論　文

小型計算機による FORTRAN チェックシステム*

栃内香次** 中村鎮雄*** 和田優子****

Abstract

In this paper, a program for pre-checking FORTRAN programs using a small scale computer is described. With the check program, the user can remove syntactical errors from his FORTRAN program before processing on the large scale main computer. Therefore, the increase of system efficiency and the reduction of job turn around time are expected.

The check program was used at the Hokkaido University Computing Center, and good effects on the performance of the main computer system were observed.

1. はじめに

一般にプログラムのエラーはコンパイルあるいは結合編集過程で発見できるいわゆる文法エラーと，目的プログラムを実行してはじめて発見できる実行時エラーとに大別できる．現在，中型以上の規模の計算機でかつ FORTRAN に限るならば，文法エラーの検出と表示に十分留意されたコンパイラを備えているのが普通であり，文法エラー検出のためにさらに特別な手段を用意する必要はないようにみえる．

しかしながら，日常多数のかつ多様なプログラムを処理している大学の計算センターという場からみると，なお種々の問題点がある．その2，3の例として，

1) エラーメッセージが適切でなく，初心者にとっては不親切な場合が多い．
2) 処理系によっては，ある種の文法エラーが存在しても検出されない場合がある．
3) 文法エラーのあるプログラムはコンパイルまたは結合編集過程で処理が打切られ，したがって処理装置 (CPU) 使用時間の極めて短いジョブとして働き，システムの効率を低下させる．

等があげられる．大学の計算センター，とくに大型計算機センターでは，初心者すなわち大学院修士課程学生の利用が非常に多く，一方プログラムは一般に短期間で更新され，文法エラー修正段階にある新しいものが多い．また，卒業研究時期には利用が集中し，ターンアラウンドタイムが数時間から1日以上になることも多く，利用者はそのような状況の下で文法エラーの修正を繰り返すことを余儀なくされる．

このような状況を改善する一つの手段として，主計算機とは別の副計算機で詳細な文法チェックを行い，文法エラーを十分除去したものを主計算機で処理するという方法が考えられる．ただし，このような方法が有効でかつ他の方法より有利であるためには，

1) 副計算機のターンアラウンドタイムが主計算機に比べて十分短い場合．
2) 副計算機での1回の処理に要する費用が主計算機に比べて十分安価な場合．
3) 主計算機の FORTRAN 処理系の検査機能に不備または不便な点があり，かつそれを修正するよりも専用の検査システムを作る方が簡単な場合．

等の条件が満足される必要がある．

北大大型計算機センターの場合，1)については前述のように混雑期にはターンアラウンドタイムが数時間になるのに対し，副計算機による文法チェックの処理は数分～十数分でできると見積られ，また 3) については，主計算機の大規模な FORTRAN 処理系に手を入れるよりは副計算機に新たに検査システムを作る方が容易であると判断された．なお，短いプログラムならば計算処理までを含めて副計算機で処理する方法も

* FORTRAN Check System Using A Small Scale Computer by Koji TOCHINAI (Faculty of Engineering, Hokkaido University), Shizuo NAKAMURA (Research Institute of Applied Electricity, Hokkaido University) and Yuko WADA (Hokkaido University Computing Center).

** 北海道大学工学部電子工学科
*** 北海道大学応用電気研究所 ME 部門
**** 北海道大学大型計算機センター

a) Source program.

```
1          DIMENSION A(100,100),B(30,30)
2          DO 400 I=1,20
3          LAMDA=((IALPHA/IBETA)**2-256*IGAMMA(I))/180
4          IERR(I+2)=LAMDA+I
5      300 GO TO I,(301,302,303,304,305)
6          IC=(2.0+3.0)+IALPHA
7      400 CONTINUE
8          STOP
9          END
```

Underlines show the place where error exists.

b) Error messages.

```
ISN    1     **  ERROR   NO.2007------ Symbol error in array declaration.
ISN    3     **  ERROR   NO.2702------ Incoincidence of the number of parentheses.
ISN    4     **  ERROR   NO.2605------ Not declared array name.
ISN    6     **  ERROR   NO.2806------ Not allowed combination of types.
```

c) List of statement numbers and names.

```
* STATEMENT NUMBERS *
   NUMBER          NUMBER          NUMBER          NUMBER          NUMBER          NUMBER
     400             300  *          301  N          302  N          303  N          304  N
     305  N
* NAMES *
   NAME      TYPE KIND    NOTE    NAME      TYPE KIND    NOTE    NAME      TYPE KIND    NOTE
   LAMDA   *   I   VAR.           IALPHA      I   VAR.           IBETA   N   I   VAR.
   I       N   I   VAR.           IC      *   I   VAR.
```

Fig. 1 An example of the result from the check program.

