

JAERI-M
82-019

CRAY-1とFACOM230-75APUによる
拡散コードVENTUREのベクトル演算化

1982年3月

鎌田 稔^{*}・角谷 浩享^{*}・原田 裕夫

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 Section, Division of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

© Japan Atomic Energy Research Institute, 1982

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

CRAY-1 と FACOM 230-75 APU による
拡散コード VENTURE のベクトル演算化

日本原子力研究所東海研究所計算センター
鎌田 稔・角谷浩享・原田裕夫

(1 9 8 2 年 2 月 2 日受理)

原子炉の炉心特性解析のための 3 次元拡散コード VENTURE を並列計算機 CRAY-1 と FACOM 230-75 APU に変換するとともに、並列計算処理が適用可能なようにプログラムのベクトル化を行った。その結果、計算時間は CRAY で約半分に、F 75 APU で約 70 % に短縮された。

VENTURE は計算機の主記憶容量にはほとんど関係なく、1 方向 1000 メッシュの問題も解ける。

高速化については、プログラム全体の計算時間の約 80 % を占める 5 つのサブルーチンを DO ループ内の IF 文の解消や逐次解法の反復解法への変更によってベクトル演算化した。

ここでは、VENTURE コードの概要、プログラムの動的特性の分析、ベクトル演算化の具体的な 5 つの方法について述べ、CRAY と F 75 APU の実行結果と並列計算効果についても述べる。

JAERI-M 82-019

Vectorization of Diffusion Code VENTURE

Using CRAY-1 and FACOM 230-75 APU

*
Minoru KAMADA , Hiroyuki KADOTANI and Hiroo HARADA

Computing Center, Tokai Research Establishment, JAERI

(Received February 2, 1982)

Three-dimensional neutron-diffusion code VENTURE has been vectorized to run on the vector processors, CRAY-1 and FACOM 230-75 APU(Array Processor Unit).

Vector processing effect is observed as the CPU time reduction of about 50% to the scalar processing case for CRAY-1, and about 70% for FACOM 230-75 APU.

The whole process of converting the code from scalar-oriented programming to vector-oriented one is described in some detail with emphasizing the importance of changing the code structure.

Keywords; Vectorization, Diffusion Code, VENTURE, CRAY-1, FACOM 230-75 APU, Parallel Computation, Computer Code

* Century Research Center Corporation

目 次

1.はじめに	1
2. VENTURE コードシステム	2
2.1 VENTURE コードの概要	2
2.2 内側の反復ループ	6
3. ベクトル化の方法	8
3.1 ベクトル化ルーチンの選択	8
3.2 ベクトル化の方針	12
3.3 DO ループの分割によるベクトル化	12
3.3.1 形状, 次元(入力データ)による判定	17
3.3.2 境界条件(DO ループ制御変数)による判定	17
3.3.3 中性子束分布による判定	17
(配列特殊関数によるベクトル化)	
3.3.4 物質領域に対する定義・参照	18
(間接アドレスによる配列参照)	
3.4 解法の変更によるベクトル化	19
4. ベクトル演算化の効果	24
4.1 サンプル問題	24
4.2 CRAY-1 によるベクトル化効果の検討	27
4.2.1 CRAY-1 によるベクトル化効果	27
4.2.2 加速係数の検討	40
4.3 F75 APU によるベクトル化効果の検討	42
4.3.1 F75 APU/CPU のためのプログラム	42
4.3.2 F75 APU による計算時間	43
4.3.3 ベクトル化による全体的効率	44
5. おわりに	48
謝 辞	48
参考文献	50
付 錄 : 計算機システムとベクトル演算機能	51
A. 1 CRAY-1 のハードウェア	51
A. 2 CRAY-1 のソフトウェア	55
A. 2.1 CFT の自動ベクトル化条件	55
A. 2.2 従属関係	56
B. 1 FACOM 230-75 APU/CPU 計算機システム	58
B. 2 AP-FORTRAN	58

CONTENTS

1. Introduction	1
2. VENTURE code system.....	2
2.1 An overview of VENTURE code	2
2.2 Inner iteration	6
3. Methods of vectorization	8
3.1 Selection of the routines to be vectorized	8
3.2 Methods of vectorization	12
3.3 Vectorization by separating DO loop.....	12
3.3.1 IF statement of geometry and dimension (input data)...	17
3.3.2 IF statement of boundary condition (DO loop control variable)	17
3.3.3 IF statement of neutron flux (vectorization using array special functions)	17
3.3.4 Definition and reference to zones (vectorization of indirect addressed array)	18
3.4 Vectorization by changing of algorithms	19
4. Effect of vectorization	24
4.1 Sample problems	24
4.2 Consideration of vectorization effect by CRAY-1	27
4.2.1 Vectorization effect by CRAY-1	27
4.2.2 Consideration of accelerated coefficient	40
4.3 Consideration of vectorization effect by FACOM 230-75 APU	42
4.3.1 Programs for FACOM 230-75 APU/CPU	42
4.3.2 Computing time by FACOM 230-75 APU	43
4.3.3 Global performance ratio on vectorization	44

5. Concluding remarks	48
Acknowledgement	48
References	50
Appendix: Computer system and vectorization facility 51	
A.1 Hardware of CRAY-1	51
A.2 Software of CRAY-1	55
A.2.1 Automatic vectorized conditions of CFT(CRAY FORTRAN)...	55
A.2.2 Dependency.....	56
B.1 FACOM 230-75 APU/CPU system.....	58
B.2 AP-FORTRAN	58

1. はじめに

原子炉の炉心特性解析で中心的な役割をはたす拡散コードは、長い発展の歴史をもっているが、この歴史の頂点に立つコードの一つに、VENTUREコード⁽¹⁾がある。VENTUREは、米国オークリッジ国立研究所で、CITATIONコード⁽²⁾を開発したグループにより1975年に発表されたコードで、入出力が全てInterface Data File（ディスク等の補助記憶装置にとられるデータイメージ）を介して行なわれる、純粋な階差近似の拡散方程式ソルバー（Solver, コードブロック）として設計された。またCITATION等の在来のコードと比べた時の際立った特徴は、使用できる電子計算機の主記憶容量にはほとんど関係なく、任意の大きさの問題（一つの次元方向について1000空間メッシュの問題がとかれており、この場合でも、他の二つの次元方向の空間メッシュ数には制限がない）を解くことができることであろう。

VENTUREは、オークリッジ研究所に設置してあるIBMの360シリーズの計算機に対して書かれたが、この計算機はやや古い世代に属するものである。その後発表された計算機の中で、現在スーパーコンピューターと呼ばれている一群の計算機がある。この中には、CDC-STAR, CRAY-1, ILLIAC-IV, TI-ASC等が実際に使用されている。これらの計算機では50 MIPS (million Instructions per Second, 百万命令／秒)以上のスピードを持っており、このスピードを達成するために様々な工夫がこらされている。特に本報告でとりあげたCRAY-1では、スカラー計算用のレジスター（従来の計算機と同じ演算器）を用いることができる他に、64語の長さをもつベクトルレジスターが利用できる。CRAY-1では、スカラーレジスターのみを用いても、従来の計算機に比べて大巾な計算時間の短縮が可能であるが、さらにベクトルレジスターを用いることにより、スカラーレジスターのみを用いる場合に比べて数倍のパフォマンスが実現できる。FACOM 230-75 APUもベクトルレジスタを持つバイオペライン方式のベクトル計算機であり、内積などの基本的演算では、22MFLOPS (Million Floating Operation Per Second) の性能のベクトル計算処理機能を有している。

本研究の目的は、このCRAY-1とFACOM 230-75 APUコンピューターへVENTUREのコンバージョンを試み、ベクトルレジスターを可能な限り利用して、計算時間の短縮をはかる事である。またこの研究により、ベクトルレジスターを有する計算機に対して経験をつみ、大型原子力コードへのベクトル計算処理の適用性に関する研究の一つの基礎資料とすることも一つの目的としている。

CRAY-1等のスーパーコンピューターへの在来のアプリケーションでプログラムのコンバージョンと高速化に関する研究は、まだ研究の歴史が浅く、会議録として参考文献[3]があり、またDOT-IV⁽⁴⁾コードをCDC STAR 100に変換した研究が発表されているが、確立した手法、アルゴリズムは存在しないと云える。しかし、文献[3]にある三次元拡散コードDIF3D⁽⁵⁾コードのCRAY-1への変換、およびDOT-IVでは、ベクトルレジスター利用のために計算の流れまたはアルゴリズムのかなり大巾な変更が行なわれているようである。そこで本研究では、VENTUREの基本的な計算の流れを変更することなく、ベクトル化につとめた。このため

に、一つはFORTRANの表現の変更を行ない、コンバイラーによる自動ベクトル化の機能を最大限用いた。他は繰返し計算における変数の意味を変更してベクトル化を行なった。このベクトル化の効果を数量的に評価するために、二次元と三次元の問題三ケースにつき、ベンチマーク計算を行なっている。

本報告書は以下、第2章ではVENTUREコードの概要を述べた後、第3章ではベクトル化のための具体的な手法、第4章ではCRAY-1とFACOM 230-75 APUによる実行結果を論じる。第5章では全体の結果と残された問題を述べる。

2. VENTURE コードシステム

2.1 VENTURE コードの概要

VENTUREは、多群、多次元の中性子輸送問題を解くためのコードブロックである。中性子輸送方程式に対する数値解法としては、有限階差近似での拡散近似と簡単な P_1 近似をとることができる。通常の増倍系に対する増倍率（固有値）と中性子束分布を求める計算がVENTUREコードの基本的な機能であり、この問題の随伴問題（Adjoint Problem）も取扱うことができる。また増倍系での種々の臨界調整問題を解く機能がある。マクロ断面積を基礎におく摂動計算も可能である。さらに、外部中性子問題（Fixed Source Problem）を解くことができる。

計算モデル上の機能としては次のようなものが用意されている。中性子の散乱は任意の群から任意の群へのエネルギー変化が許されており、この中には上方散乱も含まれている。体系の内部に境界条件を設定する内部完全吸収領域（Internal Black Absorber）を含む種々の境界条件をとることが可能であり、特に周期的境界条件、さらに回転対象条件をとることができます。

VENTUREは、有名なCITATIONコードと同じ著者達によりCITATIONの次の標準的な拡散コードとして計画され、作成された。使用者の立場から考えるとVENTUREコードとCITATIONコードの相違の主なものは、VENTUREでは解くべき問題の大きさに関する制限がないことである。またVENTUREは、カードからの入力機能を有していない。VENTUREへの入力は全て、ディスク（テープ）を通して行なわれる。なおこれらのファイルへの形式（Format）は、米国の標準フォーマット（Standard Interface Format）を用いており、このフォーマットを用いる他のプログラムにより、VENTUREを容易に利用することが可能である。しかし、VENTUREのみを単独で用いる場合には、これでは不便なため、CITATIONとほぼ同一の仕様の入力プログラムを持つVENTUREが用意されている。このVENTUREはBOLD-VENTUREと呼ばれ、本報告書で用いたVENTUREはこのBOLD-VENTUREである。

Fig.2.1にVENTUREコードの計算の流れ図を示した。階差近似で表わした拡散方程式は、通常の繰返し計算で解かれる。繰返し計算には、次に述べる二つのレベルがある。一つは中性

に、一つはFORTRANの表現の変更を行ない、コンバイラーによる自動ベクトル化の機能を最大限用いた。他は繰返し計算における変数の意味を変更してベクトル化を行なった。このベクトル化の効果を数量的に評価するために、二次元と三次元の問題三ケースにつき、ベンチマーク計算を行なっている。

本報告書は以下、第2章ではVENTUREコードの概要を述べた後、第3章ではベクトル化のための具体的な手法、第4章ではCRAY-1とFACOM 230-75 APUによる実行結果を論じる。第5章では全体の結果と残された問題を述べる。

2. VENTURE コードシステム

2.1 VENTURE コードの概要

VENTUREは、多群、多次元の中性子輸送問題を解くためのコードブロックである。中性子輸送方程式に対する数値解法としては、有限階差近似での拡散近似と簡単な P_1 近似をとることができる。通常の増倍系に対する増倍率（固有値）と中性子束分布を求める計算がVENTUREコードの基本的な機能であり、この問題の隨伴問題（Adjoint Problem）も取扱うことができる。また増倍系での種々の臨界調整問題を解く機能がある。マクロ断面積を基礎におく摂動計算も可能である。さらに、外部中性子問題（Fixed Source Problem）を解くことができる。

計算モデル上の機能としては次のようなものが用意されている。中性子の散乱は任意の群から任意の群へのエネルギー変化が許されており、この中には上方散乱も含まれている。体系の内部に境界条件を設定する内部完全吸収領域（Internal Black Absorber）を含む種々の境界条件をとることが可能であり、特に周期的境界条件、さらに回転対象条件をとることができます。

VENTUREは、有名なCITATIONコードと同じ著者達によりCITATIONの次の標準的な拡散コードとして計画され、作成された。使用者の立場から考えるとVENTUREコードとCITATIONコードの相違の主なものは、VENTUREでは解くべき問題の大きさに関する制限がないことである。またVENTUREは、カードからの入力機能を有していない。VENTUREへの入力は全て、ディスク（テープ）を通して行なわれる。なおこれらのファイルへの形式（Format）は、米国の標準フォーマット（Standard Interface Format）を用いており、このフォーマットを用いる他のプログラムにより、VENTUREを容易に利用することが可能である。しかし、VENTUREのみを単独で用いる場合には、これでは不便なため、CITATIONとほぼ同一の仕様の入力プログラムを持つVENTUREが用意されている。このVENTUREはBOLD-VENTUREと呼ばれ、本報告書で用いたVENTUREはこのBOLD-VENTUREである。

Fig. 2.1にVENTUREコードの計算の流れ図を示した。階差近似で表わした拡散方程式は、通常の繰返し計算で解かれる。繰返し計算には、次に述べる二つのレベルがある。一つは中性

子束を求める繰返し計算 (*Inner Iteration*) である。ここではあるエネルギー一群で、この群で発生する中性子源 (散乱線源、核分裂線源、外部中性子線源) が決まった時、この群の中性子束分布を求める計算が行なわれる。この時、中性子束は、隣接する空間メッシュの中性子束と拡散係数を通して結合している。中性子束はまず空間座標の一つの方向 (Fig. 2.2 参照、行方向 : *Column* 方向) について同時に求められる。列 (*Row*) を変化させてこの計算が繰返され、ある群に対して、全空間メッシュの中性子束が求まるまで進められる。列の進め方は、一つの列の中性子束の計算にはこの列に隣接する列の中性子束の結果が影響を与えるため、新に計算された列の中性子束の結果が最大限利用できるように進められる。上の計算を全ての群について行なった後で、固有値が評価され、これを用いて新しい核分裂中性子源が作成される (中性子源繰返し計算、*Outer Iteration*)。中性子束ないし固有値に対して収束がチェックされ、指定された許容誤差の外にあれば、上の計算が繰返される。

上の解法は、VENTURE で大型の (多群で空間メッシュの多い) 問題が解けることと関係している。VENTURE では、主記憶上にとるデータと、低スピードの補助記憶装置上にとるデータの分割法、およびその間のデータの移動に関して種々の手法が用意されている。利用できる主記憶の大きさに応じて、主記憶上にとられるデータの階層が決定される。メモリが充分ある場合は、全てのデータ (全群、全空間メッシュの中性子束、階差方程式の係数等) が、主記憶内にとられる。一方反対の極限は、一つの行に関するデータ (一行分) の中性子束を解くのに必要な最少限のデータが、メモリにとられる。VENTURE ではこの中間の大きさのメモリに応じて、数個の行に関するデータ、一枚から数枚の平面 (*Plane*)、一群の全空間メッシュに関するデータ等をメモリにとって繰返し計算が行なわれる。この結果 VENTURE では、主記憶の大きさに関係なく、大型の問題が取扱えることになった。特に一つの行について、その行の中性子束を解くのに必要な最少のデータを主記憶上にとる場合 (*One Row Stored Mode*) では、要求されるメモリの大きさは非常に小さなものである。しかしこの場合は、補助記憶装置との I/O の回数が多くなり、計算機内でのプログラムの滞在時間は極端に長くなろう。

CITATION では、一次元方向の空間メッシュは 210 以下であるが、VENTURE では 1000 メッシュの問題が解かれている。一次元問題が解ければ、VENTURE では他の二つの次元に関しては、メッシュ数の制限はないことになる。

