入力1文字当りの処理量によるプログラムの効率測定法の提案†

河村 知行††

プログラムの効率測定には，これまで，処理時間が多く使われてきた．しかし，仮想記憶機構などのために，正確な効率の測定や比較が困難になってきている．そこで本論文では，それに代わる測定方法として，IPC (instructions per character) を提案する．IPC は，入力1文字当りの機械語実行量である．文字処理を中心とするプログラムでは，IPC により，プログラムの改良部分だけから，改良による処理効率の向上度を知ることが可能となる．さらに，同じプログラムを異なった計算機上で走らせたときの効率の比較も，IPC では可能である．例として，PASCAL コンパイラの改良の度合を測定するために IPC を適用している．

1. はじめに

プログラムの測定方法は，これまでいろいろなものが提案され，実際に使用もされている．そして，処理効率・処理速度の測定には，その処理時間に基づくことが多かった．しかし，時間による比較は，曖昧な点が多く，さらに，異なる計算機では，比較の尺度とはならない．そのため，新しい尺度が必要であると思われた．そこで本論文では，プログラムの処理効率の新しい測定方法として，IPC (instructions per character) の概念を提案している．IPC は，ソーステキスト1文字に対する，測定対象となったプログラムの平均命令実行回数（機械命令）のことである．

本論文では，IPC の概念の具体的適用の実験例として，いままでに知られているコンパイラの高速化技法をとり上げている．2章から4章まで，実験したコンパイラ高速化技法の説明を行う．5章で，IPC の説明を行う．6，7章で，IPC を用いた実験の評価を行う．8章で，IPC を用いた発展的議論を行う．

2. 実験に使用したシステム

今回の実験に用いた言語は PASCAL である．実験は，H1PASCAL システム[1]により行った．この上で稼動する PASCAL コンパイラ（以下 H1PASCAL コンパイラと呼ぶ）は，ソーステキストを PASCAL 用機械語（以下 H1CODE と呼ぶ）に翻訳する．システムはこれをマイクロプログラムにより実行する．H1CODE は，Myers[2] の分類では「中位の機械語」に属しており，PASCAL の一文をいくつかの H1CODE に分解する方式である．H1CODE には，多重移動命令や文字列比較命令が存在するので，単純に従来の計算機と比較はできないが，一応の目安にはなろう．H1PASCAL システムのハードウェアは μCOM 16 というマイクロプログラマブル CPU により構成されているが，その実行速度が非常に遅いために（1μ 命令—2.4μ 秒），システムの実行速度はかなり遅いものとなっている．

H1PASCAL コンパイラは，H1PASCAL 自身により記述されており，TRUNK PASCAL[3] をもとにして作られている．H1PASCAL システムは，OS（入出力管理，ファイル管理）も PASCAL で記述してあり，以下の実験では OS にまで立ち入った改良もある．また，H1PASCAL コンパイラは，1PASS コンパイラであり，かつ，目的ファイルを作ることはせず，目的プログラムを主記憶上に作り出すのみである．これにより，コンパイル時の入出力は，ソーステキストの入力という必要最小限のものとなっている．

3. ソーステキストの形式

本システムでのソーステキストは，文字コードによるテキストファイルという最も古典的な形をしている．空白の圧縮等は行っていない．

今回の実験に，この形式を採用したのは，①多くのコンパイラが実際にこの形式を採用している，②実験を試みた高速化技法がこの形式に適合しやすい，③IPC の議論を明確にできる，という理由による．

† A Proposal of a Measurement of Program Efficiency by IPC (Instructions Per Character) by TOMOYUKI KAWAMURA (Department of Information Electronics, Tokuyama Technical College).