考えられるが，われわれの場合両計算機の FORTRAN に相違点がかなりあり，ジョブ制御言語も異なっている等の理由で，この方法はとらなかった．

以下に述べる文法チェックプログラムはこのような背景のもとで作成され，北大大型計算機センターで実用に供され，満足すべき成果を得たものである．なお，使用した副計算機はかなり小型のもので，前記の条件2)は明らかに成立する．さらに，このシステムの利用に対する利用者の負担を無料として，利用者側からみてこの条件が一層有効になるようにした．

2. チェックプログラムの機能

この文法チェックプログラムは北大大型計算機センターの主計算機，FACOM 230-60 の FORTRAN 原プログラムを検査対象とし，ジョブ制御文，主プログラム，副プログラムおよびデータからなる1個のジョブをそのまま処理できる．プログラムは同センターの副計算機，FACOM 230-25 (主記憶容量 24 KB) で動作し，処理の概略は以下のとおりである．

まず，原プログラムの各文が行単位で読込まれ，変数，関数名や文番号等が表に登録され，文中の区切り記号，カッコ，式の形式や型等が検査され，構文エラーがあれば検出される．さらに，文番号の相互関係が検査され，DO 文の入れ子に関するエラーや文番号の二重定義エラーなどが検出される．END 行に到達すると表をしらべて未定義の変数，文番号や副プログラム引数の個数，型の不一致等が検出される．このようにして，主プログラム，副プログラムに関する検査がプログラム単位ごとに行われる．

ジョブ制御文は FORTRAN ジョブに限っても非常に多様な組み合わせがあり，完全な検査を行うためには膨大なステップを要するので，標準的な制御文についてごく簡単な検査を行うにとどめている．遠隔地の計算センターにジョブを輸送して処理する場合のようにターンアラウンドタイムが極めて長くなる場合は，プログラムが大規模になってもジョブ制御文まで含めて厳密な検査をすべきであり，たとえば著者の一人が以前開発に参加した検査システムではジョブ制御文の検査を相当重視していた[1]．しかし，本検査プログラムの場合は同一センター内で処理されるジョブが対象なので，上記程度の検査を行えば一応満足できる．また，データに関しても詳細な検査は行っていず，カラムの位置を明示してリスティングを行い，目で見て確認しやすいようにしている程度である．

このチェックプログラムを使用して検査を行った結果の例を **Fig. 1** に示す．原プログラム（a）が印刷された後，検査結果（b）が出力される．エラーは番号で表示され，別なマニュアルを参照して内容を知るようになっている．最後に文番号，変数名等の一覧表（c）が印刷される．図では省略されているが，さらに未定義，未使用の文番号，変数名が別に印刷される．

3. プログラムの設計

3.1 構 成

プログラムは以下に示す3フェーズからなる．

1) **フェーズ1** 原プログラムの各文を読込み，分

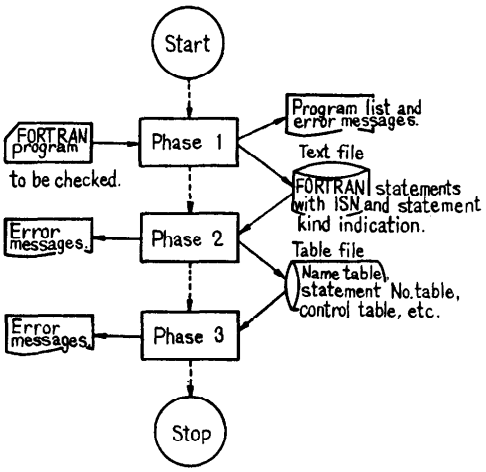

Fig. 2 General flowchart of the check program.

