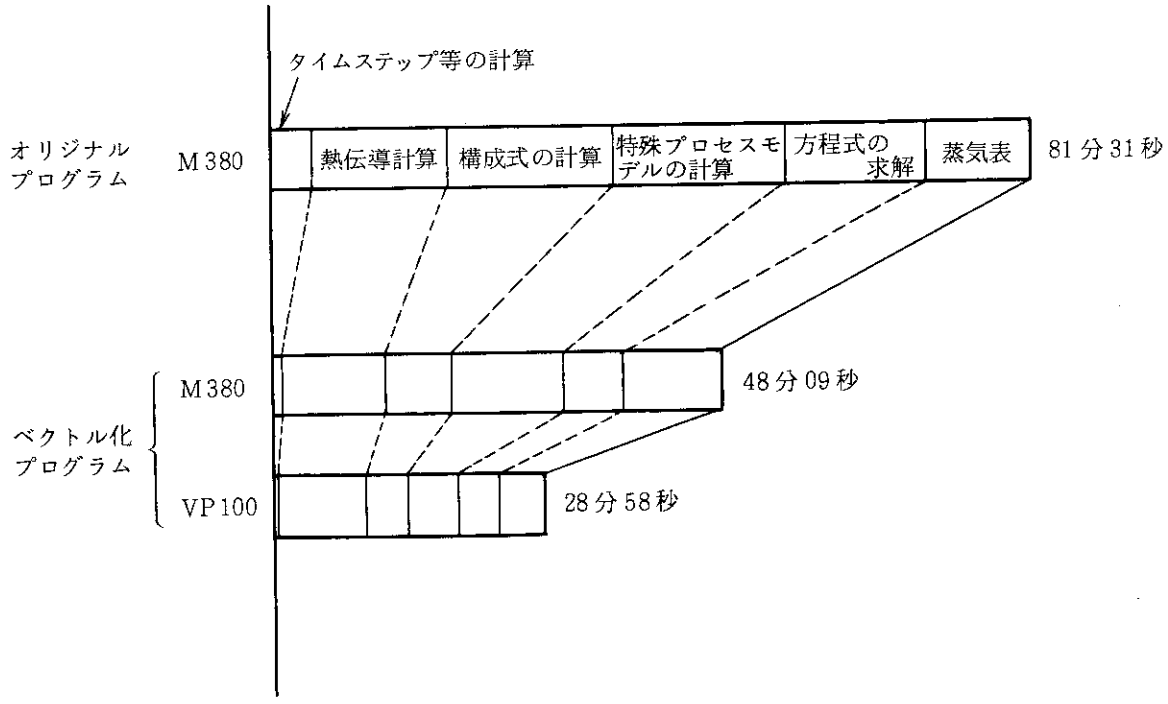


Fig. 2.11　Vectorization Effect on the Reduction of the CPU-Time of the RELAP5/MOD1 Code.

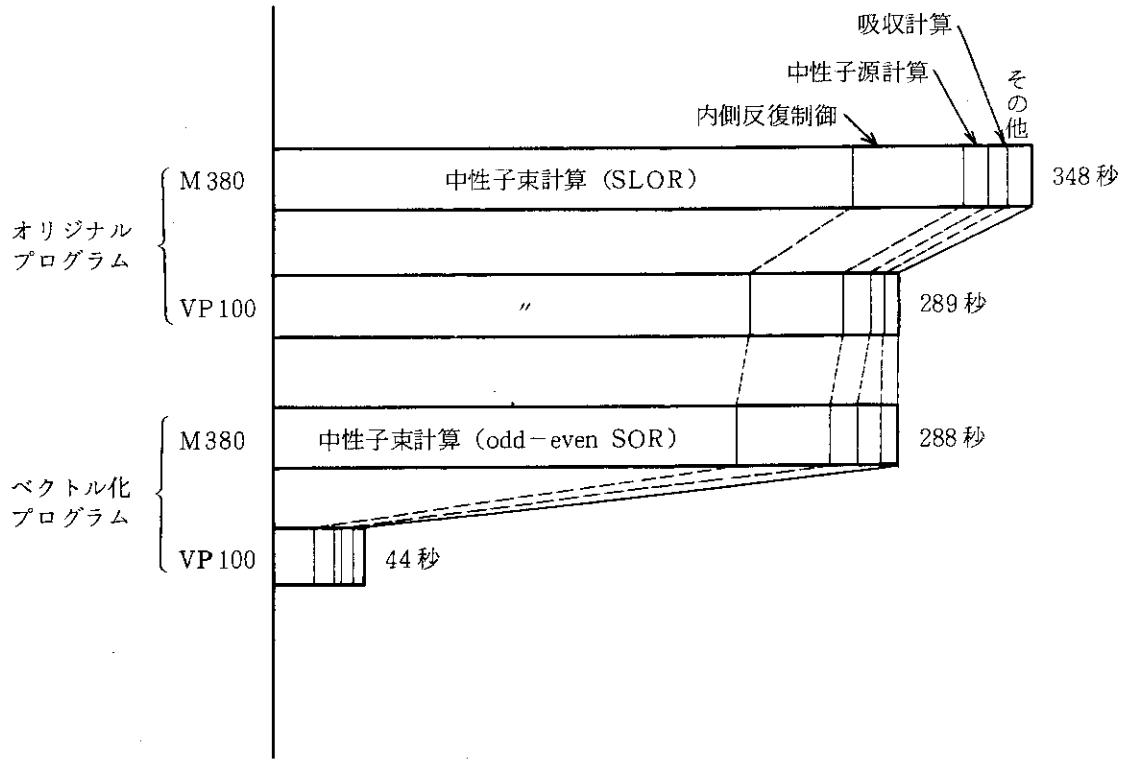


Fig. 2.12 Vectorization Effect on the Reduction of the CPU-Time of the CITATION Code.

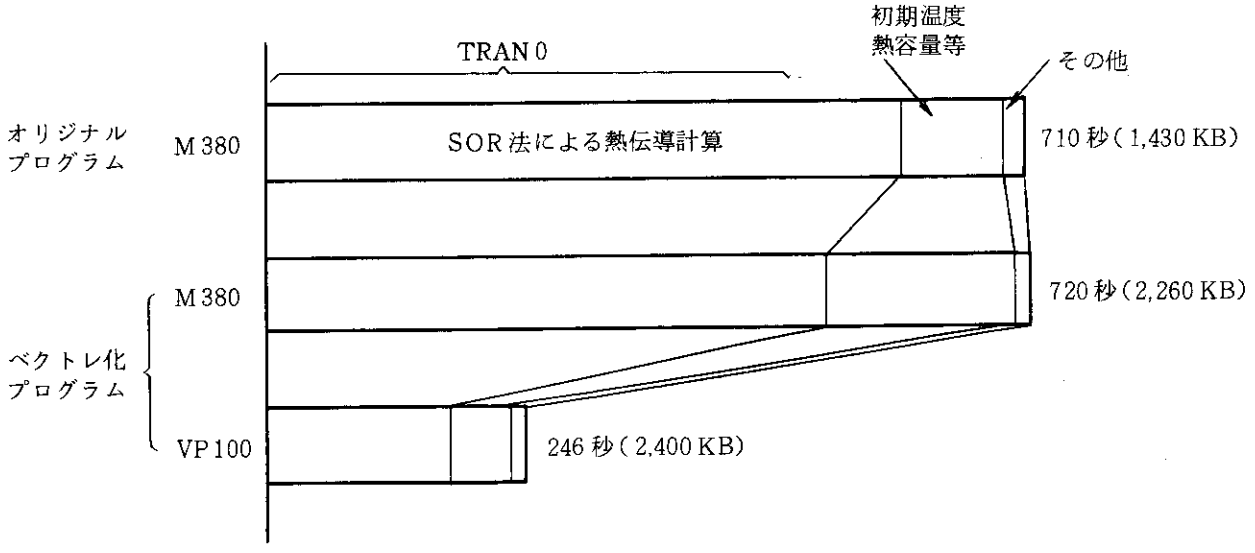


Fig. 2.13 Vectorization Effect on the Reduction of the CPU-Time of the HEATING6 Code.

3. 数値解法とベクトル化

これまで１つの問題が与えられた時に，この問題に対しどの数値解法を用いるかは全てスカラ計算機での処理を前提に考えられてきた。しかしVP－100などでベクトル処理を行う場合には，ベクトル化の観点から数値解法を見直す必要がある。またVP－100では浮動小数点演算は内部では倍精度で行われており，プログラムを陽に倍精度計算用に書き直した方が処理速度が大きいという場合があることも考慮すべきである。本章では計算精度には言及しないが，各種の数値解法をベクトル化の立場から解説を試みる。しかし，すべての数値計算法を列挙することはできないので，よく用いられる数値計算法でも，ここで記述されていないものもある。例えば常微分方程式の解法の１つであるRunge－Kutta法などは，１組の初期値問題に関しては，必然的に再帰計算が現われる為ベクトル化できない。しかし通常は複数組の初期値問題を連立させて計算することが多いので，この場合は連立した方程式の個数をベクトル長としてプログラムのベクトル化が可能である。この様に本来はベクトル化が難かしいものでも別な方向に目を向けることによりベクトル化が可能となる場合は多い。また本章で取り上げなかったものでFACR法（Fourier Analysis and Cyclic Reduction method）についてはベクトル化プログラミングの手引[3]を参照されたい。

3.1 反復解法

いろいろな数値計算問題の核として現われてくる連立１次方程式を例に考える。今，解くべき方程式を

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \qquad (3.1)$$

とする。行列の形で表わすと，

$$A\boldsymbol{x} = \boldsymbol{b} \qquad (3.2)$$

となる。Aはn×nの行列，$\boldsymbol{x}$と$\boldsymbol{b}$は列ベクトルである。

連立１次方程式の解法としては，ガウスの消去法に代表されるいくつかの直接解法と，ヤコビ法を始めとするいくつかの反復解法がある。原子力コードにおいてよく見られるような大次元の行列を扱う場合，直接解法に比べ，反復解法は誤差の累積が少ない。また，ある程度近似解がわかっている場合や，低い精度の解でよい場合は，反復解法の方が計算時間が短い。更に，疎行列の場合には，直接解法に比べて反復解法はメモリが少なくてすむなどの利点がある。

従って，原子力コードにおいては，反復解法が多く用いられるので，以降ではヤコビ法，並び

にそれを改良したガウス＝ザイデル法，SOR 法とそれらをベクトル化するための方法である odd－even 法，hyper－plane 法，multi－color 法について述べる。

(1) ヤコビ法

ヤコビ法において，第 k 回目の反復における近似解を求める式は，Fig. 3.1 のようになる。

for $i=1,n$ 〔k:*iteration*〕

$$x_i^{(k)}=a_{i,i}^{-1}\{b_i-(a_{i,1}x_1^{(k-1)}+a_{i,2}x_2^{(k-1)}+\ldots+a_{i,i-1}x_{i-1}^{(k-1)}+a_{i,i+1}x_{i+1}^{(k-1)}+\ldots+a_{i,n}x_n^{(k-1)})\}$$

Fig. 3.1 Jacobi Iterative Method.

行列の形で表わすと次のようになる。

$$\boldsymbol{x}^{(k)}=D^{-1}\{\boldsymbol{b}-(A-D)\,\boldsymbol{x}^{(k-1)}\} \qquad (3.3)$$

ただし，D は対角行列。

この式からわかるように，ヤコビ法においては第 k 回目の反復計算で参照される値は，すべて第 k－1 回目までの計算で求められた既知の値である。従って，ヤコビ法はそのままの形でベクトル計算が適用可能であるが，収束の速さに関しては，後に述べる SOR 法などに比べて一般に遅い場合が多い。

このヤコビ法を改良した数値計算法として，ガウス＝ザイデル法と SOR 法がある。

(2) ガウス＝ザイデル法

ガウス＝ザイデル法で，(3.1) 式を解く場合を考えると，第 k 回目の反復における近似解を求める式は，Fig. 3.2 のようになり，この式を i について，1 から n まで逐次的に計算する。ガウス＝ザイデル法がヤコビ法に比べて逐次計算において一般的に収束が速いのは，$x_i^{(k)}$ を求めるときに既に計算された $x_1^{(k)}$，…，$x_{i-1}^{(k)}$ を使用する点にある。しかしながら，ベクトル計算はこの $x_i^{(k)}=\cdots x_{i-1}^{(k)}\cdots$ という回帰演算のために適用できない。

for $i=1,n$ 〔k:*iteration*〕

$$x_i^{(k)}=a_{i,i}^{-1}\{b_i-(a_{i,1}x_1^{(k)}+a_{i,2}x_2^{(k)}+\ldots+a_{i,i-1}x_{i-1}^{(k)}+a_{i,i+1}x_{i+1}^{(k-1)}+\ldots+a_{i,n}x_n^{(k-1)})\}$$

Fig. 3.2 Gauss-Seidel Iterative Method.

(3) SOR 法

SOR 法で，(3.1) 式を解く場合を考えると，第 k 回目の反復における近似解を求める式は Fig. 3.3 のようになり，この式を i について，1 から n まで逐次的に計算する。SOR 法は，ガウス＝ザイデル法によって求められる $\tilde{x}_i^{(k)}$ の収束を速めるための係数 ω（$1\leq\omega<2$）を導入したものである。

ここで，$\omega=1$ の場合が，ガウス＝ザイデル法になる。SOR 法は，ガウス＝ザイデル法と同様に，$\tilde{x}_i^{(k)}$ を求めるときに，既に計算された $x_1^{(k)}\cdots$，$x_{i-1}^{(k)}$ を使用して収速を速める工夫を

しているので，そのままの形では，逐次計算向きではあるが，並列計算しようとすると回帰演算になり，ベクトル計算は適用できない。

そこで，ベクトル計算を適用可能にするために，次に3つの方法を紹介する。

```
for i=1,n                                   〔k:iteration〕
    x̃_i^(k) = a_{i,i}^{-1} {b_i - (a_{i,1}x_1^(k) + a_{i,2}x_2^(k) + ... + a_{i,i-1}x_{i-1}^(k) + a_{i,i+1}x_{i+1}^(k-1) + ... + a_{i,n}x_n^(k-1))}

    x_i^(k) = x_i^(k-1) + ω(x̃_i^(k) - x_i^(k-1))
```

Fig. 3.3　Successive Over-Relaxation Method.

(4)　Odd－even 法

2次元の楕円型偏微分方程式を有限差分法で離散化して得られる5点階差式を例に考える。

Fig. 3. 4 に示すような2次元の格子領域において，オリジナルの SOR 法による第k回目の反復における近似解を求める式は，Fig. 3. 5 のようになる。

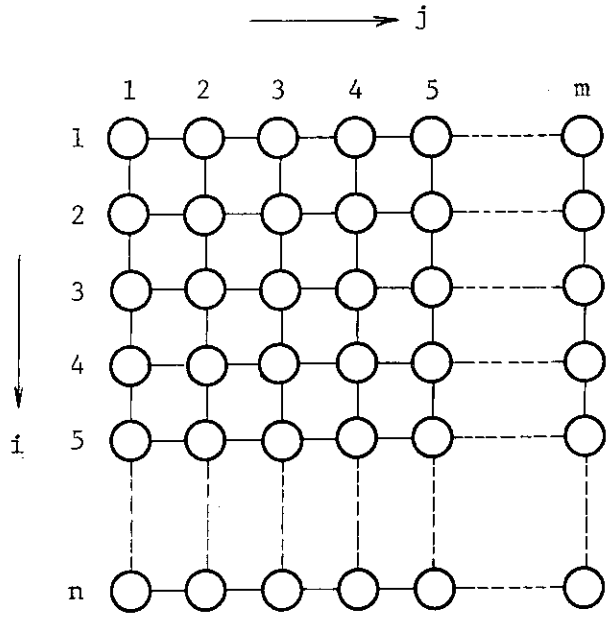


Fig. 3.4　Normal Ordering.

```
for j=1,m                                   〔k:iteration〕
    for i=1,n
        x̃_{i,j}^(k) = (1/e) {f - (a x_{i-1,j}^(k) + b x_{i,j-1}^(k) + c x_{i+1,j}^(k-1) + d x_{i,j+1}^(k-1))}

        x_{i,j}^(k) = x_{i,j}^(k-1) + ω(x̃_{i,j}^(k) - x_{i,j}^(k-1))
```

Fig. 3.5　Original SOR Method.

