

会話実行型 FORTRAN (CFO) の概要と評価

福島 脩、田中 清、奥瀬尚佐、松本匡通

(日本電信電話公社横須賀電気通信研究所)

1. はじめに

TSS 用 OS のもとで、FORTRAN 等で書かれたプログラムの作成、実行を会話的に行う方法としてエディタと運動する構文チェックを使用し、端末より FORTRAN 文を 1 文ずつ入力し構文チェックしながらプログラムを作成する方法がある。しかしながら、構文チェックによるプログラムの作成は、出来上がった原プログラムを、コンパイラによるコンパイル処理、及びリンケージ・エディタによるリンク処理の後、はじめて実行可能となる。またこの処理は、プログラムの誤りを発見することに行われる。これらの処理は、利用者にとって繁雑な処理であるし、またこの処理のために無駄な計算時間を使用しなければならない。

筆者等は、主に一般プログラムを対象とした、デバッグを含めたプログラム作成時の負担を軽減することを目的により操作性の良い会話型言語システムとして会話実行型 FORTRAN (以下 CFO と略す) を開発することとした。

2. 会話型言語プロセッサの具備すべき機能

TSS 用 OS の発達により、遠隔地より端末を介して計算機システムと直接対話できる機能が提供されるにおよび会話型言語プロセッサの本格的な開発が始まった。会話型言語として代表的なものには、BASIC, JOSS, 会話 APL などがあり、これらの言語は、言語仕様上からも会話型システムに適合し易いよう検討されている。これらの流れに対して従来からある代表的な言語である FORTRAN, COBOL などを会話型言語としてインプリメントし

た例がある。特に FORTRAN に関しては、言語仕様の簡明さ及び普及度の高いこともあって種々の会話型言語プロセッサが開発されている。この例として IBM 社の QUIKTRAN, UNIVAC 社の CFOR などがある。

筆者等は、CFO の開発に先立って会話型言語プロセッサの具備すべき機能条件を以下のように考えた。

- ① 端末より入力された原プログラムが 1 文ごとに独立である。したがって他の文の入力により原文の構文上、意味上には何ら影響を与えられないことはなく、またどの時点でも原プログラムに対するあらゆる操作(修正、実行など)が可能でなければならない。
- ② ①に関連し、プログラム構造はできるだけ静的に閉構造となっている。
- ③ プロセッサを操作するため、使い易くわかり易い、しかも豊富な機能を持つ適当な制御文が必要である。
- ④ 端末利用者に、現在操作の対象としているプログラムの状態(データ部分も含めて)を常に把握できることまたその状態の変更も可能にする。
- ⑤ プログラムの論理構造を変えない程度であれば、なるべく言語仕様により自由度を持たせ、利用者の負担を軽減する。
- ⑥ プログラムの翻訳時あるいは実行時にエラーが発生したら、その時点で直ちに適切なメッセージを出力する。このメッセージによりプログラムは、必ずなんらかの指示を与える。この結果プログラムの状態は、常に正しいものとして処理される。このため、エラーの発生した原因をプロ

グラマがすぐ把握できるよう、各種のデバッグ用補助ツールが内蔵されていることが必要である。

3. 設計方針の設定

会話型言語プロセッサの基本的処理方式として、①一括型 ②会話実行型 ③コンパイル・アンド・ゴー型が考えられる。これらの説明を図1に示す。

一括型及びコンパイル・アンド・ゴー型では、エディタで原プログラムを作成する段階で1文単位に構文チェックの機能を追加することなどにより、会話を強化することができる。

これら3方式の定性的な比較を表1に示す。一括型は、実行時間は小さいであるが、プログラムの修正時間は大きくなり、会話機能も乏しい。会話実行型は、実行時間は大であるが、プログラムの修正時間は小さく、会話機能やデバッグ機能がすぐれている。コンパイル・アンド・ゴー型は、両者の中間にあたると思われる。

筆者等は、一括型のコンパイラが既に開発済みであること (DEMOS-E: DENDEN KOSHA MULTIACCESS

表1. 各種プロセッサ方式の特徴と適用

項目 \ 方式	一括型	会話実行型	コンパイル・アンド・ゴー型
翻訳時間	大	小	中
実行時間	小	大	小
プログラム修正時間	大	小	中
会話機能	小	大	中
デバッグ機能	普通	良い	普通
実行時の論理チェック	利用者に依存	一部自動化	利用者に依存
処理可能プログラム規模	大	小	小
適用	通常のプログラム作成用 会話型のデバッグ専用 小規模のプログラムのデバッグ専用		

