FORDAP(FORTRANプログラム動的解析システム)について

牛島, 和夫
九州大学工学部 : 助教授

https://doi.org/10.15017/1468049

出版情報：九州大学大型計算機センター広報. 9 (2), pp.100-120, 1976-06-01. 九州大学大型計算機センター
バージョン：
権利関係：

FORDAP — FORTRAN プログラム動的解析システム — について

牛島和夫*

1 はじめに

書かれたプログラムはまず読んで理解しやすいことが大切である．その上で能率の良さが要求される．Knuth〔1〕やIngalls〔2〕の実験によれば，入出力にとられる時間の占める割合の少いプログラムの実行時間の大部分は，書かれた原始プログラムテキストのわずか数%に集中しているという．従ってそのわずか数%の部分を発見しそこが能率よく働くような工夫を施してやることができれば，プログラムの実行時間は全体として格段に短縮されるというわけである．それではこの数%の部分をどのように見出せばよいのであろうか．プログラマにとって自分の作成したプログラムの実行時の動作を客観的に認識するのはなかなか難しいものである．

そこで，FORTRAN プログラムについて，プログラムの実行に伴って各実行文の実行回数や実行に要した時間を測定し，出力するシステムを筆者の手許にある FACOM 230−45S OSⅡで実現した〔3〕．このシステムは Knuth〔1〕にならって FORDAPシステムと呼ぶ．

我々の作成した FORDAPは主として次のような機能を持っている．

(1) FORTRAN プログラムの各実行文の実行回数を計数する．論理 IF 文にあっては，さらに論理式の値が真となる場合の数を計数する．

(2) プログラム単位ごとに実際に要した CPU 時間を計測し実行のコストとする（計測によるオーバーヘッドが含まれる）．

(3) 上記をわかりやすい形式で出力する．

(4) デバッグ支援のためにプログラムの異常終了時もそれまで得られた上記の情報を可能な限り出力する．

このシステムは，その後九大大型計算機センターの FACOM 230−75 Monitor 7，および東大大型計算機センターの HITAC 8800/8700 OS7に移し換えを行いそれぞれの OS のもとで使用可能になった〔4〕． 各OS の性格の違いから，それぞれ多少使い勝手の差があるが，ここでは Monitor 7 のもとにおける FORDAP の使用について，簡単な例題をあげて説明しよう．

2 例　題

A，B，Cを$n \times n$実行列とすると行列の積

$$C = AB$$

の要素は

$$C_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

---

*九州大学工学部助教授

によって定義される．ここで問題を特殊化して，B＝AかつA＝A^T（対称行列）の場合にCの値を求めるプログラムを考えてみたい．A，Cのためにn^2個の記憶場所が用意されていれば，Cも対称であることを考慮してCの下3角部分の要素のみ計算を行いしかる後上3角の等しい部分（$C_{ij}=C_{ji}$）に代入するというプログラムが考えられる．これを図1に示した．これはFORDAPシステムを利用したもので$n=100$の場合のものである．第1行目にEXECUTION TIME＝3298 MSECとあるのが，FORDAPシステムのもとでのこの副プログラムの所要時間，EXECUTIONS欄が各実行文の実行回数で，例えば

SUM＝SUM＋A(K,I)＊A(K,J)

という代入文は505000回実行されている（この文に実行が集中している）ことが一目で分る．またSUBROUTINE SPROD(N,A,C)のEXECUTION欄の値1はこの副プログラムが呼出された回数を示している．なおTRUE欄は論理IF文で真となった場合の数を示す．

さてここで考えようというのは記憶容量の問題で，A，Cとも対称性を利用すれば記憶場所はn^2個の代りに$n\times(n+1)/2$で済ますことができるはずである．そこで今，寸法$n\times(n+1)/2$の1次元配列を考えてそこに

$a_{11}\ a_{21}\ a_{31}\ \cdots\cdots\ a_{n1}\ a_{22}\ a_{32}\cdots\cdots a_{n2}\ a_{33}\ \cdots\cdots\ a_{n-1\,n-1}\ a_{nn-1}\ a_{nn}$

という順に行列の要素をしまうことにする（C_{ij}も同様）．この時C＝A·Aの計算をできるだけ能率よく行うプログラムを作成したいというのである．対称性から，

$$C_{ij}=\sum_{k=1}^{n}a_{ik}a_{kj}=\sum_{k=1}^{n}a_{ki}\,a_{kj}=\sum_{k=1}^{n}a_{ik}\,a_{jk}$$

が成立つがa_{ki}の記憶場所は$k\leq i$と$k>i$では並び方の規則が異なる．すなわち$k\leq i$のときa_{1i}，a_{2i}，a_{3i}……は，A(i)，$A(i+\overline{n-1})$，$A(i+\overline{n-1}+\overline{n-2})$……となっている．添字だけ取出せば（IAとする），IAは初期値をiとし増分を$(n-1)$，$(n-2)$，……$(n-i+1)$とする数列になる．一方$k>i$では，増分は常に1である．このことは，a_{kj}についても同じだから対称行列Aの2乗をCに求める副プログラムは図2のようになる．ここでMは整合寸法で値は$n\times(n+1)/2$が実引数として与えられる．この図で実行回数の集中している（499950回）Kのループの内部で行われ

```
* SOURCE STATEMENT *          EXECUTION TIME =    3298 MSEC  EXECUTIONS      TRUE

      SUBROUTINE SPROD (N,A,C)                                        1
      DIMENSION A(N,N),C(N,N)
      DO 2 I=1,N                                                      1
        DO 2 J=1,I                                                  100
          SUM=0.0                                                  5050
          DO 1 K=1,N                                               5050
            SUM=SUM+A(K,I)*A(K,J)                                505000
    1     CONTINUE                                               505000
          C(I,J)=SUM                                               5050
    2 CONTINUE                                                     5050
      DO 9 I=2,N                                                      1
        DO 9 J=1,N                                                   99
    9   C(I-1,J)=C(J,I-1)                                        4950
      RETURN                                                          1
      END
```