このオリジナルの SOR 法は，前に述べたようにこのままの形ではベクトル計算が適用できないので，以下に述べる odd－even 法によりベクトル化する。その場合，odd－even 法はインデックスを奇数の組と偶数の組に分けてそれぞれベクトル計算するので，２次元の問題において１次元のみのインデックス，例えばFig. 3.6 のようにインデックス i に対して odd－even 法を適用したのではベクトル長が n／2 と短く効果が得られない。従って，２次元のインデックス i，j を１次元化してベクトル計算する。

〔k:iteration〕

for $j=1,m$

　for $i=1,n,2$　　〔odd index〕

$$\tilde{x}_{i,j}^{(k)}=\frac{1}{e}\{f-(ax_{i-1,j}^{(k-1)}+bx_{i,j-1}^{(k)}+cx_{i+1,j}^{(k-1)}+dx_{i,j+1}^{(k-1)})\}$$

$$x_{i,j}^{(k)}=x_{i,j}^{(k-1)}+\omega(\tilde{x}_{i,j}^{(k)}-x_{i,j}^{(k-1)})$$

　for $i=2,n,2$　　〔even index〕

$$\tilde{x}_{i,j}^{(k)}=\frac{1}{e}\{f-(ax_{i-1,j}^{(k)}+bx_{i,j-1}^{(k)}+cx_{i+1,j}^{(k)}+dx_{i,j+1}^{(k-1)})\}$$

$$x_{i,j}^{(k)}=x_{i,j}^{(k-1)}+\omega(\tilde{x}_{i,j}^{(k)}-x_{i,j}^{(k-1)})$$

Fig. 3.6　Odd-Even SOR Method.

Odd－even 法は，この１次元化されたインデックスを ℓ とすると，Fig. 3.7 に示すように ℓ について奇数の組と偶数の組に分けて計算する。その式は，Fig. 3.8 に示すようになる。第 k 回目の反復において，まず ℓ の奇数の組について $\tilde{x}_\ell^{(k)}$ を計算するときに参照される x はすべて ℓ の偶数の組に属しており，次に ℓ の偶数の組について $\tilde{x}_\ell^{(k)}$ を計算するときに参照される x はすべて ℓ の奇数の組に属しており，反復１回前の $x^{(k-1)}$ ではなく直前の奇数の組の計算で求められた $x^{(k)}$ である。

このようにすることによって，オリジナルの SOR 法に見られた回帰演算を回避し，ベクトル計算が適用可能になる。この場合のベクトル長は，２次元の格子点数を n×m とすると，nm／2 である。２次元あるいは３次元の中性子拡散計算においては，100～2000 程度になりベクトル計算の効果が大きい。

ところで，２次元のインデックス i，j を１次元化する際，１次元化されたインデックス ℓ の偶数，奇数が Fig. 3.7 のように j 方向についても互い違いになるように，i 方向のサイズが奇数であればよいが，もしそうでない場合，ダミーの格子点を設ける。また，Fig. 3.8 の $\tilde{x}_\ell$ を求める計算で，$\tilde{x}_\ell$ が格子領域の境界上に来たとき，$x_{\ell-1}$ や $x_{\ell-n}$ または $x_{\ell+1}$ や $x_{\ell+n}$ が格子領域の外側に位置することがある。その場合プログラムでは配列の外側を参照することになり問題になるが，これもダミーの格子点を設けることにより解決される。その具体的な方法については，4. 2.10節で述べる。

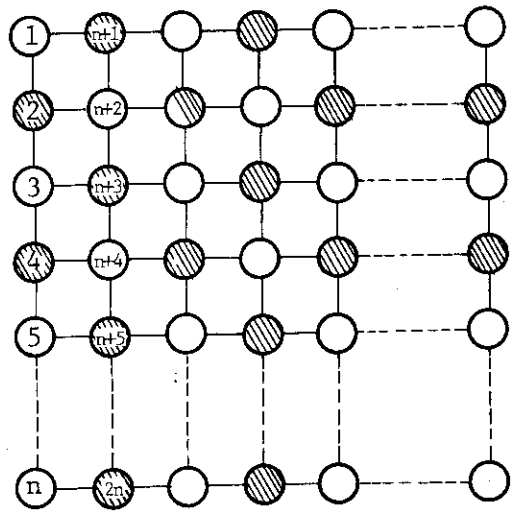


Fig. 3.7 Odd-Even (red-black) Ordering.

〔k:iteration〕

for $l=1,nm,2$ 〔odd index〕

$$\tilde{x}_l^{(k)}=\frac{1}{e}\{f-(ax_{l-1}^{(k-1)}+bx_{l-n}^{(k-1)}+cx_{l+1}^{(k-1)}+dx_{l+n}^{(k-1)})\}$$

$$x_l^{(k)}=x_l^{(k-1)}+\omega(\tilde{x}_l^{(k)}-x_l^{(k-1)})$$

for $l=2,nm,2$ 〔even index〕

$$\tilde{x}_l^{(k)}=\frac{1}{e}\{f-(ax_{l-1}^{(k)}+bx_{l-n}^{(k)}+cx_{l+1}^{(k)}+dx_{l+n}^{(k)})\}$$

$$x_l^{(k)}=x_l^{(k-1)}+\omega(\tilde{x}_l^{(k)}-x_l^{(k-1)})$$

Fig. 3.8 Odd-Even SOR Method with Transformation of Doubly Nested DO-Loops into Single DO-Loops.

また，odd－even SOR 法の場合の加速係数ωについては，オリジナルの SOR 法の場合のωとは異なる場合もあり，パラメータサーベイした方が効果的である。

Odd－even 法は，格子点を赤（Red）と黒（black）の２つの組に色分けして考えるところから red－black 法と呼ばれることもある。さらに，この方法を発展させ２色以上に色分けして考える方法は multi－color 法と呼ばれるが，この方法については後で述べる。

(5) Hyper－plane 法

Odd－even 法以外で，Fig. 3.5 で示したオリジナルの SOR 法の回帰演算を避ける方法が，hyper－plane 法である。この方法では，Fig. 3.9 (a) に示すように，右上から左下への斜め方向の格子点を対象にベクトル計算する。このベクトル計算を Fig. 3.9 (a) の番号の順に逐次的に進める。このように計算すると，Fig. 3.9 (a)，(b) で明らかなように，$\tilde{x}_{i,j}$ の計算で $x_{i-1,j}$ と $x_{i,j-1}$ は既に計算された値を用いるので，回帰演算が避けられる。

この方法の場合の加速係数ωは，オリジナルの SOR 法のωの値と同じである。

Hyper－plane 法を用いた場合のコーディング例の一部を Fig. 3.10 に示す。

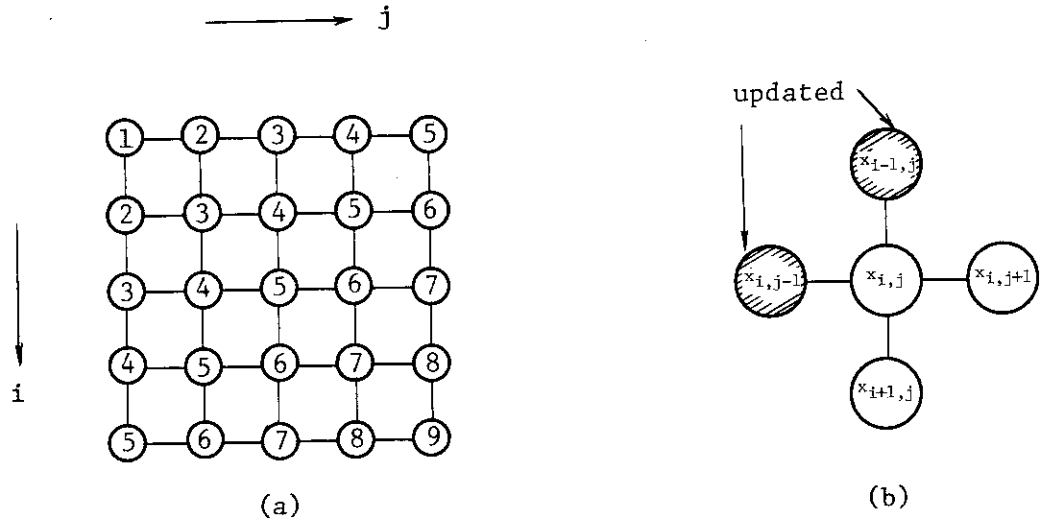


Fig. 3.9 Mesh-Point Access Sequence (a) and the Relation of Quoted Mesh-Points (b) in the Calculation with the Hyper-Plane Method.

```
      REAL  X( N, N )

      DO  10 M = 2, N+N
          DO  20 J = MAX(1,M-N), MIN(M-1,N)
              I = M - J
              XT = ( F - A*X(I-1,J) - B*X(I,J-1)
                       - C*X(I+1,J) - D*X(I,J+1) ) / E
              X(I,J) = X(I,J) + OMEGA*( XT - X(I,J) )
20        CONTINUE
10    CONTINUE
```

Fig. 3.10 A Coding-Example of Hyper-Plane Method.

n×n の格子領域を計算する場合，hyper－plane 法を用いた場合のベクトル長は，1からnまで変化し，平均で n／2 である。これに対して，odd－even 法を用いた場合は $n^2/2$ であり，一般的には odd－even 法の方がベクトル計算には有効である。

(6) Multi－color 法

Odd－even 法のところで述べた2次元の楕円型偏微分方程式を解く場合に有限差分法で離散化したときには，得られた5点階差式は，2次元領域の格子点において，前後左右の要素と結合関係がある。これに対して，Fig. 3.11(a)に示すような斜め方向の結合がある場合（例えば，三角形要素を使う有限要素法など），2色（red－black，または odd－even）の色分けでは並列計算できないが，3色の色分けでは可能になる。すなわち，ある格子点（またはメッシュ要素）に着目した場合，これに結合されている他の格子点がすべて異なる色に色分けされていれば並列計算は可能である。

例えば，Fig. 3.11(b)のようなパターンならば，中心の格子点（Blue）と結合されている前後左右と斜め方向の格子点はそれぞれ Red と Green で色分けされ，Blueの点を計算するのに Blue 以外の Red と Green のみを参照して計算できる。そして，Fig. 3.11(c)に示すように全格子点（または全メッシュ領域）がそのパターンで色分け可能ならば，各色毎にベクトル計算可能である。

この方法は，結合関係によっては3色より多くの色を必要とする場合もある[18]ので，multi－color 法と呼ばれる。

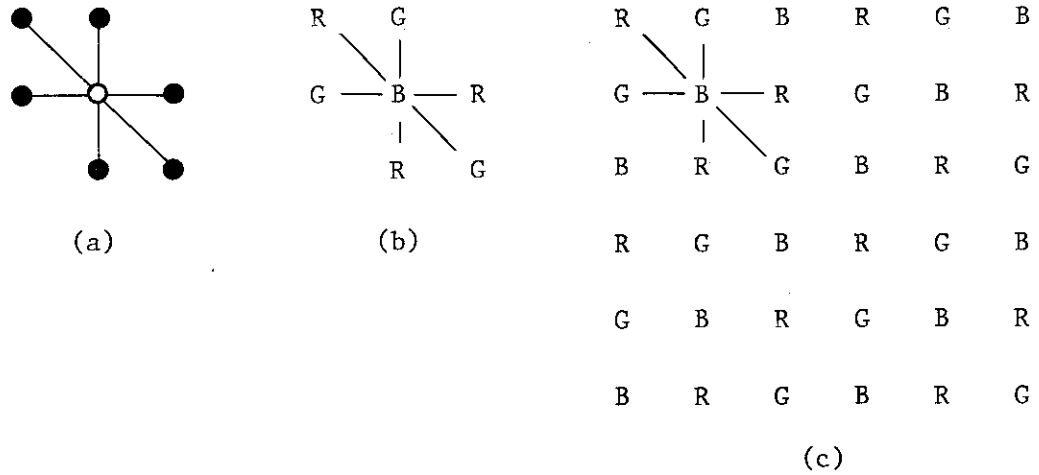


Fig. 3.11 Multi-Color Method.

3.2 共役傾斜法

CG法（共役傾斜法, Conjugate Gradient method）は対称正定値（symmetric positive definite）行列を係数行列とする連立一次方程式を解く為の反復解法の1つである。他の反復解法であるSOR法などと比べると反復1回当りの計算時間はかかるが収束に到るまでの加速が滑らかかつ自動的に行われ収束性が良いことなど有利な点がある。また特に係数行列Aの固有値がある値の近傍に密集している様な場合に収束が極めて速くなる。例えばAの固有値が3個の値の周辺に密に分布している場合には3回の反復で既に真の値からのずれは極く僅かである。

標準的なCG法による連立一次方程式の解法は以下の通りである。

$\boxed{A\vec{x}=\vec{b}\text{ の解法}}$ ； 標準的CG法

(1) $\vec{r}_0=\vec{p}_0=\vec{b}-A\vec{x}_0$, $\vec{x}_0$ は適当な初期値

↓

(2) $\alpha_i=(\vec{p}_i,\ \vec{r}_i)/(\vec{p}_i,\ A\vec{p}_i)$

又は(2)′ $\alpha_i=(\vec{r}_i,\ \vec{r}_i)/(\vec{p}_i,\ A\vec{p}_i)$

↓

(3) $\vec{x}_{i+1}=\vec{x}_i+\alpha_i\vec{p}_i$

↓

収束 (4) $\vec{r}_{i+1}=\vec{b}-A\vec{x}_{i+1}$ ここで収束判定を行う。

又は(4)′ $\vec{r}_{i+1}=\vec{r}_i-\alpha_i A\vec{p}_i$

↓

(5) $\beta_i=(-\vec{r}_{i+1},A\vec{p}_i)/(\vec{p}_i,\ A\vec{p}_i)$

又は(5)′ $\beta_i=(\vec{r}_{i+1},\ \vec{r}_{i+1})/(\vec{r}_i,\ \vec{r}_i)$

↓

(6) $\vec{p}_{i+1}=\vec{r}_{i+1}+\beta_i\vec{p}_i$

→ 終了

Fig. 3.12 Calculation Flow of the CG Method.

