

JAERI - M  
92-105

新FORTRANコンパイラの導入と  
ベクトル計算機の効果的利用法

1992年7月

根本 俊行\*・鈴木孝一郎\*・渡辺 健二\*・町田 昌彦\*  
長内 誠志\*・磯辺 信雄\*・原田 裕夫・横川三津夫

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

©Japan Atomic Energy Research Institute, 1992

編集兼発行 日本原子力研究所  
印 刷 いばらき印刷株

## 新FORTRANコンパイラの導入とベクトル計算機の効果的利用法

日本原子力研究所東海研究所情報システムセンター

根本 俊行\*・鈴木 孝一郎\*・渡辺 健二\*・町田 昌彦\*  
長内 誠志\*・磯辺 信雄\*・原田 裕夫・横川 三津夫

(1992年6月17日受理)

原研に設置されている富士通(株)製大型計算機上で動作するFORTRAN コンパイラが平成4年5月から新バージョン(V12)に移行された。V12 コンパイラの導入に先立って、代表的原子力コード16本のベンチマークテストを行い、V12 コンパイラの性能を評価した。この結果、新コンパイラでは、旧コンパイラと比較して平均で1.13倍の速度向上が得られた。

また、V12 コンパイラで新たに追加された機能の効果、及び、原子力コードに対するコンパイラの互換性等を調査した。さらに、新たな動的解析ツールANALYZERに対応したベクトル化支援ツールTOP10EXを開発した。

本報告書では、新コンパイラの評価結果やTOP10EXの使用方法について述べる。

Installation of a New Fortran Compiler and Effective  
Programming Method on the Vector Supercomputer

Toshiyuki NEMOTO<sup>\*</sup>, Koichiro SUZUKI<sup>\*</sup>, Kenji WATANABE<sup>\*</sup>  
Masahiko MACHIDA<sup>\*</sup>, Seiji OSANAI<sup>\*</sup>, Nobuo ISOBE<sup>\*</sup>  
Hiroo HARADA and Mitsuo YOKOKAWA

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

(Received June 17, 1992)

The Fortran compiler, version 10 has been replaced with the new one, version 12 (V12) on the Fujitsu Computer system at JAERI since May, 1992. The benchmark test for the performance of the V12 compiler is carried out with 16 representative nuclear codes in advance of the installation of the compiler. The performance of the compiler is achieved by the factor of 1.13 in average. The effect of the enhanced functions of the compiler and the compatibility to the nuclear codes are also examined. The assistant tool for vectorization TOP10EX is developed.

In this report, the results of the evaluation of the V12 compiler and the usage of the tools for vectorization are presented.

Keywords: FORTRAN77EX, Compiler, VP2600, Vector

---

\* On leave from FUJITSU Ltd.

## 目 次

はじめに .....	1
第1編 FORTRAN77EXへの移行 .....	3
1. FORTRAN77EXへの移行 .....	5
2. FORTRAN77EXの強化項目 .....	6
3. FORTRAN77EXへの移行手順 .....	7
3.1 移行手順 .....	7
3.2 FORTRAN77EXにおける処理手続き .....	9
3.3 TSSにおけるFORTRAN77EXの使用 .....	14
4. オプションの指定 .....	15
4.1 最適化に関するオプション .....	15
4.2 ソースプログラムに関するオプション .....	20
4.3 翻訳時の印刷情報に関するオプション .....	20
4.4 VP翻訳に関するオプション .....	21
4.5 インラインオプション .....	24
4.6 その他の翻訳オプション .....	26
4.7 実行可能プログラムオプション .....	27
5. FORTRAN77EXシステムの活用 .....	28
5.1 アドレス拡張機能の活用 .....	28
5.2 VIO/F入出力機能 .....	36
6. 移行時の問題点 .....	43
6.1 非互換項目による問題 .....	43
6.2 ロードモジュールの増大による問題 .....	45
第2編 VPプログラミング .....	47
1. VP2600/10の概要 .....	49
1.1 ベクトル処理 .....	49
1.2 VP2600/10の構成と特徴 .....	50
1.3 ベクトル処理装置での命令実行について .....	51
1.4 パイプライン方式について .....	51
1.5 高速性を達成するための技術 .....	58
2. ベクトル化の基本 .....	62
2.1 パイプライン演算機の性能 .....	62
2.2 ベクトル化による処理効率向上比 .....	65

2.3 ベクトル化例 .....	68
2.4 ベクトル化とは .....	69
3. 基本的なベクトル化技法 .....	77
3.1 ベクトル化の手順 .....	77
3.2 ベクトル化の促進 .....	79
4. アルゴリズムの変更を伴うベクトル化 .....	104
4.1 ポイントSOR法をベクトル化するには .....	104
4.2 多重積分(ガウス・チェビシェフ型積分)のベクトル化 .....	110
4.3 (複数個の)3重対角方程式のベクトル化 .....	114
4.4 楕円形方程式のベクトル化 .....	118
4.5 モンテカルロ法のベクトル化 .....	124
第3編 FORTRAN77EXにおけるベクトル化支援ツール .....	133
1. FORTRAN77EXにおけるベクトル化支援ツール .....	135
2. サンプリング解析 .....	137
2.1 サンプリング解析の手続き .....	137
2.2 サンプリング解析の解析情報 .....	138
2.3 サンプリング解析のオプション .....	140
2.4 サンプリング解析情報の評価 .....	140
2.5 サンプリング解析の使用 .....	141
3. 実行詳細解析 .....	142
3.1 実行詳細解析の手続き .....	142
3.2 実行詳細解析の解析情報 .....	143
3.3 実行詳細解析のオプション .....	147
3.4 実行詳細解析の使用 .....	148
3.5 実行詳細解析の使用上の注意 .....	150
4. TOP10EX .....	151
4.1 TOP10EXの機能 .....	151
4.2 TOP10EXの手続き .....	154
4.3 TOP10EXのオプション .....	155
4.4 TOP10EXの使用 .....	162
4.5 TOP10EXの使用上の注意 .....	163
おわりに .....	164
謝 辞 .....	164
参考文献 .....	165
付録A 非互換項目資料 .....	167
付録B カタログド・プロジェクト一覧 .....	189
付録C MERGER使用手引き .....	201

## Contents

Introduction .....	1
The part 1 FORTRAN77EX Compiler .....	3
1. Introduction to FORTRAN77EX Compiler .....	5
2. Enhanced Items of FORTRAN77EX Compiler .....	6
3. How to Use FORTRAN77EX Compiler .....	7
3.1 Execution Procedure .....	7
3.2 Standard JCLs .....	9
3.3 Compiling Procedure on TSS .....	14
4. Compiler Options .....	15
4.1 Options for Optimizations .....	15
4.2 Options for Source Program .....	20
4.3 Options for Compiler Messages .....	20
4.4 Options for VP Oriented Object .....	21
4.5 Options for Inline Expansion of Subroutine .....	24
4.6 Others .....	26
4.7 Options on Execution .....	27
5. Address Extended Function and VIO/F Function .....	28
5.1 Address Extended Function .....	28
5.2 VIO/F Function .....	36
6. Problems by Changing the Compiler .....	43
6.1 Incompatibility Items .....	43
6.2 Enlargement of Load Module .....	45
The part 2 VP Programming .....	47
1. Overview of the Fujitsu VP2600/10 Systems .....	49
1.1 Vector Processing .....	49
1.2 Architecture and Characteristics of VP2600/10 Systems .....	50
1.3 Operations on the Vector Processing Unit .....	51
1.4 Pipeline Processing .....	51
1.5 High-speed Performance Technologies .....	58
2. Introduction to Vectorization .....	62
2.1 Vector Performance .....	62
2.2 Speed Up by Vector Processing .....	65
2.3 Example of Vectorization .....	68
2.4 Vectorization .....	69
3. Basic Techniques of Vectorization .....	77

3.1	Vectorization Procedure .....	77
3.2	Vectorization Technique .....	79
4.	Vectorization by Improving Algorithm .....	104
4.1	Vectorization of the Point SOR Method .....	104
4.2	Vectorization of Multiple Integral .....	110
4.3	Vectorization of the Tridiagonal System of Equations .....	114
4.4	Vectorization of Elliptic Partial Differential Equations ..	118
4.5	Vectorization of Monte-carlo Method .....	124
The part 3	Assistant Tools on Vectorization .....	133
1.	Assistant Tools on Vectorization .....	135
2.	Sampling Analyzer .....	137
2.1	Procedure of Sampling Analyzer .....	137
2.2	Informations from Sampling Analyzer .....	138
2.3	Options .....	140
2.4	Relation between CPU Time and Sampling Count .....	140
2.5	Example of JCL .....	141
3.	Dynamic Analyzer .....	142
3.1	Procedure of Dynamic Analyzer .....	142
3.2	Informations from Dynamic Analyzer .....	143
3.3	Options .....	147
3.4	Example of JCL .....	148
3.5	Notes .....	150
4.	TOP10EX .....	151
4.1	Functions of TOP10EX .....	151
4.2	Execution Procedure .....	154
4.3	Options for TOP10EX .....	155
4.4	Example of JCL .....	162
4.5	Notes .....	163
Conclusion .....	164	
Acknowledgements .....	164	
References .....	165	
Appendix A Incompatibility Items .....	167	
Appendix B Cataloged Procedures for FORTRAN77EX .....	189	
Appendix C MERGER .....	201	

## はじめに

富士通製FORTRANコンバイラのバージョンアップに伴い、情報システムセンターでも新コンバイラ（V12）を平成4年5月より導入することとなった。V12コンバイラの新仕様の紹介と使用方法の説明、また、近年需要の高まってきたスーパーコンピュータVP2600の有効な使用方法を広める目的で、平成4年2月に「新FORTRANコンバイラ：V12とVPの効果的利用法」に関する講習会を開催した。

講習会資料はスーパーコンピュータFACOM VP-100の導入時に作成したJABRI-memo 59-215「原子力コードのベクトル化プログラミング〔1〕」（1984年7月）を参考にしながら、新たにV12コンバイラとVP2600ハードウェアに関する解説を付け加えた。

また、今回のV12コンバイラの導入にあわせて以下の作業を行った。

- (1) 代表的原子力コード16本のベンチマークテスト
- (2) V12コンバイラの新機能の性能評価
- (3) V12用カタログド・プロジェクトの整備
- (4) ベクトル化支援ツールTOP10EXの整備

(1)のベンチマークテストの結果をTable 1に示す。V12コンバイラでは、V10コンバイラと比較して平均で1.13倍の速度向上が得られた。

TOP10EXは、外来研で開発されたベクトル化支援ツールTOP10をV12コンバイラ用に改造したものである。TOP10は、プログラムの実行時の振る舞いを解析する動的解析ツールからの情報をプログラムに見やすい形に編集する。従来のTOP10が、動的解析ツールFORTUNEに対応していたのに対し、TOP10EXでは、今回新たに導入された動的解析ツールANALYZERに対応している。

本報告書は講習会の構成に従い3編に分かれている。第1編「FORTRAN77BXへの移行」では、V10コンバイラを使用していたユーザがV12コンバイラへ移行する手続きについて解説する。第2編「VPプログラミング」では、VP2600に適したプログラミング技法について解説する。第3編「FORTRAN77BXにおけるベクトル化支援ツール」では、ANALYZERとTOP10EXの使用方法について解説する。

Table 1 Results of the bench mark test (Apr. 1992)

コード名	V 1.2 コンパイラでの 実行時間	V 1.0 コンパイラでの 実行時間
ABOLUS	3 m 2 6. 0 9 s	4 m 0. 5 0 s
BBRMUDA	2 m 2. 5 8 s	2 m 1 2. 6 0 s
CITATION	6. 5 3 s	6. 8 0 s
DOT35	6 m 4 1. 9 4 s	6 m 5 3. 3 0 s
DSMC	3 m 4 2. 1 4 s	3 m 4 3. 2 0 s
FPGS	3. 0 7 s	3. 0 0 s
ISOFLOW	2 1 m 3 2. 8 2 s	2 3 m 2 9. 1 9 s
PHENIX	1 3. 6 4 s	1 6. 6 4 s
SLWALP	4 m 2 7. 4 2 s	5 m 9. 6 0 s
SONATINA	1 m 5 0. 3 4 s	2 m 6. 9 0 s
SPIN	2 m 1 9. 2 8 s	4 m 5. 9 4 s
SRAC	1 m 2 0. 4 7 s	1 m 2 2. 7 0 s
STREAM	2 m 2 5. 1 6 s	3 m 6. 8 4 s
TWOTRAN	3 m 3 8. 9 4 s	4 m 4. 9 0 s
VDIRECT	7 m 4 6. 4 7 s	8 m 3 1. 8 0 s
VIENUS	5 m 0 5. 1 0 s	4 m 5 9. 9 0 s

## 第1編 FORTRAN77EXへの移行

## 1. FORTRAN77EXの移行

FORTRAN77EX 及びFORTRAN77EX/VP( 以降、F77EX, F77EX/VPと略します。 また、両者をF77EX システム\*と略します。 ) は、実行性能の向上と次期国際標準仕様(FORTRAN90)への対応を目的に作られた新 FORTRANコンパイラです。 F77EXシステムでは、標準のFORTRAN77 文法書に記載されている仕様は全てサポートしており、ユーザは従来から使用しているプログラムをそのまま F77EXシステムで使うことができます。 しかし、従来のFORTRAN\*\*システム (以降、 F77システムと略します。 ) には、標準の文法以外の仕様にも対応していたため、この拡張仕様を用いたプログラムに関しては、 F77EXシステムを使ったときに従来とは異なった振る舞いをしたり、場合によっては文法エラーとなります (これらを非互換と呼んでいます)。 また、実行性能を強化した反面、翻訳時間、翻訳リージョン、実行リージョンは増大し、ジョブクラスを変更しなくてはならない場合もあります。

この他にも、オプション体系が大きく変わった等の理由から、 F77EXシステムを初めて使う人には戸惑いも多いことでしょう。 本編では、 F77EXシステムを有効に使うことを目的として、従来の FORTRANユーザが F77EXシステムへ移行するのに必要な手順を解説します。

\* OS IV/MSP FORTRAN77 EX 及び FORTRAN77 EX/VP V12 (富士通製)

\*\* OS IV/MSP FORTRAN77 及び FORTRAN77/VP V10 (富士通製)

## 2. FORTRAN77EXの強化項目

F77EXシステムでの新機能、強化機能を表2.1にまとめました。これらの詳細については、富士通マニュアル「FORTRAN77EX 使用手引書」及び「FORTRAN77EX/VP使用手引書」を参照してください。

表2.1 FORTRAN77EX システムにおける強化項目

■ サポート機能の強化	FORTRAN77システム(V1.0)	FORTRAN77EXシステム(V1.2)
・変数の名前の長さ ・引数の個数 ・文字定数長 ・一文の最大継続行数 ・8バイト整数型変数 ・英小文字の使用 ・アンダースコアの使用 ・インラインコメント ・暗黙の型宣言の抑止	6 文字以内 2 5 5 2 5 5 1 9 不可 不可 不可 不可 不可	31文字 (日本語15文字) 以内 1 0 0 0 3 2 7 6 7 9 9 可 可 可 可 可
日本語機能強化 ・日本語名標の使用	不可	可
ベクトル化原始プログラム表示 ・ループ・アンローリング展開数表示 ・外部手続き インライ化表示	不可 不可	可 可
精度縮小機能	OPT(0)のときのみ可	最適化実行、ベクトル実行時も可
アドレス拡張域を利用しての翻訳、翻訳＆実行	不可	可
単精度ベクトル演算	不可	可
■ 実行性能の強化		
外部手続きの インライ化展開	可	可 (制限緩和)
ベクトル関数の インライ化展開	不可 (スカラ関数のみ可)	可
多重ループ のベクトル化	最内ループが1つの多重ループをベクトル化	複数の最内ループを持つ多重ループをベクトル化
ループ・アンローリング	多重度は2	標準で実施、多重度は最大9
マスク付 ロードスト7命令の利用	不可	可

### 3. FORTRAN77EXへの移行手順

ここでは、F77システムで正常に動作していたプログラムを、F77EXシステムで翻訳～実行する場合の手順について説明します。但し、翻訳・結合・実行処理はバッチジョブを前提として話を進めます。

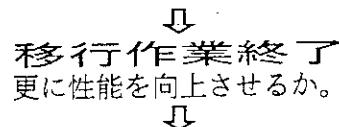
#### 3.1 移行手順

F77EXシステムへの移行は、非互換項目、非互換オプションに該当する場合には修正が必要ですが、特別なプログラムでない限り、JCLとオプションの修正くらいで済みます。

さて、それでは以下に移行手順を示します。処理中に生じたエラーの多くは、非互換に起因するものと考えられます。非互換によるエラーは実行時のみならず、翻訳時、結合編集時にも生じます。非互換項目に関しては、エラーメッセージと付録A「非互換項目資料」を参考にして修正を行ってください。実行ステップまで正常にジョブが終了したら、更に実行性能を上げるために、F77EXシステムの性能を最大限に生かしたチューニングを行います。

#### 《FORTRAN77EXへの一般的な移行手順》

- [STEP 1] 非互換項目が既知ならば、プログラムを翻訳する前に修正する。  
JCL、カタログド・プロジェクトの修正
- [STEP 2] 翻訳
  - 指定方法が変更された翻訳オプションの修正
- [STEP 3] 結合編集
  - オプション' MAP' を指定し、ロードモジュールの大きさをチェック
- [STEP 4] 実行
  - 実行リージョンの設定（結合編集時のMAPリストを参考）
  - 指定方法が変更された実行時オプションの修正



- [STEP 5] F77EXシステムを意識したチューニング
  - 4.5 「インラインオプション」、第2編「VPプログラミング」を参照

図3.1 に移行の流れを示します。

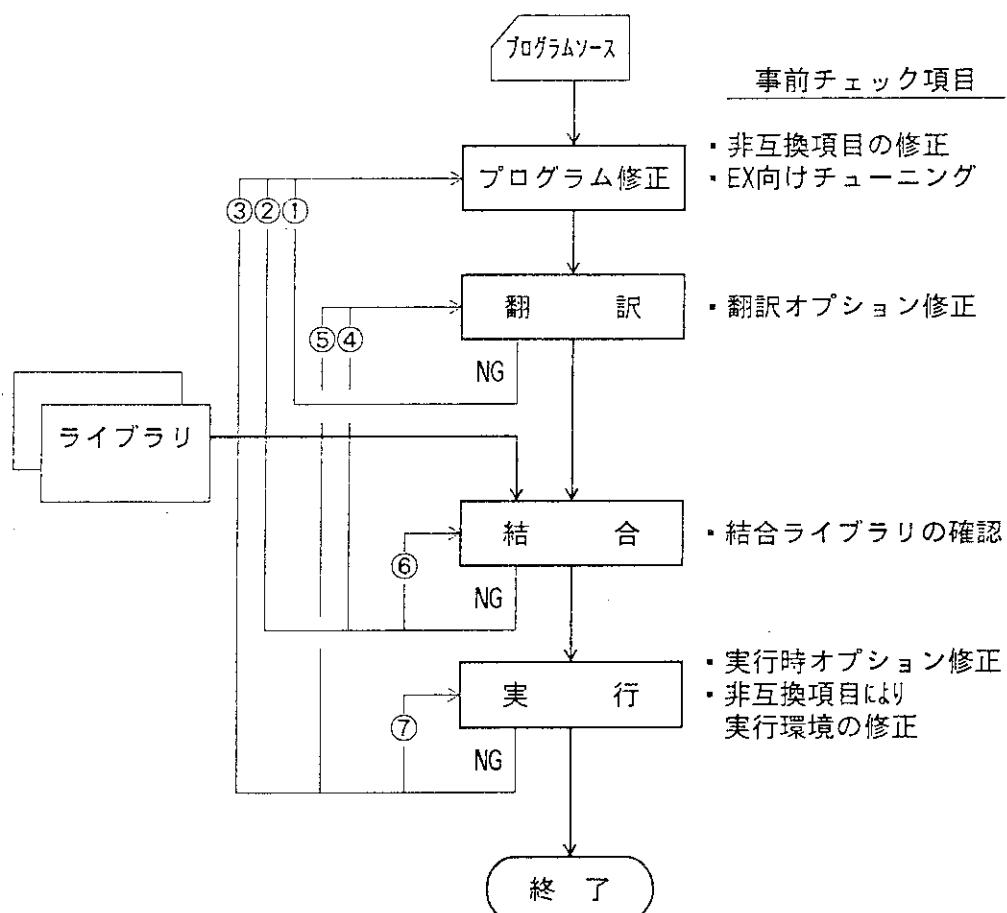


図3.1 FORTRAN77EXへの移行の処理の流れ

## No. 移行における作業の例

- ① 翻訳時に非互換項目を検出 ⇒ プログラム修正
- ② 結合編集時に非互換項目を検出 ⇒ プログラム修正
- ③ 実行時に非互換項目を検出 ⇒ プログラム修正
- ④ 結合編集時のMAP情報で、非AE化プログラムのロードモジュールの大きさが使用可能な基本領域の大きさを超えたことが判明 ⇒ AE化
- ⑤ 実行時に基本域の不足を検出（非AE化プログラム）⇒ AE化  
オプション指定に誤りがあり、ジョブが正常に終了しないとき⇒オプション修正
- ⑥ AE指定で作成されたオブジェクトと24bitアドレスのライブラリを結合する場合の処理 ⇒ リンケージエディタのオプションに'AMODE(31)'を指定
- ⑦ 実行時に非互換項目を検出 ⇒ 非互換項目を実行環境の変更により回避  
実行領域の不足 ⇒ 実行領域の拡大

### 3.2 FORTRAN77EXにおける処理手順

#### 3.2.1 カタログド・プロジェクト仕様

F77EXシステムをユーザが簡単に使用できるように、表3.1 のカタログド・プロジェクト（以降カタプロ）を用意しました。カタプロの内容に関しては付録B「カタログド・プロジェクト一覧」を参照して下さい。

表3.1 情報システムセンターの標準カタログド・プロジェクト

カタプロ名	処 理	F77システムで対応するが加
FORTEX	スカラ翻訳	FORTRAN77
FORTEXVP	ベクトル翻訳	FORTRAN77VP
LKEDEX	結合編集	LKED
LKEDCTEX	結合編集とロードモジュールの保存	LKEDCT
LKEDITEX	結合編集と一時ロードモジュールの作成	LKEDIT
LKEDUPEX	結合編集とロードモジュールの更新	LKEDUP
GOEX	実行	GO
LMGOEX	作成済ロードモジュールの実行	LMGO

#### ● FORTEX, FORTEXVP . . . スカラ翻訳, ベクトル翻訳

位置パラメタ	用 途	標準値
S O	ソースプログラムのデータセット名	NULLFILE
Q	ソースプログラムのデータセットのタイプ	.FORTRAN77
A	翻訳オプション	
B	翻訳オプション	
O P T	最適化レベル	※ 1
S R C	翻訳メッセージ出力を制御するオプション	※ 2
L C T	リスト情報の1ページの行数	0
O B J S	オブジェクトモジュールの大きさ (TRK 単位)	30, 10
D I S P	オブジェクトモジュールのディスポジション	NEW
S Y S O U T	SYSOUT出力クラス	*
R G N	翻訳リージョン (※ 5)	※ 3
E R G N	翻訳リージョン (拡張域)	※ 4

※ 1 : FORTEXでは'B'、FORTEXVPでは'E'

※ 2 : FORTEXでは無し、FORTEXVPでは'S, VMSG'

※ 3 : FORTEXでは5M、FORTEXVPでは18M

※ 4 : FORTEXでは1M、FORTEXVPでは10M (VP上で翻訳するときのみ有効)

※ 5 : M780上で翻訳する場合は基本域+拡張域の合計を指定、

VP上で翻訳する場合は基本域を指定。

ベクトル翻訳した場合のVPオプションの標準値は'VP(2600)'です。

● LKEDEX, LKEDCTEX, LKEDITEX, LKEDUPEX  
... 結合編集

位置パラメタ	用 途	標準値
RGN	結合編集リージョン	1024K
A	結合編集オプション	NOMAP
B	結合編集オプション	LIST
GRLIB	グラフィックライブラリ	NO
SSLA	科学技術計算用ライブラリ	JSSL
SSLB	科学技術計算用ライブラリ	SSL
SSLC	科学技術計算用ライブラリ	SSL2
SYSOUT	メッセージ出力先	*
WRKS	作業用ファイルの大きさ (TRK 単位)	30, 10
MODS	ロードモジュールの大きさ (TRK 単位)	30, 10, 1
PRVLIB	プライベートライブラリ	SYS9.NO
PRVQ	プライベートライブラリのタイプ	.LOAD
GGS	グラフィックライブラリ	SYS9.GGS
LM	※1 ロードモジュールのデータセット名	.LOAD
Q	※1 ロードモジュールのタイプ	
UNIT	※2 ロードモジュールのUNIT名	TSSWK
DISP	※2 ロードモジュールのディスポジション	NEW, CATLG,
CNTL	リンクエージ制御文が格納されているメンバ	NORMAL
UPS	※4 ロードモジュールの増分値	30
USER	ユーザID	OFF
CSPC	コンデンス作業用データセットの大きさ	30, 40, 10
CUNIT	コンデンス作業用データセットのUNIT名	TDS

※1 LKEDCTEX, LKEDITEX, LKEDUPEX ※2 LKEDCTEX ※3 LKEDITEX ※4 LKEDUPEX で指定

● GOEX, LMGOEX ... 実行

位置パラメタ	用 途	標準値
RGN	実行リージョン	8M
P NM	ロードモジュールのプログラム名	TEMPNAME
A	実行可能プログラムオプション	ERRCUT=0
SYSOUT	SYSOUT出力クラス	*
GOSYSIN	FT05F001に対するDD名の指定	DDNAME=SYSIN
ORECFM	SYSOUT出力のファイル編成	FBA
OBSIZE	SYSOUT出力のブロック長	19043
ORSIZE	SYSOUT出力のレコード長	137
LM	※ ロードモジュールのデータセット名	.LOAD
Q	※ ロードモジュールのタイプ	

※ LMGOEXで指定

装置参照番号 5 に対するデータセットは DD名 'SYSIN' で指定することができます。  
また、装置参照番号 6 はSYSOUT出力に設定されています。

### 3.2.2 翻訳オプションの標準値

翻訳オプションのシステム標準値、カタプロを使用した場合の標準値を表3.2に示します。

表3.2 翻訳オプションの標準値（続く）

原研における F77EXシステムインストール時 の標準オプション	カタプロ使用時の有効オプション		オプションの意味
	FORTEX	FORTEX/VP	
(1)最適化関連 OPTIMIZE (E) INLINE( INTRINSIC, ALL ) XOPT ( UNROLL , PREEX , EVAL )	OPT(B)  INTRINSIC, ALL  UNROLL , NOPREEX , NOEVAL	OPT(E)  INTRINSIC, ALL  UNROLL , PREEX , EVAL	最適化レベル 組み込み関数のインライン展開 ループアンローリング 不变式の先行評価 式の演算順序の変更
(2)ソース、出力情報、 メッセージ関連 NOASTER NOCONDINF FIXED LANGLVL(77) SSN(D) NOSEQ SRCODE(EB) FLAG(I) LINECOUNT(00) NOLIST LMSG NOMAP NOSOURCE NOSRCMSG NOSTATIS NOXREF PRINT NOTERM	NOASTER NOCONDINF FIXED LANGLVL(77) SSN(D) NOSEQ SRCODE(EB) FLAG(I) LINECOUNT(00) NOLIST LMSG NOMAP NOSOURCE NOSRCMSG NOSTATIS NOXREF PRINT NOTERM	NOASTER NOCONDINF FIXED LANGLVL(77) SSN(D) NOSEQ SRCODE(EB) FLAG(I) LINECOUNT(00) NOLIST LMSG NOMAP NOSOURCE   SOURCE NOSRCMSG NOSTATIS NOXREF PRINT NOTERM	アスタ行の扱いの指定 許容値を認めない演算の検出 原始プログラムの形式 原始プログラムの言語仕様レベル 文通番の指定 行識別番号の順序検査の指定 原始プログラムのコード種別 診断メッセージの出力レベル リスト情報の1頁あたりの行数 オブジェクトモジュールリスト出力の指定 診断メッセージの長さの指定 マップ情報の出力指定 原始プログラムリストの出力指定 診断メッセージにソースを付加する指定 コンパイラの統計情報の出力指定 相互参照リストの出力指定 リスト情報の出力指定 診断メッセージをSYSTEMへ出力
(3)VP関連 ADVANCED OCL NOVCHECK VMSG(2) VP(2600), VP2 VP2	— — — — — —	ADVANCED OCL NOVCHECK VMSG(2) VP(2600) VP2	実行モードの選択 最適化制御行を有効にする ベクトル化解析だけ(翻訳しない)の指定 ベクトル化メッセージの指定 VPハードウェアの指定 VPシリーズの指定
(4)精度関連 ALIGNC NODDOUBLE NOQUAD NOPR	ALIGNC NODDOUBLE NOQUAD NOPR	ALIGNC NODDOUBLE NOQUAD NOPR	共通ブロック内のデータ割り付け方法 精度拡張機能(倍精度)の指定 精度拡張機能(4倍精度)の指定 自動精度縮小機能の指定

表3.2 翻訳オプションの標準値（続き）

原研における F77EXシステムインストール時 の標準オプション	カタプロにおける標準オプション		オプションの意味
	FORTEX	FORTEXVP	
(5)その他			
AE	AE		巨大データの拡張域への割り付け
BYNAME	BYNAME		仮引数と実引数の結合方法
NOCOMPACT	NOCOMPACT		COMPACT オプション機能
NODEBUG	NODEBUG		デバッグ機能
ERRSSN(2)	ERRSSN(2)		実行時のトレースバックでの文通番の表示
NONAME	NONAME		リンクエディタのNAME制御文出力指定
NOLIL	NOLIL		言語間結合用のオブジェクトの出力指定
NORENT	NORENT		再入可能なオブジェクトの出力指定
NOSDF	NOSDF		シンボリックダンプ機能用のオブジェクトの出力
OBJECT	OBJECT		オブジェクトモデルの出力指定
NOSEQ	NOSEQ		行識別番号の順序検査の指定
SIZE(MAX)	SIZE(MAX)		コンパイラの使用する仮想記憶領域

翻訳時にどのオプションが有効になっているかを知りたい場合は、オプション'STATIS'を指定して下さい。'STATIS'を指定すると、メッセージリストにオプション値の一覧を出力します。

### 3.2.3 ユーザが設定するオプション

3.1.1で紹介したカタプロを使用した場合、翻訳オプションは原研でのシステムを考慮した標準的な設定がなされるようになっています。しかし、プログラムの特性や目的により、ユーザが意識して設定すべき翻訳オプションもあります。「ユーザが意識して設定する」という観点から翻訳オプションを以下のように分類しました。

また、F77EXシステムの翻訳オプションの中には、F77システムと指定の方法が異なるものがあります。これらのオプションについては4章で説明します。

	常に意識すべき	① ユーザが指定するオプション SOURCE ELM OPTIMIZE
	② 効果的なオプション VMSG XOPT INLINE STATIS LINECOUNT	
	③ その他の標準にセットされているオプション	

### 3.2.4 カタプロ使用例

FORTRAN ソースプログラム' J\*\*\*\*.TEST.FORT77' (PO ファイル) をベクトル翻訳～実行する例を示します。

#### ●翻訳～実行

```
//F77 EXEC FORTEXVP, SO='J****.TEST',
//      A='ELM(*)'
//LNK EXEC LKEDEX
//GO  EXEC GOEX
//SYSIN  DD DSN=J****.TEST, INPUT, DISP=SHR
```

●翻訳時にベクトル情報付きソース出力、ベクトルメッセージ出力、統計情報出力を指定、不变式の先行評価を抑止、またロードモジュール(' J\*\*\*\*.TEST.LOAD') を保存

```
//F77 EXEC FORTEXVP, SO='J****.TEST',
//      A='ELM(*), XOPT(NOPREEX)', SRC='S, VMSG, STATIS'
//LNK EXEC LKEDCTEX, LM=J****.TEST
//GO  EXEC LMGOEX, LM=J****.TEST
//SYSIN  DD DSN=J****.TEST, INPUT, DISP=SHR
```

●翻訳時に最適化レベルFを指定、また併せて外部手続き' ABC' のINLINE化を指定  
(ABCは実行ライン数が 30 以下とする)

```
//F77 EXEC FORTEXVP, SO='J****.TEST',
//      A='ELM(*), INLINE(EXT(ABC))', OPT=F
//LNK EXEC LKEDEX
//GO  EXEC GOEX
//SYSIN  DD DSN=J****.TEST, INPUT, DISP=SHR
```

### 3.3 TSSにおけるFORTRAN77EXの使用

TSS上でFORTRAN77EXを用いて翻訳・実行する場合は以下のコマンドを使用します。

● 翻訳

FORT77 [データセット名] \* [翻訳オプション]

● 翻訳～実行

RUN [データセット名] \*

もしくは

FORT77 [データセット名] \* [翻訳オプション] GO

従来のFORTRANコンパイラ(V10)を使用する場合には、以下の様にします。

● 翻訳

LIB ('PP1.FORT77.LINKLIB' 'PP1.FORT77.CMDLIB')

FORT77 [データセット名] \* [翻訳オプション]

● 翻訳～実行

LIB ('PP1.FORT77.LINKLIB' 'PP1.FORT77.CMDLIB' 'PP1.FORT77.FORTLIB')

RUN [データセット名] \*

もしくは

FORT77 [データセット名] \* [翻訳オプション] GO

\* エディタ上で翻訳する場合はデータセット名は必要ありません。

・ タイプ名が'FORT77'の場合は、タイプ名を省略することが出来ます(FORT77コマンドのみ)。

[使用例]

FORTRANソースプログラム J\*\*\*\*.TEST.FORT77(POデータセット)を翻訳、実行する例を示します。

① 翻訳

FORT77 TEST ELM(\*)

② 翻訳～実行

FORT77 TEST ELM(\*) GO

③ V10コンパイラによる翻訳～実行

LIB ('PP1.FORT77.LINKLIB' 'PP1.FORT77.CMDLIB' 'PP1.FORT77.FORTLIB')

FORT77 TEST ELM(\*) GO

## 4. オプションの指定

F77EXシステムでは F77システムに比べて新たな機能が追加され、オプション体系も新機能を意識したものとなりました。 本章では、指定方法が変更されたオプション(4.1～4.6 では翻訳オプション、4.7 では実行時オプション)について解説します。 センターの標準カタログ(FORTEX, FORTEXVP)をそのまま使用する場合には、特にオプションを修正する必要はありませんが、個別にオプションを指定する場合は、指定方法が変更されたオプションがないかどうか表4.1, 表4.9～表4.11, 表4.13～表4.17で確認してください。

### 4.1 最適化に関するオプション

最適化の機能が強化されると共に、一部指定方法が大きく変更されました。

表4.1 オプションの変更1：最適化に関するオプション

内 容	FORTRANT	FORTRANTEX
最適化レベルの選択	OPTIMIZE (0) OPTIMIZE ((1   2   3))	NOOPTIMIZE OPTIMIZE (({B   E   F}))
組み込み関数の展開	XOPT (({IL   INIL   NOIL}))	INLINE(INTRINSIC, {ALL   INMOST}) INLINE(NOINTRINSIC)
外部手続きの展開	PI ({pgmlist}   *)	INLINE(EXTERNAL({pgmlist} [, stno] [, dtszK]))
ループアンローリング	ITR	XOPT ((UNROLL)
不变式の先行評価	XOPT(AMOVE)	(, PREEX)
演算評価方法の変更	ADV(EVL) ※ ※ VP翻訳時のみ有効	(, EVAL))
最適化に関する メッセージの出力	XOPT ((TMSG) 同上 [, MSG])	OPTMSG ((INLINE) (, UNROLL) (, PREEX) (, EVAL))
・外部手続きインライン ・ループアンローリング ・不变式の先行評価 ・演算評価方法変更	—	—

[ ] は省略可能なことを意味します。

[ ] は | で区切られた要素をひとつ選択することを意味します。

— は対応する機能が無いことを意味します。

#### 4.1.1 最適化レベル

F77システムでは OPT(0) ~OPT(3)、F77EXシステムでは NOOPT, OPT(B) ~OPT(F)のそれぞれ4段階ずつの最適化レベルを持っていますが、それぞれの段階のレベルが同等の最適化機能を持っているわけではありません。両者の各レベルにおいて、実施される最適化の範囲と機能を表4.3~表4.5に示しましたので、両者の最適化レベルの違いに注意して下さい。

#### ■ 最適化レベル

表4.2 最適化レベルの非互換

FORTRAN77		FORTRAN77EX	
スカラ	VP	スカラ	VP
OPT(0)	選択不可能	NOOPT	-
OPT(1)	(レベルは	OPT/OPT(B)	OPT(B)
OPT(2)	OPT(3)相当	OPT(E)	OPT/OPT(E)
OPT(3)	)	OPT(F)	OPT(F)

F77EXシステムではVP翻訳時にも3つのレベルが選択出来ます。

#### ■ 最適化の実施範囲

表4.3 最適化の実施範囲

最適化レベル							
	0	NO	1	B	2	E	3
④プログラム間							
③広域的							
②DOループ							
①局所的	■	■	■	■	■	■	■

■ 標準で実施      ■ オプションにより実施

#### ① 局所的な最適化

1つの実行文の中に閉じた最適化が行われる。

#### ② DOループ内の最適化

OPT(1), NOOPTでは、配列要素と文字部分列から生成される各種の計算式(添字の計算式、文字開始位置及び文字長の計算式)に限定して最適化が行われる。

#### ③ 広域的な最適化

DOループの他、IF文やGOTO文などの制御文で構成される繰り返し部分もループと認識して、最適化が行われる。

#### ④ プログラム単位間の最適化

外部手続きを開いたうえで、各種の最適化が行われる。

## ■ 最適化レベルと最適化項目

各々の最適化機能はF77EXシステムの方がより強化されています。

表4.4 FORTRAN77 における最適化の内容

最適化機能		オプション	OPT0	OPT1	OPT2	OPT3
局所的な最適化	組込み関数の内部展開 IF文の最適化 演算式の最適化		● ●	● ●	● ●	● ●
テキストの最適化	共通式の削除 不变式の移動 不变式の先行評価 演算評価方法の変更 ループアンローリング	XOPT(AMOVE) ADV(EVL)/NOADV(EVL) ITR		● ● △	● ● △	●※ ●※ ● ●VP △
レジスタ割当の最適化				●	●	●※
分岐の最適化					●	●※
外部手続きの内部展開	PI					△

※ : OPT(2)よりも拡張された最適化が行われる。 VP : VP翻訳時に有効

表4.5 FORTRAN77EX における最適化の内容

最適化機能		オプション	NOPT	OPTB	OPTC	OPTF
局所的な最適化	組込み関数の内部展開 IF文の最適化 演算式の最適化	INLINE(INTRINSIC)	● ●	○ ● ●	○ ● ●	○ ● ●
テキストの最適化	共通式の削除 不变式の移動 不变式の先行評価 演算評価方法の変更 ループアンローリング	XOPT(PREEX) XOPT(EVAL) XOPT(UNROLL)	● ● △	● ● ○ ○	● ● ○ ○	● ● ○ ○
レジスタ割当の最適化			●	●	●	●
分岐の最適化				●	●	●
外部手続きの内部展開	INLINE(EXTERNAL)		△	△	○	

- 標準で実施される
- 標準で実施され、オプション指定により抑止可
- △ 標準で実施されないが、オプション指定により実施可  
限定した最適化

## ■ 最適化レベルの選択

実行速度の観点からみれば、上位の最適化レベルで翻訳するのが望ましいのですが、最適化が促進される程ロードモジュールが大きくなる、翻訳時間が長くなるといった欠点が生じてきます。また、OPT (E) 以上のレベルでは、プログラムによっては計算結果に違いを生じたり、実行エラーを生じることがあります（これを最適化による副作用と言います）。移行時の最適化レベルは表4.6を参考にして決めるに良いでしょう。F77システムで翻訳したことのないプログラムに関しては表4.8 の副作用を生じる可能性のある最適化に注意して最適化レベルを決定して下さい。また、ロードモジュールを増大させたくない場合は、オブジェクトの増大を招く最適化を抑止すると良いでしょう。

ベクトル翻訳の場合はベクトル化の効果を十分に發揮するため、OPT (E) 以上のレベルを選択することをお勧めします。

表4.6 最適化レベルの移行

FORTRAN77	FORTRAN77EX
OPT (0)	・・・ 該当レベル無し
OPT (1)	⇒ NOOPT
OPT (2)	⇒ OPT (B)
OPT (3)	⇒ OPT (E)
該当レベル無し	・・・ OPT (F)

表4.7 FORTRAN77EXにおける最適化レベルの比較

レベル (意味)	最適化レベルの主旨	下位レベルとの差
NOOPTIMIZE	最も基本的な最適化が実施される	
B (BASIC)	実行に影響のない最適化を実施	ループアンローリング※ 組み込み関数インライン展開※
E (EXTENDED)	OPT(B)での最適化に加え、副作用のある最適化を実施	不变式の先行評価 演算評価順序の変更
F (FULL)	OPT(E)での最適化に加え、外部手続きのインライン展開を実施	外部手続きのインライン展開※

※：オブジェクトの増大を招く最適化

更に、性能向上を望む方は外部手続きのインライン展開の機能を活用して下さい。外部手続きのインライン展開はOPT (F) のレベルで標準で実施されます。但し、必要以上に外部手続きのインライン展開をするとロードモジュールが大きくなったり、性能の劣化を招くことがあります。外部手続きのインライン展開の機能を使う前に4.5 「インラインオプション」の項を読むことをお勧めします。

#### 4.1.2 副作用を生じる可能性のある最適化

F77システムでADV(NOEVL)（或いはNOADV(NOEVL)）、またはXOPT(NOAMOVE)オプションを指定して翻訳していたプログラムは、最適化による副作用\*を生じるプログラムです。

F77EXシステムで翻訳するときも、最適化による副作用を抑止するようにオプションを指定する必要があります。

表4.8 副作用を生じる可能性のある最適化の抑止

機能	FORTRAN77	FORTRAN77EX
不变式の先行評価の抑止	XOPT(NOAMOVE)	XOPT(NOPREEX)
演算順序の評価方法変更の抑止	ADV(NOEVL) 或いは NOADV(NOEVL)	XOPT(NOEVAL)

また、F77EXシステムではこれらの最適化の機能が強化されているために、F77システムでは正常に動作していたプログラムが、F77EXシステムでは副作用を生じる場合もあります。

副作用を生じる可能性のある最適化はOPT(E)以上のレベルで実施されます。副作用を生じる可能性のある最適化が実施されたかどうかは、OPTMSGオプションを指定すれば翻訳時のメッセージで知ることができます。

OPTMSG([PREEX][, EVAL])

#### 最適化による副作用\* :

正常なプログラムに対し、不变式の先行評価、演算順序の評価方法の変更の最適化を実施した場合にエラーを生じたり、また演算結果が異なったりすることを最適化による副作用といいます。不变式の先行評価は、DOループ中で不变な式をDOループの外へ移動して先行評価する最適化です。この際、条件式によって本来は評価されないはずの式が、無条件に先行評価されることで0割りや「の中が負になる等のエラーを生じることがあります。演算順序の評価方法の変更はベクトル実行する場合に特に効果的です。この最適化では、1実行文に閉じた範囲で演算の順序を変更します。このため、計算誤差に影響を与えます。

4.2 ソースプログラムに関するオプション

表4.9 オプションの変更2：ソースプログラムに関するオプション

内 容	FORTRANT7	FORTRANT8EX
許容値を認めない演算の検出を指定	-	CONDINF
可変長ソースの開始位置	-	FIRSTCOL ( {13   c} ) c = 5~255
内部文番号の順序付	ISN (( [C   D] ))	SSN (( [C   D] ))
ソースコードの指定 ・A S C I I ・E B C D I C ・B C D	- EBCDIC BCD	SRCODE ( {EA   EB} ) -
リレーショナルデータベース 使用の指示	AQL -	SQL -

〔内部文番号の順序付け〕

オプション指定方法が変更されました。意味は同じです。

$$\begin{aligned} \text{ISN (C)} &\Rightarrow \text{SSN (C)} \\ \text{ISN (D)} &\Rightarrow \text{SSN (D)} \end{aligned}$$

4.3 翻訳時の印刷情報に関するオプション

表4.10 オプションの変更3：翻訳時の印刷情報に関するオプション

内 容	FORTRANT7	FORTRANT8EX
診断メッセージの出力レベルの指定	FLAG ( {I   W   E   S} )	FLAG ( {I   W   S} )
リストの1ページの行数の指定	LINECOUNT ( {60   l} ) l = 0 ~99	LINECOUNT ( {60   l} ) l = 0 ~999
ソースリスト出力 ・深さの明示 ・インデント ・エラー文の直後にメッセージ出力 ・イクルードの展開 ・最適化ソース出力	SOURCE ( ([LEVEL] [, INDENT]) ) - INSOURCE SOURCE (CONVERT)	SOURCE ( ([NEST] [, INDENT] [, MSG] [, INCLUDE]) ) -

〔メッセージの出力レベルの指定〕

F77システムにおけるEレベルの翻訳診断メッセージは、Sレベルに統合されました。

#### 4.4 VP翻訳に関するオプション

F77EXシステムでは、VP2000シリーズ、及びVP1000シリーズ向けのオブジェクトを作成することができます。従来のFORTRAN77/VPコンパイラでは、VP400E向けのオブジェクトまでしか作ることができませんでしたが、F77EX/VPコンパイラではVP2600に最適なオブジェクトを作ることが可能です。

VP翻訳に関するオプションで指定方法に変更があったものを表4.11に示します。

表4.11 オプションの変更7：VP翻訳に関するオプション

内 容	FORTRAN77/VP	FORTRAN77EX/VP
実行モードの選択	$\{ \text{ADV} \}$ [ $\{ \text{NOADV} \}$ ] [ $\{ \text{EVL} \}$ ] $\{ \text{NOEVL} \})$	$\{ \text{ADV} \}$ $\{ \text{NOADV} \}$
最適化制御行を有効にする	V O C L	O C L
VPオプション	VP [( ANY   30   50   100   200   400 ) ] — —	VP [( ANY   30   50   100   200   400   1100   1200   2100   2200   2400   2600 ) ]
ハードウェアリースの指定	[NOVPE   VPE] —	[VPO   VPE   VP1   VP2 ]
ベクトル化表示付ソース出力	V S O U R C E	S O U R C E
チューニングメッセージの出力	V T U N E	—
ベクトル化メッセージの出力	VMSG [( D E T A I L ) ]	VMSG [( [ 1   2   3   DETAILED ] ) ]
単精度演算の倍精度化	標準で実施	V D P R E C
DOループの回転数の指定	—	R E P E A T ( n )

#### [単精度演算のベクトル化]

F77EX/VPでは単精度演算のベクトル化が可能です。F77/VPでは単精度演算を倍精度化してベクトル化します。

#### 4.4.1 ベクトル化表示付ソース出力

F77EX/VPではオプション' SOURCE' (省略形 'S') を指定したときに、標準でベクトル化情報と最適化情報を原始(ソース)プログラムに併せて出力します。 従来の' VSOURCE' (省略形' VS' ) は意味を持ちません。

表4.12 ソースオプションと出力情報

オプション	FORTRAN77/VP	FORTRAN77EX/VP
SOURCE	ソース出力	ベクトル化表示及び最適化表示付 ソース出力
VSOURCE	ベクトル化表示付ソース出力	'SOURCE'に機能統合された

以下に、ベクトル化表示付ソースリストについて説明します。

<pre> S 2      DO 20 J=1,N V 2      DO 10 I=1,N V 2      A(I,J)=B(I,J)+C(I) V 2      10 CONTINUE S 2      20 CONTINUE </pre>	⇔DO 20 のループが多重度 2 で アンローリングされている
<pre> V       DO 30 J=1,N V 1      CALL SUB(A(3,J)) V       30 CONTINUE </pre>	⇔SUBROUTINE SUBがインライン 展開されている
	最適化表示 . . . . . 2 ~ 9 : ループアンローリングの展開数 ベクトル化表示 : 外部手続きのインライン化

スカラ翻訳の場合は、オプション' SOURCE' を指定しても最適化表示はされません。 ループアンローリングやインライン展開の最適化が実施されたかどうかは、OPTMSGオプションを指定すれば翻訳時のメッセージで知ることができます。

OPTMSG ([UNROLL] [, INLINE])

#### 4.4.2 VPハードウェアの指定

VPハードオプションとVPシリーズオプションによって、使用するハードウェアに最適なオブジェクトを生成します。原研での両オプションの標準値は' VP(2600), VP2' となっています。VP(2600)以外のVPハードウェアを指定したい場合は、表4.13に従って両オプションを指定して下さい。

VP [ ( {ANY/30/50/100/200/400/} ) ]	1100/1200/ 2100/2200/2400/2600 } ) ]
VPシリーズオプション の指定が必須	

※ 機種番号を指定せずVPとだけ指定した場合は、' VP(2600), VP2' が指定されたとみなされる。

表4.13 VPオプション, VP シリーズオプションとハードウェアの対応

VPオプション		VPシリーズオプション		VPシリーズオプション	
		VPO	VPE	[VP1]	[VP2]
VP(ANY) 又は VP	VP( 30)	VP30	VP30E	VP1100	×
	VP( 50)	VP50	VP50E	VP1200	×
	VP(100)	VP100	VP100E	VP2100	
	VP(200)	VP200	VP200E	VP2200	
	VP(400)	VP400	VP400E	VP2400	
		VP(2600)		VP2600	

#### 指定例

- |               |  |
|---------------|--|
| VP(2600), VP2 | ・・・ VP2600ハードウェア向けにベクトル化                 |
| VP(2600)      | ・・・ ' VP(2600), VP2' と同じ                 |
| VP(ANY), VPE  | ・・・ VPシリーズEモデル向けにANYでベクトル                |
| VP(ANY), VPO  | ・・・ VPシリーズ向けにANYでベクトル化                   |
| VP(ANY), VP2  | ・・・ VP2000シリーズ向けにANYでベクトル化               |
| VP(400), VPE  | ・・・ VP400Eハードウェア向けにベクトル化                 |
| VP            | ・・・ インストール時の設定に従う(原研では' VP(2600), VP2' ) |

#### 4.5 インラインオプション

F77EXシステムで強化された最適化項目の一つにインライン展開機能があります。インライン展開機能は、手続きを呼び出し元に展開することで実行性能を向上させようというものです。F77EXでのインライン展開の強化ポイントは次の2点です。

- ベクトル関数のインライン展開
- 外部手続きのインライン展開の制限の緩和

##### 4.5.1 インラインオプションの指定方法

F77EXシステムではインライン展開機能の強化に伴い、オプションの指定方法が大きく変更されました。

■ 組み込み関数の展開                          ···· O P T (B) 以上のレベルで標準実施

```
INLINE( INTRINSIC ([, ALL], INMOST))
INLINE( NOINTRINSIC)
```

INLINE(INTRINSIC, ALL)	··· 引用箇所に無関係に展開
INLINE(INTRINSIC, INMOST)	··· 最内ループでの引用のみ展開
INLINE(INTRINSIC)	··· INTRINSIC, ALL と同じ
INLINE(NOINTRINSIC)	··· 組み込み関数の展開を抑止

■ 外部手続きの展開                          ···· O P T (F) で標準実施

```
INLINE( EXTERNAL [((pgnlst) [, stno] [, dtszK])])
```

pgnlst : 外部手続き名の並び(省略すると全ての外部手続きが対象)  
 stno : 対象となる外部手続きの最大実行文数(省略すると30)  
 dtszK : 配列(共通ブロックを除く)の大きさがdtsz\*10<sup>3</sup>バイト以下の  
         外部手続きが対象となる。(省略すると無制限)

INLINE(EXT(50))	··· 実行文が50以下の外部手続きを展開
INLINE(EXT(SUBA, SUBB, FUNC))	··· INLINE(EXT(SUBA, SUBB, FUNC, 30))と同じ
INLINE(EXT)	··· INLINE(EXT(30))と同じ
INLINE(NOINTRINSIC, EXT)	··· 組み込み関数の展開はせずに、 外部手続きを展開
INLINE(EXT(2000K))	··· 配列の大きさが2000K以下、実行文が 30以下の外部手続きを展開
INLINE(EXT(SUBA, SUBB, FUNC, 50, 2000K))	··· SUBA, SUBB, FUNCのうち実行文が50以下 配列の大きさが2000K以下の 外部手続きを展開

#### 4.5.2 外部手続きのインライン展開の活用

F77EXシステムでは F77システムに比べて、インライン展開できる外部手続きの条件が緩和され、より広範囲の条件の外部手続きをインライン展開することが出来るようになりました（インライン展開できる外部手続きの条件は「FORTRAN77EX 使用手引書」を参照）。最適化レベルOPT(F)では標準で、実行文が30以下の外部手続きをインライン展開の対象としています。インライン展開が行われたかどうかを知りたい場合は、オプション'OPTMSG(INLINE)'を指定するか、またはベクトル翻訳時のベクトル化表示付ソースリスト上で確認します。

##### (1) 既存のプログラムに外部手続きのインライン展開を適用する場合

インライン展開は次のような目的で使用されます。

- ① ベクトル化の促進  
(DOループ内に外部手続き呼び出しがあると、そのDOループはベクトル化が出来ないため、外部手続きをDOループ内に展開する)
- ② 呼び出し元と一体となった各種最適化の促進
- ③ 外部手続き呼び出しに要するコストの削減

プログラムによっても異なりますが、一般には①>②>③の様にインライン展開の効果（実行性能の向上）があります。インライン展開をすると、呼び出し元毎に手続きを展開するので、ロードモジュールは増大します。また、呼び出し元の手続きが増大するために、最適化の対象となっていた変数が、展開後に最適化の対象から外されてしまい、結果として、実行性能を悪化させてしまう場合もあります（コンパイラは一つの手続きの中で、使用頻度の高い幾つかの変数に限定して最適化を行う）。従って、インライン展開する外部手続きはプログラマの判断によって選択した方が良いでしょう。以下にインライン展開する外部手続きを選ぶときの指針をあげておきます。

- インライン展開することでベクトル化が可能になる場合  
・・・ 実際にプログラムを見て判断する
- インライン展開する外部手続きの規模が小さい  
・・・ 実行文数、ローカル配列の大きさを確認する
- 呼び出し回数が非常に多い  
● 呼び出しに要するオーバヘッドが大きい・・・ ANALYZERの実行詳細解析を参考

##### (2) 新たにプログラムを作成する場合

呼び出しオーバヘッドが大きい、あるいはベクトル化ができないという理由から外部手続き化出来なかった処理も、インライン展開することを前提に外部手続きとすることが出来ます。このことにより、より読みやすいプログラムが作成出来ます。

#### 4.6 その他の翻訳オプション

その他の翻訳オプションで指定方法が変更されたものを表4.14～表4.16に示します。

##### 4.6.1 コンパイラの動作に関するオプション

表4.14 オプションの変更4：コンパイラの動作に関するオプション

内 容	FORTTRAN77	FORTRAN77EX
翻訳から実行まで行うことの指定	GO [ ( {W   E   S} ) ] —	GO [ ( {AE   NOAE} ) ] —