類番号をつけてファイルへ書込むとともにそれを印刷し，同時にジョブ制御文の検査を行う．

2) **フェーズ2** フェーズ1で作られた分類番号付きの文を読込み，文単位の構文チェックを行うとともに，新たに出現した変数，副プログラム，文番号等の名前と属性等を表に登録し，すでに登録済の場合は既登録情報との整合を検査する．一方，プログラムの流れに関する情報は別な表に記録され，検査される．

3) **フェーズ3** 登録されている変数，副プログラム，文番号などの一覧表を印刷し，あわせて未定義使用などを検出し表示する．

以上に述べた処理の概略の流れ図を **Fig. 2** に示す．

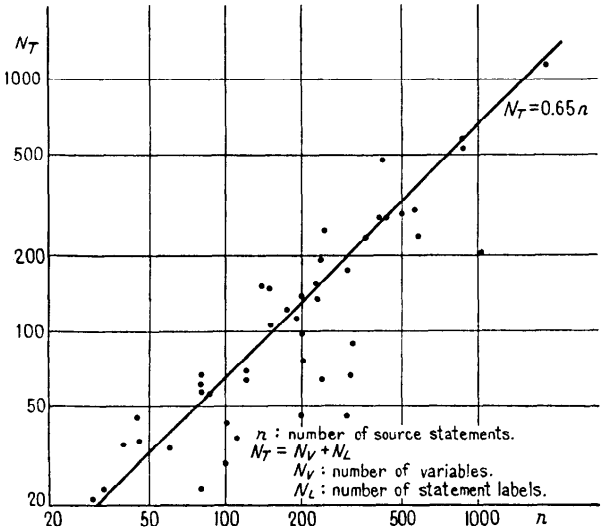

Fig. 3 Relation between n and N_T.

3.2 表の設計

変数，副プログラム，文番号，処理の流れ等の情報を記録するために種々の表が設けられている．その主なもの5種類の名称と記録内容を以下に列記する．

1) 名前表：変数，副プログラム等の名前と属性．
2) 副プログラム表：副プログラムの名前と属性．
3) 内蔵副プログラム表：ライブラリサブルーチンや基本外部関数など，システム内蔵の副プログラム類に関する情報．
4) 文番号表：文番号に関する情報．
5) 制御表：プログラムの流れに関する情報．

このチェックプログラムは，センターで日常処理しているFORTRANジョブのほとんどすべてを検査できることを目標にしているので，それを考慮して表の容量を定めた．表の容量に最も影響を及ぼす量は変数および文番号の個数である．いま，両者の和を N_T，原プログラムのカード枚数を n とおき，センターで処理しているジョブから抽出した代表的な 50 個について N_T と n との関係を求めると，**Fig. 3** に示すように両者の間にはかなりはっきりした相関があり，

$$N_T \simeq 0.65n \qquad (1)$$

とおける．そこで，以下のように表の設計を行った．

1) (1)式は変数と文番号の個数の和について成立しているので，両者を一つの領域に登録する．
2) n の最大値は本センターでは 1,600 枚位なので，(1) 式より，多少の余裕をみて N_T =1,100 とする．

Fig. 4-a)(次頁参照)に例として名前に関する情報を登録する名前表の構成を示す．1個あたりの必要語数は名前表，文番号表ともに5語なので，N_T=1,100 に対して 5,500 語の領域を必要とする．使用する計算機の主記憶容量は 12k語で，ユーザが利用できるのは 6～7k 語位しかなく，5,500 語の領域を主記憶に常駐させるのは無理である．そこで表は補助記憶装置(ドラム)におき，512 語単位でページングを行っている．すなわち表のアドレスをページ p とページ内相対アドレス A の組 (p, A) で表わし，ある名前または文番号が与えられたとき，ハッシュ法によってそれを記録すべき表のアドレスを定めている．具体的には名前または文番号を2字ずつに区切って各々 16 ビットの2進

数とみなし，符号なし加算を行ってそれらを加え合せた結果 x から次式によって p, A を求めている．

$$p=[x/11]_R \quad (2)$$

$$A=[x/97]_R\times 5+3 \quad (3)$$

ここで $[\ ]_R$ は剰余をとることを示す．(p, A) が定まった後，ページ p が主記憶にあるかどうかを調べ，なければそれまで主記憶にあったページをロールアウトしてからこのページをロールインし，ついでアドレス A にアクセスして所望の要素が得られる．以上をまとめ，ページ構成とアクセス手法を Fig. 4-b), c) に示す．

ハッシュ処理において同義語が発生したときは A に5を加える最も単純な処理を行っているが，本方式では p, A を各々独立に計算しているので分散がよく，同義語処理の結果ページがあふれて次のページにわたることはほとんどない．なお，同義語処理を円滑に行うために表には衝突および削除を表示する各1ビットのフラグ (Fig. 4-a) の c, d ビット) を設けている[2]．