図1　n^2個の記憶場所を有する場合

ている添字の計算は，ALGOL 風に書いた方が理解し易いかもしれない：

IA：＝IA＋(if K≦I　then　N－K＋1　else　1)；

JA：＝JA＋(if K≦J　then　N－K＋1　else　1)；

このプログラム自体このように分り易いものであるが，ループの中で極めて多数回行われている計算をよく見るとKとI，KとJの比較について随分馬鹿馬鹿しいことをしていることに気付く．何となれば，Kは1ずつ大きくなっているのだから，一旦K＞Iとなれば，以後は常にK＞Iのはずである．Jについても同じ．それなのにKについて前回に得られた情報をKを1増した今回も同じように論理IF文で判定している．これが計算時間増大の元凶であろう．そこでこの無駄を減らせば計算時間は短かくなると予想される．ところでこのプログラムではIとJは独立ではない：

DO　10　J＝1，N

DO　10　I＝J，N

だから必ず，J≦Iが成立つ．従ってKとI，Jとの関係は，K≦J，J＜K≦I，I＜Kの3通りであることに考え到る（ただしI＝Jの場合は2通り）．すなわち（以下でNN＝N＋1）

2≦K≦Jの場合（K≦Iでもあるから）

IA＝IA＋(NN－K)，　JA＝JA＋(NN－K)

J＜K≦Iの場合　IA＝IA＋(NN－K)，　JA＝JA＋1

I＜K　の場合　IA＝IA＋1，　JA＝JA＋1

これを再びALGOL風に書いてみると次のようになる．

```
* SOURCE STATEMENT *        EXECUTION TIME =    6112 MSEC   EXECUTIONS       TRUE

      SUBROUTINE SPROD (N,A,C,M)                                      1
      DIMENSION A(M),C(M)
      IC=0                                                            1
      DO 10 J=1,N                                                     1
         DO 10 I=J,N                                                100
            SUM=A(I)*A(J)                                          5050
            IA=I                                                   5050
            JA=J                                                   5050
            DO 1 K=2,N                                             5050
               ISTEP=1                                           499950
               JSTEP=1                                           499950
               IF (K.LE.I) ISTEP=N-K+1                           499950  333300 ( 66.7% )
               IF (K.LE.J) JSTEP=N-K+1                           499950  166650 ( 33.3% )
               IA=IA+ISTEP                                       499950
               JA=JA+JSTEP                                       499950
    1       SUM=SUM+A(IA)*A(JA)                                  499950
            IC=IC+1                                                5050
            C(IC)=SUM                                              5050
   10 CONTINUE                                                     5050
      RETURN                                                          1
      END
```

図2　記憶場所を節約した場合

```
for K:= 2 step 1 until J do
  begin
    IA:=IA+(NN-K) ; JA:=JA+(NN-K);
    SUM:=SUM+A[IA]*A[JA]
  end ;
for K:= J+1 step 1 until I do
  begin
    IA:=IA+(NN-K) ; JA:=JA+1 ;
    SUM:=SUM+A[IA]*A[JA]
  end ;
for K:= I+1 step 1 until N do
  begin
    IA:=IA+1 ; JA:=JA+1 ;
    SUM:=SUM+A[IA]*A[JA]
  end ;
```

```
* SOURCE STATEMENT *          EXECUTION TIME =  3161 MSEC   EXECUTIONS      TRUE

      SUBROUTINE SPROD (N,A,C,M)                                      1
      DIMENSION A(M),C(M)
      NN=N+1                                                          1
      IC=0                                                            1
      DO 10 J=1,N                                                     1
      DO 10 I=J,N                                                   100
         IA=I                                                      5050
         JA=J                                                      5050
         SUM=A(I)*A(J)                                             5050
         IF (J.EQ.1) GO TO 2                                       5050    100 (  2.0% )
         DO 1 K=2,J                                                4950
            IA=IA+(NN-K)                                         166650
            JA=JA+(NN-K)                                         166650
            SUM=SUM+A(IA)*A(JA)                                  166650
    1    CONTINUE                                                166650
    2    IF (J.EQ.I) GO TO 4                                       5050    100 (  2.0% )
         J1=J+1                                                    4950
         DO 3 K=J1,I                                               4950
            IA=IA+(NN-K)                                         166650
            JA=JA+1                                              166650
            SUM=SUM+A(IA)*A(JA)                                  166650
    3    CONTINUE                                                166650
    4    IF (I.GE.N) GO TO 6                                       5050    100 (  2.0% )
         I1=I+1                                                    4950
         DO 5 K=I1,N                                               4950
            IA=IA+1                                              166650
            JA=JA+1                                              166650
            SUM=SUM+A(IA)*A(JA)                                  166650
    5    CONTINUE                                                166650
    6    IC=IC+1                                                   5050
         C(IC)=SUM                                                 5050
   10 CONTINUE                                                     5050
      RETURN                                                          1
      END
```