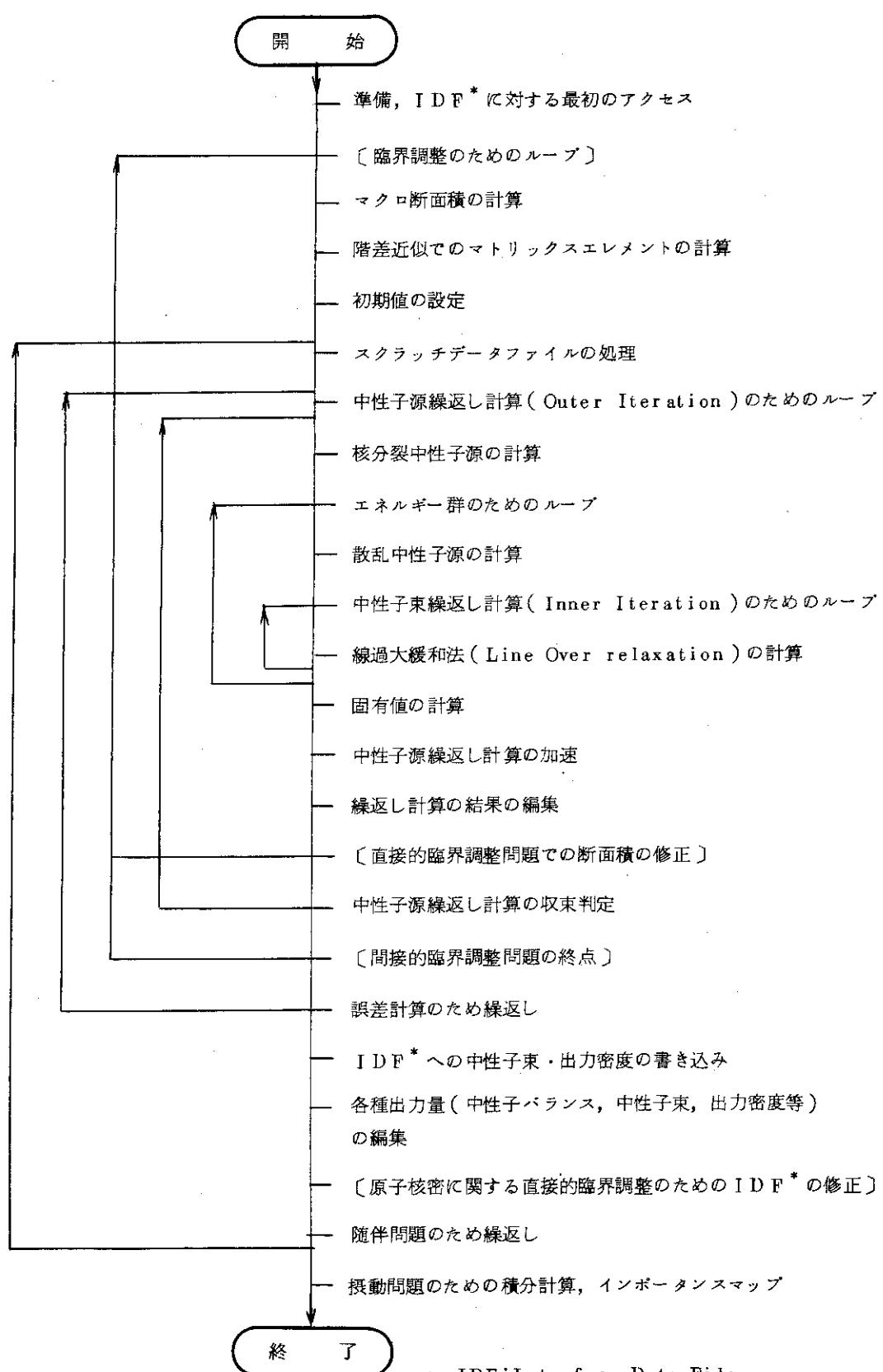
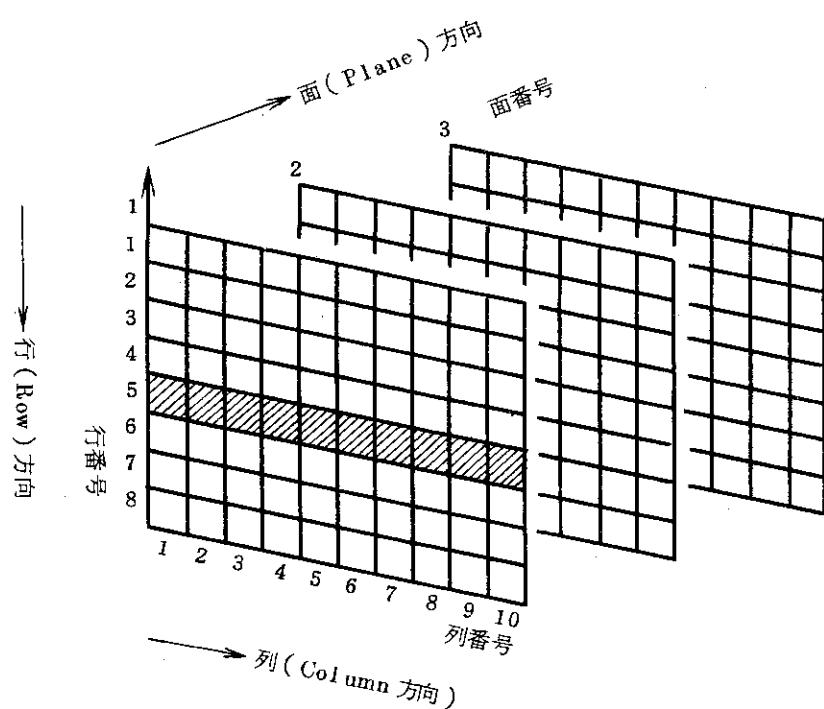


Fig. 2.1 Flow diagram of VENTURE finite-difference diffusion theory neutronics code



VENTURE コードでの空間メッシュの指定法、斜線部分が空間メッシュの一行(One Row)に相当する。二次元問題では上で一つの面(Plane), 一次元問題では、一行が解かれる。

Fig. 2.2 Space mesh of VENTURE code

2.2 内側の反復ループ

拡散方程式は、エネルギーについては群表示して積分を和で表わし、空間座標はメッシュに切って微分を階差で表現すると、次のようにマトリックスで記述できる。

$$A\phi = \frac{1}{k_e} \chi F \phi \quad (2.1)$$

この式は

$$(A^{-1} \chi F - k_e I) \phi = 0 \quad (2.2)$$

と変形すると通常の固有値問題となり、実効増倍率 k_e は、この固有値問題の最大固有値である。 ϕ は k_e に対応する固有ベクトルで、 $k_e = 1$ の時は、臨界状態での中性子束分布となる。マトリックス A は、中性子の移動、散乱による結合、吸収、もれによる中性子の損失を表わす演算子であり、 F は核分裂により発生する中性子を表わす演算子、 χ は核分裂中性子のエネルギー分布を表わす演算子である。

Inner Iteration は (2.1) 式の右辺を、仮定するかあるいは、一回前の繰返し計算で求まつた k_e 、 ϕ を用いて計算した時、左辺の ϕ を繰返し計算により求める手続きである。**Outer Iteration** は、こうして求まつた ϕ を用いて k_e と (2.1) 式の左辺を求める計算である。

2.1 節でも述べたように VENTURE コードでは、上の **Inner Iteration**において、左辺にある未知数の ϕ の内、一つの Column 方向（一つの行の ϕ ）のみを未知数として残し、他の ϕ はすべて既知として右辺におく。こうして、二次元ないし三次元の問題もすべて一次元問題に帰着して解かれる。一つの行の ϕ が求まつた時は、次の行へ計算をすすめる。この時、既知とする他の ϕ が、できるかぎり今求まつた新らしい ϕ となるように、次の行が選ばれる。こうして全ての空間メッシュ、全てのエネルギーに對して ϕ が求まつてのち、 k_e を再評価して再び **Inner Iteration** を繰返す。

この目的のために (2.2) 式のマトリックス A を次のように分割する。

$$A = D - J - L - U - S - T \quad (2.3)$$

ここで、

D : 吸収、バックリングによる吸収、他群への散乱を表わすマトリックスエレメント（対角成分のみ）

J : 一つの行について空間メッシュの結合を表わすマトリックス。この行について中性子束は一度に決められる。

L : 空間メッシュの結合に関する下半分マトリックス。

U : 空間メッシュの結合に関する上半分マトリックス、但し、J に表われるものは含まれない。

S : 下方散乱による線源を表わすマトリックス。

T : 上方散乱による線源を表わすマトリックス。

(2.3) 式を (2.1) 式に代入して変形すると、

$$(D - J) \phi = (L + U + S + T - \frac{1}{k_e} \chi F) \phi \quad (2.4)$$

となる。この段階では、左辺は一次元問題となり、標準的な三点階差式となる。

(2.4)式をマトリックスエレメントの関係として書きかえると

$$a_i X_i = S_i + b_{i-1} X_{i-1} + b_{i+1} X_{i+1} \quad (2.5)$$

となる。ここで i はある行での空間メッシュを示し、 S_i はこの点に関する (2.4) 式の右辺の値である。 a_i は i 点での中性子の損失を表わし、 b_{i+1} , b_{i-1} は i 点と $i+1$, $i-1$ 点の中性子束の結合を表わす係数である。(2.5) 式は、前進、後退繰返し法 (Forward/Backward Recurrence Method) で正確に解くことができる。すなわち、まず、 i に関して前進的に、

$$g_i = b_{i+1} \left[\frac{1}{a_i - b_i g_{i-1}} \right] \quad (2.6)$$

$$f_i = S_i + g_{i-1} f_{i-1} \quad (2.7)$$

で、 g_i , f_i を求め、次に、 i に関して後退的に

$$X_i = g_i \left[X_{i+1} + \frac{f_i}{b_{i+1}} \right] \quad (2.8)$$

X_i すなわち中性子束を求める。 g_1 , f_1 , X_N (ただし N は添字 i の最大値) は、境界条件より決められる。

Inner Iteration で求まる X_i は、過緩和法 (Over Relaxation Method) により加速される。今 t を Inner Iteration を表わす添字とすると、

$$X_{i,t} = X_{i,t}^* + \beta_t (X_{i,t}^* - X_{i,t-1}) \quad (2.9)$$

で加速計算が行なわれる。ここで $X_{i,t}^*$ は (2.8) 式で求まる t 番目の Inner Iteration での X_i , $X_{i,t-1}$ は一回前の Inner Iteration で求まつた中性子束である。 β_t は加速因子であり、 t 番目の Inner Iteration で求まる中性子束は $X_{i,t}$ である。 β_t は VENTURE により自動的に計算されるが、BOLD-VENTURE ではカードより入力した値を用いて計算することもできる。

3. ベクトル化の方法

拡散コードVENTUREのCRAY-1によるベンチマーク・テストを実施するに当り、プログラム中で最も負荷度（ここでは全CPU時間に対する割合と定義する）の高いいくつかのサブルーチンを選択し、それらに対してベクトル化を図った。ソースプログラムのベクトル化には2つの方式があり、ベクトル化コンバイラによる方式とベクトル演算用拡張FORTRAN言語による方式である。いずれの場合も入手によるベクトル化のための再構成が必要である。前者はCRAY-1フォートラン・コンバイラ（以下CFTという）による自動ベクトル化を意味する。後者は、FACOM 230-75 APUの場合である。ここではCRAY-1によるベクトル化を中心述べる。CFTによる自動ベクトル化を可能にするためには種々の条件を満足させなければならないがその条件については付録A.2.1に示す。本章は全体的なベクトル化の方針および具体的なベクトル化の方法について述べたものである。

3.1 ベクトル化ルーチンの選択

拡散コードは通常Outer Iteration（核分裂中性子源計算）とInner Iteration（中性子束計算）とからなる。このうちInner Iteration部分の全体に対する負荷度は高い（VENTUREではInner Iterationだけで全CPU時間のおよそ90%をしめる）。従ってこの部分をベクトル演算により高速化できればその効果は非常に大きいと考えられる。

またVENTUREでは第2章で記した様に種々のメモリ使用モードが用意されている。それらは例えばAll stored Mode（体系、エネルギー群の全てを主記憶に割当てる）やPlane Stored Mode（3次元計算において1平面分だけ主記憶に割当てる）等である。そしてこれら各モード毎にOuter Iteration以下のサブルーチンが区別されている。ここでのベクトル化はその効果が把握しやすく且つプログラムも比較的容易なAll Stored Modeのみに着目する。

まずFig.3.1にAll Stored Modeのみに着目したOuter Iteration以下のブロック・ダイヤグラムを掲げる。この図において*4を付したサブルーチンについてはその中にColumn方向のDOループをもつ。従ってCFTによる自動ベクトル化は最も内側のDOループのみを対象とするので、今回のベクトル化もこれらのサブルーチンのColumn方向のDOループについてのみ着目する。以下これらのサブルーチンを「ベクトル化対象サブルーチン」という。

Table 3.1にベクトル化対象サブルーチンの機能を掲げる。

ここで6個のベクトル化対象サブルーチンについてその効果を考える。仮に全ての最も内側のDOループがベクトル化できたとした場合、個々のDOループでの効果は勿論上がるであろう。しかし現実にはベクトル化の効果とはプログラム全体、例えばジョブの開始から終了までのCPU時間で評価されるべきものであり、その場合負荷度の高いDOループのベクトル化ほど有効であろう。この様にしてベクトル化対象各サブルーチンの負荷度を最も内側のColumn方向のDOループの実行回数、いいかえるとこれらのサブルーチンが呼ばれる回数で表わすとし、

以下の様な体系等を仮定するとサブルーチン毎の負荷度は Table 3.2 の如くなる。

IOUTR = 37 36 回で Outer Iteration が収束し、残差計算等のためにもう 1 回繰返す。

KVX = 3 3 群計算。

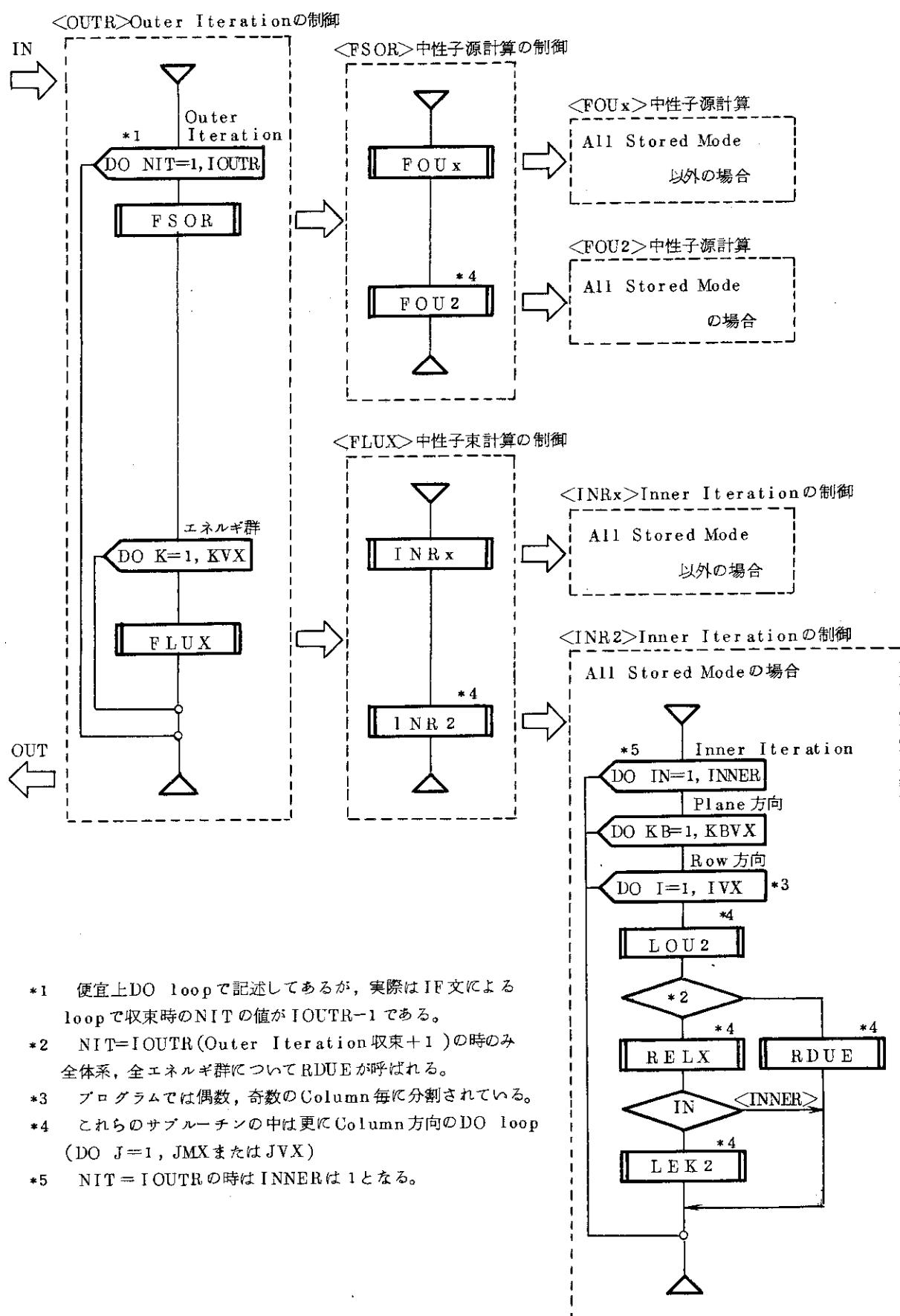
INNER = 4 Inner Iteration は 4 回で打切る。

KBVX = 18 18 Planes

IVX = 13 13 Columns } 3 次元体系。

JVX = 13 13 Rows

Table 3.2において負荷度はサブルーチンで表わしているため評価は必ずしも妥当とはいえないが、おおよその効果は予測できるであろう。従ってこの表から解る事は特に LOU2, RELX および LEK2 の負荷度が非常に大きいため、これらについてベクトル化できればその効果は期待できると考えられる。



- *1 便宜上DO loopで記述してあるが、実際はIF文によるloopで収束時のNITの値がIOUTR-1である。
- *2 NIT=IOUTR(Outer Iteration 収束+1)の時のみ全体系、全エネルギー群についてRDUEが呼ばれる。
- *3 プログラムでは偶数、奇数のColumn毎に分割されている。
- *4 これらのサブルーチンの中は更にColumn方向のDO loop (DO J=1, JMX または JVX)
- *5 NIT = IOUTRの時はINNERは1となる。

Fig. 3.1 Block diagram of outer and inner iteration

Table 3.1 Contents of vectorized subroutines

サブルーチン	機能
FOU 2	核分裂中性子源の計算
INR 2	Inner Iteration の制御
LOU 2	体系内部 Row 間、Plane 間の中性子束 リーケージ計算
RELX	1 次元拡散方程式を Forward/Backward 法で解く
RDUE	残差の計算
LEK 2	体系外への中性子漏洩の計算

Table 3.2 Execution times of vectorized
subroutines

サブルーチン	計算式	負荷度*
FOU 2	IOUTR	37
INR 2	IOUTR * KVX	111
LOU 2	(IOUTR - 1) * KVX * INNER * KBVX * IVX + KVX * KBVX * IVX	101,790
RELX	(IOUTR - 1) * KVX * INNER * KBVX * IVX	101,088
RDUE	KVX * KBVX * IVX	702
LEK 2	(IOUTR - 1) * KVX * KBVX * IVX	25,272