ON-LINE SYSTEM-EXTENDED: 用FORTRANコンパイラ以下FCOと略す。)から、TSSにおける会話機能及び操作性を重視する立場から会話実行型方式を採用することとした。

しかしながら、会話実行型では、実行時間が大きくなり、一括型に比べて数倍から数十倍になるといわれており、商用システムにおいて、コスト的に使われうるものとなるかが問題となった。

そこで、会話実行型はプログラムのデバッグに適していることから、デバッグ時に使用した場合の所要CPU時間に

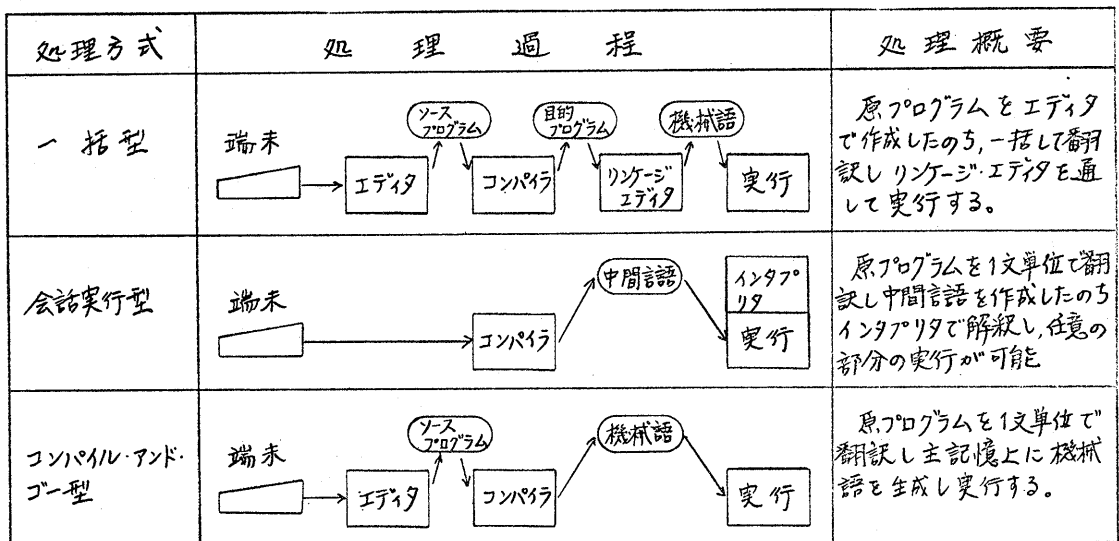


図1. TSS用言語プロセッサの基本的処理方式の例

着目して評価を行うこととした。

まず、デバッグ過程の一般的なモデルとして図2(ii)を想定した。このモデルの各処理過程に既存システムでの処理時間よりエディタの処理時間を1とした場合の各処理時間を求め、コンパイルエラーによる修正回数、実行時エラーによる修正回数を変えた場合の所要CPU時間を求めた。各処理方式における実行時間と所要CPU時間の関係を図3に示す。

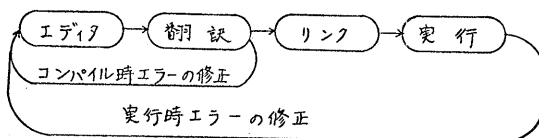
図3.は、会話実行型において、一括型に対するコンパイル時間比 $\alpha = 1/10$ 、実行時間比 $\beta = 10$ とし、かつ全プログラムに対する修正文の割合を $1/10$ 、部分実行文数の割合を $1/10$ として求めたものである。

図3.よりデバッグ時における実行回数が4~7回以上になれば、CPU時間の上でも会話実行型が有利となることが確認された。

以上の検討をふまえて、CFOの設計方針を次のように定めた。

CFOの設計方針

- ① 処理速度について次の性能を目標とする。



ii) 翻訳, 実行過程のモデル

方式 \ 過程	エディタ	翻訳	リンク	実行
一括型	1	5	2	6
会話実行型	1	$k_1 \times 5\alpha$	—	$k_2 \times 6\beta$
コンパイルアンドゴ型	1	5	—	6