図3　論理IF文の実行回数の減少

ここでもしJ＝Iであれば，2番目のループ

for K:＝J＋1　step 1　until I do　……

は実行されないので実は，I＝Jの場合も自動的に含まれている．これをFORTRANに直して，FORDAPシステムのもとで実行したのが図3である．今度は3161 msecと実行時間は半減し，しかも図1よりも早くなるという驚くべき結果になった．このような原因は今の場合何といっても論理IF文の実行回数を減らして，論理的に場合分けできるものを計算機ではなく事前に人間が行っておいたからにほかならない．実際3個の論理IF文はそれぞれ5050回しか実行されず，図2の場合の100分の1にすぎなくなった．なおALGOLでは，この論理IF文に相当するものはfor ーーdo文の中に含まれて陽に現れてこない．このプログラムはもっと速くならないだろうか．

DO　5　K＝I1，N

のループの中で用いられているIAとJAは，増分が1で実はKと同期している．FORTRANではDOの制御変数を添字式にうまく用いると最適化機能を助けることが多い．すなわちこのループの中でIA＝IA＋1，JA＝JA＋1はIA＝IA_0＋(K－I)，JA＝JA_0＋(K－I)と同じことだから，（ただしIA_0，JA_0はそれぞれのループの直前のIA，JAの値とする）DOループに入る前に

$IA = IA_0 - I$，$JA = JA_0 - I$

として

SUM＝SUM＋A(K＋IA)＊A(K＋JA)

```
* SOURCE STATEMENT *          EXECUTION TIME =    2766 MSEC   EXECUTIONS      TRUE

      SUBROUTINE SPROD (N,A,C,M)                                      1
      DIMENSION A(M),C(M)
      NN=N+1                                                          1
      IC=0                                                            1
      DO 10 J=1,N                                                     1
      DO 10 I=J,N                                                   100
         IA=I                                                      5050
         JA=J                                                      5050
         SUM=A(I)*A(J)                                             5050
         IF (J.EQ.1) GO TO 2                                       5050    100 (  2.0% )
         DO 1 K=2,J                                                4950
            IA=IA+(NN-K)                                         166650
            JA=JA+(NN-K)                                         166650
            SUM=SUM+A(IA)*A(JA)                                  166650
    1    CONTINUE                                                166650
    2    JA=JA-J                                                   5050
         IF (J.EQ.1) GO TO 4                                       5050    100 (  2.0% )
         J1=J+1                                                    4950
         DO 3 K=J1,I                                               4950
            IA=IA+(NN-K)                                         166650
            SUM=SUM+A(IA)*A(K+JA)                                166650
    3    CONTINUE                                                166650
    4    IF (I.GE.N) GO TO 6                                       5050    100 (  2.0% )
         IA=IA-I                                                   4950
         I1=I+1                                                    4950
         DO 5 K=I1,N                                               4950
            SUM=SUM+A(K+IA)*A(K+JA)                              166650
    5    CONTINUE                                                166650
    6    IC=IC+1                                                   5050
         C(IC)=SUM                                                 5050
   10 CONTINUE                                                     5050
      RETURN                                                          1
      END
```

図4　ループの制御変数と添字計算

としてやれば多少の高速化を図ることができるかもしれない（この添字式の記述はJIS FORTRANの範囲外であることに注意）．よくみるとその上のDOループ　DO　3　K＝J1，IでもJAがKと同期しておりこれにも同様の処置を施すことができる．このようにして得られたプログラムをFORDAPシステムのもとで動かしたものが図4である．実行時間は2766 msecとなって図3に比して約13％の改善になった．よく考えてみるとIAやJAは添字をつきとめるために補助的に用いている変数で，最終的に必要な結果ではない（SUMも同じ）．DO　1　K＝……のループでは，アルゴリズム上どうしても必要だったが，DO　5　K＝……では，制御変数Kで置換えることができ，IAやJAに無用の代入を行わなかったことが13％の減少につながったものと思われる．13％というのは見ようによっては少いが，しかし13％にもなったのは，IA＝IA＋1などの代入文が16万回も実行されていたからにほかならない．この回数が少なければ，多少の工夫をこらしたところで実行時間の改善には寄与しない．

ところで図4のプログラムをいきなりつきつけられたとしたらこれは必ずしも見てすぐ分るとはいいがたい．やはり図2→図3→図4の経過があって始めて理解しうるものになっている．図4の副プログラムも全体のプログラムの中で1回しか呼出されないのなら6秒（図2を用いた場合）に対して約3秒の短縮になるにすぎないので，それなら分り易い図2を採った方がよい．しかし，もしこのプログラムが100回呼出されるとしたら3×100＝300秒の節約となりこれは非常に大きい．冒頭に述べた実行時間の集中しているわずか数％の部分を発見しそこの部分の能率向上を図ればよいというのはこういった意味である．

3　FORDAPシステム処理の流れとFORTRAN－Hコンパイラ

これまで述べたプログラムの高速化はFORDAPシステムの上でのことであった．実際はFORDAPをはずしたFORTRANのコンパイル――LIED―実行における実行が速くなってほしい．それを調べる前にFORDAPの機構を簡単に説明しよう．FORDAPシステムを利用してプログラムの実行解析をしようとする利用者は，固定型マクロ制御文を用いて次のようにする（固定型マクロ制御文の場合は¥QJOBカードは不要，可変型マクロ制御文については後述）．

```
¥NO
¥USER
¥FORDAP
FORTRAN 原始プログラム デック
¥*
  データ　カード
¥JEND
```