* サブルーチンが呼ばれる回数

3.2 ベクトル化の方針

3.1 節で選択したベクトル化対象サブルーチンをベクトル化するに当たりここではその方針について簡単に記す。まずベクトル化の方法としては次の2通りが考えられる。

- (1) DO ループの分割によるベクトル化。
- (2) 解法の変更によるベクトル化。

ここで(1)については IF 文等を含むため DO ループを分割する事などによるベクトル化で、ロジックの変更はないのでベクトル化前後でその計算結果は完全に一致する。(2)については計算結果は一致しないであろう。しかしそれは例えば収束計算についていと収束回数や収束安定性だけの差異であり、収束してしまえばその解は妥当なものが得られるはずである。

VENTURE をベクトル化するに当たり Table 3.1 に掲げたサブルーチンを検討してみると RELX 以外は全て(1)の方法で解決する。RELX については(2)の方法を考えなければならないが、コードの構成からみて RELX 以外のルーチンにまでおよぶ解法の変更は困難であり、ここでは RELX 内部だけの解法の変更にとどめる。

3.3 DO ループの分割によるベクトル化

あるプログラムのベクトル化を試みる時、効果の上りそうな DO ループをみるとそのほとんどがループ内に IF 文をもっている。そしてループ内の IF 文はベクトル化の妨げとなる。この他にもベクトル化の条件があるが、それらの主なものをまとめると次のとおりである。

- IF 文がない事。
- GOTO 文がない事。
- CALL 文がない事。
- 入出力文がない事。
- 外部関数を参照していない事。
- 配列(ベクトル)の各要素が DO ループの実行の前に明確に解る事。例えば間接アドレスによる定義・参照をしていない事。
- 左辺、右辺および前後のステートメントの配列の添字に従属関係がない事。

一方、VENTUREにおけるベクトル化対象サブルーチンのうち RELX 以外のサブルーチンについてはベクトル化されない理由について以下の4つのパターンに分類できる。

- (1) Geometry, Dimension 等による判定がある。

Geometry (X-Y-Z, R-Z-θ 等), Dimension (1, 2 および 3 次元問題), また境界条件オプション(例えば繰返し境界)等の判定のために DO ループ内に IF 文がありベクトル化できない。

- (2) 境界条件を取り扱う判定がある。

体系の境界での条件を与えるため各方向(例えば Column 方向)のメッシュ・インデックスの始点、終点を参照する IF 文が DO ループ内にありベクトル化できない。

(3) 中性子束分布による判定がある。

体系の内部の境界条件をとる内部完全吸収領域(Internal Black Absorber)を取扱うためにVENTUREではその領域の中性子束を0とおいてフラックとして用いている。従って中性子束分布の値を参照する IF 文が DO ループ内に存在レベクトル化不可能となる。

(4) 物質領域(ゾーン)に対する定義・参照がある。

体系内の各セルはその物質を表わす領域番号が定義されている。各セルと領域との対応はテーブルを介して行なわれるため間接アドレスによる配列の定義・参照が必要となりベクトル化されない。

これらのうち(1)～(3)はDO ループ内に IF 文を含むもので、(4)は間接アドレスによる定義・参照である。以下に各項目毎に実際のプログラムを例にあげベクトル化の方法を説明するが、結果としてほとんどが DO ループの分割によりベクトル化できた(RELX については配列の添字に従属関係があるので次節で説明する)。尚、サンプルとして掲げたプログラムは次のとおりである。

- Fig. 3. 2(a) LOU 2 のオリジナルプログラムの一部(六角、三角格子以外の場合)。
- Fig. 3. 2(b) LOU 2 のベクトル化後のプログラムの一部(六角、三角格子以外の場合)。
- Fig. 3. 3(a) LEK 2 のオリジナルプログラム。
- Fig. 3. 3(b) LEK 2 のベクトル化後のプログラムの一部。

```

      GO TO (100,110,130), IGOGM
100 CONTINUE
C   FOR ALL GEOMETRYS EXCEPT HEX OR TRIANGULAR.
    DO 109 J = 1,JVX
    M = NCOMP(J,I,KB)           —————④
    TLPSL8(J) = TLLOSS8(J,I1, K)
    IF (ISHR.GT.0) TLPSL8(J) = TLPSL8(J)+SHARE8(34)*SHRAB(M,K)*
    * PVOL(J,I,KB)
    T18 = SOUR8(J)
    IF (NDIM.EQ.1) GO TO 108
    IF (I.LE.1) GO TO 101
    T18 = T18+PC8(J,I3, K)*DL(J,I4, K)
101 CONTINUE
    IF (I.GE.IVX) GO TO 102
    T18 = T18+PC8(J,I2, K)*DL(J,I5, K)
102 CONTINUE
    IF (NDIM.LT.3) GO TO 104           —————⑦
    IF (KB.LE.1) GO TO 103
    T18 = T18+PC8(J,I7, K)*DB(J,I1, K)
103 CONTINUE
    IF (K.P.GE.KBVX) GO TO 104       —————②
    T18 = T18+PC8(J,I6, K)*DB(J,I6, K)
104 CONTINUE
    GO TO (108,105,107), ISYMI
105 CONTINUE
    IF (J.NE.JVX) GO TO 106
    IF (I.EQ.IVX) GO TO 106
    I8 = (KB-1)*IVX+IVX
    T18 = T18+PC8(I,I8, K)*DR(JVX+1,I1, K)           ③
106 CONTINUE
    IF (I.NE.IVX) GO TO 108
    IF (J.EQ.JVX) GO TO 108
    I8 = (KB-1)*IVX+J
    T18 = T18+PC8(JVX,I8, K)*DL(J,I5, K)
    GO TO 108
107 CONTINUE
    IF (J.NE.JVX) GO TO 108
C   FOR 180 DEGREE SYMMETRY.
    I8 = (KB-1)*IVX+IVX+1-I
    T18 = T18+PC8(J,I8, K)*DR(JVXP1,I1, K)
108 CONTINUE
    TSOUR8(J) = T18
109 CONTINUE

```

Fig. 3.2(a) Part of subroutine LOU2 (original program)

COMMON/WORK/WRK1(120),WRK2(120),NWK3(120),WRK4(120)

(6)

```

C      CALL GATHER(JVX,WRK1,SHRAB(1,K),NCOMP(1,I,KB)) ——④
C
C      IF(I SHR.GT.0) GO TO 992
C----- DO 990 J=1,JVX
C----- TLPSL8(J)=TLOSS8(J,I1,K)
C----- 990 WRK2(J) =SOUR8(J)
C----- GO TO 996
C----- 992 CONTINUE
C----- DO 994 J=1,JVX
C----- TLPSL8(J)=TLOSS8(J,I1,K)+SHARE8(34)*WRK1(J)*PVOL(J,I,KB) ——⑤
C----- 994 WRK2(J) =SOUR8(J)
C----- 996 CONTINUE
C
C      GO TO (1000,2000,4000),IGOGM
C
C      FOR ALL GEOMETRYS EXCEPT HEX OR TRIANGULAR.
1000 IF(NDIM.EQ.1) GO TO 1124
      IF(I.LE.1) GO TO 1100
C----- DO 1098 J=1,JVX
C----- 1098 WRK2(J)=WRK2(J)+PC8(J,I3,K)*DL(J,I4,K)
1100 IF(I.GE.IVX) GO TO 1104
C----- DO 1102 J=1,JVX
C----- 1102 WRK2(J)=WRK2(J)+PC8(J,I2,K)*DL(J,I5,K)
1104 IF(NDIM.LT.3) GO TO 1111 ——①
      IF(KB.LE.1) GO TO 1108
C----- DO 1106 J=1,JVX
C----- 1106 WRK2(J)=WRK2(J)+PC8(J,I7,K)*DB(J,I1,K) ——②
1108 IF(KB.GE.KBVX) GO TO 1111
C----- DO 1110 J=1,JVX
C----- 1110 WRK2(J)=WRK2(J)+PC8(J,I6,K)*DB(J,I6,K)
1111 GO TO (1124,1112,1120),ISYMI
1112 IF(I.EQ.IVX) GO TO 1116
      I8=(KB-1)*IVX+IVX
      WRK2(JVX)=WRK2(JVX)+PC8(I,I8,K)*DR(JVX+1,I1,K) ——③
1116 IF(I.NE.IVX) GO TO 1124
      KB1=(KB-1)*IVX
C----- DO 1118 J=1,JVX-1
      I8=KB1+J
C----- 1118 WRK2(J)=WRK2(J)+PC8(JVX,I8,K)*DL(J,I5,K)
      GO TO 1124
1120 I8=(KB-1)*IVX+IVX+1-I
      WRK2(JVX)=WRK2(JVX)+PC8(JVX,I8,K)*DR(JVXP1,I1,K)
C----- 1124 DO 1126 J=1,JVX
C----- 1126 TSOUR8(J)=WRK2(J)

```

Fig. 3.2(b) Part of subroutine LOU2 (restructured program)

```

DO 102 J = 1,JVX
M = NCOMP(J,I,KB)      _____③
T18 = PC8(J,I1, K)      _____②
PFVOLB(M,K) = PFVOLB(M,K)+T18*PVOL(J,I,KB)      _____④
IF (J.EQ.1) XL8(1,K) = XL8(1,K)+T18*DR(J,I1, K)
IF (J.EQ.JVX) XL8(3,K) = XL8(3,K)+T18*DR(J+1,I1, K)
IF (NDIM.LE.1) GO TO 101
IF (I.EQ.1) XL8(2,K) = XL8(2,K)+T18*DL(J,I4, K)
IF (I.NE.IVX) GO TO 100
XL8(4,K) = XL8(4,K)+T18*DL(J,I5, K)
100 CONTINUE
IF (NDIM.LE.2) GO TO 101
IF (KB.EQ.1) XL8(5,K) = XL8(5,K)+T18*DB(J,I1, K)
IF (KB.EQ.KBVX) XL8(6,K) = XL8(6,K)+T18*DB(J,I6, K)
101 CONTINUE
IF (IRUD.LE.0) GO TO 102
C
C   CALCULATE LOSSES TO ROU ZONE.
IF (T18.NF.0.0) GO TO 102
ARS8(M,K) = ABS8(M,K)+TSOUR8(J)
IF (J.GT.1) ARS8(M,K) = ARS8(M,K)+PC8(J-1,I1, K)*DR(J,I1, K)
IF (J.LT.JVX) ARS8(M,K) = ARS8(M,K)+PC8(J+1,I1, K)*DR(J+1,I1, K)
IF (INR8.NE.2) GO TO 102
IF (J.EQ.1) ARS8(M,K) = ARS8(M,K)+PC8(JVX,I1, K)*DR(JVXP1,I1, K)
IF (J.EQ.JVX) ARS8(M,K) = ARS8(M,K)+PC8(1,I1, K)*DR(1,I1, K)
102 CONTINUE

```

Fig. 3.3(a) Subroutine LEK2 (original program)

COMMON/WORK/WRK1(120),WRK2(120),WRK3(120),WRK4(120),WRK5(120)

```

----- DO 1020 J=1,JVX
WRK2(J)=PC8(J,I1,K)      _____②
----- 1020 WRK1(J)=WRK2(J)*PVOL(J,I,KB)
----- DO 1021 J=1,JVX
M=NCOMP(J,I,KB)      } _____④
----- 1021 PFVOLB(M,K)=PFVOLB(M,K)+WRK1(J)      }

----- DO 1038 J=1,JVX
----- 1038 WRK3(J)=CVMGT(0.0,TSOUR8(J),WRK2(J).NE.0.0)
----- DO 1040 J=2,JVX
----- 1040 WRK3(J)=WRK3(J)+CVMGT(0.0,PC8(J-1,I1,K)*DR(J,I1,K),WRK2(J).NE.0.0)
----- JVX1=JVX-1      } _____①
----- DO 1042 J=1,JVX1
----- 1042 WRK3(J)=WRK3(J)+CVMGT(0.0,PC8(J+1,I1,K)*DR(J+1,I1,K),WRK2(J).NE.
----- *          0.0)      }

----- DO 1048 J=1,JVX
M=NCOMP(J,I,KB)      } _____⑤
----- 1048 ARS8(M,K)=ABS8(M,K)+WRK3(J)

```

Fig. 3.3(b) Part of subroutine LEK2 (restructured program)

3.3.1 形状、次元(入力データ)による判定

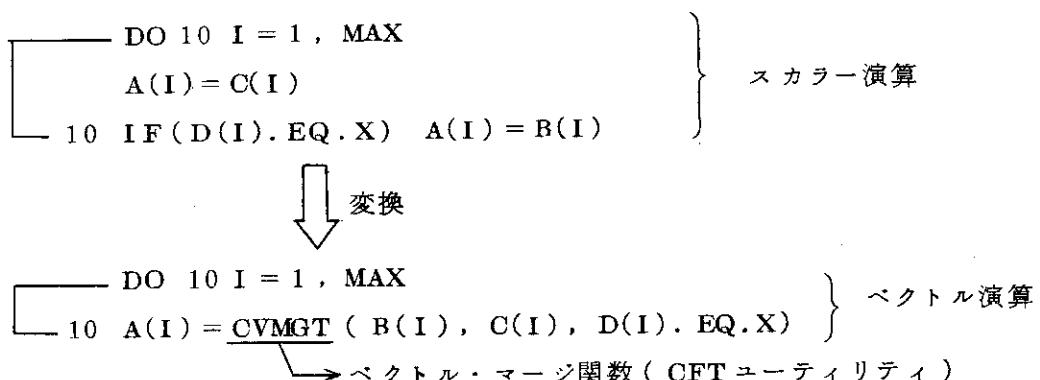
まずFig. 3.2(a)をみる。これはLOU 2のオリジナル・プログラムの一部である。この部分はステートメント番号109の大きなColumn方向のDOループ一つで形成されている。そしてその中には多くのIF文や間接アドレスの参照を含んでいる。このうちGeometryやDimension等に関するIF文はISHR, NDIM, ISYM1を参照しているものもあり(一部計算型GOTO文を含む),これらの値はループ中で変化しない。従って例えば同図(a)の①に関してはループを分割する事により同図(b)の①の如くベクトル化ができる。また以下に示すベクトル化でも同様であるがDOループを分割するために同図(b)の⑥の様にループ間のつなぎの一時的な配列を用意する必要がある。但しこの配列はこのサブルーチンでローカルなものであり, ラベル付きコモンで宣言しておけば他のサブルーチンでも共用できるので, 全体的にみてそれほど大きなメモリを必要としない。本来はこれらの配列はRowの数により正確にメモリ内に割当てなければならないが, 現在は便宜上大きさを固定している。

3.3.2 境界条件(DOループ制御変数)による判定

境界条件を取扱うための判定は3次元モデルの場合, Column方向(この場合ではDOループの始点, 終点), RowおよびPlane方向の両端の場合とが考えられる。Row, Plane方向についてはその判定のために参照される変数はDOループの中で一定であるので3.3.1項の場合と同様の取扱いができる(例えばFig. 3.2(a)(b)の②)。またColumn方向の境界の場合はDOループの始点, 終点であるのでそのインデックスで調整すればよい。例えば同図(a)の③の判定は同図(b)の③の様に分割する事によりベクトル化できる。

3.3.3 中性子束分布による判定

中性子束はDOループの中でみると定数ではない。従って中性子束の値を参照するIF文があると, ループの分割によるベクトル化は不可能である。例えばFig. 3.3(a)はLEK 2のオリジナル・プログラムであるが, このサブルーチンはステートメント番号102の一つの大きなDOループで形成されている。同図(a)の①に現われるT18は同②で定義されているPC8(J, I1, K)の中性子束分布でありこれはDOループのインデックスJをもつて値はループの中で変化する。この様なループをベクトル化するために以下に示すベクトル演算用ユーティリティ^[6]が用意されている。



ここで $D(I) = X$ のとき $B(I) \rightarrow A(I)$

$D(I) \neq X$ のとき $C(I) \rightarrow A(I)$

この CVMGT は CFT がもつベクトル・マージ関数といわれるユーティリティで、上の様に変換する事によりこのループは完全にベクトル化される。従って同図(a)の①の部分は同図(c)の①の様に変型される。ここで CVMGT の第 3 パラメータに含まれる WRK 2 は同②で予め定義してある。

3.3.4 物質領域(ゾーン)に対する定義・参照

拡散コードでは通常空間的なメッシュ(セル)に対して領域(ゾーン)を定義している。この領域はセルの物質を表わすものである。そして各セルと領域との対応づけのためのテーブルをもつ。Fig. 3.2(a)の④の NCOMP (J, I, KB) がそのテーブルで J, I, KB がセルを示し左辺の M がそのセルの領域番号である。M は同図(a)の⑤の SHRAB で参照される。この様な間接アドレスによる配列の参照もしくは定義はたとえ DO ループ中に IF 文がなくてもベクトル化はできない。そのため CFT では次の様なベクトル化のためのユーティリティを用意している。

(間接アドレスによる参照の場合)

$\boxed{\quad}$ DO 10 I = 1, MAX $\boxed{\quad}$ 10 A(I) = B(INDEX(I))	} スカラー演算
	

CALL GATHER(MAX, A, B, INDEX) } ベクトル演算

ここで MAX : 配列の長さ

A : 出力される配列

B : もとの配列

INDEX : アドレスを示すテーブル

(間接アドレスによる定義の場合)

$\boxed{\quad}$ DO 10 I = 1, MAX $\boxed{\quad}$ 10 A(INDEX(I)) = B(I)	} スカラー演算
	

CALL SCATTER(MAX, A, INDEX, B) } ベクトル演算

ここで MAX : 配列の長さ

A : 出力される配列

INDEX : アドレスを示すテーブル

B : もとの配列

従って同図(a)の④⑤は GATHER を用いる事により同図(b)の④の如く WRK 1 に置換えられ、同図⑤での WRK 1 を参照する DO ループはベクトル化できる。また GATHER の中も長さ JVX のベクトル演算として実行される。