ハッシュ法を用いると名前，文番号の出現順序は保存されない．プログラム作成者は名前，文番号の出現順序のあらましを記憶しており，それを念頭においてプログラムのデバッグを行うと考えられるので*，フェーズ3で名前，文番号の一覧表を印刷するときは出現順に配列している．このため表の各要素には次に出現する要素の格納場所を示すポインタを設けてある．

副プログラムに関する情報は変数や文番号と異なり，ジョブを通じて保存する必要がある．それで，索表の便を考えて名前表にも登録するが，さらに副プログラム表を設けてそこに記録している．また，副プログラムの引数に関する情報は別に設けた可変長の表に記録している．副プログラムには，プログラム作成者がその都度記述するものと予め作成されて組込まれているものとがあるので，表をさらに一般の副プログラム表と内蔵副プログラム表に分けている．前者は1ジョブに高々 50 個どまりなので出現順に記録し，後者は個数が多いので名前表と同様にハッシュ法を用いる．

* 他人のプログラムをデバッグするときは出現順よりもアルファベット順などに整理されている方がよいようである．

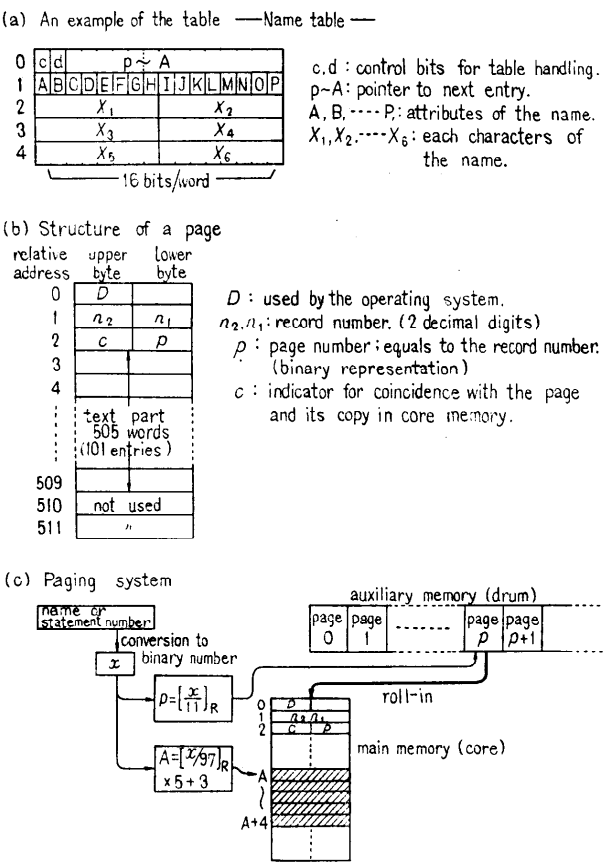

Fig. 4 The Table construction.

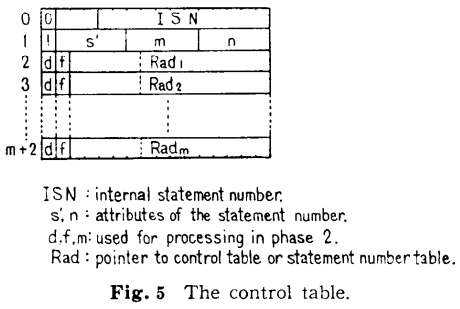

Fig. 5 The control table.

プログラムの流れに関する検査もまた，名前および文番号の検査とならぶ重要な項目である．これに必要な情報を記録する表が制御表で，GOTO 文などの制御文や文頭に文番号をもつ文が現われたときにその種類やプログラム中の位置，入れ子の深さ等の情報が記録される．表の各要素は出現順に記録され，1個の要素の構成は **Fig. 5** に示すとおりである．

3.3 検査アルゴリズム

本節では，プログラム検査の中核をなす数式や制御の流れの検査，未定義の名前や文番号の表示，およびエラー表示など，主要部分の処理について述べる．

1) 数式： 算術式や論理式は代入文をはじめさまざまの文にあらわれるので，数式処理サブルーチンによりこれらを一括して検査している．このサブルーチンは次に示す2ステップからなる．