ここで $\vec{r}_i$, $\vec{p}_i$ はそれぞれ残差ベクトル，修正ベクトルと呼ばれている。また上記中で (2)′の表式を用いる場合は必ず(4)′(5)′の表式を用いるものとする。

CG法では $\vec{r}_0$, $\vec{r}_1$, …, $\vec{r}_n$ は互いに直交しており，従ってn元一次連立方程式の場合にはn回の反復で理論的に終了する。係数行列Aの固有値がある値に密集せずすべて分散している様な場合にはCG法の収束はあまり良くない。またAが非対称の場合を取り扱う為に $A\vec{x}=\vec{b}$ を ${}^tAA\vec{x}={}^tA\vec{b}$ と変形するような場合（${}^tAA$は対称正定値行列となる）にはAが悪い条件にある（ill－conditioned）と一層収束が悪くなる。これらのことからCG法は最近まで他の反復解法や直接解法に比べあまり注目されていなかった。

前述の様にAの固有値が密集しているとCG法では収束が極めて速くなる。このことから係数行列Aに変換を施してその性質を改善し，固有値をある値の近くに密集させる様な方法が考案されている。これをPCG法（Preconditioned Conjugate Gradient method, 前処理つき共役傾斜法）と呼んでいる。PCG法では正定値行列Aをこれに近い正定値行列Kで近似する。

$A = K + R$　　；　R はKで近似した時の残り

$\approx K$

このKを用いてPCG法の計算アルゴリズムは次の様になる。

$\boxed{A\vec{x} = \vec{b} \text{ の解法}}$　；PCG法（$A = K + R \approx K$）

(1)　$\vec{r}_0 = \vec{b} - A\vec{x}_0$, $\vec{p}_0 = K^{-1}\vec{r}_0$, $\vec{x}_0$は適当な初期値

↓

(2)　$\alpha_i = (\vec{p}_i, \vec{r}_i)/(\vec{p}_i, A\vec{p}_i)$

又は(2)′　$\alpha_i = (\vec{r}_i, K^{-1}\vec{r}_i)/(\vec{p}_i, A\vec{p}_i)$

↓

(3)　$\vec{x}_{i+1} = \vec{x}_i + \alpha_i \vec{p}_i$

収束

$\vec{r}_{i+1} = \vec{r}_i - \alpha_i A\vec{p}_i$　；ここで収束判定を行う

↓

(4)　$\beta_i = (K\vec{r}_{i+1}, A\vec{p}_i)/(\vec{p}_i, A\vec{p}_i)$

又は(4)′　$\beta_i = (\vec{r}_{i+1}, K^{-1}\vec{r}_{i+1})/(\vec{p}_i, A\vec{p}_i)$

↓

(5)　$\vec{p}_{i+1} = K^{-1}\vec{r}_{i+1} + \beta_i \vec{p}_i$

終了

Fig. 3.13　Calculation Flow of the PCG Method.

ここで前述のCG法の様に，(2)′の表式を用いる場合には(4)′の表式を必ず用いるものとする。(2)′，(4)′の表式を採用すると $K^{-1}\vec{r}_i$ の計算が1回分余計になるように見えるが，これは(2)′の計算では1つ前の計算結果をストアして使うことにより演算数は最初の1回を除き増えることはない。

このPCG法の中で最近特に着目されているものとしてMeijerinkとVander Vorst[17]のICCG法（Incomplete Choleski and Conjugate Gradient method）がある。ICCG法では対称正定値行列Aを近似する対称正定値行列Kに対し不完全コレスキ分解（Incomplete Choleski decomposition）を行う。

$$K = LDL^t \qquad (3.3)$$

D：対角行列

Meijerink等はLとして2つの形を提案し，それぞれの場合でICCG(0)，ICCG(3)と呼んでいる。ここではまずICCG(0)の場合について説明する。

具体的な例として2階の偏微分方程式をm×mのメッシュで区切った領域で解くことを考える。

$$\left(\frac{\partial}{\partial x^2} + \frac{\partial}{\partial y^2}\right) u(x, y) = \phi(x, y) \qquad (3.4)$$

中心差分を用いて差分化したのちの多元連立方程式の係数行列AはFig.3.15のようになる。

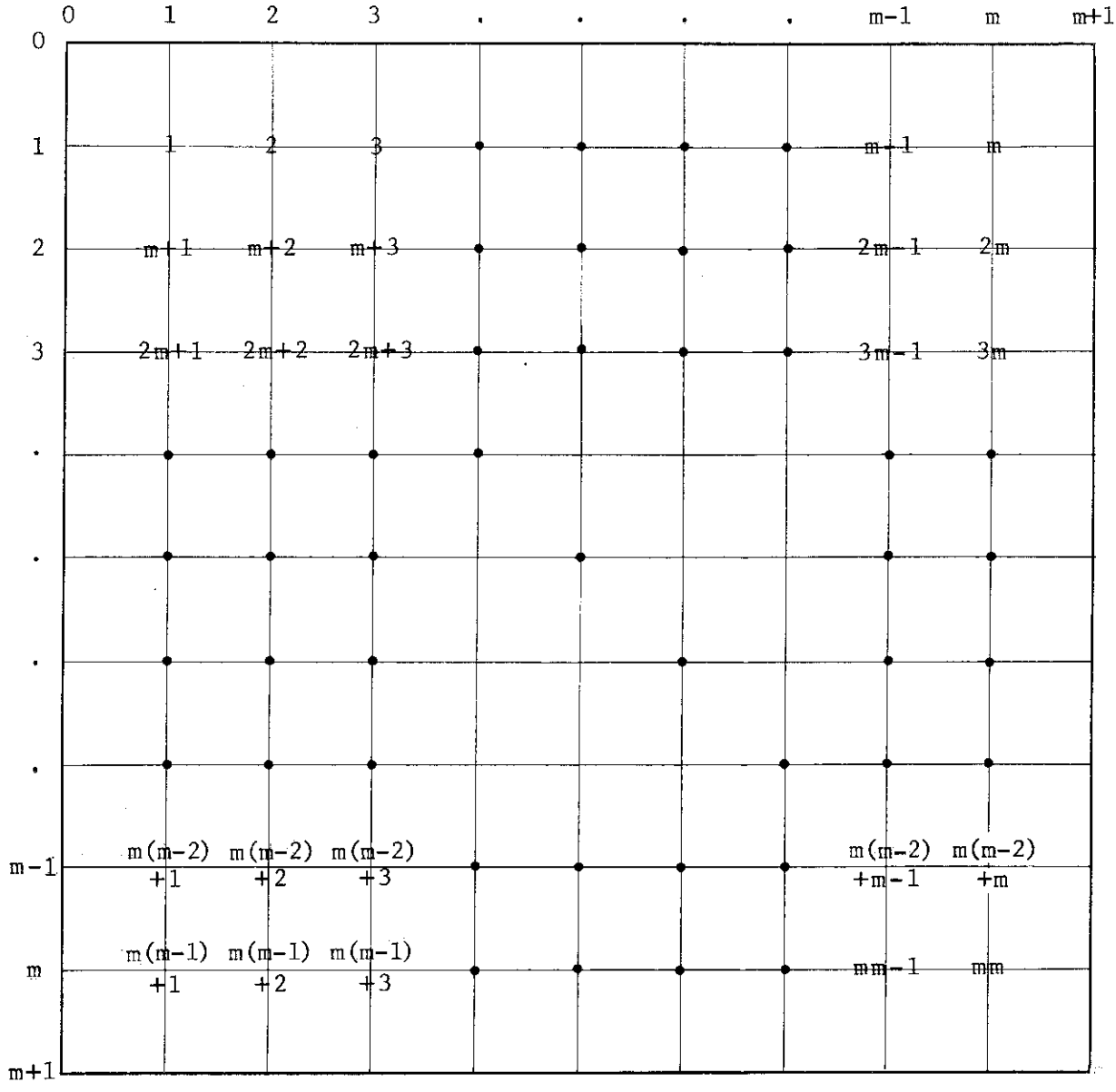
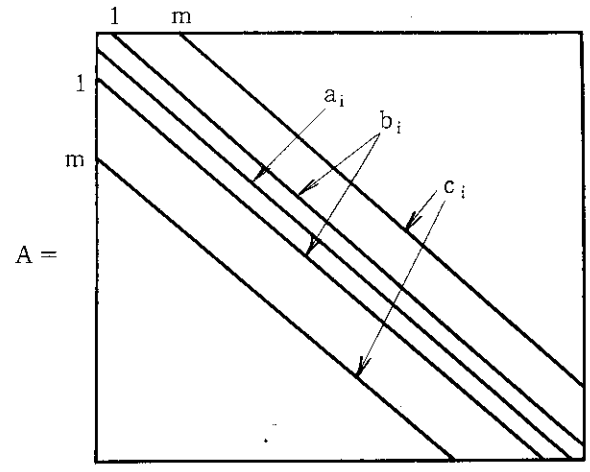


Fig. 3.14 Area Mesh-Division.



：対称正定値行列

Fig. 3.15　Coefficient Matrix for eq. 3.14 after Discretization.

このAに対しMeijerink等[17]はLを下に示すように定めた。AおよびLの非零要素をそれぞれ a_i, b_i, c_i および $\tilde{a}_i$, $\tilde{b}_i$, $\tilde{c}_i$ とし，Dの要素を $\tilde{d}_i$ とすると

$$\tilde{b}_i = \tilde{b}_i, \quad \tilde{c}_i = c_i$$

$$\tilde{a}_i = 1/\tilde{d}_i = a_i - (\tilde{b}_{i-1})^2 \tilde{d}_{i-1} - \tilde{c}_{i-m}\tilde{d}_{i-m} \qquad (3.5)$$

$i = 1, 2, \cdots n$，添字の値が0以下になる場合は各数値は0とする。

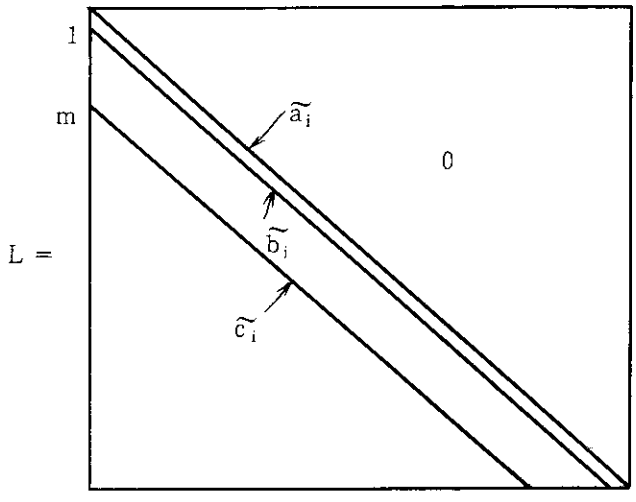


Fig. 3.16　L-Matrix for the Incomplete LDL^t Decomposition.

すなわちAの要素が0でない場合だけLの要素を定めるものである。この様にしてLDL^tが定まったのでFig. 3.13で示されるPCG法のアルゴリズムに従って計算を行うわけである。なおFig. 3.13で現われる $(LDL^t)^{-1}\vec{r}_i$ の計算はよく知られた前進消去後退代入の方法で以下のように行う。

$(LDL^t)^{-1}\vec{r}_i$の計算法

$(LDL^t)^{-1}\vec{r}_i = \vec{u}_i$ とする

⇓

$\vec{r}_i = LDL^t\vec{u}_i$ に変更

⇓

$DL^t\vec{u}_i = \vec{v}_i$ として $\vec{r}_i = L\vec{v}_i$ を解く

$L^t\vec{u}_i = \vec{w}_i$ として $\vec{v}_i = D\vec{w}_i$ を解く

$\vec{w}_i = L^t\vec{u}_i$ を解く

Fig. 3.17 Calculation Method of $(LDL^t)^{-1}\vec{r}_i$.

ICCG(3)ではICCG(0)で求めたLに，更に3つの非零要素を加えたもので，行列の形及び要素の計算法は以下に示す通りである。

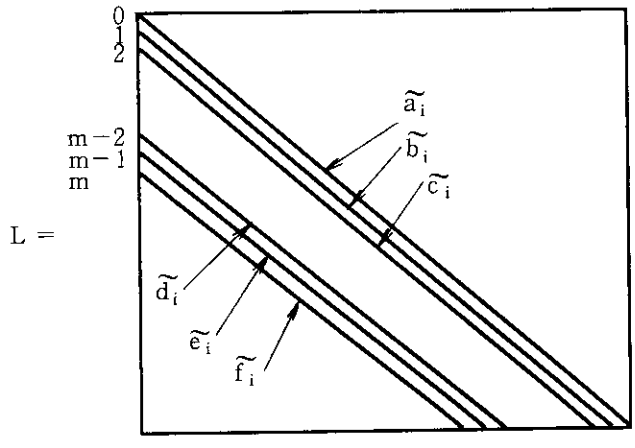


Fig. 3.18 L-Matrix for the Incomplete LDL^t Decomposition of the ICCG(3).

$$
\begin{aligned}
&\tilde{a}_i = \sqrt{a_i - \tilde{b}_{i-1}^2 - \tilde{d}_{i-2}^2 - \tilde{e}_{i-m+2}^2 - \tilde{f}_{i-m+1}^2 - \tilde{c}_{i-m}^2}\,,\\
&\tilde{b}_i = (b_i - \tilde{d}_{i-1}\tilde{b}_{i-1} - \tilde{c}_{i-m+1}\tilde{f}_{i-m+1} - \tilde{f}_{i-m+2}\tilde{e}_{i-m+2})/\tilde{a}_i\,,\\
&\tilde{d}_i = -\tilde{c}_{i-m+2}\tilde{e}_{i-m+2}/\tilde{a}_i, \quad \tilde{e}_i = -(\tilde{c}_{i-2}\tilde{d}_{i-2} + \tilde{f}_{i-1}\tilde{b}_{i-1})/\tilde{a}_i,\\
&\tilde{f}_i = -\tilde{c}_{i-1}\tilde{b}_{i-1}/\tilde{a}_i, \quad \tilde{c}_i = c_i/\tilde{a}_i \quad \text{for } i = 1, 2, \ldots, n.
\end{aligned}
\tag{3.6}
$$

なお Fig. 3.12, 3.13 における解の収束判定には次の条件を用いる。

$$\frac{|A\vec{x}_i-\vec{b}|}{|\vec{b}|}<\varepsilon \qquad (3.7)$$

ICCG(0)及び ICCG(3)は，Meijerink 等[19]及び村田[20]の報告に依れば他の反復解法である SOR や SLOR に比べ値の収束が極めて速くなっている。Meijerink 等に依れば反復1回当りの計算は ICCG(0)では SLOR に比べ約3倍弱，ICCG(3)では4倍弱となっている。しかし，実際に計算機でプログラムを実行する場合には，どの解法を採用するかについては反復回数と反復当りの演算数を共に考慮する必要があろう。

最近では ICCG法を改良したものや，非対称係数行列を持つ多元連立方程式の解法に CG 法に似た方式を採用しているものなど，多種多様の計算法が提案されている。これについては村田[20]の報告を参考にされたい。

さて CG 法及び ICCG 法で書かれたプログラムのベクトル化であるがCG法は再帰性のない行列演算のみで構成されているので，その殆んど全ての計算過程がベクトル化される。ICCG 法では次の2つの計算が再帰的となる。

① LDL^t 分解　　（(3.3)式参照）

② $(LDL^t)^{-1}\vec{r}$ の計算（Fig.3.17 参照）

ICCG(3)ではこれらの計算のベクトル化は複雑なので ICCG(0)の場合について考える。

LDL^t 分解を行う際問題となるのは式3.5で $\tilde{a}_i$ あるいは $\tilde{d}_i$ の計算についてである。これらは i 番目の値を計算するのに i－1番目と i －m番目の $\tilde{d}_i$ の値を必要とする。また $(LDL^t)^{-1}\vec{r}$ の計算でも同様である（ただし L と L^t についてのみ）。