次に、気をつけなければいけないのは Fig. 3.3(a) の③④⑤である。ここも前述の場合と同様 IC 間接アドレスによる定義、参照であるが、ここで定義された M は PFVOL 8, ABS 8 で参照

され、且つこれらが以下に示す様に左辺に表われている。

```

DO 10 J = 1, JVX
M = NCOMP ( J, I, KB )
10 PFVOL 8 ( M, K ) = PFVOL 8 ( M, K ) + T 18 * PVOL ( J, I, KB )

```

この様な時、 $NCOMP (J, I, KB)$ の値が $J = 1 \sim JVX$ の中で重複しなければ GATHER, SCATTER を使ってベクトル化できるが、実際は同じ領域番号が複数個のセルに現われるので使用できない。以上より DO ループの負荷を極力減らした形で同図(b)④⑤の如くスカラー演算で行なう。ただし、F75APUの場合には、リストベクトル(間接アドレス参照用インデックスベクトル)を用いてベクトル演算できる。

以上記した方法で Table 3.1 に掲げたベクトル化対象サブルーチンのうち RELX 以外についてほぼ満足できる程度にベクトル化を終了した。それはベクトル化後の全 DO ループ 43 個に対し 38 ループについてベクトル化ができ、個数としては全体の約 88% に相当する。その結果を Table 3.3 に示す。

Table 3.3 The number of vectorizable DO loops

サブルーチン	オリジナル プログラムの DO ループ数	ベクトル化後の DO ループ数		比率 (%)
		ベクトル演算	スカラー演算	
FOU2	2**	4	0	100
INR2***				
LOU2	4	23	3	88
RDUE	1	3	0	100
LEK2	1	8	2	80
合 計	8	38	5	88

* ベクトル化後の DO ループの個数を用い次式で定義する。

ベクトル演算 / (ベクトル演算 + スカラー演算)

** このうち 1 ループはオリジナルプログラムで既にベクトル化されている。

*** INR2 は制御ルーチンのためベクトル化効果があまりない(全体のステートメントに比してベクトル化を対象とする DO ループが少ない)ので、ここでは省略する。

3.4 解法の変更によるベクトル化

次に RELX のベクトル化について考える。オリジナル・プログラムにおける RELX はベクトル化されていないが、その理由は DO ループの中には IF 文を含む他、3.3 節に記したベクトル化条件のうち左辺、右辺および前後のステートメントの配列の添字に従属関係があるためである。従属関係とは次の様なことをいう(詳細は付録 A.2.2 を参照のこと)。

- DO ループの中のステートメントの実行順序はスカラー演算とベクトル演算とは異なる。
- 従って配列の添字のつけ方によっては配列要素を参照するときその値が別のステートメ

ントでまだ定義されていなかったり、すでに壊されていたりする。これを従属関係という。

- この様な場合には当然スカラー演算で実行した場合とベクトル演算で実行した場合とは結果が異なる。
- プログラムはスカラー演算での結果と同じになる事を期待するわけで、従属関係があるとその DO ループはベクトル化できない。

さて Fig. 3.4(a) の RELX のオリジナル・プログラムを掲げる。実際は加速のためのステートメントがあるがここでは問題を単純化するために加速しない（加速係数 1.0）場合のみに着目する。RELX の機能としては 1 次元に変換された三点階差の拡散方程式を Forward/Backward 法で解くもので詳細は 2.2 節に記されている。

同図(a)において次のステートメントに従属関係が発生していることが解る。

- line no. 5 の左辺 BET(J) と右辺 BET(J-1)。
- line no. 6 の左辺 DEL(J) と line no. 4 の右辺 DEL(J-1)。
- line no. 11 の左辺 P(J) と右辺 P(J+1)。

これらはもしベクトル化するといずれも右辺の値を参照するとき、まだ J-1 の要素 (P) については DO ループのインデックスが減少する (J+1 の要素) が定義されていないので、スカラー演算とは異なる答を算出してしまう。従って 2 つの DO ループはいずれもベクトル化されない。

ここで、この様に従属関係が発生している場合の DO ループのベクトル化を考える。まずいえる事は 3.3 節で示した様な DO ループの分割やベクトル演算用ユーティリティを用いる事だけではベクトル化は不可能である。従って従属関係が起らない DO ループへの変換、つまり解法の変更をせざるを得ない。別な解法を考える時重要な事は、Forward/Backward 法は前進、後退各々 1 回 DO ループを実行するだけで繰返計算する事なく三点階差式の解が得られる点である。従ってベクトル化はできても繰返計算が必要になってしまい解法を用いると計算時間においてはたとえスカラー演算でも Forward/Backward 法が有利であると考えられる。以上よりベクトル化でき且つ繰返計算のない解法という条件を満たすものとして Cyclic Reduction 法がある⁽⁷⁾。この方法は三重対角行列のうち 3 行を 1 組として要素の消去を行なうと、それもまた元数が半分に減った三重対角行列になる (Reduction) 事に着目した方法で、この Reduction を繰返す事により最終的には行列は 1 又は 2 本の式で表わされる。その後この式を逆にたどる事により解が得られるものである。この Cyclic Reduction 法はベクトル化は出来るが、式の型は複雑になりプログラミングした場合の演算子の数は大巾に増加する。ベクトル長も Reduction する度に半分になるので非常に大きな元数の行列を解く際は有効であろうが、VENTURE で扱う元数 (Column 方向のメッシュ数) は 10 ~ 100 程度である事を考えるとあまり効果がない様に判断できる。

さて、Forward/Backward 法では各方向 1 回づつの sweep で正確な解が得られる訳であるが、VENTURE における RELX の位置づけを考えてみるとそれは inner iteration の中の 1 ステップにすぎない。つまり、収束計算の途上では RELX で得た中性子束分布はまだ計算体

系内での中性子束分布の解（収束値）ではないといえる。従って RELX ではそれほど正確な計算をする必要はなく inner, outer 各 iteration 共収束した時に解が得られればよいはずである。この様な背景より以下の方法を試みる。

- Forward/Backward 法を強制的にベクトル化する。このベクトル化した RELX を RELXA とする。
- そのため Fig. 3.4(a) で従属関係が発生している DEL, BET, P についてはその右辺に前回の inner iteration での値を用いる。
- このうち P については現在 All Stored Mode のみに着目しているので問題はないが、 DEL, BET については RELX 内における Column 方向のみの一時的な配列であるため別に空間メッシュおよびエネルギー群数分の配列を用意する必要がある。
- 各 inner iteration の 1 回目はオリジナル（スカラー演算）の RELX を呼びだし DEL, BET および P の初期値をセットする。2 回目以降はベクトル化した RELXA を呼び出す。

この方法を用いると収束回数は増すが、ベクトル化によりその分以上高速化されればその効果は上がるであろう。また収束判定は前回と今回の inner iteration での中性子束分布の差でみるので、収束してしまえば許容誤差範囲内でオリジナルの RELX を用いたものと一致するはずである。Fig. 3.4(b) はベクトル化した RELXA を、Fig. 3.5 は iteration における流れを示す。

line no.

1.	BET(1) = TS(1)/D(2)	} 初期セット
2.	DEL(1) = D(2)/TL(1)	
3.	DO 10 J = 2, JMX	} Forward Sweep (スカラー演算)
4.	T = D(J)*DEL(J-1)	
5.	BET(J) = (TS(J)+BET(J-1)*T)/D(J+1)	
6.	DEL(J) = D(J+1)/(TL(J)-T)	
7.	10 CONTINUE	} Backward Sweep (スカラー演算)
8.	P(JMX) = BET(JMX)*DEL(JMX)	
9.	DO 20 JJ = 2, JMX	
10.	J = JMX-JJ+1	
11.	P(J) = DEL(J)*(P(J+1)+BET(J))	
12.	20 CONTINUE	• オリジナルプログラムでは加速のためのステートメントがあるが、ここでは略いてある。 • BET, DELは長さ JMX の一時的な配列である。 • Pは空間メッシュ、エネルギー群数分の大きさを持つ配列で、パラメータで1 Row 分だけ送られる。

Fig. 3.4(a) Part of subroutine RELX (original program)

line no.

1.	BETR(1) = TS(1)/D(2)	} 初期セット
2.	DELR(1) = D(2)/TL(1)	
3.	DO 10 J = 2, JMX	} Forward Sweep (ベクトル演算)
4.	T = D(J)*DEL(J-1)	
5.	BETR(J) = (TS(J)+BET(J-1)*T)/D(J+1)	
6.	DELR(J) = D(J+1)/(TL(J)-T)	
7.	10 CONTINUE	} Backward Sweep (ベクトル演算)
8.	PR(JMX) = BETR(JMX)*DELR(JMX)	
9.	DO 20 JJ = 2, JMX	
10.	J = JMX-JJ+1	
11.	PR(J) = DELR(J)*(P(J+1)+BETR(J))	
12.	20 CONTINUE	} 置き換え (ベクトル演算)
13.	DO 30 J = 1, JMX	
14.	P(J) = PR(J)	
15.	BET(J) = BETR(J)	
16.	DEL(J) = DELR(J)	
17.	30 CONTINUE	• BETR, DELR, PRは長さ JMX の一時的な配列である。 • BET, DEL, Pは空間メッシュ、エネルギー群数分の大きさを持つ配列で、パラメータで1 Row 分だけ送られる。

Fig. 3.4(b) Subroutine RELXA (restructured program)

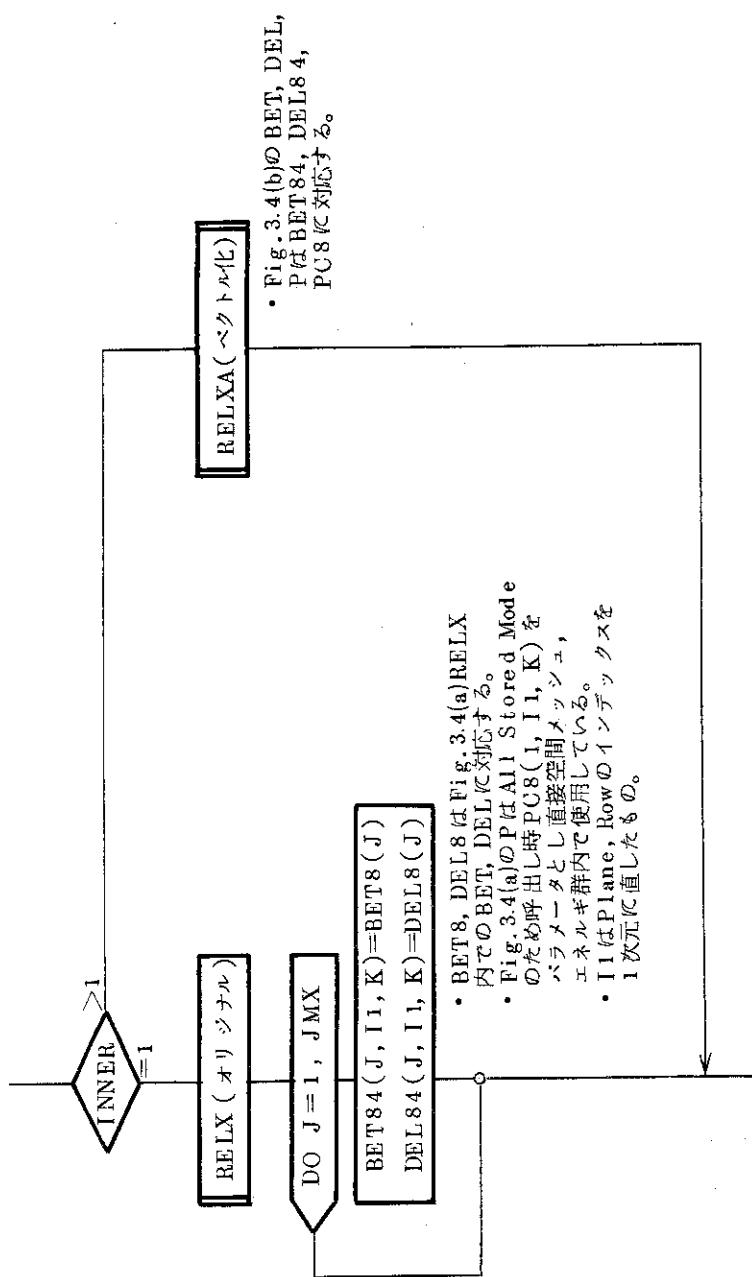


Fig. 3.5 Flow chart of the part of subroutine (RELX, RELXA)
call in subroutine INR2

4. ベクトル演算化の効果

4.1 サンプル問題

今回行なったベクトル化の効果を、次の三つの問題について調査した。

問題 1.

VENTURE のマニュアルにあるサンプル問題で、エネルギー 2 群、空間メッシュ $9 \times 9 \times 5$, X-Y-Z の 3 次元問題である。炉心は 3 領域から構成され、実際には $1/8$ 炉心が解かれる。なおこの例は、バックリングによる臨界調整（バックリングサーチ）問題である。Fig.4.1 に体系の概略を示す。

問題 2.

高速炉の臨界実験を模擬した問題で、エネルギー 2 群、空間メッシュ 58×50 , R-Z 2 次元形状の例である。炉心は 9 領域より構成され、下半分が解かれる。Fig.4.2 に体系の概略を示す。

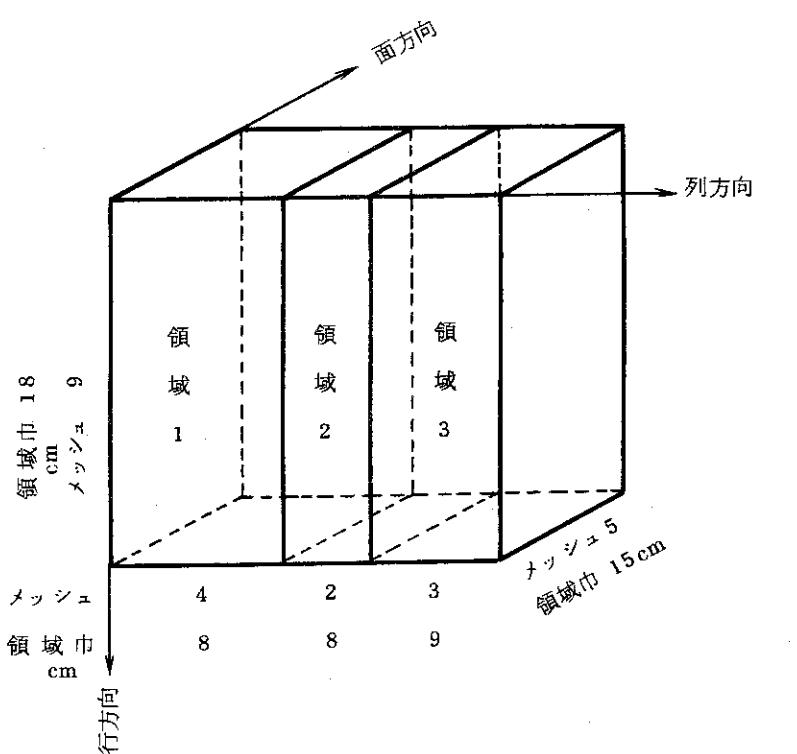
問題 3.

CITATION のサンプル問題集にある問題で、エネルギー 3 群、空間メッシュ $13 \times 13 \times 18$, X-Y-Z の 3 次元の固有値問題である。炉心は 7 領域より構成され、 $1/4$ 炉心がとされる。なおこの例では、炉心は対角線に関する対称性を有している。Fig.4.3 に体系の概略を示す。

Table 4.1 では、以上の例題の特性を比較した。

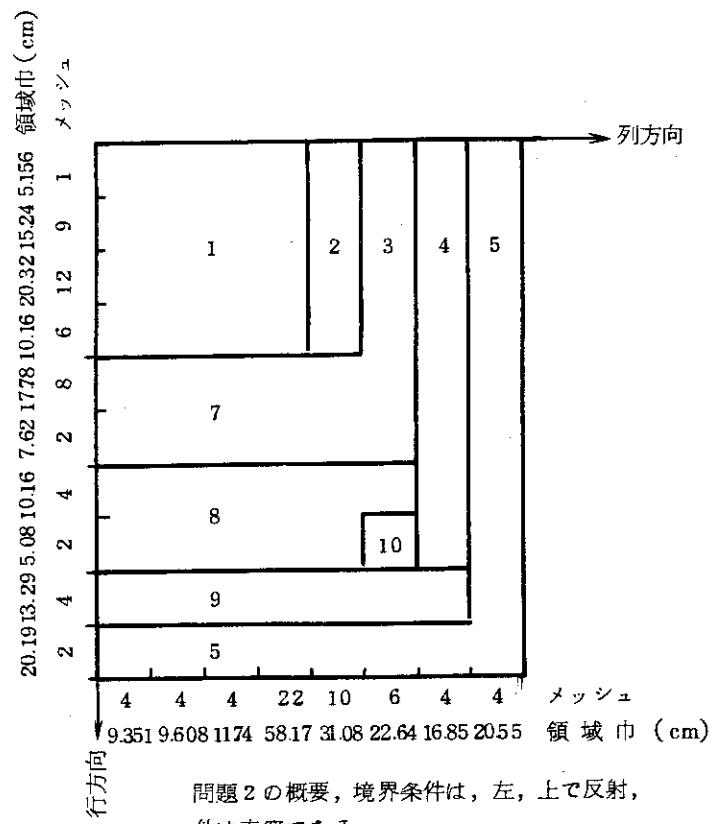
Table 4.1 Sample problems for the test of vectorized VENTURE code

問題	形状	次元	群数	メッシュ数	領域数	注
1	X-Y-Z	3	2	$9 \times 9 \times 5 = 405$	3	バックリングサーチ問題
2	R-Z	2	2	$58 \times 50 = 2900$	9	
3	X-Y-Z	3	3	$13 \times 13 \times 18 = 3042$	7	