注 k_1 : 修正文数の割合 ($k_1 < 1$)
 k_2 : 部分実行文数の割合 ($k_2 < 1$)
 α : 一括型に対する翻訳速度比 ($\alpha < 1$)
 β : 一括型との実行速度比 ($\beta > 1$)

ii) 各処理過程の時間

図2. 会話言語プロセッサ方式の比較モデル

コンパイル時間

$$CFO / FCO \leq 1/10$$

実行時間

$$CFO / FCO \leq 10$$

- ② 部分コンパイルができること。

すなわち、プログラムの修正におけるコンパイルは修正文に関連した部分のオブジェクトの修正で済むこと。

- ③ 部分実行ができること。

すなわち、プログラム入力後の実行において、デバッグに応じて部分的な実行ができること。

さらに、会話実行型の商用システムにおける有効性を早期に確認するために、初期システムにおいては言語仕様はあまり欲ばらないこと、商用システムにこたえうる処理能力を持たせることから、次の設計方針を加えた。

- ④ 言語仕様はJIS 3000レベル

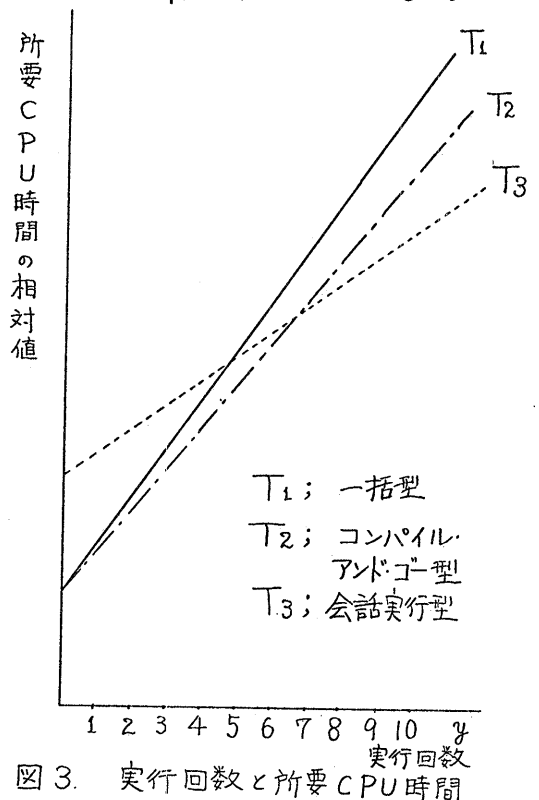


図3. 実行回数と所要CPU時間

とする。ただし、デバグが完了したプログラムがFCOでもコンパイルできるように互換性を重視する。

- ⑤ インコアで500文以上の処理ができること。

4. 機能概要

(1) 言語仕様

CFOの言語仕様は、デバグ用システムとして開発され、さらに高度な会話機能を持っていることで、処理効率及び使い易さを考慮した結果、次の水準で決定された。

- ① JIS FORTRAN 言語(水準3000)に準拠する。
- ② FCOとCFOでは前者に向けた互換性を持つ。
- ③ 会話型言語として有効な機能は、JIS 3000 レベルになくとも、FCOにあれば言語仕様に追加する。
- ④ 入力形式はカラムを意識しない完全な自由形式とする。これにともない、注釈行、継続行及び文の番号の定義がFCOと異なる。注釈行は、先頭有効文字として文字"'"がある行、また継続行については、文字"@'"と伝送制御文字(END OF TEXT)との連続投入によって次の行との継続を示すようにする。

- ⑤ 処理能力を極端に低下させ、しかも使用頻度の低い機能については、JIS 3000 FORTRANにあってモサポートしない。この結果EQUVALENCE文が削除された。

基本外部関数及び組込関数は、両者を含めて、システム関数として扱う。このシステム関数の英字名は、予約語であって、英字名として使用することはできない。

(2) サブコマンド

CFOは、1ジョブステップ内で複数の処理単位を持ち、プログラムの作成、編集及びプログラムのデバグなど

を一貫して処理することを可能としている。このため制御プログラムでサポートしているコマンドの他に、CFOプロセッサの制御下でのみ有効なサブコマンドを提供している。なお、サブコマンドは既存のDEMOS-Eのエディタ及びFORTRANデバグユーティリティ用サブコマンドと機能上の連続性を持たせている。CFOのサブコマンドは、機能上次の三つに分類できる。