これらの問題は各々の計算をリストベクトルを用いメッシュ点に関して Fig. 3.19 のように走査することによりベクトル化できる。

このメッシュ走査方法は Hyper Plane 法と呼ばれている（3.1節参照）。ベクトル長が 1～m と変化するが mが大きい場合には有利である。

CG 法を有限要素法に見られるような不規則格子に対し用いる場合には，先に示した LDL^t 分解及び $(LDL^t)^{-1}\vec{r}$ の計算における再帰性が複雑になる，すなわちAを用いた計算に複雑な再帰性が現われるのでベクトル化は難かしくなる。しかしメッシュ点の走査法をうまくすると，ベクトル長は不規則に変化するが，リストベクトルを用いることによりベクトル化できる。有限要素法では対象とする構造物の形がしばしば変るので各々の場合に対し対策が必要である。

3.3　モンテカルロ法

モンテカルロ法を用いた計算には種々の形態があり，利用範囲も積分問題や微分問題あるいは大規模粒子シミュレーションなど多岐に渡っている。複雑な構造を持つモンテカルロプログラムでは巨大な DO ループの中に何重ものＩＦ文のループや多くのサブルーチンコールがあり，これらがベクトル処理の妨げとなっている場合が多い。

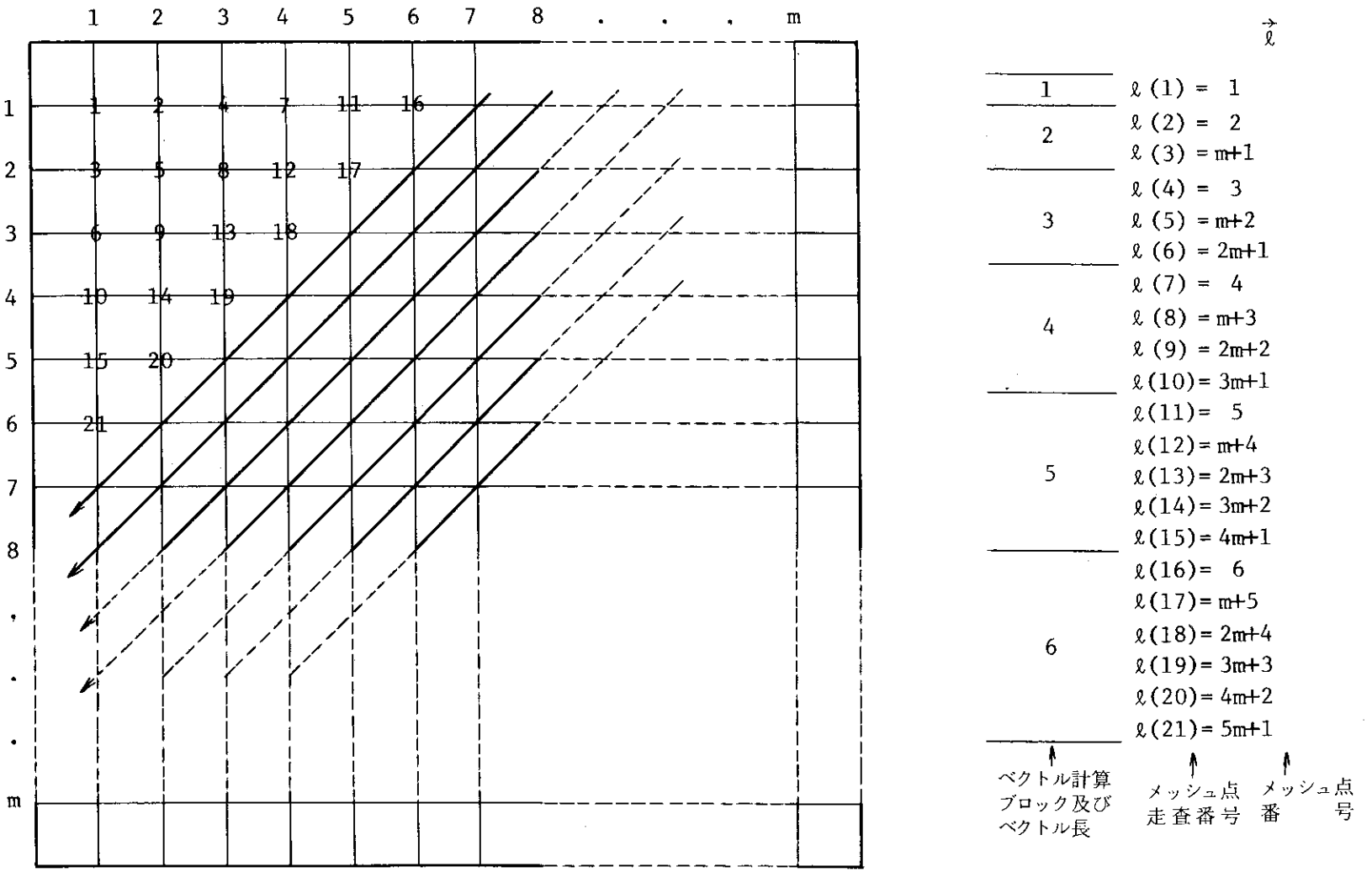


Fig. 3.19 Vectorization by the Hyper-Plane Method.

モンテカルロ計算プログラムは殆んどの変数が単精度で構成されている場合が多い。これはモンテカルロ計算自体の持つ精度に因るものである。例えば通常の単精度変数の精度10^{-6}を達成しようとするとモンテカルロ計算の試行回数Nは$1/\sqrt{N}\sim10^{-6}$よりN～10^{12}となり，実際の計算は困難である。元よりモンテカルロ計算は高精度計算の為ではないので一般に単精度で行われるのが常である。VP100のようなベクトル計算機はレジスタ構成その他が全て倍精度計算向きに作られているので，モンテカルロプログラムをリストベクトルを多用してベクトル化した場合メモリアクセスの関係で不利な点が出て来ることもある（文献(21)参照)。

仮想的なモンテカルロプログラムの計算の流れをFig. 3.20に示す。

大規模粒子シミュレーションプログラムにおいては粒子のバッチ計算と呼ばれるものがよく使われる。1バッチでは大体数百個から数千個の粒子に対して計算が行われ，バッチ終了後各種の物理量が計算される。1バッチはプログラムの中における反復計算（iteration）の1回分に当り，バッチ終了後の各種計算値が収束するまでプログラムが実行される。この形式のプログラムにおいては異なるバッチに属する粒子は原則として一緒に計算することはできないので，通常は1バッチ中の粒子に対する計算がベクトル化の対象となる。1バッチ中の粒子が充分多い（数千≦）場合にはベクトル化の効果は充分期待できる。またプログラムによっては異なるバッチを複数個一緒に計算できる場合もあるので，この際はもっとベクトル化の効果が上がることが予想される。

モンテカルロ計算プログラムのベクトル化に際し留意することは以下に述べるようなことである。

① 巨大なDOループはできるだけ避ける。
② ＩＦ文の構成をできるだけ簡単にする。
③ DOループ中のサブルーチンコールを極力減らす。あるいはサブルーチンによる計算をベクトル化するようにする。
④ 乱数をまとめて発生させる。

②に関してはＩＦ－THEN－ELSEを用いて書き直すと，ＩＦ文が整理されて有用である。乱数はDOループの外側でまとめて必要な数だけ発生させるが，この際には科学技術計算用サブルーチンライブラリSSLⅡ／VPに含まれる乱数発生関数RANU2などを用いるようにする。なおRANU2による乱数の値は単精度であるので倍精度変数を引数すると計算結果がおかしくなったり，エラーが起きてプログラムの実行が停止したりするので注意が必要である。

前述の様に，ＩＦ文をできるだけ回避する為にリストベクトル（インデックスベクトル）を用いることがしばしばある。このリストベクトルを用いて単精度配列変数の要素を連続に引用すると，メモリアクセスコンクリクトの状態に陥り，思った通りの速度向上が得られない場合がある。これを回避する方法は4章で述べる。また複雑なモンテカルロプログラムをベクトル化した例，及びその場合に用いた種々の手法が文献(21)に述べられている。

モンテカルロ法を用いた大型原子力コードのベクトル化の例としてKENOⅣコードをベクトル化した浅井らの報告[22]があるので参考にされたい。

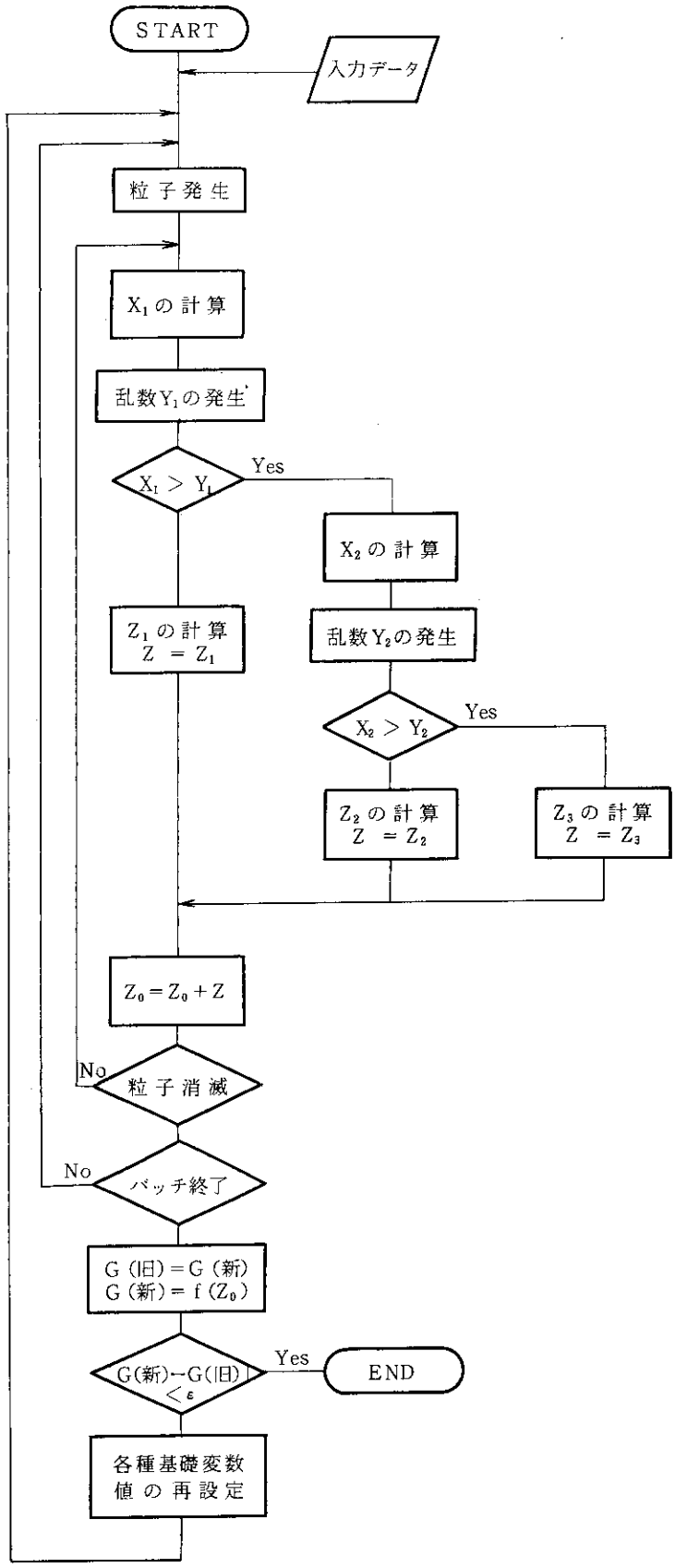


Fig. 3.20 Calculation Flow of an Attempt Monte Carlo Program Having Relatively Simple Structure.

3.4 数値積分法

定積分を行う場合に不定積分の形で計算を行うことが必ずしも常に有利であるとは限らない。不定積分は常に可能であるとは限らないし，可能であっても実際に不定積分後の式を使って計算を行うよりは近似式を用いた数値積分法の方が迅速に処理でき，また充分な精度を持っている場合も多い。

数値積分法には大別して以下の3種類がある。

- ニュートン・コーツ系（Newton－Coats）
- ガウス・チェビシェク系（Gauss－Tschebyscheff）
- モンテカルロ法

モンテカルロ法は他の2つの数値積分法とはかなり異なり，被積分関数の形や複雑さに関わらず常に同じ方法を用いている。また2次元や3次元，あるいはもっと多次元の積分に関しても同じ方法で行うことができる。このモンテカルロ法による積分プログラムのベクトル化は所謂モンテカルロ計算のベクトル化の中で最も単純な部類に属する。モンテカルロ計算のベクトル化については3.3節を参照されたい。ニュートン・コーツ系及びガウス・チェビシェフ系の数値積分法はプログラム作成方法に関しては殆んど同じである。ここではこの2つの積分法について簡単に説明し，具体的なプログラムのベクトル化の例として有限要素法で良く使われる8節点要素におけるガウスルジャンドル積分プログラムのベクトル化について述べる。

ニュートン・コーツ系の数値積分法は積分区間を等分割して行うもので，分割数の少ない台形公式やシンプソンの公式がよく用いられる。

〔台形公式〕；区間を分割せず

$$\int_{x_0}^{x_1} f(x)\,dx \fallingdotseq \frac{h}{2}(f_0+f_1),\quad h=|x_1-x_0| \tag{3.8}$$

〔シンプソン則〕；区間を2等分

$$\int_{x_0}^{x_2} f(x)\,dx \fallingdotseq \frac{h}{3}(f_0+4f_1+f_2),\quad h=|x_1-x_0|=\frac{1}{2}|x_2-x_0| \tag{3.9}$$

ただし，$f_0=f(x_0)$，$f_1=f(x_1)$ などである。

〔シンプソン$\frac{3}{8}$則〕；区間を3等分

$$\int_{x_0}^{x_3} f(x)\,dx \fallingdotseq \frac{3h}{8}(f_0+3f_1+3f_2+f_3),\quad h=|x_2-x_1|=\frac{1}{3}|x_3-x_1| \tag{3.10}$$

通常はこれらの計算を連続する小区間に対し適用する場合が多いが，良く知られていることなのでこれについての説明は省く。

ニュートン・コーツ系の数値積分法を用いる場合に注意すべきことがいくつかある。1つは必ずしもシンプソン則による積分が台形則よりも優秀であるとは限らないことである。例えば周期関数の積分や単調増加・単調減少の関数の積分などにおいては却って台形則の方が良い結果の得られる場合もある。ニュートン・コーツ系の積分則は原理的に区間を何等分にもして行える。また領域の両端の値を含む方式（Newton－Coats Close）と含まない方式（Newton－Coats open）の2種類がある。上に示した台形則とシンプソン則はこの前者の方に含まれる。Newton－Coats

Close 法では区間を8等分以上，open 法では4等分以上の場合で積分の重みがすべて同一符号(+)でない場合が出てくる。例えば

〔Newton－Coats open〕；区間を4等分

$$\int_{x_0}^{x_4} f(x)\,dx \fallingdotseq \frac{4h}{3}(2f_1 - f_2 + 2f_3),\quad h = |x_1 - x_0| = \frac{1}{4}|x_4 - x_0| \qquad (3.11)$$

重みがすべて同符号でないと桁落ちなどで計算精度が悪くなる場合があるので注意すべきである。ニュートン・コーツ系及びガウス・チェビシェフ系の積分公式については文献[23]に詳しく紹介してあるので参照されたい。

ガウス・チェビシェフ系の積分則は前述の通りプログラミング上の扱いは，ニュートン・コーツ系と殆んど同じである。異なる点は計算に用いるサンプリング点を区間の等分割点にするのではなく，計算に用いる直交関数の零点にとること，また各点における重みもこの直交関数を利用して定めることなどである。ガウス・チェビシェフ系の積分則に利用する直交関数には，実際に積分の対象とする関数の形に応じて精度良く計算する為に種々の関数が用意されている（文献[23]参照）。

このガウス・チェビシェフ系の積分則で最もよく用いられるものに，直交関数としてルジャンドル関数を用いるガウス・ルジャンドル積分がある。

具体的な例として有限要素法でしばしば用いられる8節点要素における2次元ガウス・ルジャンドル積分のプログラムのベクトル化について解説する。

〔8節点要素〕

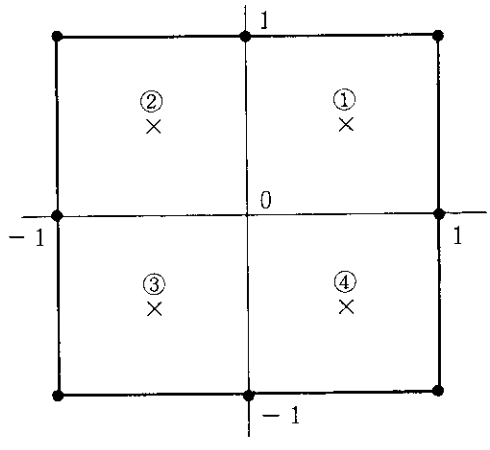


Fig. 3.21　8-Nodes Element in the FEM.