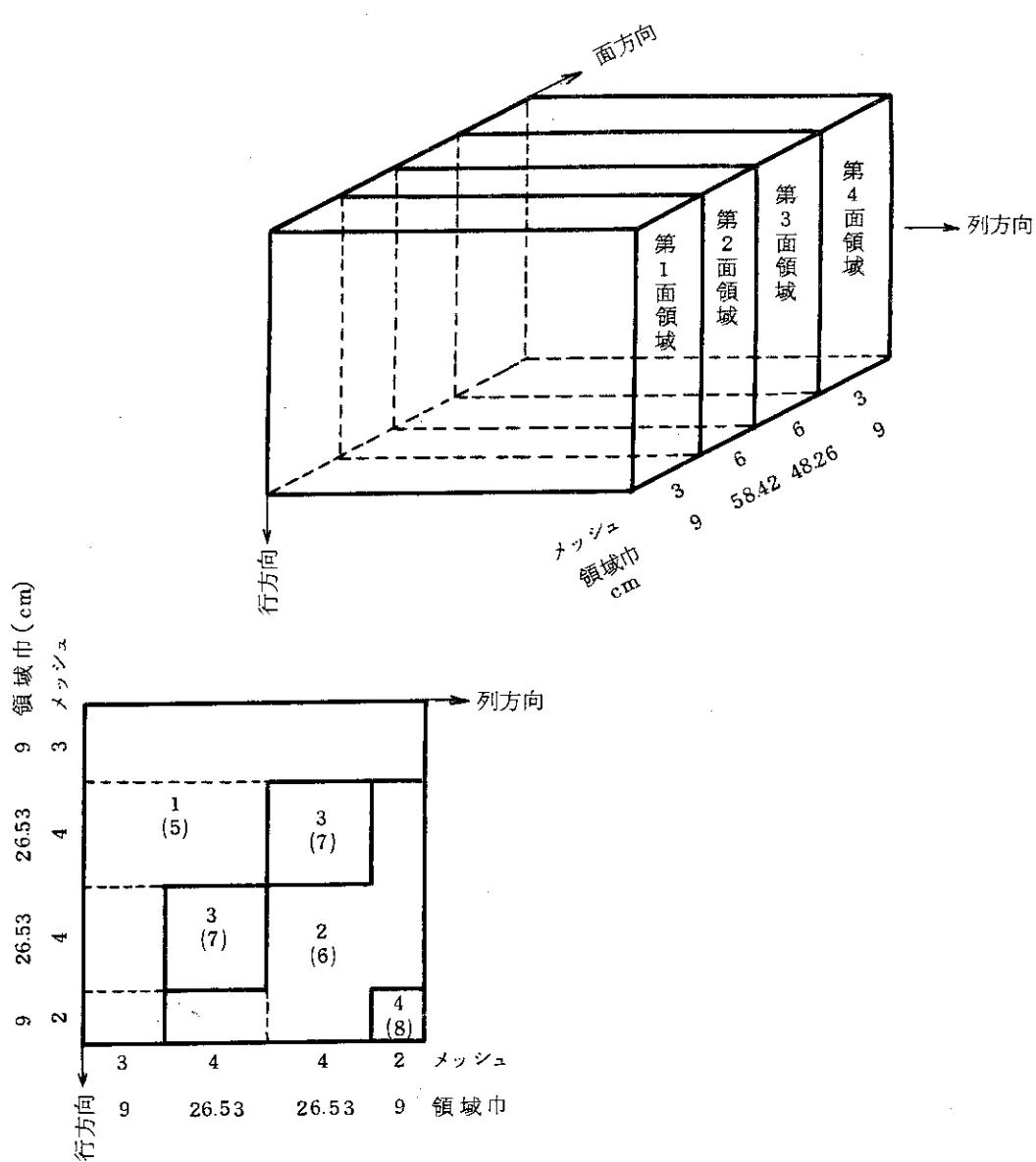
問題 1 の概要、境界条件は、左、底面が反射条件であるとは真空条件である。

Fig. 4.1 Coarse mesh sketch of sample problem no. 1



問題 2 の概要、境界条件は、左、上で反射、
他は真空である。

Fig. 4.2 Coase mesh sketch of sample problem no. 2



問題3 の概要、領域の配置は第1面領域では1，
 第2面領域では下図の上の領域番号、第3面領域
 では下図の下の領域番号、第4面領域では5であ
 る。
 境界条件は、上、右面で反射、他は真空条件であ
 る。

Fig. 4.3 Coarse mesh sketch of sample problem no. 3

4.2 CRAY-1によるベクトル化効果の検討

3章で記したベクトル化プログラムと4.1節で記したサンプル問題とを用いてCRAY-1にておけるベンチマーク・テストを実施し、その結果を検討した。まず使用したプログラムとサンプル問題についてまとめる。

(プログラム)

- 全て All Stored Mode を利用
- 次の3バージョンのプログラムを使用

① オリジナル・バージョン

オリジナル・プログラムを使用したもの。オリジナルのままでベクトル化されるDOループもあるがそれは僅かであり殆んどがスカラー演算である。他のベクトル・バージョンと比較する場合これをスカラー・バージョンとして支障はない。またこのバージョンを用いた計算の Inner Iteration は全て自動加速とした。

② RELX以外ベクトル化

3.3節で記したDOループの分割等によるベクトル化のみを施したもの。計算過程および最終結果はオリジナル・バージョンのものと完全に一致する。このバージョンを用いた計算の Inner Iteration は全て自動加速とした。

③ 全てベクトル化

3.3節で記したDOループの分割等によるベクトル化および3.4節に記したRELXのベクトル化も施したもの。オリジナル・バージョンと比べて収束回数は増加し、計算過程は一致しない。但し結果は許容誤差範囲内で一致する。このバージョンを用いた計算では Inner Iteration の加速は行なわない(加速係数 1.0 とした。加速係数の検討は次節を参照の事)。

(サンプル問題)

- Table 4.1に掲げた問題1, 2および3について各々 Column 方向のメッシュ数を体系の大きさはそのままにして変化させた(メッシュ巾が変化する)。

4.2.1 CRAY-1によるベクトル化効果

以上3つのバージョンのプログラムと3つのサンプル問題を組合せて計算を実施した。その結果を全CPU時間、Outer Iterationの繰り返し回数でまとめたものをTable 4.2に示す。またTable 4.3～4.10にベクトル化対象サブルーチン毎のCPU時間を記したものを探る。これらの表における“その他”とは上段に掲げたベクトル化対象サブルーチン以外の全てのルーチンを示している。

まず基本サンプル問題でのメッシュ数を用いたものについて検討する。Table 4.2をみてみるとおり各問題共「RELX以外ベクトル化」したものでは「オリジナル」に対して約60%の

CPU時間で計算できる。これは例えば問題2に着目するとTable 4.6に示されるとおりLOU2とLEK2のベクトル化効果が非常に大きいためである。「RELX以外ベクトル化」したバージョンは「オリジナル」と結果は完全に一致する事は先に述べた。つまりプログラムでとられている計算手法、手順は一切変更せずに全体で約40%計算時間が減少した意義は大きいと考えられる。

次に「全てベクトル化」したバージョンについて調べる。Table 4.2をみると「オリジナル」に対して約50%の計算時間である。RELXのベクトル化をする事による効果のみに着目すると約20%の減少である。Table 3.1に掲げたベクトル化対象サブルーチンの負荷度をみるとRELXのそれは大きくベクトル化効果は期待される訳だが、実際には繰り返し数が増している事と、初期値計算のために各Outer Iteration毎にオリジナルのRELXを呼びだしているためにそれほど顕著ではない。例えばTable 4.6をみるとベクトル化されたRELX（すなわち、RELXA）は1.9秒であるが、オリジナルのRELXも3.4秒消費しており合計で5.3秒を必要としている。これはオリジナルのRELXのみを使用した場合の10.4秒と比べると約50%の減少に留まり、LOU2のベクトル化効果の90%（同表9.5秒→0.8秒）に對してかなり低い。

また問題による違いも大きい。問題1、3ではRELXをベクトル化する事により繰り返し数は約12~14%増加するが、問題2では約76%も増えてしまう。問題3でメッシュ数を26にした例では収束に到っていない。この様にRELXを3.4節で示した方法でベクトル化する事は問題により収束性に大きな影響を与える。その収束性は体系の大きさ、メッシュ巾、核定数等によるものと予想されるが、一般的に定義する事は困難である。また加速係数の影響も考慮せねばならないが、これについては4.2.2節で簡単に述べる。

次にベクトル化対象サブルーチン個々のベクトル化効果を見る。Table 4.6を例にとりそれらを1回当たりのCPU時間に換算し、更にオリジナルを1.0として規格化するとTable 4.11の如くなる。これをみるとINR2以外はベクトル化の効果は顕著で計算時間はいずれも70~90%減少している。Table 3.2の負荷度と合わせてみるとLOU2とLEK2の効果が特に注目される。そしてこれらの全体に与える影響は既に述べたとおりである。RELXについては約90%減少しているが、実際にはオリジナル（スカラー）のRELXとの組合せで使用し、且つ繰り返し数も増すのでここでの比較はあまり意味がない。INR2についてはベクトル化する事により逆に計算時間が増すのはベクトル化したDOループの全体に対する割合がわずかであり、かえってベクトル化手続のためによるオーバーヘッドが増加したためであろう。

またベクトル長（Column方向メッシュ数）のベクトル化効果に与える影響を問題1および2に着目して検討する。Fig. 4.4は横軸にベクトル長を、縦軸に計算時間をオリジナルバージョンでの値を1.0にして規格化したものを示す。この図においてサブルーチンについては1回当たりの計算時間に換算したものである。尚INR2はTable 4.11に示す様にあまり効果がないので、RELXについては既述の様に無意味なのでここでは省いた。ベクトル長40~50でグラフが不連続となるのは問題が異なるためで、ここは破線で示した。まず各サブルーチンをみる

とベクトル長の影響が顕著なのが解る。しかし 60 メッシュを超えるとその効果は殆んどない。これは CRAY-1 のベクトルレジスタの語数（1 度に処理できる大きさ）が 64 語であることからうなづけよう。全体の計算時間（図中の○印）をみても同様にベクトル長の影響が現われている。「全てベクトル化」したバージョンのものは問題 1（ベクトル長 36）で約 6.1% CPU 時間が減少している。しかしこのバージョンは収束性があまり安定していないため問題 1 では良好であるが、問題 2 ではベクトル長が増すとかえって逆効果となっている。サブルーチン個々のベクトル長の効果に対して全計算時間の効果が少ないと（グラフの傾きが緩やか）のは高速化される事によりベクトル化したサブルーチン以外の部分の計算時間が逆に浮き出てくるためである。

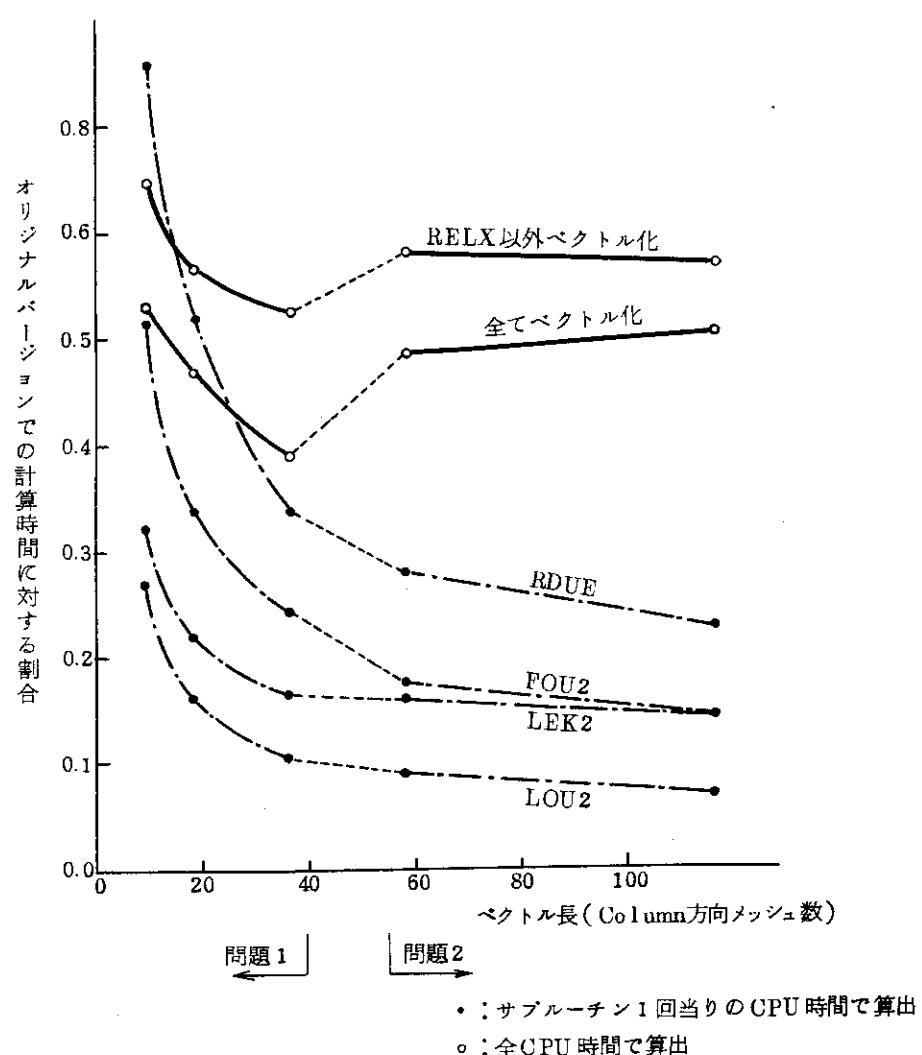


Fig. 4.4 Vectorization effect to the vector length by CRAY-1

Table 4.2 Computing time of all sample problems by CRAY-1

問 題	Column 方 向 メッシュ数	プログラム			オリジナル・バージョン			RELX以外ベクトル化			全てベクトル化		
		全CPU時間 (sec)	比率 (%)	繰返回数									
問題 1	9 **	2.603	100.0	33	1.674	64.2	33	1.382	53.0	37			
3 次元 X-Y-Z 2群	18	5.312	100.0	36	3.000	56.5	36	2.482	46.7	48			
	36	8.916	100.0	31	4.683	52.5	31	3.468	38.9	42			
問題 2	58 **	24.980	100.0	46	14.506	58.1	46	12.122	48.5	81			
2 次元 R-Z 2群	116	54.291	100.0	44	30.938	57.0	44	27.391	50.5	95			
											収束せず		
問題 3	13 **	28.224	100.0	36	16.231	57.5	36	12.426	44.0	41			
3 次元 X-Y-Z 3群	26	66.522	100.0	44	34.832	52.4	44						
	52	122.285	100.0	41	61.073	49.9	41	62.195	50.9	80			

* : オリジナル・バージョンの全CPU時間を100.0とした比率

** : 基本となるサンプル問題でのメッシュ数

Table 4.3 Computing time of problem no.1(vector length:9) by CRAY-1

	オリジナルバージョン			RELX以外ベクトル化			全てベクトル化		
	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)
FOU2	0.0979	3.75	34	0.0506	3.02	34	0.0566	4.09	38
INR2	0.0563	2.16	68	0.0602	3.59	68	0.0829	6.00	76
LOU2	1.0451	40.08	11970	0.2798	16.71	11970	0.3135	22.69	13410
LEK2	0.1844	7.07	2970	0.0591	3.53	2970	0.0661	4.79	3330
RDU2	0.0042	0.16	90	0.0036	0.21	90	0.0036	0.26	90
RELX	0.8748	33.55	11880	0.8759	52.32	11880	0.2423	17.54	3330
RELX(Vector)	—	—	—	—	—	—	0.2567	18.58	9990
その他	0.3448	13.23	—	0.3450	20.62	—	0.3601	26.05	—
合計	2.6076	100.00	—	1.6741	100.00	—	1.3816	100.00	—
比率	1.0	—	—	0.6420	—	—	0.5298	—	—
繰返回数	3	—	—	33	—	—	37	—	—

Table 4.4 Computing time of problem no.1 (vector length:18) by CRAY-1

	オリジナルベクトル			RELX以外ベクトル化			全ベクトル化		
	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)
FOU2	0.2036	3.83	37	0.0686	2.29	37	0.0908	3.66	49
INR2	0.0622	1.17	74	0.0676	2.25	74	0.1109	4.47	98
LOU2	2.2323	42.03	13050	0.3560	11.87	13050	0.4730	19.06	17370
LEK2	0.3887	7.32	3240	0.0847	2.82	3240	0.1128	4.55	4320
RDUE	0.0085	0.16	90	0.0044	0.15	90	0.0044	0.18	90
RELX	1.8521	34.87	12960	1.8540	61.80	12960	0.6047	24.37	4320
RELX(Vector)	—	—	—	—	—	—	0.4109	16.56	12960
その他	0.5643	10.62	—	0.5648	18.82	—	0.6743	27.15	—
合計	5.3116	100.00	—	3.0000	100.00	—	2.4817	100.00	—
比率	1.0	—	—	0.5648	—	—	0.4672	—	—
繰回国数	36			36			48		

Table 4.5 Computing time of problem no.1 (vector length:36) by CRAY-1

	オリジナルバージョン			RELX以外ベクトル化			全てベクトル化		
	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)
FOU2	0.3431	3.85	32	0.0831	1.77	32	0.1117	3.22	43
INR2	0.0557	0.62	64	0.0595	1.27	64	0.1051	3.03	86
LOU2	3.8168	4.281	11250	0.3979	8.50	11250	0.5378	15.51	15210
LEK2	0.6578	7.38	2790	0.1080	2.31	2790	0.1463	4.22	3780
RDUE	0.0169	0.19	90	0.0057	0.12	90	0.0057	0.16	90
RELX	3.1400	35.22	11160	3.1425	67.10	11160	1.0511	30.31	3780
RELX(Vector)	—	—	—	—	—	—	0.4583	13.21	11340
その他	0.8861	9.93	—	0.8863	18.93	—	1.0523	30.34	—
合計	8.9164	100.00	—	4.6829	100.00	—	3.4681	100.00	—
比率	1.0	—	—	0.5252	—	—	0.3890	—	—
繰回国数	31			31			42		