① 制御用サブコマンド

プロセッサ及び利用者プログラムの実行制御、並びに処理の流れの切替を行うサブコマンドである。

② 編集用サブコマンド

プログラムの編集処理を行うサブコマンドである。

③ デバグ用サブコマンド

プログラムをデバグするために必要な機能を提供するサブコマンドである。

サブコマンドの一覧を表2.に示す。

表2. サブコマンド一覧

分類	サブコマンド名 (太字は省略形を示す)	機 能 概 要
制御用サブコマンド	/NEW	プログラム単位の新規作成
	/OLD	プログラム単位のファイルからの読込み
	/CLEAR	メモリ上のプログラムの初期状態化
	/COMPILE	再 翻 訳
	/START	プログラムの実行
	/CONTINUE	クイット割込みおよび/STOP サブコマンドによる一時停止後の処理続行
	/DISCONTINUE	クイット割込みおよび/STOP サブコマンドによる一時停止後の処理打ち切り
	/END	プロセッサの処理終了
編集用サブコマンド	/INSERT	文の置きかえと挿入
	/DELETE	文 の 削 除
	/REPLACE	文字列の置きかえ
	/RENUMBER	行番号の付けかえ
	/LIST	原プログラムのリストの出力
デバグ用サブコマンド	/TRACE	変数、配列および配列要素の値の追跡
	/FLOW	プログラムの流れの追跡
	/STOP	プログラムの一時停止点の設定
	/DUMP	変数、配列および配列要素の値の出力
	/SET	変数および配列要素の値の設定
	/REMOVE	デバグ情報の取消し

(3) 翻訳方法

プログラムを新規作成するときは、端末より1行単位に原文を入力する。コンパイラは、継続行を含めて1文単位にインクリメンタル・コンパイル処理を行い、中間言語と各種テーブルを作成する。

既存プログラムの再コンパイルは、/COMPILE サブコマンドにより行い、指定したプログラム単位の原文と行番号テーブルをもとに中間言語と各種テーブルを再び作成し直し、結果的に空をエリアのがベージ・コレクションができるようにしている。

(4) 実行方法

プログラムの実行には、部分実行と

〔例〕

```

      :
      :
#000500 [STX] /NEW△SUB [ETX]
      :
      :
#000020 [STX] SUBROUTINE SUB(A,B,C,D) [ETX]
      :
#000030 [STX] DIMENSION B(3,4) [ETX]
      :
      :
#000300 [STX] /START△SUB [ETX]
      :
      :
カリンスウ ロ セッテイ シマスカ、スルナラ「Y」、シグイナラ「N」デス・・・

[STX] Y [ETX]
A=[STX] 10.0 [ETX]
B([STX] 1,1)=3.0 [ETX]
B([STX] * [ETX]
C=[STX] 2.0 [ETX]
D=[STX] * [ETX]
      :
      :

```

図4. 副プログラムの実行

1文実行がある。

部分実行では/START サブコマンドで指定したプログラム単位の指定した区間の中間言語と各種テーブルを解釈実行し、その結果を出力する。実行の指定は副プログラム単独でも可能である。なお区間を指定しなかった場合はプログラムの先頭の実行文より実行する。

特に、副プログラムだけを取り出して実行する例を図4に示す。この場合副プログラムが仮引数を持っていると/STARTの入力後、プロセッサから仮引数の値の設定が必要か否かの問合わせがある。ここで「Y」を入力すると個々の仮引数の値の入力要求がある。また、設定の必要のないときは「N」を入力する。

1文実行は、FORTRAN文とサブコマンドによるプログラムの作成とは独立したカルキュレータ的な機能であり、算術代入文の形式での入力に対してただちに計算(実行)結果が出力されるものである。1文実行文は、図5に示すごとく、1文実行文であることを示すために第1桁目に\$ (ドル記号)を

〔例〕

```

      :
      :
#000100 [STX] $ X=625.0 [ETX]
      :
X = 625.0000

#000100 [STX] $ Y=SQRT(SQRT(X))+X [ETX]
      :
Y = 630.0000

#000100 [STX] /SET△T=0.630E+3 [ETX]
#000100 [STX] $ Z=X+SQRT(X) [ETX]
      :
F111 E ヘンスウメイ「X」ガ ミテイギ デアル