このようにして計算機に投入するとまず FORDAP プリコンパイラが働いて，第1節に述べたような機能を果すように原始プログラムの間に主として回数を計数する文を挿入したものを出力する．プリコンパイラのこの出力を普通の FORTRAN コンパイラ（ここでは FORTRAN－H OPT＝0）に渡し，コンパイル－結合編集－実行のステップを踏む．原始プログラムに対応する実行が終了すると，図1～図4に示したような FORDAP システムからの情報を付加した原始プログラムリストを出力し，その後に計算結果の印刷が続くようになっている．従って，FORTRAN コンパイラのソースリストは出力させないのが普通である．

以上の流れからわかるように，FORDAP システムが出力する EXECUTION TIME というのは，回数などを計数するための文の実行を含んだプログラムを FORTRAN－H（OPT＝2）でコンパイルしたものの実行時間である．そこで，FORDAP プリコンパイラをはずして原始プログラムを直接 FORTRAN－H（OPT＝0）にかけて計算した時の所要時間と比較してみよう．これを表1（a）に示す．

ここでF時間は FORDAP システムのもとでの所要時間，実時間は通常の場合の所要時間である．このようにF時間が実時間より時間が多くかかるのは当然としても改善度についてはかなり忠実に反映していることが分るであろう．ところで図3から図4への改善策は，コンパイラの最適化機能を意識したものであった．そこで FORTRAN－H（OPT＝2）を用いた場合を表1（b）に挙げる．図2から図4への改善度は，OPT＝0 を用いた場合（表1（a）より向上していることが分る．さら

表1　F時間と実時間の比較

（a）FORTRAN－H (OPT=0)の場合

プログラム	図 2	図 3	図 4	図 1
F時間	6112	3161	2766	3298
実時間	3489	2401	2231	2842

（b）FORTRAN－H (OPT=2)の場合

プログラム	図 2	図 3	図 4	図 1
F時間	6045	2827	2286	1235
実時間	3627	1841	1625	797

単位 msec

表2　FORTRAN－D，FORTRAN－H における計算時間の比較

処理系＼プログラム	図 2	図 3	図 4	図 1
D（NOOPT）	6838	3407	3694	3878
D （OPT）	4104	2546	2581	1255
H（OPT=0）	3489	2401	2231	2842
H（OPT=2）	3627	1841	1625	797

単位 msec

に記憶場所をn^2個用いた図1のプログラムが最も改善された図4の約半分の所要時間である．すなわち最適化コンパイラを用いた場合記憶容量を半分にすると所要時間がおよそ2倍必要ということになるのは面白い．

FORDAP システムは，FORTRAN－Hの上でしか動かない．その理由は主としてFORTRAN－Hのエラーモニター機能を利用していることによる（FORTRAN－Dにはこの機能がない）．もちろん，FORDAP を利用して改善したプログラムは，大抵の場合 FORTRAN－D でもそのまま計算できる．比較のため同じ4通りのプログラムを FORTRAN－D（OPT，NOOPT）にかけた結果を併せて表2に示した．

最適化なしの場合はいずれも（DもHも）図4の方が図1より速くなっているのは興味深い．この表から配列の添字の最適化機能などについては，FORTRAN－H（OPT＝2）はかなり高度なものがありそうである．九大センターでは，FORTRAN－H はまだ虫がいるという風評でDコンパイラの利用者が圧倒的に多いようであるが FACOM 230－75 のハードウェアに適したHコンパイラも早く使い良いものに仕上げて行くべきであろう．例えば表1のF時間と表2 D（NOOPT）の時間は同程度でデバッグ文を使うためにD（NOOPT）を使うというのなら，FORDAP を使ってみられるのも如何かと思う．

4　実行時における異常終了の処理

プログラムが異常終了すると計算機は計算機が検知したエラー原因，箇所，レジスタの内容などをダンプして停止する．これらの情報から原始プログラム上での誤り発生箇所やその原因を追求することはかなり困難な仕事である．そのために，コンパイル時や LIED 時の MAP などを予め採っておかなければならない．FORDAP システムでは FORTRAN－Hのエラーモニター機能を利用して表3に示すようなエラーが発生した場合には実行を中止して，それまで得られた実行回数の情報を付加した FORDAP リストを直ちに出力する．このリストで実行回数を調べて実行回数の喰い違いのある

表3　FORDAPシステムで対処できるエラー

エラー番号	内　　容
600 ～ 604	演算割込み関係
605 ～ 669（613,620,630,659 を除く.*）	入出力関係その他
670 ～ 689	単精度基本外部関数
690 ～ 709	倍精度基本外部関数
710 ～ 739	4倍精度基本外部関数
740 ～ 752	ベキ乗その他の関数

*
613：END OF FILE
620：READ，PRINT，PUNCH，F01～F99のファイル定義名でマルチファイルを利用しようとした．
630：非同期入出力のエラー
659：副プログラムの引数の個数が違う．