拡張域を使用しての翻訳・実行が可能となりました。また、F77EX システムでは、翻訳時のエラーレベルが 4 を超えた場合は結合編集と実行は行われません。

##### 4.6.2 オブジェクトモジュールに影響するオプション

表4.15 オプションの変更5：オブジェクトモジュールに影響するオプション

内 容	FORTTRAN77	FORTRAN77EX
トレースバックにおける文番号の表示	ERRISN[( {0   1   2} )]	ERRSSN[( {0   1   2} )]

##### 4.6.3 デバッグに関するオプション

表4.16 オプションの変更6：デバッグに関するオプション

内 容	FORTTRAN77	FORTRAN77EX
文番号のトレース 副プログラムのトレース 値の割当て表示 添字の検査 実・仮引数の対応 未定義データの引用 整数演算のオーバーフロー 重複実引数の値変更	DEBUG [ ( [TRACE] [, SUBTRACE] [, INIT] [, SUBCHECK] [, ARGCHECK] [, UNDEF] [, I OVERFL] ) ] —	DEBUG [ ( [TRACE] [, SUBTRACE] [, INIT] [, SUBCHECK] [, ARGCHECK] [, UNDEF] [, I OVERFL] [, OVERLAP] ) ]

#### 4.7 実行可能プログラムオプション

実行可能プログラムオプション（実行時オプション）は実行ステップのPARMパラメタに以下の形式で指定します。センターの標準カタログGOEX, LMGOEXを使用する場合は、位置パラメタAにオプションのサブパラメタのみを記述します。

F L I B (サブパラメタ [, サブパラメタ . . . ] )

実行可能オプションのうち指定方法が変更されたものについて表4.17に示します。

表4.17 オプションの変更 7 : 実行可能プログラムオプション

内 容	FORTRAN77でのサブパラメタ	FORTRAN77EXでの用法
行送り制御文字の チェック	A N S I = { 0   1 }	{ A N S I   N O A N S I }
トレースバックマップ プログラム 単位の仮引数値を 出力	A R G O U T = { 0   1 }	{ A R G O U T   N O A R G O U T }
1ブロックを複数の FORTRAN 記録として 扱う データセットの 動的割当て	D F B = { Y E S   N O }	{ D F B   N O D F B }
	D Y N A L L O C = { 0   1 }	{ D Y N A L L O C   N O D Y N A L L O C }
高速入出力機能指定	H I O = ( u 1 [, u 2] . . . )	
整数型オーバフロー による割り込み検出	I O V E R F L = { 0   1 }	{ I O V E R F L   N O I O V E R F L }
実行時の仕様の指定 F77システムとの動作互換	L A N G L V L = { 77   66 } -	※翻訳の時点で仕様を指定する { N O R U N 77   R U N 77 }
入力促進メッセージ 出力	P R O M P T = { Y E S   N O }	{ P R O M P T   N O P R O M P T }
指数下位桁あふれに による割り込み検出	U N D E R F L = { 0   1 }	{ U N D E R F L   N O U N D E R F L }

#### [高速入出力処理の指定]

F77EXシステムではDD名で高速入出力処理の指定をします。

F77システムでのHIO の指定

```
//GO EXEC GO, A='HIO=(1, 2, 3)'
//FT01F001 DD ~
//FT02F001 DD ~
//FT03F001 DD ~
```

F77EXシステムでのHIO の指定

```
⇒ //GO EXEC GOEX
//FT01S001 DD ~
//FT02S001 DD ~
//FT03S001 DD ~
```

#### [FORTRAN77システムとの動作互換]

オプションRUN77 を指定することにより、一部の非互換項目が解決されます。

## 5. FORTRAN77EXシステムの活用

### 5.1 アドレス拡張機能の活用

F77EXシステムへの移行にあたり、アドレス拡張機能のオプション'AE'をシステム標準値としました。従って、ユーザがオプション'NOAE'を指定しない限り、F77EXシステムによって作成される実行可能プログラムはアドレス拡張域を使って実行されます。また、F77EXではアドレス拡張機能を使っての翻訳が可能となり、従来のFORTRANシステムでは扱えなかった大規模プログラムへの対応がなされています。本章では、アドレス拡張機能と、アドレス拡張機能を使うための作業(AE化)について解説します。

#### 5.1.1 アドレス拡張機能の概要

アドレス拡張(AE)機能とは、16MB以下の仮想記憶領域(基本域)だけを使用して実行していたプログラムを16MBを超えた仮想記憶領域(拡張域)も使用して実行出来るようにする機能です。ここで注意しなければいけないことは、基本域には実行可能プログラム\*を実行する領域の他にシステム制御プログラムが使用する領域も存在するために、実行可能プログラムが使用出来る大きさは16MBよりも小さくなっているということです。原研では、実行可能プログラムが使用可能な基本域の大きさは8MB程度になっています。

F77EXシステムにおけるアドレス拡張機能は以下の通りです。

- 実行可能プログラムの拡張域での実行
- 巨大データ(配列)の拡張域への割り付け
- 拡張域を使っての翻訳

\* 実行可能プログラム：

プログラムから拡張域に割り付けられるデータ領域を除いた部分。ロードモジュール。

### 5.1.2 実行可能プログラムの拡張域での実行

使用可能な基本域の大きさを超える実行可能プログラムは A E 化する必要があります。 A E 化により、最大で 16 MB の実行可能プログラムを動作させることができます。

#### ■ 拡張域でプログラムを実行するときの手続き

##### (1) 翻訳時

- ① 翻訳オプション A E を指定する
- ② 翻訳オプション G O を指定する場合はサブパラメタに ' A E ' を指定する  
G O (A E)

##### (2) 結合編集時

- ① 24 bit アドレスのライブラリを結合する場合は、リンクエディタのオプションに ' AMODE(31)' を指定する。  
(情報システムセンターが用意しているライブラリのうち、一部のグラフィック・ライブラリに 24 bit アドレスのものがあります。)
- ② オーバーレイ構造\*\*のプログラムの場合は、オーバーレイ構造を外す。

#### \*\*オーバーレイ構造：

一定の大きさの主記憶領域を、プログラムの各部分が入れ替わり使用することによって、その領域より大きなサイズのプログラムを実行させるためにプログラムに与える構造。 オーバーレイ構造のプログラムは拡張域で実行出来ない。

### 5.1.3 巨大データの拡張域への割り付け

巨大データを拡張域に割り付けるには、翻訳オプションA EまたはA E (共通ブロック名)を指定します。これらのオプションによって拡張域に割り付けられるデータは16MBを超える巨大データ(配列)とすることが出来ます。拡張域に割り付けられたデータは実行可能プログラムの外に動的に確保されるので、実行可能プログラムの大きさは割り付けられるデータの大きさを除いた大きさとなります。一方、A E化しないと全てのデータ領域は実行プログラム中に静的に確保されます。



commonlist : 共通ブロック名(COMMON名)の並び

#### ■ A E オプションによる拡張域への割り付け

A Eと指定した場合、以下のデータ領域が拡張域に動的に確保されます。

- 共通ブロック(COMMON)に属する変数及び配列
- 共通ブロック(COMMON)に属さない変数及び配列

但し、以下の条件が満たされていなければなりません。

- ① 共通ブロックに属する変数及び配列に対して、また共通ブロックに属さない変数及び配列に対してもBLOCK DATAやDATA文で初期値が設定されていないこと。
- ② 同じ共通ブロックを含む全てのプログラム単位がA Eオプションを指定して翻訳されていること。

次に、A Eオプションでデータを拡張域へ割り付ける例をあげます。

#### ● 例1 翻訳オプションA Eによる巨大データの使用

PRGAとSUBAを別々に翻訳し、リンクエディタで結合編集したとします。共通ブロック COMに属する配列と変数、及び配列X、Y、Zが拡張域に動的に確保されます。

@FORTRAN AE

```

PROGRAM PRGA
COMMON /COM/A, B, C
DIMENSION C(100, 100)
REAL X(500, 500)
:
CALL SUBA
:
END

```

@FORTRAN AE

```

SUBROUTINE SUBA
COMMON /COM/AA, BB, CC
DIMENSION CC(100, 100)
REAL Y(100), Z(100)
:
END

```

● 例 2 AE 化されない共通ブロックの例(1)

PRGAとSUBAを別々に翻訳し、リンクエディタで結合編集したとします。 共通ブロックCOM1に属する変数に初期値が設定されているため拡張域に動的に確保されるのは配列Y、Zだけになります。

<pre>@FORTRAN AE PROGRAM PRG1 COMMON /COM1/A, B, C  DIMENSION C(100, 100) : CALL SUBA : END</pre>	<pre>@FORTRAN AE SUBROUTINE SUBA COMMON /COM1/AA, BB, CC  DIMENSION CC(100, 100) REAL Y(100), Z(100) : END</pre>
---	--

C

<pre>BLOCK DATA BLOCK1 COMMON /COM1/A, B, C DATA A, B/1.0, 10.0/ : END</pre>
--

● 例 3 AE 化されない共通ブロックの例(2)

PRG2を'AE'で翻訳し、SUB2を'NOAE'で翻訳して結合編集した場合、共通ブロックCOM2は実行可能プログラムの中に確保され、拡張域は使用されません。

<pre>@FORTRAN AE PROGRAM PRG2 COMMON /COM2/D, E, F DIMENSION F(100, 100) : CALL SUB2 : END</pre>	<pre>@FORTRAN NOAE SUBROUTINE SUB2 COMMON /COM2/DD, EE, FF DIMENSION FF(100, 100) : END</pre>
--	---

### ■ 共通ブロック名付き A E オプションによる拡張域への割り付け

A E (*commonlist*) と指定した場合、以下のデータ領域が拡張域に動的に確保されます。このときに確保される大きさは、その共通ブロックを含む最初に実行されたプログラム単位での共通ブロックの大きさとなります。

#### ● *commonlist*に指定した共通ブロック (COMMON) に属する変数及び配列

但し、以下の条件が満たされていなければなりません。

- ① *commonlist*に指定した共通ブロックに属する変数及び配列に対して、初期値が設定されていないこと。
- ② 同じ共通ブロックを持つ全てのプログラム単位が、その共通ブロックを指定した翻訳オプション A E (*commonlist*) を指定して翻訳されていること。
- ③ *commonlist*に指定した共通ブロックの大きさがプログラム単位毎に異なる場合、その共通ブロックを含む最初に実行されたプログラム単位での大きさが、以後に実行される同一の共通ブロックを持つプログラム単位での大きさよりも大きいこと。

次に、共通ブロック名付き A E オプションで共通ブロックを拡張域へ割り付ける例をあげます。

#### ● 例 4 コンパイラオプション A E (*commonlist*) による巨大データの使用

共通ブロック COMに属する配列及び変数が拡張域に動的に確保されます。

<pre>@FORTRAN AE(COM) PROGRAM PRGA COMMON /COM/A, B, C DIMENSION C(100, 100) REAL X(500, 500) : CALL SUBA : END</pre>	<pre>@FORTRAN AE(COM) SUBROUTINE SUBA COMMON /COM/AA, BB, CC DIMENSION CC(100, 100) REAL Y(100), Z(100) : END</pre>
---	---

#### ● 例 5 動作が保証されない A E 化の例(1)

PRGAとSUBAを別々に翻訳し、リンクエディタで結合編集したとします。 PRG1で共通ブロック COM1に属する変数に初期値を設定しているため、 COM1は実行可能プログラムの中に確保されます。しかし、SUBAが実行されたときに COM1は実行可能プログラムの外に動的に確保されるので動作は保証されません。

```

@FORTRAN AE(COM1)
PROGRAM PRG1
COMMON /COM1/A, B, C
DIMENSION C(100,100)
:
CALL SUBA
:
END
C
BLOCK DATA BLOCK1
COMMON /COM1/A, B, C
DATA A, B/1.0, 10.0/
:
END

```

```

@FORTRAN AE(COM1)
SUBROUTINE SUBA
COMMON /COM1/AA, BB, CC
DIMENSION CC(100,100)
REAL Y(100), Z(100)
:
END

```

### ● 例 6 動作が保証されない A E 化の例(2)

PRG2を' AE' で翻訳し、SUB2を' NOAE' で翻訳して結合編集した場合、共通ブロックCOM2は、プログラムSUB2が翻訳オプションNOABを指定して翻訳されているため実行可能プログラムの中に静的に確保されます。しかし、プログラムPRG2が実行されると実行可能プログラムの外にCOM2が動的に確保されるため動作は保証されません。

```

@FORTRAN AE(COM2)
PROGRAM PRG2
COMMON /COM2/D, E, F
DIMENSION F(100,100)
:
CALL SUB2
:
END

```

```

@FORTRAN NOAE
SUBROUTINE SUB2
COMMON /COM2/DD, EE, FF
DIMENSION FF(100,100)
:
END

```

#### 5.1.4 A E 化の手続き

原研のカタプロを利用しての翻訳から実行までの処理を解説します。原研におけるA Eオプションのシステム標準値は' AE' となっていますので、ユーザが特に指定する必要はありません。

#### ■ 翻訳・結合編集

以下に翻訳・結合編集のJCL例を示します。

```

TWC(1 4 4 0 5)
//***** TEST *****
//F77EX EXEC FORTEXP, SO='J****.TEST',
// A='ELM(*)'
//LKEX EXEC LKEDCTEX, LM='J****.TEST',
// A=MAP, UNIT=TSSWK
//
```

-----翻訳用カタプロ  
-----結合編集用カタプロ

この例では、'J\*\*\*\*.TEST.FORT77' というソースプログラムを翻訳オプションAEで翻訳・結合編集して 'J\*\*\*\*.TEST.LOAD' というロードモジュールを作成しています。実行時に拡張領域をどのくらい使うのかを知りたい場合には、リンクエディタのオプション 'MAP' を指定します。'MAP' オプションをつけたときに出力される MAP リストを図5.1に示します。

```

LINKAGE EDITOR

OPTIONS SPECIFIED - LIST, LET, MAP
      SIZE1 <DEFAULT VALUE>= 524288
      SIZE2 <DEFAULT VALUE>= 83968
      MAX. LENGTH OF OUTPUT TEXT BLOCK = 18432
      LINECOUNT = 60

** MEMBER NAME ** TEMPNAME NOW ADDED TO LIBRARY.
** OUTPUT LIBRARY NAME ** J****.TEST.LOAD
      AUTHORIZATION CODE IS 0 .
** TTR ** ( 0 / 3 - 91 / 5 )
LOAD MODULE IS AMODE 31 AND RMODE ANY.
**NOW    1 TRACK(S) LEFT UNUSED IN DATA SET COVERING 2 EXTENT(S).
      *** MODULE MAP ***
*** CONTROL SECTION ***
      NAME ORIGIN LENGTH A-R MODE          ENTRY
                                         NAME LOCATION
      MAIN      0     316 ANY/ANY
      JWDCCM#   318     8 ANY/ANY
      ABPR     320   1B56 ANY/ANY
      BIGS     1E78   4DAC ANY/ANY
      CLEA     6C28   46C ANY/ANY
      }

TOTAL LENGTH OF EXTENDED COMMON SECTION 2C50C00 ( 46468096 IN DECIMAL ) ①
ENTRY ADDRESS 0 ①
TOTAL LENGTH 2B8540 ( 2852160 IN DECIMAL ) ②

```

図5.1 MAPリスト

図5.1 のリンクエディタの MAP リストで、①は実行時に拡張領域に動的に確保される大きさを表し、②は実行可能プログラムの大きさを表しています。この例では、実行時に指定する拡張領域の大きさは50MB(拡張領域の指定は10MB単位のため) 基本域には 3MBを指定すれば良いことが分かります。

## ■ 実行

前例で作成したロードモジュール J\*\*\*\*.TEST.LOADを実行するJ C Lの例を以下に示します。

```
TWC(6 2 3 5 3) CLASS(1)
/* T W C E I
//TEST EXEC LMGCEX, LM='J****.TEST', RGN=53M
//SYSIN DD DSN=J****.TEST1, DATA, DISP=SHR
//FT10F001 DD DSN=J****.TEST2, DATA, DISP=SHR
//FT115F001 DD SYSOUT=*
//
```

原研でJCLを記述する場合は、使用する資源量を指定する必要があります。ここでは結合編集時のMAPリスト(図5.1)に出力された情報から、C(基本域)に3(3M)、E(拡張域)に5(50M)を指定しています。

## 5.2 VIO/F入出力機能

FORTRANでプログラムを書いたり、ベクトル化をしたりするときに、プログラムによっては、「演算数が少なくとも、I/O回数が多い」ために処理時間が多くなってしまう場合があります。そのような場合、一時データセットに対してのI/Oを速くしたり、見かけのI/O回数を減らしてくれるのがVIO/F入出力機能です。

### 5.2.1 概要

VIO/F入出力機能は、一時データセットに対する入出力処理を高速に行う機能です。例えば、解析途中の一時保存用ファイルのI/Oを早くしたい時に用いることが出来ます。VIO/F入出力機能をDD文で定義すると、主記憶（またはシステム記憶）の拡張領域にVIO/Fファイルと呼ばれる領域を確保します。このファイルを一時データセットと見なし、入出力処理のための入出力バッファを用いず、主記憶領域間でのデータ転送により高速入出力を行います（図5.2, 図5.3）。

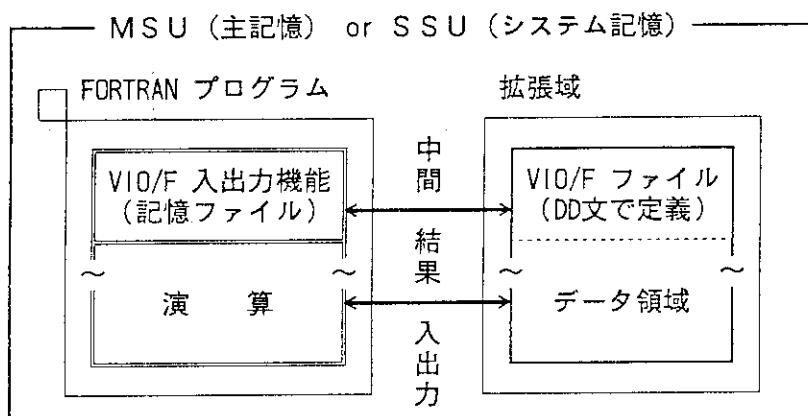


図 5.2 VIO/F入出力機能の概要図

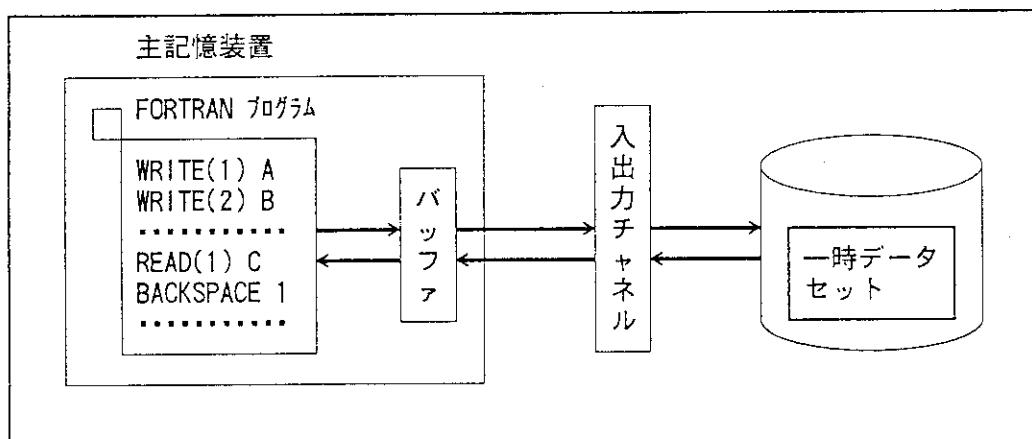


図 5.3 通常の入出力機能の概要図

### 5.2.2 定義方法

VIO/F ファイルは、ジョブ制御文で次の指定を行うことにより定義します。

```
//ddname DD SUBSYS=(VPCS, 'SPACE= { nnnK, } , )  
          nnn[M]
```

※ V10L30以降で VSU が装備されているシステムにおいては、単位(K, M)を省略した場合は、VIO/F ファイルは、可能なら VSU に確保されます。主記憶上に確保したい場合は単位を省略出来ません。原研の場合は SSU も VSU もありませんので、MSU(主記憶) に確保されます。

#### ■ 定義時の注意

例.

```
//SUBSYS DD SUBSYS=(VPCS, 'SIZE=(5M, 10M)' )      ※②  
//FT01F001 DD SUBSYS=(VPCS, 'SPACE=2M' )           ※①  
//FT02F001 DD SUBSYS=(VPCS, 'SPACE=5M' )
```

- ① 一つのジョブステップ内で、この DD 文を任意枚指定出来ます。
- ② スペース量の合計が、ジョブステップで獲得可能な拡張リージョンサイズを超える指定は出来ません。つまり、DD 文の SUBSYS パラメータの SIZE オペランドの拡張リージョンサイズは、SPACE オペランドで指定した VIO/F ファイルのスペース量を加算して指定しなければなりません。スペース量の見積り法は 5.2.4 節を参照して下さい。
- ③ VIO/F ファイルはジョブステップ間でパスすることは出来ません。
- ④ VIO/F ファイルを定義する DD 文の連結は出来ません。
- ⑤ VIO/F ファイルは、拡張リージョンに確保される関係上、リンク時には注意が必要です。24bit モードでのロードモジュール作成や、24bit モードで作成されたライブラリを結合した場合は、動作も結果も保証されません。
- ⑥ VIO/F 入出力機能は、FACOM VP ハードウェア以外では使用出来ません。
- ⑦ VIO/F ファイルのスペース量は、各 FORTRAN 記録の前後に 8 バイトの情報が必要となるので、入出力の件数を考慮して指定する必要があります。

※ ③～⑥は 5.2.5、⑦は 5.2.4 で詳しく述べます。

### 5.2.3 プログラムの記述例

```
DIMENSION AAA(1000), BBB(500), CCC(1000)  
.....  
WRITE(UNIT = NIN) AAA  
WRITE(UNIT = NIN) BBB  
BACKSPACE NIN  
BACKSPACE NIN  
READ(UNIT = NIN) CCC  
.....
```

図 5.4 プログラム記述例

図5.4の例では、FORTRAN記録は次の図5.5のようになります。

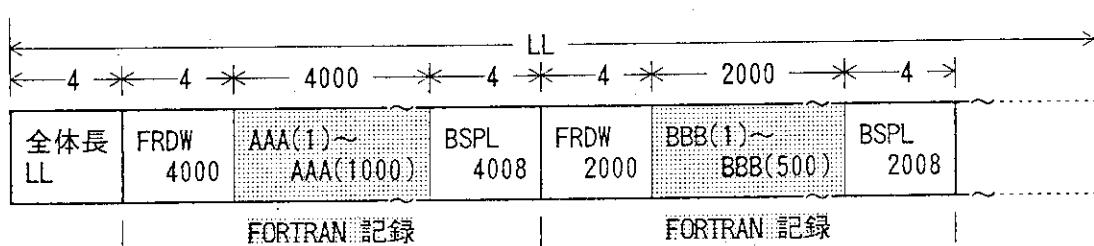


図 5.5 VIO/F の FORTRAN 記録

LL : DD文で指定したVIO/Fファイルのスペース量。

FRDW : FORTRAN記録の長さであり、出力文の並び項目の合計長。

BSPL : BACKSPACE文のための情報であり、直前の記録までの長さ。

**備考** VIO/F ファイルは、レコード形式は、順次入出力文の場合可変長(V) 形式であり、直接入出力文の場合は固定長(F) 形式です。

また、定義時の注意で述べた通り、各FORTRAN 記録の前後に 8バイトの情報が必要となりますので、入出力の件数を考慮してスペース量を指定することが必要になります。

### ■ 使用上の注意

- ① ジョブステップ内に閉じた一時データセットに対する入出力処理なので、プログラムの VIO/Fファイルに対する最初の動作が READ 文であってはいけません。
- ② CLOSE 文を実行した後は、事前の結合が解除となるので、どの様な入出力文も実行出来ません。（内部的に CLOSE文が実行された効果となる場合も同様）
- ③ 一時データセット扱いなので、OPEN 文の FILE 指定子は指定しても意味を持ちません。
- ④ DD 文で定義した VIO/Fファイルのスペース量を超えて出力することは出来ません。

### 5.2.4 VIO/Fファイルのスペースの見積り方法

#### ■ 手順

- (1) VIO/F 機能を使うファイルを最初 TSSWKに割り当ててデータセットをカタログします。
- (2) コマンド MSSF または LISTD(SP オプション必須) を用いてそのデータセットの大きさを確かめます。
- (3) 原研の場合は、1 CYL = 15 TRK, 1 TRK = 47 KB ですから、SUBSYS パラメータの SPACEオペランドにどれだけの容量を確保すればいいか簡単に計算出来ます。

■ 見積りの具体例

- (1) FT10F001 が VIO/Fを使いたいデータセットの場合、図5.6 の様に、TSSWK に一時的にカタログ指定して実行して下さい。

```
TWC(Tw Ww Cw Ew Iw) CLASS(?)
//RUN      EXEC GO
//FT05F001 DD DSN=J0000.TEST, INDATA, LABEL=(,,,IN), DISP=SHR
//FT06F001 DD SYSOUT=*
//FT10F001 DD DSN=J0000.TEST, WORK, DISP=(NEW,CATLG,DELETE),
//              SPACE=(TRK,(100,100),RESE), UNIT=TSSWK,
//              DCB=(RECFM=FB,LRECL=100,BLKSIZE=3000,DSORG=PS)
//FT20F001 DD DSN=J0000.TEST, OUTDATA, DISP=SHR
//
```

図 5.6 実行JCL の例

- (2) 次に MSSF, LISTD, LISTVOL 等のコマンドでスペース量を確認して下さい。

LISTD 'J0000.TEST.WORK' SP
J0000.TEST.WORK
--RECFM-LRECL-BLKSIZE-DSORG
FB 100 3000 PS
--VOLUMES--
WKB1A4
---TOTAL-----EXT----SECONDARY---EMPTY
(TRK) (NO.) (TRK) (TRK)
90 1 100 10

※ LISTD を用いた場合、  
SPACE 量は、  
TOTAL - EMPTY  
で求められます。  
この場合は、  
 $90 - 10 = 80$  (TRK)  
となります。

図 5.7 LISTD を用いた例

今、'J0000.TEST.WORK' のスペース使用量が図5.7 のように 80 TRK であるとします。

1TRK ≈ 47 KBですから、

$$47 \text{ KB}/1\text{TRK} \times 80 \text{ TRK} \approx 3,800 \text{ KB}$$

※参考

1 CYL = 15 TRK
1 TRK = 47,476 Byte

です。また、5.2.2 で述べたように、FORTRAN 記録の前後に 4バイトずつ情報が入りますから、その分を考慮してSPACE オペランドには 4M を指定します。

(3) VIO/F 機能を使った JCL例

- (2)の結果をもとに VIO/F機能を使ってみました。 (図5.8 )

```

TWC(Tv Wv Cv Ev Iv) CLASS(?)
//RUN EXEC GO
//FT05F001 DD DSN='J@@@@.TEST.INDATA',LABEL=(,,IN),DISP=SHR
//FT06F001 DD SYSOUT=*
//FT10F001 DD SUBSYS=(VPCS, 'SPACE=4M')
//FT20F001 DD DSN=J@@@@.TEST.OUTDATA,DISP=SHR
//

```

図 5.8 VIO/F を使った実行JCL の例

但し、ジョブ終了時の作業データセットの大きさが、ジョブ実行中に確保する大きさよりも小さくなる場合はこの方法は勧められません。

#### ■ 原研におけるシステム下で使用する場合の注意点

VIO/F ファイルは必ず拡張域に展開されますので、TWCEI の E 値（拡張領域のサイズ）を SUBSYS パラメータの SPACE オペランドの値以上に設定する必要があります。 I 値（入出力回数）は VIO/F を使った分だけ小さくなります。

今、TSSWK を使用した時の TWCEI の値をそれぞれ、 T<sub>w</sub> W<sub>w</sub> C<sub>w</sub> E<sub>w</sub> I<sub>w</sub> 、 VIO/F ファイルを使用した時の TWCEI の値をそれぞれ、 T<sub>v</sub> W<sub>v</sub> C<sub>v</sub> E<sub>v</sub> I<sub>v</sub> とします。 TSSWK 使用時の I/O 回数を N<sub>v</sub> 、 VIO/F 使用時の I/O 回数を N<sub>w</sub> 回、 SPACE オペランドの値を S とすると次式が成り立ちます。

$$\begin{aligned}
 T_v &= T_w \\
 W_v &= W_w \\
 C_v &= C_w \\
 E_v &= E_w + S \\
 I_v &\leftrightarrow N_v \text{ に対応した値} \\
 I_w &\leftrightarrow N_w \text{ に対応した値} \quad \text{※但し } I_v < I_w
 \end{aligned}$$

原研の情報システムセンターで用意しているカタログを使用する場合は、ユーザが SUBSYS DD 文を定義しなくとも、TWCEI の値がカタログの中に定義されている SUBSYS パラメータの SIZE オペランド（下線部 ↓①, ↓②）に自動設定されます。

↓① ↓②

```
//SUBSYS DD SUBSYS=(VPCS, 'SIZE=(0000K,000M)')
```

原研のカタログを使わずに JCL を定義する場合は、必ずこの SUBSYS DD 文を定義して下さい。この時は上式の ↓①、 ↓② の値は、

↓① : USER が使用する基本リージョンサイズ  
 ↓② : VIO/F を定義した SUBSYS パラメータの SPACE オペランドの値の和 + USER が使用する拡張リージョンサイズ

となります。

### 5.2.5 その他の留意点

(1) ジョブをスカラで流す場合

VIO/F は VP 上においてのみスカラでもベクトルでも実行出来ます。

(2) コンパイル時、リンク時にも使用出来るか

コンパイル時、リンク時に例えば SYSUT1, SYSUT2 等で作業用データセットを定義します。これらのデータセットも一時データセットなのでもしも使えると、I/O 回数やそれに伴うCPU タイムも短縮できそうです。しかし、結果は 013-C8 (オープン時のエラー) で ABEND します。

要するに、VIO/F は FORTRANで有効な手段ですので、コンパイラが FORTRANで書かれていて、しかも5.2.3 にあるような使用上の注意をすべてパスしたような作りをしてあるなら使えるでしょう。しかし、実際にはアセンブラーが主体です。コンパイル時の作業データセットに VIO/Fで定義しても、VIO/F を動作させるための FORTRANのライブラリがオープンされません。

(3) 異なるジョブステップで同一のVIO/F ファイルを使用出来ない

原研の場合、VIO/F ファイルは MSUの拡張域に展開されますので、1つの EXEC 文が終了すると、その VIO/F ファイルにはファイル終了記録が出され、開放されてしまいます。そのため、初めの GO ステップで、計算途中の結果を格納する TEMP ファイルを作り、次の GO ステップでそのファイルを読み出したい場合は使用出来ません。

※ システムに SSUか VSUが装備されている場合、FORTRAN77EX/VP に限りパス出来ます。

(4) 24bitアドレシングモードで作られたライブラリのリンク

プログラム自身が基本域だけで動作するような場合でも、VIO/F ファイルは必ず拡張域に展開されます。そのため 24bitアドレシングモードで作られたライブラリのリンクは出来ません。ロードモジュールも 31bitアドレシングモードで動作出来るようになる必要があります (E-31 or E-ANY のみ有効)。24bitアドレシングモードの場合は、動作も結果も保証されませんので注意して下さい。

FORTRAN77/VP および FORTRAN77EX/VP では、リンク時にオプションの SECTIONや AMODE, RMODE 等の指定をしなくても 31bitアドレシングモードで作られます。

## 5.2.6 実際のコードでの応用結果

## — 热流体解析コード STREAM (クレイドル社開発)での応用結果と考察 —

STREAM の場合 FT11F001 で解析途中の結果を一時データセットに書き込む処理をしています。そこでこの11番のファイルに VIO/F機能を使ってみました。

外部記憶装置(WK10)にアサインした場合と VIO/Fにアサインした場合では、 GO ステップにおいて表5.1のような結果の違いが見られました。

表 5.1 STREAM2.6 の実行結果

使用マシン：那珂研究所 VP2600/10		実行プログラム STREAM V2.6 (class1)
	WK10	VIO/F
CPU 時間	6.66 SEC	6.39 SEC
ELAPSED 時間	49.76 SEC	18.69 SEC
I/O アクセス回数	1300 TIMES	46 TIMES
基本域使用量	296 KB	312 KB
拡張域使用量	10980 KB	23268 KB
テスト時の環境	CLASS-2, 6	CLASS-2, 6
* 本テスト時に同時に実行されてた他ジョブ	※ テスト時の環境（他ジョブの混み具合等）は両者とも同等である。	

- ELAPSED 時間 .... テスト時の環境（他ジョブの混み具合等）は両者とも同等であるので、表5.1の結果に大きな差は無いと考えていいでしょう。
- CPU 時間 ..... VIO/F 入出力機能の利点は I/Oが格段に減ることです。CPU 時間ににおける I/O関連に費やす時間は誤差の範囲程ですが、I/O が発生すると、CPU は入出力監視プログラムが作動します。そのため、入出力監視プログラムが作動させるための時間×作動回数 分だけ短縮されます。
- I/O 回数 ..... 1300 回が 46 回まで少なくなりました。  
I/O 回数で、上位のクラスでしか実行出来ないプログラムでも、下位クラスで実行可能となる場合も考えられます。※
- 基本域使用量と拡張域使用量  
..... VIO/Fが拡張域に展開されていることが分かります。

※ データによっては、I/O 回数の関係で CLASS-6でしか実行出来ないジョブが CLASS-1 で実行可能になったケースが実際にありました。

## 6. 移行時の問題点

原研内で使用頻度の高い原子力コード 17 本について、F77EX/VPでの翻訳～実行テストを行いました。 テストした17本のコードは以下の通りです。

AEOLUS	BERMUDA	CITATION	DOT35	DSMC	FPGS
ISOFLOW	PHENIX	RELAP5	SLWALF	SONATINA	SPIN
SRAC	STREAM*	TWOTRAN	VDIRECT	VENUS	

\* STREAMはクレイドル社製の汎用熱流体解析コードです。

これらのコードをF77EX/VPへ移行する際に生じた問題点を具体的にあげていきます

### 6.1 非互換項目による問題

#### (1) 実行時のエラー識別番号の違い

[該当コード] RELAP5(MOD2.5 以降), SCDA P

##### [内容]

F77EX システムと F77システムとでは、エラー識別番号が対応していない。従って、ERRSETルーチンを使用しているプログラムをそのまま F77EXシステムで翻訳～実行した場合、該当するエラーが生じたときにエラーの回避が出来ない。

```
CALL ERRSET(213, 256, -1, 1, 1)
CALL ERRSET(202, 256, -1, 1, 1)
```

##### [検出方法]

エラーメッセージ JWB0305I-W (エラー識別番号を持つサブルーチンが実行された) が output される。

##### [対処]

ERRSETルーチンに渡すエラー識別番号を修正する。 F77システムとのエラー識別番号の対応は付録A「非互換項目資料」を参照のこと。

```
CALL ERRSET(-13, 256, -1, 1, 1)
CALL ERRSET(-152, 256, -1, 1, 1)
```

## (2) 同一装置参照番号に対する書式付き出力と書式無し出力の混在

[該当コード] I S O F L O W

## [内容]

サブルーチンSAVESZの中に同一装置参照番号に対する書式付き出力と書式無し出力がある。F77EXシステムでは、同一装置参照番号に対する書式付き出力と書式無し出力の混在はエラーとなる。

```
:
  WRITE(LPLT,'(A5)') 'VER20'
  WRITE(LPLT,'(A64)') TITLE
  WRITE(LPLT) NX, NY, NXP1, NYP1, NXP2, NYP2
:
```

## [検出方法]

エラーメッセージ JWE0113I-E が出力される。

## [対処]

書式無し出力を書式付き出力に変更する。

```
:
  WRITE(LPLT,'(A5)') 'VER20'
  WRITE(LPLT,'(A64)') TITLE
  WRITE(LPLT,*) NX, NY, NXP1, NYP1, NXP2, NYP2
:
```

あるいは、実行可能プログラムオプション RUN77を指定する。

## (3) スプールデータセットに対するREWIND

[該当コード] S L W A L F

## [内容]

SLWALFではJSSLのサブルーチンDTLISTを使用している。DTLISTはインプットカード（装置参照番号：5）のベタ打ちリストをとる機能を持っている。そのため、DTLIST内でインプットカードを読み込み、書き出しを行った後、REWINDする処理を行っている。しかし、F77EXではスプールデータセット（JCL中に記述された入力データ）に対してREWIND文を実行することは出来ず、エラーとなる。

```
:
//FT05F001 DD *
      入力データ
/*
:
```

[検出方法]

エラーメッセージ JWE0042I-E が出力される。

[対処]

スプールデータセットを通常のデータセットに移す。

```
//FT05F001 DD DSN=J****,SLWALF,INPUT,DISP=SHR
```

あるいは、実行可能プログラムオプション RUN77を指定する。

## 6.2 ロードモジュールの増大による問題

F77EXシステムでは F77システムに比べて実行性能が強化されている反面、ロードモジュールの大きさは増大します。ループアンローリング、外部手続きのインライン展開等の最適化が実施された場合では、ロードモジュールの増大は顕著になります。これらの理由により、F77システムでは基本域(8M以下)で動作していたプログラムが、F77EXシステムで翻訳～実行した場合には8Mを超してしまうことがあります。このような場合はA E化によって対処しますが、コードの中には様々な事情により、翻訳時にA Eオプションを指定するだけではA E化出来ない場合があります。

(1) オーバレイ構造プログラムのロードモジュールが基本領域を超える。

[該当コード] R E L A P 5

[内容]

F77EXシステムでロードモジュールを作成すると基本領域(8M)を超ってしまうが、オーバレイ構造のためにAE化が出来ない。

[対処]

オーバレイ構造を外してAE化する。

- ① アセンブラーソースのAE化
- ② DATA文による初期値設定の廃止
- ③ LINK時におけるオーバレイ構造の入力データを廃止し、COMMONブロックの順序付けを行う。

- (2) 24bit アドレッシングモードのライブラリを結合しているプログラムのロードモジュールが使用可能な基本領域の大きさを超える。

[該当コード] SLWALF, DSMC, FPGS

[内容]

24bit アドレッシングモードのライブラリ (SSL, グラフィック等) を結合すると原則的にはAE化が出来ない。

[対処]

AE 指定で翻訳し、結合時にリンクエージオプション' AMODE(31)' を指定してロードモジュールを作成する。（但し、場合によっては、この方法でAE化出来ない場合もある。）

```
//LINK EXEC LKEDEX, A='NOMAP, AMODE(31)'
```

## 第2編 VPプログラミング

近年、スーパーコンピュータの需要が顕著に伸びています。原研においてもスーパーコンピュータ VP2600/10 が導入され、盛んに利用されています。VP2600 はパイプライン方式のアーキテクチャを採用しており、逐次処理を基本とする従来の汎用機とは構造を異にしています。

VP2600 は今までFORTRAN77 コンパイラを使用していましたが、今年より原研にバージョンアップされたコンパイラFORTRAN77EX が導入されます。VP2600 では、通常のFORTRAN プログラムを通常の手順で実行することができますが、計算機の性能を十分に引き出すためにはハードウェア、及びFORTRAN77EX の特徴を生かした効率良いプログラムを作成する必要があります。

本編ではこれらのことと踏まえて、VP2600 のアーキテクチャ、ベクトル処理の概要、及びベクトル化基本技法などについて解説します。

## 1. VP2600/10の概要

### 1.1 ベクトル処理

VP2600 はベクトル処理計算機です。ベクトル処理とはDOループ内の実行文の順序を変更して、複数のスカラ命令で処理していたデータを単一のベクトル命令で複数のデータを処理することです。例えば、次のようなプログラムがあった場合のスカラ処理、ベクトル処理の違いを図1.1 に示します。

```
DO 10 I = 1, 100
    A(I) = B(I) + C(I)
    D(I) = E(I) * F(I)
10 CONTINUE
```

#### スカラ処理

```
A(1) = B(1) + C(1)
D(1) = E(1) * F(1)
A(2) = B(2) + C(2)
D(2) = E(2) * F(2)
A(3) = B(3) + C(3)
D(3) = E(3) * F(3)
:
A(99) = B(99) + C(99)
D(99) = E(99) * F(99)
A(100) = B(100) + C(100)
D(100) = E(100) * F(100)
```

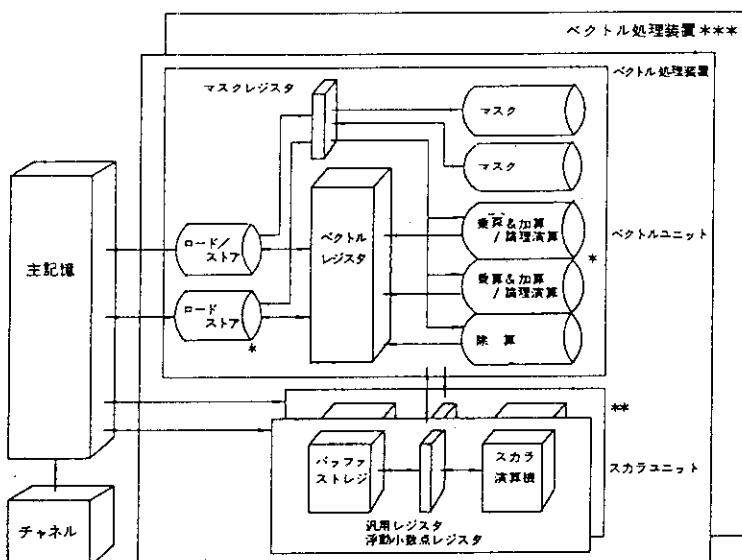
#### ベクトル処理

```
Ai = Bi + Ci (i=1,...100)
Di = Ei * Fi (i=1,...100)
```

図1.1 スカラ命令とベクトル命令

## 1.2 VP2600/10の構成と特徴

VP2600/10のベクトル処理装置は図1.2 のようになっています。ただし、原研のシステムには、システム記憶はありません。



\*\* : 原研では1台

\*\*\*: 原研では1台

図1.2 VP2600の構成

ベクトル計算機の特徴であるベクトルパイプラインは、全部で7本あります。 それらは、以下のものから構成されます。

ロード／ストア	× 2本
マスク	× 2本
乗算＆加算／論理	× 2本
除算	× 1本

この7本のうち6本が並列動作可能であり、乗算＆加算／論理パイプラインが2本同時に動作した場合、最大ベクトル性能 5 G F L O P S が得られます。

VP2600の特徴として、(1)多数のベクトル命令、(2)大容量ベクトルレジスタ、(3)条件付きベクトル演算があります。(1)は、ベクトル命令が132種(VP100では83種)用意されており、多くの処理をベクトル処理に置き換えられます。(2)は128KB(同32KB)の容量を持っており、主記憶とのロード／ストアの高速化、及び演算の中間結果を保持することにより、主記憶とのデータ転送を少なくします。(3)はベクトルレジスタに対応してマスクレジスタがあり、このマスクレジスタの値によってベクトルデータの演算を制御することができます。これによりベクトル化できる範囲が広がります。

### 1.3 ベクトル処理装置での命令実行について

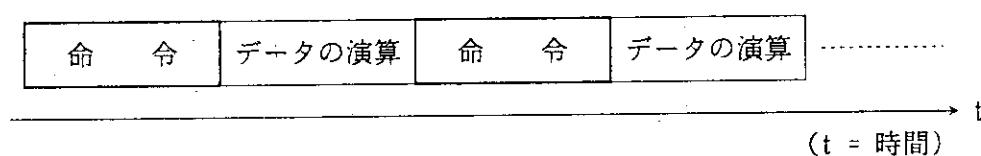
プログラムが実行されるときの、命令の動きを説明します。命令が発生すると、最初にスカラユニットがどのような命令かを読み取ります。次にその命令を解読し、スカラ命令ならば、スカラユニットで実行します。ベクトル命令であれば、その命令をベクトルユニットに渡します。

ベクトル命令を受け取ったベクトルユニットは、ロード／ストアパイプライン、演算パイプラインを起動してベクトル処理を行います。ベクトルユニットが起動中、スカラユニットは、可能なかぎりスカラ処理を実行し続けます。

ベクトルユニットはベクトル処理が終わると、スカラユニットへ終了通知をします。そして、次のベクトル命令を待ちます。ベクトル命令が連続している時は、スカラユニットからベクトルユニットへ連続してベクトル命令を渡します。

### 1.4 パイプライン方式について

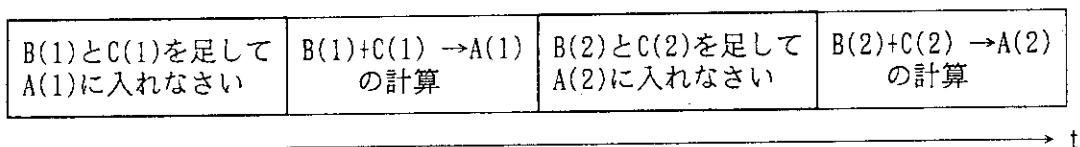
VPの特色であるパイプライン方式について順を追って説明します。ごく普通の計算機(逐次式のスカラ計算機)は、次のように処理をします。



例えば

$D0\ 10\ I=1, 100$   
  $A(I)=B(I)+C(I)$

のときは、以下のように処理されます。

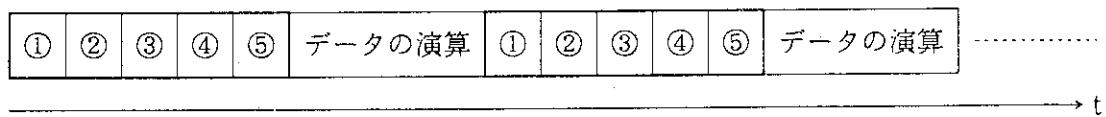


#### 1.4.1 命令のパイプライン方式

1命令は、次の5段階のステップ(これらステップをフェーズと呼んでいます)から成り立っています。

- フェーズ1 命令アドレス計算
- フェーズ2 命令語の読み出し
- フェーズ3 命令の解読
- フェーズ4 アドレス計算
- フェーズ5 オペランドの読み出し

したがって、スカラ計算機は、以下の様に動いているわけです。



この命令の 5 フェーズをパイプライン方式で実行すると図1.3 のようになります。

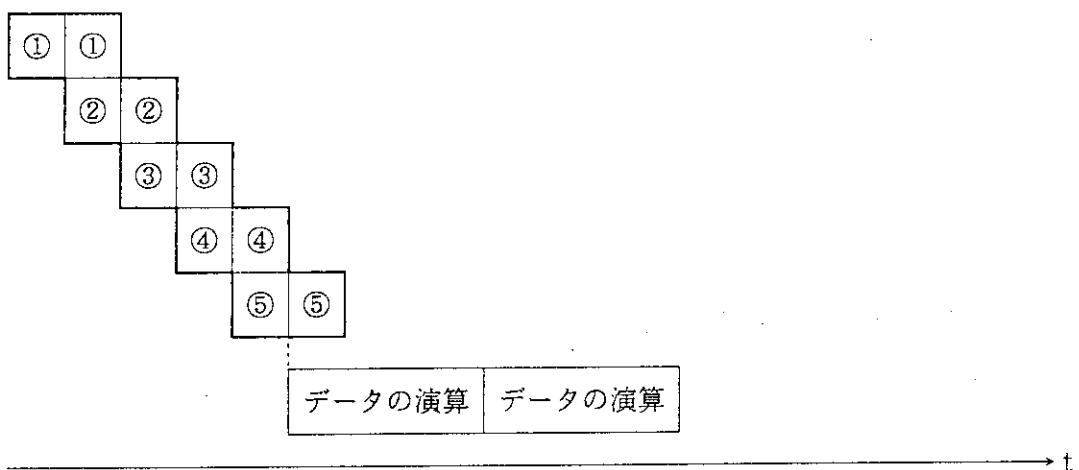
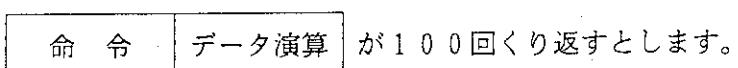
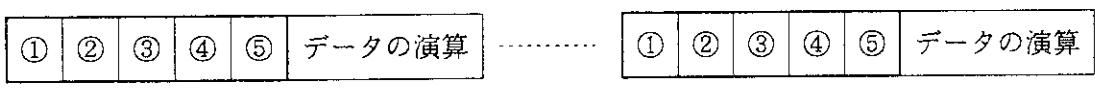


図1.3 命令パイプラインの実行イメージ

命令が並列に発生することが分かると思います。次に命令の処理時間について注目してみましょう。



命令パイプラインを用いていない計算機の場合は、



のように実行が行われ、命令の 1 フェーズの C P U タイムを仮に  $1\tau$  とすると、全命令処理時間は  $5\tau \times 100 = 500\tau$  です。

パイプライン方式の場合は、図1.4 のようになり、全命令処理時間は  $104\tau$  となります。

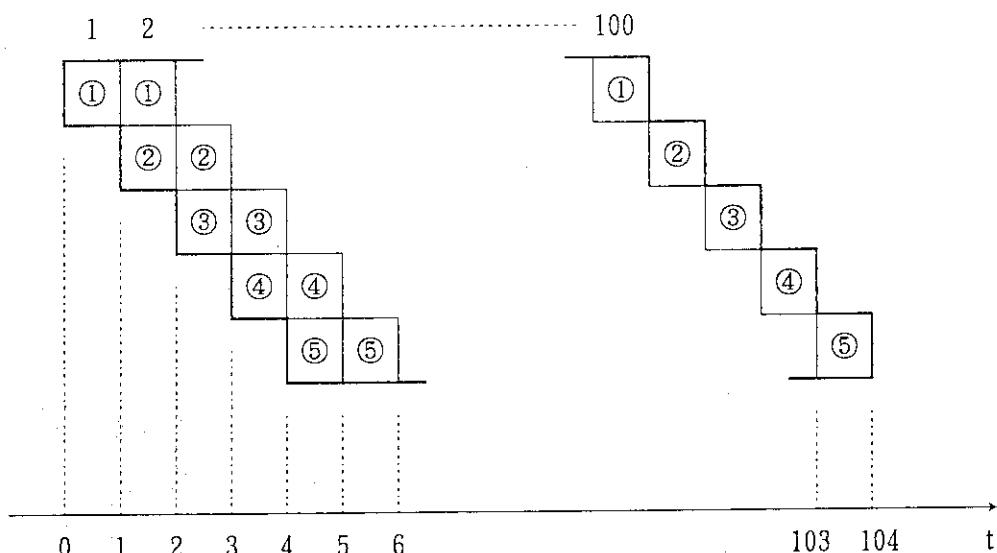


図1.4 命令パイプラインの動き

この方式は、すでに富士通(株)のMシリーズ（M780など）で実現しています。

#### 1.4.2 演算のパイプライン方式

命令のパイプライン化に加えて、データ演算をパイプライン化したものがベクトル計算機です。

加算の場合には、1回の演算は以下の4つのフェーズから成り立ちます。

- C ----- 比較 (Compare Exponent)
- S ----- シフト (Shift)
- A ----- 加算 (Add)
- N ----- 正規化 (Normalize)

逐次式計算機の場合は図1.5、あるいは図1.6のように実行が行われます。

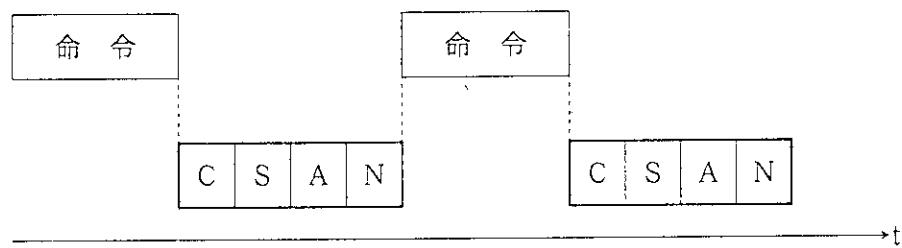


図1.5 逐次式計算機の実行イメージ

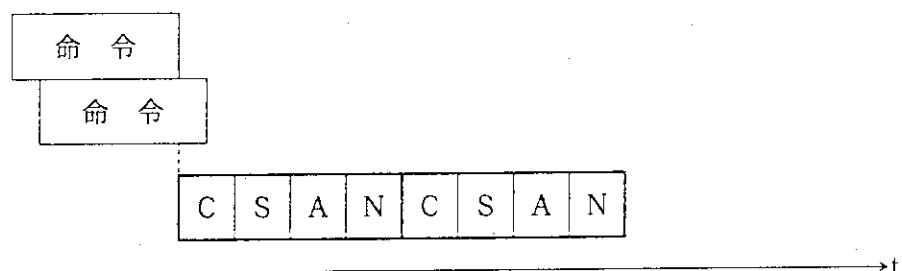


図1.6 命令パイプラインを採用した逐次式計算機の実行イメージ

演算パイプライン方式の場合には図1.7 のように実行されます。

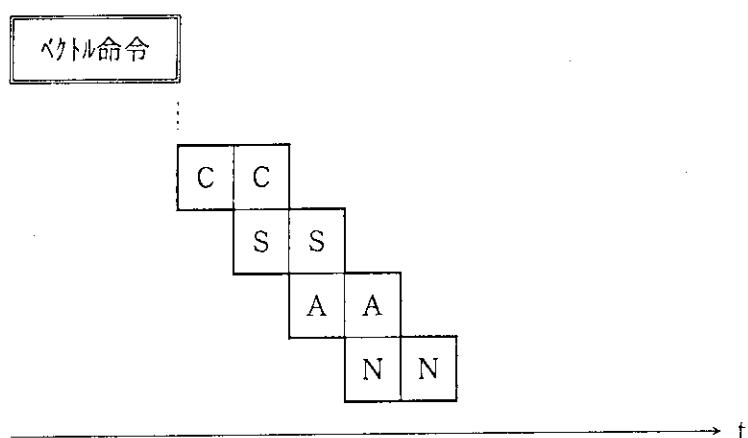


図1.7 演算パイプラインの実行

1フェーズのC P Uタイムを仮に $1\tau$ として、100回実行するとなると、全実行時間は

$$\begin{array}{ll} \text{逐次処理} & \cdots 4\tau \times 100 = 400\tau \\ \text{パイプライン} & \cdots 103\tau \end{array}$$

となります。

1.4.3 回帰計算

以下のプログラムがあるとします。

```
DO 10 I=1,100
    A(I) = B(I) + C(I)
    C(I) = A(I) * D(I)
10 CONTINUE
```

これをスカラ計算で実行すると

$$\begin{aligned} & \left| \begin{array}{l} A(1) = B(1) + C(1) \\ C(1) = A(1) * D(1) \\ A(2) = B(2) + C(2) \\ C(2) = A(2) * D(2) \\ A(3) = B(3) + C(3) \\ C(3) = A(3) * D(3) \\ \vdots \\ A(100) = B(100) + C(100) \\ C(100) = A(100) * D(100) \end{array} \right. \\ \downarrow & \end{aligned}$$