```

図5. 1文実行

付ける。また算術代入文の形式に限られ、システム関数の引用が可能である。

(5) デバッグ指定

デバッグ用サブコマンドの指定により、プログラムの流れの追跡(/FLOW)、実行途中の変数の値の出力(/TRACE、/DUMP)、変数などへの値の設定(/SET)及びプログラムの一時停止点の設定(/STOP)などを行う。なお上記のデバッグ用ディレール・ポイントの解除は、/REMOVEで行う。

デバッグ方法の1例を図6.に示す。ここでは実行中の変数、配列及び配列要素の値の追跡を行うために、実行の前に/TRACEで、指定点及び変数などを指定したものである。

〔例〕

```

      :
      :
#000500 [STX] /NEW△ABC [ETX]
      :
      :
#000080 [STX] J=1 [ETX]
      :
#000090 [STX] J=J+1 [ETX]
      :
#000100 [STX] A(J)=2 0+3・0 [ETX]
      :
      :
#000300 [STX] /TRACE△ABC. 100, J, A(J) [ETX]
#000300 [STX] /START△ABC [ETX]
      :
      :
      (実行)
      :
      :
[ABC      000100] J=      2
      :
      :
      A(2)=      5. 000000
      :
      :

```

図6. /TRACEによるプログラムのデバッグ指定

(6) 修正方法

プログラムの修正は、編集用サブコマンドを使用する。編集用サブコマンドの指定により、文の削除(/DELETE)、置換え及び挿入(/INSERT)、文字列の置換え(/REPLACE)、行番号の付替え(/RENUMBER)、原プログラムリストの出力(/LIST)の出力を行うことができる。なお原プログラムが修正されると同時に修正された文に関してコンパイルを行い、中間言語及び各種テーブルも修正されるため、直ちに実行可能となる。

図7.に/REPLACEを用いて、文とその文中の文字列の内容とそれに置換わるべき新しい文字列の内容を指定することによりプログラムを修正する例を示す。

〔例〕

```

      :
      :
#000500 [STX] /NEW△ABC [ETX]
      :
      :
#000200 [STX] READ (1, 101) D, E, F [ETX]
#000210 [STX] 101 FORMAT (3E12.5) [ETX]
      :
F065 ショシキショウ ガ カッコ デ カコマレテイナイ
#000210 [STX] /REPLACE△, :△3::△(3: [ETX]
#000220 [STX] IF (D) 250, 280, 300 [ETX]

```

図7. /REPLACEによるプログラムの修正

5. C F Oの特徴的処理

(1) プロセッサの制御方法

C F Oは、キーボード・プリンタを介して、プログラムと計算機システム間でインタラクティブなやりとりを行いプログラムを作成するマンマシンシステムである。マンマシンのインタフェース上に存在するものは、物理的には入出力装置があるが、論理的には、原プログラム、サブコマンド、メッセージ

などがあり、これらを言葉として会話を
を行う。

FORTRANは、元来一括処理向きに
言語仕様が制定されている。この一括
処理向きという意味は、次のように考
えられる。プログラム単位は、END
行が入力された時点で閉塞するもので
あってコンパイル処理は、この時点で
行わなければならない。

宣言文、文関数定義文をサポートす
るためには、プログラム単位の閉塞性
を前提として考えなければならない。

以上の状況で、FORTRANの言語
仕様に完全に忠実な会話型プロセッサ
をインプリメントしようとするれば、

- ① プログラム原文入力時には、構文
チェックのみを行い、入力を文単位に適
当な *internal form* に変換する。
- ② END 行入力により、プログラム
全体にわたる意味上のチェックを行い
処理系で処理可能な目的プログラム
(機械語)を作成する。
- ③ プログラム修正時には、修正文に
対する構文チェックを行った後②の処
理を行う。

となり、特に修正時における③の処理
は、単にプロセッサの制御を困難とす
るのみならず、システム効率の面から
も好ましくない。これらの状況を考慮
した結果、CFOにおいては、インク
リメンタルコンパイル後の *internal
form* として中間言語を作成し、その
際英字名テーブル、定数テーブル及び
各種のリスト・テーブル等の情報をその
まま保存しておくこととした。実行時
には中間言語を擬似命令として、ソフ
トウェアでシミュレートするインタプリ
タ方式を採用している。しかも実行時
には英字名テーブル等の情報を参照し
ながら行うため、英字名の属性のある
程度の変更は、英字名テーブル中の属
性情報を変更するのみでよい。