†† 徳山工業高等専門学校情報電子工学科

4. コンパイラ高速化の技法

4.1 1文字読込み手続きの改良

NEXTCH (TRUNK PASCAL コンパイラにならって，ソースプログラムの1文字読込み手続きを以下こう呼ぶ) は，コンパイラのなかで最も処理時間を消費している手続きであるといわれる[4]．実際，H1 PASCAL コンパイラでも，後述するように多くの処理時間を消費している．しかし，よく調べてみると，その処理時間の多くが入出力関係の手続きの呼出しに費されていることがわかる．TRUNK PASCAL コンパイラの NEXTCH には，EOF, EOLN, READ の三つの入出力手続き（関数）がある．一般に，ファイルに関する入出力手続きでは，そのファイルの状態（オープン/クローズ，リード/ライト状態等）が検査される．H1 PASCAL システムでは，この検査を1文字ごとに行っているため，かなりのオーバヘッドとなっている．これを解決する一つの方法として，READ 文の機能を拡張して，1行読込みを行う方法[4]がある．しかし，本実験は，「IPC の意味・測定を明確にする」という理由により，1文字読込み手続き NEXTCH の改良だけを行っている．

NEXTCH の改良は，次の3点により行った．

1) READ 文の代わりに GET 関数（1文字読込み関数．標準 PASCAL の仕様とは異なる）を用いる．GET 関数は，ファイル状態を検査しない関数で，本来 OS のなかでしか使われていない関数である．
2) 行末・ファイル末を示す制御文字をシステムで定め，行末・ファイル末に実際にその制御文字を置くことを OS の標準とした．行末文字として ASCII$(0A)_{16}$，ファイル末文字として ASCII$(03)_{16}$ を用いた．
3) GET 関数は制御文字もそのまま読み込むので，その値を保存しておいて，EOF, EOLN 検査は，文字コードの比較により行った．

以上のように，本来 OS しか使えない危険な関数を自由に使用している．これは，コンパイラは OS に準ずるプログラムであるという判断による．

具体的な改良は，**図1**(a) の NEXTCH を図1(b) に変更することにより行った．図中 CHINT は整数型変数で，直前に読み込まれた文字 (CH) に対する文字コードが格納されている．LF と ETX は，それぞれ，行末文字コードとファイル末文字コードを値とする定数である．GET 関数は広域変数 FCBP(図1(b) にはない．図1(c) では直接扱っている）が示す入力ファイルから，1文字入力を行う．

```
begin                                                    計182.4
  if EOL then ENDOFLINE;                                 2+(0.4)
  if EOF (SRCFILE) then EOFENCOUNTERED;                  56
  EOL:=EOLN (SRCFILE);                                   57
  READ (CH);                                             49
  if (CH<'␣')OR('_'<CH) then ILLCHAR;                    6
  if CHCNT < MAXCHCNT then                               3
    begin CHCNT:=CHCNT+1;
          LINEP↑[CHCNT]:=CH; end                         8
  else ERROR;
end;                                                     1
```

(a)

```
begin                                                    計43.7
  if CHINT=LF then ENDOFLINE;                            3+(0.67)
  1: CHINT:=GET;                                         16
  if CHINT=ETX then EOFENCOUNTERED;                      3
  if CHINT=LF then
        CH:='␣'else CH:=CHR(CHINT);                      4+(0.03)
  if (CH<'␣') OR ('_'<CH) then ILLCHAR;                  6
* if CHCNT < MAXCHCNT then                               3
    begin CHCNT:=CHCNT+1;
          LINEP↑[CHCNT]:=CHINT;end                       7
  else ERROR:
end;                                                     1
```

(b)

```
begin                                                    計10
  with FCBP↑ do
  begin
    if COLUMN>BLOCKSIZE then
      READ_NEXT_BLOCK;                                   3
    CH:=BLOCKBUFFER[COLUMN];
    COLUMN:=COLUMN+1;                                    6
  end;
end;                                                     1
```

(c)

図1 NEXTCH のプログラム
Fig. 1 Programs of NEXTCH.