となります。ベクトル計算機は、

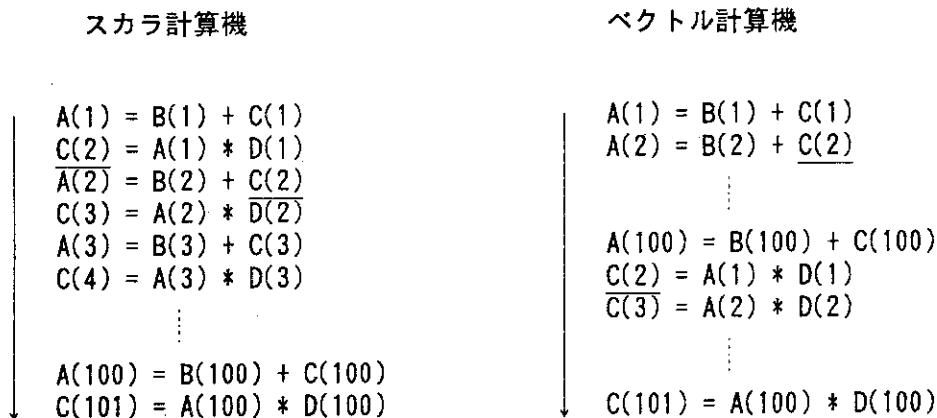
$$\begin{aligned} & \left| \begin{array}{l} A(1) = B(1) + C(1) \\ A(2) = B(2) + C(2) \\ \vdots \\ A(100) = B(100) + C(100) \end{array} \right. \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} 1 \text{ ベクトル処理} \\ & \left| \begin{array}{l} C(1) = A(1) * D(1) \\ C(2) = A(2) * D(2) \\ \vdots \\ C(100) = A(100) * D(100) \end{array} \right. \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} 1 \text{ ベクトル処理} \\ \downarrow & \end{aligned}$$

の順で計算が行われます。この場合は、ベクトル計算を行っても結果は変わりません。

次に、以下のプログラムを考えます。

```
DO 10 I=1,100
    A(I) = B(I) + C(I)
    C(I+1) = A(I) * D(I)
10 CONTINUE
```

これをスカラ、及びベクトル計算機で強制的に実行すると



となります。ここで $C(2)$ について注目してみましょう。

スカラ計算機ではまず、 $C(2)$ を計算します。次に今計算された $C(2)$ を用いて $A(2)$ を計算します。ベクトル計算機で強制的にベクトル計算すると、まず古い値の $C(2)$ を用いて $A(2)$ を計算しています。次に $C(2)$ 自身を計算します。したがって $A(2)$ の値の計算結果は、スカラ実行の計算結果と異なってしまいます。このような実行手順を持つループを回帰計算ループと言います。

#### 1.4.4 パイプラインの動き

VP-2600/10にはパイプラインが7本あります。これらは6本まで並列に作動します。例えば、

```
□ DO 10 I = 1, 100
    A(I) = B(I) + C(I) * Z + F(I)
```

を実行する場合を考えてみましょう。パイプラインの動きは図1.8 のようになります。全処理時間は  $211\tau$  となります（1つのデータが、パイプラインを通過する時間を  $4\tau$  とする）。

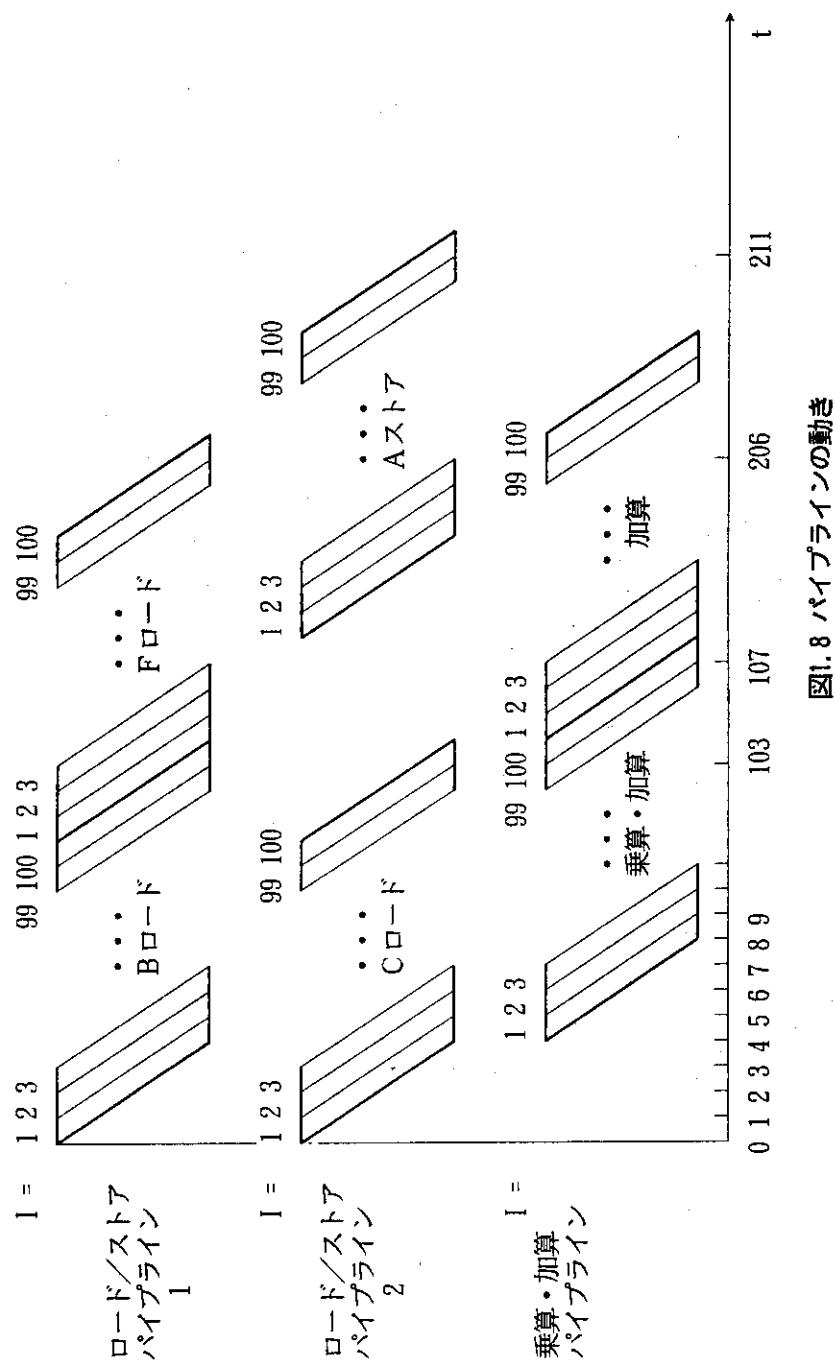


図1.8 パイプラインの動き

## 1.5 高速性を達成するための技術

ベクトル計算機ではベクトル演算の他に少しでも計算を高速に行うため、いくつかの技術を採用しています。

### 1.5.1 実メモリ方式

通常のスカラ計算機は仮想記憶方式を採用していますが、VPでは実メモリ方式を採用しています。

#### (1) 仮想記憶方式

実際の主記憶より大きいプログラムを実行させることができる技術です。なぜ主記憶より大きいプログラムが実行ができるかというと、プログラムをいくつかに分割（分割された1つの単位をページと呼ぶ）し、必要なページだけを主記憶にのせて、残りを外部直接アクセス装置に移動させる。必要に応じて、そのページを呼び出し、不必要的ページは外部に移動します（これらの動作をページングという）。このように、必要なページだけを主記憶に置くことで記憶容量の節約ができます。

#### (2) 実メモリ

VPは、主記憶メモリが大きく（原研では基本領域8MB、拡張領域512MB）、プログラムの全てを主記憶上に展開することができます。また、ページング動作がないので経過時間は仮想記憶方式より短くなります。

### 1.5.2 ベクトルレジスタ

ベクトルレジスタは、これからベクトル計算を行うのに使用するベクトルデータ、及び計算途中での中間結果を保持する機能を持っています。この機能により、主記憶メモリとの間でのデータのやり取りが少なくなり、データ転送にかかる時間が少なくて済みます。このデータのやり取りを最小にし、処理の高速化をはかるために、VPは再構成可能なベクトルレジスタを備えています。ベクトルレジスタの基本構成を図1.9、そのレジスタ個数と1レジスタあたりの要素数の組み合わせの一覧を表1.1に示します。

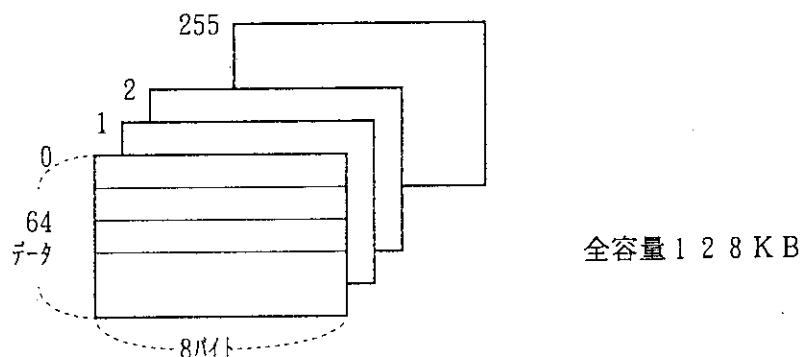


図1.9 ベクトルレジスタの基本構成

表1.1 ベクトルレジスタの組み合わせ一覧

バイト	ベクトル要素数	ベクトルレジスタ個数	連結数
8バイト	64	256	1
8バイト	128	128	2
8バイト	256	64	4
8バイト	512	32	8
8バイト	1024	16	16
8バイト	2048	8	32

ベクトル要素数 (=ベクトル長) によって、連結するレジスタの個数が動的に変わります。例えば次のプログラムを考えましょう。

```
DO 10 I = 1, 100
    A(I) = B(I) + C(I)
10 CONTINUE
```

このDOループは、実際の処理においては再構成されたベクトルレジスタにそったベクトル長を持つDOループに分割されます。このプログラムはループ回転数が 100なので表1.1 より、ベクトル長64のDOループとベクトル長36のDOループに分割されます。実行イメージは次のようにになります。

```
LVR=64
DO 10 K =1, 100, LVR
    DO 10 I = K, MIN(K+LVR-1, 100)
        A(I) = B(I) + C(I)
    10 CONTINUE
```

すなわち

$A_i = B_i + C_i \quad i = 1, 64$	← 1 ベクトル処理
$A_i = B_i + C_i \quad i = 65, 100$	← 1 ベクトル処理

のように処理されます。

しかし実際には、DOループの繰り返し数と、DOループ内の変数、演算の数によってコンパイラが最適なレジスタ構成を選択するので、ユーザが意識する必要はありません。

## (3) リスト・ベクトル・アクセス命令

ベクトルデータのうち、マスク部分が真の要素だけを示すリストベクトルを作成し、これに基づいて演算を行います（図 1.12）。

実行時間は、マスク対象のベクトル要素分の演算時間とリストベクトル生成操作分の時間を加えたものとなります。

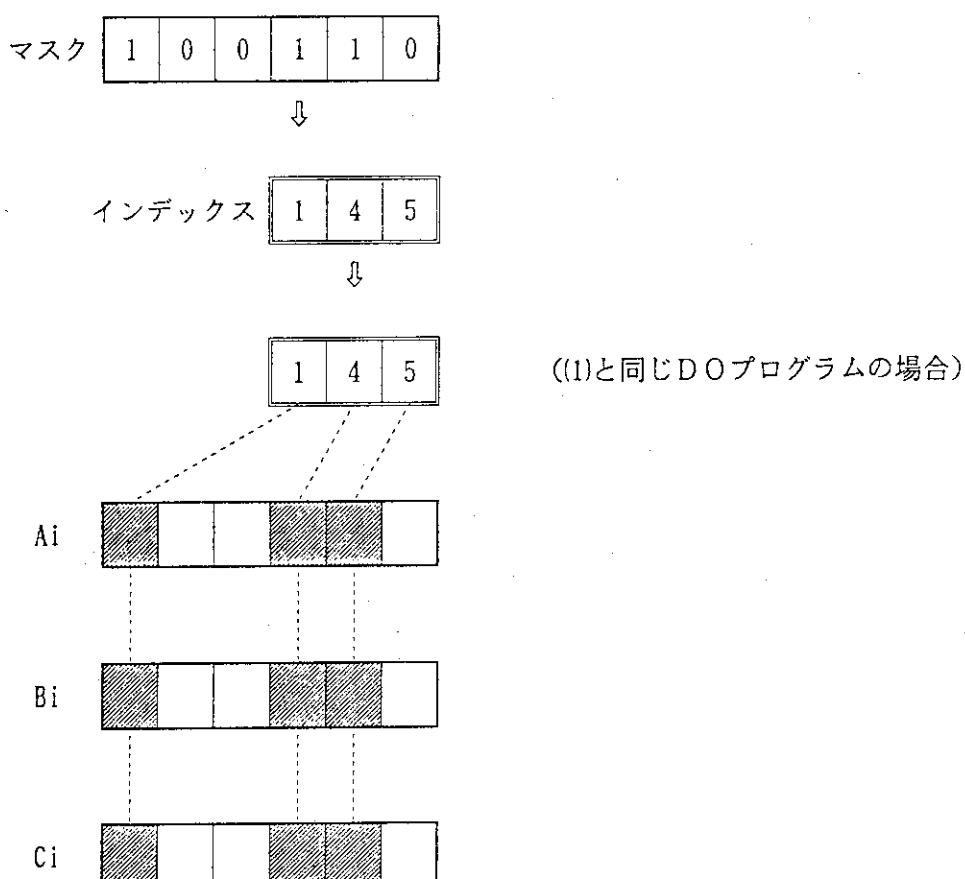


図 1.12 リストベクトルによる方法

これらの手法は、コンパイラがそのプログラムに最適なものを選択する。選択する基準は、以下の通りです。

- |       |       |                       |
|-------|-------|-----------------------|
| マスク   | ----- | IF文の真率が高い場合           |
| 収集／拡散 | ----- | IF文の真率が低く、組み込み関数を含む場合 |
| リスト   | ----- | IF文の真率が低い場合           |

ここで、論理IF文、ブロックIF文、及び ELSE IF文の真率とは、IF文の条件式が評価された回数に対して、真と評価された回数の比率のことです。 算術IF文の真率とは、算術式が演算された回数に対して、その値が負、ゼロまたは正になる回数の比率のことです。

一般に、0 ~ 50% は真率が低いとみなされ、50~100%は真率が高いとみなされます。

## 2. ベクトル化の基本

ここでは第1章のV P 2 6 0 0 のアーキテクチャを基礎とし、ベクトル化チューニングに必要な知識について述べることにします。

### 2.1 パイプライン演算機の性能

$n$  個のデータを処理する場合、スカラ処理時間  $t_s$  、及びパイプライン処理時間  $t_v$  は

$$t_s = \tau_s \cdot n$$

$$t_v = t_o + \tau_v \cdot n$$

と表すことができます。ここで

$\tau_s$  : スカラ処理による一要素処理時間

$t_o$  : パイプライン立上り時間

$\tau_v$  : ベクトル処理による一要素処理時間

となります。ベクトル処理の場合  $t_o$  があるので、ベクトル長が短いと  $t_v$  中の  $t_o$  の比率が大きくなり、倍率は上がらないことになります。

スカラ・ベクトル処理性能比  $p$  を

$$p = t_s / t_v = \tau_s \cdot n / (t_o + \tau_v \cdot n)$$

と定義します。一般に  $\tau_v \ll \tau_s \sim t_o$  の関係があります。従って

ベクトル長が長くなると ( $n \rightarrow \infty$ )  $p \rightarrow \tau_s / \tau_v$

ベクトル長が短いと ( $n \approx 1$ )  $p \rightarrow \tau_s \cdot n / t_o$

となります。ベクトル長  $n$  が大きい程、パイプラインによる処理効率が上がります。

## (2) 収集／拡散命令

ベクトルデータのうち、マスク部分が真になる要素だけをあらかじめ収集しておいて、その収集されたデータだけでベクトル演算をします。演算終了後は、その計算結果を元のベクトルデータに拡散します。 計算時間は、マスク対象のベクトル要素分の演算と収集／拡散の補助操作で済みます(図 1.11)。

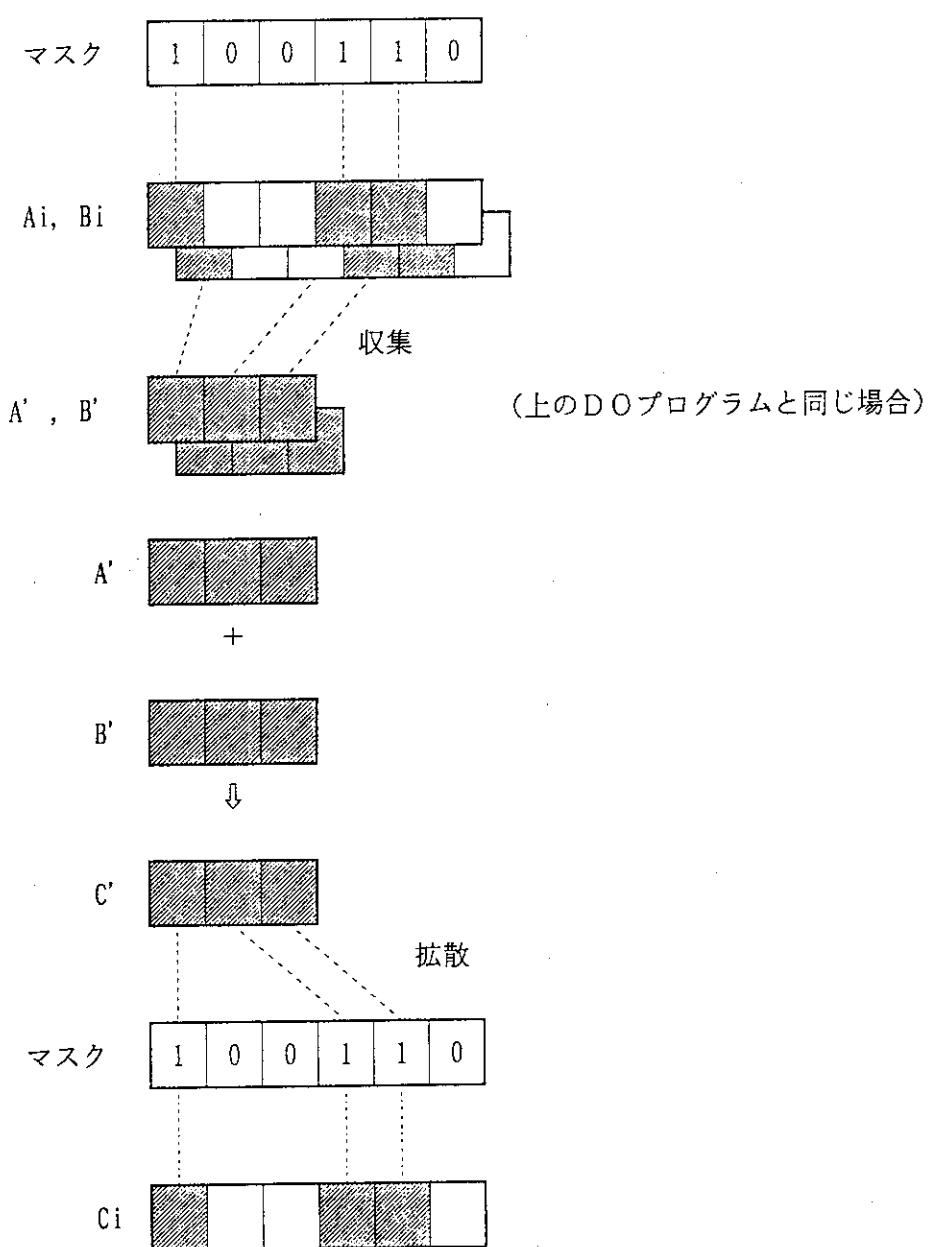


図 1.11 収集／拡散による方法

## (3) リスト・ベクトル・アクセス命令

ベクトルデータのうち、マスク部分が真の要素だけを示すリストベクトルを作成し、これに基づいて演算を行います(図 1.12)。

実行時間は、マスク対象のベクトル要素分の演算時間とリストベクトル生成操作分の時間を加えたものとなります。

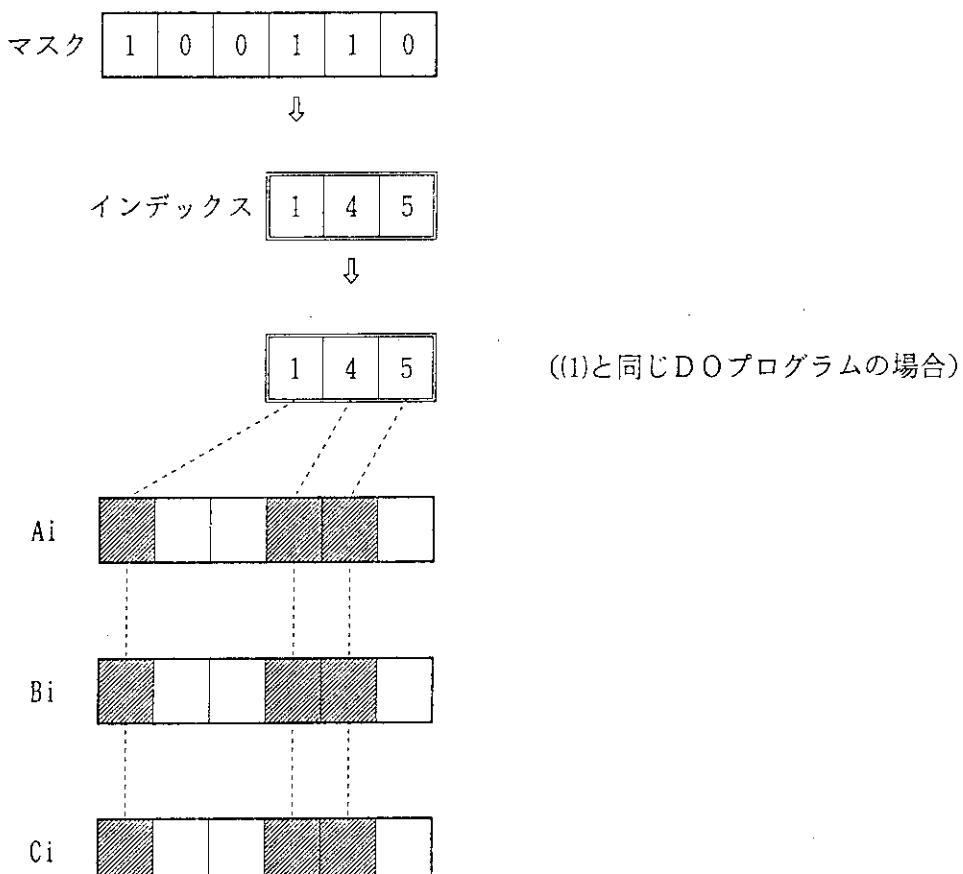


図 3.12 リストベクトルによる方法

これらの手法は、コンパイラがそのプログラムに最適なものを選択する。選択する基準は、以下の通りです。

- |       |                          |
|-------|--------------------------|
| マスク   | …… IF文の真率が高い場合           |
| 収集／拡散 | …… IF文の真率が低く、組み込み関数を含む場合 |
| リスト   | …… IF文の真率が低い場合           |

ここで、論理IF文、ブロックIF文、及び ELSE IF文の真率とは、IF文の条件式が評価された回数に対して、真と評価された回数の比率のことです。 算術IF文の真率とは、算術式が演算された回数に対して、その値が負、ゼロまたは正になる回数の比率のことです。

一般に、0 ~ 50% は真率が低いとみなされ、50~100%は真率が高いとみなされます。

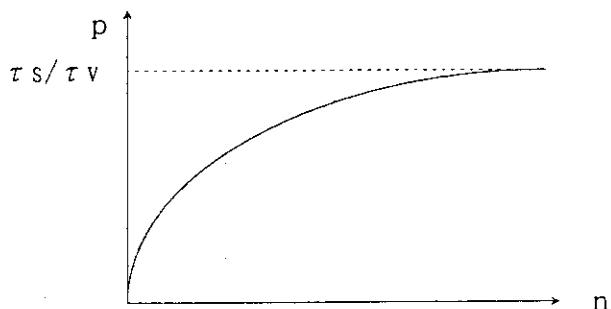


図2.1 DOループ長と処理性能pの関係

## 2.2 ベクトル化による処理効率向上比

スカラ計算機とベクトル計算機で実行したプログラムの相対処理向上比Pは、

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

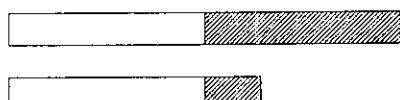
で表されます。

ここで、Vはベクトル化率と言い、プログラムをスカラで処理した場合に、ベクトル命令におきかわる部分の時間比率を表します。すなわち、プログラムがベクトル命令をどれだけ利用できるかを表しています（求め方は後の章で説明）。

$\alpha$ はベクトル／スカラ処理速度比です。ベクトル化できる部分をベクトル演算した時のスカラ演算速度に対するベクトル演算速度の比です（求め方は後の章で説明）。

### 2.2.1 ベクトル化率Vの効果

Vが小さい場合



効果が小さい

Vが大きい場合

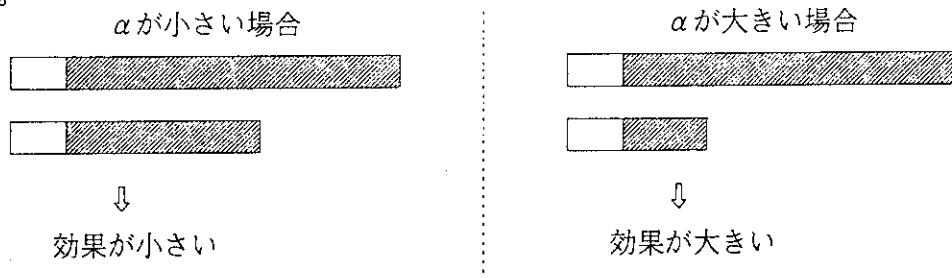


効果が大きい

この例から、ベクトル化可能のDOループが多いほど効果は大きくなります。しかし、ベクトル化率が高くても、速度向上が得られるとは限りません。ベクトル化率Vが高いことは、高い速度向上比を得るために必要な条件ですが、十分条件ではありません。

### 2.2.2 ベクトル／スカラ速度比 $\alpha$ の効果

ベクトル実行部分のベクトル処理による効果が大きい場合は、高い速度向上が得られます。



このための条件は、D O ループに対し

- (1) ベクトル長が長い
- (2) データ引用・定義が連続的
- (3) 演算子 (+, -, ×, ÷) が多い
- (4) IF 文が少ない
- (5) メモリアクセスが少ない

を満足させる必要があります。

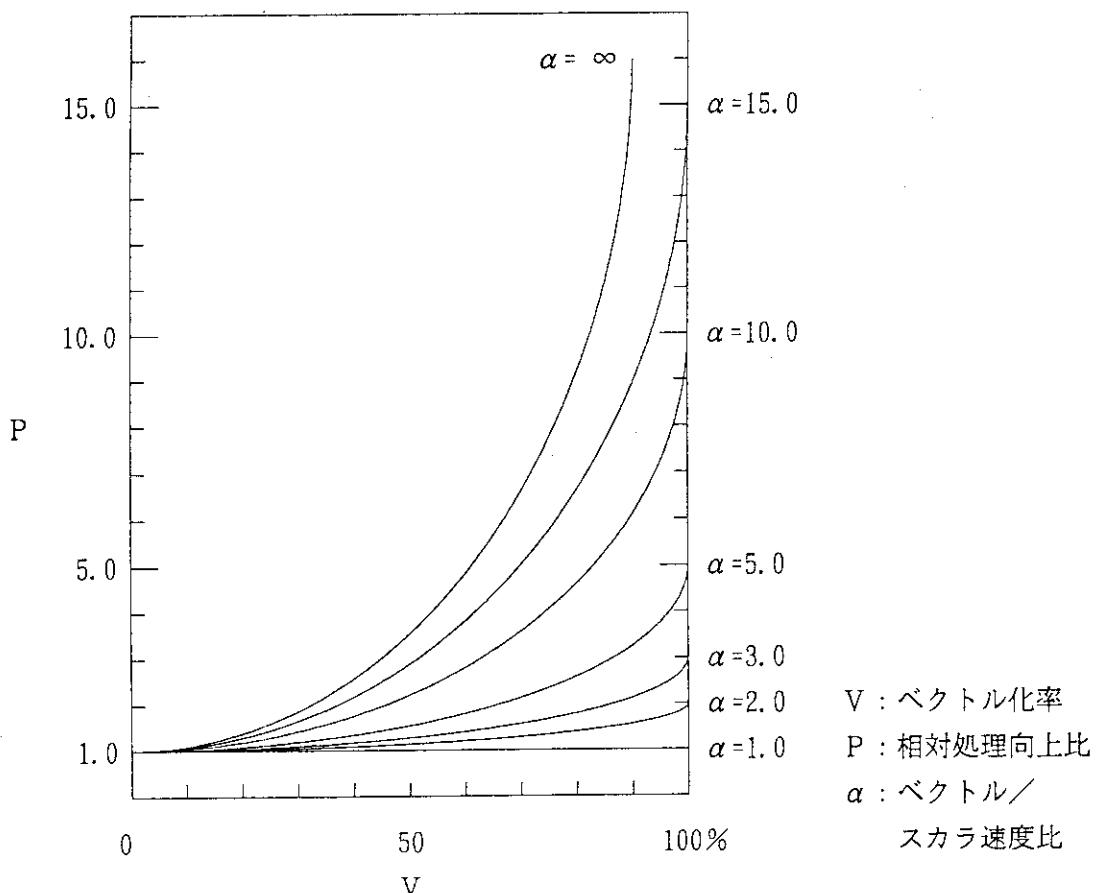
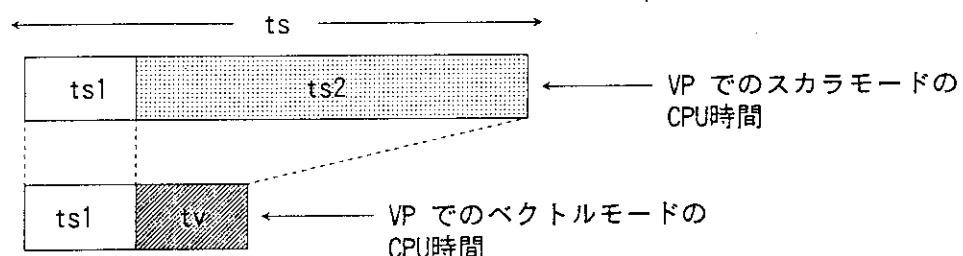


図2.2 ベクトル化率とプログラム処理速度の関係

### 2.2.3 ベクトル化率、ベクトル／スカラ速度比について

ベクトル化作業の指針となるベクトル化率とベクトル／スカラ速度比の定義を以下に示します。



$$\text{ベクトル化率}(V) = ts_2 / ts \quad (ts = ts_1 + ts_2)$$

$$\text{ベクトル / スカラ速度比}(\alpha) = ts_2 / tv$$

ts: プログラム全体あるいは一部をVPのスカラモードで実行した時のCPU時間

ts1: ベクトル処理できない部分のスカラCPU時間

ts2: ベクトル処理可能な部分のスカラCPU時間

tv: ベクトル処理可能な部分のベクトルCPU時間

このことから、Vと $\alpha$ が共に大きい程、プログラムの処理時間が短くなることが分かります。

### 2.3 ベクトル化例

実際の原子力コードをベクトル化して、V P 2 6 0 0上でベクトルモード、スカラモードで実行し、あるDOループのそれぞれのモードでのCPU 時間を測定ました。そのDOループを図2.3に示します。

```
*VOCL LOOP, LCMAX2, GT. 0
V DO 150 I=2, IRMAXM
V RIK3 = RIM(I, K3)
V RX3S = RIK3*DPS(I, K4)
V RX4S = CUX(I, K4)
V COD0 = RX1(I)*RX3S-RX2(I)*RIK3*AUX(I, K4)
V COD1 = -RX1(I)*RX4S-RX2(I)*(RX3S-DUX(I, K4))
V COD2 = RX2(I)*RX4S
V KXX = KX+(I-2)*LCMAX2
V A(KXX) = A(KXX)+ COD1*D1R(1, I)+COD2*D3R(1, I)
V B(KXX) = B(KXX)+COD0+ COD1*D1R(2, I)+COD2*D3R(2, I)
V C(KXX) = C(KXX)+ COD1*D1R(3, I)+COD2*D3R(3, I)
V 150 CONTINUE
```

図 2.3 ある原子力コードのDOループ

このDOループは 2876742 回実行され、ベクトル長は 199でした。CPU 時間はそれぞれのモードで次のようになりました。

ベクトル実行	.....	53.26 秒
スカラ実行	.....	956.30 秒

よって処理速度向上比は約18倍になることが分かります。

## 2.4 ベクトル化とは

ベクトル化は VP を有効に利用するための大変な作業です。ベクトル化の内容を理解することで、VP のハードウェアを充分に活用することができます。この章では、実際のコーディングレベルでのベクトル化例を述べます。

### 2.4.1 「ベクトル化」の定義

一般に我々が使っている「ベクトル化」という言葉は、VP を有効利用できるようにプログラムソースを改良することを言っています。しかし、狭義な意味でのベクトル化は、コンパイラによる自動ベクトル化を指します。

### 2.4.2 自動ベクトル化

自動ベクトル化とはFORTRAN77EX/VPコンパイラが、FORTRAN プログラムのベクトル処理可能な部分を認識して、ベクトル演算命令に置き換えることです。ベクトル演算とは、ベクトルデータを処理する演算のことと言います。

ここでベクトルデータとは、配列全体または配列の部分など同一属性を持った複数個のスカラデータより構成されるデータを言います。スカラデータとは、変数、配列要素などの単一データを言います。

表2.1 に自動ベクトル化の基本条件を示します。

表2.1 自動ベクトル化の基本条件

項目	ベクトル化対象項目	ベクトル化非対象項目
対象範囲	DOループ	DO WHILEループ DO UNTILループ IF/GOTO ループ
データの型	LOGICAL*4 INTEGER*4 REAL*4 REAL*8 COMPLEX*8 COMPLEX*16	LOGICAL*1 INTEGER*2 INTEGER*8 (注1) REAL*16 COMPLEX*32 CHARACTER NCHARACTER
文の要素	算術式 論理式 文関数の引用 組込み関数の引用	外部関数の引用
文	算術代入文 論理代入文 算術IF文 論理IF文 ブロックIF文 ELSE文 END IF文 単純GOTO文 CONTINUE文	CALL文 入出力文 計算形GOTO文 割当て形GOTO文 文番号代入文 PAUSE文 STOP文 RETURN文 END文
データの定義・引用	ベクトル化できる順序 (回帰演算でないこと)	回帰的なデータの定義・引用 (注2)

(注1) 論理積、論理和、排他的論理和、及び論理シフトの組込み関数の引数として引用された場合には、ベクトル化できる。

(注2) VP-2600 では、1次の回帰演算はベクトル化が可能である。

例 A(1)=A(1-1)\*B(1)+C(1)

### 2.4.3 自動ベクトル化の例

ここでは自動ベクトル化の例を示します。例題にあるFORTRAN プログラムの左端の V、S、M の文字は、V P コンパイラが output するベクトル化メッセージです。「V」はその実行文がベクトル命令で、「S」はスカラ命令で、そして「M」は一部がベクトル命令で、他の部分がスカラ命令で実行されることを示します。

#### (1) 連続データ、一定間隔データ、逆方向一定間隔データ、及びランダムな位置のデータを含む DO ループ

例 1

```
V      DO 100 I=1,N
V          A(I) = B(I) + C(I)
V  100 CONTINUE
```

例 2

```
V      DO 100 IDX=1, IDXMX
V          I = LV(IDX)
V          A(I) = B(I) + C(I)
V  100 CONTINUE
```

例 1 では、連続データの例で、配列 A, B, C に関する演算は DO 変数 I に関してベクトル化されています。また、例 2 では各配列、例えば A (L V (I D X)) はランダムな位置を指し示すリストベクトル L V によって間接的に指し示されています。このループは制御変数 I D X に関してベクトル化されています。

#### (2) DO の制御変数を含む DO ループ

例

```
V      DO 100 I=1,N
V          I1=I+1
V          A(I) = FLOAT(I1)
V  100 CONTINUE
```

DO の制御変数 I を含むステートメントも I に関してベクトル化されます。

## (3) 算術 IF 文、ブロック IF 文を含む DO ループ

例 1

```
V      DO 100 I=1,N
V          IF(A(I)) 10, 100, 20
V      10      A(I) = -A(I)
V      20      B(I) = S/A(I)
V  100 CONTINUE
```

例 2

```
V      DO 100 I=1,N
V          IF(A(I).GT.0.0) THEN
V                  B(I) = S/A(I)
V          ELSE
V                  B(I) = -S/A(I)
V          ENDIF
V  100 CONTINUE
```

例 1 は、算術 IF 文、例 2 は IF - THEN - ELSE 文を含む DO ループのベクトル化の例です。ともに、制御変数 I に関してベクトル化されています。さらに、より複雑なネスト構造になっている IF 文も自動ベクトル化されます。

## (4) 総和の計算、内積の計算を行う DO ループ

例 1

```
S = 0.0
V      DO 100 I=1,N
V          S = S + A(I)
V  100 CONTINUE
```

例 2

```
S = 0.0
V      DO 100 I=1,N
V          S = S + A(I)*B(I)
V  100 CONTINUE
```

例 1 は、配列 A の総和計算、例 2 は、配列 A、B の内積の計算を行う DO ループです。ともに、回帰演算ですがベクトル化されます（但し、最適化レベル OPT(E)以上）。

## (5) 最大値／最小値、及びそのインデックスを検索する DO ループ

例 1

```
AMAX = 0.0
AMIN = 0.0
V      DO 100 I=1,N
V          AMAX = MAX(AMAX,A(I))
V          AMIN = MIN(AMIN,A(I))
V  100 CONTINUE
```

例 2

```
AMAX = 0.0
V      DO 100 I=1,N
V          IF(AMAX.LT.A(I)) THEN
V                  AMAX = A(I)
V                  MXIDX = I
V          ENDIF
V  100 CONTINUE
```

例1は、配列Aの最大値、最小値、例2は配列Aの最大値とそのインデックスを検索するDOループでともに自動ベクトル化されます。

#### (6) ベクトル化非対象の型、データ、文を含むDOループ（部分ベクトル化）

例

```
M      DO 100 I=1,N
V          A(I) = B(I)*C(I)
V          B(I) = A(I)*S - A(I)
V          D(I) = B(I) + C(I)
M          E(I) = A(I)*B(I)
S          WRITE(6,1000) E(I)
M 100 CONTINUE
```

コンパイラは、一部のベクトル化不可能な部分のためにループ全体をベクトル化不可能とはせずに、ループをベクトル化可能部分と不可能部分に分割して、可能部分はベクトル化します。これを部分ベクトル化と呼びます。

この例では、DOループにベクトル化非対象のWRITE文を含んでいますが、部分ベクトル化によってWRITE文以外がベクトル化されます。

ただし、部分ベクトル化による処理は、常に速くなるとは限りません。スカラ処理があるためにスカラユニットとの割り込み命令のオーバーヘッドが生じるためです。

#### (7) 最内ループの外側のDOループ

例

```
M      DO 200 J=1,M
V          A(J) = S*A(J)
V          B(J) = S+A(J)
V          C(J) = A(J)+B(J)
M          D(J) = S*C(J)
V          DO 100 I=1,N
V              E(I,J) = E(I,J) + D(J)
V 100      CONTINUE
V 200      CONTINUE
```

自動ベクトル化は、原則的には最内DOループ（この場合100のループ）を対象としています。しかし、外側のループも分割を行って制御変数Jに関してベクトル化しています。なお、配列Dは、最内ループと外側のループで定義・引用の関係を持つために、回帰

演算となりベクトル化されません。

#### (8) 中間スカラ変数を含むDOループ

例

```
V      DO 100 I=1, N-1
V      DA = A(I+1)-A(I)
V      B(I) = B(I) * DA
V 100 CONTINUE
```

DOループ内にDAのような中間スカラ変数がある場合には、コンパイラはDA(I)に相当する作業用配列を内部発生して中間スカラ変数もベクトル化しています。

#### (9) 多重DOループ

例 1

```
V      DO 100 I=1, 30
S      DO 100 J=1, 30
S      DO 100 K=1, 10
V      A(I, J) = A(I, J) + B(K, I)*B(K, J)*S
V 100 CONTINUE
```

多重ループの場合、基本的に最内ループをベクトル化の対象とするが、翻訳時に各ループの回転数が判明している場合は、ベクトル長が長いか、あるいはVPハードウェアが効率良く処理できるDOループを選択し、そのDO変数についてベクトル化を行います。この例では、ベクトル長が長く、かつメモリ・アクセスの効率の良いDO変数Iに関してベクトル化されます。ベクトル長については、長い方が実行効率が良くなります。また、2次元配列A(I, J)はメモリ上では A(1, 1), ..., A(100, 1), A(1, 2), ... と列方向に連続して記憶されているため、連続的にアクセスするIに関してベクトル化した方が効率が良くなります。

## 例 2

```

REAL A(10,10), B(10,10), C(10,10)
V   DO 200 J=1,10
V   DO 200 I=1,10
V       IF(C(I,J).GT.0.0) THEN
V           A(I,J) = A(I,J) + B(I,J)*SQRT(C(I,J))
V       END IF
V   200 CONTINUE

```

この例も多重ループの最適化の例です。DO ループの回転数が少ない場合、ベクトル化の効果は低くなります。そのため多重ループで連続した領域を参照する場合、DO ループは一重化されます。この場合、DO ループの回転数は100 となり、2 次元配列A、B、C は一次元配列として扱われます。

## (10) 飛び出しのあるDO ループ

## 例

```

V   DO 100 I=1,N
V       T = SQRT(A(I))
V       IF (T.LT.1.0) GOTO 200
V       B(I) = T
V   100 CONTINUE
200 CONTINUE

```

いくつかの条件を満たした場合、ループ中から外へ飛び出しがあるDO ループがベクトル化されます。

## (注意)

飛び出しのあるDO ループがベクトル化された場合、飛び出す以前に実行された文が、ベクトル要素数分実行されることによって、エラーが発生する場合があるので注意が必要です。例えば、上の例でA の値が順に A(1) = 3.0, A(2) = 0.0, A(3) = -1.0 . . . であるとすると、スカラ計算ではA が 0.0のとき飛び出して正常終了します。一方、ベクトル演算ではS = -3.0 も含めて前もってSQRTを演算するため、エラーが発生します。

## (1 1) 収集／拡散を行うDOループ

例 1

```

ICNT = 0
V      DO 100 I=1,N
V          IF(A(I).GT.S) THEN
V              ICNT = ICNT + 1
V              B(!CNT) = A(I)
V          ENDIF
V      100 CONTINUE

```

例 2

```

ICNT = 0
V      DO 100 I=1,N
V          IF(A(I).GT.S) THEN
V              ICNT = ICNT + 1
V              A(I) = B(ICNT)
V          ENDIF
V      100 CONTINUE

```

例 1 では、配列Aから値がSよりも大きいものを取り出して配列Bに収集しています。逆に、例 2 では、配列Bから配列Aへ拡散手続きを行っています。いずれもコンパイラにより自動ベクトル化されます。

## (1 2) 一次回帰演算を含むDOループ

例

```

V      DO 100 I=2,N
V          A(I) = A(I-1)*B(I) + C(I)
V          D(I) = D(I) + A(I)*S
V      100 CONTINUE

```

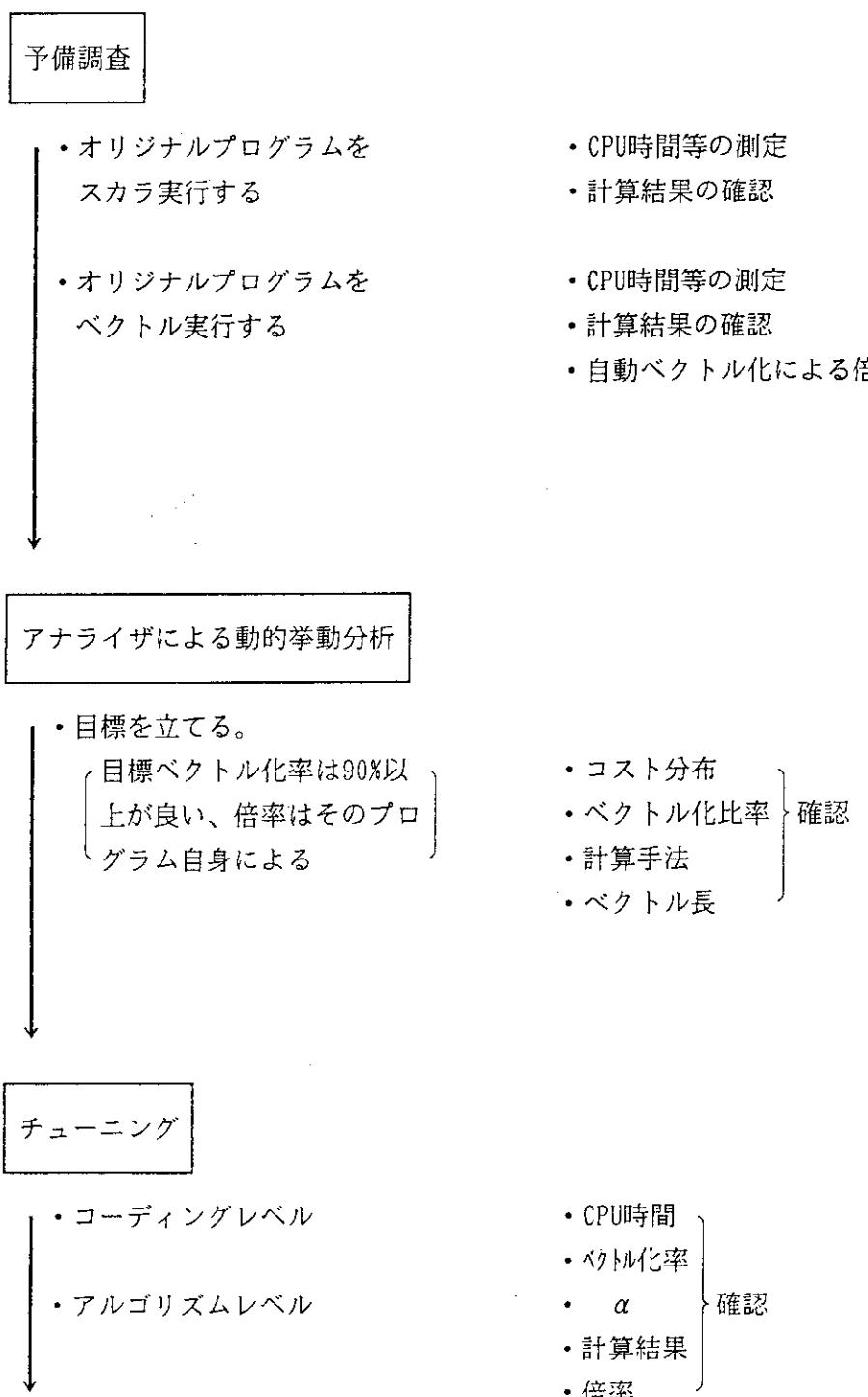
回帰演算の中で、一次の回帰演算を含むDOループは、いくつかの条件を満たした場合にはベクトル化されます。このベクトル化においては、V P 2 6 0 0 に関しては、コンパイル・オプションでV P 2 またはV P E が指定され、かつ最適化レベルがOPT(E)以上指定されている必要があります。ただし、条件を満たしていても、コンパイラが性能上の観点からベクトル化をしない場合もあります。

一次回帰演算のベクトル命令は、通常のベクトル命令に比べ高速には実行されません。

### 3. 基本的なベクトル化技法

#### 3.1 ベクトル化の手順

ベクトル化（ベクトル化チューニング）作業を効率良く進めるために、標準的な手順を示します。



ベクトル化版の完成

- ・チューニングプログラムをスカラ実行

- ・CPU時間等の測定
- ・計算結果の確認

- ・チューニングプログラムをベクトル実行

- ・CPU時間等の測定
- ・計算結果の確認

ベクトル化率

スカラ/チューニングベクトル倍率

チューニングスカラ/チューニングベクトル倍率

作業のまとめと検討

- ・ベクトル化チューニングを終わりにして良いか。

### 3.2 ベクトル化の促進

第2章では、ベクトルコンパイラが自動的にベクトル化する例について見てきました。ここでは、そのままでは自動ベクトル化されない、あるいは自動ベクトル化されるが、より高い実行効率を得るために利用者がプログラムに簡単な変更を行ってベクトル化する典型的な2つの方法について述べます。1つは、コンパイラへの制御情報を与えるために、直接プログラム中にベクトル化促進のための情報を書き込む方法です。この制御情報文は最適化制御行と呼ばれます。もう一つはDOループの構成を変える方法です。

#### 3.2.1 最適化制御行の活用

プログラムによってはベクトル化可能なDOループであるにもかかわらず、コンパイラの解釈能力不足のため自動ベクトル化されないために、コンパイラにベクトル化可能情報を与えベクトル化したい場合があります。また実行効率の観点から自動ベクトル化を抑止したい、あるいは効率の良い方法でベクトル化したい場合があります。

このような場合、ベクトルコンパイラへの制御情報を制御すべき対象のFORTRAN文の直前に挿入することによって自動ベクトル化を制御することができます。その一般形式は、

\*V O C L 制御行の有効範囲, 制御情報 [, 制御情報 . . . ]

です。最適化制御行の種類と意味、及び指定例を表3.1に示します。

ここでは、最適化制御行のうち、NOVREC、REPEAT、IF、SCALAR制御行を用いた効果的なベクトル化促進の方法について説明します。また、FORTRAN77EX/VPコンパイラの強化ポイントであるUNROLL制御行とEXVOL制御行について、実際の適用事例とそのときの実行速度向上について説明します。

表 3.1 最適化制御行の形式（続く）

有効範囲			制御情報	意味	指定例
TOTAL	LOOP	STMT			
○	○	—	V1 relop V2	V1とV2の関係を指示	*VOCL LOOP, M. GT. N
○	○	—	NOVREC (配列名 [, 配列名])	ベクトル化可能な配列であることを指示	*VOCL LOOP, NOVREC(A, B)
—	—	○	IF (P) [L]	I F 文の真率を P で指定	*VOCL STMT, IF(80) GO TO 400
—	—	○	IF(P1, P2, P3) n1, n2, n3	算術IF文で各分岐先への分岐確率を指定	*VOCL STMT, IF(10, 30, 60) 100, 200, 300
○	○	—	REPEAT(m)	ループの回数を指定	*VOCL LOOP, REPEAT(30 0)
○	○	—	SCALAR	ベクトル化の抑止	*VOCL TOTAL, SCALAR
○	○	—	VECTOR	ベクトル化の指定	*VOCL LOOP, VECTOR
○	○	—	NOEVL (EXでは EVAL)	式の評価順序を変更する最適化の抑止	*VOCL LOOP, NOEVL
○	○	—	TEMP(S, [S])	一時的変数であるとの指示	*VOCL LOOP, TEMP(S)
—	○	—	VI(a)	多重DOループにおけるベクトル化DO変数の指示	*VOCL LOOP, VI(I)
○	○	—	VDOPT	ベクトル長に応じた命令列の選択	*VOCL LOOP, VDOPT
○	○	—	NOVDOPT	VDOPT の打ち消し	*VOCL LOOP, NOVDOPT

表 3.1 最適化制御行の形式（続き）

有効範囲			制御情報	意味	指定例
TOTAL	LOOP	STMT			
( 以下はEX/VP コンパイラのみ有効)					
○	○	—	UNROLL(m)	アンローリング指示	*VOCL LOOP, UNROLL
○	○	—	NOUNROLL	UNROLLの打ち消し	*VOCL LOOP, NOUNROLL
○	○	—	EXVOL	多重ループの強制一重化	*VOCL TOTAL, EXVOL
○	○	—	VOL	システムの判断による一重化	*VOCL TOTAL, VOL
○	○	—	NOVOL	EXVOL/VOL の打ち消し	*VOCL LOOP, NOVOL
○	○	—	PREEX	不变式の先行評価の最適化	*VOCL LOOP, PREEX
○	○	—	NOPREEX	PREEX の打ち消し	*VOCL LOOP, NOPREEX

TOTAL ; 制御行の有効範囲がプログラム単位

LOOP ; 制御行の有効範囲がループ単位

STMT ; 制御行の有効範囲が文単位

## (1) NOVRECの使用例

S DO 100 IDX=1, IDMXM	V *VOCL LOOP, NOVREC(B)
V I = LV(IDX)	V DO 100 IDX=1, IDMXM
M A(I) = S*B(I)	V I = LV(IDX)
S C(I) = A(I)*D(I)	V A(I) = S*B(I)
S B(I) = (A(I)+C(I))/2	V C(I) = A(I)*D(I)
V 100 CONTINUE	V B(I) = (A(I)+C(I))/2
	V 100 CONTINUE

この例では、配列BはLV(IDX)によって間接的にアドレスが指定されています。したがってLV(IDX)に同じ値がある場合には、配列Bについて回帰的な演算となるため、ベクトル実行できません。コンパイラには、LVの値については何の情報もないでメッセージを出力してベクトル化を行いません。

利用者が事前にLVの値に重複がないことが分かっている場合、すなわち非回帰的な演算であることがわかっている場合には、最適化制御行を指定してベクトル化を促進することができます。

## (2) REPEART の使用例

V DO 100 I=1, N	V *VOCL LOOP, REPEAT(512)
V A(I) = (B(I)+C(I)+D(I)+E(I))/4 + (F(I)+G(I)+H(I)+O(I))/4	V DO 100 I=1, N
1 +(P(I)+Q(I)+R(I)+S(I))/4 + (T(I)+U(I)+V(I)+W(I))/4	V A(I) = (B(I)+C(I)+D(I)+E(I))/4 + (F(I)+G(I)+H(I)+O(I))/4
2 +(X(I)+Y(I)+Z(I)+A(I))/4	1 +(P(I)+Q(I)+R(I)+S(I))/4 + (T(I)+U(I)+V(I)+W(I))/4
V 100 CONTINUE	2 +(X(I)+Y(I)+Z(I)+A(I))/4

↓

V *VOCL LOOP, REPEAT(512)	V *VOCL LOOP, REPEAT(512)
V DO 100 I=1, N	V DO 100 I=1, N
V A(I) = (B(I)+C(I)+D(I)+E(I))/4 + (F(I)+G(I)+H(I)+O(I))/4	V A(I) = (B(I)+C(I)+D(I)+E(I))/4 + (F(I)+G(I)+H(I)+O(I))/4
1 +(P(I)+Q(I)+R(I)+S(I))/4 + (T(I)+U(I)+V(I)+W(I))/4	1 +(P(I)+Q(I)+R(I)+S(I))/4 + (T(I)+U(I)+V(I)+W(I))/4
2 +(X(I)+Y(I)+Z(I)+A(I))/4	2 +(X(I)+Y(I)+Z(I)+A(I))/4
V 100 CONTINUE	V 100 CONTINUE

ベクトル・レジスタの容量は個数\*語数=一定ですが、その構成はコンパイラが翻訳時に最適な構成を選択しています。V P 2 6 0 0の場合、可能な構成は、(個数、語数)=(8, 2048)、(16, 1024)、(32, 512)、(64, 256)、(128, 128)、(256, 64)であり、基本構成は(256, 64)となっています。

N=512とすると、組み合わせは(32, 512)となるので、最適化制御行でDOループの繰り返しの回数を512と指定すれば、コンパイラはレジスタの構成を(32, 512)に変更します。これによって演算対象データがすべてレジスタ上にロードするために実行効率が改善されます。

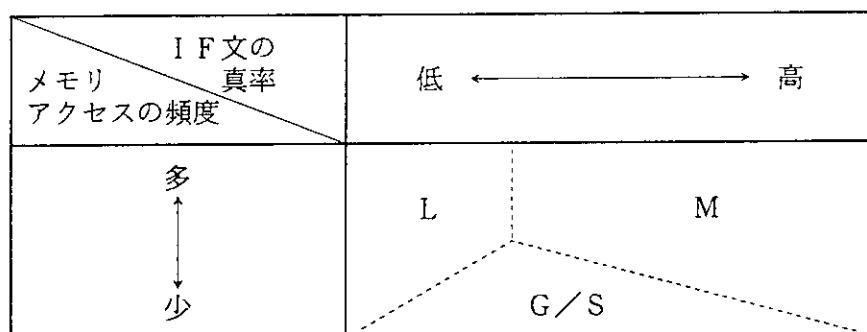
ただし、REPEAT制御行によって繰り返し回数が指定されるとそれ以上はループを繰り返し実行しないために、DOループの繰り返し回数(N)は最適化制御行で指定した繰り返し数(512)より必ず小さくなければなりません。

## (3) IF文の真率指定

DOループ内にIF文が含まれる場合には、コンパイラは1.5.3章で説明したようにマスク操作による方法、収集／拡散による方法、そしてリスト・ベクトルによる方法のいずれかでベクトル化します。どの方法が最適かは表3.2に示すようにIF文の真率、及びIF文内のメモリアクセスの頻度に依存するために個々のDOループごとに異なります。

利用者が事前にIF文の真率がわかる場合には、最適化制御行で真率を指定して、より効率の良い方法でベクトル化することが出来ます。ここでの例は、IF文の真率が3%であることが事前に分かっている場合で、このときはリストベクトルによる方法でベクトル化されています。

表3.2 IF文のベクトル化の方法



L:リストベクトル, M:マスクベクトル, G/S:収集・拡散

V DO 100 I=1,N	V DO 100 I=1,N
V A1(I) = S1*B1(I)-C1(I)	V D1(I) = Z1*E1(I)-F1(I)
V D1(I) = Z1*E1(I)-F1(I)	⇒ V *VOCL STMT, IF(3)
V IF( A1(I).EQ.0. ) THEN	V IF( A1(I).EQ.0. ) THEN
V D1(I) = D1(I)/A1(I)	V D1(I) = D1(I)/A1(I)
V E1(I) = S2*A1(I)+S3	V E1(I) = S2*A1(I)+S3
V END IF	V END IF
V 100 CONTINUE	V 100 CONTINUE

THE STATEMENTS IN THIS RANGE ARE  
VECTORIZED BY MASKED OPERATION METHOD.