CFO設計時には上記の諸条件を加
味して、以下に示す基本的な方針を固

めた。

- ① プロセッサは、特別の状況を除き
任意のFORTRAN文、サブコマン
ドを受付ける。
- ② ①にあげた特別な状況においては、
利用者の次になすべき指示に制限が
加えられる。この際プロセッサは、
メッセージを出力し利用者の指示で
きる選択事項を提示する。
- ③ プログラム作成中であっても、そ
のプログラムは、実行文を含めば、
いつでも実行可能である。
- ④ ③に関連して、プログラム単位は
常に閉じられることがない。したか
って言語仕様上END行は除外した。
なおEND行が入力されてもプロセ
ッサは無視する。
- ⑤ 入力待ち状態であれば、対象とし
ているプログラム単位のどの部分で
も修正可能とする。
- ⑥ カルキュレータ機能を持たせた1
文実行は、それまで対象としていた
プログラムの処理と独立であり、し
かも連続に1文実行文を入力した場
合、1文実行シリーズとして、その
シリーズ中の1文実行の結果(変数
値等)は保持される。

(2) 原プログラム修正時の処理

1)で述べたように原プログラム修正
時には、場合によって英字名の属性の
変更を伴うことがある。

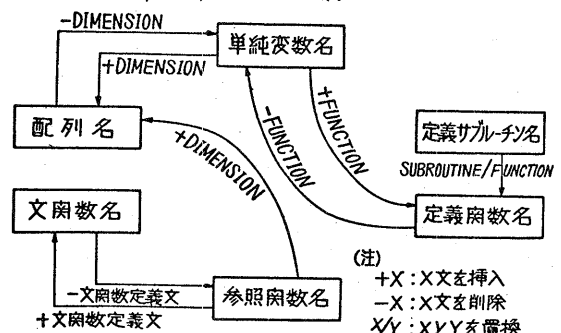


図8 修正時の英字名の扱い

英字名の属性のほとんどは、属性を宣言する文 (SUBROUTINE文、FUNCTION文、DIMENSION文、COMMON文及び文関数定義文) によって決定される。この属性を宣言する文を修正したときの英字名を何にするかが問題となる。そのため修正によって英字名の属性と、その英字名の使用形態との間に矛盾を引起しても何らかの手段で今エックアウトできる範囲で認めることとし、さらに現実面での有効性を重視し図8. に示した遷移を採用した。