図1のプログラムの右側の数字は，NEXTCH を呼び出したときの，H1 CODE での命令実行回数を示したものである．これは，H1 PASCAL コンパイラの動作を模擬して，ハンドコンパイルすることにより得られる．ENDOFLINE・EOF・EOLN・GET を含む行の値を求めるには，それぞれの手続き・関数のソーステキストが必要であるが，本議論には直接関係しないので割愛してある．カッコ内の数は，行末文字の処理に必要な命令数を1行の文字数（平均）で割った値である．

```
K:=0;
repeat
  if K< ALFALENG then
    begin K:=K+1; ID[K]:=CH; end;
  NEXTCH;
until not (CH in ['0'..'9','A'..'Z','_']);
if K>=KK then KK:=K
else repeat ID[KK]:='␣'; KK:=KK-1 until KK=K;
                    (a)
K:=0; ID:='␣␣␣␣␣␣␣␣';
repeat
  if K<ALFALENG then
    begin K:=K+1; ID[K]:=CH end;
  NEXTCH;
until not (CH in ['0'..'9','A'..'Z','_']);
                    (b)
```

図 2 識別子読込み部のプログラム
Fig. 2 Programs of identifier-reader.

4.2 識別子変数 (ID) の初期化

H1 CODE には，多重命令があるので，これを用いて，INSYMBOL（シンボル読込み手続きを以下こう呼ぶ）の中の識別子読込み処理部を**図 2**(a) から図 2(b) のように変更した．図中，K は読み込んだ識別子の長さを表し，KK は，前回読み込んだ識別子の長さを覚えている変数である．多重移動命令をもたない機械では，図 2(b) は改悪になるであろうが，これからの計算機は，この種の命令をもつべきであると考える．

4.3 予約語の判定のハッシュ化

TRUNK PASCAL コンパイラでは，予約語か名前かの判定は表探索により行われている．これを完全ハッシュにより判定するように改良した．

TRUNK PASCAL コンパイラでは，予約語と名前を最初の 8 文字で識別している．その 8 文字を C_i $(i=1,\cdots,8)$ とするとき，ハッシュ値 (H) は次の式により求めた．

$$B=(((((((C_1\cdot h_1+C_2)\cdot h_2+C_3)\cdot h_3+C_4)\cdot h_4+C_5)\cdot h_5+C_6)\cdot h_6+C_7)\cdot h_7+C_8$$

$$H=B \bmod M$$

この式の h_i $(i=1,\cdots,7)$ と M をいろいろ変えて実験を行い，標準 PASCAL の全予約語（'NIL' も含む）を完全ハッシュする h_i と M の組を探した．現在までに，次のような組が発見されている．

h_1	h_2	h_3	h_4	h_5	h_6	h_7	M
3,	2,	2,	2,	2,	2,	2	179
3,	3,	2,	2,	2,	2,	2	145
3,	2,	2,	3,	2,	2,	2	140
3,	2,	2,	2,	3,	2,	2	150
3,	2,	2,	2,	2,	3,	2	112
3,	2,	2,	2,	2,	2,	3	99
3,	2,	2,	2,	2,	2,	1	99
4,	3,	2,	2,	2,	2,	2	138

注 1) H1PASCAL システムが 16 bit 語のシステムであるため，Bの値が任意の名前に対して，32767 を超えないようになっている．

注 2) H1PASCAL システムは，外部ファイル文字コードは ASCII であるが，内部では内部文字コード (ASCII-$(20)_{16}$) を用いているので，C_i は，$(10)_{16}$ 以上 $(3A)_{16}$ 以下の値である．

注 3) 予約語または名前が i 文字 $(i\leqq 7)$ のときには，B の値は対応する C_i までの計算を行い，h_i 以後の計算を行わないで B の値としている．

今回の改良では，2 番目の (h_i, M) の組を用いて，INSYMBOL 中の識別子の読込みの部分を**図 3** のように行った．

```
識別子変数 ID と値 B の決定;
SY:=IDENT;
case B mod 145 of
  16: if ID='IF'          then SY:=IFSY;
  10: if ID='DO'          then SY:=DOSY;
       ⋮                       ⋮
 102: if ID='PROGRAM      then SY:=PROGRAMSY;
       ⋮
  3 4,5,6,7,8,9,11,⋯,144:;
end;
```

図 3 完全ハッシュによる予約語の処理
Fig. 3 Processing of reserved word by complete hash.