上図に示される積分点の座標値 0.57735，－0.57735 は2次のルジャンドル関数 P_2 の零点である。P_2 のルジャンドル関数を用いた場合の各点の重みは1である。多くの場合このような8節点要素N個に対し積分を行うので，実際の計算は以下のようになる。

$$S(j) = \sum_{i=1}^{4} W(i)\,f(i,j),\quad j = 1, 2, \cdots, N \qquad (3.12)$$

W(i), f (i, j) はそれぞれ各点における重みと値

Fig. 3. 22 にこの計算をベクトル化した場合の例を 4 つ示す。
往々にして上述の式（3.12）で計算された各 S(j) が 2 次元的に配列されていて S（ℓ, m）となっており，各要素からの寄与を寄せ集める計算を行う場合がある。

$$TS = \sum_{\ell=1}^{N1} \sum_{m=1}^{N2} S(\ell, m) \qquad (3.13)$$

N 1, N 2 がそれ程大きな数でない場合には Fig.3. 23 に見られるように 2 重の DO ループをリストベクトルを作って一重化しベクトル長を大きくすることがしばしば行われる。リストベクトルを作る為に余計な計算が加わるので Fig. 3. 23 に見られるようなプログラムの変更だけでは大きな計算速度向上が得られない場合もある。しかし有限要素法では Fig. 3. 23 のような計算を何回も反復して行うケースが多いので，リストベクトルは 1 回作製して保存しておくか予めデータ文で定義しておくなどで，無駄な計算を省くことができ，高速にベクトル計算できる。

```
          DIMENSION S(100),W(4),F(4,100)
          N=100
  C
  C
S         DO 2 J=1,N                                    例1　スカラ計算及びベ
S           P = 0.0                                     クトル計算共に効
V           DO 1 I=1,4                                  率の最も悪いプロ
V             P = P + W(I)*F(I,J)                       グラミング。
V       1   CONTINUE
S           S(J) = P
S       2 CONTINUE
  C
          END
```

```
          DIMENSION S(100),W(4),F(4,100)
          N=100
  C                                                     例2　短いベクトル長の
  C                                                     最内DOループが
S         DO 2 J=1,N                                    ベクトル化されて
S           S(J) = 0.0                                  おりベクトル計算
V           DO 1 I=1,4                                  向きでない。
V             S(J) = S(J) + W(I)*F(I,J)
V       1   CONTINUE
S       2 CONTINUE
  C
          END
```

```
          DIMENSION S(100),W(4),F(4,100)
          N=100
  C
  C
V         DO 1 J=1,N                                    例3　ベクトル長の大きい
V           S(J) = 0.0                                  外側のDOループが
V       1 CONTINUE                                      ベクトル化されてい
  C                                                     るのでベクトル計算
V         DO 3 J=1,N                                    向き。
S           DO 2 I=1,4
V             S(J) = S(J) + W(I)*F(I,J)
S       2   CONTINUE
V       3 CONTINUE
  C
          END
```

```
          DIMENSION S(100),W(4),F(4,100)
          N=100
  C
  C                                                     例4　スカラ計算及びベク
V         DO 1 J=1,N                                    トル計算共に最速の
V           S(J) =    W(1)*F(1,J)                     プログラミング。
         &          + W(2)*F(2,J)
         &          + W(3)*F(3,J)
         &          + W(4)*F(4,J)
V       1 CONTINUE
  C
          END
```

Fig. 3.22　Vectorization of a Sample Program in a FEM Code.

```
        SUBROUTINE SUB1(S,N1,N2,TS)
        DIMENSION S(1,1)
  C
        TS = 0.0
S       DO 1 M=1,N2
V         DO 2 L=1,N1
V           TS = TS + S(L,M)
V     2   CONTINUE
S     1 CONTINUE
  C
        END
```

```
        SUBROUTINE SUB1(S,N1,N2,TS,STEMP)
        DIMENSION S(1,1),STEMP(1)
  C
        ID = 0
S       DO 1 M=1,N2
V         DO 2 L=1,N1
V           ID = ID + 1
V           STEMP(ID) = S(L,M)
V     2   CONTINUE
S     1 CONTINUE
  C
        TS = 0.0
V       DO 3 K=1,ID
V         TS = TS + STEMP(K)
V     3 CONTINUE
  C
        END
```

→ 通常の２重DOループの一重化

```
        SUBROUTINE SUB1(S,N1,N2,TS,LTEMP,MTEMP)
        DIMENSION S(1,1),LTEMP(1),MTEMP(1)
  C
        ID = 0
S       DO 1 M=1,N2
V         DO 2 L=1,N1
V           ID = ID + 1
V           LTEMP(ID) = L
V           MTEMP(ID) = M
V     2   CONTINUE
S     1 CONTINUE
  C
        TS = 0.0
V       DO 3 K=1,ID
V         L = LTEMP(K)
V         M = MTEMP(K)
V         TS = TS + S(L,M)
V     3 CONTINUE
  C
        END
```

→ リストベクトルは一度作ったら保存しておく。

Fig. 3.23 Vectorization of a Summation Program in a FEM Code.

4. ベクトル計算向きのコーディング例

本章ではプログラムのベクトル化について具体的な例を示すが，先ずベクトル化の際に注意すべき基本事項を列挙することにする。

4.1 ベクトル化プログラミングの基本事項

以下に示す事柄はプログラムのベクトル化に当って最も基本的なものである。

① プログラムは論理が明解なように書く。IF－THEN－ELSEをGO TO文の代りに使用することにより見易くなる。誰が見ても論理の流れが良く分るプログラムは実際に計算機で実行しても効率が良いことが多い。

② 巨大なDOループを作らない。巨大なDOループはコンパイラにとっても大きな負担となり，計算時のベクトルレジスタの使い方に無理が生じやすい。VPコンパイラ自体はどんなに大きなDOループもベクトル化するが，①と同様に人間が理解し易いような長さ（1ページ≒60行程度）にしておくことが望ましい。

③ DOループ内にベクトル化されないような部分をなくす。一部でもDOループ内にベクトル化されない部分があるとベクトル処理がそこで一旦途切れるので連続的な滑らかなベクトル処理ができなくなる。

④ どうしてもベクトル化できないサブルーチンコールはサブルーチンを上位展開するか，リストベクトルによりDOループの情報をサブルーチンに持ち込むことによりベクトル化する。

上記以外の細かな留意事項は前述のベクトル化プログラミングの手引[3]やプログラミングハンドブック[24]を参照されたい。また4.2節の中にも細かな事柄は散見されるであろう。

4.2 サブルーチン・DOループのベクトル化例

本節ではプログラムのベクトル化の具体的な例を示す。VPコンパイラは自動ベクトル化機能を持っており，これにより効率良くベクトル化されるものについては，ここでは述べない。これらに関しては，ベクトル化プログラミングの手引[3]やプログラミングハンドブック[24]を参考にして頂きたい。本章では現実の問題や実際のプログラムに即して，VPコンパイラでは簡単にはベクトル化されないものを示すことにする。ただしプログラム全体や大規模なサブルーチンを載せるのは無理があるので，ここでは比較的小さなサブルーチンや大きなサブルーチンの一部を取り出して解説を行う。なお本節の中で実際のプログラムを紹介した図の中で左端にV, M, Sなどの記号が現われるがこれらの意味は以下の通りである。

Table 4.1 Description of Symbols.

記　号	意　　味
V	ベクトル処理されるFORTRAN文
M	1部ベクトル処理される　〃
S	スカラ処理される　〃

4.2.1 スカラ変数を定義するIF文を含むDOループのベクトル化

Fig. 4.1に示すオリジナルプログラムのDOループではIF文の条件を満たすことがあればスカラ変数IRHSが1となる。スカラ変数だけでなく，条件を満たす時にフラグをオンにしようとする場合も同様にベクトル化されない。このIF文がベクトル化されない原因はスカラ変数の定義を配列変数を引用するIF文と組み合わせて行うことにある。スカラ変数の定義だけならば通常はコンパイラの最適化によりDOループの外で計算される。

オリジナル　　変　更　後

```
S         DO 130 I=2,IRMAXM                    V         DO 130 I=2,IRMAXM
S         IF(DABS(RHS(I)).GT.1.D5) IRHS=1  ⇒   V         IF(DABS(RHS(I)).GT.1.D5) GO TO 131
S   130   CONTINUE                             V   130   CONTINUE
                                                         GO TO 132
                                                   131   IRHS=1
                                                   132   CONTINUE
```

Fig. 4.1 Vectorization of an Example DO-Loop Including an IF-Statement.

変更後のDOループではGOTO文と組み合わせることによりベクトル化を行っている。なおVPコンパイラではGOTO文によるDOループからの飛びだしは1回のみ許される。DOループから飛びだすIF文が2個以上ある場合はベクトル化されない。Fig. 4.1における変更は今のところ，この形式のDOループに関しては最も速いと思われる。この変更方法の他にFig. 4.2に示す2通りの方法が考えられるがこのDOループのみの場合を考える限り最も良い方法ではない。

```
          IRHS = 0                                        RHSTMP=0.0
V         DO 130 I=2,IRMAXM                         V     DO 130 I=2,IRMAXM
V         IF(DABS(RHS(I)).GT.1.D5) IRHS=IRHS+1      V     RHSTMP=DMAX1(RHSTMP,DABS(RHS(I)))
V   130   CONTINUE                                   V 130 CONTINUE
          IF(IRHS.NE.0) IRHS=1                            IF(RHSTMP.GT.1.D5) IRHS=1
```

Fig. 4.2 Other Vectorization Examples for the DO-Loop in Fig. 4.1.

4.2.2 DOループの分割とリストベクトルによるベクトル化

下記のオリジナルのDOループには3つ問題があった。

① 2つのIF文はまったく違う条件であるに関わらず常に評価される。

② 2番目のIF文の条件は実際はほぼ100％満たされる（正常の計算の場合は100％）のにこのIF文以下の式の為にベクトル化されない。

③ N＝NCR(J)の式により変数APVのインデックスが再帰的になる可能性があることからDOループが全ったくベクトル化されない。

オリジナル

```
S        DO 6000 J=1,NN2
S        IF (NCR(J).EQ.2)  APV(J)=0.0
S        IF (NCR(J).LE.2)  GO TO 6000
         N = NCR(J)
S        APV(N) = APV(N)+DU(J)*APV(J)
S        APV(J) = 0.0
S        PV(J)  = 0.0
S   6000 CONTINUE
```

⇒

変　更　後

```
         IFGN=0
V        DO 5950 J=1,NN2
V          IF(NCR(J).EQ.2) THEN
V            APV(J)=0.0
V          ELSEIF(NCR(J).GT.2) THEN
V              IFGN=IFGN+1
V              IFGNCR(IFGN)=NCR(J)
V              IFGJ(IFGN)=J
V          ENDIF
V   5950 CONTINUE
M          DO 6000 JJ=1,IFGN
V            N = IFGNCR(JJ)
V            J = IFGJ(JJ)
M            APV(N) = APV(N)+DU(J)*APV(J)
S            APV(J) = 0.0
V            PV(J)  = 0.0
V   6000   CONTINUE
```

Fig. 4.3　Vectorization of a Sample DO-Loop.

まず2つのIF文をIF－THEN－ELSE文を使って書き変えることによりプログラムをベクトル化する。Fig. 4.3のようにDOループを分割すると変数APVのインデックスの再帰関係に変化が生ずるがこれが計算結果に影響するかどうかはプログラム作成者がチェックしなければいけない。変更後のリストベクトルを使用したDOループ（DO 6000）は再帰関係をなくすことができないのでこのまま残しておく。このDOループは実際は実行されない（IFGN＝0）のでオリジナルのDOループ全体は極めて高速化されたことになる。

オリジナル

```
      SUBROUTINE XXXXXX (AP,A,P,KII,IOSM,NN,NB)
C
      DIMENSION A(2,2,NB),AP(2,NN),P(2,NN),KII(NN),IOSM(NB)
C
      CALL RCLEAR (AP,2*NN)
      K1 = 1
C
S     DO 5000 NJ=1,NN
S     K2 = KII(NJ)-1
S     IF (K1.GT.K2)  GO TO 3000
S     DO 2000 K=K1,K2
S     NI = IOSM(K)
      DO 1000 J=1,2
      DO 1000 I=1,2
      AP(J,NJ) = AP(J,NJ)+A(J,I,K)*P(I,NI)
      AP(J,NI) = AP(J,NI)+A(I,J,K)*P(I,NJ)
 1000 CONTINUE
S 2000 CONTINUE
S 3000 K = K2+1
      DO 4000 J=1,2
      DO 4000 I=1,2
      AP(J,NJ) = AP(J,NJ)+A(I,J,K)*P(I,NJ)
 4000 CONTINUE
S     K1 = K2+2
S 5000 CONTINUE
C
      RETURN
      END
```

⇒

変 更 後

```
      SUBROUTINE XXXXXX (AP,A,P,NN,NB)
C
      COMMON /INDEX/ INI(5000),INJ(5000),IK(5000),IKK(5000),
     &               INI2(5000),INJ2(5000),IK2(5000)
      COMMON /IC/ ID1,ID2,KS
      DIMENSION A(2,2,NB),AP(2,NN),P(2,NN)
C
      CALL RCLEAR (AP,2*NN)
C
S     DO 1100 IX=1,KS
 *VOCL LOOP,NOVREC
V     DO 1000 I=IX,ID1,KS
V     K=IK(I)
V     NJ=INJ(I)
V     NI=INI(I)
V     AP(1,NJ)=AP(1,NJ)+A(1,1,K)*P(1,NI)+A(1,2,K)*P(2,NI)
V     AP(1,NI)=AP(1,NI)+A(1,1,K)*P(1,NJ)+A(2,1,K)*P(2,NJ)
V     AP(2,NJ)=AP(2,NJ)+A(2,1,K)*P(1,NI)+A(2,2,K)*P(2,NI)
V 1000 AP(2,NI)=AP(2,NI)+A(1,2,K)*P(1,NJ)+A(2,2,K)*P(2,NJ)
S 1100 CONTINUE
 *VOCL LOOP,SCALAR
      DO 1500 I=1,ID2
      K=IK2(I)
      NJ=INJ2(I)
      NI=INI2(I)
      AP(1,NJ)=AP(1,NJ)+A(1,1,K)*P(1,NI)+A(1,2,K)*P(2,NI)
      AP(1,NI)=AP(1,NI)+A(1,1,K)*P(1,NJ)+A(2,1,K)*P(2,NJ)
      AP(2,NJ)=AP(2,NJ)+A(2,1,K)*P(1,NI)+A(2,2,K)*P(2,NI)
 1500 AP(2,NI)=AP(2,NI)+A(1,2,K)*P(1,NJ)+A(2,2,K)*P(2,NJ)
 *VOCL LOOP,NOVREC
V     DO 2000 I=1,NN
V     KK=IKK(I)
V     AP(1,I)=AP(1,I)+A(1,1,KK)*P(1,I)+A(2,1,KK)*P(2,I)
V 2000 AP(2,I)=AP(2,I)+A(1,2,KK)*P(1,I)+A(2,2,KK)*P(2,I)
C
      RETURN
      END
```

Fig. 4.4 Vectorization of a Subroutine for Computing the Correction Vector of the CG Method.