箇所から誤りの発生点を推定または決定することができる．図5はゼロで割ったために計算を中止して得られた出力である．これはあまりよい例とは云えないが，もしゼロで割ることがなければSTOP文の実行回数が1になっているはずである．ところがその直前の（Ｉ＋１）／Ｉで，ゼロで割ったため制御がこの文に達せず回数がゼロになっている．すなわちＪ＝（Ｉ＋１）／Ｉで誤りが発生したと推定できる．ここで推定できると書いたのは，もし代入文 Ｊ＝（Ｉ＋１）／Ｉでゼロで割る誤りが発生したとすれば，その次の論理ＩＦ文の実行回数は19999回のはずである．それにもかかわらず20000回と印刷されている．これは FORDAP システムの実行回数出力機構の事情による．このシステムでは一直線に実行できる一連の実行文を一かたまり（これを専門用語では基本ブロックという）にして実行回数を計数しているので，その中で誤りが生じたときでも基本ブロックに含まれる実行文は一応全部実行したものとして回数を割付けて出力する．従ってここでは20000回という同じ実行回数の一連の実行文のかたまりの中で誤りが発生していることをいいあて，ゼロで割ったのは（Ｉ+1）／Ｉと断定できるのである．

そのほか，最初に FORDAP システムを作成した FACOM 230−45 Ｓのもとでは，CPU 時間の超過（無限ループに陥った時などに生ずる）で計算が打切られた時もそれまでに得られた FORDAP 情報が出力できる．これは種々の点で大変便利であり（最も便利といってよい），Monitor 7のもとでも実現するべく，研究開発部で検討してもらっている．

```
* SOURCE STATEMENT *          EXECUTION TIME =      0 MSEC    EXECUTIONS       TRUE

C     ERROR EXAMPLE
C
      I=20000                                                        1
  100 I=I-1                                                      20000
      J=(I+1)/I                                                  20000
      IF (I.GT.0) GO TO 100                                      20000   19999 (100.0% )
C
      STOP                                                           0
      END



      FT604W  PROGRAM INTERRUPT-DIVIDE CHECK AT 0037D3

      TRACEBACK      ROUTINE     ISN    ENTRY LOC.   RETURN LOC.
                     MAIN                0037D62          *
```

図５　誤りを検出して異常終了した場合の例

5 FORDAPシステムの種々の利用法と使用上の注意

我々のプログラムはいつも例題に示したような簡単なものばかりではない．例えば SSL を使用したり，あるいはすでに作成ずみで RB としてファイルしておいた副プログラムを組込んだり，FORDAP で解析するのは一部のプログラム単位に留めたりという要求があるだろう．このような場合には，先にあげた固定型マクロ制御文ではこれを行うことができないので，可変型マクロを用意してある．固定型マクロと同じことを可変型マクロを用いて行うには次のようにする．

```
¥ NO
¥ USER
¥ QJOB
¥ FORDAPC
┌──────────┐
│ 原始プログラム │
└──────────┘
¥ FORDAPL
¥ FORDAPGO
┌──────────┐
│ データカード │
└──────────┘
¥ JEND
```

マクロ制御文については付録に一括しておいたので使用に際して参照してほしい．

このほか，使用に当って次の点に留意する必要がある：

(1) このシステムは FORTRAN－Hコンパイラとのみ接続し，Dコンパイラとは接続しない．

(2) 入力される原始プログラムは一旦 FORTRAN－Hコンパイラにかけて文法エラーがなく，さらに LIED でもエラーのないことを確認したものに限る．

(3) 継続行は 19 行までとする．（これは FORTRAN－DおよびHの制限と同じ）

(4) 論理 IF 文の論理式中に文字定数 '(' と ')' を含むと正しく処理されない．

(5) 次に示す英字名は使用できない：
SYS 900, SYS 901, SYS 902, SYS 903, SYS 904, SYS 905, SYS 906, SYS 907, SYS 908, SYS 909

(6) FORDAP プリコンパイラは原始プログラムの変換上必要な文番号を 65535 から0に向けて使用している．従って原始プログラムが 65535～65000 程度の文番号を使用していると文番号の二重定義になる場合がある．

(7) ファイル参照番号 97, 98, 99 が使用できない．

(8) 原始プログラムの第1欄が＊で始るカードは使用できない．

(9) 各副プログラム単位ごとに計測されている EXECUTION TIME は Monitor 7 の提供している時間を副プログラムの入口で CALL CLOCKM (*n*1) によって知り，RETURN 時に再

び時間$n2$を聞いて $n2-n1$ を副プログラムの実行のたびに累積するようにしている．ところでこの時間計測の最小単位は 1 msec なので，$n2-n1$ が極めて短かくしかも副プログラムが極めて多数回呼ばれると計測された時間は不正確になるので要注意．FACOM 230−45S OS Ⅱ の場合は最小時間間隔が 16 μsec なので問題はない．

6 おわりに

FORDAP システムの効用は第2節に述べた例題である程度理解していただけたものと思うが，特に異常終了時の処置を設けた事によって異常終了点の発見を原始プログラム上で行うことが容易になったこと，実行されていない箇所が一目見て明らかなこと，実行回数をチェックすることによってアルゴリズムのチェックができることなど，プログラム作成支援のために可成有力な手段であることを経験している．

我々の手許で FORDAP システムから得られた情報をもとにプログラムを評価し能率化した例については，文献〔5〕，〔6〕などを参照していただきたい．もしこのシステムを利用することによってプログラムの能率向上に著るしい効果をあげることができた場合には，できれば筆者または研究開発部までお知らせいただければ幸いである．プログラミング技術の know how を蓄積したいと考えているので紙上を借りてお願いする次第である．

なお，九大センターの Monitor 7 のもとにおける FORDAP 用ジョブ制御マクロの設計は，我々の研究室の藤村直美君が行った．マクロをセンターへ登録するに当っては，研究開発部の石田いつ子さんにお世話になった．

参 考 文 献

〔1〕 D. E. Knuth : An Empirical Study of FORTRAN Programs, Software Practice and Experience, Vol. 1 (1971), pp. 105−133.