THE STATEMENTS IN THIS RANGE ARE  
VECTORIZED BY LIST VECTOR METHOD.

## (4) スカラ処理の指定

<pre>V      DO 100 J=1, 2 V      DO 100 I=1, 2 V          A(I, J) = 0.0 V  100 CONTINUE       RETURN       END</pre>	$\Rightarrow$	<pre>*VOCL TOTAL, SCALAR SUBROUTINE SUB DO 100 J=1, 2 DO 100 I=1, 2 A(I, J) = 0.0 100 CONTINUE RETURN END</pre>
--	---------------	---

この例のように、ベクトル長が短い場合など、サブルーチン全体を自動ベクトル化する必要のない場合は、最適化制御行によって、ルーチン全体のベクトル化を抑止することができます。

## (5) UNROLLING (ループ展開機能)

アンローリングとはDOループ内の実行文をn重に展開し、その代わり回転数を1/nに減らす機能です(n=多重重度)。FORTRAN77EX/VPでは、最大多重重度が9重まで可能で、アンローリングするDOループや多重重度はコンパイラが自動的に選択します。このことを、自動アンローリングと言います。

ベクトル化対象DOループにおけるアンローリングは、多重DOループにおいてその効果が発揮されます。その例を次に示します。

## 例1

```
S2    DO 10 J=1, 50
V2    DO 20 I=1, 50
V2    A(I)=A(I)+B(I, J)
V2 10 CONTINUE
S2 20 CONTINUE
```

実行イメージ⇒

```
DO 10 J=1, 50, 2
DO 20 I=1, 50
A(I)=A(I)+B(I, J)
+B(I, J+1)
10 CONTINUE
20 CONTINUE
```

例1では、外側のDOループの回転数を減らすことによって内側のDOループ内の演算密度を高め、パイプラインの有効活用ができるようになっています。

## 例2

```
REAL*8 A(50, 50), B(50, 50), C(50, 50)
V    DO 10 J = 1, 50
V3   DO 20 I = 1, 3
V3   A(I, J)=B(I, J)+C(I, J)    実行イメージ⇒
V3 20 CONTINUE
V 10 CONTINUE
```

```
V    DO 10 J = 1, 50
V    A(1, J)=B(1, J)+C(1, J)
V    A(2, J)=B(2, J)+C(2, J)
V    A(3, J)=B(3, J)+C(3, J)
V 10 CONTINUE
```

例2では、内側のDOループを全展開しています(V3の表示は多重重度3で展開されたという意味)。この結果、Iに関するDOループ手続きのオーバーヘッドがなくなり、ベクトル長の長いJに関するループでベクトル化されています。

多重重度3で最内ループをアンローリングした場合としない場合(FORTRAN77/VPと同等)の実行時間は次のようになります。

自動アンローリング	$2.612 \times 10^{-4}$ (ミリ秒)
アンローリング抑止	$3.481 \times 10^{-4}$ (ミリ秒)

処理速度向上比は約1.3倍です。図3.1にこのケースでのFORTRAN77/VPとFORTRAN77EX/VPのパイプライン処理を示します。

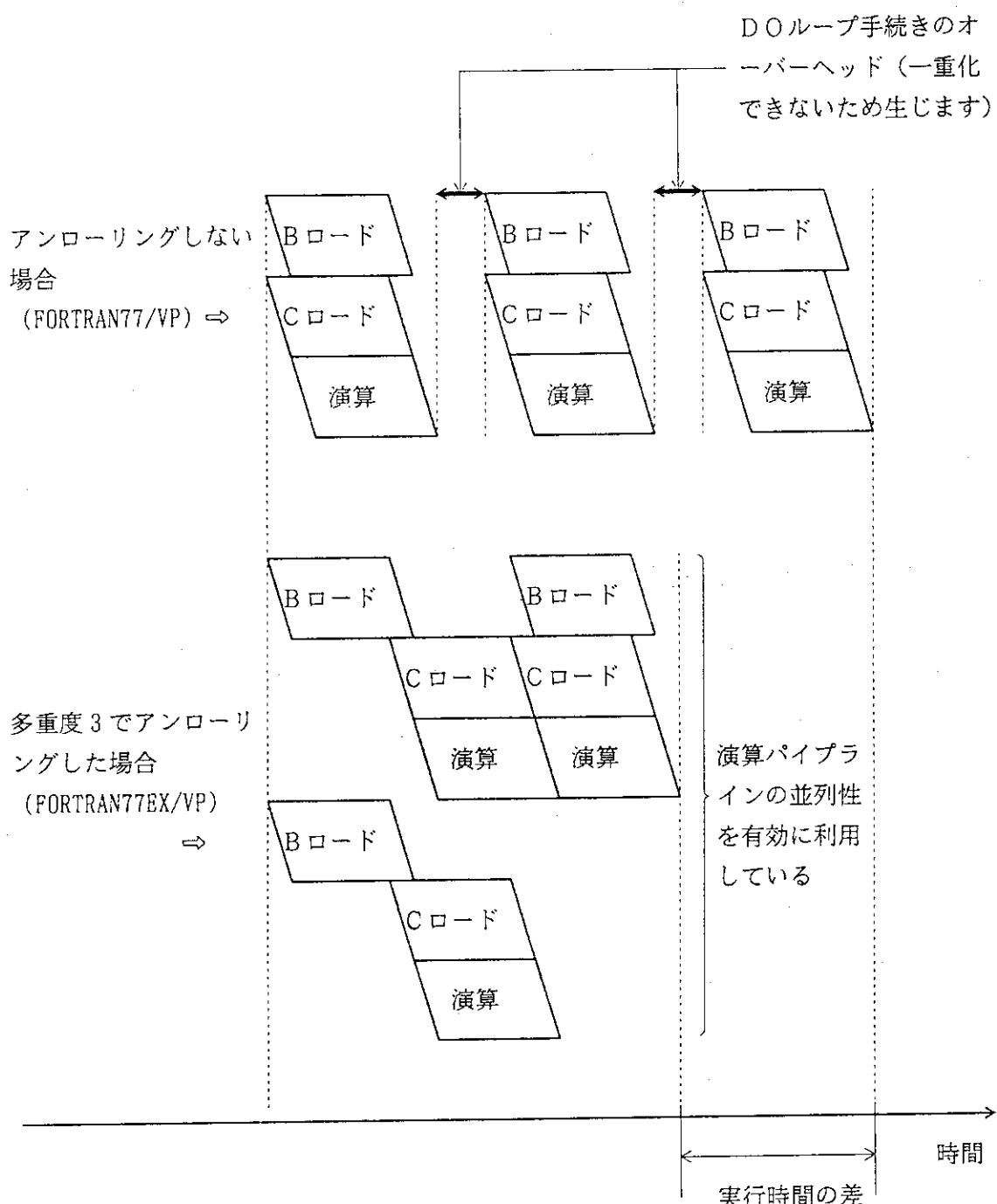


図 3.1 例 2 におけるパイプライン処理の違い

アンローリングした場合は、演算パイプラインを並列動作させて、効率良く処理していることが分かります。

しかし、ベクトル化対象D O ループの特性が前もって分かっている場合には、最適化制御行を指定することで、さらに効率の良いアンローリングをすることができる場合があります。次に、このケースを挙げます。

## 〔ケース 1〕 最内ループの回転数が 4 以上 9 以下の場合

<pre> DIMENSION A(50, 50), B(50, 50), 1      C(50, 50) V2    DO 10   I = 1, 50 S2    DO 20   J = 1, 5 V2    A(J, I)=B(J, I)+C(J, I) S2    20 CONTINUE V2    10 CONTINUE </pre>	<pre> DIMENSION A(50, 50), B(50, 50), 1      C(50, 50) V     DO 10   I = 1, 50 *VOCL LOOP, UNROLL(5) ⇒ S5    DO 20   J = 1, 5 V5    A(J, I)=B(J, I)+C(J, I) S5    20 CONTINUE V     10 CONTINUE </pre>
--	--

自動ベクトル化では、最大多度が 3 までのアンローリングを行います。4 以上の場合は最適化制御行 UNROLL を指定する必要があります。この例では内側ループを 5 重（全展開）にアンローリングを行い高速化しています。このケースでの最適化指示をした場合と自動アンローリングの場合との実行時間の比較は次のようにになります。

アンローリング指示	$2.056 \times 10^{-3}$ (ミリ秒)
自動アンローリング	$2.253 \times 10^{-3}$ (ミリ秒)

処理速度向上比は約 1.1 倍になっています。

## 〔ケース 2〕 3 重ループにおいて内側の 2 つのループをアンローリングする例

<pre> REAL*8 A(10, 10, 10), 1 B(10, 10, 10), 2 C(10, 10, 10), V3   DO 30   I = 1, 100 S3   DO 20   J = 1, 3 V9   DO 10   K = 1, 3 V9   A(J, K, I)=B(J, K, I)+C(J, K, I) V9   10 CONTINUE S3   20 CONTINUE V3   30 CONTINUE </pre>	<pre> REAL*8 A(10, 10, 10), 1 B(10, 10, 10), 2 C(10, 10, 10), V     DO 30   I = 1, 100 *VOCL LOOP, UNROLL('FULL') ⇒ V3   DO 20   J = 1, 3 V9   DO 10   K = 1, 3 V9   A(J, K, I)=B(J, K, I) 1           +C(J, K, I) V9   10 CONTINUE V3   20 CONTINUE V     30 CONTINUE </pre>
---	---

この例では、内側の 2 つのループの回転数がそれぞれ小さいので（一重化した場合でも多度が 9 以下）、最適化制御行 UNROLL (' FULL ') を指定することで内側の 2 つのループを全展開しています。最適化指示をした場合と自動アンローリングの場合との実行時間の比較は

アンローリング指示	$1.066 \times 10^{-2}$ (ミリ秒)
自動アンローリング	$1.340 \times 10^{-2}$ (ミリ秒)

となり、処理速度向上比は約 1.26 倍です。

## [ケース 3] 外側ループの回転数が小さい場合

<pre>         DIMENSION A(30,30), B(30,30),         1   C(30,30) S2    DO 10 J = 1, 5 V2    DO 20 I = 1, 30 V2    A(J,I)=B(J,I) + C(J,I) V2    20 S2    10 CONTINUE       </pre>	<pre>         DIMENSION A(30,30), B(30,30),         1   C(30,30) *VOCL LOOP, UNROLL(5) S5    DO 10 J = 1, 5       ⇒ V5   DO 20 I = 1, 30 V5    A(J,I)=B(J,I) + C(J,I) V5    20 CONTINUE S5    10 CONTINUE       </pre>
--	--

## 実行イメージ

```

DO 10 I=1,100
A(I,1) = B(I,1) + C(I,1)
A(I,2) = B(I,2) + C(I,2)
A(I,3) = B(I,3) + C(I,3)
A(I,4) = B(I,4) + C(I,4)
A(I,5) = B(I,5) + C(I,5)
10 CONTINUE
      
```

このケースではループの入れ換えを行った上で、全展開を行い演算密度を高くしています。最適化指示をした場合と自動アンローリングの場合との実行時間の比較は次のようにになります。

アンローリング指示	$1.612 \times 10^{-3}$	(ミ秒)
自動アンローリング	$1.924 \times 10^{-3}$	(ミ秒)

処理速度向上比は約1.2倍です。

次に、アンローリングをすることで遅くなるケースを挙げます。

## [ケース 4] 一重化可能な例

<pre> REAL*8 A(50,50), B(50,50), 1 C(50,50) V     DO 10 J = 1, 50 V3    DO 20 I = 1, 3 V3    A(I,J)=B(I,J)+C(I,J) V3    20 CONTINUE V     10 CONTINUE       </pre>	<pre> REAL*8 A(50,50), B(50,50), 1 C(50,50) *VOCL LOOP, NOUNROLL V     DO 10 J = 1, 50       ⇒ V   DO 20 I = 1, 3 V     A(I,J)=B(I,J)+C(I,J) V     20 CONTINUE V     10 CONTINUE       </pre>
--	---

ループの一重化可能な例ではアンローリングより一重化を選択したほうが速くなります。自動ベクトル化ではアンローリングされるので、最適化制御行NOUNROLLを指定してアンローリングを抑止します。

## (6) EXVOL (強制一重化機能)

例1 プログラムのような不連続なデータアクセス ( $I = 2, N$ ) をする多重DOループがあるとします。このような場合、FORTRAN77/VPでは一重ベクトル化することはできませんでしたが、FORTRAN77EX/VPでは例2のように最適化制御行 (EXVOL) を指示することでそれが可能になりました。

## 例1 不連続データをアクセスする多重ループの例

```
REAL*8 A(N,N), B(N,N), C(N,N)
S    DO 10 J = 1 , N
V    DO 20 I = 2 , N
V      A(I, J) = B(I, J) + C(I, J)
V    20 CONTINUE
S    10 CONTINUE
```

## 例2 EXVOL 指示による強制一重化 (FORTRAN77EX/VP)

```
REAL*8 A(N,N), B(N,N), C(N,N)
*VOL LOOP, EXVOL
S    DO 10 J = 1 , N
V    DO 20 I = 2 , N
V      A(I, J) = B(I, J) + C(I, J)
V    20 CONTINUE
S    10 CONTINUE
```

FORTRNAN77 EX/VP VECTORIZATION MESSAGES  
----- THE NESTED DO LOOPS ARE VECTORIZED BY I AND J  
----- THE STATEMENT IN THIS RANGE ARE VECTORIZED BY  
MASKED OPERATION METHOD

例2のコンパイルリストにスカラ実行を示すSが表示されていますが、ベクトル化メッセージの通りマスク処理により一重ベクトル化されています。

例1と例2の場合でのパイプラインの動作を比較します。配列B( $I, J$ )はメモリ上では図3.2の様に配置されています。

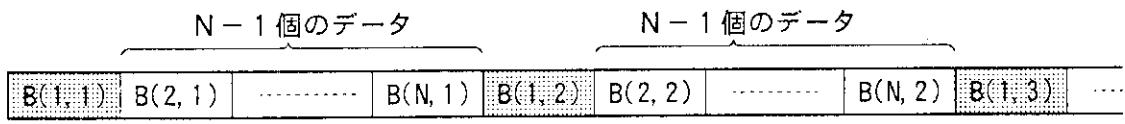


図 3.2 配列 B( $I, J$ ) のメモリ上での配置

例 1 では図3.2 のようなデータの不連続性から、(N-1) 個のデータのベクトルロードがN回不連続に実行されます。その結果、演算パイプラインの動作も不連続となり、パイプラインのオーバーヘッドが生じます。(図 3.3)

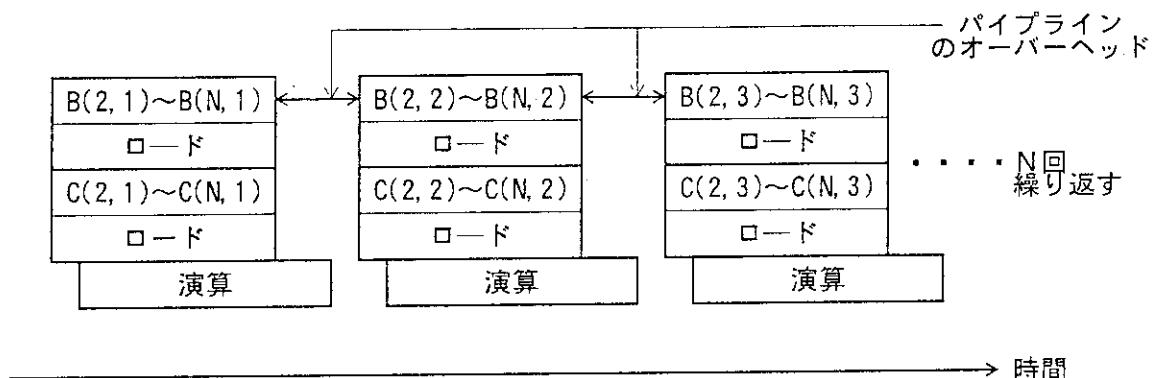


図 3.3 EXVOL 指示がない場合のパイプラインの動作 (例 1)

一方、EXVOL 指示により一重化した場合は、アクセスしないデータも含めて  $N * N$  個のデータを一度にベクトルロードします。このときデータは連續にアクセスされるので図 3.3でのパイプラインのオーバーヘッドは無くなります。また、ベクトル長が長くなるので、ベクトル演算の効率を高めることができます。一重化した場合のパイプラインの動作を図 3.4 に示します。

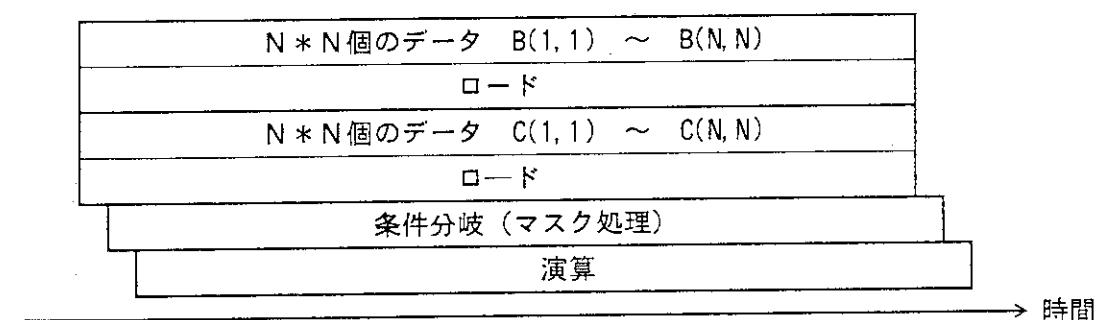


図 3.4 EXVOL 指示で一重化した場合のパイプラインの動作 (例 2)

EXVOLで一重化した場合のイメージは以下のプログラムになります。

```
V      DO 10 J = 1, N
V      DO 20 I = 1, N
V      IF(I.LT.2) THEN
V          GO TO 20
V      ELSE
V          A(I,J) = B(I,J) + C(I,J)
V      END IF
V 20 CONTINUE
V 10 CONTINUE
```

EXVOLによる一重化処理では、このように IF文のマスク処理を含んでいるために必ずしも高速化ができるとは限りません。逆に遅くなるケースもあるので注意が必要です。EXVOLで高速化できる場合の指標を表3.3にまとめます。

表 3.3 EXVOLの高速化の指標

アクセスする配列要素の 配列全体に対する割合	演算数
大きい	少ない

- \*) 一重化で効果が上がるにはVP2600の場合、ベクトル長が2048までです。また、マスク処理にかかる時間はベクトル長に比例するので、ベクトル長が長い場合、パイプラインのオーバーヘッドより大きくなっていることがあります。
- \*) DOループ内の演算数が多い場合はEXVOLを指示しても一重化できないことがあります。

実際に次の様なプログラムについてEXVOLによる高速化の効果を調べてみました。

```
DIMENSION A(10,10,10)
& B(10,10,10),C(10,10,10)
*VOCL LOOP,EXVOL
DO 10 K = 1, 10
DO 10 J = 2, 10
DO 10 I = 2, 10
A(I,J,K) = B(I,J,K) + C(I,J,K)
10 CONTINUE
```

EXVOL 指示	$1.348 \times 10^{-4}$ (ミリ秒)
EXVOL なし	$2.247 \times 10^{-4}$ (ミリ秒)

EXVOL指示と指示無しのときの処理速度向上比は、約1.7倍です。

### 3.2.2 FUNCTION、SUBROUTINE文を含むDOループのベクトル化

DOループ内に、FUNCTION、SUBROUTINEなどの外部参照を含む場合には、その参照部分はもちろんのこと、ループ全体がベクトル化されないことがあります。FORTRAN77EX/VPが提供している組み込み関数では、ベクトル化される関数とされない関数が、また科学技術計算用サブルーチン・ライブラリ(SSL II)でもベクトル計算機用に開発されたベクトル版(SSL II VP)と通常のスカラ版があります。これらを使う場合には、FORTRAN77EX/VPの使用手引書、科学技術計算用サブルーチン・ライブラリ使用手引書で調査して使う必要があります。

ここでは、ベクトル化されない外部参照を含むDOループについて、(1)コンパイルオプションINLINEを指定する方法、(2)ループを分割する方法、(3)参照される副プログラムをループ内に展開する方法、逆に(4)DOループを副プログラムへ引き込む方法、及び(5)DOループに先立ってテーブルを作成しループ内ではテーブルを引用する方法について述べます(注意:この章以降本文中の例は、何も変更せずにコンパイラによってベクトル化、最適化ができる場合があります。これは説明を簡単にするためのあくまでの例ですのでご考慮下さい)。

#### (1) コンパイルオプションINLINE(EXTERNAL)を指定する

副プログラムを呼び出し元で展開することをインライン展開と呼びます。FORTRAN77EX/VPではいくつかの条件を満たせば、コンパイルオプションINLINE(EXTERNAL)を指定することによって、インライン展開を行います。さらに最適化も行うので、ベクトル処理が可能ならばベクトル命令が発行されます。

S DO 100 I=1,N	V DO 100 I=1,N
S A(I) = S1*S2	V A(I) = S1*S2
S CALL SUB(A(I))	V CALL SUB(A(I))
S C(I) = S5*A(I)	V C(I) = S5*A(I)
S 100 CONTINUE	V 100 CONTINUE
⇒	
SUBROUTINE SUB(A)	
B = A + S3	
A = S4*B	
RETRUN	
END	
SUBROUTINE SUB(A)	
B = A + S3	
A = S4*B	
RETURN	
END	

## (2) DO ループを分割し、外部参照以外の部分をベクトル化する

一部のベクトル化不可能部分のために、ループ全体をベクトル化不可能としないようにループの分割を行います。

M      DO 100 I=1,N	V      DO 100 I=1,N
V      A(I) = S1*B(I)	V      A(I) = S1*B(I)
S      CALL SUB	⇒      DO 200 I=1,N
S      C(I) = S2*A(I)	CALL SUB
S      100 CONTINUE	200 CONTINUE
	V      DO 300 I=1,N
	V      C(I) = S2*A(I)
	V      300 CONTINUE

## (3) 副プログラムを上位展開する

CALLされる副プログラムをその場所に組み込むことを上位展開と呼びます。コンパイラオプションINLINE(EXTERNAL)も上位展開の機能を持ちますが、いくつかの条件を満たさないと行いません。そのため、手作業による上位展開を行う必要がある場合があります。以下の例では、展開前には副プログラムもベクトル化されないが、上位展開によって副プログラムの部分もベクトル化されます。さらに、上位展開によって呼び出しのためのオーバヘッドが省かれるために、より効率が良くなります。

S      DO 100 I=1,N	V      DO 100 I=1,N
S      A(I) = S1*S2	V      A(I) = S1*S2
S      CALL SUB(A(I))	⇒      V      B = A(I) + S3
S      C(I) = S5*A(I)	V      A(I) = S4*B
S      100 CONTINUE	
SUBROUTINE SUB(A)	V      C(I) = S5*A(I)
B = A + S3	V      100 CONTINUE
A = S4*B	
RETURN	
END	

## (4) DO ループを副プログラムに引き込む

(3) の上位展開する方法とは逆に、上位のDO ループを下位の副プログラムに引き込み副プログラムをベクトル化する方法です。(2) で述べたDO ループの分割を行った後に、ベクトル化されないで残った外部参照部分をベクトル化する場合に用います。

```

S      DO 100 I=1,N          CALL SUB(A, B)
S      CALL SUB(A(I), B(I))
S  100 CONTINUE
S
S      SUBROUTINE SUB(A, B)    SUBROUTINE SUB(A, B)
S      A = S1*B + S2          DIMENSION A(20), B(20)
S      RETRUN
S
S      END
S
V      DO 100 I=1,N
V      A(I) = S1*B(I) + S2
V  100 CONTINUE
V      RETURN
V
V      END

```

## (5) 外部参照される関数をテーブル化する

この例では、一様乱数を発生する RANDOM(0) がループ内にあるためにループ全体がベクトル化されません。あらかじめ乱数のテーブルを作成しておき、演算処理ループ(200 のループ) ではこのテーブルを引用するように変更すれば、テーブル作成部分はベクトル化されませんが演算処理部分はベクトル化されます。

```

S      DO 100 I=1,N          S      DO 100 I=1,N
S      RANDU(I) = RANDOM(0)  S  100 CONTINUE
S      A(I) = S*B(I)          S
S      C(I) = A(I)*RANDOM(0)  V      DO 200 I=1,N
S  100 CONTINUE
S
V      A(I) = S*B(I)
V      C(I) = A(I)*RANDU(I)
V  200 CONTINUE

```

3.2.3 入出力文を含むループのベクトル化

入出力文をループ中に含む場合は、外部参照を含む場合と同様にその部分はベクトル化されません。したがって、入出力文がDO ループの他の部分のベクトル化の妨げにならないようにするDO ループを修正する必要があります。

(1) DO ループの分割をして、入出力文以外の部分をベクトル化する。

S        DO 100 I=1, N S        A(I) = S1*C(I) S        B(I) = S2*C(I) S        WRITE(6, 1000) A(I), B(I) S        A(I) = A(I) + S3*B(I) S   100 CONTINUE	V        DO 100 I=1, N V        A(I) = S1*C(I) V        B(I) = S2*C(I) V        100 CONTINUE V        WRITE(6, 1000) (A(I), B(I), I=1, N)
	V        DO 200 I=1, N V        A(I) = A(I) + S3*B(I) V        200 CONTINUE

例のように、入出力はDO形並びにして一括処理するほうが効率が良くなります（命令が1回実行するだけになります）。

(2) 作業用配列を用いて、入出力を後で一括処理する

S        DO 100 I=1, N S        A(I) = S1*C(I) S        B(I) = S2*C(I) S        WRITE(6, 1000) A(I), B(I) S        A(I) = A(I) + S3*B(I) S   100 CONTINUE	V        DO 100 I=1, N V        A(I) = S1*C(I) V        B(I) = S2*C(I) V        AA(I) = A(I) V        A(I) = A(I) + S3*B(I) V        100 CONTINUE V        WRITE(6, 1000) (AA(I), B(I), I=1, N)
--	---

この例は、出力する変数に対しWRITE文の後で代入がされている場合の変更方法を示しています。

一般に、コンパイラの自動ベクトル化機能は、ロード／ストア・パイプラインによるデータの供給と並列動作可能な演算パイプラインが有効に動作するように最適化を図っているため、ループ中の演算密度を高くすることが高速実行のためには有効です。

(2) の方法は、(1) に較べて作業用配列を作成する余分の時間を費やしますが、ループを分割していないために演算密度が高く、全体的には効率が良くなります。

### 3.2.4 多重ループのベクトル化

多重ループでは、原則として最内ループが自動ベクトル化の対象となります。コンパイラの進歩により、一重化やループの反転、アンローリングなどの最適化が行われるようになりました。しかし、これらはいくつかの条件が整った場合に行われるため、まだ十分とは言えません。したがってループを書き換えて効率良いベクトル化を行うことになります。手作業で行うには、(1) ループの順序を入れ換えてベクトル化する方法、(2) 多重ループを一重化してベクトル化する方法、(3) 最内ループを陽に書き下してより外側のループを最内ループにしてベクトル化させる方法などがあります。

#### (1) ループの順序を入れ換えてベクトル化する方法

##### 例1 ループの順序入れ換え

例にあるように、添字が I、J の順で使われている 2 次元配列が何箇所にも使われる場合には、行方向より列方向から計算するよう変換します。

一般に配列はメモリ上では  $A(1,1), \dots, A(100,1), A(1,2), \dots$  と列方向に連続してストアされているため、アクセス効率を良くするためにメモリアクセスが連続になるように添字を動かします。

<pre>DIMENSION A(100, 100) S    DO 100 I=1, 50 V    DO 100 J=1, 50 V    A(I, J) = 0. V    ..... V    100 CONTINUE</pre>	$\Rightarrow$	<pre>DIMENSION A(100, 100) S    DO 100 J=1, 50 V    DO 100 I=1, 50 V    A(I, J) = 0. V    ..... V    100 CONTINUE</pre>
---	---------------	---

##### 例2 ループの順序入れ換え

ベクトル長が長い方が実行効率が良いので、例のようにベクトル長の長い I についてのループが内側になるように変換します。

<pre>DIMENSION A(100, 100) S    DO 100 I=1, 50 V    DO 100 J=1, 10 V    A(I, J) = 0. V    ..... V    100 CONTINUE</pre>	$\Rightarrow$	<pre>DIMENSION A(100, 100) S    DO 100 J=1, 10 V    DO 100 I=1, 50 V    A(I, J) = 0. V    ..... V    100 CONTINUE</pre>
---	---------------	---

## 例 3 ループの順序入れ換え

配列Aは、Jのループについては回帰演算であるためにベクトル化されません。ループの順序を入れ換えると計算上の矛盾がない場合には入れ替えます。この変更によってAはIのループについては非回帰演算なので内側がベクトル化されます。

S DO 100 I=1, N		S DO 100 J=2, N
S DO 100 J=2, N	⇒	V DO 100 I=1, N
S B(I, J) = S1*A(I, J-1)		V B(I, J) = S1*A(I, J-1)
S A(I, J) = B(I, J)/S2		V A(I, J) = B(I, J)/S2
S .....		V .....
S 100 CONTINUE		V 100 CONTINUE

## (2) 多重ループを一重化する

## 例 1 EQUIVALENCE を用いた一重ループ

変換前は最内ループのベクトル長は10であるが、変換後は一重ループとなりベクトル長は200である。

DIMENSION A(20, 10), B(20, 10)	DIMENSION A(20, 10), B(20, 10)
V DO 100 I=1, 20	DIMENSION AA(200), BB(200)
S DO 100 J=1, 10	EQUIVALENCE( A(1, 1), AA(1) ),
V A(I, J) = S*B(I, J)	( B(1, 1), BB(1) )
V 100 CONTINUE	V DO 100 IJ=1, 200
	V AA(IJ) = S*BB(IJ)
	V 100 CONTINUE

## 例 2 EQUIVALENCE を用いた一重ループ

例 1 とは異なって、DIMENSION宣言で宣言された寸法のすべてを使っていない場合は、リスト・ベクトルLVを使って一重ループ化する。

例 2 のLVは、任意の配列要素 A(I, J) の先頭番地 A(1, 1) からの変移を表している。変換前のベクトル長はNであるが、変換後のそれはINDMX=N\*Mである。

```

DIMENSION A(10, 20), B(10, 20)
S      DO 100 J=1, M
V      DO 100 I=1, N
V          A(I, J) = S*B(I, J)
V 100 CONTINUE

```

ただし、M<20, N<10

⇒

```

DIMENSION A(10, 20), B(10, 20)
DIMENSION AA(200), BB(200)
EQUIVALENCE( A(1, 1), AA(1) ),
              ( B(1, 1), BB(1) )

```

```

IND = 0
S      DO 100 J=1, M
V      DO 100 I=1, N
V          IND = IND + 1
V          LV(IND) = 10*(I-1) + J
V 100 CONTINUE
INDMX = IND
V      DO 200 IND=1, INDMX
V          I = LV(IND)
V          AA(I) = S*BB(I)
V 200 CONTINUE

```

リスト  
作成

配列A

( 1, 1)
( 2, 1)
.....
(10, 1)
( 1, 2)
( 2, 2)
.....
( I, J)
.....
(10, 20)

配列A A

( 1 )
( 2 )
.....
( 10 )
( 11 )
( 12 )
.....
(10*(I-1)+J)
.....
( 200 )

### (3) 最内ループを書き下す

最内ループのベクトル長が3と短くベクトル実行しても実行効率が悪い場合には、最内ループを例のように書き下します（ただし、最適化制御行 UNROLL を指定することで可能もあります）。この変更によって、二重ループはJに関する一重ループとなり、Jに関してベクトル化されベクトル長は100となります。

```

M      DO 200 J=1, 100
V          A1 = A(J)
V          S = 0
V          DO 100 I=1, 3
V              FUN = A1*B(I)
V              S = S + W(I)*FUN
V 100    CONTINUE
M          C(J) = S*C(J)
S 200    CONTINUE

```

⇒

```

V      DO 200 J=1, 100
V          S = A(J)*( W(1)*B(1)
V                           +W(2)*B(2)
V                           +W(3)*B(3) )
V          C(J) = S*C(J)
V 200    CONTINUE

```

## (4) 外側ループのアンローリング

ループの繰り返し回数を減らして効率をあげることをループのアンローリングと呼びます。下に示すように、外部ループのアンローリングを行えば最内ループの演算密度が高まり、ロード／ストア・パイプライン、演算パイプラインの有効利用が図られるために効率が上がります（これも最適化制御行UNROLLを指定することで可能な場合あります）。

S DO 100 J=1, M	S DO 100 J=1, M, 2
V DO 100 I=1, N	V DO 100 I=1, N
V A(I) = A(I) + B(I, J)*C(J)	V A(I) = A(I) + B(I, J)*C(J)
V 100 CONTINUE	V 1 + B(I, J+1)*C(J+1)
	V 100 CONTINUE

逆に、下の例に示すような内側のループのアンローリングは、最内ループの演算密度は高くなっているが、ベクトル長が半減するので、処理が遅くなることがあります。

S DO 100 J=1, M	S DO 100 J=1, M
V DO 100 I=1, N	V DO 100 I=1, N, 2
V A(I) = A(I) + B(I, J)*C(J)	V A(I) = A(I) + B(I, J)*C(J)
V 100 CONTINUE	V A(I+1) = A(I+1) + 1 B(I+1, J)*C(J)
	V 100 CONTINUE

3.2.5 ベクトル化する上での留意点

ここでは、プログラムをベクトル化する上での留意点について述べます。

## (1) DO ループ中の中間変数

V       DO 100 IND=1, INDMX	V       DO 100 IND=1, INDMX
V        I = LV(IND)	V        I = LV(IND)
V        DX(I) = VX(I)*DT(I)	⇒      V        DX = VX(I)*DT(I)
V        XN(I) = XO(I) + DX(I)	V        XN(I) = XO(I) + DX
V   100 CONTINUE	V   100 CONTINUE

上の例のDXのようにDO ループ外では定義引用されない中間変数は、無理に配列化しないでスカラ変数のままにしておいて、コンパイラにDX(I)に相当する作業用配列を内部発生させてベクトル化したほうが実行効率がよくなります。特に例のように配列DXのインデックスがリストベクトルLVによって間接的に示されるときはスカラ変数にしたほうが効率がよくなります。

## (2) DO ループ内にベクトル部分とスカラ部分が混在する

S       DO 100 I=1, N-1	S       DO 100 I=1, N-1
S        A(I) = S1*B(I)	S        A(I) = S1*B(I)
S        C(I) = S2*A(I) + S3	S        B(I+1) = A(I) + S5
S        D(I) = S4*C(I)	S   100 CONTINUE
S        B(I+1) = A(I) + S5	V       DO 200 I=1, N-1
S   100 CONTINUE	V        C(I) = S2*A(I) + S3
	V        D(I) = S4*C(I)
	V   200 CONTINUE

上の例では、配列Bが回帰演算のためにベクトル化されない。このようにループ内にベクトル化できる部分とできない部分が混在する場合には、ベクトル化できる部分とできない部分にループを分割したほうが効率がよい場合があります。

## (3) メモリ・アクセスの競合

例 1

V	DO 100 I=1, N	V	DO 100 I=1, N
V	CC(I) = C(ID(I))	V	100 CONTINUE
V	DO 200 I=1, N	V	A(I) = S
V	A(I) = S	V	B(I) = A(I)*C(ID(I))
V	B(I) = A(I)*C(ID(I))	V	D(I) = B(I)
V	D(I) = B(I)	V	200 CONTINUE
V	100 CONTINUE		

リスト・ベクトルID(I)が同じ値になることが多い場合には、C(ID(I))は同じメモリ領域をアクセスするためにメモリ競合が起こり効率が良くありません。特に、何箇所かでC(ID(I))を参照すると、さらに効率の悪いプログラムとなってしまいます。

そこで、前もってテーブルを作成しておき、演算処理部分ではテーブルを引用する方式にすれば、200 のループの実行効率が改善されます。

例 2

REAL*8 A(128, 128), B(128)	REAL*8 A(129, 128), B(128)
S DO 100 I=2, 128	S DO 100 I=2, 128
V DO 100 J=1, N	V DO 100 J=1, N
V A(I, J) = S*A(I-1, J)+B(J)	V A(I, J) = S*A(I-1, J)+B(J)
V 100 CONTINUE	V 100 CONTINUE

メモリ上のとびとびのデータをアクセスする場合、そのデータ間隔が適当でないとメモリ・バンクの競合を起こし性能が低下してしまいます。

一般に、2次元配列上を行方向に計算を進める場合、配列の第一寸法宣言子の値を以下のように定めるとメモリバンクの競合が生じません。

type * m	REAL * 4 INTEGER * 4 LOGICAL * 4	REAL * 8 COMPLEX * 8 COMPLEX * 16
k	4 n + 2	2 n + 1

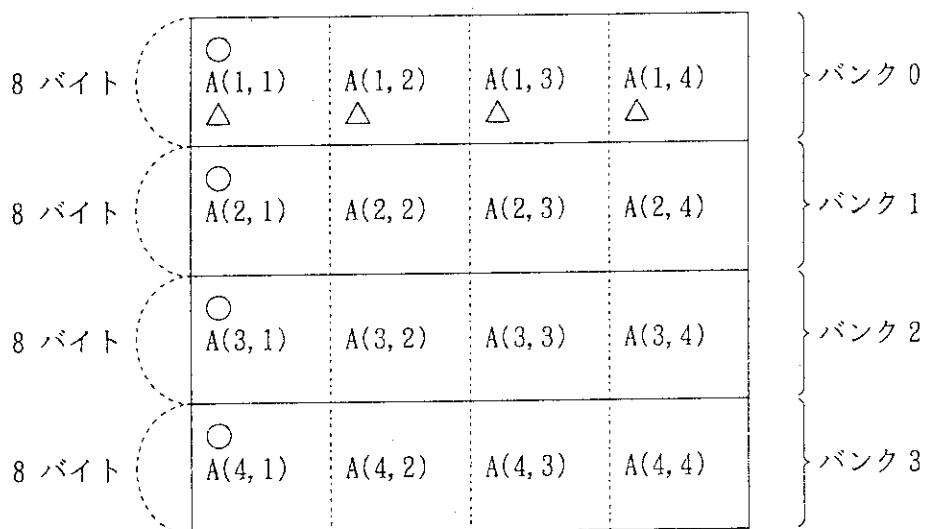
k : 第一寸法宣言子  
A (k, m)  
n : 正整数

メモリバンクの競合とは、次のような現象です。メモリバンクは、主記憶を各々独立に動作可能な複数個の部分に分割する手法です。分割された各々の部分をバンクと呼び、各々のバンクは、独立してアクセス可能です。

例えばメモリバンク数を 4 とした場合に例 3 のプログラムを実行するとします。

```
例 3  REAL*8 A(4, 4)
      DO 10 I = 1, 4
           A(I, J) = 0. D0
      10 CONTINUE
```

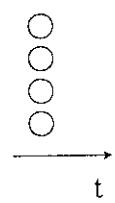
配列のメモリ上の配置は次のようにになります。



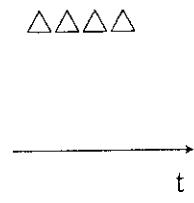
例 3 のプログラムを実行すると、メモリ上の○印のところをアクセスすることになります。各アクセス点は全て独立しています。ところが、例 4 のプログラムを実行すると、△印のところをアクセスすることになります。

```
例 4  REAL*8 A(4, 4)
      DO 10 J = 1, 4
           A(I, J) = 0. D0
      10 CONTINUE
```

同じバンクに属する配列要素を順次アクセスしていくことになり、アクセス時間は例 3 の 4 倍かかることになります。



例 1 のアクセス時間



例 2 のアクセス時間

1 つの独立部分内でメモリアクセスが集中することを、バンクの競合と言います。配列の添字の順序を変更せずにバンクの競合を避けるには、配列の宣言を変更することで可能です。例 4 の場合、例 5 のように配列を宣言するとバンクの競合が生じません。

```

例 5   REAL*8 A(5, 4)
      DO 10 J = 1, 4
            A(I, J) = 0. D0
10 CONTINUE

```

$\star$ A(1, 1)	A(5, 1) ダミー	A(4, 2)	A(3, 3)	A(2, 4)	} バンク 0
A(2, 1)	$\star$ A(1, 2)	A(5, 2) ダミー	A(4, 3)	A(3, 4)	
A(3, 1)	A(2, 2)	$\star$ A(1, 3)	A(5, 3) ダミー	A(4, 4)	
A(4, 1)	A(3, 2)	A(2, 3)	$\star$ A(1, 4)	A(5, 4) ダミー	

もちろん、この場合に例 3 のようなアクセスをしても、メモリバンクの競合は起こりません。

## 4. アルゴリズムの変更を伴うベクトル化

ベクトル計算機でプログラムを高速実行させるための方法としては、

- ① コンパイラの自動ベクトル化機能を利用
- ② ベクトル化されないDOループの再構成
- ③ 計算スキーム、アルゴリズムの変更

などが考えられます。 プログラムの実行時の C P U 時間の大部分を費やしている部分が上記の①②の方法によって高速化の効果が見込めないときは③のレベルの高速化を考慮する必要があります。 計算スキームの変更は多くの場合プログラム全体を最初から作り直すと同程度の工数を要します。 本章では既存のプログラムを高速化する方法を扱っており、計算スキームの変更によるベクトル化については範囲外として取り扱いません。 実際のプログラムで使用しているアルゴリズムは種々のものがありますが、ここでは幾つかの代表的なものに絞って述べます。 それらは 1) ポイント S O R 法のベクトル化、  
2) ガウス・チェビシェフ型積分のベクトル化、3) 3重対角方程式のベクトル化、  
4) F A C R ( Fourier Analysis and Cyclic Reduction ) 法を使用したベクトル化、および、5) モンテカルロ法のベクトル化です。 ポイント S O R 法の例では、マトリクスの逐次解法として頻繁に利用されている S O R 法のプログラムをポイント O D D - E V E N 法(4.1.2)、H Y P E R - P L A N E 法(4.1.3) を用いてベクトル化する方法を示します。 ガウス・チェビシェフ型数値積分(4.2) の例では作業用配列又はリストベクトルを用いて D O ループを一重化して高速化させる方法について述べます。 3重対角方程式(4.3) の例では線形 3重対角方程式を複数組同時に解く方法について述べます。 F A C R 法の例では F F T ( Fast Fourier Transform ) と C R ( Cyclic Reduction ) 法を橜円形偏微分方程式(境界値問題)に適用(4.4) する例について述べます。 モンテカルロ法(4.5) の例では、最初に一般的なベクトル化の考え方を示し、その後で実際に中性子挙動解析に応用した事例について述べます。

### 4.1 ポイント S O R 法をベクトル化するには

偏微分方程式を有限要素法や差分法で離散化すると得られる係数行列は、密行列、帯行列、スペース行列となります。 この中で密行列、帯行列の場合は通常ガウスの消去法、コレスキー法に代表される直接法で解かれています。 ベクトル化の観点からは、メモリ・アクセスの効率が良いコレスキー法が優れています。

一方、得られる係数行列が大次元でスペースな場合は、1) 適用が簡便である、2) 解法途中でスペース性が保存される、3) 解の近似値がわかっているときは、直接法で解くよりも効率的に解が求められる、などの理由により反復法が使用されることが多くなります。 ここでは繰り返し法として実際のコードでよく使用されているポイント S O R 法(逐次過大緩和法)をベクトル化する方法について述べます。

#### 4.1.1 ポイントSOR法とは

ポアソン方程式を矩形領域で差分近似すると(4.1)に示す式になります。

$$\phi^{*}_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1} - 4\phi_{i,j} = q_{i,j} \quad (4.1)$$

電場の解析の場合には、 $\phi_{i,j}$ は節点(i,j)でのポテンシャル、 $q_{i,j}$ は節点(i,j)に関した電荷です。

上式の解法にポイントSOR法を適用すると、まず全ての節点において適当な $\phi^0$ の初期値を与えます。次に、(4.1)式と $\phi_{i,j}$ の値より $\phi_{i,j}^1$ を求めます。これを $\phi_{i,j}^1$ とします。このようにして第nステップでの $\phi_{i,j}^n$ と $\phi_{i,j}^{n-1}$ との残差 $\phi_{i,j}^*(n)$ が収束条件を満たすまで繰り返されます。収束条件が満たされないときは“新しい”計算ステップでの値は“古い”値と $\phi^*$ との重み平均で求められます。

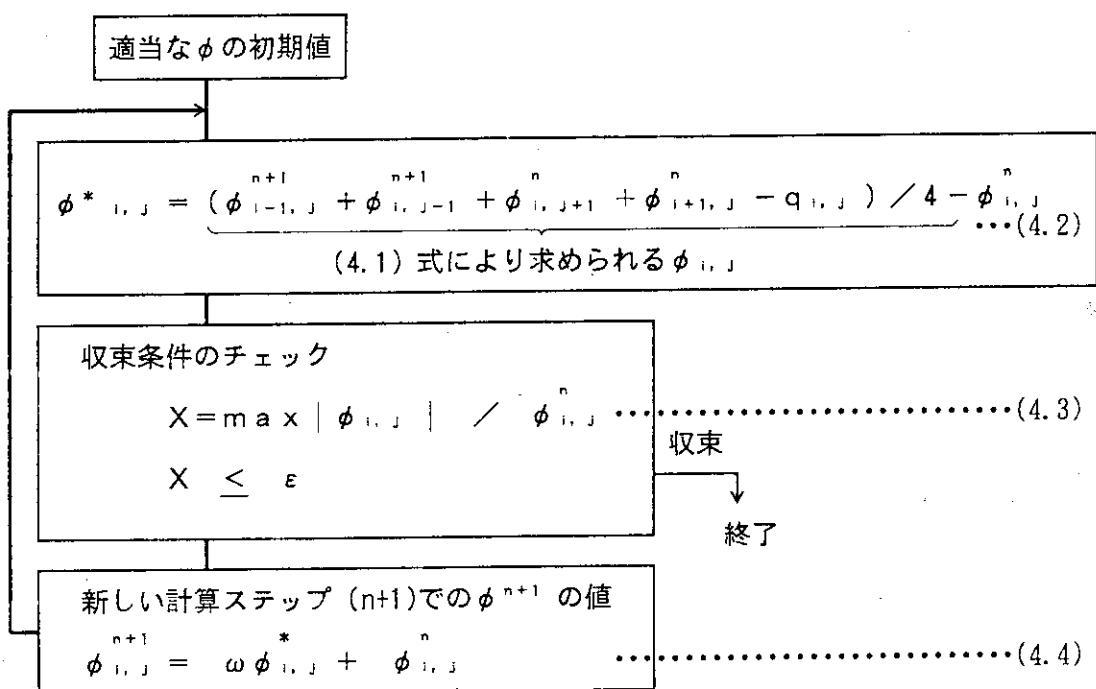


図 4.1 ポイントSOR法の解法

ポイントSOR法の計算の核となる部分は(4.2)式の $\phi_{i,j}^1$ を求める部分で回帰演算の形をしています。この逐次計算を避けるためには①ポイントODD-EVEN法、②HYPER PLANE法、③ポイントJACOBI法のいずれかを採用します。

## 4.1.2 ポイント ODD-EVEN 法によるベクトル化

図4.2に示すように、境界節点を除いた節点( $i, j$ )の組み合わせを考え、 $i + j = \text{偶数}$ (even)と $i + j = \text{奇数}$ (odd)の点の2つのグループに分けます。はじめに  $i + j = \text{odd}$  の点についてのみ  $\phi_{i,j}^{n+1}$  の計算を行います。このとき、隣接する4点の  $\phi$  の値はひとつ前の計算ステップで計算した  $\phi_{i,j}^n$  を使用します。次に  $i + j = \text{even}$  の点についての  $\phi_{i,j}^{n+1}$  の計算には新しく計算された隣接する奇数点についての  $\phi_{i,j}^{n+1}$  の値を使用します。ODD-EVEN法を用いるとベクトル長は  $i$  方向あるいは  $j$  方向の節点数の約半分となります。このため、節点数が少ないとときは  $i$  方向又は  $j$  方向に1次元化したベクトル長を大きくすることによってベクトル計算機で効率よく処理されます。図4.5ではポイント ODD-EVEN 法のコーディング例を示しています。

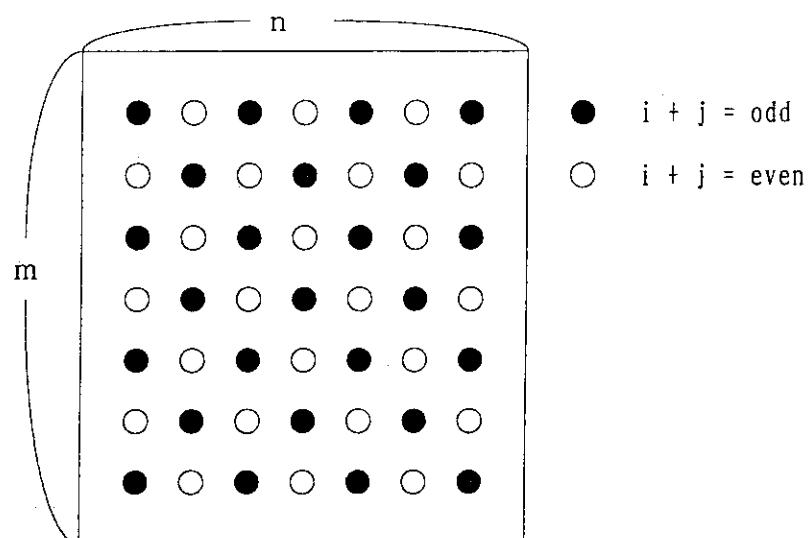


図 4.2 ODD-EVEN 法のメッシュ点の分類

#### 4.1.3 H Y P E R - P L A N E 法によるベクトル化

この方法は図4.3に示す様に $\phi^{n+1}$ の計算を節点の斜め方向に進めて並列処理を実現するものです。この方法ではベクトル長が1から‘i方向またはj方向の最大節点’と変化し、平均するとi方向またはj方向の節点数の半分となります。また、 $\phi^{n+1}$ の計算はその順序を変えただけであるため、加速パラメータ $\omega$ はポイントSOR法の場合と同じです。図4.6はHYPERPLANE法のコーディング例を示しています。

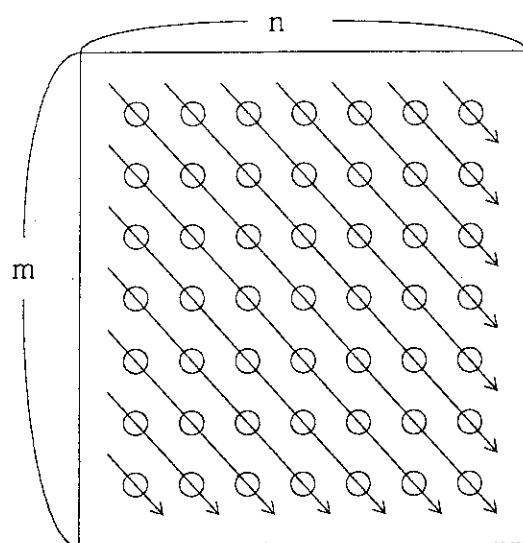


図 4.3 HYPERPLANE法のメッシュ点の分類

```

SUBROUTINE SOR(U, Q, OMEGA, M, N)
REAL    U(M, N), Q(M, N)

C
S      DO 10 J=2, N-1
M          DO 10 I=2, M-1
M              US=0.25*( U(I-1, J)+U(I+1, J)+U(I, J-1)+U(I, J+1)-Q(I, J) )
&                  -U(I, J)
S              U(I, J)=U(I, J)+OMEGA*US
V      10 CONTINUE
      RETURN
      END

```

図 4.4 S O R 法による計算

```

SUBROUTINE ODDEVN(U, Q, OMEGA, M, N)
REAL    U(M, N), Q(M, N)

C
S      DO 30 J=2, N-1
V          DO 10 I=2, M-1, 2
V              US=0.25*( U(I-1, J)+U(I+1, J)+U(I, J-1)+U(I, J+1)-Q(I, J) )
&                  -U(I, J)
V              U(I, J)=U(I, J)+OMEGA*US
V      10 CONTINUE
V          DO 20 I=3, M-1, 2
V              US=0.25*( U(I-1, J)+U(I+1, J)+U(I, J-1)+U(I, J+1)-Q(I, J) )
&                  -U(I, J)
V              U(I, J)=U(I, J)+OMEGA*US
V      20 CONTINUE
S      30 CONTINUE
C
      RETURN
      END