4.4 識別子探索のハッシュ化

TRUNK PASCAL コンパイラでは，識別子の探索は 2 分木探索により行っていた．これをハッシングによる探索に改良した．ハッシュ表は，識別子宣言のスコープごと（グローバルなスコープに一つ，ローカルなスコープにおのおの，RECORD 宣言のスコープにおのおの）に，ヒープ領域に作り出した．ハッシュ値は，識別子切出し処理部で求めた値 B を，ハッシュ表の大きさで割ったあまりを用いた．ハッシュ表は，ポインタだけの要素からなる配列として，配列名〔ハッシュ値〕が示す要素に，識別子に関する情報をもつレコードへのポインタを代入することにより，ハッシュ化を行った．同一のハッシュ値をもつ識別子は，対応するレコードをポインタで線形に連結（リンク）することにより，あふれを解決した．

メインレベルで宣言される識別子と，各ローカルレベル（トップレベル以外はすべてローカルレベルとする）で宣言される識別子の数には大きな差があるた

め，2種類のハッシュ表を用意した．後述の実験により，現在のところ，ハッシュ表の大きさは，メインレベル用で500，ローカルレベル用で30となっている．このハッシュ化により，コンパイラは，セルフコンパイル時に，4.8k語の余分なデータ領域を必要とした．

4.5 エラー検出機能の除去

コンパイラのある種の応用（8章で説明）では，入力ソースプログラムはコンパイル時エラーのないものであると仮定することができる．そこで，エラーを検出する部分を取り去ったコンパイラを作成した．ただし，case文の使用により，オーバヘッドなしにエラーの検出できる部分は残してある．また，そのような意味でのcase文による書換えも積極的に行った．

4.6 実行時エラー検査用コードの除去

実行時エラー検出用コードを生成するためのコンパイルオプションをON ($D+) にして生成したコンパイラよりは，オプションをOFF ($D−) にして生成したコンパイラのほうが高速と考えられる．この実験も行った．ただし$D−により除去されるのは，ポインタの値検査と，ポインタ変数への代入時の値検査と，範囲型変数への代入時の値検査である．case文と配列の添字に関する範囲検査は，H1 CODEでは必ず行われる．また，H1 CODEは，ポインタ値検査，範囲検査のための専用命令をもっているので，その検査のためのオーバヘッドは，一般の計算機に比べて，小さくなっている．

4.7 NEXTCH機能のINSYMBOLへの埋込み

NEXTCHの中で行われていた機能のほとんどをINSYMBOL中のcase文の中に埋め込むことにより，INSYMBOLの効率を落とすことなく，NEXTCHのいっそうの効率向上を図った．

1) NEXTCH中にあったEOF・EOLN・不正文字検査に対する処理部分をINSYMBOLの中に埋め込んだ．
2) コンパイル時エラーリストすらも出力しないシステムとすることにより，NEXTCH中の出力用バッファに関する処理を除去した．
3) NEXTCH中のGET関数の呼出しを除去し，代わりにOS用関数GETの必要な部分だけをNEXTCH中に展開した．
4) INSYMBOLにおける空白の読み飛ばし処理部分を，case文の中に埋め込んだ．