〔2〕 D. Ingalls : The Execution Time Profile as a Programming Tool, Design and Optimization of Compilers, ed. R. Rustin, Prentice-Hall, 1972.

〔3〕 藤村，牛島： プログラムの実行解析システムの作成と使用について，九大工学集報，Vol. 48, No. 4 (1975), pp. 95−100.

〔4〕 藤村，牛島： FORTRAN プログラムの動的解析システムの移し換えについて，第17回プログラミングシンポジウム報告集，情報処理学会 (1976年1月), pp. 99−110.

〔5〕 牛島，藤村： 計算の手間の評価とプログラムの動的解析システム，京大数理解析研講究録 250 (1975), pp. 163−180.

〔6〕 牛島 : 数値計算プログラムの動作解析とその段階的高速化 ——— 3項方程式解法のプログラムの場合，bit 臨時増刊数値計算における誤差 (1975年12月)，共立出版 pp. 173−183.

付録　FORDAP ジヨブ制御マクロに関する説明*

(A) 固定型ジョブ制御マクロ

(1) 形式

¥FORDAP 〔PARAM=(〔(2,▼翻訳時のパラメータ▼)〕〔,(4,▼実行時のパラメータ▼)〕)〕

(2) パラメータの説明

通常は後述する例のようにパラメータは不要であるが，必要に応じて次に示すパラメータが指定可能である．なお特に指定しない時のパラメータは，

翻訳時のパラメータ＝▼NOLIST,COMP＝1，AREA＝16▼

実行時のパラメータ＝▼SIZE＝2▼

である．下線の部分はパラメータを指定する時は必ず記述しなくてはならない．

<table>
<tr><th></th><th>パラメータ</th><th>記　入　し　た　と　き</th><th>省略したとき</th></tr>
<tr><td rowspan="6">翻訳時に指定できるパラメータ</td><td>OPT2</td><td>目的プログラムの最適化をプログラム全体にわたって行う．</td><td>最適化しない．</td></tr>
<tr><td>BYNAME</td><td>すべての仮引数を番地によって引用する．
（CALL by reference）</td><td>原始プログラムで指定された通りに仮引数を引用する．</td></tr>
<tr><td>CDEX</td><td>FORTRAN D コンパイラのための追加機能を許す．</td><td>許さない．</td></tr>
<tr><td>NOTRBACK</td><td>実行時に発生するエラーの逆追跡情報を印刷しない．</td><td>印刷する．</td></tr>
<tr><td>NOARGCHK</td><td>実行時に実引数と仮引数の数の対応をチェックしない．</td><td>チェックする．</td></tr>
<tr><td>DOUBLE
DOUBLE (1)
DOUBLE (2)
DOUBLE (3)
DOUBLE (4)</td><td>プログラムで使用されている変数及び定数の精度下記の要領で拡張する．
<table>
<tr><th>パラメータ</th><th>機　　能</th></tr>
<tr><td>DOUBLE
DOUBLE (1)</td><td>REAL*4→REAL*8
COMPLEX*8→COMPLX*16</td></tr>
<tr><td>DOUBLE (2)</td><td>REAL*8→REAL*16
COMPLEX*16→COMPLEX*32</td></tr>
<tr><td>DOUBLE (3)</td><td>REAL*4→REAL*8
REAL*8→REAL*16</td></tr>
</table></td><td>原始プログラムの精度のままとする．</td></tr>
</table>

* 付録は藤村直美君（九大大学院工学研究科 D2）がまとめたものである．

	パラメータ	記入したとき		省略したとき
翻訳時に指定できるパラメータ			COMPLEX*8→COMPLEX*16 COMPLEX*16→COMPLEX*32	
		DOUBLE(4)	REAL*4, REAL*8 → REAL*16 COMPLEX*8, COMPLEX*16 → COMPLEX*32	
	COMP＝n	1度にコンパイルするエレメントの数を指定する.		n＝1となる.
	AREA＝n	コンパイル時の作業領域の大きさを指定する.		n＝16(KW)となる.
実行時のパラメータとして指定できるもの	DATAON＝5	ファイル参照番号5からの入力データを印刷する.		印刷しない.
	SIZE＝n*1	FORDAP システムでは入出力の作業領域として2KW必要とする. ENCODE, DECODE文を使用して余分な作業領域が必要な時にn(＞2)を指定する.		省略できない. n＝2とする.
	COND＝YES	STOP nを実行した時nを実行ジョブステップの完了コードとする. ただし実行中エラーが発生した時は無視される. (116ページ参照)		完了コードは000となる.

(3) 固定型ジョブ制御マクロ使用上の注意, 制限

(a) ¥QJOB カードは不要である.

(b) 一つのジョブ中で複数回使用できない.

(c) 大記憶上の原始プログラム, データを入力できない.

(d) 個人の相対形式プログラムライブラリからプログラムを組込めない.

(e) 実行時にファイルを使用できない.