Table 4.6 Computing time of problem no.2 (vector length:58) by CRAY-1

	オリジナルバージョン			RELX以外ベクトル化			全てベクトル化		
	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)
FOU2	0.7639	3.06	47	0.1322	0.91	47	0.2303	1.90	82
INR2	0.1087	0.43	94	0.1178	0.81	94	0.2762	2.28	164
LOU2	9.5010	38.04	25400	0.8486	5.85	25400	1.4903	12.29	44650
LEK2	1.4009	5.61	4600	0.2232	1.54	4600	0.3928	3.24	8100
RDUE	0.0294	0.12	100	0.0082	0.06	100	0.0082	0.07	100
RELX	10.6404	42.60	25300	10.6388	73.34	25300	3.4067	28.10	8100
RELX(Vector)	—	—	—	—	—	—	1.8961	15.64	44650
その他	2.5353	10.15	—	2.5373	17.49	—	4.4213	36.47	—
合計	24.9795	100.00	—	14.5062	100.00	—	12.1220	100.00	—
比率	1.0	—	—	0.5807	—	—	0.4853	—	—
繰返回数	46	—	—	46	—	—	81	—	—

Table 4.7 Computing time of problem no.2 (vector length: 116) by CRAY-1

	オリジナルバージョン			RELX以外ベクトル化			全てベクトル化		
	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)
FOU2	1.4501	2.67	45	0.2110	0.68	45	0.4497	1.64	96
INR2	0.1268	0.23	90	0.1375	0.44	90	0.4273	1.56	192
LOU2	21.3898	39.40	28700	1.5439	4.99	28700	3.3248	12.14	61850
LEK2	2.6618	4.90	4400	0.3859	1.25	4400	0.8328	3.04	9500
RDUE	0.0587	0.11	100	0.0133	0.04	100	0.0133	0.05	100
RELX	23.9997	44.21	28600	24.0381	77.70	28600	7.9783	29.13	9500
RELX(Vector)	—	—	—	—	—	—	4.2767	15.61	52250
その他	4.6043	8.48	—	4.6080	14.90	—	10.0883	36.83	—
合計	54.2912	100.00	—	30.9376	100.00	—	27.3911	100.00	—
比率	1.0	—	—	0.5698	—	—	0.5045	—	—
繰回国数	44			44			95		

Table 4.8 Computing time of problem no.3 (vector length:13) by CRAY-1

	オリジナルベージョン			RELX以外ベクトル化			全てベクトル化		
	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)
FOU2	0.9460	3.35	37	0.3158	1.95	37	0.3584	2.88	42
INR2	0.4397	1.56	111	0.4835	2.98	111	0.6866	5.53	126
LOU2	12.2588	43.43	101790	2.4549	15.12	101790	2.7924	22.47	115830
LEK2	2.1155	7.50	25272	0.5211	3.21	25272	0.5936	4.78	28782
RDUE	0.0440	0.16	702	0.0298	0.18	702	0.0298	0.24	702
RELX	9.8559	34.92	101088	9.8587	60.74	101088	2.8062	22.58	28782
RELX(Vector)	—	—	—	—	—	—	2.3410	1.84	86346
その他	2.5644	9.08	—	2.5673	15.82	—	2.8190	22.68	—
合計	28.2243	100.00	—	16.2311	100.00	—	12.4264	100.00	—
比率	1.0	—	—	0.5751	—	—	0.4403	—	—
繰返回数	36	36	36	36	36	36	36	36	41

Table 4.9 Computing time of problem no.3 (vector length:26) by CRAY-1

	オリジナルバージョン			RELX以外ベクトル化			全てベクトル化		
	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)
FOU2	2.2381	3.36	45	0.4889	1.40	45			
I NR 2	0.5575	0.84	135	0.5911	1.70	135			
L OU 2	29.4954	44.34	124254	3.6858	10.58	124254			
L EK 2	5.0449	7.58	30888	0.9089	2.61	30888			
R DUE	0.0870	0.13	702	0.0388	0.11	702			
RELX	23.5658	35.43	123552	23.5888	67.72	123552			
RELX(Vector)	—	—	—	—	—	—			
その他	5.5331	8.32	—	5.5296	15.88	—			
合 計	66.5218	100.00	—	34.8320	100.00	—			
比 率	1.0	—	—	0.5236	—	—			
繰返回数	4			44					

Table 4.10 Computing time of problem no. 3 (vector length:52) by CRAY-1

	オリジナルバージョン				RELX以外ベクトル化				全てベクトル化		
	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)	CPU時間 (sec)	比率 (%)	呼出回数 (回)		
FOU2	4.1069	3.36	42	0.6687	1.09	42	1.2893	2.07	81		
INR2	0.5411	0.44	126	0.5744	0.94	126	1.5540	2.50	243		
LOU2	54.5920	44.64	115830	4.7735	7.82	115830	9.2862	14.93	225342		
LEK2	9.2804	7.59	28782	1.3559	2.22	28782	2.6459	4.25	56160		
RDUE	0.1717	0.14	702	0.0539	0.09	702	0.0539	0.09	702		
RELX	43.5711	35.63	115128	43.6293	71.44	115128	21.2521	34.17	56160		
RELX(Vector)	—	—	—	—	—	—	8.2068	13.20	168480		
その他	10.0219	8.2	—	10.0175	16.4	—	17.9067	28.79	—		
合 計	122.2850	100.00	—	61.0732	100.00	—	62.1949	100.00	—		
比 率	1.0	—	—	0.4994	—	—	0.5086	—	—		
繰返回数	41				41				80		

Table 4.11 Vectorization effect of each subroutine
 in problem no.2 (vector length:58)
 by CRAY-1

サブルーチン	オリジナル	ベクトル化
FOU2	1.0	0.1731
INR2	1.0	1.0837
LOU2	1.0	0.0893
LEK2	1.0	0.1593
RDUE	1.0	0.2789
RELX	1.0	0.1010

- ベクトル長58の場合
- サブルーチン1回当たりの計算時間をオリジナルプログラムの値で規格化したもの。

4.2.2 加速係数の検討

2.2節で述べたように、VENTUREではInner Iterationの過程で、中性子束に対して、Over-Relaxation法による加速が行なわれる。この時用いられる加速係数(2.9式の β)は通常はコード内で最適値が求められる。しかし、ベクトル化のため、Inner Iterationの数値解法に変更が加えられた。すなわち、原VENTUREでは、一行分の中性子束は与えられた源に対して正確に求まるが、ベクトル化VENTUREでは、求まる中性子束は正確ではなく、一回前の繰返し計算で求まった中性子束に依存した形で求められる(3.4節参照)。このため、原コードで求まる β は、ベクトル化VENTUREでは最適値でなくなる可能性がある。そこで、問題1、を用いて検討を加えた。

まず原VENTUREコードと同一の手法で β を決めかつ、ベクトル化したVENTUREと原コードの比較を行なった(Table 4.12)。ベクトル化VENTUREではOuter Iterationの収束回数は、原コードの33から37に増加した。一方ベクトル化VENTUREで、加速因子を1.0とおいた場合(Table 4.12)、すなわち、Over-Relaxationによる加速を行なわない場合の収束回数は、37であった。これから、問題1.ではあまり加速効果がないことがわかる。むしろ β を計算するに要する計算時間だけ、加速した場合の方が計算時間は長くなるようである。しかしこの問題は小型で収束しやすい問題であり、この結論は、一般化できない。

加速係数の影響を調べるために β を固定した計算を行なった(Table 4.12)。この結果Outer Iterationは、 $\beta = 1.6$ で最少(32回)で収束し、また計算時間も、加速のない場合に比べて88%に減少した。しかし β を固定する手法では全ての問題を収束させるのは困難であり、通常はVENTUREにより自動的に β を決めるオプションを用いるべきであろう。ちなみに、この問題1.でVENTUREにより決まる β の値は、約1.6からはじまり、Outer Iterationを重ねるとともに減少し、Outer Iterationの23回目から1.0となり以後は1.0である。

Table 4.12 Effect of accelerated coefficient on computing time

加速係数の計算時間に与える影響。
ケース1～10はベクトル化VENTUREによる計算、4.1節の問題1についての結果。

ケース	加速係数	収束回数	Inner Iteration \mathcal{O}		計算時間比
			計算時間 (CPU sec)	計算時間 (sec)	
1	0.6	41	0.557092	1.1165	
2	0.8	40	0.543895	1.0900	
3	1.0	37	0.498954	1.0000	
4	1.2	34	0.463705	0.9293	
5	1.4	35	0.476968	0.9559	
6	1.6	32	0.437420	0.8766	
7	1.8	50	0.677057	1.3569	
8	2.0	収束せず	—	—	
9	VENTUREによる自動決定	37	0.576692	1.1558	
10	原VENTURE	33	0.874842	1.7533	

4.3 F 75 APUによるベクトル化効果の検討

FACOM 230-75 CPUとAPU⁽⁸⁾（アレイプロセッサユニット）（以後F 75 APUまたはCPUと略称する）によって、サンプル問題についてテストランを行い、ベクトル化の効果を検討した。ここでは、使用したプログラム、実行結果の計算時間、ベクトル化の効果について述べる。なお、F 75 APU/CPU計算機システムとAP-FORTRAN^[9,10]については付録B.1, B.2で述べる。

4.3.1 F 75 APU/CPUのためのプログラム

プログラムは次の3種類のバージョンを使用した。ただし、CRAY-1で使用したプログラムと異なる点はあるが、ベクトル化の方法は本質的には同じである。

(1) FORTRANオリジナルバージョン

オリジナルのソースプログラムの内、アセンプラで書かれたサブルーチンについては、その機能をFORTRANで書き直し、すべてFORTRAN(FACOM 230 M-V FORTRAN V-H)のプログラムにした。

サブルーチンRELXの拡散方程式の解法は、オリジナル(Forward/Backward法(逐次解法))のままである。従って、サブルーチンRELXAは含まれていない。

翻訳は、FORTRANコンバイラを使用し、実行はCPUのみで行った。

(2) ベクトル化再構成バージョン

Inner Iterationで使用される6個のサブルーチン(INR2, LOU2, RELX, RELXA, LEK2, RDUE)について、3.3節で述べたDOループの分割によるベクトル化および、3.4節で述べたRELXの解法の変更(反復解法)によるベクトル化などの再構成を行ったプログラムである。言語はすべてFORTRANで記述し、次に述べるベクトル演算用のAP-FORTRANのプログラムへの変換を想定したものである。上記6個のサブルーチン以外のMAINを含むその他のルーチンについては、FORTRANオリジナルバージョンのままである。

再構成バージョンの翻訳は、Inner Iterationの6個のサブルーチンについては、次のベクトル演算用バージョンとの比較のためにAPコンバイラ(APで実行するためのオブジェクトコードに翻訳するコンバイラ)で行い、それ以外は、FORTRANコンバイラで行った。

従って、実行はCPUとAPUの両方で、タスクを交換しながら行われる。ただし、Inner Iterationの部分はAPコンバイラで翻訳されているのでAPUで処理されるが、プログラムがFORTRANなのでベクトル演算命令は実行されない。

(3) ベクトル演算用バージョン

前記のベクトル化再構成バージョンについて、Inner Iterationの6個のサブルーチンのうちRELXを除く5個のサブルーチンをFORTRANからAP-FORTRAN^[9,10]に変換したプログラムである。変換の方法については、付録B.2(Fig. B.2参照)で述べる。

上記6個のサブルーチン以外は、FORTRANのままである。これについては、すべてをAP-FORTRANに変換し、APコンバイラで翻訳したとしても、READ, WRITEなどの入出力やファイルの参照は、APUでは処理できず、CPUにタスクを移して行われるので、かえって

タスクスイッチ回数が増して処理が遅くなるからである。

ただし、計算時間が比較的かかり、タスクスイッチ回数の少ないサブルーチンについては、ベクトル化可能部分の割合を増す意味からも、AP-FORTRANへの変換を考慮する余地がある。

ベクトル演算用バージョンの翻訳は、Inner Iteration部分の RELX を含む 6 個のサブルーチンについては、AP コンバイラで行い、それら以外は FORTRAN コンバイラで行った。

実行は、CPU と APU の両方で行われ、AP-FORTRAN のプログラムの部分ではベクトル演算命令が実行される。

4.3.2 F 75 APU による計算時間

サンプル問題 1 の 3 次元 X-Y-Z 形状 2 群でベクトル長 9 の入力データによるテストランの結果を Table 4.13 に示す。なお、F 75 APU/CPU では VENTURE の標準指定である倍精度で計算を行った。その表に見られるように、ベクトル化のために再構成したプログラムは、CPU 時間と APU 時間の合計で 29.718 秒かかったに対して、Inner Iteration に含まれるサブルーチンを最大限にベクトル化したプログラムでは合計時間で 22.232 秒に短縮された。ただし、オリジナルプログラムの CPU 時間と比較して、再構成プログラムの計算時間が増加したのは、ベクトル計算のための前準備によるオーバーヘッドのためで、ベクトル長が長くなれば、その割合は小さくなる。

Outer Iteration に含まれるサブルーチンのうち、呼び出し回数の多い SOU 2, FOU 2 のサブルーチンについては、ベクトル化することによって、そのサブルーチンの 1 回の計算時間は短くなるが、全体の計算時間は長くなる。それは、呼び出している元のサブルーチンが CPU で実行されているので、ベクトル化して APU で実行すると、タスク・スイッチ回数が増加するためである。ただし、これもベクトル長が長くなれば、その割合は小さくなる。

ベクトル演算用バージョンのプログラムにおいても、各サブルーチンについて単に AP-FORTRAN に変換するだけでなく、ベクトル演算可能な部分の割合をふやす工夫や最適化を行うことによって、APU 時間の短縮がなされた。

Table 4.13 に示されている中 3 列のベクトル演算用プログラム (Partially vectorized program) の計算時間の減少は、Inner Iteration に含まれるサブルーチンのベクトル演算化と、タスク・スイッチ回数の削減によるものである。

再構成プログラムの APU 時間を 100 とすると、ベクトル演算用プログラムの APU 時間は、68.6% に短縮されたことになる。

Table 4.13 Computing time of problem no.1
(vector length:9) by FACOM230-75APU

Source program, Compiler	
F FORTRAN-H, FORTRANH Compiler
FA FORTRAN-H, APFORT Compiler
A AP-FORTRAN, APFORT Compiler

Program Subroutine name \n	Original	Restructure	Partially Vectorized	Partially Vectorized	Partially Vectorized	Vectorized	Vectorized
INR2	F	FA	A	A	A	A	A
LOU2	F	FA	A	A	A	A	A
RELX	F	FA	FA	FA	FA	FA	FA
RELXA	-	FA	A	A	A	A	A
LEK2	F	FA	FA	A	A	A	A
RDUE	F	FA	FA	FA	FA	A	A
OUTR	F	F	F	F	F	F	F
SOU2	F	F	A	F	F	F	F
CHB2	F	F	A	A	F	F	F
FOU2	F	F	A	A	A	F	F
Others	F	F	F	F	F	F	F
全CPU時間(秒)	21.141	5.819	4.828	5.415	5.429	5.835	5.835
全APU時間(秒)	-	23.899	19.339	18.129	18.037	17.104	16.397
合計 時間(秒)	21.141	29.718	24.167	23.544	23.466	22.939	22.232
タスク・スイッチ回数	-	214	382	280	265	214	214

4.3.3 ベクトル化による全体的効率

ベクトル化の効果によるプログラム全体の総合性能比(P)を決める要因は、ベクトル化可能率(V)と並列計算平均性能比(α)である。

ベクトル化可能率 V は、オリジナルプログラムの全計算時間の中でベクトル化可能部分の計算時間の占める割合である。値は、全計算時間を1として求めたものである。

並列計算平均性能比 α は、ベクトル化された部分の総体的な平均の性能比で、逐次計算に対する並列計算の速度の比である。詳しくは、プログラム中でベクトル化可能なループについて、DOループによって逐次計算した場合とベクトル命令によって並列計算した場合の計算時間の比である。

この α の値は、ベクトル長や各ベクトル命令の処理速度、およびそれら命令(演算)の組合せなどに依存する。

ベクトル長は、解くべき問題、入力データ、数値解法などによって決まる。

各ベクトル命令の処理速度は、並列計算機のハードウェアおよびアーキテクチャによって決まる。また各並列計算機によって用意しているベクトル命令の種類や数も異なる。

これらのVと α の値が求まるとPの値が次の式によって求められる。

$$P = \frac{1}{\frac{V}{\alpha} + (1 - V)} \quad (4.1)$$

この式によって、オリジナルプログラムの計算時間に比べて何倍速くなるかがわかる（Fig. 4.5 参照）。

Fig. 4.5 からもわかるように、総合性能比Pを大きくするためにはベクトル化可能率Vを大きくすることが重要である。ベクトル長を長くしたり、並列計算機のハードウェア性能を上げることもさることながら、プログラムに潜在する並列計算可能な部分の割合を増すことがポイントになる。ただし、F75 APU/CPUの場合には、かりにベクトル化可能率が大きくなるとしても、タスクスイッチの回数が極端に増加する場合は考慮を要する。

また、

$$Q = \left\{ V \times \frac{1}{\alpha} + (1 - V) \right\} \times 100 \quad (4.2)$$

の式によって、計算時間が何パーセントに短縮されたかがわかる。

ただし、実際には、 α の値を求めるのは簡単でなく、公称されている程の性能も、現実の原子力コード等では得られにくい。従って、全体の計算時間等からPまたはQの値は求められるので、逆にこの式から並列計算平均性能比 α を求めることができる。

F230-75 の計算時間からベクトル長9のサンプル問題のときのQを求めると、APU時間だけで比較すると、68.6%であり、CPU時間を加えた計算時間で比較すると、74.8%である。ベクトル化可能率Vは、Inner Iterationに含まれるサブルーチンだけベクトル化したとき、0.627(62.7%)である。