```

図 4.5 ポイント O D D - E V E N 法による計算

```

SUBROUTINE HYPER1(U, Q, OMEGA, M, N)
REAL    U(M, N), Q(M, N)
C
I=1
J=1
DO 10 K=4, M+N-2
  IF(I.LT.M-1) THEN
    I=I+1
  ELSE
    J=J+1
  ENDIF
  LMAX=MIN(N-1, K-2)-MAX(2, K-M+1)
V   DO 10 L=0, LMAX
V     US=0.25*( U(I-L-1, J+L)+U(I-L+1, J+L)+U(I-L, J+L-1)
V       &           +U(I-L, J+L+1)-Q(I-L, J-L) )-U(I-L, J+L)
V     U(I-L, J+L)=U(I-L, J+L)+OMEGA*US
V   10 CONTINUE
C
RETURN
END

SUBROUTINE HYPER2(U, Q, OMEGA, M, N)
REAL    U(M, N), Q(M, N)
C
S   DO 10 I=4, M+N-2
  L=I-M
V   DO 10 K=MAX(2, I-M+1, MIN(N-1, I-2))+1
V     US=0.25*( U(L-1, K)+U(L+1, K)+U(L-M, K)
V       &           +U(L+M, K)-Q(L, K) )-U(L, K)
V     U(L, K)=U(L, K)+OMEGA*US
V   10 CONTINUE
C
RETURN
END

```

図 4.6 H Y P E R P L A N E 法による計算

#### 4.2 多重積分（ガウス・チェビシェフ型積分）のベクトル化

インダクタンスの計算などでは、しばしば多重積分が現れます。形状の複雑さのため多重積分の解析的な取扱いは難しいことがあります。この場合には数値積分法、モンテカルロ法によって積分することが出来ます。

多重積分を数値積分法で計算するとき、分点の数が多い場合には計算すべき点の数が飛躍的に多くなります。このような場合にはモンテカルロ法を用いて計算することができます。

一方、空間領域を小領域に分割して、その小領域を積分領域として多重積分の計算をすることがあります。この場合には、積分関数の振舞いは一般におだやかになり、従って分点の数も少なく出来ます。計算すべき点の数も少ないので、数値積分法が用いられます。

ここでは、数値積分法で少ない分点で多重積分の計算を行う場合について考えることにします。

数値積分法には、分点を等間隔にとり単純な和として求めるニュートン・コツ系と分点を不等間隔にとって精度を高めるガウス・チェビシェフ系の2つの系列がありますが、ここではガウス・チェビシェフ系の数値積分法によって多重積分の計算を行う場合を例にあげています。

例えば、四重積分

$$\int \int \int \int P(u_1)Q(u_2)R(v_1)S(v_2) du_1 du_2 dv_1 dv_2$$

は、一般公式で次のようにになります。

$$\sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{l=1}^n W P_i W Q_j W R_k W S_l P_i Q_j R_k S_l$$

ここで、 $P_i = P(u_{1i})$ ,  $Q_j = Q(u_{2j})$ ,  $R_k = R(v_{1k})$ ,  $S_l = S(v_{2l})$

$u_{1i}$ ,  $u_{2j}$ ,  $v_{1k}$ ,  $v_{2l}$  : 区分をNヶに分割したときの分点の座標

$W P_i W Q_j W R_k W S_l$  : 分点における関数値の重み

図4.7 に、三重積分

$$\int \int \int A(x)(B(y) - C(z)) \cdot D(z) dx dy dz$$

をN分点のガウス・チェビシェフ系の積分で書き下したコーディング例を示します。図から分かるように、三重積分と3重のDOループになり、分点の数Nが各DOループの繰り返し回数、すなわちベクトル長になります。

前述のように、数値積分法によって多重積分の計算を行う場合には、分割された小領域で積分の計算を行うために分点の数が少なくなります。従って、最内のDOループのベクトル長が短く、ベクトル化されても実行効率は良くありません。

このような場合には、作業用配列を導入して、多重のDOループの一重化を行いベクトル長を大きくして実行効率の改善をはかることが出来ます。しかしながら、一重化に伴

い、作業用配列を作成するオーバーヘッドや演算数が増加するなどの実行効率を悪くする要因も新たに発生します。従って、プログラムのベクトル化においては、この両面の要素を考え合わせた上で、多重DOループの一重化を行うか決める必要があります。図4.8では、3重積分に由来する3重DOループがVA、VBなどの作業用配列を用いて一重化されています。

ベクトル長はNから $N^3$ と長くなっています。DO 100のループが作業用配列を作成するオーバーヘッドになります。また、乗算の演算数は $3N^3 + N^2 + N$ から $5N^3$ と増加していますが、高々 1.67 倍です。オーバーヘッド部分が、総和計算部分のCPU時間に比べて小さければ十分に採算のとれるベクトル化です。

また参考として、リスト・ベクトルを用いて多重DOループを一重化する方法も図4.9に示しました。

この場合、ベクトル長は $N^3$ 、乗算の演算数は $5N^3$ となり、リスト・ベクトルであるLI, LJ, LKを作成する部分がオーバーヘッドになります。

作業用配列を用いる方法とリスト・ベクトルを用いる方法と比較すると、ベクトル長と演算数は同じです。しかし、リスト・ベクトルの場合は演算速度自体が遅いだけではなく、図4.10に示すようにリスト・ベクトルの内容が同一であることが多いため、配列であるA(LI(IDX))を引用する場合、同じ要素を引用することが多くなります。このためにメモリ・アクセスの競合を起こし、効率は更に悪くなります。

従って、作業用配列を多く用いるためにメモリ使用量が増加しますが、この増加量が問題とならないならば作業用配列を用いて一重化する方法が良いでしょう。

```

S1=0.
M   DO 100 I=1, N
S     TA=A(I)
S     SJ=0.
M     DO 110 J=1, N
S       TB=B(J)*
S       SK=0.
V     DO 120 K=1, N
V       TBC=TB-C(K)
V       S =TA*TBC*D(K)
V       SK =SK+S *WZ(J)
V 120    CONTINUE
M     SJ=SJ+SK*WY(J)
S 110  CONTINUE
M     SI=SI+SJ*WX(I)
S 100 CONTINUE

```

図 4.7 三重積分

```

DIMENSION VA(N), VB(N), VC(N)VD(N), WXYZ(N)
C
IDX=0
DO 100 I=1, N
DO 100 J=1, N
DO 100 K=1, N
IDX=IDX+1
VA(IDX) =A(I)
VB(IDX) =B(J)
VC(IDX) =C(K)
VD(IDX) =D(K)
WXYZ(IDX)=WX(I)*WY(J)*WZ(K)
100 CONTINUE
C
SI=0.
DO 200 IDX=1, N*N*N
TBC=VB(IDX)-VC(IDX)
SI =SI+WXYZ(IDX)*VA(IDX)*TBS*VD(IDX)
200 CONTINUE

```

図 4.8 三重DOループの一重化（作業配列）

```

C      DIMENSION  LI(N), LJ(N), LK(N)
C      IDX=0
V      DO 100 I=1, N
V      DO 100 J=1, N
V      DO 100 K=1, N
V          IDX=IDX+1
V          LI(IDX)=I
V          LJ(IDX)=J
V          LK(IDX)=K
V 100 CONTINUE
C
C      S1=0.
V      DO 200 IDX=1, N*N*N
V          I=LI(IDX)
V          J=LJ(IDX)
V          K=LK(IDX)
V          WXYZ=WX(I)*WY(J)*WZ(K)
V          TBC=B(J)-C(K)
V          S1 =S1+WXYZ*A(I)*TBC*D(K)
V 200 CONTINUE

```

↑  
リストベクトルを作成  
↓

図 4.9 三重DOループの一重化（リストベクトル）

IDX	LI(IDX)
1	1
⋮	⋮
$N^2$	1
$N^2+1$	2
⋮	⋮
$(N-1)N^2+1$	N
⋮	⋮
$N^3$	N

図 4.10 ループの一重（図4.9）に用いたリストベクトル

#### 4.3 (複数個の) 3重対角方程式のベクトル化

単純な2重ループのベクトル化に際しては、最内ループのベクトル長をできるだけ大きくする必要があることは3.2.4で説明しました。ここではその応用として、最内ループに熱伝導方程式などの線形方程式を処理する部分を含む場合のベクトル化の方法について述べます。最内ループのベクトル長が充分大きいときはそのままループの再構成を考慮するだけで良いことになります。しかし、ベクトル長が短く、かつ最内ループが独立に処理可能なときは、外側ループの処理を最内ループに取り込んだ方が効率良く解ける場合があります。以下では一次元熱伝導方程式、輸送方程式などを差分法で表現したときに頻繁に現れる線形3重対角方程式を用いて説明します。

軽水炉の安全性解析コードRELAP5コードでは熱構造体(heat structure)を一次元熱伝導問題として解いており、差分化することにより最終的には各構造体ごとに3重対角方程式を解いています。

$$\begin{pmatrix} b_1 & c_1 \\ a_2 & b_2 & c_2 \\ & \ddots & \ddots & \ddots \\ & & & c_{N-1} \\ a_N & b_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}$$

図 4.11 3重対角方程式

RELAP5の例では、各構造体ごとの差分点の個数(N)はまちまちで3~8と少なく、一方では熱構造体の総数(M)は40~50です(図4.12)。通常3重対角方程式の解法ではガウスの消去法が用いられ、前進消去、後退代入のステップがあります。その際には3重対角行列のそれぞれの要素は直前に計算された値を引用しており、完全な逐次計算法であるため並列計算法(ベクトル処理)には不適です。

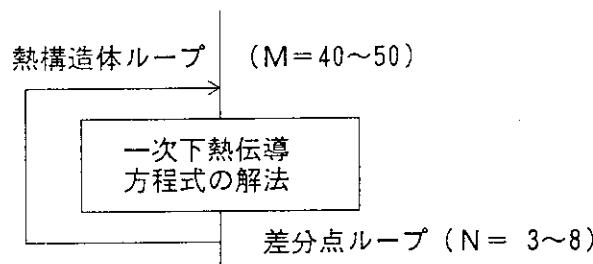


図 4.12 RELAP5の熱伝導解法ループ

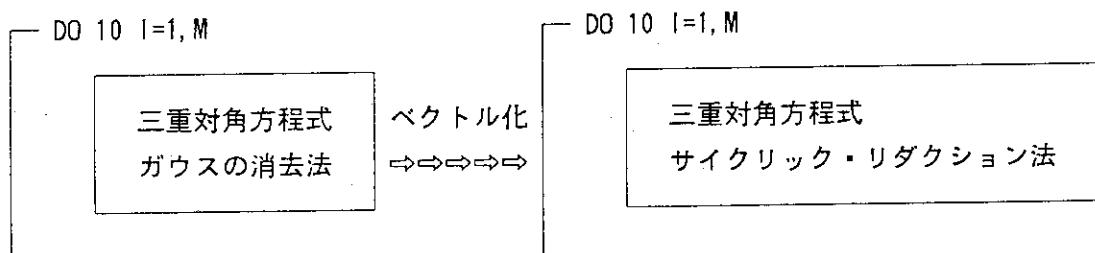
#### 4.3.1 3重対角方程式の並列解法

3重対角方程式の並列解法としてはサイクリック・リダクション法、リカーシブ・ダブルリング法、パラレル・クラーメル法などが知られており、これらの並列解法の数値実験を行った結果が文献 (JAERI-M 9703) に詳しく述べられています。各種の並列解法の中で、サイクリック・リダクション法は、C R A Yのスーパーコンピュータや富士通のV P の科学技術計算サブルーチンとして採用されており、未知変数Nが大きい場合にはかなり有効な手段です。しかし、サイクリック・リダクション法には、1) 計算途中でD O ループの繰り返し回数(ベクトル長)が繰り返しの度に半分になる、2) メモリ参照の距離が繰返し計算の度に2倍となっていく等の弱点があるので、これらに留意して使う必要があります。

#### 4.3.2 複数組の3重対角方程式のベクトル化

実際上の問題ではR E L A P 5の例の様に、独立したM組の3重対角方程式を解く場合も多くなります。この様な場合には2通りのベクトル化の手法(図4.13)が適用出来ます。①の方法は前節で述べたサイクリック・リダクション法に基づく解法ルーチンをM回呼び出すものです。②の方法は、ガウスの消去法の各計算ステップにベクトル長Mのループを持ち込んでベクトル処理させるもので、線形方程式の多重処理と呼ぶことにします。図4.16は多重処理の場合のコーディング例を示しています。M>NでかつNが比較的小さい(N<10)場合には②の多重処理が適しています。

① 並列解法をM回呼び出す。



② 多重処理

ガウスの消去法の中にMに関するループを持ち込む。

図 4.13 M組の3重対角方程式のベクトル化手法

3重対角方程式の解法以外に、複数組のFFTを同時に解く場合などにも、②の多重処理が適用できます。4.4では橜円形の偏微分方程式をFFTとガウスの消去法を組み合わせて高速に解く場合について述べます。

次に多重処理を行う場合の留意点について述べます。まず、多重処理ではメモリ使用量が1次元から2次元に増加します。更に2次元化するときには2次元配列A(i,j)のインデックスi方向にMに関するループを構成します。こうすることにより連続領域を参照するためベクトル計算機での処理効率が上がります。

これまでM組の3重対角方程式の全ての変数の個数が一定(M)であるものとして進めてきました。変数の個数が異なる場合には、変数の個数の最大のものをNとし、Nより小さい次数の方程式の場合はNに満たない部分の主対角項と解ベクトル部分に‘1’を補います。更にサブ対角項には‘0’を設定すると良いでしょう。(図4.14)

$$\begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_k & b_k & c_k \\ & & & 0 & 1 & 0 \\ & & & & 0 & 1 & 0 \\ & & & & & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \\ \vdots \\ y_n \end{bmatrix}$$

図 4.14 次数がNに満たない方程式の係数設定

$$\begin{array}{c} \xleftarrow{\quad N \quad} \\ \uparrow \quad \downarrow \\ \left[ \begin{array}{ccccc} & & & & \\ & \ddots & & & \\ & & 0 & & \\ & & & \ddots & \\ & & & & 0 \end{array} \right] \left[ \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right] = \left[ \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right] \end{array} \quad \text{次元Nの3重対角方程式}$$

M組

$$\left[ \begin{array}{ccccc} & & & & \\ & \ddots & & & \\ & & 0 & & \\ & & & \ddots & \\ & & & & 0 \end{array} \right] \left[ \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right] = \left[ \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right]$$

$$\left[ \begin{array}{ccccc} & & & & \\ & \ddots & & & \\ & & 0 & & \\ & & & \ddots & \\ & & & & 0 \end{array} \right] \left[ \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right] = \left[ \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right]$$

図 4.15 次元Nを持つM組の3重対角方程式

```

SUBROUTINE TRI( A, B, C, X, Y, N, M )
C *** THIS ROUTINE SOLVES M SYSTEMS OF TRIDIAGONAL EQUATIONS ***
C
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(M,N), B(M,N), C(M,N), X(M,N), Y(M,N)
C
      DO 20 K=1, N-1
          DO 10 L=1, M
              C(L,K) = C(L,K)/B(L,K)
              Y(L,K) = Y(L,K)/B(L,K)
              B(L,K+1)=B(L,K+1)*A(L,K+1)*C(L,K)
              Y(L,K+1)=Y(L,K+1)*A(L,K+1)*Y(L,K)
10      CONTINUE
20      CONTINUE
C
      DO 30 L=1, M
          X(L,N)=Y(L,N)/B(L,N)
30      CONTINUE
C
      DO 50 K=N-1, 1, -1
          DO 40 L=1, M
              X(L,K)=Y(L,K)-C(L,K)*X(L,K+1)
40      CONTINUE
50      CONTINUE
C
      END

```

図 4.16 M組の3重対角方程式の解法のプログラミング例

#### 4.4 楕円形方程式のベクトル化

楕円形偏微分方程式はラプラス方程式、ポアソン方程式、トカマク・プラズマのMHD平衡方程式、中性子拡散方程式をはじめとして工学上の多くの問題に現われてきます。このタイプの方程式は通常境界値問題として扱われ、系の平衡状態または定常状態を記述しています。楕円形偏微分方程式の解法は差分化することにより連立一次方程式の解法に帰着します。本章では楕円形偏微分方程式を高速に解く直接解法として知られるFACR (Fourier Analysis and Cyclie Reduction)法について説明します。例として取り上げる方程式としてはトカマク・プラズマのMHD平衡を記述しているGrad-Shafranov方程式を解く際に現れる楕円形方程式 ( $r-z$  座標系) を考え、これを固定境界値問題として解くことにします。FACR法においては、 $z$  方向にFourier 級数展開し、 $r$  方向にはサイクリック・リダクション法（或いはGauss の消去法）を適用します。（図4.17）この方法を用いるとFFT(Fast Fourier Transform)、サイクリック・リダクション法（或いは3重対角マトリックスの多重解法）などの並列処理向きの解法が積極的に利用できるため、ベクトル計算機での適用性が高くなります。

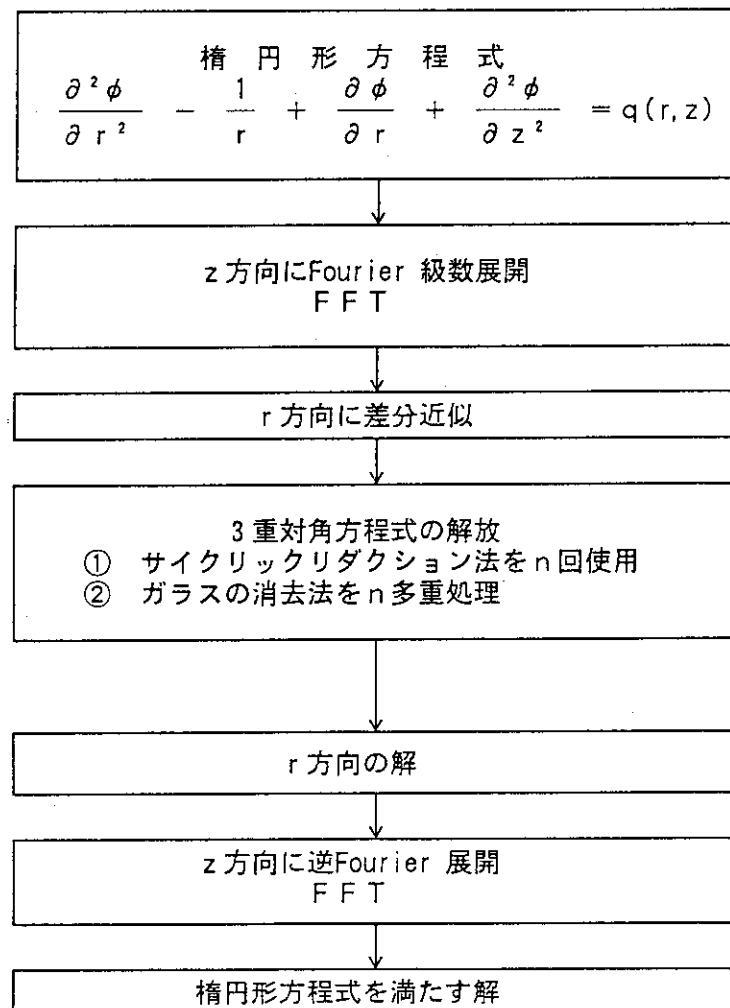


図4.17 FACR法の計算手順

#### 4.4.1 楕円形方程式とその境界条件

楕円形方程式としては  $(r - z)$  座標系で定義される方程式を例とします。

$$\frac{\partial^2 \phi}{\partial r^2} - \frac{1}{r} \cdot \frac{\partial \phi}{\partial r} + \frac{\partial^2 \phi}{\partial z^2} = q(r, z) \quad (4.5)$$

境界条件としては、境界節点上で  $\phi$  が既知（値  $\phi_B$ ）で、内部では  $q$  が既知であるとします。図 4.18 に  $r - z$  座標系での領域分割と節点番号付けを示しました。 $z$  方向の番号  $n_z$  は FFT を使用するために 2 の巾乗となります。 $r$  方向に関しては、サイクリック・リダクション法を使用するときは  $n_r$  は 2 の巾乗となり、ガウスの消去法を使用するときは任意の数値を用います。

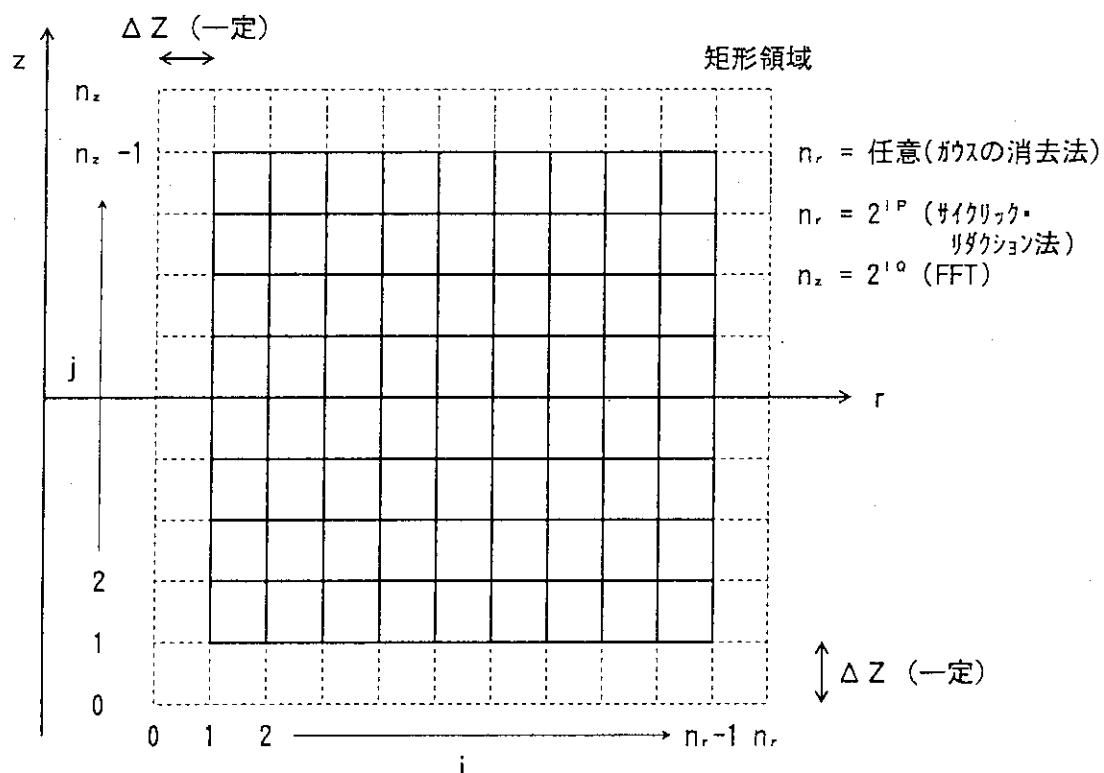


図4.18 領域分割

#### 4.4.2 F A C R 法による楕円形方程式の解法

F A C R 法においては一方向の Fourier 展開を行います。このとき境界条件として、1) 値が既知、2) 微分値が既知、3) 周期境界条件の場合、それぞれ、1) sin 展開、2) cos 展開、3) sin + cos 展開を行います。本例題では  $z$  方向の両端で既知の値  $\phi_B$  をもつことからポテンシャル  $\phi$  に対して sin 展開を行います。

$$\phi_j = \sum_{k=0}^{n_z-1} \tilde{\phi}_k \cdot \sin \frac{\pi k}{n_z} j + \phi_B \quad (4.6)$$

同様に charge 項  $q$  に対しても sin 展開を行います。

$$q_j = \sum_{k=0}^{n_z-1} \tilde{q}_k \cdot \sin \frac{\pi k}{n_z} j + q_B \quad (4.7)$$

(4.7) 式の  $q_B$  は境界での  $\phi_B$  より計算されます。

(4.6) および(4.7) 式を楕円形偏微分方程式(4.5) に代入すると Fourier 振幅  $\tilde{\phi}_k$ 、 $\tilde{q}_k$  に関する  $n_z$  組の方程式が得られます。

$$\begin{aligned} & \sum_{k=1}^{n_z-1} \left[ \left\{ \frac{\partial^2 \tilde{\phi}_k}{\partial r^2} - \frac{1}{r} \frac{\partial \tilde{\phi}_k}{\partial r} - \left( \frac{\pi k^2}{\ell} \right) \tilde{\phi}_k - \tilde{q}_k \right\} \sin \frac{\pi k}{n_z} j \right] \\ & + \left( \frac{\partial^2 \tilde{\phi}_B}{\partial r^2} - \frac{1}{r} \frac{\partial \tilde{\phi}_B}{\partial r} - q_B \right)_{k=0} = 0 \end{aligned} \quad (4.8)$$

ただし、 $\ell$  は  $z$  方向の領域の長さ、 $\ell = n_z \Delta z$

(4.8) 式がすべての点  $j$  に関して成り立つために、以下に示す関係式が成り立ちます。

$$\frac{\partial^2 \tilde{\phi}_k}{\partial r^2} - \frac{1}{r} \frac{\partial \tilde{\phi}_k}{\partial r} - \left( \frac{\pi k^2}{\ell} \right) \tilde{\phi}_k = \tilde{q}_k \quad k=1 \sim (n_z-1) \quad (4.9)$$

$$q_B = \frac{\partial^2 \phi_B}{\partial r^2} - \frac{1}{r} \cdot \frac{\partial \phi_B}{\partial r} \quad (4.10)$$

### 差分式

$$\frac{\partial^2 \phi}{\partial r^2} = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{(\Delta r)^2} \quad (4.11)$$

$$\frac{\partial \phi}{\partial r} = \frac{\phi_{i+1} - \phi_{i-1}}{2(\Delta r)}$$

差分式(4.11)を用いると(4.10)式の  $q_B$  は境界ポテンシャル  $\phi_B$  より計算されます。

$$q_B = \frac{\phi_B^{i+1} - 2\phi_B^i + \phi_B^{i-1}}{(\Delta r)^2} - \frac{1}{r_i} \cdot \frac{\phi_B^{i+1} - \phi_B^{i-1}}{2(\Delta r)} \quad (4.12)$$

- ただし  
 •  $i$  は  $r$  方向の節点番号で  $i = 1 \sim (n_r - 1)$   
 •  $\phi^i$  は  $r$  方向の  $i$  番目の境界での値

次に(4.9)式を考えます。右辺の  $\tilde{q}_k$  は境界内部の  $q$  から境界上の  $q_B$  を差し引いた値に関しての Fourier 変換の振幅です。つまり、

$$q_j - q_B = \sum_{k=1}^{n_z-1} \tilde{q}_k \cdot \sin \frac{\pi k}{n_z} j \quad (4.13)$$

(4.13)式の操作を行うことにより(4.9)式の右辺は既知となります。

(4.10)式の関係式が成り立つときは、境界上で  $\phi$  の値が 0 の境界値問題を解くことと等価となり(4.9)式がこの場合の方程式です。

(4.9)式を  $r$  方向に差分化 ( $n_r \cdot \Delta r = r$ ) すると、 $k$  番目のライン上では、

$$\begin{aligned} & -\left(1 + \frac{\Delta r}{2r_i}\right) \tilde{\phi}_k^{i-1} + \left\{2 + \left(\frac{\Delta r}{\Delta z}\right)^2 \left(\frac{\pi}{n_z}\right)^2 k^2\right\} \tilde{\phi}_k^i - \left(1 - \frac{\Delta r}{2r_i}\right) \tilde{\phi}_k^{i+1} \\ & = -(\Delta r)^2 \tilde{q}_k^i \quad i = 1 \sim (n_r - 1) \end{aligned} \quad (4.14)$$

と表わされます。一般的には以下の様に表わされます。

$$E_{i,k} \tilde{\phi}_k^{i-1} + D_{i,k} \tilde{\phi}_k^i + F_{i,k} \tilde{\phi}_k^{i+1} = \tilde{Q}_k^i$$

ただし、

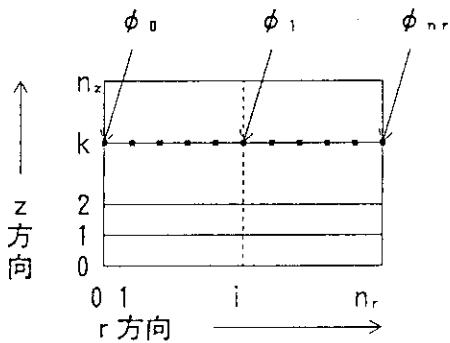
$$E_{i,k} = -\left(1 + \frac{\Delta r}{2r_i}\right)$$

$$D_{i,k} = 2 + \left(\frac{\Delta r}{\Delta z}\right)^2 \left(\frac{\pi}{n_z}\right)^2 k^2$$

$$F_{i,k} = -\left(1 - \frac{\Delta r}{2r_i}\right)$$

$$Q_{i,k} = -(\Delta r)^2 \tilde{q}_k^i$$

(4.15)



$z$  方向の  $k$  番目のライン上では  $\left[ \begin{array}{l} \tilde{\phi}_i \rightarrow \tilde{\phi}_i \\ E_{i,k} \rightarrow E_i \end{array} \right]$

$$\begin{aligned}
 E_1 \tilde{\phi}_0 + D_1 \tilde{\phi}_1 + F_1 \tilde{\phi}_2 &= Q_1 \\
 E_2 \tilde{\phi}_1 + D_2 \tilde{\phi}_2 + F_2 \tilde{\phi}_3 &= Q_2 \\
 E_3 \tilde{\phi}_2 + D_3 \tilde{\phi}_3 + F_3 \tilde{\phi}_4 &= Q_3 \\
 E_{n_r-1} \tilde{\phi}_{n_r-2} + D_{n_r-1} \tilde{\phi}_{n_r-1} + F_{n_r-1} \tilde{\phi}_{n_r} &= \tilde{Q}_{n_r-1}
 \end{aligned} \tag{4.16}$$

(4.16)式の最初の式は既知量  $E_1 \phi^0$  を含んでおり、最後の式は既知量  $F_{n_r-1} \phi_{n_r}$  を含んでいます。既知量（境界値の導入されたもの）を右辺に移すことにより(4.16)式の最初と最後の式は次の様になります。

$$D_1 \tilde{\phi}_1 + F_1 \tilde{\phi}_2 = Q_1 - E_1 \tilde{\phi}_0 \tag{4.17}$$

$$E_{n_r-1} \tilde{\phi}_{n_r-2} + D_{n_r-1} \tilde{\phi}_{n_r-1} = \tilde{Q}_{n_r-1} - F_{n_r-1} \tilde{\phi}_{n_r} \tag{4.18}$$

(4.16)式の右辺の  $\tilde{\phi}$  ( $= \tilde{\phi}_k^0$ ) は、 $r$  方向の左端 ( $i = 0$ ) での既知量に関するFourier 変換後の  $k$  番目の振幅で次の様にして計算されます。つまり、 $z$  方向の両端での既知量  $\tilde{\phi}_B^0$  を、 $i = 0$  での境界上の他の  $\tilde{\phi}_j^0$  から差し引いた値のFourier 変換の  $k$  番目の振幅が  $\tilde{\phi}^0$  ( $= \tilde{\phi}_k^0$ ) となります。



$$\begin{aligned}\phi_j^0 - \phi_B^0 &= \sum_{k=0}^{n_r-1} \tilde{\phi}_k^0 \cdot \sin \frac{\pi k}{n_r} j \quad (\text{左端}) \\ \phi_{j'}^n - \phi_B^n &= \sum_{k=0}^{n_r-1} \tilde{\phi}_k^n \cdot \sin \frac{\pi k}{n_r} j \quad (\text{右端})\end{aligned}\quad (4.19)$$

(4.17)、(4.18)式を考慮して(4.16)式をマトリクス表示すると次の様になります。

$$\left[ \begin{array}{ccccc} D_1 & F_1 & & & \\ E_2 & D_2 & F_2 & & \\ E_3 & D_3 & F_3 & & \\ & & & \diagdown & \\ & E_{n_r-1} & D_{n_r-1} & & \end{array} \right] \cdot \left[ \begin{array}{c} \tilde{\phi}_1 \\ \tilde{\phi}_2 \\ \tilde{\phi}_3 \\ \vdots \\ \tilde{\phi}_{n_r-1} \end{array} \right] = \left[ \begin{array}{c} \tilde{Q}_1 - E_1 \tilde{\phi}_0 \\ \tilde{Q}_2 \\ \tilde{Q}_3 \\ \vdots \\ \tilde{Q}_{n_r-2} \\ \tilde{Q}_{n_r-1} - F_{n_r-1} \tilde{\phi}_{n_r} \end{array} \right] \quad (4.20)$$

左辺の  $\tilde{\phi}$  に掛かる行列は 3 重対角行列となっています。

(4.20) 式は、Fourier 変換のすべての振幅に関して解くため、前節で述べた 3 重対角方程式の多重処理の方法を使用できます。

境界内部の全ての点における  $\tilde{\phi}$  が既知となったならば(4.6) 式に示す様に  $\tilde{\phi}_k$  を入力して Fourier 逆変換を行い  $\phi_j - \phi_B$  を得ます。

$$\phi_j - \phi_B \leftarrow \sum_{k=0}^{n_r-1} \tilde{\phi}_k \cdot \sin \frac{\pi k}{n_r} j \quad (4.21)$$

境界内部の各節点における  $\phi_j$  は、境界での  $\phi_B$  を加えることによって得られます。

#### 4.5 モンテカルロ法のベクトル化

モンテカルロ法の適用は自然現象以外にも積分問題などにも応用することができます。しかし、ここでは基本概念として自然現象を念頭に置き、さらにその中でも粒子のランダム・ウォークを取り扱うプログラムを対象とします。

とりわけ、原子炉における中性子輸送や核融合炉における中性粒子の挙動などにおいては、単なるランダム・ウォークの問題に加え、粒子の生成消滅あるいは体系外への飛び出しなど複雑な要素が絡んできます。これらの要素はモンテカルロ・コードをベクトル化した場合に非常に大きな問題となります。問題点に関しての詳しい内容は4.5.2で述べることにします。

通常のモンテカルロプログラムでは1個の粒子を計算終了まで追跡した後、次の1個を発生させてこれを追跡します。追跡する粒子の総数は、数百個以上の場合が多くあります。ベクトル化に際しては、これらの数百個以上の粒子に対しモンテカルロ計算過程に含まれる各ステップの計算を順次行う方式を採用します。すなわち原理的にはオリジナルプログラムにおけるモンテカルロ計算のDOループとその外側の粒子数のDOループを交換します。

このような変更が許されるにはモンテカルロ計算における1粒子に対する計算結果がその後の粒子の計算に影響を及ぼさないということが必要です。この条件を満たすプログラムでは、同時に処理される数百個の粒子はすべて互いに独立となるため、各々の粒子について並列処理（ベクトル化）が可能となります。逆に各粒子が互いに独立とならないようなモンテカルロプログラムの場合、並列処理は不可能です。

##### 4.5.1 中性粒子挙動解析用モンテカルロコードの構造概要

本プログラムは核融合炉におけるプラズマ・ダイバータ・チャンバ設計用のプログラムの一部で、全体の構造は図4.19の通りです。

モンテカルロ計算ではしばしば粒子の“世代”と言う言葉が用いられます。ここでは図4.19のモンテカルロ計算部を1回実行し、そこで発生する全粒子の履歴を追跡することで“一世代の終了”となります。このモンテカルロ計算部は全計算時間の大部分を占めており、本節ではモンテカルロ計算部のベクトル化について解説します。

#### 4.5.2 ベクトル化の方法及び問題点とその対策

前述のようにモンテカルロ計算では通常1個の粒子に対し追跡を終了するまで各ステップでの計算を行い、追跡が終了（粒子の死）した場合には次の粒子を発生させます。追跡終了となる原因には様々なものがありますが、ここで扱うプログラムでは吸収、イオン化、体系外への飛び出し、タイムオーバー、重み減少、反復回数の最大値オーバーなどです。ベクトル化にあたっては、基本的には、粒子を一度に大量発生し、これら全体に対しモンテカルロの各ステップ計算を行う方に変更します（図4.20）。“処理A、B、Cの準備”としてはリストベクトル、Gather/Scatter、フラグなどの方法を必要に応じて使い分けています（4.3.3節参照）。

この様にしてモンテカルロ計算をベクトル化するとしばしば次に示すような問題が現れます。

- ① 粒子の死（追跡中止）による粒子数の減少
- ② 粒子の死による生存粒子のメモリ配列上の不規則な不連続性
- ③ 記憶容量の増大

①、②はベクトル計算の効率を低下させる要因となります。②によるベクトル計算の効率低下を避ける為には、生存粒子のみを集める“パッキング”を行う必要があります。しかしこのパッキングはプログラム実行時のデータ転送を増加させ、大きなオーバーヘッドとなります。③はベクトル化に伴う、粒子に関する各種変数の配列化によるものであり、通常粒子数に比例して増加します。

粒子の発生法、粒子の管理、パッキングの方法これらについては後節で詳述します。

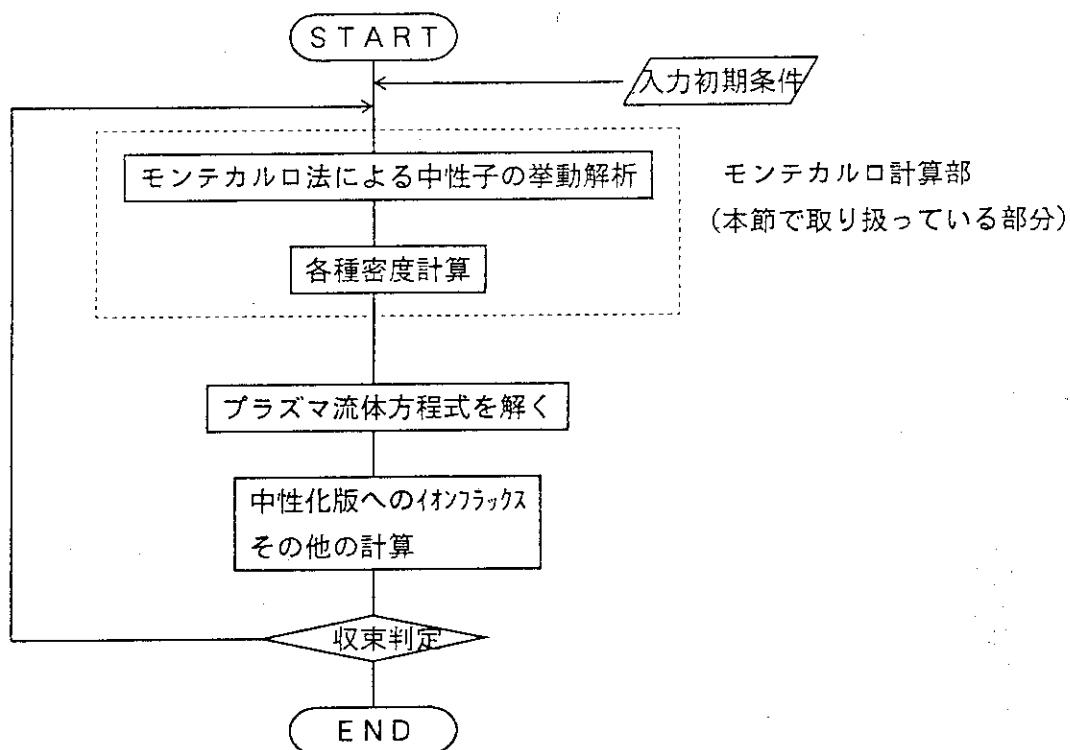


図 4.19 プログラムの概略の構造

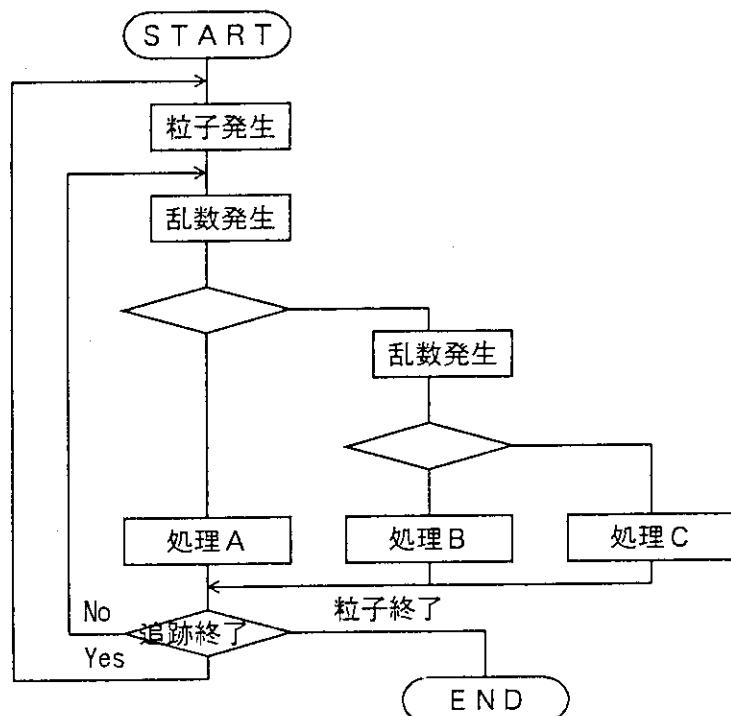


図 4.20.1 簡単なモンテカルロ計算のスカラ処理例

※ 図 4.20.1 に示される処理をベクトル化したのが、図 4.20.2 に示される処理です。

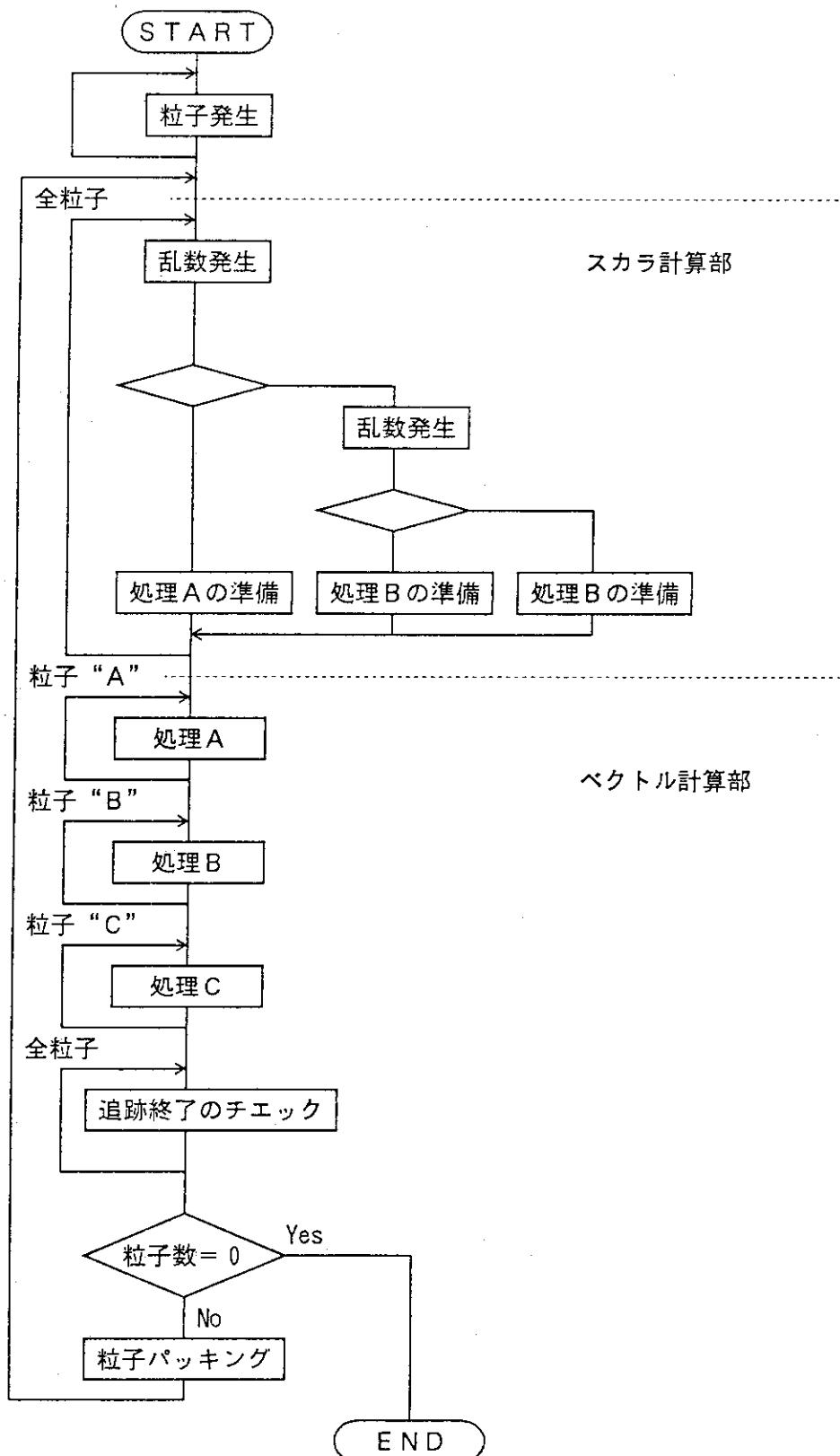


図4.20.2 簡単なモンテカルロ計算のベクトル処理例

#### 4.5.3 粒子の発生

前節に述べたようにベクトル化されたプログラムでは粒子数百個をDOループでまとめて発生させるようにしています(図4.21及び図4.20)。

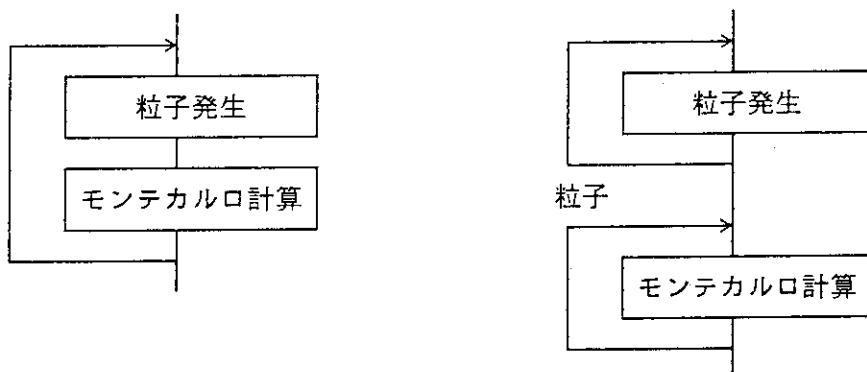


図4.21 粒子の多量同時発生

これらの粒子は粒子スタックに蓄えられ、その粒子数はモンテカルロ計算のベクトル化におけるベクトル長となります。

ベクトル化されたプログラムでは、更に死粒子(追跡中止粒子)の排除を効率良く行うために、粒子スタックに特色ある構造を採用しています。

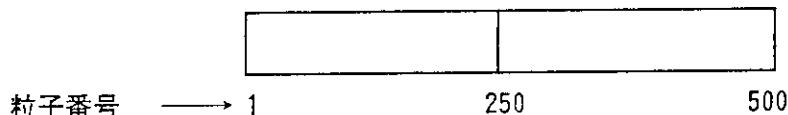


図4.22 粒子スタックの例

DOループ中の発生粒子数は初期条件によって異なりますが、常に250個以下になるようにし、スタック中の粒子が250個以下ならば、もう一度この粒子発生のDOループを起動してスタック中の全粒子数が500以下となるようにしています。この方法により、プログラムが終了する直前までスタック中の粒子数は常に250以上500以下に保たれます。すなわちベクトル長は常に充分長く保たれます。また粒子を大量に発生させる場合(例えば数万個)でも、これに伴う変数配列化によるプログラム記憶容量の増大も低く抑えられます。

#### 4.5.4 モンテカルロ計算の各ステップにおける粒子の管理

##### ——リストベクトル（インデックスベクトル）とパッキング——

モンテカルロ計算では計算途中で IF 文による分岐が起こり、各々の分岐に対し別々の計算を行わなければなりません。これらの各々の計算も当然 DO ループで行う（図4.2）ので各々の分岐に対し粒子のグループを与えなければなりません。これを行うには次の 3 つの方法が考えられます

- ① リストベクトル（インデックスベクトル）を用いる。
- ② ワーク用配列を用意してパッキング( Gathert/Scatter )する。
- ③ フラグを用いる。

①、②の方法は原理的に同じ考え方を用いますが（図 4.23）、①では最初に、変数に粒子の配列を用意するだけで変数を置き換える必要はありません。従ってプログラムの変更は最小限に押さえられます。②では、各分岐中の計算ごとにワーク用配列を用意するので、①の方法より大きな記憶領域が必要となります。

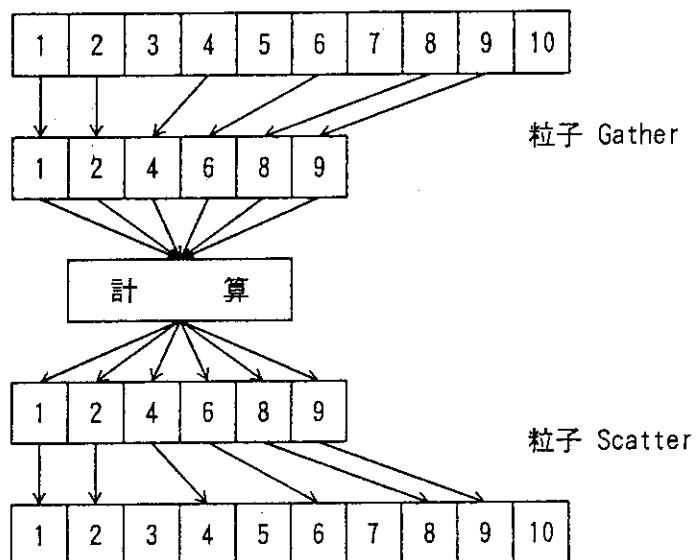


図 4.23 Gather/Scatter の原理

リストベクトルとパッキングのどちらが優れているかは、プログラムに依存するため一概には言えませんが、一般的には、各分岐中に含まれる変数の数が少なく、かつ計算量が多い（計算式が長い）場合はパッキング形式が有利です。ここで扱っているモンテカルロ計算では必要に応じて①のリストベクトルを用いる方法と②のパッキングを用いる方法を使い分けています。③のフラグを用いる方法はDO ループ中に WRITE 文がある場合などに用い、この場合は計算の DO ループが終了してからまとめて WRITE 文による書き出しを行います。

#### 4.5.5 死粒子（追跡終了粒子）の排除とパッキング

各粒子に対して全ステップ (= 1 サイクル) の計算を終えると、粒子の中には様々な原因（体系から飛び出す、吸収される、等々）により追跡終了となるものがあります。これらの死粒子と他の生存粒子と一緒にしておくのはメモリの無駄であり、また粒子の死が不規則であるため、D O ループのコントロールが難しくなります。このため死粒子を排除し、生存粒子のみをパッキングしてメモリの連続領域上に保存する必要があります。

パッキングの方法は、3節で述べてあるように特別の作業用配列を用いず、粒子に関する変数自身の配列領域上で行います。まず死粒子を配列上の先頭から探し、これを後尾より捜した生存粒子に入れ替えます。（図 4.24）。このパッキング法を以後 B F P 法（Back-to-Front Packing Method）と呼ぶことにします。

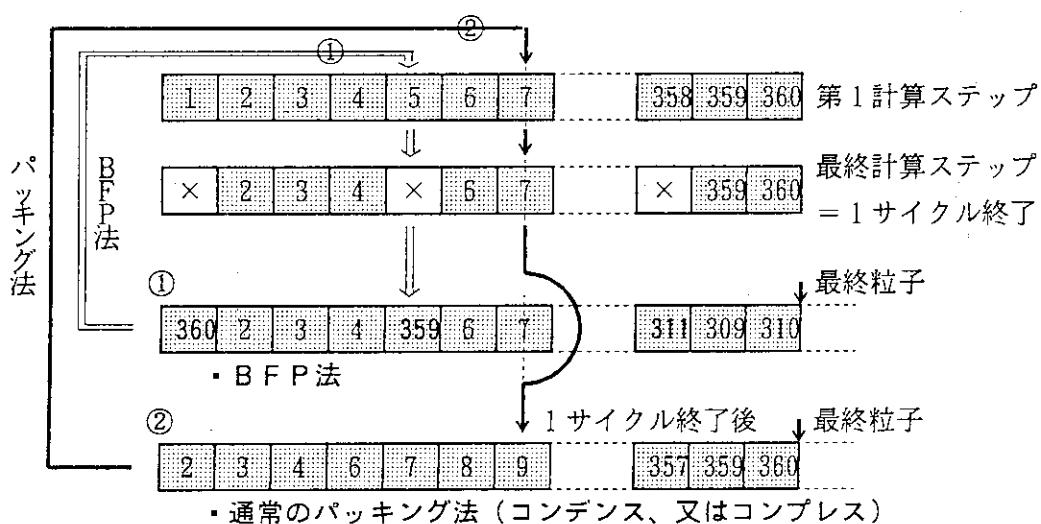


図 4.24 B F P 法①と通常のパッキング法②

※ ×印は死粒子を示す

上図 4.24 より明らかなように、B F P 法を採用することによりデータの転送の回数が大幅に減少します。

#### 4.5.6 ベクトル化による乱数列の乱れの影響のチェック

モンテカルロプログラムをベクトル（最適）化のために変更を行うと、しばしば乱数の発生順序（使用順序）がオリジナルプログラムと違ってきます。この様な場合モンテカルロ計算の性格上プログラムの実行結果にも違いが生じてきます。これはベクトル化にあたって避けることの出来ない問題点であり、エラーとは言えません。しかし、オリジナルプログラムと変更後のプログラムの実行結果の違いが許容範囲内にあるかどうかはその都度検査する必要があります。

その検査の方法には2つの異なったアプローチを考えられます。

##### ① 物理的手法

計算結果が物理的に妥当と思われる範囲に収まっているかどうかチェックします。

##### ② 統計的手法

オリジナル及び変更後のプログラムの乱数列を故意に乱して何回か実行し、各々の実行結果のバラツキの範囲が、オリジナルプログラムと変更後のプログラムとで、許容範囲内の収まっているかどうかチェックします。

②の方法は乱数列を故意に乱したプログラムを数多く実行する程、ベクトル化後のプログラムが正常に動作しているか正確にチェック出来ます。ここで取り扱った中性粒子挙動解析用モンテカルロコードでは、オリジナル・変更プログラムの両者について十数回にわたり乱数列を乱して実行し、計算結果のバラツキがほぼ一致することを確かめています。

## 第3編 FORTRAN77EXにおけるベクトル化支援ツール

## 1. FORTRAN77EXにおけるベクトル化支援ツール

プログラムをベクトル化するときは、プログラムの静的な解析の他に、実行時の動作を解析（動的解析）することが必要です。従来のFORTRANシステムでは、プログラムの実行時の動作を解析し、プログラムのチューニングに役立つ情報を提供する支援ツール（動的解析ツール）としてFORTUNEがありました。また、FORTUNEの動的解析情報にベクトル化支援のための情報を加え、プログラマに読み易い形で編集出力するツールとしてTOP10がありました。

F77EXシステムでは、FORTUNE, TOP10に代わる支援ツールとして、それぞれANALYZERとTOP10EXがあります。ANALYZERの実行詳細解析ではプログラムをベクトルモードで実行しながら解析することができる、VU（ベクトルユニット）使用時間、ベクトルモードで実行したときのコスト、ベクトル演算の統計情報等を得ることができます。また、ANALYZERのサンプリング解析機能を使用すれば、プログラムの大局的な解析が容易に出来ます。

### ■ ベクトル化支援ツールの体系

- ANALYZER ..... サンプリング解析  
..... 実行詳細解析
- TOP10EX ..... ANALYZERの実行詳細解析結果の編集  
..... ソースプログラムの編集出力
- ANALYSIS77 ..... 静的解析ツール
- FORTRAN77EX/VPコンパイラ ..... ベクトル情報付きソース出力機能