(4) カードデックの構成例

標準的なカードデックの構成例を次に示す。

```
¥NO
¥USER
¥FORDAP ……①
    FORTRAN 原始プログラムカード
¥*      ……②
    データカード
¥JEND
```

① パラメータを指定する時は例えば次のようになる。

```
¥FORDAP  PARAM=(2,'NOLIST, OPT2')
¥FORDAP  PARAM=((2 'NOLIST, OPT2'),(4,'SIZE=
                2, DATAON=5, COND=YES'))
```

② データカードがない時は入力データ区切文は省略可能である。

(B) 可変型ジョブ制御マクロ

(1) 形式

```
¥FORDAPC  〔STEP=n〕〔,SOURCE=FILE〕〔,OUTPUT=REMOTE〕
¥FORDAPL  〔OPT2〕〔,BYNAME〕〔,CDEX〕〔,NOTRBACK〕
          〔,NOARGCHK〕
          〔,{ DOUBLE
             { (DOUBLE(1))
             { (DOUBLE(2))
             { (DOUBLE(3))
             { (DOUBLE(4))
          〔,COMP=n〕〔,AREA=n〕〔,NOLIST〕
          〔,STEP=n〕〔,OUTPUT=REMOTE〕〔,SYSINPRM
           =YES〕
```

¥ FORDAPGO 〔 STEP＝n 〕〔, SIZE＝n 〕〔 UNIT＝n 〕
〔 READ＝FILE 〕〔, DATAON＝n 〕〔, EBNAME＝実行形式プログラム名〕
〔, COND＝YES 〕〔, OUTPUT＝REMOTE 〕

常に3枚を1組として使用する.

(2) パラメータの説明

(a) ¥ FORDAPC

パラメータ	記入したとき	省略したとき
STEP＝n	1ジョブ中に FORDAPC を複数回使用する時異なった番号を指定する.（3桁以内） ジョブステップ名が重複しないように区別するためのもので, この指定があるとジョブステップ名は FORDAPCn となる.（119ページ例(e)参照）	ジョブステップ名は FORDAPC となる.
SOURCE＝FILE	カードでなく大記憶や磁気テープから原始プログラムを入力する時指定する. この時, ファイルを定義する制御文（¥ POFILE など）が後に必要となる. この時のファイルはデータファイルの形式なので注意が必要である.（118ページ例(d)参照）	原始プログラムはこの制御文に続くカードになる.
OUTPUT＝REMOTE	リストの出力先を端末とする.	センターのラインプリンタとなる.

(b) ¥ FORDAPL

パラメータ	記入したとき	省略したとき
OPT2 BYNAME CDEX NOTRBACK NOARGCHK { DOUBLE / (DOUBLE (1)) / (DOUBLE (2)) / (DOUBLE (3)) / (DOUBLE (4)) } COMP＝n AREA＝n	固定型ジョブ制御マクロ¥ FORDAP の翻訳時に指定できるパラメータに同じ. （111ページ参照）	左に同じ.

パラメータ	記入したとき	省略したとき
NOLIST	LIED の制御文のリストを出力しない.	出力する.
STEP＝n	¥FORDAPC（114ページ）に同じ. ただしジョブステップ名は FORTHn と LIEDHn となる. （119ページ　例(e)参照）	ジョブステップ名は FORTH と LIEDH となる.
OUTPUT＝REMOTE	リストの出力先を端末とする.	センターのラインプリンタとなる.
SYSINPRM＝YES	個人の相対形式プログラムライブラリからプログラムを組込む時など, LIED の制御文を必要とする時に指定する. （117ページ　例(b)参照）	LIED の制御文を必要としない.

(c) ¥FORDAPGO

パラメータ	記入したとき	省略したとき
STEP＝n	¥FORDAPC（114ページ）に同じ. ただしジョブステップ名は FORDAPGOn となる. （119ページ　例(e)参照）	ジョブステップ名は FORDAPGO となる.
SIZE＝n *1	FORDAP システムの実行時には入出力の作業領域として 2 KW 必要とする. この他に, ファイルを利用するなどして余分な作業領域が必要な時に指定する. 作業領域が不足の時は FT 905 Z のエラーが発生するので大きめに指定して, ジョブステップ・メッセージに出力される USED DOMAIN SIZE n KW のnにあわせてもよい.	SIZE＝2となる.
READ＝FILE	カードでなく大記憶や磁気テープなどのデータをファイル参照番号5から入力する時指定する. このときデータファイルを定義する制御文 （¥POFILE など）が後に必要となる. （118ページ　例(d)参照）	データは¥FORDAPGO に続くカードとなる.
DATAON＝n	ファイル参照番号nからの入力データを印刷する. nが5以外の時書式なし入力に関しては印刷されない.	印刷しない.
EBNAME＝実行形式プログラム名	¥FORDAPL で SYSINPRMを指定し, LIED の制御文 NAME で EXQTPRGM 以外の名前を付けた時に指定する. （117ページ　例(b)参照）	EXQTPRGM となる.

パラメータ	記　入　し　た　と　き	省略したとき
COND＝YES	¥FORDAP（112 ページ）に同じ.	
OUTPUT＝REMOTE	リストの出力先を端末とする.	センターのラインプリンタとなる.

(3) FORDAP システムにおけるエラー処理

FORDAP システムではプログラムの実行時に FORTRAN－H のエラーモニター機能を利用して次に示すエラー発生時には，実行を中断し FORDAP システムのリストを出力する．そこで，その情報を検討することによって，エラーの発生した場所とそのエラーの論理的な原因を容易に推定することができる．

FORDAPシステムで対処し得るエラー

エラー番号	内　　容
600 ～ 604	演算割込み関係
605 ～ 669（613,620,630, 659 を除く*）	入出力関係（その他）
670 ～ 689	単精度基本外部関数
690 ～ 709	倍精度基本外部関数
710 ～ 739	4 倍精度基本外部関数
740 ～ 752	べき乗その他の関数

* 613: END OF FILE

620: READ, PRINT, PUNCH, F01～F99 のファイル定義名でマルチファイルを利用しようとした.