(i) ステップ1：数式を被演算子一演算子対に分解し，被演算子はさらに名前登録等の処理を行ってから実数型，整数型等の型を表わす情報に還元する．

(ii) ステップ2：(i)で得られた被演算子一演算子対の列を構文則に基づいて解析し，式全体の演算結果に対する型を定める．

関数を引用している場合は副プログラム表を用いて以下のように引数の型と個数の整合を検査する．

(i) はじめて出現した関数の場合はその名前および引数の個数とその各々の型を表に記録する．

(ii) すでに記録されている関数の場合は記録されている上記の情報との一致を検査する．

2) 制御の流れ： この検査は制御表と文番号表を用いて行われる．文頭に文番号のついている文が現われると，その文のISN（内部文番号）およびこの文番号が記録されている文番号表の (p, A) が制御表に記録され，さらに定義点であることを示す情報がセットされる．一方文番号表には以上の記録を行った制御表の (p, A) が記録される．以下，この文番号を指定している制御文が現われると，対応する制御表の Rad 部にこの文番号の定義点に対応する制御表の (p, A) を記録する．計算型 GOTO 文や IF 文のように複数個の文番号を指定するものがあるので，制御表の Rad 部は可変長になっている．(Fig. 5, 6 参照)

文番号が未定義のまま何回も制御文で指定される場合の処理は，1-パス型のアセンブラなどで用いられる方法にならい，制御表中にその文番号に関するチェインを作り，定義されたときにこのチェインをたどって定義点の (p, A) を埋込むようにしている．制御表にはこれらの指示子の他，文の種類，DO の入れ子の深さなどの情報が記録され，それらを用いて入れ子構造の検査等が行われる．

3) 未定義，未使用の検出： 各文の処理が進み，END 行が現われると制御はフェーズ3に移り，

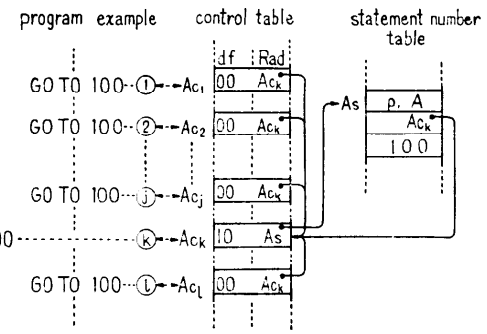

Fig. 6 Relation between control and statement number table.

定義，使用に関する情報を付けて名前，文番号の一覧表を印刷するが，さらに未定義および未使用のものを各々一括して印刷している．なお，これらの印刷は前述のように原プログラム中での出現順に行われる．

4) エラー処理： 各文の検査の結果エラーが発見されてもなるべく処理を続行できるよう，以下に示すようにエラーを3つのレベルに分けて処理している．

(i) レベル1：警告であって，放置してもコンパイル，実行可能なもの．検査もそのまま続行される．

(ii) レベル2：通常の構文エラーで，当該の文の検査は打切られるが，検査は次の文から続行される．

(iii) レベル3：表の容量超過等，設計条件を超えている場合で，検査はそこで打切られる．

エラーが発見された場合はそれに関する詳細な情報を出力し，容易に修正できるようにする必要がある．通常，コンパイラ等ではこれらエラーメッセージは英文またはローマ字表記日本文で印刷される場合が多いが，内容が不適切であったり，短かすぎて説明不十分である等の欠点が目につく．このチェックプログラムでは詳細なメッセージを出力することを意図したが，記憶容量の制約でメッセージをプログラムに内蔵することはやめて番号で表示し，内容は別に発行するマニュアルによることにした．この方法は必ずマニュアルを手許にもち，参照しなければならない不便さはあるが，エラーの種類に応じ，プログラム例をつけたり，図解を加えたり，あるいは修正法を示すなど非常に多彩なエラーメッセージを用意することができる．

エラー表示と原プログラムのリスティングとはフェ

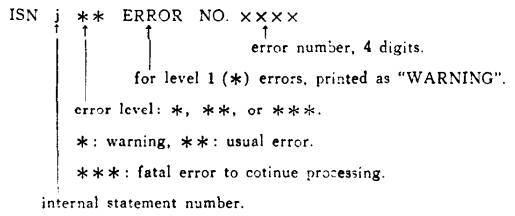

Fig. 7 Error message format.