これを(4.2)式に代入して α を求めると、APU時間では2.00、CPU時間を加えた計算時間では、1.67である。

すなわち、VENTUREにおいてベクトル長が9、間接アドレスによる配列参照（リストベクトル）などを使用しているときの並列計算平均性能は約2倍であるといえる（Table 4.14 参照）。

プログラムの中で核となるいくつかの重要なDOループについて、基本的な性能比を測定した松浦氏の資料によると、F75 APUにおいてベクトル化効果の現われるのはベクトル長50ぐらいからで、並列計算平均性能は約5倍になり、ベクトル長が200以上になると約10倍になる。ただし間接アドレスによる配列参照（リストベクトル）がある場合には約半分の性能になる。また加算や乗算の混じた混合演算の配列計算の場合には、ベクトル長が30ぐらい以下

のときは、リストベクトルを使用した方がインデックス変数を使用するより速いが、ベクトル長が長くなるとインデックス変数を使用する方が速い。

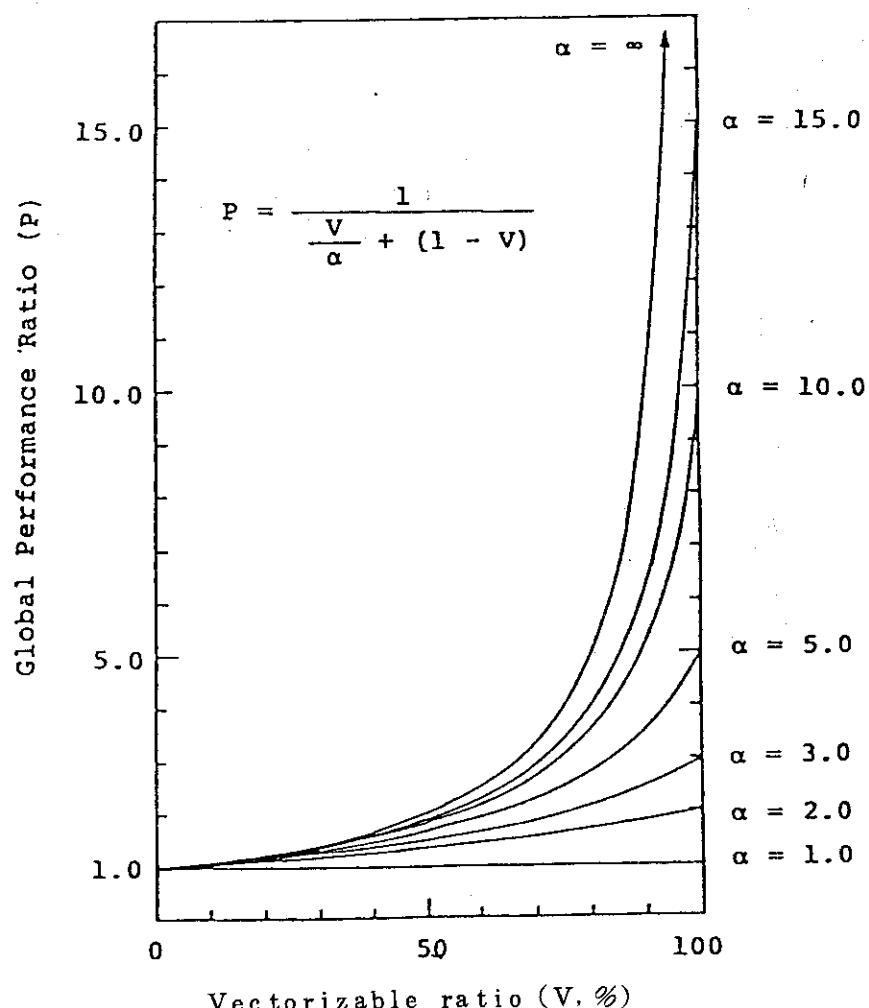


Fig. 4.5 Global performance ratio

Table 4.14 Vectorizable ratio of diffusion code VENTURE

Subroutine name	Time cost by FORTUNE(%)	Relatively vectorizable ratio (%)	Vectorizable part time cost (%)
I NR 2	3.8	19.6	0.7
LOU 2	24.0	91.5	22.0
RELX	18.4	0.0	0.0
RELXA	39.3	95.3	37.5
RDUE	0.4	71.7	0.3
LEK 2	2.8	77.5	2.2
Subtotal	88.7		62.7
Others	11.3		
Total	100.0		

(Sample problem: 3-D, X-Y-Z, 2-Group, Vector length=9)

FACOM 230-75 APU/CPU
 (Vectorizable part time cost) (Performance ratio) (Scalar part time cost) (%)

$$\begin{array}{cccccc}
 62.7 & \times & \frac{1}{1.67} & + & 37.3 & = 74.8 \\
 62.7 & \times & \frac{1}{2.0} & + & 37.3 & = 68.7
 \end{array}$$

5. おわりに

現在電子計算機の計算速度を早める有力な手法の一つは、レジスターをベクトル化することにより、一つの命令で多数の演算をほとんど同時に行なうことを可能にすることであろう。CRAY-1やF75APUはこのベクトルレジスターを持つ実用に用いられている計算機の一つである。CRAY-1での実験では、逆行列を求める等の簡単な線型計算では、演算を一つずつ実行するスカラー・レジスターを用いる場合に比べて、ベクトルレジスターを有効に用いれば、計算時間が数十分の1(12)に短縮できることが知られている。しかし、実際の計算で用いられるアプリケーションプログラムでの、このベクトルレジスターの積極的な利用は、まだ始まったばかりであると云える。このためには、解くべき問題（方程式等）へ加えられる近似法、数値解法、数値解法のプログラム化、さらには、プログラムを表現する言語について、新しい観点からの研究が必要である。また一方においては、既在のプログラムを書きかえることにより、計算時間を短縮をはかり、ベクトルレジスターの利用法について経験を深める必要がある。

ここで報告した研究は後者に属し、現在もっとも進んだ拡散コードの一つであるVENTUREをCRAY-1とF75APUへ変換し、そのベクトル化を行なったものである。ここでは、VENTUREの持つ計算法、アルゴリズムを基本的に変更することなく、FORTRAN上の表現の変更及び繰返し計算での変数の意味を変えて、ベクトル化を行なった。従って、新しいVENTUREは、原VENTUREの有していた特長を全て持っている。

ベクトル化の効果はかなり大きく、CRAY-1では計算時間を全体で約1/2に短縮できた。また個々のサブルーチンで見ると、ベクトル化により計算時間が1/10程度になったものもあり、上手に用いられた時は、ベクトル化の効果が大きいことがわかった。一方では、ベクトル化のための準備計算に時間がとられ、計算時間がやや長くなったものも存在している。この様なサブルーチンを無くすためには、最初からベクトルレジスターを意識して、プログラム設計を行なう必要があり、現在のVENTUREを変更するのでは困難である。

今後できれば、ベクトルレジスターの機能を含む、並列計算機全体のパフォーマンスを考えた、大型原子力コードへの適用性を調査し、現在の計算機での極限のスピードを評価・把握しておく必要があるものと考えられる。

謝 辞

この報告の一部は東海研究所原子力コード総合化専門部会の昭和55年度作業のひとつとしておこなわれたものです。東海研熱中性子炉体系標準コード・システムとして開発された核計算コード・システムSRACの一つのモジュールとして、VENTUREが使われる予定となっていました。これはSRACコード・システム中で最も計算時間のかかるモジュールで、ベクトル計算処理によって、その計算時間の短縮がどの程度達成できるかを調査するために行なわれた作業です。

5. おわりに

現在電子計算機の計算速度を早める有力な手法の一つは、レジスターをベクトル化することにより、一つの命令で多数の演算をほとんど同時に行なうことを可能にすることであろう。CRAY-1やF75APUはこのベクトルレジスターを持つ実用に用いられている計算機の一つである。CRAY-1での実験では、逆行列を求める等の簡単な線型計算では、演算を一つずつ実行するスカラー・レジスターを用いる場合に比べて、ベクトルレジスターを有効に用いれば、計算時間が数十分の1に短縮できることが知られている⁽¹²⁾。しかし、実際の計算で用いられるアプリケーションプログラムでの、このベクトルレジスターの積極的な利用は、まだ始まったばかりであると云える。このためには、解くべき問題（方程式等）へ加えられる近似法、数値解法、数値解法のプログラム化、さらには、プログラムを表現する言語について、新しい観点からの研究が必要である。また一方においては、既在のプログラムを書きかえることにより、計算時間を短縮をはかり、ベクトルレジスターの利用法について経験を深める必要がある。

ここで報告した研究は後者に属し、現在もっとも進んだ拡散コードの一つであるVENTUREをCRAY-1とF75APUへ変換し、そのベクトル化を行なったものである。ここでは、VENTUREの持つ計算法、アルゴリズムを基本的に変更することなく、FORTRAN上の表現の変更及び繰返し計算での変数の意味を変えて、ベクトル化を行なった。従って、新しいVENTUREは、原VENTUREの有していた特長を全て持っている。

ベクトル化の効果はかなり大きく、CRAY-1では計算時間を全体で約1/2に短縮できた。また個々のサブルーチンで見ると、ベクトル化により計算時間が1/10程度になったものもあり、上手に用いられた時は、ベクトル化の効果が大きいことがわかった。一方では、ベクトル化のための準備計算に時間がとられ、計算時間がやや長くなったものも存在している。この様なサブルーチンを無くすためには、最初からベクトルレジスターを意識して、プログラム設計を行なう必要があり、現在のVENTUREを変更するのでは困難である。

今後できれば、ベクトルレジスターの機能を含む、並列計算機全体のパフォーマンスを考えた、大型原子力コードへの適用性を調査し、現在の計算機での極限のスピードを評価・把握しておく必要があるものと考えられる。

謝 辞

この報告の一部は東海研究所原子力コード総合化専門部会の昭和55年度作業のひとつとしておこなわれたものです。東海研熱中性子炉体系標準コード・システムとして開発された核計算コード・システムSRACの一つのモジュールとして、VENTUREが使われる予定となっていました。これはSRACコード・システム中で最も計算時間のかかるモジュールで、ベクトル計算処理によって、その計算時間の短縮がどの程度達成できるかを調査するために行なわれた作業です。

援助していただきました総合化専門部会長 桂木 学 安全工学部長はじめとする部会専門委員の方々に感謝致します。

サンプル入力データの作成に当たり、御協力と助言をいただきました動力炉開発・安全性研究管理部、高速炉設計研究室の飯島 進氏に感謝致します。

また、FACOM 230-75 APUの使用ならびにベクトル計算処理について適切なる助言をいただきました外来研究員（富士通（株））の松浦俊彦氏に感謝致します。

本作業の全般にわたり御指導いただきました、計算センターの室長代理浅井 清氏、石黒美佐子氏に感謝致します。

References

- (1) D.R. Vondy, T.B. Fowlert, et al.: VENTURE : A Code Block for Solving Multigroup Neutronics Problems Applying the Finite-Difference Diffusion-Theory Approximation to Neutron Transport, ORNL-5062, (1975)
- (2) T.B. Fowler, D.R. Vondy, et al.: Nuclear Reactor Core Analysis Code: CITATION, ORNL-TM-2496, (1969, Revision 2: 1971)
- (3) Scientific Computer Information Exchange Meeting, Thema: Impact of Advanced Systems on Scientific Computations, Proceedings, CONF-790902, (1979)
- (4) C.D. Swanson, L.K. Kin, et al.: Experience with DoT IV on a Vector Processing Computer, ANS Transactions, 33, 315, (1979)
- (5) R.W. Hardie, W.W. Little, Jr.: 3DB, A Three-dimensional Diffusion Theory Burnup Code, BNWL-1264, (1970)
- (6) CRAY-1 Computer Systems: Library Reference Manual, SR-0014, CRAY Research, Inc. (1980)
- (7) Don Heller: Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems, SIAM J. Numer. Anal., Vol.13, No.4, (1976)
- (8) 三輪 修, 他 : FACOM 230-75 アレイプロセサシステム, FUJITSU, Vol.29, No.1, (1978)
- (9) AP-FORTRAN 文法書, 75SP-0370-1, 昭和 52 年 5 月, 富士通(株)
- (10) AP-FORTRAN 使用手引書, 75SP-0500-1, 昭和 53 年 6 月, 富士通(株)
- (11) Toshihiko Matsuura: Vector Processing Efficiency of Plasma MHD Codes Using FACOM 230-75 APU, FUJITSU Scientific and Technical Journal, Vol.17, No.3, (1981)
- (12) CRAY-1 セミナー・テキスト, センチュリーリサーチ センタ(株), (1980)
- (13) CRAY-1 Computer Systems: Fortran (CFT) Reference Manual, 2240009, CRAY Research, Inc. (1980)

付録：計算機システムとベクトル演算機能

A.1 CRAY-1 のハードウェア

ここではVENTUREのベンチマークテストに用いたCRAY-1計算機システムのベクトル演算機能について説明する。CRAY-1はいわゆるスーパーコンピューターであり、その大きな特徴はベクトル演算機能をもつことである。ベクトル演算機能はベクトルレジスタとバイブライイン化された演算ユニットからなるハードウェアと、そのハードウェアの能力を最大限引き出す様に開発されたソフトウェアからなる。以下にベクトル演算におけるCRAY-1のハードウェア、ソフトウェアの機能について記す。

CRAY-1の外観を Photo. 1 に示す。

CRAY-1のベクトル演算機能をハードウェアの面からみると以下の様に分類できる。

(1) クロック周期

CPUのクロック周期は12.5ナノ秒である。以下これを1 CP という。

(2) ベクトルレジスタ

各々64語(1語は各64ビット)を同時に格納できる8個のベクトルレジスタをもつ。各要素(語)のストア、ロードは1 CP毎に連続して行なえる。

(3) バイブライイン化された演算ユニット

加算器、乗算器等の演算ユニットはそれぞれ独立に動作でき、且つ1 CP毎に完全にセグメント化(バイブルайн化)されている。例えば浮動小数点加算ユニットでは2つのレジスタの内容を加えてその結果をレジスタに戻すのに6 CPを要する。このユニットは6つの操作段階に分割されていて、各段階の操作結果が次の段階へ1 CP毎に送られるので、このユニットへは1 CP毎にオペランドの対を送り込み、更に演算結果を1 CP毎にとり出す事ができる。

(4) チェイニング

ベクトル処理の連鎖化をチェイニングという。つまりある演算ユニットからの結果は1 CP毎に1要素がベクトルレジスタに格納されるが、全要素がレジスタに入るのを待たず最初の1要素の結果が得られたら直ちにそれを次の演算ユニットに供給し、演算の平行処理を行なう事である。

これらの機能を次のDOループを例にとり図示すると、Fig.A.1の如く表わされる。

```
DIMENSION A(64), B(64), C(64), D(64), E(64)
```

```
DO 10 I = 1, 64
```

```
    C(I) = A(I) + B(I)
```

```
    E(I) = C(I) * D(I)
```

```
10 CONTINUE
```

ここで配列 A, B, D は予め値がセットされているものとする。また Fig. A. 1 ではメモリーからベクトルレジスタへのオペランドのロード、およびベクトルレジスタからメモリへのストアは省略してある。

Fig. A. 1 において A(1), B(1) が V₁, V₂ レジスタを出発し E(1) が V₅ IC 到着するまでの時間は 17 CP であり、E(2) ~ E(64) は 1 CP 每 IC V₅ IC 到着する。従って全部の要素を計算するのに必要な時間は次式で得られる。

$$17 \text{ CP} + 1 \text{ CP} \times 63 = 80 \text{ CP} = 1 \text{ ミリ秒}$$

ここでもし演算ユニットがバイブライン化されておらず、かつチェイニングも行なわなければ同様の計算に要する時間は次式の様になる（スカラー演算）。

$$8 \text{ CP} \times 64 + 9 \text{ CP} \times 64 = 1088 \text{ CP} = 13.6 \text{ ミリ秒}$$

加算 乗算

以上よりベクトル化による効果はここに掲げた例では 13.6 倍、つまり CPU 時間で約 93% 減少する事になる。この効果はチェイニングされる演算が多くなればなるほど、ベクトル長が長くなるほど上る事が予想される。しかし一般的なプログラムで考えると例えば加算ユニットを使おうとした時これを他のオペランドの演算でまだ使っているという事は頻繁に起る訳で、この場合はチェイニングされない。ベクトル長についてはベクトルレジスタが 64 語であるので、それ以上の演算は 64 語毎のかたまりとして処理される。従って上の例でベクトル長を 128 語とし単純に計算するとベクトル演算では 2 ミリ秒、スカラー演算では 27.2 ミリ秒となる。