630: 非同期入出力のエラー.

659: 副プログラムの引数の個数が違う.

上記以外のエラー，例えば FT900 ～908，記憶保護侵害，未定義命令の実行といったエラーには対処できない，なお，無限ループに陥った時などに発生する CPU 時間の予定超過は検討中である．FORDAP システムが正しくエラーに対処した時は，見かけ上正常終了するので，ジョブステップの完了コードは０００となる．

(4) カードデックの構成例

(a) ¥FORDAP（固定型ジョブ制御マクロ）と等価な例

```
¥NO
¥USER
¥QJOB
¥FORDAPC
```

原始プログラム　カード

```
¥FORDAPL ……①
¥FORDAPGO
```

データ　カード

```
¥JEND
```

①にパラメータ（OPT2）を指定すると次のようになる。

```
¥FORDAPL　OPT2
```

実行時に（作業用の）ファイルを使用する時はデータカードの次に　¥PSFILE,
¥F.WORK などの制御文を挿入する。

(b) プログラムの結合・編集時にライブラリから作成済みのプログラムを組込む例

```
¥NO
¥USER
¥QJOB
¥FORDAPC
```

原始プログラム　カード

```
¥FORDAPL　SYSINPRM=YES
NAME　EXQTPRGM,ENTRY=ELM(MAIN) ……①
CALL　　FORTLIBH,F.SSLH,P.LIB
SELECT　RELBIN,RLIB ……②
SELECT　PRVLIB ……③
FIN
¥POFILE　PRVLIB,F0027.PLIB ……④
```

```
¥ FORDAPGO
┌──────────────┐
│ データ　カード │
└──────────────┘
¥ JEND
```

①で EXQTPRGM 以外の名前を指定した時は，¥ FORDAPGO の EBNAME でもその名前を指定しなくてはならない．

②の RELBIN には現在翻訳された相対形式プログラムが，RLIB には FORDAP システムが必要とするプログラムが入っているので必ずこのように記入する．

③は④で定義される F 0027. PLIB というファイルに入っている相対形式プログラムを組込む指定である．

SYSINPRM を指定しない時は③を除いたものと同じ LIED 制御文が使用される．

(c) 原始プログラムの一部分を FORDAP システムにかける例

```
¥ NO
¥ USER
¥ QJOB
¥ FORTRANH
┌──────────────────────┐
│ FORDAP システムにかけな │ ……→ 主プログラムおよび STOP を含む副プログラムを除く
│ い原始プログラム　カード │
└──────────────────────┘
¥ FORDAPC
┌──────────────────────┐
│ FORDAP システムにかけた │ ……→ 主プログラムを必ず含むこと
│ い原始プログラム　カード │      STOP を含むプログラム単位は必ずここに含まれていること
└──────────────────────┘
¥ FORDAPL
¥ FORDAPGO
┌──────────────┐
│ データ　カード │
└──────────────┘
¥ JEND
```

(d) 大記憶から原始プログラムやデータを入力する例

```
¥ NO
¥ USER
¥ QJOB
¥ FORDAPC    SOURCE= FILE
¥ POFILE     READ,( F0027. DATA ( SC1 ))
```

```
¥FORDAPL
¥FORDAPGO READ=FILE
¥POFILE   READ,(F0027.DATA(DT1))
¥JEND
```

ここでは，F 0027. DATA というファイルに入っている SC1 という名前のついた原始プログラムを FORDAP システムにかけ，同じくF 0027. DATA というファイル中の DT1 という名前のついたデータを使用して実行している．
ここで FORDAP のプリコンパイラは FORTRAN－H で作成されているため，原始プログラムはデータとして読まれる．そこで，次に示すように大記憶上にはデータの形式で作成する必要がある．また，一度に複数個のエレメントは処理できないので注意が必要である．

```
¥NO
¥USER
¥QJOB
¥DPLIBE   F0027.DATA,BS=300
EDIT   DDOLD0,*(SC1)
```

原始 プログラム　カード

```
EDIT/
FIN
¥JEND
```

(e)　一つのジョブで複数回 FORDAP システムを使用する例

```
¥NO
¥USER
¥QJOB
¥FORDAPC   STEP=1
```

原始プログラム　カード

```
¥FORDAPL   STEP=1
¥FORDAPGO  STEP=1
```

データ　カード

```
¥ FORDAPC   STEP＝2
┌──────────────────┐
│ 原始プログラム　カード │
└──────────────────┘
¥ FORDAPL   STEP＝2
¥ FORDAPGO  STEP＝2
┌──────────────────┐
│ デ ー タ　カ ー ド │
└──────────────────┘
¥ JEND
```

注＊1　n を計算する目安として次の計算式を示す．

$$W = \left\lceil \sum_{i=1}^{\text{個数}} \left(\frac{\text{ファイル}\,i\,\text{の作業領域}}{1024} \right) + 1 \right\rceil \quad (\text{Kw})$$

ファイル$_i$ の作業領域 ＝ 150 ＋バッファー長（w）× バッファー数

バッファー長：$\left\{ \begin{matrix} \text{BLKSIZE} \\ \text{BUFL} \end{matrix} \right\} / 4$　　バッファー数：　$\text{BUFNO} = \left\{ \begin{matrix} 1 \\ \underline{2} \end{matrix} \right\}$

これで求まる W ＝ FORDAP システムの作業場所 2 Kw 分を足して n とする．