4.2.3　同一インデックスが周期的に出現する場合のベクトル化

有限要素法のプログラムにおいては同じインデックス（同じノード番号）を何度も引用して計算を行う場合が多い。また対象とする構造物のジオメトリーは計算の途中で変更されることはないのでノードのアクセス順序は常に同じである。Fig. 4.4で示してあるのは剛体の釣り合い方程式をCG法で解いているプログラムの中のサブルーチンである。このプログラムではさらに記憶領域の節約の為に，多元連立方程式の係数行列をスカイライン法により蓄えている。従ってオリジナルのプログラムでも既に，計算中のマトリクス要素のインデックスはFig. 4.4にある通りリストベクトルにより指定されている。前述の様にマトリクス要素のアクセス順序は常に同じなのでオリジナルのサブルーチンに見られる様に毎回多重のDOループを用いてリストベクトルで指定する必要はなく最初に1回だけ計算しておけば良い。Fig. 4.5に示してあるのがこの計算部分で，これは上位のサブルーチンでただ1回行われている。

```
      C
      C     THE ACCESS SEQUENCE OF MATRIX ELEMENTS IS ALWAYS SAME.
      C     THEREFORE, THE INDICES ARE CALCULATED ONLY ONCE, BEFORE
      C     THE CG-ITERATION.
      C
            K1 = 1
            ID1=0
            ID2=0
S           DO 100 NJ=1,NN
S           K2=KII(NJ)-1
S           IF(K1.GT.K2) GO TO 150
S           IF((K2-K1+1).EQ.KS) THEN
        *VOCL LOOP,SCALAR
                DO 200 K=K1,K2
                  ID1=ID1+1
                  IK(ID1)=K
                  INJ(ID1)=NJ
                  INI(ID1)=IOSM(K)
          200   CONTINUE
S           ELSE
        *VOCL LOOP,SCALAR
S               DO 300 K=K1,K2
                  ID2=ID2+1
                  IK2(ID2)=K
                  INJ2(ID2)=NJ
                  INI2(ID2)=IOSM(K)
          300   CONTINUE
S           ENDIF
S         150 IKK(NJ)=K2+1
S         100 K1=K2+2
```

Fig. 4.5　Preparation of Index-Vectors for the Vectorized Subroutine in Fig. 4.4.

このインデックスのリストベクトル作成はFig. 4.4のオリジナルのプログラムと違って2つの場合に分けられている。このプログラムではK2−K1+1≦KSであり，K2−K1+1=KSの場合が最も多かった（約96％）。Fig. 4.5からも分る様にリストベクトルINJは同じものがKS回続く。INIは同じものがやはりKS回現われるが不連続であった。しかしKSでMODを計算した場合同じインデックス番号に関してはMODの値は常に同じである。K2−K1+1＜KSの場合は少ないのでこの場合は全部まとめて行っている。なおFig. 4.5でDOループにVOCL行でベクトル化を抑制してあるのはここで用いた入力データではKSの値が小さい（＝3）為ベクトル化されても効果が出ないからである。

Fig. 4.5で作ったインデックスのリストベクトルを用いてFig. 4.4に示してある様に効率良く

ベクトル化することができる。DO 1000のDOループではインデックスをKS おきに引用することで再帰的関係を避けることができる。DO 1000とDO 2000のDOループにはVOCL 行で再帰的関係を否定しているが，これはインデックスをリストベクトルで指定するとインデックスの再帰的関係が陽に分らないのでコンパイラによりベクトル化されないからである。DO 1500のDOループではFig．4.5 で明らかなようにK2－K1＋1＜KSの場合でK2－K1＋1＝1,2が混っており再帰的関係が発生するからである。なお変更後のプログラムでは，オリジナルのプログラム中の短いDOループを書き下してある。短いDOループは書き下して計算した方が速くなることが多い。

4.2.4　中間変数の導入によるベクトル化

DOループ全体がベクトル化されず，DOループ中のFORTRAN文の何個かに対しVP コンパイラのベクトル化メッセージがSあるいはMになっている場合，このDOループを極めて効率良くベクトル処理することは難しい。これはこのベクトル化されないFORTRAN 文の為にパイプラインやベクトルレジスタがうまく使用できないからである。従ってDOループ中のプログラムラインはすべてベクトル化されていることが望ましい。

Fig．4.6に示してあるオリジナルの2重DOループの内側のループではX＝X＋DELXYがベクトル化されていない。これはスカラ変数Xに対し再帰的計算が行われているからである。この影響で次のFORTRAN文も完全にベクトル化されておらずベクトル化メッセージはMとなっている。これらをベクトル化するには中間変数を導入してスカラ変数Xの再帰的関係を解消する必要がある。Fig．4.6の中段に示してあるように

$$X=X+DELXY \Rightarrow XT=X+DELXY*IX$$

と書き変えることにより内側のDOループはすべてベクトル化されることが分る。この書き変えだけによりこの2重DOループはオリジナルより3倍程度速く処理されるようになった（ベクトル長100程度の時）。

Fig．4.6下段に示してある例は更にベクトル化率を向上させることを目指しており，新たに中間配列変数YTが導入されている。2重DOループの外側ループは分割されて一部分は独立したDOループで処理されている。このようにするとオリジナルの2重DOループは殆んどベクトル処理されることになる。なおこの下段の例で2重DOループの内側がベクトル化されず外側のループがベクトル化されているのは，DOループのインデックスＩＹでベクトル化すると各配列変数のメモリアクセスが連続となるのでデータアクセス時間が短縮されるからである。このような2重DOループのベクトル化はVP コンパイラにより自動的に計算に有利なように行われる。

オリジナルDOループ

```
         Y=START-DELXY
S        DO 400 IY=1,MPMAX
S          Y=Y+DELXY
S          X=START-DELXY
M          DO 400 IX=1,MPMAX
S            X=X+DELXY
M            T=2.0*PI*(FH*X+FK*Y)
V            Z=-T-TF
V            ZZ=Z-6.28318*INT(Z/6.28318)
V            ZZ=(ZZ+6.28318)-6.28318*INT((ZZ+6.28318)/6.28318)
V            IZ=INT(ZZ*3183.1)+1
V            XCOS=TACOS(IZ)
V            XSIN=TASIN(IZ)
             EMIKAI=CMPLX(XCOS,XSIN)
V            ETSM(IY,IX)=ETSM(IY,IX)+UN(IJ2)*EMIKAI
V            I2=I2+1
V            IJ1=IJ1+1
V 400    CONTINUE
```

⇓

```
         Y=START-DELXY
S        DO 400 IY=1,MPMAX
S          Y=Y+DELXY
           X=START-DELXY
V          DO 400 IX=1,MPMAX
V            XT=X+DELXY*IX
V            T=2.0*PI*(FH*XT+FK*Y)
V            Z=-T-TF
V            ZZ=Z-6.28318*INT(Z/6.28318)
V            ZZ=(ZZ+6.28318)-6.28318*INT((ZZ+6.28318)/6.28318)
V            IZ=INT(ZZ*3183.1)+1
V            XCOS=TACOS(IZ)
V            XSIN=TASIN(IZ)
             EMIKAI=CMPLX(XCOS,XSIN)
V            ETSM(IY,IX)=ETSM(IY,IX)+UN(IJ2)*EMIKAI
V            I2=I2+1
V            IJ1=IJ1+1
V 400    CONTINUE
```

⇓

```
         Y=START-DELXY
         X=START-DELXY
V        DO 395 IY=1,MPMAX
V          YT(IY)=Y+DELXY*IY
V    395 CONTINUE
V        DO 400 IY=1,MPMAX
S          DO 400 IX=1,MPMAX
             XT=X+DELXY*IX
V            T=2.0*PI*(FH*XT+FK*YT(IY))
V            Z=-T-TF
V            ZZ=Z-6.28318*INT(Z/6.28318)
V            ZZ=(ZZ+6.28318)-6.28318*INT((ZZ+6.28318)/6.28318)
V            IZ=INT(ZZ*3183.1)+1
V            XCOS=TACOS(IZ)
V            XSIN=TASIN(IZ)
             EMIKAI=CMPLX(XCOS,XSIN)
V            ETSM(IY,IX)=ETSM(IY,IX)+UN(IJ2)*EMIKAI
V 400    CONTINUE
         I2=I2+MPMAX*MPMAX
         IJ1=IJ1+MPMAX*MPMAX
```

Fig. 4.6 Vectorization of a DO-Loop in the Subroutine of Fourier Transformation.

4.2.5　最内DOループの書き下しと中間変数導入によるベクトル化

FORTRAN 77以前のFORTRAN言語を用いているプログラムではＩＦ－THEN－ELSEを用いた構造化プログラミングが施されておらず，GOTO文が多用されている為極めて見にくいプログラムとなっているのが普通である。Fig. 4.7に示してあるオリジナルのDOループもこの古いタイプのプログラムの典型である。このようなプログラムは計算ロジックの流れの把握が難しくまたベクトル化も容易でない。この様なプログラムは先ずＩＦ－THEN－ELSEにより書き変えて計算内容を良く理解した上でベクトル化を行うべきである。

Fig. 4.7下段に示す変更後のDOループでは以下の3点について変更がなされている。

- ベクトル長の小さい最内DOループの書き下し
- 計算ロジックの見直しとDOループの分割
- 中間配列変数の導入によるベクトル化

オリジナルプログラム中の最内DOループを書き下しただけではDO 290のループは完全にはベクトル化されない。これはオリジナルのＤＯループではスカラ変数TAUO，TAUI，ＳＩＧＭの再帰関係が原因である。この再帰関係は中間配列変数ＧＩＴＥＭＰとDOループの分割（DO 290 K＝1，NSTとDO 291 K＝2，NST）により回避することができる。このように一見ベクトル化不可能なように見えるDOループでも計算ロジックを詳細に検討するとベクトル化が可能な場合が非常に多い。

オリジナル DO ループ

```
      NNST=NST+1
S     DO 290 K=1,NNST
S     IF(K.EQ.NNST)                          GO TO 280
S     TAUI=TAUX(K)
S     IF(TAUI.GT.0.1D0)                        GO TO 240
S     GI=1.D0-4.D0*DSQRT(TAUI/PI)+1.5D0*TAUI
S     GO TO 260
  240 CONTINUE
S     TOT = 0.0D0
V     DO 250 N=1,3
V     TOT = TOT+DEXP(-TN2(N)*PI2*TAUI)/(TN4(N)*PI4)
V 250 CONTINUE
S     GI=1.D0/(15.D0*TAUI)-6.D0/TAUI*TOT
  260 CONTINUE
S     IF(K.EQ.1)                             GO TO 270
S     SIGM=SIGM+PD(K-1,I,NODE)*(TAUO*GO-TAUI*GI)
  270 CONTINUE
S     TAUO=TAUI
S     GO=GI
S     GO TO 290
  280 CONTINUE
S     SIGM= SIGM+PD(K-1,I,NODE)*TAUO*GO
S 290 CONTINUE
```

```
V     DO 290 K=1,NST
V       TAUI=TAUX(K)
V       IF(TAUI.LE.0.1D0) THEN
V         GITEMP(K)=1.D0-4.D0*DSQRT(TAUI/PI)+1.5D0*TAUI
V       ELSE
V         TOT =  DEXP(-TN2(1)*PI2*TAUI)/(TN4(1)*PI4)
      &          +DEXP(-TN2(2)*PI2*TAUI)/(TN4(2)*PI4)
      &          +DEXP(-TN2(3)*PI2*TAUI)/(TN4(3)*PI4)
V         GITEMP(K)=1.D0/(15.D0*TAUI)-6.D0/TAUI*TOT
V       ENDIF
V 290 CONTINUE
V     DO 291 K=2,NST
V       SIGM=SIGM+PD(K-1,I,NODE)
      &            *(TAUX(K-1)*GITEMP(K-1)-TAUX(K)*GITEMP(K))
V 291 CONTINUE
      SIGM=SIGM+PD(NST,I,NODE)*TAUX(NST)*GITEMP(NST)
```

Fig. 4.7 Vectorization of a DO-Loop in the Subroutine for Computing the Fission-Gas Release from Fuel Pellets.

4.2.6　サブルーチンの上位展開によるDOループのベクトル化

VPコンパイラの性能向上に伴い比較的小さなサブルーチンは条件が整えば自動的に上位展開されるようになるが，複雑なものは上位展開不可能なのでなるべくプログラム作成者自身の手で上位展開しておくのが賢明である。

Fig. 4.8上段に示してあるオリジナルのDOループではループ中に2回のFunctionコールがある為にこのDOループが効率良くベクトル化されていない。変更後図中下段に示すようにDOループ中に2つのFunctionを展開することによりDOループが完全にベクトル化されている。

オリジナルDOループ

```
M         DO 120   I=1,10
M         TEMP=0.5D0*(TPSTG(I)+TPSTG(I+1))
S         THEX = PTHEX( TEMP  ,TCOOL )
  C
S         URS = 0.0D0
S         IF(ICHI.EQ.0) URS = URSWEL(BU,TEMP,FDEN)
M         SUMURS = SUMURS + URS*RING
M         SUMTHX = SUMTHX + THEX*RING
V         SUMDEN = SUMDEN + DEN*RING
V         SUMSWL = SUMSWL + SWELLL*RING
  C
V    120 CONTINUE
```

⇓

```
V         DO 120   I=1,10
V           TEMP=0.5D0*(TPSTG(I)+TPSTG(I+1))
  CYT       THEX = PTHEX( TEMP  ,TCOOL )
V           THEX = PTHEX1(TEMP) - PTHEX1(TCOOL)
  C
V           URS = 0.0D0
  CYT       IF(ICHI.EQ.0) URS = URSWEL(BU,TEMP,FDEN)
V           IF(ICHI.EQ.0) THEN
V             TC = TEMP-110.0D0
V             URS = 0.4417646D0*DEXP(-1.645D4/TC)*BU*FDEN
V             IF(URS.GT.0.02D0)   URS=0.02D0
              ENDIF
V           SUMURS = SUMURS + URS*RING
V           SUMTHX = SUMTHX + THEX*RING
V           SUMDEN = SUMDEN + DEN*RING
V           SUMSWL = SUMSWL + SWELLL*RING
  C
V    120 CONTINUE
```

Fig. 4.8　Vectorization of a DO-Loop in the Subroutine for Computing the Gap Thermal Conductance of a Fuel Rod.