6. 評価

CF0の完成後は、プロセッサの性能及び有効性を明確にする見地から評価を行った。

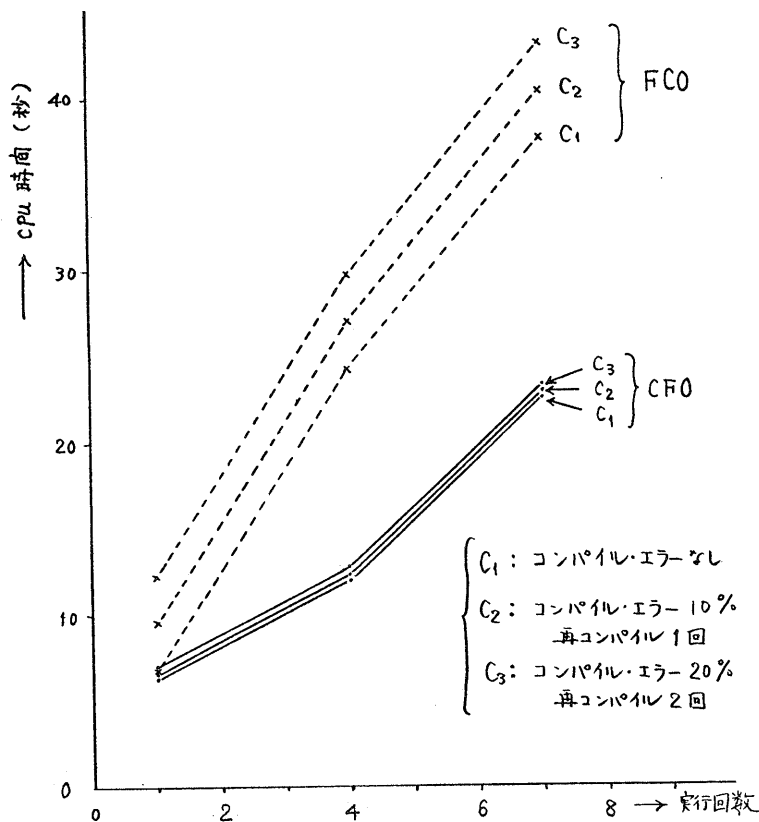


図9 プログラムデバグ完了までのCPU時間

6.1 性能評価

FC0とCF0との処理速度の比較を行った。

1) コンパイル速度

CF0については、/COMPILEを使用し再コンパイルを行った場合の処理速度について測定し次の結果を得た。

- ・CPU時間^(注1) $CF0/FC0=1/12$
- ・ダイナミックステップ数^(注2) $CF0/FC0=1/8.5$

2) 実行速度

- ・CPU時間^(注1) $CF0/FC0=1.7\sim 9.0$

(注1) 制御プログラム走行分を含む

(注2) 処理プログラム走行分のみ

6.2 ベンチマーク・テスト

デバグ時にCF0とFC0を同一条件で使った場合のCPU時間の比較を行うために、デバグ時の使用形態を模

擬し、コンパイル回数及び実行回数を変化させた場合のCPU時間を測定した。測定の結果を図9に示す。

図9より、実行回数が同じであればCF0の方がCPU時間が少なく有利であることがわかる。これは、修正に対して部分コンパイルで済み、FC0のようにエディット→コンパイル→リンク→ランの手順を必要としないことにより、無駄が少なかったためと考えられる。またコンパイル・エラーによる再コンパイル回数を変えてもCF0の場合にはCPU時間がさほど増加しないのは、部分コンパイルなどにより再コンパイルが効果的にできることによるものであると考えられる。

6.3 一括型コンパイラとの比較実験

CF0を実際に使った場合の有効性を評価するために、例題を用意し、コーディングを行いCF0とFC0で作成デバッグを行った場合のプログラム完成までのCPU時間などの要因について測定評価を行った。

この種の評価は、個人差に対する影響を少なくするために、できるだけ多勢でしかも綿密な実験計画のもとに行う必要がある。今回の評価では、10名を対象とし、図10に示す実験計画のもとで2回にわけて実験を行った。なお、本実験により、少なくとも1名につき両システムによりそれぞれ2題ずつ作成、デバッグを行ったことになるので実験終了時に、操作上の感想などを中心にアンケート調査を行った。

1) 評価項目

- ・コーディング時間(オc)
問題の理解からコーディング完了までの時間
- ・キーパンチ時間(オk)
最初にプログラムを入力する時のキーパンチ時間
- ・回線保留時間(オo)
端末を使用していた時間
- ・机上デバッグ時間(オd)
端末を離れて机上でのデバッグに費やした時間
- ・CPU時間
プログラム完成までの全所要CPU時間

	実行型	一括型	内 容	推定規模
1回目	P ₁	B	事務処理(図書館業務)	100
	P ₂	A	線型計画(生産計画)	100
2回目	P ₃	D	シミュレーション(バス路線)	100
	P ₄	C	行列演算(固有値計算)	100

A, B, C, Dは各5名のグループ

図 10 生産性評価の実験計画

- ・システム使用料金
回線保留時間とCPU時間より求めたシステム使用時間
- ・工数
プログラム完成までの全工数
- ・実行回数
- ・コンパイル回数
- ・リンク・エディット回数
- ・その他

2) 実行結果とその考察

実験データの平均値について図11に示す。データには個人差によるバラツキがみられたが、平均値としてとると、かなりよくシステムの特性をあらわしていると考えられる。

- ① コーディング時間はシステムに依存するものではないが、実際の実験においても大差はなかった。
- ② 実行回数はCF0の方が多くなっているが、これはいつでも実行できるという操作性の良さによるものと考えられる。
- ③ 机上デバッグ時間はCF0の方が短くなり、逆に、回線保留時間は多くなる。
- ④ CPU時間は、CF0の方が短くシステム使用料金の上でも経済的である。
- ⑤ 工数については、さほど差はみら