#### [目的と解析方法]

- プログラムの大局的な（ルーチン単位）解析を行いたい ⇒ サンプリング解析
- プログラムの詳細な（文単位）解析を行いたい ⇒ 実行詳細解析
- ベクトル化情報を得たい ⇒ FORTRAN77EX/VP
- コストの高い幾つかのルーチンについてのみ詳細な解析を行いたい  
⇒ サンプリング解析によりプログラム全体のコスト分布を求め、高コストのルーチンについてのみ実行詳細解析を行う（LOADINオプション）
- プログラムの構造を解析したい ⇒ ANALYSIS77
- 実行詳細解析から必要な情報を選択してプリントに出力したい ⇒ TOP10EX

## ■ ANALYZER, TOP10EXの機能

ANALYZERの実行詳細解析はFORTUNEに比べて多くのベクトル化支援情報を提供します。そのため、従来のTOP10の機能の一部はANALYZERに含まれています。

表1.1 ベクトル化支援ツールの機能

機能 または 得られる情報	FORTUNE	TOP10	A N A L Y Z E R		TOP10EX
			サンプリング	実行詳細	
■手続き単位の統計情報 ・実行回数 ・スカラコスト ・ベクトルコスト ・CPU/VU ・ベクトル化率 ・オーバヘッド時間 ・ベクトル演算数	● ● ※1	○ ○ ●	● ● ※2	● ● ● ● ●	○ ○ ○ ○ ○
■ループ単位の統計情報 ・スカラコスト ・ベクトルコスト ・平均ループ長 ・ベクトル化率 ・ベクトル化効果 ・ベクトル演算数		● ● ●		● ● ● ● ●	○ ○ ○ ○ ○
■一文単位の統計情報 ・実行回数 ・ベクトル化情報 ・最適化情報 ・スカラコスト ・ベクトルコスト	● ●	○ ● ○		● ● ● ●	○ ● ○ ○
■プログラムのネスト付表示 ■外部手続きの明示 ■ベクトル化メッセージ		● ● ●		●	● ● ●

● : 得られる情報 ○ : 解析ツールからの情報を編集出力する

※1 : CPU時間のみ

※2 : サンプリング解析では実行頻度の形で出力されます

## 2. サンプリング解析

サンプリング解析機能は、F77EXシステム及びF77システムによって作成されたロードモジュールを対象として、その実行時の振る舞いを任意の時間間隔(SAMPLING INTERVAL TIME)で監視することによって、プログラムの実行頻度分布を求めます。サンプリング解析では、ルーチン単位での実行頻度を求めることができます。サンプリング解析の特徴は、通常に翻訳・結合されたロードモジュールで解析が出来ることと(解析のための手続きが必要なのは実行ステップのみ)、解析に要する時間が実行詳細解析に比べて少ないことです。

### 2.1 サンプリング解析の手続き

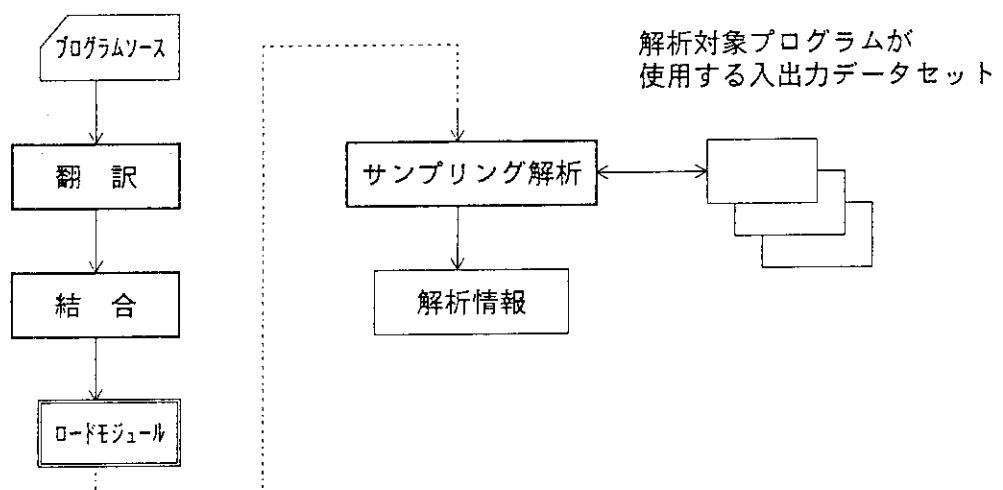


図2.1 サンプリング解析の手続き

図2.1に示すように、通常の翻訳・結合処理によって作成したロードモジュールを入力としてサンプリング解析を行います。

## 2.2 サンプリング解析の解析情報

MHD 安定性解析コード AEOLUS をサンプリング解析した結果を図2.2～図2.4に示します。

### ■ プログラム全体情報

ANALYZER V10L20 L00000 SAMPLER		DATE 91.12.19 TIME 09.57.48	
<b>SAMPLING TOTAL LIST</b>			
SAMPLING DATE	:	91.12.19	
TIME	:	09.57.48	
TARGET PROGRAM NAME	:	TEMPNAME	
SAMPLING INTERVAL TIME	:	104	
TOTAL SAMPLING COUNT	:	2400274	100.0 %
EXECUTION	:	2384880	99.3 % 100.0 %
SCALAR	:	2103787	87.6 % 88.2 %
VECTOR	:	281093	11.7 % 11.7 %
WAITING	:	15394	0.6 % 100.0 %
WAIT	:	13043	0.5 % 84.7 %
EVENTS	:	0	0.0 % 0.0 %
OTHER	:	2351	0.0 % 15.2 %
VECTOR LENGTH	MAXIMUM	:	2048
	MINIMUM	:	1
	AVERAGE	:	66
VECTOR SAMPLING RATIO	:	11.7 %	
SYSTEM COMPLETION CODE	:	000	
USER COMPLETION CODE	:	008	
ELAPSED TIME	:	24:45.483	
CPU TIME	:	03:33.556	
VU TIME	:	01:53.030	

図2.2 サンプリング解析のプログラム全体情報

#### [出力情報の解説]

TARGET PROGRAM NAME	: サンプリング解析の解析対象コードモジュールのプログラム名
SAMPLING INTERVAL TIME	: サンプリング間隔時間 (単位: $\mu$ s)
TOTAL SAMPLING COUNT	: 全体の実行頻度の値
EXECUTION	: 実行中のサンプリング数
SCALAR	: スカラ命令のサンプリング数
VECTOR	: ベクトル命令のサンプリング数
WAITING	: 待ち状態のサンプリング数
WAIT	: WAIT命令による待ち状態のサンプリング数
EVENTS	: EVENTS命令による待ち状態のサンプリング数
OTHER	: その他の要因による待ち状態のサンプリング数
VECTOR LENGTH	: ベクトル長
VECTOR SAMPLING RATIO	: ベクトル命令の実行頻度
ELAPSED TIME	: 経過時間
CPU TIME	: C P U 時間
VU TIME	: V U 時間

### ■ モジュール単位情報

ANALYZER V10L20 L00000 SAMPLER				DATE 91.12.19 TIME 09.57.48		
SAMPLING MODULE LIST						
MOD-NAME	LENGTH	COUNT	%	V-COUNT	%	
TEMPNAME	481288	2383344	99.9	281093	11.7	*****
<SYSTEM>	-	1536	0.0	-	-	

図2.3 サンプリング解析のモジュール単位情報

### [出力情報の解説]

MOD-NAME	: ロードモジュールのメンバ名
<SYSTEM>	: 上記以外のロードモジュール
LENGTH	: モジュール長
COUNT	: サンプリング数と全体に対する割合
V-COUNT	: ベクトル命令のサンプリング数と全体に対する割合

### ■ ルーチン単位情報

ANALYZER V10L20 L00000 SAMPLER				DATE 91.12.19 TIME 09.57.48		
SAMPLING ROUTINE LIST						
MODULE-NAME: TEMPNAME LENGTH:	481288	COUNT:	2383344	99.9	X	V-COUNT: 281093 11.7 X
RTN-NAME	LENGTH	OFFSET	COUNT	X	V-COUNT	X
DEC01	8196	05A130	531490	22.3	30343	5.7 *****
THROHI	20324	04CB18	417797	17.5	47023	11.2 *****
DEC	2640	05C800	333583	13.9	94201	28.2 *****
SOLY	5112	05D320	331262	13.8	78404	23.6 *****
TMORH	18084	036750	263132	11.0	4128	1.5 *****
TMPERP	6908	051A80	225312	9.4	4856	2.1 *****
SOL	1940	05C138	119104	4.9	9394	7.8 **
SOLBT	7636	05B358	100695	4.2	4620	4.5 **
RECUR	3624	05E718	32459	1.3	33	0.1 *
THCIR	8008	031A38	9125	0.3	474	5.1
TMPSIM	11728	049108	9008	0.3	5034	55.8
TMVOL	4916	053580	3622	0.1	1635	45.1
INVCRT	5168	00A610	1110	0.0	111	10.0
TMPA1	4976	046B50	741	0.0	333	44.9
TMOPS	6988	034C00	565	0.0	288	50.9
INVSQL	4796	00C278	399	0.0	17	4.2
TMENR	11948	03ADF8	357	0.0	18	5.0
TMDET	4132	033B08	298	0.0	140	46.9
STEQU	51916	0100E0	93	0.0	9	9.6
STMAT	4812	02E920	92	0.0	4	4.3
TMREP	2928	04BFAB	88	0.0	23	26.1
TMETRQ	9516	03F410	49	0.0	-	-
INVRAY	4600	00D538	45	0.0	1	2.2
THETA	1952	03DCAB	45	0.0	-	-
ZZZCOM	1852	05F540	-	-	-	-
FSEMSG	42	0735A8	-	-	-	-
<LIBS>	-	-	2807	0.1	1	0.0

図2.4 サンプリング解析のルーチン単位情報

### [出力情報の解説]

RTN-NAME	: 該当ロードモジュールの中のルーチン名
<LIBS>	: ライブライ
LENGTH	: ルーチンの長さ
OFFSET	: ルーチンのモジュールの先頭からの相対オフセット
COUNT	: サンプリング数と全体に対する割合
V-COUNT	: ベクトル命令のサンプリング数と全体に対する割合

### 2.3 サンプリング解析のオプション

サンプリング解析のオプションは EXEC 文の PARM パラメタで指定します。

表2.1 サンプリング解析のオプション

オプション	機能
INTERVAL ( $\left\{ \frac{2000}{m} \right\}$ )	サンプリング間隔 (単位: $\mu s$ ) $104 \leq m \leq 214748648$
PGMNAME ( $\left\{ \frac{\text{TEMPNAME}}{\text{entname}} \right\}$ )	ロードモジュールのプログラム名
TPARM (options)	解析対象プログラムへ渡す実行時オプションを指定する

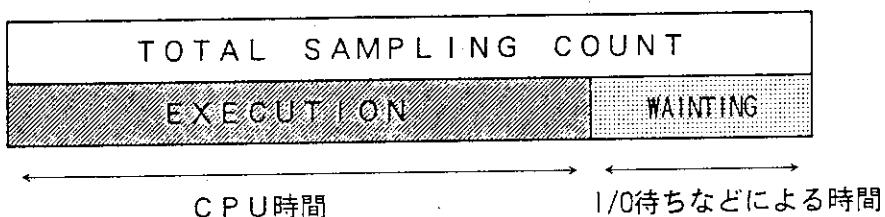
※ 下線は標準値

サンプリング間隔を小さくとれば解析の精度は高くなりますが、計測にかかるオーバヘッドが大きくなるため、解析に要する経過時間は長くなります。

### 2.4 サンプリング解析情報の評価

サンプリングによる誤差はありますが、サンプリング間隔とサンプリング数を掛け合わせた値は CPU 時間に相当します（但し、VU 時間については、ベクトル命令の実行形態の都合上、サンプリング数とサンプリング間隔を掛け合わせた値になりません）。

サンプリング解析により CPU 時間の大きいルーチンを限定したら、F77EX/VP コンパイラの出力するベクトル情報付きソースリストを手掛かりにしてベクトル化作業を進めることも出来ます。 ループ単位、文単位のコスト、あるいはループ毎のベクトル長を知りたい場合は、これらのルーチンを実行詳細解析（3 章）して下さい。



$$(\text{EXECUTION}) * (\text{SAMPLING INTERVAL TIME}) \approx (\text{CPU TIME})$$

## 2.5 サンプリング解析の使用

### 2.5.1 カタログド・プロジェクト

サンプリング解析のためのカタプロ SAMPLER を用意しました。

#### ● SAMPLER

パラメタ	用	途	標準値
LM	解析対象ロードモジュール		.LOAD
Q	解析対象ロードモジュールのタイプ		TEMPNAME
P NM	解析対象プログラム名		2000
I N V	サンプリング間隔時間 [単位: $\mu$ s]		ERRSET=0
A	実行時オプション		
SMP OUT	サンプリング解析結果のデータセット		.SMP OUT
SMP Q	サンプリング解析結果のデータセットのタイプ		8 M
R GN	実行リージョン		VBA
O R E C F M	サンプリング解析結果データセットのレコード形式		12504
O B S I Z E	サンプリング解析結果データセットのブロック長		125
O R S I Z E	サンプリング解析結果データセットのレコード長		*
S Y S O U T	SYSOUT出力クラス		DDNAME=SYSIN
G O S Y S I N	FT05F001に対するDD名の指定		

装置参照番号 5 に対するデータセットは DD 名 'SYSIN' で指定することが出来ます。  
また、装置参照番号 6 は SYSOUT 出力に設定されています。

### 2.5.2 使用例

#### ● 翻訳・結合～サンプリング解析

```
//F77 EXEC FORTEXVP, SO='J****.TEST',
//      A='ELM(*)'
//LINK EXEC LKEDCTEX, LM='J****.TEST'
//*
//SMP EXEC SAMPLER, LM='J****.TEST',
//      INV=104,
//      SMP OUT='J****.TEST'
//SYSIN DD DSN=J****.TEST, INPUT, DISP=SHR
```

この例では、サンプリング解析の結果は 'J\*\*\*\*.TEST.SMP OUT' に出力しています。

### 3. 実行詳細解析

実行詳細解析は、実行文の実行回数やコストなどによりプログラムの動作を詳細に解析する機能です。 本機能では、プログラム単位毎、或いは文単位毎に実行回数、コスト値、C P U時間等の統計情報を求め出力します。

実行詳細解析は通常の実行に比べて、多くのC P U時間と I/O回数を要します。 大規模プログラムを解析する場合は、サンプリング解析により全体のコスト分布を分析し、コストの高いルーチンについてのみ実行詳細解析をすることを勧めます。

#### 3.1 実行詳細解析の手続き

実行詳細解析は翻訳から実行までを 1 つのステップで行います。 従って、被測定プログラムが翻訳～実行に必要とするデータセットは全て指定します。

実行詳細解析には、①プログラム全体を解析対象とする方法と ②プログラムの一部を解析対象とする方法 (LOADINオプション指定時) の 2 つの方法があります。 プログラムの一部を解析対象とする場合は、通常に翻訳・結合して作成したロードモジュールを入力しなければなりません。 このとき、ANALYZERは翻訳したプログラムのオブジェクトと入力したロードモジュールから解析用ロードモジュール（一時データセット）を新たに作成し、実行します。

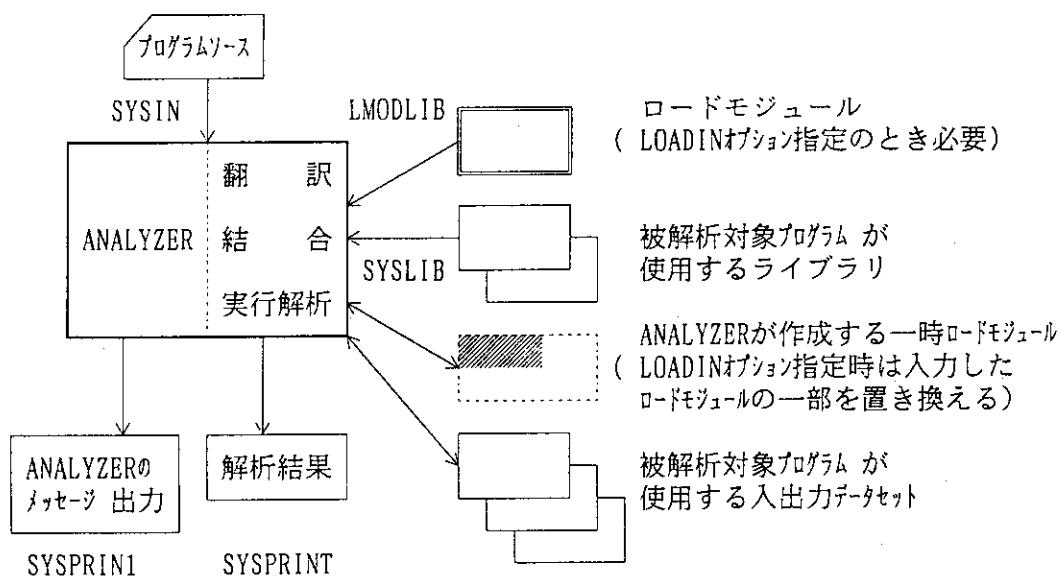


図3.1 実行詳細解析の手続き

### 3.2 実行詳細解析の解析情報

ANALYZERの実行詳細解析では、図3.2 の形式で解析情報が outputされます。 解析出力の1行あたりの桁数は LLINEオプションを指定した場合は 250桁、 SLINEオプションを指定した場合は 120桁となります。 実行詳細解析の解析出力をそのままプリンタに出力したい場合は、 SLINEオプションを選択してください。 但し、 SLINE オプションでは、プログラムソースは50桁までしか出力されません。 また、 T O P 1 0 E Xで解析情報を編集する場合は必ず LLINE オプションを指定して下さい。

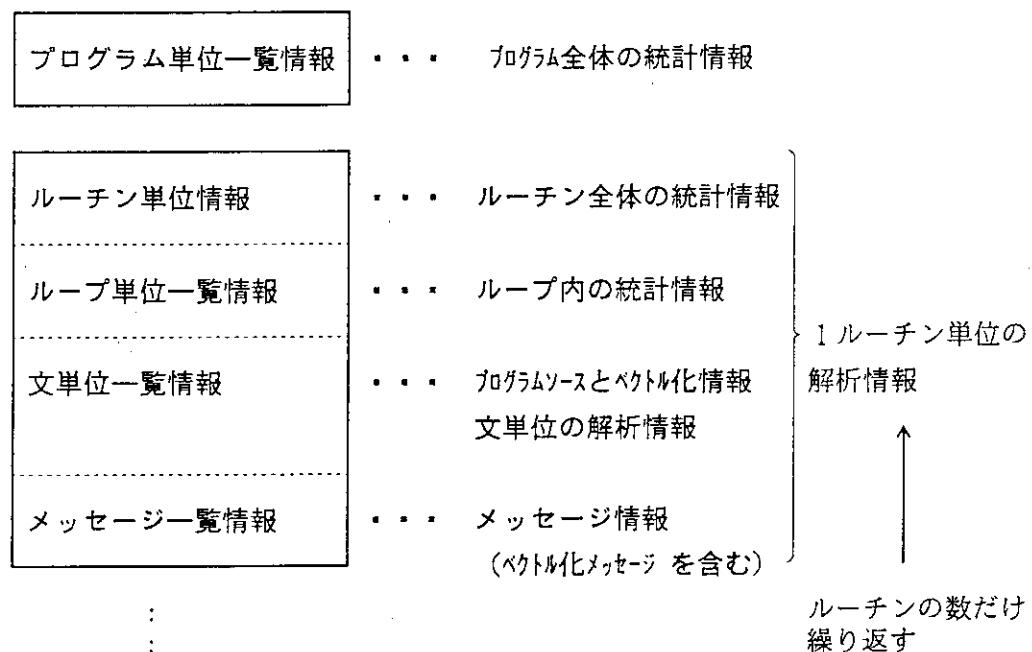


図3.2 実行詳細解析の出力形態

実行詳細解析の出力例を図3.3 ~図3.7 に示します。このときのオプションは 'COUNT, CPU, SLINE' です。

#### ■ プログラム単位一覧情報

ANALYZER V10L20 L00000 JPJVPRFV DATE 91.11.06 TIME 17.14.02 PAGE 1										
VECTORIZE - TOTAL LIST -----										
<b>NAME EX-COUNT CPU-AVG VU-AVG CPU-TIME X VU-TIME X V-COST X S-COST X</b>										
SEID 10000 0.000 0.000 4.140 96.2 3.965 99.9 .1115E09 99.8 .7820E10 99.9										
MAIN 1 0.160 0.002 0.160 3.7 0.002 0.0 86214 0.0 2590032 0.0										
DIFAB 1 0.000 0.000 0.000 0.0 0.000 0.0 35670 0.0 2500058 0.0										
GETB 1 0.000 0.000 0.000 0.0 0.000 0.0 10661 0.0 748006 0.0										
(TOTAL) ----- ----- 4.302 100.0 3.968 100.0 .1116E09 100.0 .7825E10 100.0										
<b>NAME V-LENG V-RATE V-EFFECT OVERHD EX-L/ST EX-LIST EX-DIST EX-ADD EX-MLT EX-DIV</b>										
SEID 10000 99.9 60.3- 81.8 7.3 0 .2000E09 0 .8000E09 .5000E09 .1000E09										
MAIN 8462 98.0 28.1- 31.9 0.0 10000 0 100000 220000 230000 110000										
DIFAB 8334 99.9 60.3- 81.7 17.4 200000 0 100000 300000 200000 100000										
GETB 5000 99.9 60.4- 81.8 15.7 10000 10000 0 70000 50000 10000										
(TOTAL) ----- 99.9 60.3- 81.7 ----- 220000 .2000E09 200000 .8000E09 .5004E09 .1002E09										

図3.3 実行詳細解析のプログラム単位一覧情報

## 〔出力情報の解説〕

NAME	: ルーチン名 (副入口の場合は先頭に'>'が付加される)
EX-COUNT	: 実行回数
CPU-AVG	: ルーチンの一回呼び出しあたりの CPU使用時間
VU-AVG	: ルーチンの一回呼び出しあたりの VU 使用時間
CPU-TIME	: CPU 使用時間
VU-TIME	: VU 使用時間
V-COST	: ベクトルコスト (VPで実行した場合のコスト)
S-COST	: スカラコスト (汎用機で実行した場合のコスト)
V-LENG	: 論理的な平均ベクトル長、 ベクトル化されていないループでは平均回転長
V-RATE	: おおよそのベクトル化率 [=ベクトル化された部分のスカラコストの合計 / スカラコストの合計]
V-EFFECT	: ベクトル化効果 (最大値 - 最小値) [=スカラコスト / ベクトルコスト]
OVERHD	: ルーチンの出入口にかかる CPU 使用時間のルーチン全体に対するおおよその比率

## ■ ルーチン単位情報

ROUTINE LIST												
ROUTINE	EX-COUNT	CPU-AVG	VU-AVG	CPU-TIME	X	VU-TIME	X	V-COST	X	S-COST	X	
SETD	10000	0.000	0.000	4.140	96.2	3.965	99.9	.1115E09	99.8	.7820E10	99.9	
ROUTINE	V-LENG	V-RATE	V-EFFECT	OVERHD								
SETD	10000	99.9	60.3-	81.8	7.3							

図3.4 実行詳細解析のルーチン単位一覧情報

## ■ ループ単位情報

LOOP LIST MAIN												
SSN	DO-ID	KIND	V	EX-COUNT	V-COST	X	S-COST	X	V-LENG	V-RATE	V-EFFECT	
00000025-00000027	100	DO	M	1	53418	61.9	290000	11.1	10000	82.7	5.3-	5.4
00000032-00000033	200	DO	V	10	32763	38.0	2300000	88.8	10000	100.0	60.4-	81.8
00000031-00000033	200	DO	S	1	20	0.0	20	0.0	10	0.0	1.0-	1.0
00000021-00000035	-----	TOTAL		1	12	0.0	12	0.0	1	0.0	1.0-	1.0

図3.5 実行詳細解析のループ単位一覧情報

## 〔出力情報の解説〕

SSN またはLINE-NUMBER, LINE-NO.	: 内部順序番号 または ソースの行番号
DO-ID	: DOループに対してのみ、その端末文の文番号が出力される
KIND	: ループの種別 (DO:DO ループ, WHILE:DO WHILEループ, UNTIL:DO UNTILループ, IF:IF-GOTOループ, TOTAL: ルーチンのループに属さない部分)

## ■ 文単位情報

VECTORIZE - STATEMENT LIST -- MAIN --										
SSN	EX-COUNT	TRUE	V-COST	S-COST	CPU-TIME	VU-TIME	V	O	C	
C234567890										
00000021										IMPLICIT REAL*8 (A-H,O-Z)
00000022										DIMENSION A(10000),B(10000),C(10,10000),D(10
00000023										CHARACTER*136 CHAR
00000024	1		2	2			\$ 9			DO 99 I=1,136
00000025	136		1088	1088			\$ 9	99		CHAR(I:I)*' '
00000026	1		1	1	0.004	0.000				WRITE(6,*), ANATEST START
00000027	1		1	1	0.000	0.000				WRITE(6,'(IX,A136)') CHAR
00000028	1		2	2	0.000	0.000				CALL GETB(D)
00000029	1		2	2			H			DO 100 I=1,10000
00000030	10000		3418	240000			V			A(I) = DFLOAT(MOD(I,5))*100.0/2.0
00000031	10000		20000	20000	4.914	3.969 S				CALL SETD(D,10000)
00000032	10000		30000	30000			S	100		CONTINUE
00000033	1		2	2	0.000	0.000				CALL DIFAB(A,B,C)
00000034	1		1	1						S = 0.0
00000035	1		2	2			S 2			DO 200 J=1,10
00000036	10		20	20			V 2			DO 200 I=1,10000
00000037	100000		24216	1700000			V 2			S = S+C(J,I)*DFLOAT(MOD(I,8))
00000038	100000		8547	600000			V 2	200		CONTINUE
00000039	1		1	1	0.000	0.000				WRITE(6,*), S=S, S
00000040	1		1	1	0.000	0.000				WRITE(6,'(IX,A136)') CHAR
00000041	1		1	1	0.000	0.000				WRITE(6,*), ANATEST END
00000042	1		2	2						END

図3.6 実行詳細解析の文単位一覧情報

## 〔出力情報の解説〕

V	: ベクトル化情報
V	: ベクトル化されている
M	: 部分的にベクトル化されている
S	: ベクトル化されていない
空白	: ベクトル化対象外
O	: 最適化情報
	: インライン展開
数字	: アンローリングの多重度
TRUE	: IF文の真率
CPU-TIME	: 手続き呼び出しの CPU時間
VU-TIME	: 手続き呼び出しのVU時間

■ ベクトル命令統計情報 (VOPオプション)

VOPオプションが指定された場合は、プログラム単位一覧情報とループ単位一覧情報にベクトル命令の統計情報を加えて出力します。

VECTORIZE - ROUTINE LIST -----												
ROUTINE	EX-COUNT	CPU-AVG	VU-AVG	CPU-TIME	X	VU-TIME	X	V-COST	X	S-COST	X	
DIFAB	1	0.000	0.000	0.000	0.0	0.000	0.0	35670	0.0	2500058	0.0	
ROUTINE	V-LENG	V-RATE	V-EFFECT	OVERHD	EX-L/ST	EX-LIST	EX-DIST	EX-ADD	EX-MLT	EX-DIV		
DIFAB	8334	99.9	60.3- 81.7	17.4	200000	0	100000	300000	200000	100000		
ROUTINE	EX-MASKC	EX-MA	EX-VC	EX-MAC	EX-REC	EX-COMP	EX-FUNC	EX-OTHER				
DIFAB	0	0	0	0	0	0	0	0				

図3.7 実行詳細解析のベクトル命令統計情報（プログラム単位）

VECTORIZE - LOOP LIST ----- SETD -----												
SSN	DO-ID	KIND	V	EX-COUNT	V-COST	X	S-COST	X	V-LENG	V-RATE	V-EFFECT	
00000040-00000045	10	DO	V	10000	.1113E09	99.9	.7820E10	99.9	10000	100.0	60.4- 81.8	
00000036-00000046	-----	TOTAL		10000	110000	0.0	110000	0.0	10000	0.0	1.0- 1.0	
SSN	EX-L/ST	EX-LIST	EX-DIST	EX-ADD	EX-MLT	EX-DIV	EX-MASKC	EX-MA	EX-VC			
00000040-00000045	0	.2000E09	0	.8000E09	.5000E09	.1000E09	.3000E09	.2000E09	0			
00000036-00000046	0	0	0	0	0	0	0	0	0			
SSN	EX-MAC	EX-REC	EX-COMP	EX-FUNC	EX-OTHER							
00000040-00000045	0	0	.4000E09	.2000E09	0							
00000036-00000046	0	0	0	0	0							

図3.8 実行詳細解析のベクトル命令統計情報（ループ単位）

[出力情報の解説]

- EX-L/ST : ベクトルロード/ストア 命令のおおよその実行回数
- EX-LIST : ベクトルインダイレクトロード/ストア命令のおおよその実行回数
- EX-DIST : ベクトルディスタンス付きロード/ストア命令のおおよその実行回数
- EX-ADD : ベクトル加算命令のおおよその実行回数
- EX-MLT : ベクトル乗算命令のおおよその実行回数
- EX-DIV : ベクトル除算命令のおおよその実行回数
- EX-MASKC : マスクペイゲ系命令のおおよその実行回数
- EX-MA : ベクトル加算＆乗算命令のおおよその実行回数
- EX-VC : ベクトル制御命令のおおよその実行回数
- EX-MAC : VSUM/FIND系命令のおおよその実行回数
- EX-REC : ベクトルリカレンス系命令のおおよその実行回数
- EX-COMP : 圧縮／拡散系命令のおおよその実行回数
- EX-FUNC : ベクトル組み込み関数のおおよその実行回数
- EX-OTHER : 上記以外のベクトル命令のおおよその実行回数

※ タイトルに'EX-' が付加されないで出力された場合( 例えば L/ST ) は、( DOループやルーチンが) 一回実行されたときのそれぞれのベクトル命令の実行回数となる。

### 3.3 実行詳細解析のオプション

実行詳細解析のオプションは EXEC 文の PARM パラメタで指定します。 PARM パラメタには翻訳オプション、リンクエージオプション、実行時オプションを併せて指定することが出来ます（各々のオプション並びは / で区切ること）。 実行詳細解析を行う場合は、アナライザのオプション並びに COUNT か CPU のいずれかを必ず指定して下さい。 どちらも指定しない場合は見積り解析となります。

アナライザの ( 翻訳オプション ( / リンケージ オプション並び ( / 実行オプション ) ) )  
 オプション並び ( / 並び \* )

\* V P ハードオプションと V P シリーズオプションを除く翻訳オプションを指定する

表3.1 ANALYZERのオプション

オプション		機能概要
COUNT	<u>NOCOUNT</u>	実行詳細解析の実行回数計測を指定する
CPU	<u>NOCPU</u>	実行詳細解析のプログラム実行時間、手続き呼び出し実行時間、及び入口／出口のオーバヘッド比率の計測を指定する
DLIST	<u>NODLIST</u>	実行回数及びコストを実数形式に統一して印刷する
ENTRY	<u>NOENTRY</u>	副入口の CPU 時間及び VU 時間を主入口と分けて出力する
INCLUDE INC	<u>NOINCLUDE</u> <u>NOINC</u>	INCLUDE 展開部分の印刷を指定する
LINECOUNT( [ 60   n ] ) LC ( [ 60   n ] )		解析情報出力の 1 ページ当たりの行数を指定する 0 ≤ n ≤ 99 (2 以下はベタ打ち)
SCALAR	<u>VECTOR</u>	実行モードの指定 (汎用計算機で解析する場合は SCALAR を指定)
S LINE SL	<u>LLINE</u> <u>LL</u>	解析情報出力の桁数を指定する (120 桁または 250 桁)
STATIC	<u>NOSTATIC</u>	見積り詳細解析を指定する
SPACE ( [ 1   n ] )		作業用データセットの大きさ (インストール時の設定) の倍率を指定する 0 ≤ n ≤ 999
VOP	<u>NOVOP</u>	ベクトル演算数の印刷を指定する
VP (30   50   100   200   400 30E   50E   100E   200E   400E 1100   1200 2100   2200   2400   2600)		プロセッサの指定
LOADIN		再翻訳したオブジェクトでロードモジュールを入れ換える (プログラムの一部を詳細解析する場合のオプション)

\* 下線は標準値

3.4 実行詳細解析の使用3.4.1 カタログド・プロジェクト

実行詳細解析のために、カタログド・プロジェクトANALYZER, ANALDINが用意されています。 プログラムの一部を解析する場合はANALDINを使用します。

**●ANALYZER, ANALDIN**

パラメタ	用	標準値
AOP ※1	アナライザの実行詳細解析のオプション	※3
COP ※2	翻訳オプション／結合オプション／実行時オプション	OPT(E)
SO	解析対象ソースのデータセット名	NULLFILE
Q	解析対象ソースのタイプ名	.FORT77
RGN	実行リージョン	8 M
ANAOUT	解析結果出力データセット名	ANAOUT
ANAQ	解析結果出力データセットのタイプ	TSSWK
UNIT	解析結果出力データセットのユニット名	VBA
ORECFM	解析結果出力データセットのファイル形式	255
ORSIZE	解析結果出力データセットのレコード長	2554
OBSIZE	解析結果出力データセットのブロック長	NO
GRLIB	グラフィックライブラリ	JSSL
SSLA	科学技術計算ライブラリ	SSL
SSLB	科学技術計算ライブラリ	SSL2
SSLC	科学技術計算ライブラリ	SYS9.GGS
GGS	グラフィックライブラリ	SYS9.NO
PRVLIB	プライベートライブラリ	.LOAD
PRVQ	プライベートライブラリのタイプ	DDNAME=
GOSYSIN	FT05F001に対するDD名の指定	GOSYSIN
SYSOUT	SYSOUT出力クラス	*
LM ※4	入力ロードモジュール名	
LOADQ※4	入力ロードモジュールのタイプ名	.LOAD
PNM ※4	入力ロードモジュールの主入口名	TEMPNAME

※1： 表3.1 ANALYZERのオプションを参照のこと

※2： 翻訳オプション、結合オプション、実行時オプションを'/'で区切って指定する。 それぞれのオプションを省略することも可能

※3： 'LL,COUNT' (ANALDINでは'LOADIN'をPARMパラメータ中に設定)

※4： ANALDINのみ必要

装置参照番号5に対するデータセットはDD名'GOSYSIN'で指定することができます。

また、装置参照番号6はSYSOUT出力に設定されています。

3. 4. 2 使用例

- 全てのルーチンを' COUNT, CPU' の両オプションで実行詳細解析する

```
//ANA EXEC ANALYZER, RGN=18M,
//      SO=' J****.TEST',
//      AOP=' LL(CPU,COUNT),
//      COP=' ELM(*),OPT(E)',
//      ANAOUT=' J****.TEST
//GOSYSIN DD DSN=J****.TEST, DATA, DISP=SHR
```

- 全てのルーチンを' COUNT, CPU' の両オプションで実行詳細解析し、解析結果をスブルに出力する。

```
//ANA EXEC ANALYZER, RGN=18M,
//      SO=' J****.TEST',
//      AOP=' CPU,COUNT,SL',
//      COP=' ELM(*),OPT(E)'
//SYSPRINT DD SYSOUT=*,*
//      DCB=(RECFM=FBA,LRECL=137,BLKSIZE=19043)
//GOSYSIN DD DSN=J****.TEST, DATA, DISP=SHR
```

- 一部のルーチン（メンバSUBA, SUBB）のみを実行詳細解析する

```
//F77 EXEC FORTEXVP, SO=' J****.TEST',
//      A=' ELM(*)',
//LNK EXEC LKEDCTEX, LM=' J****.TEST
//*
//ANA EXEC ANALDIN, RGN=18M,
//      SO=' J****.TEST',
//      COP=' ELM(SUBA,SUBB),OPT(E)',
//      ANAOUT=' J****.TEST',
//      LM=' J****.TEST'
//GOSYSIN DD DSN=J****.TEST, DATA, DISP=SHR
```

### 3.5 実行詳細解析の使用上の注意

- ① 原始プログラムを格納した複数のデータセットをDD名SYSINに指定することが出来ます。しかし、P0データセットとPSデータセットを混在して指定することは出来ません。
- ② 実行詳細解析ではベクトルモードで実行した場合の解析を行います。部分的にスカラモードで実行したいルーチンがある場合は、そのルーチンの先頭に最適化制御行'\*VOCL TOTAL, SCALAR'を挿入して下さい。  
また、プログラム全体をスカラモードで実行した場合の解析を行いたいときは、オプション' SCALAR'を指定して下さい。
- ③ 実行詳細解析では、通常に翻訳～実行した場合に比べて非常に多くのCPU時間とI/Oを要します。
- ④ 'CPU' と 'COUNT' の両オプションを指定して実行詳細解析した場合は、いずれか一方だけを指定して解析した場合の約2倍のCPU時間がかかります。
- ⑤ プログラムの一部を解析対象とした場合は、プログラム全体のコスト分布を求めることが出来ません。

複数のデータセットに分割されている原始プログラムを1つのデータセットにまとめたり、スカラ実行するルーチンに最適化制御行'\*VOCL TOTAL, SCALAR'を挿入したりする作業にはMERGER(付録Cを参照)を利用すると便利です。

## 4. TOP10EX

ANALYZERの実行詳細解析では多くの解析情報を提供する反面、その出力形式はプリンタ出力に適しているとは言えません。また、解析情報を選択して出力することが出来ないので、大規模プログラムを解析した場合、情報量は紙に印刷できる量としてはあまりに多すぎます。そこで、ANALYZERの解析出力の中から、必要な情報を効率良く取り出し、プログラマに見やすい形で提供するツールとしてTOP10EXを開発しました。

### 4.1 TOP10EXの機能

TOP10EX (VER. 1.1)は以下の機能を備えています。

- ANALYZER実行詳細解析の出力を入力とする場合
  - (1) プリンタ出力を意識した桁編集
  - (2) 出力するルーチンの選択出力
  - (3) DOループ単位情報をソース行に併せて出力
  
- FORTRAN原始プログラムを入力とする場合
  - (1) FORTRAN77EX/VPコンパイラを呼び出し、その結果得られるベクトル化情報、最適化情報、ベクトル化メッセージを付加する。  
(VP翻訳情報の付加)
  - (2) SSN(内部順序番号)付け

また、いずれの機能を使用した場合にも、次の編集がなされます。

- (1) DOループの構造、IF～THEN～ELSE構造の範囲及びループの深さを明示 (ネスト付出力機能)
- (2) 手続き引用箇所の引用部分にアンダーラインを引く (手続き引用の明示)

TOP10EX で出力される文単位編集リストを、ANALYZERの実行詳細解析の出力を入力とした場合とFORTRAN 原始プログラムを入力とした場合について、それぞれ図4.1 と図4.2 に示します。

ループに関する情報は以下の形式で出力します。ベクトルコストとスカラコストはルーチン全体に対する比率を示します。

ループID	VRATE=ベクトル化率 %	ベクトルコスト %	スカラコスト %
	AV. LEN= 平均回転長	EFF.=ベクトル化効果	

手続き呼び出しの時間は以下の形式で出力します。

==== CPU/VU ===== CPU時間 S VU時間 S

図4.1 TOP10EXによるANALYZERの実行詳細解析出力の編集

```

*****+
** CORR **
*****+
      FILE=J9051.SPINJ.FORT77(CORR)          FORTRAN 77/VF V12L10
      -----1-----2-----3-----4-----5-----6-----7-----+
C *** CORRELATION FUNCTION OF SPIN GLASS           *CORR   ( 1 )
C
      SUBROUTINE CORR(Z,XO,Z0,MXO),Z,SS(XO,MYO)
      REAL*8 Z(XO),Z0(MXO),Z,SS(XO,MYO)
      COMMON /CN01/ XO,MY0,MS0,T
      COMMON /CN02/ LBO,MX0,MY0,MS0,MX0
      LOGICAL LBO
      COMMON /CLT/ LY0
      LOGICAL LY0
      COMMON /CIG/ IGO
      COMMON /KDS/ K00,KD0,K01,KD1,HD
      C +
      C !
      C !
      PARAMETER  LBMAX = 1024
      COMMON /CBIT1/ LBITT(LBMAX,21)
      COMMON /CBIT2/ J1MAX,LOT(LBMAX),J2T(LBMAX)
      LOGICAL LBITT
      INTEGER*4 LOT,JOT,J2T,J1MAX
      REAL*8 01
      C !
      C !
      C +
      C !
      C !
      REAL*8 SA(Z0),SB(Z0),SC(Z0),ST(Z0),SD(Z0)
      DATA IOUT /2/
      MY0=+0.0
      IFLB01=MY1=0
      MTO=MY0+2*MY1
      MS0=XO-MYO-MX0/2
      IFLB02=MS0=MY0-MYO-(MX0+MY0)/2
      MX0=MX0-1
      MXA=MX0/2
      X1=MX0+2*MXA
      IF(XM1.EQ.1) MXA=MXA+1
      XIB=MX0-MXA
      DO 12 NX1=MX0
      DO 12 NY1=MY0
      S(NX1,NY1)=0.000
      12 CONTINUE
      20 CONTINUE
      C ..
      SSN=0017 1-----V--10 DO 12 NX1=MX0
      SSN=0018 1-----V--12 NY1=MY0
      SSN=0019 1-----V-----S(NX1,NY1)=0.000
      SSN=0020 1-----V-----CONTINUE
      SSN=0021 1-----V-----20 CONTINUE
      SSN=0022 1-----V-----C..
      SSN=0023 1-----V-----IF(LB01.EQ.1) LBITT(LB01)=1
      SSN=0024 1-----V-----IF(LB02.EQ.1) LBITT(LB02)=1
      SSN=0025 1-----V-----IF(XM1.EQ.1) MXA=MXA+1
      SSN=0026 1-----V-----XIB=MX0-MXA
      SSN=0027 1-----V-----DO 26 J1 = 1,J1MAX
      SSN=0028 1-----V-----S     J0 = JOT(J1)
      SSN=0029 1-----V-----S     J2 = J2T(J1)
      26 CONTINUE
      C ..
      SSN=0030 1-----V-----DO 111 L2 = 1,JO
      SSN=0031 1-----V-----V     LOTT(L2) = L2 + J2 - 1
      111 CONTINUE
      20 CONTINUE
      C ..
      SSN=0032 1-----V-----DO 112 L1 = 1,MX0
      SSN=0033 1-----V-----DO 112 L2 = 1,JO
      SSN=0034 1-----V-----V     L4 = ISHFT(LOT(L2), - 1)
      SSN=0035 1-----V-----LBITT(L2,L1) = LOTT(L2).GT.(2 + L4)
      SSN=0036 1-----V-----LOT(L2) = L4
      112 CONTINUE
      C ..
      SSN=0037 1-----V-----DO 111 L2 = 1,JO
      SSN=0038 1-----V-----V     LOTT(L2) = L2 + J2 - 1
      111 CONTINUE
      C ..
      SSN=0039 1-----V-----DO 112 L1 = 1,MX0
      SSN=0040 1-----V-----DO 112 L2 = 1,JO
      SSN=0041 1-----V-----V     L4 = ISHFT(LOT(L2), - 1)
      SSN=0042 1-----V-----LBITT(L2,L1) = LOTT(L2).GT.(2 + L4)
      SSN=0043 1-----V-----LOT(L2) = L4
      112 CONTINUE
      C ..
      SSN=0044 1-----V-----V-----CONTINUE
      SSN=0045 1-----V-----V-----112
      C ..
      SSN=0046 1-----V-----LBITT(L2,MX0+1) = LBITT(L2,1)
      111 CONTINUE
      C ..
      SSN=0047 1-----V-----DO 25 NX=1,MX0
      SSN=0048 1-----V-----DO 25 NX=1,MX0
      25 CONTINUE
      C ..
      SSN=0049 1-----V-----DO 25 NX=1,MX0
      SSN=0050 1-----V-----DO 25 NX=1,MX0
      25 CONTINUE
      C ..
      SSN=0051 1-----V-----NX1=NX+MX
      SSN=0052 1-----V-----IF(NX1.GT.MX0) NX1=NX1-MX0
      25 CONTINUE
      C ..
      SSN=0053 1-----V-----DO 24 IXM = 1,JO
      24 CONTINUE
      C ..
      SSN=0054 1-----V-----V     01 = 2((IXM + J2) * ZQ(IXM + J2))
      SSN=0055 1-----V-----IF (LBITT(IXM,NX) .NEqv. LBITT(IXM,NX1)) 01=-01
      24 CONTINUE
      C ..
      SSN=0056 1-----V-----V     SS(NX,MXX)=SS(NX,MXX)+01
      SSN=0057 1-----V-----V-----24 CONTINUE
      SSN=0058 1-----V-----V-----25 CONTINUE
      SSN=0059 1-----V-----V-----26 CONTINUE
      SSN=0060 1-----V-----DO 28 NX=1,MX0
      SSN=0061 1-----V-----DO 28 NX=1,MX0
      SSN=0062 1-----V-----V     SS(NX,NX1)=SS(NX,NX1)/2Z
      SSN=0063 1-----V-----V-----28 CONTINUE
      SSN=0064 1-----V-----30 WRITE(IOUT,600) D
      SSN=0065 1-----V-----WRITE(IOUT,610)  MY0,MY0,MS0,KD1,MS0,IGO,MY0,T
      SSN=0066 1-----V-----DO 32 NX1=1,MX0
      SSN=0067 1-----V-----V     ST(NX1)=0.000
      SSN=0068 1-----V-----V     ST(NX1)=0.000
      SSN=0069 1-----V-----V     ST(NX1)=0.000
      SSN=0070 1-----V-----V-----32 CONTINUE
      SSN=0071 1-----V-----DO 34 NX=1,MX0
      SSN=0072 1-----V-----DO 34 NX=1,MX0
      SSN=0073 1-----V-----V     SA(NX1)=SS(NX,MX1)
      SSN=0074 1-----V-----V     ST(NX1)=ST(NX1)-SA(NX1)
      SSN=0075 1-----V-----V     SQ(NX1)=SQ(NX1)+SA(NX1)*SA(NX1)
      SSN=0076 1-----V-----V     SC(NX1)=SC(NX1)+1.000
      SSN=0077 1-----V-----V     HX2=(MX0+1)-MX1
      SSN=0078 1-----V-----V     SB(NX1)=SS(NX,MX2)
      SSN=0079 1-----V-----V     IF(NX1.GT.MX0) GO TO 34
      SSN=0080 1-----V-----V     ST(NX1)=ST(NX1)+SB(NX1)
      SSN=0081 1-----V-----V     SQ(NX1)=SQ(NX1)+SB(NX1)*SB(NX1)
      SSN=0082 1-----V-----V     SC(NX1)=SC(NX1)+1.000
      34 CONTINUE
      C ..
      SSN=0083 1-----V-----DO 42 NX=1,MX0
      SSN=0084 1-----V-----V     WRITE(IOUT,620) NX,(SA(NX1),NX1=1,MX0)
      SSN=0085 1-----V-----V     WRITE(IOUT,622) (SB(NX1),NX1=1,MX0)
      42 CONTINUE
      C ..
      SSN=0086 1-----V-----S-----36 CONTINUE
      SSN=0087 1-----V-----V     WRITE(IOUT,602)
      SSN=0088 1-----V-----DO 42 NX=1,MX0
      SSN=0089 1-----V-----V     ST(NX1)=ST(NX1)/SC(NX1)
      SSN=0090 1-----V-----V     SQ(NX1)=SQ(NX1)/SC(NX1)
      SSN=0091 1-----V-----V     SQ(NX1)=DSQRT(DABS(SQ(NX1)-ST(NX1))*ST(NX1)))
      42 CONTINUE
      C ..
      SSN=0092 1-----V-----V-----42 CONTINUE
      SSN=0093 1-----V-----V-----WRITE(IOUT,602)
      SSN=0094 1-----V-----WRITE(IOUT,622) (ST(NX1),NX1=1,MX0)
      SSN=0095 1-----V-----WRITE(IOUT,622) (SQ(NX1),NX1=1,MX0)
      SSN=0096 1-----V-----RETURN
      SSN=0097 1-----V-----600 FORMAT('1',3X,'*** CORRELATION FUNCTION OF SPIN GLASS MODEL ***')
      601 FORMAT('1',3X,'***')
      602 FORMAT(' ')
      610 FORMAT(4X,'XO=' I3, 4X,'YO=' I3, 4X,'MS0=' I4, 5X,'T='
      * I4, 3X,'IGO=' I4, 8X,'IGO=' I4, 4X,'MY0=' I4, 10X,'T='
      * F5.2 /)
      C ..