4.2.7　繰り返し回数の少ないDOループを含む三重DOループのベクトル化方法の比較

Fig. 4.9に示すサブルーチンは有限要素法を用いた構造解析用プログラムに含まれているもので，ここでは部分対称剛性マトリックスを計算している。このサブルーチンを最も効率良く実行する為のプログラムの変更方法は同図中下段に示すものである。この例では単に同図上段のオリジナルDOループの最外ループを書き下しただけであるが，この場合はDOループ内の演算量が少ない為この方式が最適となっている。この他にFig. 4.10に示すような2つの方法が考えられるがこのサブルーチンに関する限り最も良い方法ではない。

Fig. 4.10に示す2つの例では方法は違うが共にベクトル長を大きくすることを目的にして変更が加えられている。上段に示す例はリストベクトルを用いるものでＩＳ３の値が小さめ（この例ではＩＳ３≦20）の場合には下段に示すものより高速である。下段に示すものはリストベクトルは用いず，VPコンパイラによる2重DOループの一重化を利用するものである。この例のDO 100の2重DOループはコンパイラにより自動的に一重ループ化され，全部がベクトル化される。DO 10のループではBDTEMP（I，J）＝0.0とする余計な計算が加わっているが，IS3≧20では下段の例の方が上段のものよりも高速である。Fig. 4.10に示したこれら2つの書き変えの例は，DOループ内の演算量が少ない為にFig. 4.9下段の絡よりも計算効率は良くないのであるが，演算量が多くなってくるとどの変更方法が最も効率的かは一概に言えなくなってくる。特にFig. 4.10上段に示す例では4.2.3項で述べたようにリストベクトル作成は最初に1回だけやって保存しておけば良いので，何回も反復計算が行われる場合にはこれが最も良いという場合もあり得る。

オリジナルサブルーチン

```
      SUBROUTINE XXXXX
     1      (BD    ,B     ,ESM   ,GAREA ,RB    ,IS3   ,L           )
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION
     1       BD(IS3,4),B(IS3,4),ESM(IS3,IS3),GAREA(4)
C
      AR = GAREA(L)*RB
S     DO 100 K=1,4
S     DO 100 J=1,IS3
V     DO 100 I=J,IS3
V     ESM(I,J) = ESM(I,J) + BD(I,K)*B(J,K)*AR
V100  CONTINUE
C
      RETURN
      END
```

⇓

```
      SUBROUTINE XXXXX
     1      (BD    ,B     ,ESM   ,GAREA ,RB    ,IS3   ,L           )
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION
     1       BD(IS3,4),B(IS3,4),ESM(IS3,IS3),GAREA(4)
C
      AR = GAREA(L)*RB
S     DO 100 J=1,IS3
V     DO 100 I=J,IS3
V     ESM(I,J) = (   BD(I,1)*B(J,1)
     &             + BD(I,2)*B(J,2)
     &             + BD(I,3)*B(J,3)
     &             + BD(I,4)*B(J,4) ) * AR
V100  CONTINUE
C
      RETURN
      END
```

Fig. 4.9　Vectorization of a Subroutine for Preparing the Stiffness Matrix.

```
      SUBROUTINE XXXXX
     1      (BD    ,B     ,ESM   ,GAREA ,RB    ,IS3   ,L     ,BDTEMP)
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION
     1       BD(IS3,4),B(IS3,4),ESM(IS3,IS3),GAREA(4),
     &       BDTEMP(IS3,IS3)
C
      AR = GAREA(L)*RB
S     DO 10 J=1,IS3
V     DO 10 I=1,IS3
V       IF(I.GE.J) THEN
V         BDTEMP(I,J)=   BD(I,1)*B(J,1)
     &                  + BD(I,2)*B(J,2)
     &                  + BD(I,3)*B(J,3)
     &                  + BD(I,4)*B(J,4)
V       ELSE
V         BDTEMP(I,J)=0.0
V       ENDIF
V   10 CONTINUE
V     DO 100 J=1,IS3
V     DO 100 I=1,IS3
V       ESM(I,J) = ESM(I,J) + BDTEMP(I,J)*AR
V  100 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE XXXXX
     1      (BD    ,B     ,ESM   ,GAREA ,RB    ,IS3   ,L     ,
     &       IBDI  ,IBDJ   )
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION
     1       BD(IS3,4),B(IS3,4),ESM(IS3,IS3),GAREA(4),
     &       IBDI(IS3*IS3),IBDJ(IS3*IS3)
C
      ID=0
      AR = GAREA(L)*RB
S     DO 10 J=1,IS3
V     DO 10 I=J,IS3
V       ID=ID+1
V       IBDI(ID)=I
V       IBDJ(ID)=J
V   10 CONTINUE
*VOCL LOOP,NOVREC
V     DO 100 IJ=1,ID
V       I=IBDI(IJ)
V       J=IBDJ(IJ)
V       ESM(I,J) = (   BD(I,1)*B(J,1)
     &               + BD(I,2)*B(J,2)
     &               + BD(I,3)*B(J,3)
     &               + BD(I,4)*B(J,4) ) * AR
V  100 CONTINUE
C
      RETURN
      END
```

Fig. 4.10 Modification to Obtain Higher Vector Processing Efficiency (to Attain Larger Vector Length).

4.2.8　リストベクトルの使用と計算順序の見直しによるベクトル化

配列変数のインデックスをリストベクトルで指定しているような場合はコンパイラはこれを自動ベクトル化の対象としない。これは計算内容の再帰性が明らかでない為にコンパイラは結果を保証する安全な方法を選択するからである。Fig. 4.11 に示すオリジナルの DO ループでは一見リストベクトルがないように見えるが，実は配列変数 TSM のインデックス NN はリストベクトルとなっている。この DO ループでは実際は NN の値が同じであることがなかったので，単に VOCL 行で再帰性を否定することで 2 重 DO ループの内側ループはベクトル化される（Fig. 4.11 中段の例）。

オリジナル DO ループ

```
S        DO 110 I=1,IS3
S        NXI = NFI(I)
S        IF(NXI.EQ.0)                          GO TO 110
S        NPI = NPRO(I)
  C
S        DO 100 J=1,IS3
S        NXJ = NFI(J)
S        IF(NXJ.EQ.0)                          GO TO 100
S        IF(NXI.LT.NXJ)                        GO TO 100
S        NN  = NXJ-NXI+NPI
S        TSM(NN) = TSM(NN) + ESM(I,J)
S    100 CONTINUE
  C
S        TF(NXI) = TF(NXI) + F(I)
S    110 CONTINUE
```

⇓

```
S        DO 110 I=1,IS3
S        NXI = NFI(I)
S        IF(NXI.EQ.0)                          GO TO 110
S        NPI = NPRO(I)
  C
  *VOCL LOOP,NOVREC
V        DO 100 J=1,IS3
V        NXJ = NFI(J)
V        IF(NXJ.EQ.0)                          GO TO 100
V        IF(NXI.LT.NXJ)                        GO TO 100
V        NN  = NXJ-NXI+NPI
V        TSM(NN) = TSM(NN) + ESM(I,J)
V    100 CONTINUE
  C
S        TF(NXI) = TF(NXI) + F(I)
S    110 CONTINUE
```

⇓

```
         ID=0
  *VOCL LOOP,NOVREC
V        DO 10 I=1,IS3
V          IF(NFI(I).NE.0) THEN
V            ID=ID+1
V            IT(ID)=I
V            NXITMP(ID)=NFI(I)
V            TF(NFI(I))=TF(NFI(I))+F(I)
V          ENDIF
V     10 CONTINUE
  C
         II=0
S        DO 20 I=1,ID
V          DO 20 J=1,ID
V            IF(NXITMP(I).GE.NXITMP(J)) THEN
V              II=II+1
V              NNTEMP(II)=NXITMP(J)-NXITMP(I)+NPRO(IT(I))
V              ESMTMP(II)=ESM(IT(I),IT(J))
V            ENDIF
V     20 CONTINUE
  C
  *VOCL LOOP,NOVREC
V        DO 30 I=1,II
V          TSM(NNTEMP(I))=TSM(NNTEMP(I))+ESMTMP(I)
V     30 CONTINUE
```

Fig. 4.11　Vectorization of a DO-Loop for Computing Symmetric Stiffness Matrix.

しかし実はオリジナルのDOループは無駄なIF文の実行を含んでおり，これをリストベクトルの使用とDOループの分割により除くことができる。Fig. 4.11下段に示す例ではIF文が1つ減っている。またベクトル化も飛躍的に進んでいる。ここでもIF-THEN-ELSEでプログラムを書き変えることが計算アルゴリズムを見直すことに繋がり，無駄な計算を省くことに寄与している。なお下段に示す例でIDが大きい場合にはDO 30のループでの計算はDO 20のループに繰り込んでしまう方が効率が良い（下図参照）。

```
      ID=0
*VOCL LOOP,NOVREC
V     DO 10 I=1,IS3
V       IF(NFI(I).NE.0) THEN
V         ID=ID+1
V         IT(ID)=I
V         NXITMP(ID)=NFI(I)
V         TF(NFI(I))=TF(NFI(I))+F(I)
V       ENDIF
V  10 CONTINUE
C
S     DO 20 I=1,ID
*VOCL LOOP,NOVREC
V       DO 20 J=1,ID
V         IF(NXITMP(I).GE.NXITMP(J)) THEN
V           NN=NXITMP(J)-NXITMP(I)+NPRO(IT(I))
V           TSM(NN)=TSM(NN)+ESM(IT(I),IT(J))
V         ENDIF
V  20 CONTINUE
C
      RETURN
      END
```

Fig. 4.12 The Most Efficient Coding for the Original DO-Loop in Fig. 4.11 When ID is Large.

4.2.9 熱平衡計算コードにおける熱流総和計算DOループのベクトル化

(1) 概 要

過渡および定常時の熱伝導計算コードTRUMP 4で用いられている熱平衡計算部分のベクトル化事例を示す。

Fig. 4.13のオリジナル版に現れる特殊ノードに対する熱伝導補正計算

ERROR (N 1) =ERROR (N 1) +……

はN 1＝NODE 1 (N)において，N 1が同一値となる可能性があり，多重代入計算と呼ばれる。多重代入計算は，このため，コンパイラでは自動ベクトル化しない。

上記は熱平衡計算の一部分であり，熱の伝導は各ノード点に対し，それと直接的に結合が存在するノードからのみ生じるとの仮定のもとに計算がなされている。

Fig. 4.14のような結合関係があるとすれば，次の関係が成立する（Table 4.2）。

```
                C...CALC CORRECTIONS FOR SPECIAL NODES.
                      IF(KNOCK)4410,4410, 6185
1-------------S-6185 DO 4405  N = 1,NOCON
1             S       N1 = NOD1(N)
1             S       N2 = NOD2(N)
1             S       IF(NTYPE(N1)*NTYPE(N2)) 6190,4405, 6190
1               6190 CONTINUE
1               C...BOTH MUST BE SPECIAL.
1             S       ERROR(N1) = ERROR(N1) + TRAN(N)*ERRORX(N2)
1             S       ERROR(N2) = ERROR(N2) + TRAN(N)*ERRORX(N1)
+-------------S4405  CONTINUE
```

Fig. 4.13　Original Heat-Flow Summation DO-Loop.

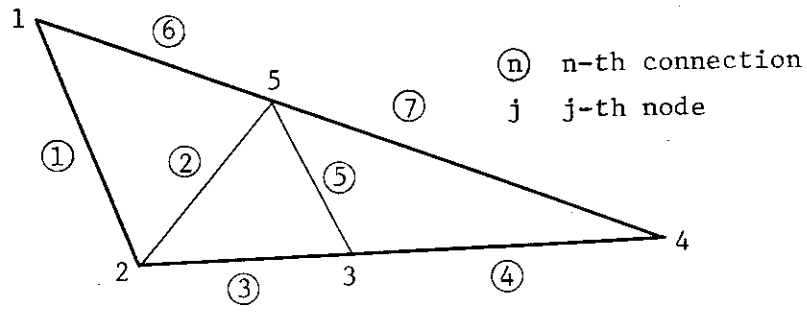


Fig. 4.14　Nodes and Connections.

Table 4.2　Connection Index Between Node 1 and Node 2.

結合インデックス	Node 1	Node 2
①	1	2
②	2	5
③	2	3
④	3	4
⑤	3	5
⑥	1	5
⑦	5	4

Fig. 4.13 のコーディングでは，結合インデックスをN，その両端ノードをN 1＝NODE 1 (N)，N 2＝NODE 2 (N)で求めている。

(2) ベクトル化方法

結合インデックスを，次のようにノード1，ノード2に対し各々3分割する。

N { N 11　はじめて現れたノード1番号を持つ結合インデックス　(226)
  { N 12　2回目に現れた　〃　(158)
  { N 13　3回目以降に現れた　〃　(41)

N { N 21　はじめて現れたノード２番号を持つ結合インデックス　(285)
　 N 22　２回目に現れた　〃　(140)
　 N 23　３回目以降に現れた　〃　(0)

( ) 内は，ノード数290，結合数425を持つ事例に対する分類後の結合数

上記のように結合インデックスを分類すると，N 11，N 12各グループ内では，もはやNODE 1(N)が同じ数字になることはない。N 21，N 22についても同様のことが言える。

(3) ベクトル化事例

上記の分類に従ってDOループを６分割し，N 11，N 12．N 21，N 22をベクトル計算，N13 N 23を従来どおりのスカラー計算（加算のみスカラー）するように変更する(Fig．4.15)。

このように，ベクトル化不可能とされる多重内積代入文も，このようにベクトル化できる場合がある。

```
  C...CALC CORRECTIONS FOR SPECIAL NODES.
        IF(KNOCK)4410,4410, 6185
  C  MODIFIED BY M. ISHIGURO FOR VECTORIZATION
  *VOCL LOOP,NOVREC
V  6185 DO 44051 NN= 1,I11
V       N=N11(NN)
V       N1 = NOD1(N)
V       N2 = NOD2(N)
V       IF(L1(N))
  C...BOTH MUST BE SPECIAL.
      1ERROR(N1) = ERROR(N1) + TRAN(N)*ERRORX(N2)
V 44051 CONTINUE
  *VOCL LOOP,NOVREC
V       DO 44052 NN= 1,I12
V       N=N12(NN)
V       N1 = NOD1(N)
V       N2 = NOD2(N)
V       IF(L1(N))
  C...BOTH MUST BE SPECIAL.
      1ERROR(N1) = ERROR(N1) + TRAN(N)*ERRORX(N2)
V 44052 CONTINUE
S       DO 44053 NN= 1,I13
S       N=N13(NN)
S       N1 = NOD1(N)
S       N2 = NOD2(N)
S       IF(L1(N))
  C...BOTH MUST BE SPECIAL.
      1ERROR(N1) = ERROR(N1) + TRAN(N)*ERRORX(N2)
S 44053 CONTINUE
  *VOCL LOOP,NOVREC
V       DO 44054 NN= 1,I21
V       N=N21(NN)
V       N1 = NOD1(N)
V       N2 = NOD2(N)
V       IF(L1(N))
  C...BOTH MUST BE SPECIAL.
      1ERROR(N2) = ERROR(N2) + TRAN(N)*ERRORX(N1)
V 44054 CONTINUE
  *VOCL LOOP,NOVREC
V       DO 44055 NN= 1,I22
V       N=N22(NN)
V       N1 = NOD1(N)
V       N2 = NOD2(N)
V       IF(L1(N))
  C...BOTH MUST BE SPECIAL.
      1ERROR(N2) = ERROR(N2) + TRAN(N)*ERRORX(N1)
V 44055 CONTINUE
S       DO 44056 NN= 1,I23
S       N=N23(NN)
S       N1 = NOD1(N)
S       N2 = NOD2(N)
S       IF(L1(N))
  C...BOTH MUST BE SPECIAL.
      1ERROR(N2) = ERROR(N2) + TRAN(N)*ERRORX(N1)
S 44056 CONTINUE
```

Fig. 4.15 Vectorized DO-Loop of Fig. 4.13.