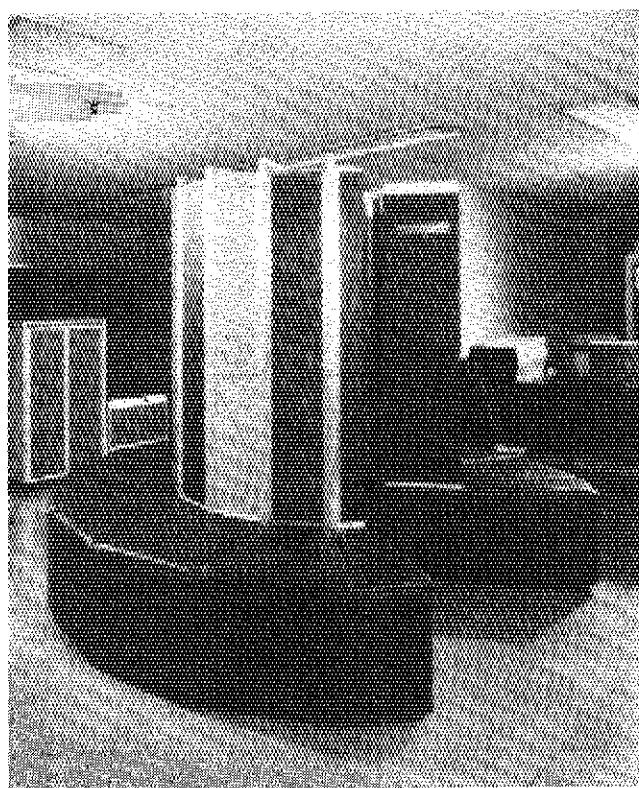
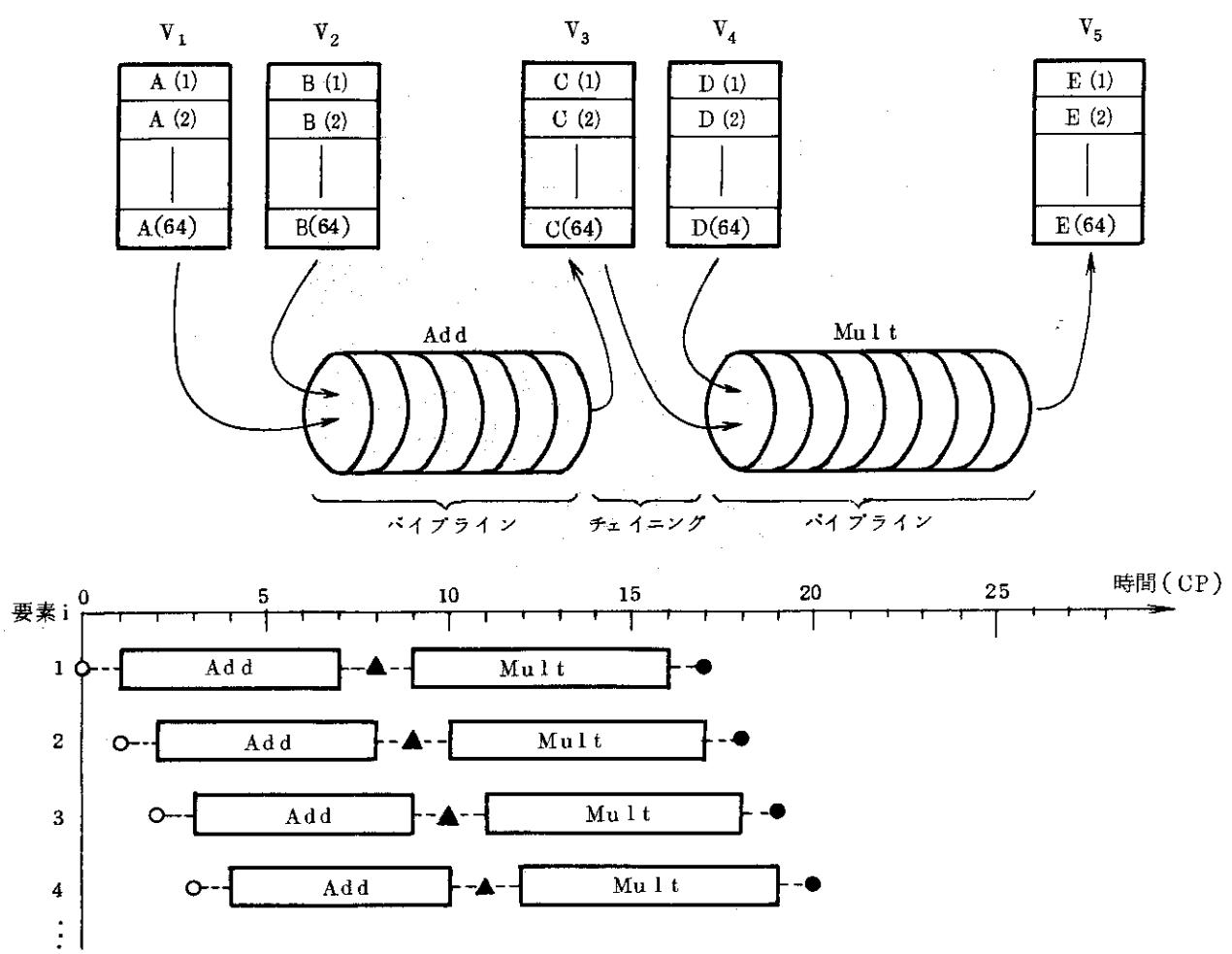


Photo. 1 CRAY-1 system



ここで Add : 加算ユニット (6セグメントに分けられている)
 Mult : 乗算ユニット (7セグメントに分けられている) } データが1セグメント通過するのに1CP
 要する。

CP : Clock Period (1 CP = 12.5 nsec)

V_n : ベクトル・レジスタ

○ : V_1, V_2 より A_i, B_i の要素が出発する時間

▲ : C_i が V_3 に到着する時間

● : E_i が V_5 に到着する時間

Fig.A.1 Data flow on vector operation

A.2 CRAY-1のソフトウェア

CRAY-1のベクトル演算のために用意されたハードウェアを利用してできる言語としては特別なものではなく、現在 **Assembler (CAL)** と **Fortran (CFT)** がその機能をもつだけである。ここでは一般的な **Fortran** 言語（以下 CFT という）によるベクトル化の説明のみに限定する。

まずベクトル化とはベクトル演算が行える様にコンパイラがオブジェクトコードを作り出す事であり、ベクトル演算とは配列に対して同一種類の演算を行なう場合、その配列のある要素についての演算が進行中に同じ配列の他の要素に対する演算が同時に進行するような一連の演算をいう。一方 CFT によるベクトル化についてはユーザー（プログラマー）がそれを明確に指示する必要はない。つまり CFT はある一定の条件を満たすステートメントがあればそれを自動的にベクトル化するものである。

A.2.1 CFT の自動ベクトル化条件

CFT は次に示す 4 つの条件を全て満たすステートメントに対してベクトル化を行なう。

条件(1) 最も内側の DO ループで、かつ配列要素の値を計算するものである事。

配列の演算を同時に進行させる事がベクトル演算であるので、まず DO ループにおける配列要素の計算でなければならない。但し配列の和、積を求める単純な DO ループに対してはこれをベクトル化する。多重 DO ループの場合はその最も内側の DO ループのみに着目する。**IF 文** による反復計算はベクトル化の対象としない。

条件(2) DO ループの中にある配列要素名および変数名の全てが次のいずれかであること。

- i. 不変変数名（不变配列要素名）
 - ii. 等差整数名でかつその定義式に表われる他は、配列の要素を示す目的にのみ用いられる。
 - iii. ベクトル配列要素名
 - iv. スカラーワン時変数名
 - v. 低減配列演算に使われる変数または不变配列要素名
- これらについての詳細な説明は参考文献〔13〕を参照のこと。

条件(3) DO ループの中に次に示すステートメントがない事。

- i. IF 文
- ii. GOTO 文
- iii. CALL 文
- iv. 外部関数を参照する文
- v. スカラー演算による Fortran 組込関数を参照する文
- vi. 入出力文

条件(4) ベクトル化した場合、配列とその添字に以下に示す関係が生じない事。

i. 未完型従属関係

ii. 破壊型従属関係

これらの従属関係については次項で説明する。

A.2.2 従属関係

まず従属関係の説明をする前にスカラー演算とベクトル演算のステートメントの実行順序を説明する。例えば次の様な DO ループがあるとする。

```
DO 10 I = 1, 10
    A(I) = B(I) + C(I)
    D(I) = E(I) * F(I)
10 CONTINUE
```

これをスカラー演算とベクトル演算について各ステートメントの実行順序を示すと以下の様になる。

スカラー演算	ベクトル演算
$A(1) = B(1) + C(1)$	$A(1) = B(1) + C(1)$
$D(1) = E(1) * F(1)$	$A(2) = B(2) + C(2)$
$A(2) = B(2) + C(2)$	\vdots
$D(2) = E(2) * F(2)$	$A(10) = B(10) + C(10)$
\vdots	$D(1) = E(1) * F(1)$
$A(10) = B(10) + C(10)$	$D(2) = E(2) * F(2)$
$D(10) = E(10) * F(10)$	\vdots
	$D(10) = E(10) * F(10)$

ここでスカラー演算については説明する必要はないが、ベクトル演算における実行順序について簡単に触れる。

- (1) $B(1) \sim B(10)$ の値をメモリーからベクトルレジスタに入れる。
- (2) $C(1) \sim C(10)$ の値をメモリーから別のベクトルレジスタに入れる。
- (3) 2 個のベクトルレジスタのそれぞれの要素の和を計算し、結果を他のベクトルレジスタに入れる。
- (4) 得られた和を $A(1) \sim A(10)$ のメモリに記憶する。
- (5) $D(I) = E(I) * F(I)$ も同様の手順で実行される。

従って配列とその添字の関係によってはスカラー演算とベクトル演算とでは答が異なってしまう場合が生じる。そこで次の様な例をあげる。

```
DO 10 I = 1, 9
    A(I) = C(I)
    B(I) = A(I+1) + 1.0
```

10 CONTINUE

ス カ ラ 演 算	ベ ク ツ ル 演 算
① $A(1) = C(1)$	① $A(1) = C(1)$
② $B(1) = A(2) + 1.0$	② $A(2) = C(2)$
③ $A(2) = C(2)$	⋮
④ $B(2) = A(3) + 1.0$	③ $A(9) = C(9)$
⋮	④ $B(1) = A(2) + 1.0$
⑤ $A(9) = C(9)$	⑤ $B(2) = A(3) + 1.0$
⑥ $B(9) = A(10) + 1.0$	⋮
	⑥ $B(9) = A(10) + 1.0$

これらに簡単な数値をあてはめるとわかるが、スカラー演算では②の右辺の $A(2)$ は③の左辺で定義される前に参照される。しかしベクトル演算では④の右辺で $A(2)$ を参照する前に②で壊されてしまっている。勿論プログラムはスカラー演算での実行順序で得られる答を期待しているのでCFTはこの様にベクトル化する事によって正しい答が得られないDOループはスカラー演算で実行する様にコンパイルする。この例を破壊型従属関係という。

またこれとは逆に参照すべき配列の要素がまだ定義されていない未完型依属関係があり、これもCFTはベクトル化しない。しかし破壊型従属関係の全てと未完型従属関係の殆んどはステートメントを入れ替えたり、また一時的配列を用いる事により従属関係が取り除かれベクトル化が可能である。先の破壊型従属関係の生じているDOループは中の2つのステートメントを入れ替える事によりベクトル化できる。

さてどの様にしてもベクトル化できない未完型従属関係の例をあげる。それは Recurrence を引き起こしている場合である。次のDOループにおいてベクトル化した際の実行順序を考える。

```
DO 10 I = 2, 10
    A(I) = A(I-1) + B(I)
```

10 CONTINUE

スカラー演算では左辺で定義された $A(I)$ は I が1増加したあとすぐ右辺の $A(I-1)$ で参照される。ところがベクトル演算ではまず最初に $A(1) \sim A(9)$ をメモリーからベクトルレジスタに全て移してしまう。従って右辺の $A(I-1)$ は前の演算での結果が残っているまま参照することになる。この様な未完型従属関係はベクトル化する事は不可能である。3.4節で述べたサブルーチン RELX の中の DO ループが正にこの関係であり、そこに記した様に解法を変更するなどして Recurrence を取除く外はベクトル化の方法はない。

B.1 FACOM 230-75 APU/CPU計算機システム

F 75 APU/CPU計算機システムの詳細については、文献[8]に述べられている。ここでは、F 75 APUの外観、計算機構成と基本性能について、Photo. 2, Fig.B.1 と Table B.1(a), B.1 (b)で示す。

B.2 AP-FORTRAN

FORTRAN から AP-FORTRANへの代表的な変換例を Fig.B.2 に示す。

また、VENTUREコードのベクトル演算用プログラムで使用した主な配列特殊関数の例を Table B.2 に示す。

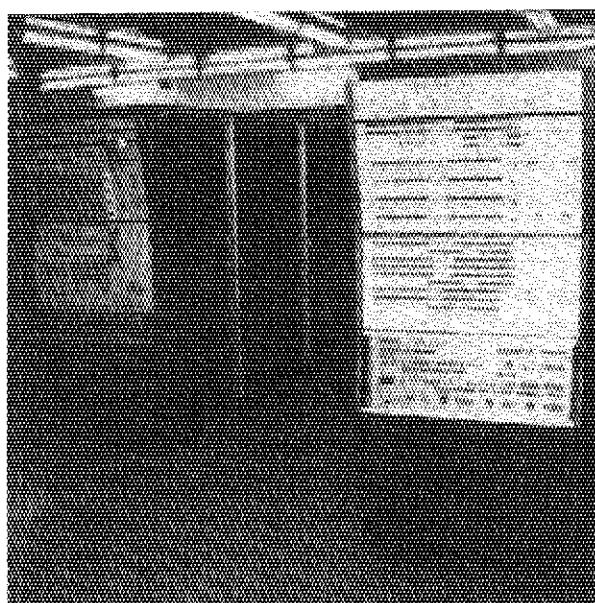


Photo. 2 FACOM 230 - 75 CPU/APU system

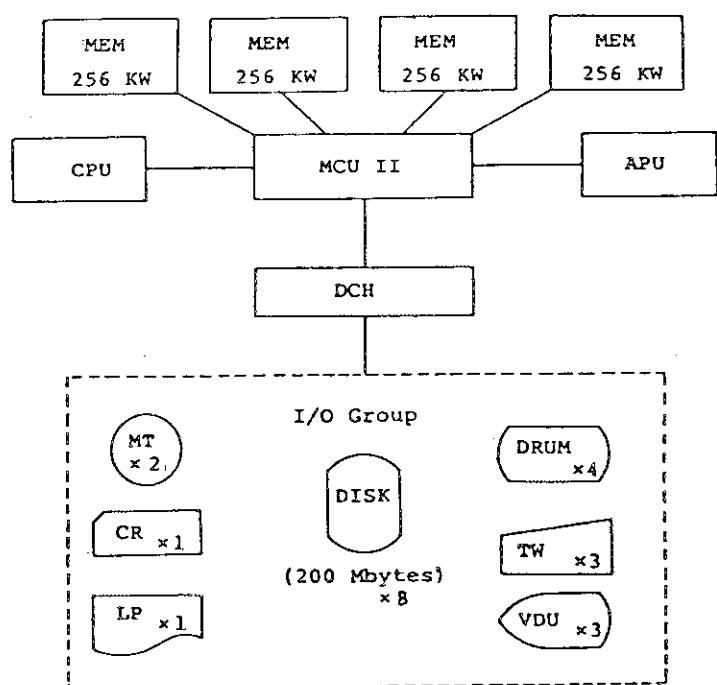


Fig. B.1 System configuration of FACOM 230-75 APU/CPU

Table B.1(a) Characteristics and performances of FACOM 230-75 APU

Items		Specifications
Machine cycle (Basic clock for pipeline)		90 ns
Elements	High-speed CML (Current Mode Logic)	3 ns
	NLT (Non Threshold Logic)	1 ns
	IC memory	14 ns or 35 ns
Buffers	Instruction buffer	8-word
	Data buffer	256-kiloword × 3
	Buffer memory	2-kiloword
Registers	Data register	256
	Vector register	1792-word
Operation pipeline		Addition pipeline Multiplication pipeline Logical operation pipeline
Vector operation time (Floating point, single precision)		Addition 22 MFLOPS Multiplication 11 MFLOPS Division 1.2 MFLOPS Inner product 22 MFLOPS Summation 22 MFLOPS

Table B.1(b) Characteristics and performances of FACOM 230-75 CPU

Items		Specifications	
CPU cycle time		90 ns	
Buffer memory		2- or 4-kiloword 90 ns/2-word	
Core memory		64- (min)/1024-kiloword (max) 1 μs/2-word	
Bit pattern		40-bit (36 data bits + 4 flag bits)	
Operation time		Fixed point	Floating point
		108 ns	360 ns
		450 ns	540 ns
Gibson mix		2250 ns	
Operating system		1350 ns	
Program language		FACOM MONITOR VII	
Compiler		FACOM FORTRAN-H	
		FACOM FORTRAN IV	

<u>FORTRAN (Scalar)</u>	<u>AP-FORTRAN (Vector)</u>
<pre>DO 100 I=1,NLONG A(I)=A(I)-AL0*B(I) 100 CONTINUE</pre>	<pre>INDEX IX/1,NLONG/ A(IX)=A(IX)-AL0*B(IX)</pre>
<pre>DO 20 IN=1,INE DO 10 IR=1,IRE A(IR,IN)=B(IR,IN)+C(IR,IN) 10 CONTINUE 20 CONTINUE</pre>	<pre>INDEX IRX/1,IRE/ DO 20 IN=1,INE A(IRX,IN)=B(IRX,IN)+C(IRX,IN) 20 CONTINUE</pre>
<pre>TEMP=0.0 DO 10 IR=1,IRE TEMP=TEMP+A(IR) 10 CONTINUE TOTAL=TEMP</pre>	<pre>INDEX IRX/1,IRE/ TOTAL=VSUM(A(IRX))</pre>

Fig. B.2 Example of transform from FORTRAN to AP-FORTRAN

Table B.2 Examples of Array special functions in AP-FORTRAN

Instruction	Calling procedure	Definition
Vector element summation	$S = VSUM(B(*)$)	$S = \sum_{i=1}^n b_i$
Inner product	$S = IPD(B(*), C(*)$)	$S = \sum_{i=1}^n b_i \times c_i$
Find index of max. element	$I = IFMX(B(*)$)	$I = \min\{i b_i = \max_j(b_j)\}$
Find maximum value	$S = FMXV(B(*)$)	$S = \max_j(b_j)$
Gather	$A(*) = GAT(M(*), B(*)$)	$a_i = b_j \text{ for } m_j; \text{mask on}$
Scatter	$A(*) = SCAT(M(*), B(*), C(*)$)	$a_i = \begin{cases} c_j & \text{for } m_i; \text{mask on} \\ b_i & \text{for } m_i; \text{mask off} \end{cases}$
Mask	$A(*) = MASK(M(*), B(*), C(*)$)	$a_i = \begin{cases} b_i & \text{for } m_i; \text{mask on} \\ c_i & \text{for } m_i; \text{mask off} \end{cases}$