ーズ 1 を除き分離されているので，メッセージには **Fig. 7**に示すように ISN を付してその場所を示している．エラーの大部分はフェーズ 2 で検出されるが，フェーズ 1 および 3 で検出されるものが若干ある．エラー番号は 4 桁で，プログラム作成上の便宜からその 1,000 位，100 位で **Table 1** に示すようにフェーズおよび文の種類の分類を行っている．エラーの総数は約 230 種で，その 90% 以上がフェーズ 2 に属する．

4. 使 用 実 績

4.1 主システムのジョブ処理への寄与

本チェックプログラムは 1973 年夏に試験的に稼動開始し同年 11 月より正式に公開された．その後，平均 1 日に 10 件程度利用されている．センターで処理されるジョブは 1 日 300～350 件程度で，その約 1 割，30件程度が新しく作られたプログラムであると考えることができ，その約 1/3 がチェックプログラムによって検査されていることになる．

既に述べたように，このプログラムで文法チェックを行うことにより，主計算機においてコンパイル時にエラーが検出されて処理を打切られるものが減少することが期待できる．このようなジョブは主記憶中に滞在する時間（コア占有時間あるいは USE 時間）に比べ処理装置を使用する時間（CPU 時間）が極めて短く，システムの効率を低下させる．したがって本プログラムの使用によってこのようなジョブが減少し，主計算機の処理効率が向上すると考えられる．**Fig. 8** は主計算機のジョブ処理統計から抽出した USE 時間/CPU 時間で，本プログラムの稼動開始後この比が低下し，効率が向上したことが明瞭に認められる．この図は，センターで処理するジョブのうち最も小さいクラスであるジョブ種別 A（平均CPU 時間 30 秒前後で上限は 3 分に制限される）に属するものの各月ごとの USE 時間/CPU 時間であって，1971 年 9 月より 1973 年 11 月までの平均値が 2.73 であるのに対し，それ以後は 2.21 になっている．なお，より大きなジョブの属するクラスである種別 B，C あるいは種別 A と同程度であるが遠隔地の端末からリモートバッチジョブとして投入され処理される種別 R のジョブ等については，この値は上記期間中ほぼ一定である．一般に新しいプログラムはまず種別 A のジョブとして文法ならびに内容の検討が行われ，エラーが十分除去されてからより大きな種別に移行すると考えられるので，種別 A のジョブについてのみ USE 時間/CPU 時間の低下がみられるのはこのようなテスト段階のジョブから文法チェックの部分がチェックプログラム使用へ移行したためであるといえる．また，種別 R のジョブで変化がみられないのは，これがリモートバッチジョブで，このプログラムの利用ができないためであり，これから逆に種別 A のジョブにみられる上述の変化が，本チェックプログラムが有効に利用された結果生じたことをうらづけている．

Table 1 Error number classification

error numbers	kinds of errors
0— 999	control card errors.
1,000—1,999	errors found in phase 1.
2,000—2,999	errors found in phase 2.
2,000—2,199	declaration or subprogram statement.
2,200—2,399	input/output statement.
2,400—2,599	control statement.
2,600—2,899	expression or statement function.
2,900—2,999	others.
3,000—3,999	errors found in phase 3.

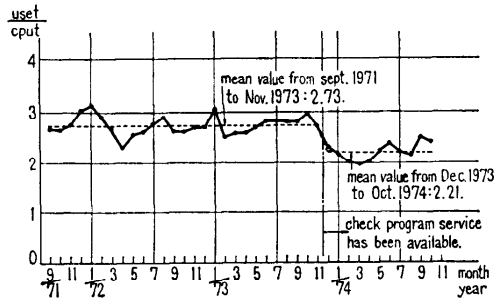

Fig. 8 Ratio of use time to CPU time for class A jobs.

次に，**Fig. 9**（次頁参照）は同じ期間における各月ごとの総処理件数/運転時間であって，システムが現実に単位時間あたり処理している量をあらわす．センターで処理されるジョブ量は 1 年周期で大きく変動し，既に述べたように卒業研究の計算が集中する 1，2 月に極大となり，4，5 月に極小となる．したがってその経年変化はあまり明瞭ではないが，12 ケ月以上の長期間にわたって平均操作を行い，周期変化をと

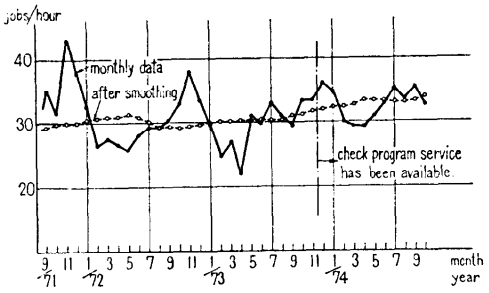

Fig. 9 Amout of jobs processed in unit time.