4.2.10　ダミー領域の設定による境界処理の除去とodd－even 法によるベクトル化

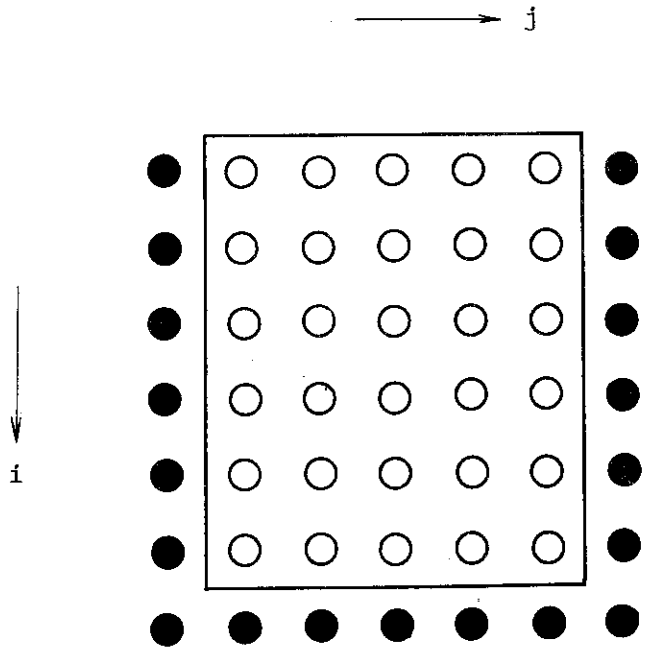
3.1節の(4)odd－even 法のところで述べたように，２次元の問題において，５点階差式をodd－even SOR 法で解く場合，ベクトル計算の効果を得るために，２次元配列を１次元化する。そのときに，Fig. 3.4 の格子領域を考えると，i 方向の格子点数が奇数であればよいが，そうでなければ，ダミーの格子点を１行加えて奇数にする。それは，Fig. 3.7 に示したように１次元化したインデックスの偶数と奇数がj 方向についても互い違いになるようにするためである。

また，Fig. 3.8 の式を計算する場合，$\tilde{x}_{\ell}^{(k)}$ が境界上に位置するとき，参照される$x_{\ell-1}$ や$x_{\ell-n}$ または$x_{\ell+1}$ や$x_{\ell+n}$ が格子領域の外側に位置することになり，プログラムの実行上は配列の外側を参照することになり問題がある。これを解決するために，通常のプログラムではＩＦ文を用いてこの境界処理を行っているが，この方法はベクトル計算の効果を下げてしまう。これに対して，本来計算すべき格子領域の外側にダミー領域を設ける方法は，ＩＦ文やリストベクトルを使用せず，直接参照方式でデータ参照できるのでベクトル計算の効果は大きい。計算結果についても，ダミーの格子点に対応したｘや係数に常にゼロを入れておけば影響を与えない。

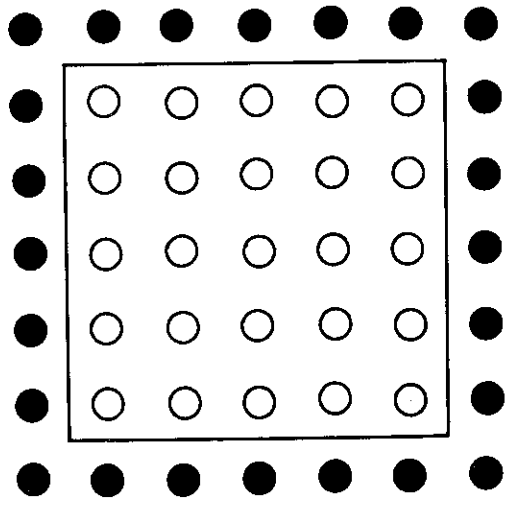
従って，上に述べた２種類のダミー領域は共用できるので，２次元の問題に対して，

- j 方向には，左右にそれぞれ１列のダミーの格子点を設ける。
- i 方向には，本来の計算領域の格子点数が，
  - 偶数ならば，ダミーの格子点を１行設ける。
  - 奇数ならば，ダミーの格子点を２行設ける。

ダミー領域の設け方をFig. 4.16 に示す。図の中で，枠の外側がダミーの格子点である。



(a) imax = even



(b) imax = odd

Fig. 4.16 Additional Dummy Mesh-Points (denoted by full circles ●).

4.2.11　3次元配列の連続番地直接参照方法

3次元空間を対象としたプログラムでは，3重ループが多く見られる。ベクトル化コンパイラでは，単純なネスト（入れ子）構造のとき以外は最内ループしかベクトル演算しない場合が多い。従って，外側の2重，3重のループも1重化してベクトル演算するよう再構成する。こうすることによって，ベクトル化対象部分が広がりベクトル化率が上がると共に，ベクトル長も長くなり，その部分のベクトル処理効率もよくなる。

Fig。4.17 上に示したものは，3次元の多群中性子拡散計算コードに見られる中性子源計算の一部であるが，ここでは，3次元空間x－y－z 形状の各インデックスに関する3重ループとその内側に，エネルギー群に関するループがある。最内ループは内積計算なので，そのままでベクトル計算可能であるが，3次元空間のメッシュサイズの方が，一般にエネルギー群数より大きいので，このエネルギー群に関するループを外側に出し，3次元空間に関する3重ループを内側に入れ，かつ1次元化してベクトル計算する。

3次元配列を1次元化してベクトル化する場合，別な配列を新に宣言し，EQUIVALENCE文によって同じ番地を参照するよう結合し，演算の部分は配列名を変更する方法があるが，修正箇所が多くなる。これに対して，そのサブルーチン内で3次元空間の全体に渡って連続的に参照している箇所が多い場合，オリジナルの3次元配列を配列名はそのままで，サイズに関する宣言だけを3次元分のトータルサイズを1次元で宣言する。この方法によれば，演算部分で参照されている配列名の変更も不用であり，ベクトル化コンパイラも1次元配列として長いベクトル長でベクトル化する。Fig．4.17 上のオリジナル版を1重化してベクトル化したものをFig．4.17 下に示す。

ただし，この方法について，文献(25) に述べられているように，FORTRANプログラムのドキュメント性を損う可能性もあり，今後，ベクトル化コンパイラの多重DOループの自動ベクトル化機能に期待するところである。

```
                                        [Original version]
REAL  A( imax,jmax,kmax )
for k=1,kmax
  for j=1,jmax
    for i=1,imax
      C = 0.0
      for l=l1,l2
        C = C + S(i,j,k,l)×φ(i,j,k,l)
      A(i,j,k) = C×V(i,j,k) + R(i,j,k)×E
---------- ---------- ---------- ---------- ---------- ----------
                                        [Vectorized version]
REAL  A( ijkmax ), C( ijkmax )
for n=1,ijkmax
  Cn = 0.0

for l=l1,l2
  for n=1,ijkmax
    Cn = Cn + Sn,l×φn,l

for n=1,ijkmax
  An = Cn×Vn + Rn×E
```

Fig. 4.17　Vectorization of a Triple-Nested DO-Loops in the Fission Source Calculation.

5. おわりに

ベクトル計算機向きのプログラミングについて，主に数値計算法と計算アルゴリズムの検討という面を中心にして話を進めて来た。ここで本報告中で述べたことの中で最も根本的な事柄であるというものを捨って列挙してみる。

① 数値解法の選択の際には計算の精度や値の収束性の良さなどだけに限らずベクトル化した場合の効率も或る程度考えに入れて判断するようにする。

② 入力データの設定に関しては，例えばメッシュによる空間の区切り方や節点番号の付け方など，ベクトル計算の効率を良く考えて行うようにする。

③ プログラムを実際にコーディングする場合，4章の最初で述べた基本的な事柄に留意する。

①は非常に多くの問題点を含んでおり本報告中で到底議論し尽せるものではない。多元連立方程式を解く場合を考えてみても驚く程多種の解法があり，その各々が係数行列の次元の大小や性質に依ってベクトル計算に対し異なった効率を示す。これらの問題は筆者らがこれから力を入れて研究してゆくべきものであり，各々の解法について詳しく調べられたものが独立した報告となって提示せらるべきものであろう。

計算センターにおける筆者らの課題は数値計算法の研究だけに限らず計算機利用者の実際のプログラミングの手助けとなることもその1つである。4章では具体的な例として実際に使われているプログラムを引用してベクトル化の方法について説明している。これらは既に存在しているプログラムのベクトル化に役立つであろう。

プログラムをベクトル化する為の最適の方法は前述の様にプログラム各々によって違い，また計算対象によっても異なる。計算機使用者が新しくベクトル計算機用にプログラムを作成する時あるいは既存のプログラムをベクトル化する時に我々の知識が役立つであろうし，またベクトル化の問い合わせを通じて我々も多くのことを得ると思う。本報告を利用して多くの方々がベクトル計算機VP－100を有効に利用されることを願っている。

謝　辞

計算センター室長平川隆氏には本報告を書く機会を与えて頂くと共に全般に関する御指導を受けました。心より感謝致します。

計算センター室長代理浅井清氏からは技術的な御指導・御援助がありました。

また2章の内容に関して理論解析研究室栗田源一氏と環境第一研究室石川裕彦氏から御教示を受けました。

本報告は計算センターの多くの方々の御協力の下に作成されました。ここで深く感謝致します。

5. おわりに

ベクトル計算機向きのプログラミングについて，主に数値計算法と計算アルゴリズムの検討という面を中心にして話を進めて来た。ここで本報告中で述べたことの中で最も根本的な事柄であるというものを捨って列挙してみる。

① 数値解法の選択の際には計算の精度や値の収束性の良さなどだけに限らずベクトル化した場合の効率も或る程度考えに入れて判断するようにする。

② 入力データの設定に関しては，例えばメッシュによる空間の区切り方や節点番号の付け方など，ベクトル計算の効率を良く考えて行うようにする。

③ プログラムを実際にコーディングする場合，4章の最初で述べた基本的な事柄に留意する。

①は非常に多くの問題点を含んでおり本報告中で到底議論し尽せるものではない。多元連立方程式を解く場合を考えてみても驚く程多種の解法があり，その各々が係数行列の次元の大小や性質に依ってベクトル計算に対し異なった効率を示す。これらの問題は筆者らがこれから力を入れて研究してゆくべきものであり，各々の解法について詳しく調べられたものが独立した報告となって提示せらるべきものであろう。

計算センターにおける筆者らの課題は数値計算法の研究だけに限らず計算機利用者の実際のプログラミングの手助けとなることもその1つである。4章では具体的な例として実際に使われているプログラムを引用してベクトル化の方法について説明している。これらは既に存在しているプログラムのベクトル化に役立つであろう。

プログラムをベクトル化する為の最適の方法は前述の様にプログラム各々によって違い，また計算対象によっても異なる。計算機使用者が新しくベクトル計算機用にプログラムを作成する時あるいは既存のプログラムをベクトル化する時に我々の知識が役立つであろうし，またベクトル化の問い合わせを通じて我々も多くのことを得ると思う。本報告を利用して多くの方々がベクトル計算機VP－100を有効に利用されることを願っている。

謝　辞

計算センター室長平川隆氏には本報告を書く機会を与えて頂くと共に全般に関する御指導を受けました。心より感謝致します。

計算センター室長代理浅井清氏からは技術的な御指導・御援助がありました。

また2章の内容に関して理論解析研究室栗田源一氏と環境第一研究室石川裕彦氏から御教示を受けました。

本報告は計算センターの多くの方々の御協力の下に作成されました。ここで深く感謝致します。

References

1) 浅井清；スーパーコンピューターの動向と原子力分野におけるベクトル計算処理，第16回「炉物理若手夏の学校」テキスト，p. 67，1984年7月24日～27日
2) 石黒美佐子；原子力コードのベクトル化経験，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，p. 151，1985年3月
3) 奈良岡賢逸，徳永康男，栗田豊，能村博人，篠沢尚久；原子力コードのベクトル化プログラミング〔I〕，私信
4) T.B. Fowler, D.R. Vondy, and G.W. Cunningham ; Nuclear Reactor Core Analysis Code : CITATION, ORNL-TM-2496, (Rev.2, 1971)
5) 原田裕夫，石黒美佐子；3次元中性子拡散コードCITATIONのベクトル化，日本原子力学会誌 Vol. 27, No.12 または Vol. 28, No.1 に掲載予定
6) F.R. Mynatt ; Development of Two-dimensional Discrete Ordinates Transport Theory for Radiation Shielding, Ph.D. thesis of Univ. Tennessee, (1969)
7) M. Azumi, G. Kurita, T. Matsuura, T. Takeda, Y. Tanaka, and T. Tsunematsu ; A Fluid Model Numerical Code System for Tokamak Fusion Research, Proc. 4th Int. Symp. on Computing Methods in Applied Sciences and Engineering, Versailles (1979)
8) G. Kurita, M. Azumi, T. Tsunematsu, and T. Takeda ; Numerical Studies of Toroidal Coupling on Low-m Resistive Modes, Plasma Physics 25 (1983) 1097
9) 茅野政道，石川裕彦；3次元風速場を用いた粒子拡散法による複雑地形上の被曝線量評価モデル，日本原子力学会誌 Vol. 26, No.6 (1984), p. 72
10) 茅野政道，他；SPEEDI：緊急時環境線量情報予測システム，JAERI－M 84－050，1984年
11) 茅野政道，石川裕彦；3次元風速場を用いたパフモデルによる緊急緒の被曝線量，日本原子力学会誌，Vol. 26, No.10 (1984) p. 73
12) E.L. Wilson, K.J. Bathe, F.E. Peterson, and H.H. Dovey ; SAP - A Structural Analysis Program for Linear Systems, Nucl. Eng. Des. 25 (1973) 257 and K.J. Bathe, private communication
13) A. Jennings ; Matrix Computation for Engineers and Scientists, John Wiley & Sons (1977)
14) V.H. Ransom, et al. ; RELAP5/MOD1 Code Manual, NUREG/CR-1826, EGG-2070, (1982)
15) 中原康明；原子炉流力特性解析における数値計算法の特徴と問題点，私信

16) M. Ishiguro, H. Harada, N. Shinozawa, and K. Naraoka ; Vectorization of LWR Transient Analysis Code RELAP5/MOD1 and its Effect, JAERI-M 85-040 (1985)

17) D. Barkai and K.J.M. Moriarty ; Vectorizing the Monte Carlo Program Algorithm for Lattice Gauge Theory Calculations on the Cyber 205, Computer Physics Communications 25 (1982) 57

18) L. Adams and J. Ortega ; A Multi-Color SOR Method for Parallel Computation, Proc. IEEE International Conference on Parallel Processing, pp.53-56 (1982)

19) J.A. Meijerink and H.A. van der Vorst ; An Iterative Solution Method for Linear Systems of Which the Coefficient Matrix is a Symmetric M-Matrix, Methematics of Computation 31 (1977) 148

20) 村田健郎；科学技術計算と高速算法, Computer Today, №2（1984年7月）

21) Y. Tokunaga, Y. Kurita, M. Sugihara, S. Hitoki and S. Saito ; Vectorization of the Monte Carlo Program DICON, Computer Physics Communications, in press

22) K. Asai, K. Higuchi, J. Katakura, and Y. Kurita ; Vectorization of KENO IV Code, Proc. International Meeting on Advances in Nuclear Engineering Computational Methods, Knoxville 1984, p.844

23) Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, ed. M. Abramowitz and I. Stegun (National Bureau of Standards), John Wiley & Sons

24) FACOM OS IV/F4 MSP FORTRAN77/VP プログラミングハンドブック, 78SP－5740（1985年）, 富士通（株）

25) 松浦俊彦, 他；FORTRANプログラムのドキュメント性と多重DOループの自動ベクトル化, 情報処理学会第29回全国大会論文集, p. 152, 1984