```

図4.2 TOP10EXによるFORTRAN原始プログラムの編集

#### 4.2 TOP10EXの手続き

- ANALYZERの実行詳細解析結果を入力とする場合

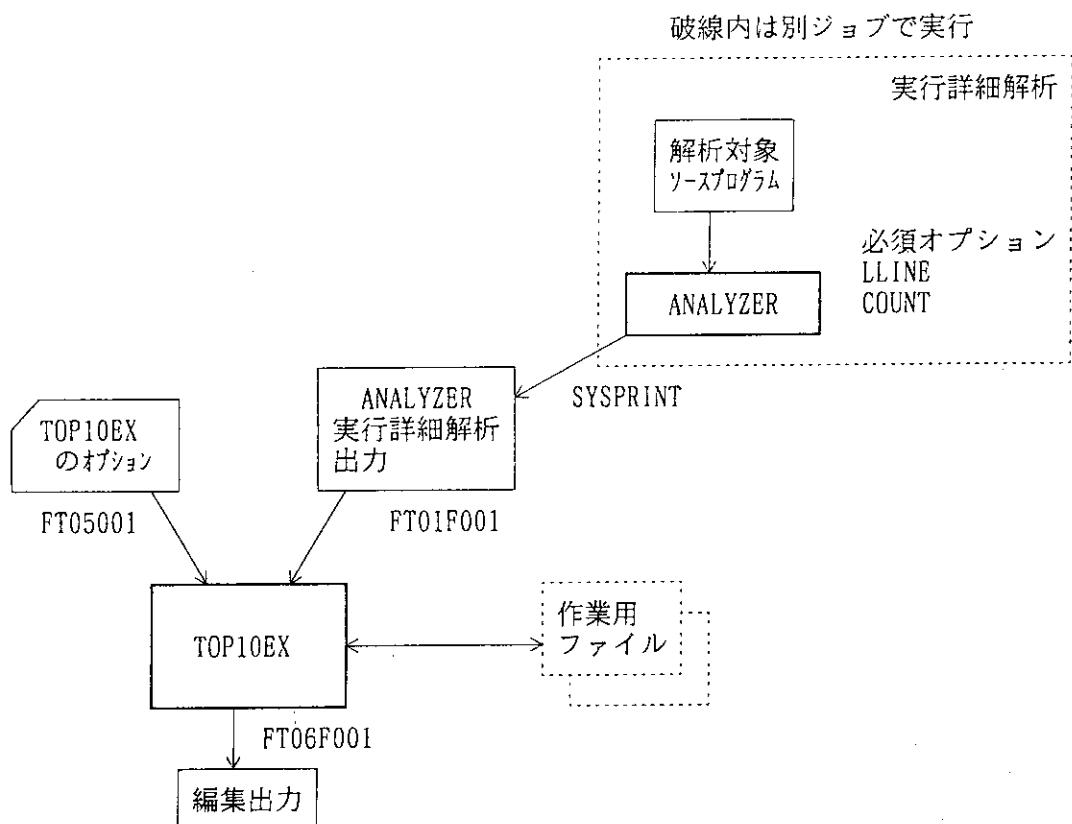


図4.3 TOP10EX の手続き( 実行詳細解析出力を入力する場合 )

- FORTRANソースプログラムを入力とする場合

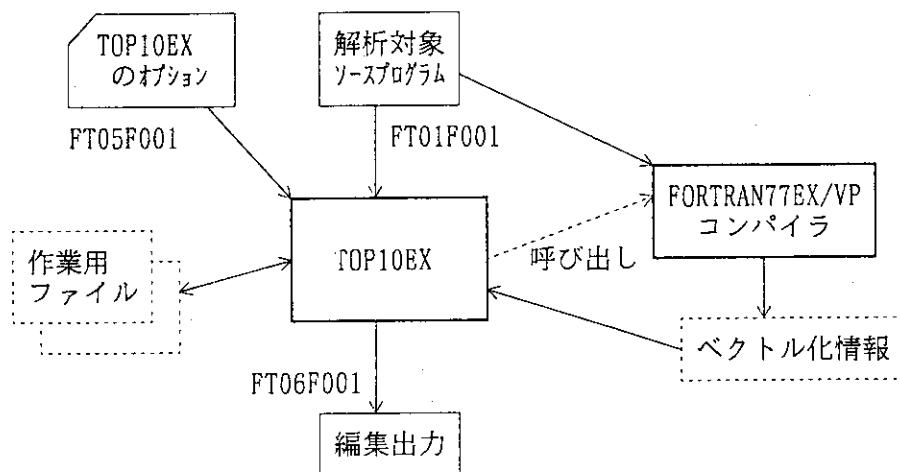


図4.4 TOP10EX の手続き( 原始プログラムを入力する場合 )

#### 4.3 TOP10EXのオプション

TOP10EX(VER.1.1)で指定できるオプションは表4.1の通りです。編集対象がANALYZERの実行詳細解析出力なのかFORTRAN原始プログラムなのかは、入力するデータセットの属性によりTOP10EXが判断します。

表4.1 TOP10EX VER.1.1 のオプション

オプション	機能	編集対象 実行詳細 解析	編集対象 原始プロ グラム
ELM	メンバの選択、あるいはルーチンの選択	○	○
RANK	指定した範囲内のルーチンを選択	○	×
V COST	指定した範囲内のベクトルコストを持つルーチンを選択	○	×
TTL	ANALYZERのTOTAL LIST出力を編集出力	○	×
VOP	ANALYZERのベクトル命令統計情報の編集出力	○	×
LOOP	ANALYZERのループ単位情報の編集出力	○	×
L C	TOP10EXの1頁の最大ライン数	○	○
VP	FORTRAN77EX/VPコンパイラの呼び出し	×	○
TAG	ヘッダに出力するルーチン名を、ルーチン毎に印刷位置をずらして印刷する	○	○
NOUNDER	アンダーライン編集の抑止	○	×

#### 4.3.1 ELMオプション

```
ELM (pgmst)
ELM (*)
```

pgmst : メンバ名、メンバの範囲の並び( カンマで区切って並べる)

TOP 10 EXで出力対象とするメンバ(注)の指定を行います。メンバ指定には①メンバ名を直接指定する方法、②メンバ名の範囲を指定する方法および特定のメンバ名を除外する方法があります。全てのメンバを出力対象とするときは'ELM(\*)'と指定します。メンバ名の範囲指定では“-”の前後のメンバ名を書きます。始まりと終わりを示す名前は必ずしも実在の名前でなくても構いません。

メンバ名は最大長8文字を前提としており、範囲指定がなされた場合、8文字に満たないときは範囲指定の始まりを示すメンバ名にブランクを補い、終わりを示すメンバ名に文字「g」を補います。特定のメンバを除外したいときはメンバ名の前に'-'をつけて下さい。出力順序はメンバ名のEBCDIC順序です。

[指定例]

ELM(A-A, LOCF)	...	頭1文字がAで始まるメンバとLOCF
ELM(AB-INDUCT)	...	A B ~ I N D U C T の範囲のメンバ
ELM(A-Z, -MAIN)	...	MAIN以外のすべてのメンバ
ELM(*)	...	全てのメンバ

注：ここでいうメンバ名は、編集対象が実行詳細解析の出力のとき、又は編集対象の原始プログラムがPSファイル(順編成ファイル)のときにはプログラム単位名を意味しています。編集対象の原始プログラムがPOファイル(区分編成ファイル)のときには、データセットのメンバ名を意味しています。

#### 4.3.2 RANKオプション

```
RANK (i1-i2)
```

指定された範囲( $i_1 - i_2$ )のベクトルコスト順位を持つルーチンのANALYZERの詳細実行解析情報を選択出力する( $i_1, i_2$ ともに正の整数)。

[指定例]

RANK(1-10) または RANK(10-1)  
ベクトルコスト順位が1位から10位までのルーチンを選択する。

#### 4.3.3 VCOSTオプション

VCOST( $r_1 - r_2$ )

指定された範囲( $r_1 - r_2$ )のベクトルコスト比率(%)を持つルーチンの内容を選択出力する ( $r_1, r_2$ 共に正の実定数)。

[指定例]

COST(20.0-90.0) または COST(90.0-20.0)

ベクトルコスト比率が20.0(%)以上で、かつ90.0%以下のルーチンの実行詳細解析の結果を選択する。

#### 4.3.4 TTLオプション

TTL(0|1|2|3)

実行詳細解析のプログラム単位一覧情報(TOTAL LIST)を出力します。サブパラメタによって出力するテーブルを選択することができます。

- 0 : 出力しない
- 1 : コスト・CPUテーブル、CPU・VUテーブルを出力する (図4.5, 図4.6)
- 2 : ベクトル化情報テーブルを出力する (図4.7)
- 3 : 1と2を同時に指定する

<< ANALYZER TOTAL LIST : COST TABLE >>													
RANK	NAME	EX-COUNT	CPU-AVG	VU-AVG	CPU-TIME	X	VU-TIME	Z	V-COST	Z	S-COST	Z	OVERHO
1	STAGE3	3	46.870	7.908	140.611	97.0	23.725	85.4	.2003E11	94.4	.2024E12	83.4	0.0
2	DOOR	3	3.06	1.341	4.219	2.9	4.025	14.5	.1178E10	5.3	.4017E11	16.5	0.0
3	BOND	1	0.001	0.000	0.001	0.0	0.000	0.0	109368	0.0	116697	0.0	4.6
4	MAP	1	0.002	0.000	0.002	0.0	0.000	0.0	58227	0.0	58227	0.0	3.0
5	>FCOMV	2	0.000	0.000	0.000	0.0	0.000	0.0	22351	0.0	245244	0.0	14.0
6	>ICONV	2	-----	-----	0.000	0.0	0.000	0.0	-----	-----	-----	-----	-----
7	>VCONV	1	-----	-----	0.000	0.0	0.000	0.0	-----	-----	-----	-----	-----
8	>BCOUNT	1	-----	-----	0.000	0.0	0.000	0.0	-----	-----	-----	-----	-----
9	>FCOUNT	1	-----	-----	0.000	0.0	0.000	0.0	-----	-----	-----	-----	-----
10	BIN	7	0.000	0.000	0.000	0.0	0.000	0.0	4088	0.0	4088	0.0	62.9
11	MNSITE	1	0.000	0.000	0.000	0.0	0.000	0.0	2742	0.0	23348	0.0	35.8
12	START	3	0.000	0.000	0.000	0.0	0.000	0.0	2369	0.0	11454	0.0	25.8
13	CLEAR	3	0.000	0.000	0.000	0.0	0.000	0.0	1039	0.0	54318	0.0	57.6
14	MAIN	1	0.002	0.000	0.002	0.0	0.000	0.0	300	0.0	300	0.0	0.0
15	BLOCKD	0	0	0	0	0	0	0	0	0	0	0	0
16	BMODEL	0	0	0	0	0	0	0	0	0	0	0	0
17	>CMODEL	0	-----	-----	0	0	0	0	-----	-----	-----	-----	-----
18	>XMODEL	0	-----	-----	0	0	0	0	-----	-----	-----	-----	-----
19	>DMODEL	0	-----	-----	0	0	0	0	-----	-----	-----	-----	-----
20	>EMODEL	0	-----	-----	0	0	0	0	-----	-----	-----	-----	-----
21	>FMODEL	0	-----	-----	0	0	0	0	-----	-----	-----	-----	-----
22	>GMODEL	0	-----	-----	0	0	0	0	-----	-----	-----	-----	-----
23	BTRAN	0	0	0	0	0	0	0	0	0	0	0	0
24	MAP1	0	0	0	0	0	0	0	0	0	0	0	0
25	SAVE	0	0	0	0	0	0	0	0	0	0	0	0
26	>COUNT	0	-----	-----	0	0	0	0	-----	-----	-----	-----	-----
27	SMAP	0	0	0	0	0	0	0	0	0	0	0	0
28	STAGE2	0	0	0	0	0	0	0	0	0	0	0	0
	T (TOTAL)				144.838	100.0	27.751	100.0	.2121E11	100.0	.2425E12	100.0	

図4.5 コスト・CPUテーブル

<< ANALYZER TOTAL LIST : CPU/VU BAR CHART >>			* : CPU RATE IN TOTAL CPU (%)		# : VU RATE IN TOTAL CPU (%)	
RANK	NAME	V-RATE	CPU-TIME	X	VU-TIME	X
1	STAGES	91.5	140.611	97.0	23.725	85.4
2	CORR	98.4	4.219	2.9	4.025	14.5
3	BOND	6.4	0.001	0.0	0.000	0.0
4	MAP	0.0	0.002	0.0	0.000	0.0
5	FCONV	98.2	0.000	0.0	0.000	0.0
6	BIN	0.0	0.000	0.0	0.000	0.0
7	NN SITE	98.3	0.000	0.0	0.000	0.0
8	START	80.4	0.000	0.0	0.000	0.0
9	CLEAR	99.7	0.000	0.0	0.000	0.0
10	MAIN	0.0	0.002	0.0	0.000	0.0
11	BLOCKD	0	0	0	0	0
12	BMODEL	0	0	0	0	0
13	BTRAN	0	0	0	0	0
14	MAPI	0	0	0	0	0
15	SAVE	0	0	0	0	0
16	SHAP	0	0	0	0	0
17	STAGE2	0	0	0	0	0
(TOTAL)		92.6	144.838	100.0	27.751	100.0

図4.6 CPU・VU情報テーブル

<< ANALYZER TOTAL LIST : V-COST BAR CHART >>			* : V-COST RATE (%)			
RANK	NAME	V-LEN	V-RATE	V-EFFECT	V-COST	X
1	STAGES	759	91.5	9.8- 10.3	.2003E11	94.4
2	CORR	971	98.4	31.6- 36.5	.1178E10	5.5
3	BOND	274	6.4	1.0- 1.0	109388	0.0
4	MAP	20	0.0	1.0- 1.0	58227	0.0
5	FCONV	23	98.2	7.2- 13.9	22351	0.0
6	BIN	19	0.0	1.0- 1.0	4088	0.0
7	NN SITE	20	98.3	5.1- 11.2	2743	0.0
8	START	258	80.4	4.7- 4.8	2369	0.0
9	CLEAR	153	99.7	43.1- 59.9	1039	0.0
10	MAIN	1	0.0	1.0- 1.0	300	0.0
11	BLOCKD	0	0	0	0	0
12	BMODEL	0	0	0	0	0
13	BTRAN	0	0	0	0	0
14	MAPI	0	0	0	0	0
15	SAVE	0	0	0	0	0
16	SHAP	0	0	0	0	0
17	STAGE2	0	0	0	0	0
(TOTAL)		92.6	11.0- 11.7	.2121E11	100.0	1

図4.7 ベクトル化情報テーブル

#### 4.3.5 VOPオプション

VOP ( {0 | 1 | 2 | 3} )

実行詳細解析のベクトル命令統計情報を出力します。 但し、実行詳細解析時にVOPオプションを指定して解析された出力に対してのみ有効です。

- 0 : ベクトル命令統計情報を出力しない
- 1 : プログラム単位のベクトル命令統計情報を出力する (図4.8, 図4.9)
- 2 : ループ単位のベクトル命令統計情報を出力する (図4.10)
- 3 : 1と2を同時に指定する

<< ANALYZER TOTAL LIST : VECTOR OPERATION 1 >>								
RANK	NAME	EX-L/ST	EX-LIST	EX-DIST	EX-ADD	EX-MLT	EX-DIV	EX-MASKC
1	STAGE3	.2884E11	0	0	.1335E11	.1025E11	0	.2642E10
2	CDRR	.4979E10	0	1200	.3778E10	.1261E10	60	.1195E10
3	BOND	2466	0	0	2466	0	0	0
4	MAP	0	0	0	0	0	0	0
5	FCONV	7149	4200	5022	48844	22681	820	11680
	>ICONV							
	>VCONV							
	>BCOUNT							
	>FCOUNT							
6	BIN	0	0	0	0	0	0	0
7	NNSITE	3280	0	0	13120	4920	0	1640
8	START	2048	0	0	3072	2048	0	1024
9	CLEAR	20439	0	0	15399	0	0	0
10	MAIN	0	0	0	0	0	0	0
11	BLOCKD	0	0	0	0	0	0	0
12	BMODEL	0	0	0	0	0	0	0
	>CMODEL							
	>AMODEL							
	>DMODEL							
	>EMODEL							
	>FMODEL							
	>GMODEL							
13	BTRAN	0	0	0	0	0	0	0
14	MAPI	0	0	0	0	0	0	0
15	SAVE	0	0	0	0	0	0	0
	>COUNT							
16	SMAP	0	0	0	0	0	0	0
17	STAGE2	0	0	0	0	0	0	0
	(TOTAL)	.3382E11	4200	6222	.1712E11	.1151E11	880	.3837E10

図4.8 ルーチン単位ベクトル命令統計情報 1

<< ANALYZER TOTAL LIST : VECTOR OPERATION 2 >>								
RANK	NAME	EX-MA	EX-VC	EX-MAC	EX-REC	EX-COMP	EX-FUNC	EX-OTHER
1	STAGE3	.1934E10	0	.2642E09	0	0	5040	0
2	CDRR	630	0	.1195E10	0	0	30	0
3	BOND	0	0	0	0	0	0	0
4	MAP	0	0	0	0	0	0	0
5	FCONV	0	0	4980	0	0	0	0
	>ICONV							
	>VCONV							
	>BCOUNT							
	>FCOUNT							
6	BIN	0	0	0	0	0	0	0
7	NNSITE	0	0	0	0	0	0	0
8	START	0	0	0	0	0	0	0
9	CLEAR	0	0	0	0	0	0	0
10	MAIN	0	0	0	0	0	0	0
11	BLOCKD	0	0	0	0	0	0	0
12	BMODEL	0	0	0	0	0	0	0
	>CMODEL							
	>AMODEL							
	>DMODEL							
	>EMODEL							
	>FMODEL							
	>GMODEL							
13	BTRAN	0	0	0	0	0	0	0
14	MAPI	0	0	0	0	0	0	0
15	SAVE	0	0	0	0	0	0	0
	>COUNT							
16	SMAP	0	0	0	0	0	0	0
17	STAGE2	0	0	0	0	0	0	0
	(TOTAL)	.1934E10	0	.1459E10	0	0	5070	0

図4.9 ルーチン単位ベクトル命令統計情報 2

<< STATISTICS OF VECTOR OPERATION FOR EACH LOOP >>											
ISSN	DO-ID	KIND	V	I	EX-L/ST	EX-LIST	EX-DIST	EX-ADD	EX-MLT	EX-DIV	EX-MASKC
100000454-00000457	24	DO	V	I	.4781E10	0	0	.3586E10	.1195E10	0	.1195E10
100000442-00000445	112	DO	V	I	.1857E09	0	0	.1887E09	62914560	0	0
100000451-00000458	25	DO	S	I	0	0	0	0	0	0	0
100000447-00000448	113	DO	V	I	.6291456	0	0	0	0	0	0
100000438-00000439	111	DO	V	I	.3145728	0	3145728	3145728	0	0	0
100000441-00000445	112	DO	S	I	0	0	0	0	0	0	0
100000450-00000458	25	DO	S	I	0	0	0	0	0	0	0
100000435-00000459	24	DO	S	I	0	0	0	0	0	0	0
100000473-00000483	34	DO	V	I	.9600	0	1200	.4200	1200	0	.600
100000462-00000463	28	DO	V	I	.2280	0	0	0	.1140	0	0
100000472-00000486	36	DO	S	I	0	0	0	0	0	0	0
100000431-00000432	12	DO	V	I	.2520	0	0	.2520	0	0	0
100000489-00000492	42	DO	V	I	.240	0	0	.30	0	.60	0
100000430-00000432	12	DO	V	I	0	0	0	0	0	0	0
100000461-00000463	28	DO	V	I	0	0	0	0	0	0	0
100000401-00000502	-----	TOTAL	1	0	0	0	0	90	0	0	0
100000467-00000470	32	DO	V	I	.90	0	0	0	0	0	0

<< STATISTICS OF EXECUTION FOR EACH LOOP >>											
ISSN	DO-ID	KIND	V	I	EX-MA	EX-VC	EX-MAC	EX-REC	EX-COMP	EX-FUNC	EX-OTHER
100000454-00000457	24	DO	V	I	0	0	.1195E10	0	0	0	0
100000442-00000445	112	DO	V	I	0	0	0	0	0	0	0
100000451-00000458	25	DO	S	I	0	0	0	0	0	0	0
100000447-00000448	113	DO	V	I	0	0	0	0	0	0	0
100000438-00000439	111	DO	V	I	0	0	0	0	0	0	0
100000441-00000445	112	DO	S	I	0	0	0	0	0	0	0
100000450-00000458	25	DO	S	I	0	0	0	0	0	0	0
100000435-00000459	24	DO	S	I	0	0	0	0	0	0	0
100000473-00000483	34	DO	V	I	.600	0	0	0	0	0	0
100000462-00000463	28	DO	V	I	0	0	0	0	0	0	0
100000472-00000486	36	DO	S	I	0	0	0	0	0	0	0
100000431-00000432	12	DO	V	I	0	0	0	0	0	0	0
100000489-00000492	42	DO	V	I	.30	0	0	0	0	.30	0
100000430-00000432	12	DO	V	I	0	0	0	0	0	0	0
100000461-00000463	28	DO	V	I	0	0	0	0	0	0	0
100000401-00000502	-----	TOTAL	1	0	0	0	0	0	0	0	0
100000467-00000470	32	DO	V	I	0	0	0	0	0	0	0

図4.10 ループ単位ベクトル命令統計情報

#### 4.3.6 LOOPオプション

実行詳細解析のループ単位一覧情報を出力します。 ループ単位一覧情報に含まれる平均回転長、ルーチンに占めるベクトルコストの率、スカラコストの率、ベクトル化効果、ベクトル化率の情報は、TOP10 EX の文単位情報出力に併せて出力されます。 ループ単位の情報をテーブルの形で出力したい場合はLOOPオプションを指定して下さい。

<< STATISTICS OF EXECUTION FOR EACH LOOP >>												
ISSN	DO-ID	KIND	V	I	EX-COUNT	V-COST	X	S-COST	X	V-LENG	V-RATE	V-EFFECT
100000454-00000457	24	DO	V	I	1167360	.1142E10	96.9	.3884E11	96.6	1024	98.4	31.5- 36.4 I
100000442-00000445	112	DO	V	I	61440	17924376	1.5	.1258E10	3.1	1024	100.0	60.4- 81.8 I
100000451-00000458	25	DO	S	I	61440	16926720	1.4	.16926720	0.0	19	0.0	1.0- 1.0 I
100000447-00000448	113	DO	V	I	3072	.403298	0.0	.28311552	0.0	1024	100.0	60.4- 81.8 I
100000438-00000439	111	DO	V	I	3072	.313676	0.0	.22020096	0.0	1024	100.0	60.4- 81.8 I
100000441-00000445	112	DO	S	I	3072	.307200	0.0	.307200	0.0	20	0.0	1.0- 1.0 I
100000450-00000458	25	DO	S	I	3072	.307200	0.0	.307200	0.0	20	0.0	1.0- 1.0 I
100000435-00000459	26	DO	S	I	3	.82944	0.0	.82944	0.0	1024	0.0	1.0- 1.0 I
100000473-00000483	34	DO	V	I	60	.9781	0.0	.37800	0.0	10	.99.6	1.9- 5.7 I
100000462-00000463	28	DO	V	I	60	.3039	0.0	.29640	0.0	19	100.0	5.5- 13.6 I
100000472-00000486	36	DO	S	I	3	.1200	0.0	.1200	0.0	20	0.0	3.0- 1.0 I
100000431-00000432	12	DO	V	I	60	.1184	0.0	.27720	0.0	42	100.0	17.5- 29.2 I
100000489-00000492	42	DO	V	I	3	.738	0.0	.2880	0.0	10	100.0	1.9- 5.8 I
100000430-00000432	12	DO	V	I	3	.300	0.0	.300	0.0	20	0.0	1.0- 1.0 I
100000461-00000463	28	DO	V	I	3	.300	0.0	.300	0.0	20	0.0	1.0- 1.0 I
100000401-00000502	-----	TOTAL	1	0	282	0.0	282	0.0	3	0.0	1.0- 1.0 I	
100000467-00000470	32	DO	V	I	3	.69	0.0	.270	0.0	10	100.0	1.9- 5.8 I

図4.11 ループ単位情報

#### 4.3.7 VPオプション

VP ( options )

options : 翻訳オプションの並び (コンマで区切って並べる)

FORTRAN77EX/VPコンパイラを呼び出す際に、コンパイラに渡す翻訳オプションを指定します。標準値は以下のようにになっており、これらのオプションと矛盾しないものを追加する形で指定することができます。但し、VMSGを指定した場合についてのみ、標準値のNOVMSGは打ち消されます。

FORTRAN77EX/VPコンパイラ呼び出し時のオプションの標準値

NOOBJ, S, VP(2600), SSN(D), ELM(\*), NOVMSG

#### 4.3.8 TAGオプション

TOP10EXでは、ヘッダにメンバ名を出力しますが、メンバが変わることにヘッダに印刷されるメンバ名の位置をずらしながら印刷します。

```
#####
## BIN ##
#####
*** VECTORIZED SOURCE LIST ***
SSN EX-COUNT TRUE V-COST S-COST           VO C-----1-----2-----3-----4-----5-----6-----7-
#
## BLOCKD ##
#####
*** VECTRIZE STATEMENT LIST ***
SSN EX-COUNT TRUE V-COST S-COST           VO C-----1-----2-----3-----4-----5-----6-----7-
#
## BMODEL ##
#####
*** VECTRIZE STATEMENT LIST ***
SSN EX-COUNT TRUE V-COST S-COST           VO C-----1-----2-----3-----4-----5-----6-----7-
#
## BOND ##
#####
*** VECTRIZE STATEMENT LIST ***
SSN EX-COUNT TRUE V-COST S-COST           VO C-----1-----2-----3-----4-----5-----6-----7-
```

図4.12 TAGオプションを指定した場合のヘッダ出力

4.3.9 L C オプション

**L C (62 | n)**

TOP10EXで出力される情報の一頁の最大ライン数を指定します。通常のプリント出力では、62行が適しています。コンパクト出力では108(TYPE2)を指定すると良いでしょう。但し、コンパクト出力の場合、TOP10EXが付加したアンダーラインは表示されない箇所があります。

4.3.10 NOUNDERオプション

**NOUNDER**

アンダーライン編集を行いません。実行詳細解析を編集したリストをコンパクト出力する場合に効果的です。

4.4 TOP10EXの使用4.4.1 カタログド・プロジェクト

TOP10EXを使うためのカタログTOP10EXが用意されています。

**● TOP10EX**

位置パラメタ	用 途	標準値
DSN	実行詳細解析の結果、或いはFORTRAN ソースプログラムのデータセット	
INC	インクルードファイル	NULLFILE
OUT6	TOP10EX による編集出力の出力先	SYSOUT=*

※ DSNにFORTRAN ソースプログラムを指定した場合で、かつそのプログラムがインクルードファイルを使用している場合のみ必要

TOP10EX のオプションはDD名SYSIN に指定されたデータセットに記述して下さい。

#### 4.4.2 使用例

- 実行詳細解析の結果を入力とする場合

```
//TOP10EX EXEC TOP10EX,DSN='J****.TEST.ANAOUT'
//SYSIN   DD *
      TTL, ELM(*), VOP, TAG
/*
```

- ソースプログラムを入力とする場合

```
//TOP10EX EXEC TOP10EX,DSN='J****.TEST.FORT77'
//SYSIN   DD *
      ELM(*), VP(VMSG)
/*
```

#### 4.5 T O P 1 0 E X の使用上の注意

- T O P 1 0 E Xでは、標準の FORTRAN77の仕様を外れたプログラムに対して正常な動作を保証しません。
- 原始プログラムを入力して、V Pオプションを指定する場合、入力する原始プログラムが大きいと、メモリが不足してジョブが正常に終了しない（ベクトル情報が付加されない等）場合があります。 その時は、メモリを拡張して再実行して下さい（アドレス拡張域も使用できます）。
- T O P 1 0 E Xの仕様は、バージョンアップに伴い変更される場合があります。

## おわりに

本報告書を用いた講習会は、東海研究所と那珂研究所において計3回行われ、100人以上の参加者が得られた。多くの人が、新しいコンパイラV12に興味を持っている様子であった。

原子力コード16本を使った新コンパイラのベンチマークテストでは、コンパイラオプションの変更、あるいは非互換項目によるプログラムの修正等の作業が必要であった。これらの事例は、新コンパイラへ移行する際の例題として参考になると思われる。また、ベクトル化技法については過去8年間、情報システムセンター外来研に蓄積されたものである。今後、新コンパイラを用いて実行したり、或いはベクトル化を進めていく際に本報告書を役立てて頂ければ幸いである。

## 謝辞

情報システムセンター室長秋元正幸氏には、本報告書を書く機会を与えて頂くと共に、講習会を開催する機会を与えて頂きました。ここに、深く感謝致します。

本報告書を作成するにあたって、貴重なご意見を賜りました情報システムセンターの富山峯秀氏、中村康弘氏、小沼吉男氏、庄司誠氏、富士通(株)の青木正樹氏、相沢広氏、折居茂夫氏を始めとする諸氏に深く感謝致します。富山氏には、講習会開催の便宜も図って頂きました。深く感謝致します。また、外来研の山田圭子さんには、本報告書作成にあたって御助力頂きました。深く感謝致します。

講習会当日は、富士通(株)の伊奈博氏に講演の一部をお願い致しました。ここに、深く感謝致します。

## おわりに

本報告書を用いた講習会は、東海研究所と那珂研究所において計3回行われ、100人以上の参加者が得られた。多くの人が、新しいコンパイラV12に興味を持っている様子であった。

原子力コード16本を使った新コンパイラのベンチマークテストでは、コンパイラオプションの変更、あるいは非互換項目によるプログラムの修正等の作業が必要であった。これらの事例は、新コンパイラへ移行する際の例題として参考になると思われる。また、ベクトル化技法については過去8年間、情報システムセンター外来研に蓄積されたものである。今後、新コンパイラを用いて実行したり、或いはベクトル化を進めていく際に本報告書を役立てて頂ければ幸いである。

## 謝 言

情報システムセンター室長秋元正幸氏には、本報告書を書く機会を与えて頂くと共に、講習会を開催する機会を与えて頂きました。ここに、深く感謝致します。

本報告書を作成するにあたって、貴重なご意見を賜りました情報システムセンターの富山峯秀氏、中村康弘氏、小沼吉男氏、庄司誠氏、富士通(株)の青木正樹氏、相沢広氏、折居茂夫氏を始めとする諸氏に深く感謝致します。富山氏には、講習会開催の便宜も図って頂きました。深く感謝致します。また、外来研の山田圭子さんには、本報告書作成にあたって御助力頂きました。深く感謝致します。

講習会当日は、富士通(株)の伊奈博氏に講演の一部をお願い致しました。ここに、深く感謝致します。

## 参考文献

- 1) 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〕, JAERI-memo 59-215, 1984
- 2) 富士通(株); VP FORTRANプログラミング, 教育事業部テキスト UV000L1N\*4, 1991
- 3) 富士通(株); FORTRAN77EX使用手引書 V12用, マニュアル 79SP-5031-1, 1991
- 4) 富士通(株); FORTRAN77EX/VP使用手引書 V12用, マニュアル 79SP-5041, 1991
- 5) 富士通(株); アナライザ使用手引書(FORTRAN, VP用) V10L20用,  
マニュアル 79SP-5080, 1991
- 6) 富士通(株); FORTRAN77使用手引書 V10用, マニュアル 78SP-5300, 1991
- 7) 富士通(株); FORTRAN77/VP使用手引書 V10用, マニュアル 78SP-5680, 1991
- 8) 富士通(株); FORTUNE使用手引書 V10用, マニュアル 70SP-5730-2-5, 1991
- 9) 富士通(株); FORTRAN77文法書 1990年6月版, マニュアル 99SP-8031, 1991
- 10) 情報システムセンタ・外来研;  
原子力コードのベクトル化基礎 FORTRAN77/VP V10L20対応, 講習会資料, 1986
- 11) 富士通(株)システム部; VP FORTRANプログラミング, 技術資料, 1987
- 12) 富士通(株)PP事業部第4開発部; FORTRAN77 EX/VPの御紹介, 技術資料, 1991
- 13) 富士通(株)PP事業部第4開発部; FORTRAN77EX V12 非互換項目, 技術資料, 1991
- 14) 富士通(株)PP事業部第4開発部; FORTRAN77互換性の改善について, 技術資料, 1991

## 付録A 非互換項目資料

F O R T R A N 7 7 互換性の改善について  
《M S P / M V S 版》

### 1. 互換性の考え方

FORTTRAN 77 EXでは、基本的にはFORTTRAN 77文法書に記載されている仕様は全てサポートしています。しかし、FORTTRAN 77文法書に記載されていない旧仕様の内、明らかに退化機能と見なせる仕様については、顧客資産の将来に向けての保護、流通性及び処理系開発の軽量化の観点から、サポートを中止しています。

### 2. FORTTRAN 77互換性改善

FORTTRAN 77 EX V11では、以下の仕様を非互換と致しました。

- ① 明らかな文法エラーである仕様、
- ② ファイル破壊の可能性がある仕様、
- ③ 旧仕様から新仕様への移行を強く推奨する仕様

しかし、実運用上はFORTTRAN 77互換を改善する必要があることを認識し、下記の観点から重要なと思われる仕様についてV12で改善しております。

- ① 慣例として処理系が認めており、非互換にすると自由度が大きく下がるもの。
- ② FORTTRAN 77との媒体非互換を発生させるもの。
- ③ FORTTRAN 77と実行結果が異なるもの。ただし、FORTTRAN 77で値が不定なものや、FORTTRAN 77 EXで機能拡張したものは除く

次項より、互換改善項目及び非互換項目一覧表を示します。

#### 記号の説明

<非互換の検出> FORTTRAN77 EXに移行した場合の非互換の検出箇所を示します。

- 翻 : 翻訳時に検出される。
- 結 : 結合編集(LKED/LOADER) 時に検出される。
- 実 : 実行時に検出/発生する。(EXライブリの再結合でも発生します。)

<管理番号> 非互換項目の管理番号(ソフトウェア説明書)に対応しています。

<F77モ動作> I、W、E、S: それぞれのレベルのエラーメッセージを出力する。  
—はFORTTRAN77 EXではエラーが出力されていないことを示す。

<非互換理由> V12で未吸収の非互換についての分類を示します。

- ① : 明らかな文法エラーである仕様を示す。
- ② : ファイル破壊につながる可能性のある仕様を示す。
- ③ : 旧仕様から新仕様への移行を強く推奨する仕様を示す。
- ④ : その他  
1. 機能拡張 2. 仕様の統一 3. チェック強化 4. 誤動作防止

## 3. V1.2での非互換項目

V1.2で残る非互換について、以下に記述します。結果が異なる項目およびエラーの出力の有無に分類して記述してあります。基本的には非互換の影響は少ないと考えていますが、網掛け部分の非互換については、注意が必要です。

## (1)正常終了で実行結果が異なる項目

## (a)FORTRAN77の仕様を変更している項目

本非互換はエラーメッセージが出力されないため、ユーザへの影響がありますので、移行する際は注意が必要です。

No	非互換項目名	検出	管理番号	非互換理由	P77での動作	備考
1	オームリストREAD文における空項目	実	HMSPI-10193	④-2	—	空項目の仕様統一
2	エラー発生時のIOSTAT指定子の値	実	HMSPI-10188	④-1	—	一部エラー出力

## (b)FORTRAN77の値が不定であった項目又は拡張機能によりFORTRAN77 EXで値を設定した項目

本非互換はエラーメッセージが出力されませんが、FORTRAN77では値が不定な場合について、FORTRAN77 EXで値を確定させた非互換であり、影響は少ないと考えています。

No	非互換項目名	検出	管理番号	非互換理由	P77での動作	備考
1	リスト指示READ文におけるCOMPLEX*32に対する実数データの入力	実	HMSPI-10195	④-2	—	COMPLEX*8,16と仕様の統一
2	リスト指示READ文における文字型以外の配列名に対する文字データの入力	実	HMSPI-10194	④-2	—	
3	高速入出力機能で指定されたNUM指定子の値	実	HMSPI-10687	④-1	—	拡張部分が非互換
4	REDLENサービスサブルーチンの返却値	実	HMSPI-10210	④-1	—	拡張部分が非互換
5	INQUIRE文で確定値が返却できない場合の返却値	実	HMSPI-10209	④-2	—	値が確定しない指定子への値返却

## (2) I, W又はEレベルのエラーが発生するが、実行結果が異なる項目

本非互換は、I、W又はEレベルのエラーメッセージが出力されます。

No	非互換項目名	検出	管理番号	非互換理由	P77での動作	備考
1	エラー識別番号の変更 エラー識別番号及びエラー処理サービスサブルーチンに記述したエラー番号	実	HMSPI-10213	④-1	—	ERRMON,ERRSET, ERRSAV,ERRSTR, ERRCMM
2	INQUIRE文のNEXTREC指定子の返却値	実	HMSPI-10208	④-2	—	最大記録長を超えた場合
3	実行時BCDコードデータの扱い	実	HMSPI-10190	③	—	
4	文字型の入出力並びに対する論理、整数及び実数編集	実	HMSPI-10197	④-2	W	
5	直接アクセスにおける記録変数の値	実	HMSPI-10207	③	E	総記録数を超えた

No	非互換項目名	検出	管理番号	非互換	F77	備考
				理由	での	
6	書式付き入出力文と書式なし入出力文の混在 (REWIND/ENDFILE 文がなく混在した場合)	実	HMSPI-11047	②	—	
7	区分編成ファイルの既存メンバに対する入出力 (READ 後 WRITE の実行)	実	HMSPI-11046	②	—	
8	文字書式仕様のエラーチェック	実	HMSPI-10201	①	—	
9	OPEN文のACTION指定子にWRITE を指定した 場合の、入出力文の文の種類のチェック	実	HMSPI-10199	④-2	—	
10	浮動小数点アンダーフローエラーのエラーレベル	実	HMSPI-10183	④-2	W	
11	母国語型の組込み関数LEN 及びINDEX	翻	HMSPI-10277	③	—	

(3) 実行時 S / U レベルのエラーが output され実行が打ち切られる。

No	非互換項目名	検出	管理番号	非互換	F77	備考
				理由	での	
1	IBSET, IBCLR 及びBTTEST 関数の第2引数に範囲外の値が指定された場合	実	HMSPI-11045	①	—	V12 ではISHFT は互換を取る
2	A E オプション指定時の拡張域の確保 [AMODE (24) の場合]	実	HMSPI-10893	③	—	

(4) 翻訳時 S レベルのエラーが output される。

No	非互換項目名	検出	管理番号	非互換	F77	備考
				理由	での	
1	AQL 仕様	翻	HMSPI-10292	③	—	SQL の利用
2	*INCLUDE文のFIXED/FREE	翻	HMSPI-10288	③	—	
3	並びのないSAVE文と並びのあるSAVE文の混在	翻	HMSPI-10276	①	—	
4	DATA文で添字の範囲が超えている	翻	HMSPI-10274	③	W 動作	
5	旧形式の16進定数	翻	HMSPI-10282	③	W 動作	
6	FORTRAN77 におけるE レベルエラーの動作	翻	HMSPI-10271	①	E	
7	型宣言した名前をCALL文で引用	翻	HMSPI-10275	①	W	
8	共通ブロック名と組込み関数名の一致	翻	HMSPI-10287	①	W/S	
9	DATA文のDO形並び要素の配列名	翻	HMSPI-10289	①	W	
10	PARAMETER 文による論理型と算術型の混在	翻	HMSPI-11067	①	W	

## (5) L K E D時に未定義／未解決となる項目

以下の非互換については、結合編集時にエラーが出力されますので影響範囲は明確になります。

No	非互換項目名	検出	管理番号	非互換 F77		備考
				理由	での動作	
1	組込み関数名と同じ名前の外部関数名	結	HMSM-10270	①	—	
2	FORTRAN IVオブジェクト互換	結	HMSM-10269	①	—	
3	P L / Iとの言語間結合	結	HMSM-10184	①	—	

## (6)その他(運用、エラーレベルの相違等)

以下の非互換については、実行時のべき乗演算の精度／エラーレベルの相違等でありプログラムへの影響は少ないと考えています。

No	非互換項目名	検出	管理番号	非互換 F77		備考
				理由	での動作	
1	実数型のべき乗演算の精度が異なる	翻	HMSM-10280	③	—	
2	W又はIレベルのメッセージが出力される	翻	HMSM-10281	③	—	
3	長さ0の文字定数	翻	HMSM-10273	①	E	
4	けた移動子(スケールファクタ)のエラーリカバー	実	HMSM-10191	③	—	
5	J I S C O B O Lとの言語間結合	実	HMSM-10185	③	—	
6	実行可能プログラムオプション	実	HMSM-11048	③	—	旧オプションでも動作可
7	書式付きREAD文におけるアストロフィ及びH形編集のつかい	実	HMSM-10196	①	—	動作は同じ
8	未創成のV S A Mデータセットに対して順次READ文又は直接READ文を実行した場合	実	HMSM-10198	④-3	S	動作は同じ
9	OPEN文のR E C L指定子の値とVSAM(RRDS)定義値のレコード長が矛盾した場合	実	HMSM-10200	④-2	S	動作は同じ
10	直接入出力でファイルの初期化時に指定された値分のスペースが確保できない場合	実	HMSM-10206	④-2	E	動作は同じ
11	新規ファイルに対するREAD文の実行	実	HMSM-10205	④-4	—	誤動作の防止
12	論理値以外を持つ論理演算の値	実	HMSM-11066	④-1	—	

## 〔備考〕

以下の非互換項目（一覧表の網掛の部分）については、結果が異なる等の影響がありますが、現FORTRAN77 EX仕様を継続する方が良いと考えています。

- ❶ ネームリストREAD文における空項目 (HMSPI-10193)  
FORTRAN77では空項目に対してゼロを代入しているが、FORTRAN77 EXでは空項目に対しては、何も代入しない（値は変更しない）。  
変更した理由は、空項目の扱いをリスト指示入出力文と仕様を統一するためです。
- ❷ エラー発生時のIOSTAT指定子の値 (HMSPI-10188) [一部エラーが出力される]  
FORTRAN77 EXでエラー識別番号を変更したことおよびIOSTATに代入する値を完了コードからエラー識別番号に変更したために、FORTRAN77と非互換になっています。  
IOSTATの値を完了コードからエラー識別番号に変更したのは、プログラムの流通性を向上（システムによって返却値が異ならないようにする）させるためです。
- ❸ エラー識別番号の変更 (HMSPI-10213) [エラーメッセージは出力される]  
エラー識別番号は、以下の理由で変更しています。
  - (a) FORTRAN77のエラー識別番号は一つのエラー識別番号で複数のエラーを検出するケースがあり、分かりづらいと御指摘を受けてきました。
  - (b) 分かりやすいエラーとするために、エラーの種別に対応してエラー識別番号を割り振りました。
 エラー識別番号の変更に伴いエラー処理サービスサブルーチン(ERRMON, ERRSET, ERRSTR, ERRSAVE, ERRTRA, ERRCMN)を引用した場合に、実行時に診断メッセージを出力しています。

以 上

【非互換項目票】			管理番号 H M S P - 1 0 2 1 3 - 0 5
O S , 製品, E / V L , P T F	移行前 O S IV / F 4 M S P F O R T 7 7 V 1 0 L 3 1	移行後 O S IV / M S P F O R T 7 7 E X V 1 0 L 1 0	
製品内機能大分類	実行時ライブラリ	製品内機能小分類	エラーモニタ
非互換項目名	エラー識別番号		
発生の条件 (製品組合せ)	エラー発生時、エラー識別番号を引数として指定できるエラー処理サービスサブルーチンを実行した場合		
非互換発生の理由	1 機能追加 5 障害修正	2 機能改善 6 機能体系の変更	3 性能改善 7 (旧処理系の仕様矯正) 4 チェック強化
対処方法の分類	1 ジョブ制御文 / C L の変更 4 再翻訳 7 システムパラメタ変更	2 制御データの変更 5 再結合編集 8 オペレータの慣れ	3 原始プログラムの変更 6 ファイルの変更 9 ( )
対処すべき人	1 システム管理者 4 センタオペレータ	2 システムプログラマ 5 エンドユーザー	3 一般プログラマ 6 ( )
対処しない場合の結果	1 資産破壊 3 異常終了 5 資源 / 性能の変化	2 処理結果の相違 4 起動が不可能 6 変化を感じる	異常時のメッセージ / 完了コード

## 【概要】

実行時のエラー識別番号が F O R T R A N 7 7 と F O R T R A N 7 7 E X で異なる。  
このため、エラー識別番号を引数として指定できるエラー処理サービスサブルーチンを実行した時正常に動作しなくなる。

【項目】	【移行前】	【移行後】
エラー識別番号を引数として指定できるエラー処理サービスサブルーチンを実行した場合。	F O R T R A N 7 7 エラー識別番号	F O R T R A N 7 7 E X エラー識別番号
- F O R T R A N システムで定めているエラー識別番号。	J Z L 1 3 0 I - S ~ J Z L 3 0 1 I - E の範囲内のもの	J W E 0 0 0 5 I - S ~ J W E 0 3 3 0 I - W の範囲のもの、
- システム編集時に追加できるエラー識別番号。	J Z L 3 0 2 I ~ J Z L 8 9 9 I の範囲のもの。	J W E 0 3 5 1 I ~ J W E 0 3 9 9 I の範囲のもの。
- 入力欄整数型の値が範囲外である。	J Z L 2 0 6 I - E	J W E 0 1 7 1 I - E
- 浮動小数点オーバーフローが発生した。	J Z L 2 0 7 I - E	J W E 0 0 1 1 I - E
- 浮動小数点アンダーフローが発生した。	J Z L 2 0 8 I - W	J W E 0 0 1 2 I - E
- 浮動小数点ゼロ割りが発生した。	J Z L 2 0 9 I - E	J W E 0 0 1 3 I - E
- 固定小数点ゼロ割りが発生した。	J Z L 2 0 9 I - E	J W E 0 0 1 4 I - E
- 固定小数点オーバーフローが発生した。	J Z L 2 1 0 I - E	J W E 0 0 1 5 I - E
- 入力欄の整数型 / 実数型データに 1 0 進数でない文字を指定した。	J Z L 2 1 5 I - W	J W E 0 1 7 3 I - W
- 入力欄に 1 6 進数字でない文字を指定した。	J Z L 2 2 5 I - W	J W E 0 1 7 4 I - W
- 入力欄の実数型データの絶対値が範囲外である。	J Z L 2 2 6 I - E	J W E 0 1 7 6 I - E
- F O R T R A N 記録の長さが論理レコードを超えている。	J Z L 2 1 2 I - E	J W E 0 1 3 2 I - E
- 直接入出力で記録番号の値が最大記録番号を超えている。	J Z L 2 3 2 I - E	J W E 0 0 8 2 I - E
- DD名が不足している。	J Z L 2 1 9 I - S	J W E 0 0 2 8 I - S

## 【対処方法】

エラー処理サービスサブルーチンの引数に指定されているエラー識別番号が、  
FORTRAN 77系ならば、FORTRAN 77 EX系のエラー識別番号に修正する必要がある。  
変更が必要なケースは次のものがある。

- エラー処理サービスサブルーチン（除くERRTRA）の第一引数に記述されたエラー識別番号
  - IOSTAT指定子の値を参照している場合
  - 利用者修正サブルーチンの第二引数の値を参照している場合
- 原始プログラムの修正例については、添付資料を参照されたい。  
又、エラー処理サービスサブルーチンのエラー識別番号に対しては、実行可能プログラムオプション（ERRPRT/NOERRPRT）が用意されている。これは、エラー処理サービスサブルーチン実行時に「FORTRAN 77とFORTRAN 77 EXでエラー識別番号が非互換である」旨のメッセージの出力を制御するものである。省略値はERRPRTで、当該メッセージを出力する。

エラー識別番号を修正した後、NOERRPRTオプションを指定すれば、「エラー識別番号が非互換である」旨のメッセージは出力されない。また、エラー処理サービスサブルーチンも正常に動作する。

(詳細説明 有)

## 【添付資料】

例) 索引WRITE文における重複キー状態エラーの例

FORTRAN 77 エラー識別番号 159 を意識している原始プログラムを、  
FORTRAN 77 EX 上で動作するように修正する。

(移行前の原始プログラム)

```
PROGRAM MAIN
EXTERNAL SUB
INTEGER ENO/159/           ! エラー処理サービスサブルーチンの第一引数となる。
CALL ERRSET(ENO, 1, 1, 1, SUB) ! 159は、FORTRAN77 のエラー識別番号である。
:
WRITE(10, IOSTAT=IOS) A    ! 索引WRITE文で重複キー状態エラー発生。
IF(IOS, EQ, 159) GOTO 100  ! IOSTAT 指定子の値を参照。
:
100 STOP
END
SUBROUTINE SUB(RET, INO)    ! 第二引数にエラー識別番号が渡る。
:
IF(INO, EQ, 159) THEN      ! 利用者修正サブルーチンの第二引数の値を参照。
  PRINT *, 'ERR-NO=', INO, ' OCCURS.'
END IF
END
```

(移行後の原始プログラム)

```
PROGRAM MAIN
EXTERNAL SUB
INTEGER ENO/ 25/            ! FORTRAN77 EX のエラー識別番号に修正した。
CALL ERRSET(ENO, 1, 1, 1, SUB) ! エラー識別番号 25 の処理を行う。
:
WRITE(10, IOSTAT=IOS) A    ! エラー発生により、IOS にはエラー識別番号25が
IF(IOS, EQ, 25) GOTO 100  ! 設定される。
:
100 STOP
END
SUBROUTINE SUB(RET, INO)
:
IF(INO, EQ, 25) THEN       ! エラー識別番号 25 に対応する処理を行う。
  PRINT *, 'ERR-NO=', INO, ' OCCURS.'
END IF
END
```

## F O R T R A N 7 7 E X V 1 2 非互換項目 O H P 資料

## &lt; 目 次 &gt;

H M S P - 1 0 1 9 3 -----	[ネームリストREAD文に於ける空項目]
H M S P - 1 0 1 8 8 -----	[エラー発生時のIOSTAT指定子の値]
H M S P - 1 0 1 9 4 -----	[リスト指示READ文での文字型以外の配列名に対する文字データ]
H M S P - 1 0 2 7 9 -----	[論理値を持つ論理演算の値]
H M S P - 1 0 2 1 3 -----	[実行時のエラー識別番号の変更]
H M S P - 1 0 1 0 7 -----	[文字型の入出並びに対応する論理、整数及び実数編集]
H M S P - 1 0 1 8 9 -----	[書式付きと書式なし入出力文の混在]
H M S P - 1 0 2 1 2 -----	[区分編成ファイルの既存メンバに対する入出力]
H M S P - 1 0 2 0 1 -----	[文字書式仕様のエラーチェック]
H M S P - 1 0 1 9 9 -----	[OPEN文のACTIONにWRITEを指定した時の入出力文のチェック]
H M S P - 1 0 1 8 3 -----	[浮動小数点アンダーフローエラーのエラーレベル]
H M S P - 1 0 2 7 7 -----	[母国語の組込み関数LENおよびINDEX]
H M S P - 1 0 2 1 4 -----	[IBSET, IBCLR, BTEST関数の第2引数に範囲外の値]
H M S P - 1 0 2 7 2 -----	[PARAMETER文による論理型と算術型の混在]
H M S P - 1 0 2 7 0 -----	[結合編集時に組込み関数名が未解決になる]
H M S P - 1 0 2 6 9 -----	[FORTRAN IVオブジェクト互換]

## 現象：処理結果のオ류述 (HSP-10193)

## 【非互換の内容】

ネームリスト入力データ中に空項目が出現した場合の異常作業となる。

- F O R T R A N 7 7  
空項目に対する並びに0を代入する。
- F O R T R A N 7 7 E X  
空項目に対する並びの値は変わらない。

## 【非互換の現れ方】

例：  
INTEGER\*4 A(3) /9999, 9999, 9999/  
NAMELIST/NN/A  
READ (5, FMT=NN)

入力データ  
&NN  
A=100,.300  
&END

F O R T R A N 7 7 では、A(2) は0になる。  
F O R T R A N 7 7 E X では、A(2) は9999になる。

## 【非互換の検出方法】

なし。

## 【非互換の回避方法】

入力データの空項目を0に修正しておく。又は、  
入力並びを0で初期化しておく。

**現象： エラー発生時の処理結果との相違** (HSP-10188)

### I O S T A T 指定子（その 1）

#### 【非互換の内容】

助動的割り当てエラー発生時、 I O S T A T に返却する値が異なる

- F O R T R A N 7 7  
8 (助動的割り当てノーチンの復帰値) を返却する。
- F O R T R A N 7 7 E X  
23 (メッセージ番号) を返却する。

#### 【非互換の現れ方】

例：

```
INTEGER IOS
OPEN(1, FILE=' FORT. DATA', IOSTAT=IOS)
```

助動的割り当てエラーが発生した場合。

F O R T R A N 7 7 では、 IOS は 8 になる。

F O R T R A N 7 7 E X では、 IOS は 23 になる。

#### 【非互換の検出方法】

J W E 0 0 2 3 I - S メッセージ (助動的割り当てでエラー発生)  
) が output される。

#### 【非互換の回避方法】

I O S T A T 指定子の値の判定値を変更する。

### I O S T A T 指定子（その 2）

#### 【非互換の内容】

O P E N 文の F I L E 指定子に指定されたデータセットの有無  
と、 S T A T U S 指定子に指定された値に矛盾がある場合、  
I O S T A T に返却する値が異なる。

- F O R T R A N 7 7  
8 (助動的割り当てノーチンの復帰値) を返却する。
- F O R T R A N 7 7 E X  
105 (メッセージ番号) を返却する。

#### 【非互換の現れ方】

例：

```
INTEGER IOS
OPEN(1, FILE=' FORT. DATA', IOSTAT=IOS, STATUS=' NEW')
```

データセット FORT. DATA がすでに存在する場合。

F O R T R A N 7 7 では、 IOS は 8 になる。

F O R T R A N 7 7 E X では、 IOS は 105 になる。

#### 【非互換の検出方法】

J W E 0 1 0 5 I - E メッセージ (S T A T U S 指定子の値が  
ファイルの状態と矛盾している) が output される。

#### 【非互換の回避方法】

I O S T A T 指定子の値の判定値を変更する。

## I O S T A T指定子（その3）

## 【非互換の内容】

素引|READ文を実行したとき、ファイル終了記録を検出した場合、I O S T A T指定子に返却される値が異なる。

- F O R T R A N 7 7  
1 5 8 (メッセージ番号) が返却される。
- F O R T R A N 7 7 E X  
-1 (E O D検出) が返却される。

## 【非互換の現れ方】

例：

```

INTEGER*4 IOST, EOD
CHARACTER CH40*40
OPEN(1, ACCESS='KEYED')
EOD=158
READ(1, KEY='KEY1', FMT='(A40)' ) CH40
DO 100 I=1, 99
READ(1, FMT='(A40)', IOSTAT=IOST) CH40
IF(IOST, EQ, EOD) GOTO 10
100 CONTINUE
CLOSE(1)

.
.

10 WRITE(6, *)' EOD ERROR'

F O R T R A N 7 7 では、 I O S T に返却される値は 1 5 8 に
なる。
F O R T R A N 7 7 E X では、 I O S T に返却される値は
-1 になる。

```

## 【非互換の検出方法】

なし。

## 【非互換の回避方法】

判定する値を -1 に変更する。

## I O S T A T指定子（その4）

## 【非互換の内容】

直接処理の素引|READ文、直接処理のREWRI TE文、及び直接処理のDELET E文を実行したとき、指定したキーを持つレコードが存在しない場合、I O S T A T指定子に返却される値が異なる。

- F O R T R A N 7 7  
1 5 8 (メッセージ番号) が返却される。
- F O R T R A N 7 7 E X  
2 4 (メッセージ番号) が返却される。

## 【非互換の現れ方】

例：

```

INTEGER*4 IOST, NOKEY
CHARACTER CH40*40
OPEN(1, ACCESS='KEYED')
NOKEY=158
READ(1, KEY='KEY1', FMT='(A40)', IOSTAT=IOST ) CH40
IF(IOST, EQ, NOKEY) GOTO 10
REWRITE(1, FMT='(A40)', IOSTAT=IOST) CH40
IF(IOST, EQ, NOKEY) GOTO 10
DELETE(1, KEY='KEY2', IOSTAT=IOST)
IF(IOST, EQ, NOKEY) GOTO 10
CLOSE(1)

10 WRITE(6, *)' NOT FOUND KEY ERROR'

F O R T R A N 7 7 では、 I O S T に返却される値は 1 5 8 に
なる。
F O R T R A N 7 7 E X では、 I O S T に返却される値は
2 4 になる。

```

## 【非互換の検出方法】

なし。

## 【非互換の回避方法】

判定する値を 2 4 に変更する。

## I O S T A T 指定子（その 5）

## 【非互換の内容】

素引 W R I T E 文、 R E W R I T E 文を実行したとき、指定したキーを持つレコードが既に存在する場合、 I O S T A T 指定子に返却される値が異なる。

- F O R T R A N 7 7  
1 5 9 (メッセージ番号) が返却される。
- F O R T R A N 7 7 E X  
2 5 (メッセージ番号) が返却される。

## 【非互換の現れ方】

例：

```
INTEGER*4 IOST, DUPKEY
OPEN(1, ACCESS='KEYBD')
DUPKEY=159
WRITE(1, FMT='(A40)', IOSTAT=IOST) 'KEY1 WRITE'
IF(IOST.EQ.DUPKEY) GOTO 10
REWRITE(1, FMT='(A40)', IOSTAT=IOST) 'KEY2 REWRITE'
IF(IOST.EQ.DUPKEY) GOTO 10
CLOSE(1)
```

10     WRITE(6,\*)'DUPLICATE KEY ERROR'

F O R T R A N 7 7 では、 I O S T に返却される値は 1 5 9 になる。  
F O R T R A N 7 7 E X では、 I O S T に返却される値は 2 5 になる。

## 【非互換の検出方法】

なし。

## 【非互換の回避方法】

半固定する値を 2 5 に変更する。

## 現象：メッセージ 出力 (IMSP-10194)

## 【非互換の内容】

リスト指示 R E A D 文における文字型以外の配列に対しても  
文字データの入力を行った場合の動作が異なる場合がある。

- F O R T R A N 7 7  
文字列の残りに空白を設定する。
- F O R T R A N 7 7 E X
  - 文字列の残りの値は変わらない。
  - J W E 0 1 8 2 I -W メッセージ (文字型以外の並びに対する  
定数が文字型である) が出力される。

## 【非互換の現れ方】

例：

INTEGER\*4 I (3)

READ(5,\*) I

, ABCDEFG'

I (1)	A	B	C	D
I (2)	E	F	G	□
I (3)	□	□	□	□

I (1)	A	B	C	D
I (2)	E	F	G	□
I (3)	□	□	□	□

I (2) は □ で示す。

(□ は空白を示す)

7 7 では、 文字列の残りに空白を設定する。  
E X では、 文字列の残りの値は変わらない。

## 【非互換の検出方法】

J W E 0 1 8 2 I -W メッセージが出力された場合、 本非互換による影響を受けている可能性がある。

## 【非互換の回避方法】

- 入力データの長さを配列全体の長さと等しくし、 残りに空白を付加する。  
(例) 'ABCDEGFI□□□' (□ は空白を示す)
- 配列全体を空白で初期化しておく。

現象： 处理結果の相違 (HMSP-10279)

【非互換の内容】

FORTRAN 77 EX の V1.2 で “論理値以外をもつ論理演算” の非互換が改善されますか、その中の一部の項目は、非互換として残ります。

- 論理 IF 文、DO WHILE/UNTIL またはブロック IF 文の以下のいずれかの論理式に、論理値（真、偽）以外の値をもつ場合、論理式の結果が異なることがあります。
- 論理式の中と外に論理演算子 AND がある。
- 論理式の中に論理否定 (NOT) が現れ、その一次子が配列要素であるか、その論理式に関係式がある。

【非互換の一部残留理由】

- 論理型データに論理値以外を代入し、その後、その論理型データを論理値として使用している場合
- 論理演算の実行性能の向上

【非互換の現れ方】

FORTRAN 77 EX では、FORTRAN 77 より、論理式の評価の最適化を強化している。

例：

```

LOGICAL L1/ZF00000000/
LOGICAL L2/Z000F00000/
LOGICAL L3/Z00000000F/
LOGICAL LA(2)/2*Z000F0000/
I=1
IF ((L1 .AND. L2) .AND. L3) GOTO 2
IF (.NOT. LA(2) .AND. I .EQ. 1) GOTO 3
1

```

FORTRAN 77 では、以下のようにいずれも論理式を最終評価後、分岐する。

```

t1=L1 .AND. L2
t2=t1 .AND. L3
IF (t2 .ne. 0) GOTO 2

t3=.NOT. LA(2)
t4=I .EQ. 1
t5=t3 .AND. t4
IF (t5 .ne. 0) GOTO 3

```

FORTRAN 77 EX では、以下のように論理式を最終評価しないで途中で分岐する。

```

IF (L1 .eq. 0) GOTO α
IF (L2 .eq. 0) GOTO α
IF (L3 .ne. 0) GOTO 2
α CONTINUE

IF (I .ne. 1) GOTO β
IF (LA(2) .eq. 0) GOTO 3
β CONTINUE

```

【非互換の検出方法】

なし

【非互換の回避方法】

論理値でない値をもつ論理演算には、推奨演算子に修正する。

## 現象：メッセージ 出力 (IMSP-10213)

## 【非互換の内容】

実行時のエラー識別番号がFORTRAN77とFORTRAN77 EXで異なる。  
このため、エラー識別番号を引数として指定できるエラーツ理系サービスサブルーチンを実行した時、実行結果が異なる場合がある。

## 【非互換の現れ方】

例：  
 INTEGER\*4 ERRNO/200/, MPRINT/-1/  
 CALL ERRSET(ERRNO, 0, MPRINT)  
 CLOSE(10)  
 WRITE(10, \*)