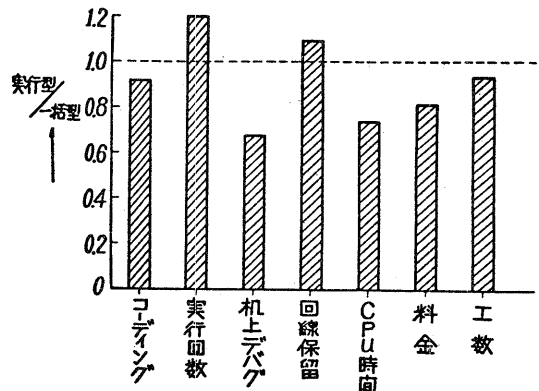


図 11. 生産性評価の結果

れない、これは本評価に用いた100
ステップ程度の問題では、全工程に
占めるコーディング時間の割合が約
1/2と大きく、CFOのデバグのし
やすさによる効果が全工数に影響を
与えなかったためと考えられる。

(3) アンケートによる調査

10名の担当者のうちFORTRANの
経験が1年以上のものは3名、1年以
下のものは7名であった。実験者のレ
ベルと使用後の意見についても考察を
行ったが、特に相関はみられなかった。

使用後の感想として、全般にCFO
のうが良いと答えた者が多かった。そ
の理由としては、①使い易いこと、②
修正やデバグ機能を手軽に利用できる
ことという意見が多かった。

なお、改善すべき項目としては、
①メッセージの簡素化、②言語仕様の
拡張、③リストを見やすくすること、
などが挙げられた。

7. おわりに

最近のコンピュータ化の急速な
発展により、その利用技術としての
ソフトウェア開発の爆発的増加が顕著
なものとなり、まさにソフトウェア危
機が現実のものとなっている。プログ
ラミングの生産性を向上させるため
には、ドキュメントの標準化及び簡素化
、生産性の良いプログラミング言語の
開発などと並んで、デバグ機能を強化
したデバグ用システムの開発が有効と
されている。

筆者等は、CFOを開発し、おもに
TSS利用者のインタラクティブなプ
ログラム作成及びプログラムテストに
効力を発揮することを確認した。当シ
ステムは、有効性を評価するためJIS
3000レベルで作られた試行システム
であり、現在社内サービスに提供され
ている。今回機能拡張を前提とした検
討において、上記の状況を加味して特
にプログラミングのデバグ機能を豊富
に持つデバグ用システムの重要性の認
識を新たにした。

付録. CFOの使用例

DEMOS-Eカーブ上でCFOを用いて4方陣を求めるプログラムを作成し、デバッグして解を求めるまでの使用例を以下に示す。

実行時におけるエラー・メッセージ (F111) は一括型コンパイラFCOではチェックアウトされない例である。

```

コチラへ DEMOS-E センタ デス

/000/ ON

74-09-17,17:39

/001/ LIB CFO
GEORTRAN カイシ
/NEW マタハ /OLD サブ コマント* ラ イレクタ*サイ.../NEW EX1
#000010 DIMENSION M4(4)
#000020 WRITE(2,100)
#000030 DO 60 I=1,4
#000040 DO 50 J=1,4
#000050 IF(I-J)10,20,10
#000060 10 IF(I+J-5)30,20,30
#000070 K=1
#000080 L=16
#000090 20 M4(J)=L
#000100 GO TO 40
#000110 30 M4(J)=K
#000120 40 K=K+1
#000130 50 L=L-1
#000140 60 WRITE(2,200)M4
#000150 100 FORMAT(1H1////10X,10H4 - HOUJIN
#000160 200FORMAT(/6X,4I4)
#000170 STOP
#000180 /STA
4 - HOUJIN

F111 E ハンズウェイ "L" カ ミテイ* デ*アル (EX1 000090)
#000180 /DUM L
[EX1 ] L = 1198
#000180 /SET L=16
#000180 /STA 90
F111 E ハンズウェイ "K" カ ミテイ* デ*アル (EX1 000120)
#000180 /DEL 70/80
#000180 /INS 22
#000022 K=1
#000180 /INS 24
#000024 L=16
#000180 /STA
4 - HOUJIN

16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1

STOP
  
```

← CFOの起動

4方陣を求めるプログラム

実行を指示

解の一部
(エラーのため、これ以降は出力されない)
エラー・メッセージ

Lの値の出力指示

Lに値(16)をセットする

90行目以降の実行を指示

修正

解

—— 下線部分はプロセッサからの出力を示す。