り除くとやはり若干の処理量の増加が認められ，1971年9月～1973年11月については平均 29.5 件/時間，それ以後については平均 32.7 件/時間という値が得られる．

なお，この量はシステムの処理能力をあらわすものではない．チェックプログラムの稼動により CPU 使用時間の短いジョブが減少すれば単位時間あたり処理件数はむしろ減少する．したがってこの結果は，チェックプログラムの利用によってデバッグに要するターンアラウンドタイムが減少し，同一の利用者が主計算機をより多数回利用できるようになったためと考えられる．すなわち主計算機の処理能力自体には余裕があり，ターンアラウンドタイムの減少により実際に利用それ自体が増大したことを示している．

また，Fig. 7，8 で期間を 1971 年 9 月から 1974 年10月に限ってあるが，これは主計算機 FACOM 230-60，副計算機 FACOM 230-25 というシステム構成で運転されたのがこの期間に限られているためである．

4.2 処理速度

本プログラムはドラムに格納され，オーバーレイ制御によって主記憶にロードされ，動作する．各種の表についてはロールアウトが必要なため，オーバーレイ制御によらず，前述のようなページングを行っている．最初の設計では名前表，文番号表の最初の 1 ページ分は主記憶常駐にし，カード 200 枚程度までの原プログラムはこの部分だけで処理して高速化をはかる方針であった．この方法ではより大きな原プログラムの場合でも，その先頭に出現しかつ全域にわたって使われることの多い配列名が表の常駐部分に記録され，高速化の点で有利と考えられた．しかし，プログラム本体が大きくなって常駐の表をおく余地がなくなり，この方法は実現できなかった．そのため表を引く際にほとんど必ずドラム～主記憶間の転送がおこり，またプログラムのオーバーレイ回数も多いので処理速度はあまり大きくはなく，平均規模のプログラムについて，カード入力からの全処理時間は原プログラム 1 行あたり 1 秒程度である．しかし，前述のような結果からみて，実用上は一応満足できるといえる．

5. おわりに

既に述べたように，このチェックプログラムは北大大型計算機センターにおいて，主計算機で処理する FORTRAN プログラムの文法チェック用に使用された．使用した副計算機の主記憶容量が小さいため，プログラムおよび表を常駐にできず，そのため速度は当初の想定より小さかったが，その他の点では満足できる性能が得られ，かなりよく利用された．その結果主計算機の動作状況が変化し，USE 時間/CPU 時間の減少，単位時間あたり処理件数の増加など，処理効率の向上や処理量の増加を示す結果が得られ，設計目的を達成することができた．

しかしながら，プログラムエラーに関するより大きな問題は文法エラーよりも実行時エラーにある．それ故，実行時に検出され表示されるエラー情報を解析し，その原因である原プログラム中のエラーを指示できる検査システムの開発が今後の課題となり，文法エラーの検出を主な目的とする本チェックプログラムを包含するより総合的なエラーチェックシステムへの発展を検討したい．

最後に，本研究に際し，歴代センター長，本学理学部田中 一，同工学部仲丸由正*，三浦良一，同応用電気研究所吉本千禎各教授には種々御示唆ならびに御便宜をいただいた．また，本学工学部村田茂昭，同理学部長田博泰両氏にはプログラムの設計，作成に際し種々御討論いただき，同大型計算機センター相良功，村守秀雄，斎藤 清，尾崎順子の各氏にはコーディング，デバッグ等にお手伝いいただいた．これらの方々の御助力に対し厚く感謝の意を表します．

* 現．釧路工業高等専門学校長

参考文献

1) 田中，村田，坂内他：HIPAC 103 による HARP 5020 ソースプログラムのエラーチェック（第Ⅱ報）；情報処理学会第 9 回大会講演予稿集 p.51 (1968 年 12 月)
2) 古川：コーフリクト・フラグをもったハッシュ記憶法，情報処理，Vol. 13, No. 8, p. 554 (1972 年 8 月)

（昭和 51 年 3 月 22 日受付）
（昭和 51 年 7 月 19 日再受付）