FORTRAN77では、JZL200I-Eメッセージ（補助入出力文の次に許されない入出力文を実行しようとした）の出力が抑止される。  
FORTRAN77 EXでは、JWE0111I-Eメッセージ（一文は一文の直後に実行できない）が出力される。

## 【非互換の検出方法】

JWE0305I-Wメッセージ（エラー識別番号を持つサブルーチンが実行された）が出力される。

## 【非互換の回避方法】

FORTRAN77系のエラー識別番号をFORTRAN77 EX系に修正する。

## 現象：メッセージ 出力 (IMSP-10197)

## 【非互換の内容】

文字型の入出力並びに対し、L, I, F, E, D, Q形編集を行った場合並び作が異なる。

- FORTRAN77 Wレベルの診断メッセージを出力し、編集記述子に従った処理を行う。
- FORTRAN77 EX Eレベルの診断メッセージを出力し、以降の入出力並びの処理を行わない。

## 【非互換の現れ方】

例：  
 CHARACTER\*6 CH  
 CH='ABCDEF'  
 I = 123  
 WRITE(6, 100) CH, I  
 100 FORMAT(I5, I5)  
 END

FORTRAN77では、JZL202I-Wメッセージ（編集記述子と入出力並びの型が矛盾）を出力し、結果として\*\*\*□123が出力される。  
FORTRAN77 EXでは、JWE0151I-Eメッセージ（文字型の入出力並びに対してゆるぎれない編集記述子が指定されている）を出力し、結果として空白のレコードが出力される。  
注) □は空白を示す

## 【非互換の検出方法】

JWE0151I-Eメッセージが出力される。

## 【非互換の回避方法】

- 文字型の入出力並びに対しては、A, G又はZ形編集記述子に修正する。

## 現象：メッセージ 出力 (HNSP-10189)

## 【非互換の内容】

同一装置参照番号に対して、REWIND文又はENDFILE文（マルチファイル）なしで、書式付き入出力文と書式なし入出力文が混在した場合の動作が異なる。

 FORTRAN 77

正常終了する。

 FORTRAN 77 EX

JWE0113I-Eメッセージ（書式付き入出力文は、書式なしとして接続されている装置に実行できない、又は書式なし入出力文は、書式付として接続されている装置に実行できない）を出力し、文を無視する。

## 【非互換の現れ方】

例：  
書式付き、書式なしを混く場合

```
WRITE(1,100)A
WRITE(1)B
100   FORMAT(A)
```

FORTRAN 77では、正常終了。  
FORTRAN 77 EXでは、書式なし入出力文で  
JWE0113I-Eメッセージを出力し、文を無視する。

## 【非互換の検出方法】

JWE0113I-Eメッセージが出力された場合。

## 【非互換の回避方法】

同一の装置参照番号において、書式付き、書式なし入出力文が混在しないよう、入出力文の書式指定を修正する。

## 現象：メッセージ 出力 (HNSP-10212)

## 【非互換の内容】

区分編成ファイルの既存メンバに対し、READ文の後にWRITE文を実行した場合の動作が異なる。

 FORTRAN 77

正常終了する。

 FORTRAN 77 EX

JWE0112I-Eメッセージを出力し、WRITE文を無視する。

## 【非互換の現れ方】

例：  
READ(1)A
 .
 .
 .
 WRITE(1)B

FORTRAN 77では、正常終了する（ただし、以降のメンバが壊れる場合がある）  
FORTRAN 77 EXでは、JWE0112I-Eメッセージを出力し、WRITE文を無視する（ファイル破壊防止）。

## 【非互換の検出方法】

JWE0112I-Eメッセージが出力される。

## 【非互換の回避方法】

WRITE文を削除する。

現象：メッセージ 出力 (MSP-10201)

【非互換の内容】

文字書式中にエラーを検出した場合の動作が異なる。

- F O R T R A N 7 7  
書式仕様中に無効な文字を検出した場合及び書式仕様が左括弧で開始していない場合はエラーとするが、他の書式仕様の誤りを検出した場合は、F O R T R A Nの解釈をし、処理を続行する。
- F O R T R A N 7 7 E X  
書式仕様中に無効な文字を検出した場合及び書式仕様が左括弧で開始していない場合はエラーとし、かつ、次の場合も、エラーメッセージを出力し、以下の処理を行う。
  - 書式の入れ子が8以上の場合、7を超えた入れ子を無視し、処理を続行する。
  - 純記述子の指定に誤りがある場合、書式仕様の最後の右括弧と見なす。
  - 書式仕様の中の反復子に誤りがある場合、書式仕様の最後の右括弧と見なす。
  - 書式仕様が右括弧で終了していない場合、右括弧を補い処理を続行する。

【非互換の現れ方】

例：

```
CHARACTER*6 CH
CH='(I300)'
I = 123
WRITE(6,CH) I
END
```

F O R T R A N 7 7 では、300の値(バイナリ)の下位1バイトを有効にする。

F O R T R A N 7 7 E X では、J W E 0 1 5 5 I — E メッセージ(純記述子の指定に誤りがある)を出力し、書式仕様の最後の右括弧と見なす。

【非互換の検出方法】

```
J W E 0 1 5 4 I — W メッセージ
(書式の入れ子が8以上である)。
J W E 0 1 5 5 I — E メッセージ
(純記述子の指定に誤りがある)。
J W E 0 1 5 6 I — E メッセージ
(書式仕様の中の反復子に誤りがある)。
または。
J W E 0 1 5 8 I — W メッセージ
(書式仕様が右括弧で終了していない)
が出力された場合、本非互換を受けている可能性がある。
```

【非互換の回避方法】

文字書式仕様中の誤った書式を正しい書式に修正する。

## 現象：メッセージ 出力 (HNSP-10199)

## 【非互換の内容】

OPEN文のACTION指定子にWRITEを指定した場合のBACKSPACE文の動作が異なる。

## □ FORTRAN 77

BACKSPACE処理を行う。

## □ FORTRAN 77 EX

JWE0112I-Eメッセージ (BACKSPACE文は、OPEN文のACTION指定子の値がWRITEの装置に対して実行できない) を出力し、文を無視する。

## 【非互換の現れ方】

例：  
OPEN(1, ACTION='WRITE', FORM='UNFORMATTED')  
WRITE(1)A  
BACKSPACE 1

FORTRAN 77では、BACKSPACE処理を行なうか又は入出力エラーとなる。

FORTRAN 77 EXでは、JWE0112I-Eメッセージを出力し、文を無視する。

## 【非互換の検出ノルマ】

JWE0112I-Eメッセージが出力された場合。

## 【非互換の回避方法】

- OPEN文のACTION指定子を BOTH に変更する。

## 現象：メッセージ 出力 (HNSP-10183)

## 【非互換の内容】

浮動小数点アンダーフローエラーのエラーレベルが異なる。

## □ FORTRAN 77

浮動小数点アンダーフローエラーが発生すると、JZL208I-Wメッセージ (浮動小数点の演算結果の絶対値が0でなく  $1.6^{-6}$  より小さい) を出力する。浮動小数点アンダーフローエラーは、エラー打ち切り回数が無制限であるため、何回発生してもプログラムの実行は打ち切られない。

## □ FORTRAN 77 EX

浮動小数点アンダーフローエラーが発生すると、JWE0012I-Eメッセージ (浮動小数点の演算結果の絶対値が  $1.6^{-6}$  より小さい) を出力する。浮動小数点アンダーフローエラーは、エラー打ち切り回数が10回であるため、10回発生した時点でプログラムの実行は打ち切られる。

## 【非互換の現れ方】

例：  
REAL\*4 A/10E-50/, B/10E-50/, C  
DO 10 I=1, 10  
C=A\*B  
10 CONTINUE

FORTRAN 77では、以下の現象となる。  
エラー領域のメッセージ最大印刷回数の標準値が0回であるため、JZL208I-Wメッセージは出力されない。またプログラムの実行は打ち切られない。

FORTRAN 77 EXでは、以下の現象となる。  
JWE0012I-Eメッセージを5回だけ出力し、これ以後はメッセージが出力されず、10回発生するとプログラムの実行が打ち切られる。

## 【非互換の検出方法】

JWE 00121-E メッセージが出力される。

## 【非互換の回避方法】

ERRETサービスサブルーチンを使用してエラー打ち切り回数を無制限にし、メッセージ最大印刷回数を0にする。

現象： 結果 NG (HMS-P-10277)

## 【非互換の内容】

母国語型（日本語型）の引数をもつ組込み関数（LEN, INDEX）の結果が異なる。

- FORTAN 77  
母国語型（日本語型）の引数をもつ組込み関数は、文法で定義されていない（組込み関数はない）と明記）。ただし、翻訳時にエラーが表示されず、占有領域単位の値が返却される。
- FORTAN 77 EX  
母国語型（日本語型）の引数をもつ組込み関数（LEN, INDEX）を、文法で定義した。この結果は、FORTRAN 90 規格（予定）に準拠している。

## 【非互換の現れ方】

例：  
 NCHARACTER\*3 NCH  
 PRINT \*, LEN (NCH)  
 FORTRAN 77 では、6カ所印刷される。  
 FORTRAN 77 EX では、3カ所印刷される。

## 【非互換の検出方法】

FORTAN 77 EX V1.2 では、I レベルの診断メッセージが表示される。

## 【非互換の回避方法】

FORTAN 77 と FORTAN 77 EX の相互の利用は不可。FORTAN 77 EX で翻訳し、文字長単位の結果を考慮したプログラムに修正する。

## 現象：メッセージ 出力 (HMS-P-10214)

## 【非互換の内容】

IBSET, IBCLR, BTEST 関数の第2引数に  
範囲外の値が指定された場合の動作が異なる。

- FORTRAN 77  
何もせず処理を続行する。
- FORTRAN 77 EX  
JWE0257I-Sメッセージ (組込み関数の第2引数の値  
が範囲外) を出力し、処理を続行する。

## 【非互換の現れ方】

例：

IBSET	IBCLR	BTEST
INTEGER*2 I, ANS	INTEGER*4 I, J, ANS	INTEGER I*4, J*2
INTEGER*4 J	I=100	LOGICAL ANS*4
I=10	J=33	I=100
J=16	ANS=IBCLR(I, J)	J=65
ANS=IBSET(I, J)		ANS=BTEST(I, J)

FORTRAN 77 では、何もせず処理を続行する。  
FORTRAN 77 EX では、JWE0257I-Sメッセージを出力し、処理を続行する。

## 【非互換の検出方法】

JWE0257I-Sメッセージが出力される。

## 【非互換の回避方法】

第2引数の値が範囲を超えないように修正する。

## 現象：翻訳時の S レベルエラー (HMS-P-10272)

## 【非互換の内容】

FORTRAN 77 EX の V1.2 で “論理型と算術型の混在”  
の非互換が改善されますが、その中の一部の以下の項目は  
非互換として残ります。  
 — PARAMETER 文の定数式の中に論理型と算術型が  
混在する。  
 — PARAMETER 文の左辺と右辺が算術型と論理型の  
組み合わせである。

【非互換の一時残留理由】  
 — PARAMETER 文は、FORTRAN 77 言語仕様で  
追加された仕様であり、その仕様に FORTRAN 66 以前  
の旧仕様を混在している場合であり、影響は少ない。  
 — 旧仕様からの移行を強く推奨したいため。

## 【非互換の現れ方】

FORTRAN 77 EX では、定数式において算術型と  
論理型を混在した場合、翻訳時に S レベルのエラーが  
出力されます。

例：  
 PARAMETER (I = 1 . TRUE.)  
 PARAMETER (J = . TRUE.)

FORTRAN 77 では、JZK375-W が output され、  
受け入れられます。

FORTRAN 77 EX では、S レベルのエラーが output  
されます。

## 【非互換の検出方法】

翻訳時に S レベルエラーが output されます。

## 【非互換の回避方法】

論理値を算術型で演算しない。論理値を算術型に変更して  
下さい。

現象：結合編集時に組込み関数名が未解決になる

(HMS-P-10270)

## 【非互換の内容】

結合編集時に組込み関数名が未解決になります。以下の二つに分類されます。

- 文法書に記述されている関数名が未解決になる。

例：SIN, ABS, AINTなど

原因：EXTERNAL文に出現しています。

対処：EXTERNAL文をINTRINSIC文に変更してください。

- FORTRAN IV言語仕様以前の関数名が未解決になる。

例：LMAX1, DMAX0, IDBLE, DCABSなど

原因：FORTRAN 77言語仕様の組込み関数ではないので外部関数として解釈されています。

対処：FORTRAN 77言語仕様の組込み関数に変更してください。

## 【非互換の現れ方】

一 静的結合されているコードモジュールに対して。  
FORTRAN 77ライブラリをFORTRAN 77 EX  
ライブラリに入れ換えた場合

一 結合編集時にFORTRAN 77 EXライブラリを結合  
した場合

現象：結合編集エラー

(HMS-P-10269)

## 【非互換の内容】

FORTRAN GE/HEで作成したオブジェクトと、  
FORTRAN 77 EXの実行時ライブラリとの結合。

- FORTRAN 77  
FORTRAN GE/HEで作成したオブジェクトと、  
FORTRAN 77 の実行時ライブラリとの結合ができる。
- FORTRAN 77 EX  
FORTRAN GE/HEで作成したオブジェクトと、  
FORTRAN 77 EXの実行時ライブラリとの結合が  
できない。

## 【非互換の検出方法】

なし。

## 【非互換の回避方法】

FORTRAN 77 EXコンバイラで、再翻訳する。

## 付録B カタログド・プロジェクト一覧

● F O R T E X . . . スカラ翻訳

```
//FORTEX      PROC  RGN=5M, ERGN=1M,
//              A=, B=,
//              SRC=NOS,
//              OPT=B,
//              SYSOUT='*', LCT=0,
//              OBJS='30, 10', SO=NULLFILE, Q='FORT77', DISP=NEW
//-----*
//FORTEX      EXEC PGM=JWD@FORT, REGION=&RGN, COND=(4, LT),
//              PARM='OPT(&OPT), LC(&LCT), &SRC, &A, &B'
//SUBSYS     DD SUBSYS=(VPCS, 'SIZE=(&RGN, &ERGN)')
//SYSUT4     DD DSN=&&WKUT4, DISP=(NEW, DELETE), UNIT=WK10,
//              SPACE=(TRK, (300, 50, 50)),
//              DCB=(LRECL=4096, BLKSIZE=20480, RECFM=FB)
//SYSPRINT   DD SYSOUT=&SYSOUT,
//              DCB=(RECFM=FBA, LRECL=137, BLKSIZE=19043)
//SYSTEM     DD SYSOUT=&SYSOUT,
//              DCB=(RECFM=FBA, LRECL=137, BLKSIZE=19043)
//SYSLIN     DD DSN=&&OBJ, DISP=(&DISP, PASS), UNIT=WK10,
//              SPACE=(TRK, (&OBJS)), DCB=BLKSIZE=3200
//SYSIN      DD DSN=&SO. &Q, DISP=SHR
```

● F O R T E X V P . . . ベクトル翻訳

```
//FORTEXVP    PROC  RGN=18M, ERGN=10M
//              A=, B=,
//              SRC='S, VMSG',
//              OPT=E,
//              SYSOUT='*', LCT=0,
//              OBJS='30, 10', SO=NULLFILE, Q='FORT77', DISP=NEW
//-----*
//FORTEXVP    EXEC PGM=JWD@FORT, COND=(4, LT), REGION=&RGN,
//              PARM='VP, OPT(&OPT), LC(&LCT), &SRC, &A, &B'
//SUBSYS     DD SUBSYS=(VPCS, 'SIZE=(&RGN, &ERGN)')
//SYSPRINT   DD SYSOUT=&SYSOUT,
//              DCB=(RECFM=FBA, LRECL=137, BLKSIZE=19043)
//SYSTEM     DD SYSOUT=&SYSOUT,
//              DCB=(RECFM=FBA, LRECL=137, BLKSIZE=19043)
//SYSUT4     DD DSN=&&WKUT4, DISP=(NEW, DELETE), UNIT=WK10,
//              SPACE=(TRK, (300, 50, 50)),
//              DCB=(LRECL=4096, BLKSIZE=20480, RECFM=FB)
//SYSLIN     DD DSN=&&OBJ, DISP=(&DISP, PASS), UNIT=WK10,
//              SPACE=(TRK, (&OBJS)), DCB=BLKSIZE=3200
//SYSIN      DD DSN=&SO. &Q, DISP=SHR
```

● L K E D E X . . . 結合編集

```
//LKEDEX PROC A=NOMAP, B=LIST, GRLIB=NO, SSLA=JSSL, SSLB=SSL,
//      SSLC=SSL2, SYSOUT='*', WRKS='30,10', MODS='30,10,1',
//      PRVLIB='SYS9. NO', PRVQ='LOAD', GGS='SYS9. GGS', RGN=1024K
//*****
//LINK EXEC PGM=JQAL, REGION=&RGN, COND=(4, LT),
//      PARM='&A, &B, LET'
//SUBSYS   DD SUBSYS=(VPCS, 'SIZE=(&RGN, 00M)')
//SYSLIB    DD DSN=&PRVLIB. &PRVQ, DISP=SHR
//          DD DSN=&GGS. LOAD, DISP=SHR
//          DD DSN=SYS9. &GRLIB. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLA. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLB. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLC. LOAD, DISP=SHR
//          DD DSN=SYS2. FORTLIB, DISP=SHR
//SYSPRINT  DD SYSOUT=&SYSOUT, DCB=(BLKSIZE=4840)
//SYSTEM    DD SYSOUT=&SYSOUT
//SYSUT1    DD UNIT=V10, SPACE=(TRK, (&WRKS))
//SYSLMOD   DD DSN=&LM, UNIT=WK10, DISP=(NEW, PASS, DELETE),
//              SPACE=(TRK, (&MODS))
//SYSLIN    DD DSN=&&OBJ, DISP=(OLD, DELETE)
//          DD DDNAME=SYSIN
```

● L K E D C T E X . . . 結合編集とロードモジュールの保存

```
//LKEDEX PROC A=NOMAP, B=LIST, GRLIB=NO, SSLA=JSSL, SSLB=SSL,
//      SSLC=SSL2, SYSOUT='*', WRKS='30,10', MODS='30,10,1',
//      PRVLIB='SYS9. NO', PRVQ='LOAD', GGS='SYS9. GGS', RGN=1024K,
//      LM=, Q='LOAD', UNIT=, DISP='NEW, CATLG, DELETE'
//*****
//LINK EXEC PGM=JQAL, REGION=&RGN, COND=(4, LT),
//      PARM='&A, &B, LET'
//SUBSYS   DD SUBSYS=(VPCS, 'SIZE=(&RGN, 00M)')
//SYSLIB    DD DSN=&PRVLIB. &PRVQ, DISP=SHR
//          DD DSN=&GGS. LOAD, DISP=SHR
//          DD DSN=SYS9. &GRLIB. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLA. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLB. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLC. LOAD, DISP=SHR
//          DD DSN=SYS2. FORTLIB, DISP=SHR
//SYSPRINT  DD SYSOUT=&SYSOUT, DCB=(BLKSIZE=4840)
//SYSTEM    DD SYSOUT=&SYSOUT
//SYSUT1    DD UNIT=V10, SPACE=(TRK, (&WRKS))
//SYSLMOD   DD DSN=&LM, &Q, UNIT=&UNIT, DISP=(&DISP),
//              SPACE=(TRK, (&MODS), RLSE)
//SYSLIN    DD DSN=&&OBJ, DISP=(OLD, DELETE)
//          DD DDNAME=SYSIN
```

● L K E D U P E X . . . 結合編集とロードモジュールの更新

```

//LKEDEX PROC A=NOMAP, B=LST, GRLIB=NO, SSLA=JSSL, SSLB=SSL,
//      SSLC=SSL2, SYSOUT='*', WRKS='30,10',
//      PRVLIB='SYS9. NO', PRVQ='LOAD', GGS='SYS9. GGS', RGN=1024K,
//      LM=, Q='LOAD',
//      UPS=30, CNTL=NORMAL, USER=OFF, CSPC='30,40,10', CUNIT=TDS
//*****+
//LINK EXEC PGM=JQAL, REGION=&RGN, COND=(4, LT),
//      PARM='&A, &B, LET'
//SUBSYS DD SUBSYS=(VPCS, 'SIZE=(&RGN, 00M)')
//SYSLIB DD DSN=&PRVLIB. &PRVQ, DISP=SHR
//          DD DSN=&GGS.. LOAD, DISP=SHR
//          DD DSN=SYS9. &GRLIB.. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLA.. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLB.. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLC.. LOAD, DISP=SHR
//          DD DSN=SYS2. FORTLIB, DISP=SHR
//SYSPRINT DD SYSOUT=&SYSOUT, DCB=(BLKSIZE=4840)
//SYSTERM DD SYSOUT=&SYSOUT
//SYSUT1 DD UNIT=VIO, SPACE=(TRK, (&WRKS)), DISP=(NEW, DELETE)
//SYSLMOD DD DSN=&LM. &Q, DISP=OLD, SPACE=(TRK, (0, &UPS), RLSE)
//SYSLIN DD DSN=&&OBJ, DISP=(OLD, DELETE)
//          DD DSN=SYS9. LINKC. CNTL(&CNTL), DISP=SHR
//          DD DDNAME=SYSIN
//OLDLIM DD DSN=&LM. &Q, DISP=OLD
//*
//%CFCHECK IF USER=OFF, CONDENS2
//CONDENS EXEC PGM=CALLUTY, COND=(4, LT), REGION=&RGN,
//      PARM=' JSECOPY/ COPY INDD=1, OUTDD=T/ COPY INDD=1, OUTDD=1/'
//SUBSYS DD SUBSYS=(VPCS, 'SIZE=(&RGN, 00M)')
//STEPLIB DD DSN=SYS9. GENUTY. LOAD, DISP=SHR
//I DD DSN=&LM. &Q, DISP=OLD
//T DD DSN=&LM. &Q.. C, DISP=(NEW, DELETE, CATLG), UNIT=&CUNIT,
//      SPACE=(TRK, (&CSPC))
//SYSPRINT DD SYSOUT=&SYSOUT
//SYSUT3 DD UNIT=VIO, SPACE=(TRK, (&CSPC))
//SYSUT4 DD UNIT=VIO, SPACE=(TRK, (&CSPC)), DCB=(KEYLEN=8)
//SYSIN DD DSN=&&WORK, UNIT=VIO, SPACE=(TRK, 1)
//FT06F001 DD SYSOUT=&SYSOUT
//PEXIT
//*
//CONDENS2 EXEC PGM=CALLUTY, COND=(4, LT), REGION=&RGN
//      PARM=' JSECOPY/ COPY INDD=1, OUTDD=T/ COPY INDD=1, OUTDD=1/'
//SUBSYS DD SUBSYS=(VPCS, 'SIZE=(&RGN, 00M)')
//STEPLIB DD DSN=SYS9. GENUTY. LOAD, DISP=SHR
//I DD DSN=&LM. &Q, DISP=OLD
//T DD DSN=&USER. . &LM. &Q.. C, DISP=(NEW, DELETE, CATLG),
//      SPACE=(TRK, (&CSPC)), UNIT=&CUNIT
//SYSPRINT DD SYSOUT=&SYSOUT
//SYSUT3 DD UNIT=VIO, SPACE=(TRK, (&CSPC))
//SYSUT4 DD UNIT=VIO, SPACE=(TRK, (&CSPC)), DCB=(KEYLEN=B)
//SYSIN DD DSN=&&WORK, UNIT=VIO, SPACE=(TRK, 1)
//FT06F001 DD SYSOUT=&SYSOUT

```

● L K E D I T E X . . . 結合編集と一時ロードモジュールの作成

```
//LKEDITEX PROC A=NOMAP, B=LIST, GRLIB=NO, SSLA=JSSL, SSLB=SSL,
//      SSLC=SSL2, SYSOUT='*', WRKS='30,10', MODS='30,10,1',
//      PRVLIB='SYS9.N0', PRVQ='LOAD', GGS='SYS9.GGS', RGN=512K,
//      CNTL=NORMAL, LM=, Q='LOAD'
//*********************************************************************
//LINK EXEC PGM=JQAL, REGION=&RGN, COND=(4, LT),
//      PARM='&A, &B, LET'
//SUBSYS DD SUBSYS=(VPCS, 'SIZE=(&RGN, 00M)')
//SYSLIB DD DSN=&PRVLIB. &PRVQ, DISP=SHR
//          DD DSN=&GGS.. LOAD, DISP=SHR
//          DD DSN=SYS9. &GRLIB.. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLA.. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLB.. LOAD, DISP=SHR
//          DD DSN=SYS9. &SSLC.. LOAD, DISP=SHR
//          DD DSN=SYS2.FORTLIB, DISP=SHR
//SYSPRINT DD SYSOUT=&SYSOUT, DCB=(BLKSIZE=4840)
//SYSTEM DD SYSOUT=&SYSOUT
//SYSUT1 DD UNIT=VIO, SPACE=(TRK, (&WRKS))
//SYSLMOD DD DSN=&&LM. &Q, UNIT=WK10, DISP=(NEW, PASS, DELETE),
//           SPACE=(TRK, (&MODS))
//SYSLIN DD DSN=&&OBJ, DISP=(OLD, DELETE)
//          DD DSN=SYS9.LINKC.CNTL(&CNTL), DISP=SHR
//          DD DDNAME=SYSIN
//OLDDLM DD DSN=&LM. &Q, DISP=SHR
```

● GOEX . . . 実行

```
//GOEX    PROC PNM=TEMPNAME,
//          A='ERRCUT=0',
//          SYSOUT='*', GOSYSIN='DDNAME=SYSIN',
//          ORECFM=FBA, OBSIZE=19043, ORSIZE=137,
//          RGN=8M
//*-*
//RUN   EXEC PGM=&PNM, COND=(4, LT), PARM='FLIB(&A)', REGION=&RGN
//SUBSYS      DD SUBSYS=(VPCS, 'SIZE=(00000K, 00M)')
//STEPLIB      DD DSN=&&LM, DISP=(OLD, DELETE)
//SYSPRINT     DD SYSOUT=&SYSOUT,
//              DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
//FT05F001     DD &GOSYSIN
//FT06F001     DD SYSOUT=&SYSOUT,
//              DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
```

● LMGOEX . . . 作成済ロードモジュールの実行

```
//LMGOEX   PROC PNM=TEMPNAME,
//          A='ERRCUT=0',
//          SYSOUT='*', GOSYSIN='DDNAME=SYSIN',
//          ORECFM=FBA, OBSIZE=19043, ORSIZE=137,
//          RGN=8M,
//          LM=, Q='LOAD'
//*-*
//RUN   EXEC PGM=&PNM, COND=(4, LT), PARM='FLIB(&A)', REGION=&RGN
//SUBSYS      DD SUBSYS=(VPCS, 'SIZE=(00000K, 00M)')
//STEPLIB      DD DSN=&LM, &Q, DISP=SHR
//SYSPRINT     DD SYSOUT=&SYSOUT,
//              DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
//FT05F001     DD &GOSYSIN
//FT06F001     DD SYSOUT=&SYSOUT,
//              DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
```

## ● SAMPLER . . . サンプリング解析

```
//SAMPLER      PROC A='ERRCUT=0', RGN=8M,
//                  LM=, Q='LOAD', SYSOUT='*',
//                  PNM=TEMPNAME, INV=2000,
//                  SMPOUT=, SMPQ='SMPOUT', UNIT=TSSWK,
//                  ORECFM=VBA, OBSIZE=12504, ORSIZE=125,
//                  SYSOUT='*', GOSYSIN='DDNAME=SYSIN'
//-----*
//ANA EXEC PGM=JPLMAIN, COND=(4, LT), REGION=&RGN,
//                  PARM='PGMNAME(&PNM), INTERVAL(&INV), TPARM(&A)'
//SUBSYS DD SUBSYS=(VPCS, 'SIZE=(00000K, 00M)')
//JPLL1B DD DSN=&LM, &Q, DISP=SHR
//JPLPRINT DD DSN=&SMPOUT, &SMPQ, DISP=(NEW, CATLG), UNIT=&UNIT,
//                  SPACE=(TRK, (10, 10), RLSE),
//                  DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
//FT05F001 DD &GOSYSIN
//FT06F001 DD SYSOUT=&SYSOUT,
//                  DCB=(RECFM=FBA, LRECL=137, BLKSIZE=19043)
```

● ANALYZER . . . 実行詳細解析

```

//ANALYZER PROC RGN=8M,
//      SO=NULLFILE, Q=' .FORT77' ,
//      AOP=' LL, COUNT' ,
//      COP=' OPT(E)' ,
//      GRLIB=NO, SSLA=JSSL, SSLB=SSL, SSLC=SSL2, GGS=' SYS9. GGS' ,
//      PRVLIB=' SYS9. NO' , PRVQ=' .LOAD' ,
//      ORECFM=VBA, ORSIZE=255, OBSIZE=2554,
//      ANAOUT=, ANAQ=' .ANAOUT' , UNIT=TSSWK,
//      GOSYSIN=' DDNAME=GOSYSIN' , SYSOUT=' *'
//-----*
//ANA    EXEC PGM=JPJ@ANA, REGION=&RGN,
//      PARM=' &AOP/&COP'
//SUBSYS  DD SUBSYS=(VPCS, ' SIZE=(00000K, 00M)' )
//SYSLIB  DD DSN=PP1.LEBASE, ANALIB, DISP=SHR
//      DD DSN=&PRVLIB. &PRVQ, DISP=SHR
//      DD DSN=&GGS. .LOAD, DISP=SHR
//      DD DSN=S9. &GRLIB. .LOAD, DISP=SHR
//      DD DSN=S9. &SSLA. .LOAD, DISP=SHR
//      DD DSN=S9. &SSLB. .LOAD, DISP=SHR
//      DD DSN=S9. &SSLC. .LOAD, DISP=SHR
//      DD DSN=S2. FORTLIB, DISP=SHR
//SYSIN   DD DSN=&SO. &Q, DISP=SHR, LABEL=(,, IN)
//SYSPRINT DD DSN=&ANAOUT. &ANAQ, DISP=(NEW, CATLG), UNIT=&UNIT,
//      SPACE=(TRK,(10,10),RLSE),
//      DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
//SYSPRINT DD SYSOUT=*
//FT05F001 DD &GOSYSIN
//FT06F001 DD SYSOUT=&SYSOUT,
//      DCB=(RECFM=FBA, LRECL=137, BLKSIZE=19043)

```

● ANALDIN . . . 一部のプログラム単位の実行詳細解析

```

//ANALDIN PROC RGN=8M,
//      SO=NULLFILE, Q=' . FORT77' ,
//      AOP=' LL, COUNT' ,
//      COP=' OPT(E)' ,
//      GRLIB=NO, SSLA=JSSL, SSLB=SSL, SSLC=SSL2, GGS=' SYS9. GGS' ,
//      PRVLIB=' SYS9. NO' , PRVQ=' . LOAD' ,
//      ANAOUT=, ANAQ=' . ANAOUT' , UNIT=TSSWK,
//      ORECFM=VBA, OBSIZE=2554, ORSIZE=255,
//      LM=NULLFILE, LOADQ=' . LOAD' , PNM=' TEMPNAME' ,
//      GOSYSIN=' DDNAME=GOSYSIN' , SYSOUT=' *'
//-----*
//ANA    EXEC PGM=JPJ@ANA, REGION=&RGN,
//      PARM=' LOADIN, &AOP/&COP'
//SUBSYS DD SUBSYS=(VPCS, ' SIZE=(00000K, 00M)' )
//LMDLIB DD DSN=&LM. &LOADQ. (&PNM), DISP=SHR
//SYSLIB  DD DSN=PP1.LEBASE, ANALIB, DISP=SHR
//        DD DSN=&PRVLIB. &PRVQ, DISP=SHR
//        DD DSN=&GGS. . LOAD, DISP=SHR
//        DD DSN=S9. &GRLIB. . LOAD, DISP=SHR
//        DD DSN=S9. &SSLA. . LOAD, DISP=SHR
//        DD DSN=S9. &SSLB. . LOAD, DISP=SHR
//        DD DSN=S9. &SSLC. . LOAD, DISP=SHR
//        DD DSN=S2. FORTLIB, DISP=SHR
//SYSIN   DD DSN=&SO. &Q, DISP=SHR, LABEL=(,, IN)
//SYSPRINT DD DSN=&ANAOUT. &ANAQ, DISP=(NEW, CATLG), UNIT=&UNIT,
//           SPACE=(TRK, (10, 10), RLSE),
//           DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
//SYSPRINT1 DD SYSOUT=*
//FT05F001 DD &GOSYSIN
//FT06F001 DD SYSOUT=&SYSOUT,
//           DCB=(RECFM=FBA, LRECL=137, BLKSIZE=19043)

```

## ● TOP10EX . . . TOP10EX

```

//TOP10EX PROC PNM=TOP10EX, LM=' J0000. TOOL. LOAD',
//           OB=19043, ORECFM=FBA, OUT6=' SYSOUT=*',
//           DSN=, INC=NULLFILE, UNIT=WK10,
//           SPC8=' 100, 50, 50', SPCA=' 200, 100, 50', SPCB=' 200, 50',
//           SPCS=' 200, 10', SPCV=' 100, 10, 50', SPCW=' 5, 1'
//*-
//TOP10EX EXEC PGM=&PNM          * PROGRAM NAME OF "TOP10"
//SUBSYS   DD SUBSYS=(VPCS, 'SIZE=(00000K, 00M)')
//STEPLIB  DD DSN=&LM, DISP=SHR    * LOAD MODULE NAME OF "TOP10"
//          DD DSN=PP1.FORT77EX.LINKLIB, DISP=SHR
//*-
//FT01F001 DD DSN=&DSN,          * "ANALYZER OUTLIST"
//          DISP=SHR,             * OR "FORTRAN SOURCE PROGRAM"
//          LABEL=(,,,IN)
//*-
//SYSINC   DD DSN=&INC,          * FORTRAN SOURCE PROGRAM
//          DISP=SHR, LABEL=(,,,IN) * TO BE INCLUDED
//*-
//SYSPRINT DD DSN=&S, DISP=NEW,   * ERROR MESSAGE FILE OF VP COMPILER
//          UNIT=&UNIT,
//          SPACE=(TRK, (&SPCS)),
//          DCB=(RECFM=VB, LRECL=125, BLKSIZE=12504)
//*-
//SYSTERM  DD &OUT6,
//          DCB=(RECFM=FBA, LRECL=125, BLKSIZE=19043)
//*-
//VPMMSG   DD DSN=&&V, DISP=NEW,   * VECTORIZATION MESSAGE FILE
//          UNIT=&UNIT,            * (( PO ))
//          SPACE=(TRK, (&SPCV)),
//          DCB=(RECFM=VB, LRECL=125, BLKSIZE=3120)
//*-
//WORK80   DD UNIT=&UNIT, DISP=NEW, * WORK SPACE (( PO ))
//          SPACE=(TRK, (&SPC8)),
//          DCB=(BLKSIZE=6000, LRECL=80, RECFM=FB)
//*-
//FT10F001 DD UNIT=WK10, DISP=NEW,
//          SPACE=(TRK, (10, 10))
//*-
//FT99F001 DD DUMMY             * DUMMY DATA SET
//*-
//TMP1     DD DSN=&&TMP1, DISP=(NEW, DELETE), UNIT=WK10,
//          SPACE=(TRK, (10, 10, 10)),
//          DCB=(RECFM=VBA, LRECL=255, BLKSIZE=2550)
//TMP2     DD DSN=&&TMP2, DISP=(NEW, DELETE), UNIT=WK10,
//          SPACE=(TRK, (10, 10, 10)),
//          DCB=(RECFM=VBA, LRECL=255, BLKSIZE=2550)
//*-
//FT05F001 DD DDNAME=SYSIN
//*-
//FT06F001 DD &OUT6,             * "TOP10EX" PRINT FILE
//          DCB=(LRECL=137, BLKSIZE=&OB, RECFM=&ORECFM)
//*-

```

## ● M E R G E R . . . M E R G E R

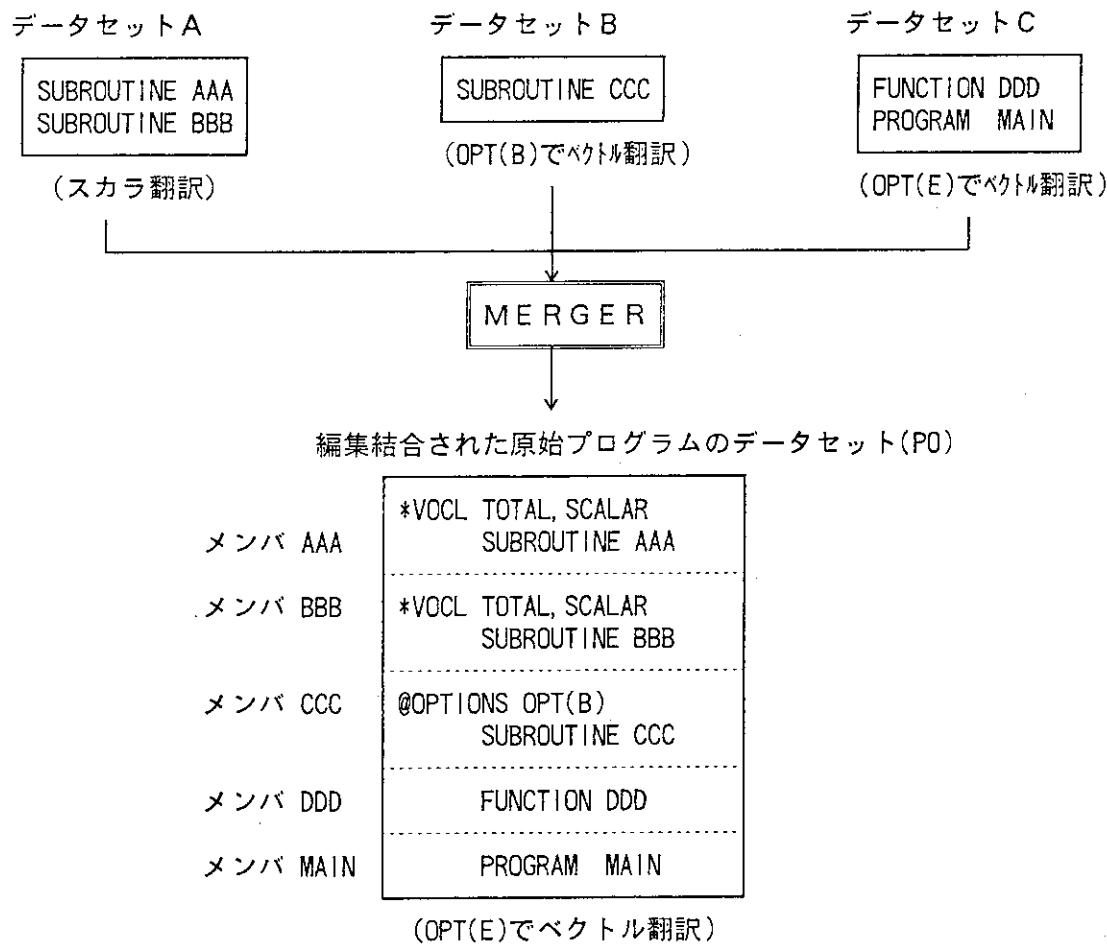
```
//MERGER PROC PNM=MERGER, LM='J0000. TOOL. LOAD',
//           SYSOUT='*', GOSYSIN='DDNAME=SYSIN',
//           ORECFM=FBA, OBSIZE=19043, ORSIZE=137,
//           RGN=1M
//*-----*
//MERGE EXEC PGM=&PNM, COND=(4, LT), REGION=&RGN
//SUBSYS      DD SUBSYS=(VPCS, 'SIZE=(00000K, 00M)')
//STEPLIB     DD DSN=&LM, DISP=SHR
//             DD DSN=PP1.FORT77EX.LINKLIB, DISP=SHR
//SYSPRINT    DD SYSOUT=&SYSOUT,
//             DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
//FT05F001   DD &GOSYSIN
//FT06F001   DD SYSOUT=&SYSOUT,
//             DCB=(RECFM=&ORECFM, LRECL=&ORSIZE, BLKSIZE=&OBSIZE)
```

## 付録C MERGER使用手引き

## MERGER使用手引き

原子力コードの中には、翻訳条件等の違いにより複数のデータセットに原始プログラムを分けて保管しているもののが多々あります。MERGERは、このような複数のデータセットに分割されているFORTRAN原始プログラムを1つの区分編成データセットに結合編集するツールです。また、MERGERでは原始プログラム中に、VOCAL文、@OPTIONS行を任意に挿入することが出来ます。これらの機能により、プログラム単位毎に翻訳の条件が異なるような場合でも、一度のステップで翻訳することを可能にします。

MERGERは、分割翻訳が許されないFORTUNEやANALYZERの入力用原始プログラムの作成に役立てることが出来ます。



図C.1 MERGERによる原始プログラムの結合編集

## 1. 結合編集の方法

M E R G E R は区分編成データセット、或いは順編成データセットを入力とし、副プログラムと同じ名前のメンバを持つ区分編成データセットに編集出力します。

### ○ 主プログラムの扱い

プログラム名に係わらず M A I N というメンバ名になります。

### ○ 名前無しブロックデータの扱い

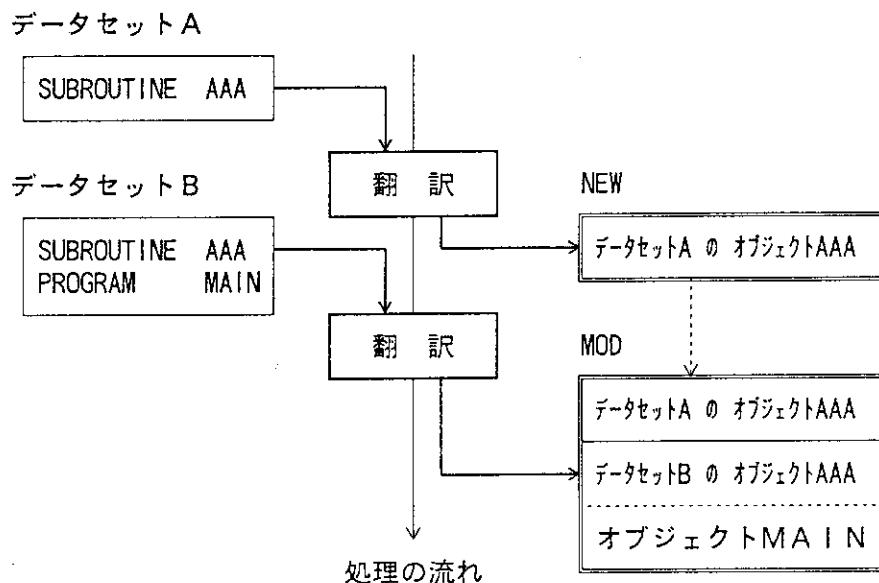
B L K \* \* \* \* というメンバ名が付けられます (\* \* \* \* には、0 0 0 0 から 9 9 9 9 までの数字が順番に付けられます)。

### ○ 同一名のプログラム単位が複数存在する場合の扱い

同一名のプログラム単位が複数存在する場合、2度目以降に見つかったプログラム単位は出力されません。 M E R G E R では、プログラム単位名と出力先データセットのメンバ名を照合しているので、同一名のメンバが出力先に存在している場合は出力をいません。 従って、翻訳処理と同じ順序で編集出力を行えば、複数のデータセットに分割された原始プログラムと等価な一つの原始プログラムデータセットを作成することが出来ます。

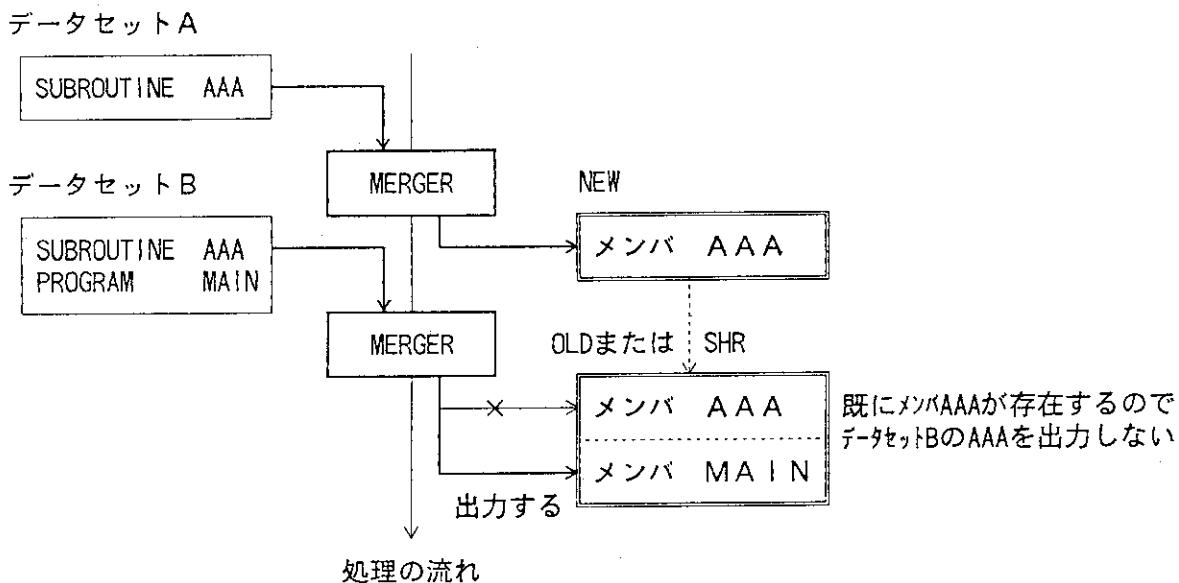
(FORTRAN コンパイラによって作成されたオブジェクトは全て一つのロードモジュールに結合されますが、同じ名前のプログラム単位が存在する場合、先に結合されたオブジェクトが有効になります。)

図C.2 と図C.3 に同一名のプログラム単位が複数存在する場合の翻訳処理流れと M E R G E R による結合編集の処理流れを示します。



このように作成されたオブジェクトではデータセットAのプログラム単位 A A A が有効となる。

図C.2 同一名のプログラム単位が複数存在する場合の翻訳処理流れ



図C.3 同一名のプログラム単位が複数存在する場合のMERGERの処理流れ

## 2. MERGERのオプション

MERGERのオプションは装置機番5に指定されたデータセット中に記述します。

### (1) ELMオプション

入力するデータセットが区分編成の場合、メンバ名を指定します。メンバ名は翻訳時のELMオプションと同じメンバを指定します。ELMオプションを省略すると全メンバが指定されたとみなします。

ELM (*)	全メンバを入力
ELM (memb [, memb] . . . )	メンバの選択入力

memb : メンバ名 (20個まで指定できる)

### (2) SCLオプション

スカラ翻訳するプログラム単位名を指定します (\*VOCL TOTAL, SCALAR文を挿入する)。ELMオプションが併せて指定されている場合は、ELMオプションで指定されたメンバの中にSCLオプションで指定するプログラム単位が含まれている場合に有効になります。主プログラム名が付けられている場合は、主プログラム名も指定できます。

SCL (*)	全プログラム単位が対象
SCL (rout [, rout] . . . )	対象プログラム単位の選択

rout : プログラム単位名、或いは主プログラム名 (20個まで指定できる)

## (3) OPTオプション

指定されたプログラム単位に対して、@OPTIONS行を挿入します。OPTオプションを指定したときは必ず、挿入する@OPTIONS行を入力オプションデータセット中に併せて記述しなければなりません。ELMオプションが併せて指定されている場合は、ELMオプションで指定されたメンバの中にOPTオプションで指定するプログラム単位が含まれている場合に有効になる。主プログラム名が付けられている場合は、主プログラム名も指定できます。

OPT(\*)

OPT(rout[, rout]...)

全プログラム単位が対象

対象プログラム単位の選択

rout : プログラム単位名、或いは主プログラム名（20個まで指定できる）

3. MERGERの使用3.1 カタログド・プロジェクト

MERGERを使用するためのカタログMERGERが用意されています。MERGERの使用にあたって、以下のDD名で必要なデータセットをJCLに定義して下さい。

DD名	内 容	DSORG	備 考
INPUT	入力原始プログラム	PS/P0	
OUTPUT	編集された原始プログラム	P0	新たに作成する場合は、DCBを指定すること。 LRECL=80
SYSIN	オプションデータ	PS	省略可, LRECL=80

※ オプションデータは省略可能（このとき、オプションELM(\*)のみが有効となる）

### 3.2 使用例

入力原始プログラムが複数のデータセットに存在する場合は、その数の分だけMERGERによる結合編集のステップを設けます。結合編集するデータセットの順序とMERGERのオプションの指定方法は翻訳時のJCLを参考にします。MERGERによる結合編集と結合編集されたデータセットを翻訳する例を以下に示します。

#### ●例1 スカラ翻訳とベクトル翻訳の混在

翻訳

```
//COMPILE EXEC FORTEX, SO='J****.INPUT.FORT77', DISP=NEW,
//          A='ELM(MEM1, MEM2)'
/*
//COMPILE EXEC FORTEXVP, SO='J****.INPUT.FORT77', DISP=MOD,
//          A='ELM(*)'
```

⇒メンバMEM1, MEM2  
をスカラ翻訳

⇒残りのメンバを  
ベクトル翻訳

MERGER

```
//MERGE EXEC MERGER
//SYSIN DD *
ELM(MEM1, MEM2)
SCL(*)
/*
//INPUT DD DSN=J****.INPUT.FORT77, DISP=SHR
//OUTPUT DD DSN=J****.MERGED.FORT77, DISP=(NEW, CATLG),
//          UNIT=TSSWK, SPACE=(TRK,(10,10,10)),
//          DCB=(RECFM=FB, LRECL=80, BLKSIZE=3200)
/*
//MERGE EXEC MERGER
//INPUT DD DSN=J****.INPUT.FORT77, DISP=SHR
//OUTPUT DD DSN=J****.MERGED.FORT77, DISP=SHR
/*
//COMPILE EXEC FORTEXVP, SO='J****.MERGED.FORT77',
//          A='ELM(*)'
```

⇒メンバMEM1, MEM2  
に含まれるプログラム  
単位に対し、  
スカラ翻訳指定

⇒出力されるデータ  
セットを作成

⇒残りプログラム単位  
を結合

⇒ベクトル翻訳

#### ●例2 翻訳オプションの指定

翻訳

```
//COMPILE EXEC FORTEXVP, SO='J****.INPUT.FORT77', DISP=NEW,
//          A='ELM(MEM1, MEM2)', OPT=B
/*
//COMPILE EXEC FORTEXVP, SO='J****.INPUT.FORT77', DISP=MOD,
//          A='ELM(*)', OPT=E
```

⇒メンバMEM1, MEM2  
をOPT(B)で翻訳

⇒残りのメンバを  
をOPT(E)で翻訳

```

MERGER
//MERGE EXEC MERGER
//SYSIN DD *
  ELM(MEM1, MEM2)
  OPT(*)
  @OPTIONS OPT(B)
  /*
  //INPUT DD DSN=J****, INPUT, FORT77, DISP=SHR
  //OUTPUT DD DSN=J****, MERGED, FORT77, DISP=(NEW, CATLG),
  //          UNIT=TSSWK, SPACE=(TRK,(10,10,10)),
  //          DCB=(RECFM=FB, LRECL=80, BLKSIZE=3200)
  */
  /*
  //MERGE EXEC MERGER
  //INPUT DD DSN=J****, INPUT, FORT77, DISP=SHR
  //OUTPUT DD DSN=J****, MERGED, FORT77, DISP=SHR
  */
  /*
  //COMPILE EXEC FORTEXVP, SO= J****, MERGED, FORT77,
  //          A='ELM(*)', OPT=E

```

↔メンバMEM1, MEM2  
に含まれるプログラム  
単位に対し、  
OPT(B)を指定

↔出力されるデータ  
セットを作成

↔残りのプログラム  
単位を結合

↔OPT(E)で翻訳

#### 4. MERGERの使用上の注意

(注1) OPT(\*), SCL(\*) オプションが指定された場合は、@OPTIONS行、\*VOCAL行は各プログラム単位の先頭に挿入されます。しかし、OPT、SCLオプションでプログラム単位名を個々に指定した場合、@OPTIONS行、\*VOCAL行はプログラム単位を宣言している文の直前に挿入されます。@OPTIONS行は各プログラム単位中において、コメント行を含む全ての文よりも先に宣言されなければなりません。従って、OPTオプションで個々のプログラム単位を指定した場合、指定されたプログラム単位において、プログラム単位の宣言文の前にコメント行がある場合は翻訳時のエラーとなるので注意して下さい。

(注2) OUTPUTで指定されるデータセットはメンバが存在しないか、或いは次の条件を満たしている必要があります。

- ・ プログラム単位名とメンバ名が1対1に対応している。
- ・ 主プログラムが格納されているメンバ名がMAINである。

MERGERによって作成されたデータセットであれば問題はありません。

(注3) FORTRAN 原始プログラムにおいて、プログラム単位を宣言している文が以下の条件を満たしていることが必要です。

- ・ 宣言文とプログラム単位名が同一行にあること。
- ・ プログラム単位名が8文字以下であること。

## 5. MERGERのメッセージ出力

MERGERはメッセージを装置機番 6 に指定されたデータセットに出力しています。 出力内容は以下の通りです。

### (1) MERGERのヘッダ

### (2) 有効オプション

ELM OPTION : *****	... ELMオプションのエコーバック
SCL OPTION : *****	... SCLオプションのエコーバック
OPT OPTION : *****	... OPTオプションのエコーバック
OPT STATE : *****	... 指定@OPTIONS文のエコーバック
INVALID OPTION : *****	... オプションの指定に誤りがある
NO ELM OPTION => ELM(*)	... ELMオプションが指定されていないので、 ELM(*) を有効にする。

### (3) 各種メッセージ

- \*\*\*\*\* IS ALREADY EXIST, THEN NOT COPIED.  
... 出力データセットに既に同名のメンバ名が存在するので、新たにメンバを作成しない。
- MAIN ROUTINE IS ALREADY EXIST, THEN NOT COPIED.  
... 出力データセットに既にメンバMAINが存在するので、新たにメンバを作成しない。
- NON LABEL BLOCK DATA => \*\*\*\*\*  
... 名前無しブロックデータを\*\*\*\*\*というメンバに格納する。
- NON LABEL ROUTINE => MAIN  
... 名前無しプログラムをMAINというメンバに格納する。
- PROGRAM \*\*\*\*\* => MAIN  
... 主プログラム\*\*\*\*\*をMAINというメンバに格納する。
- MEMBER : \*\*\*\*\* IS SELECTED.  
... 入力データセットからメンバ\*\*\*\*\*を選択した。  
(ELM(\*) が指定されているときはこのメッセージは出力しない。)
- @OPTIONS STATEMENT => \*\*\*\*\*  
... プログラム単位\*\*\*\*\*に@OPTIONS行を挿入した。  
(OPT(\*) が指定されているときはこのメッセージは出力しない。)
- SCALAR COMPILE => \*\*\*\*\*  
... プログラム単位\*\*\*\*\*に\*VOCL TOTAL, SCALAR文を挿入した。  
(SCL(\*) が指定されているときはこのメッセージは出力しない。)
- NUMBER OF MEMBERS SPECIFIED IS GREATER THAN 20.  
... ELM, OPT, SCL オプションで指定できるメンバ名、またはプログラム単位名が20個を超えている。