以上により，NEXTCHは図1(b)から(c)のように変更され，INSYMBOLは図4(a)から(b)のように変更された．図中，#CHARとある部分は，H1 PASCAL独特の機能で，#印の前にある値や変数の型を強制的に変更する機能（いわゆる型紙機能）である．

```
begin
  while CH=' ' do NEXTCH;
  case CH of
    'A','B',…,'Z': …;
    '0','1' …,'9': …;
          ⋮
  end
end
```

(a)

```
begin 1:
  case CH of
    'A','B',…,'Z': …;
    '0','1',…,'9': …;
    ' ',LF#CHAR: begin NEXTCH; goto 1 end;
    ETX#CHAR: EOFENCOUNTERED;
          ⋮
  −32#CHAR, −31#CHAR, −30#CHAR, −28#CHAR,
  −23#CHAR, −21#CHAR,…,−1#CHAR,
    64#CHAR,…,95#CHAR: ILLCHAR;
  end
end
```

(b)

図4 INSYMBOL中のcase文
Fig. 4 Case statement in INSYMBOL.

「二つのファイルブロックにまたがる行はない」というファイル構成（たとえば，可変長レコード可変長ブロックファイル）にすれば，ファイルバッファの終りは，行末文字ということになり，図1(c)のif文は，図4(b)のLF#CHARの部分に埋め込むことが可能である．しかし，本システムでは，固定長ブロック化されたストリームI/Oが基本であるので，if文は必要である．ただし，「固定長ブロックの中の実データの最後は，必ず行の終り（行末文字）で，残りはNULL文字」という構成にすれば，上記の改良も可能である．本実験では，この方式は試みていない．

5. IPCの概念

IPC (instructions per character) は，入力テキスト1文字当りの，命令実行回数である．IPCは，同じ働きをする二つ以上の異なったプログラムの効率を定量的に測定する尺度である．

従来，この種の尺度には，実行時間 (execution time) が使われてきた．しかし，キャッシュメモリ・仮想記憶が備わった計算機では，議論を不正確，不明確にする欠点があった．IPCの概念は，従来の実行時間に代わって，より比較・議論の容易な尺度を与えるものである．同じ命令体系をもつ計算機（速さは異

なってもよい．たとえば，同シリーズの計算機）上で動く，同じ働きをする異なったプログラムの効率を比較するのが，IPC の本来の使用方法である．しかし，商用の多くのノイマン型計算機は，COBOL 用の特殊命令を除けば，その命令体系は大きく異なることはないので，異機種上で動くプログラムについても，大雑把な効率の比較に利用することができる．これは，実行時間を尺度としたのではできないことである．

コンパイラの IPC は，あるソーステキストをコンパイルするのに，ソーステキスト1文字当り何個の機械命令（本実験では，H1 CODE）を実行したかという値である（コンパイラを X とするとき，IPC(X) で表す）．この値が小さいほうが，高速のコンパイラだと考えることができる．

定義から明らかなように，

IPC＞(NEXTCH の1回の呼出しでの命令実行回数)

が成立する．また，部分 IPC を，コンパイラの各機能別の IPC の値と定義する．すなわち，コンパイラが n 個の機能より成り，各機能の部分 IPC を IPC_i とするとき次式が成り立つ．

$$IPC=\sum_{i=1}^{n} IPC_i$$

今回の実験では，各機能別の IPC だけでなく，それぞれの改良による IPC の減少分を，その改良点に関する部分 IPC とした．すなわち，コンパイラ V を改良して，コンパイラ W を得るとき，IPC(V)－IPC(W) を，この改良に関する部分 IPC とする．

6. 各バージョンの IPC による評価

H1 PASCAL システムでは，CPU パネルスイッチの指定により，命令実行回数（H1 CODE の実行回数）を計測できるようになっている．この機能を用いて IPC の測定を行った．

本実験のテストプログラム（ソーステキスト）としては，H1 PASCAL コンパイラ V0 版を用いた．これは，2,966 行，86,252 文字よりなるプログラムである．

各版のコンパイラ V_i と，その改良点を以下に述べる．

V0： H1 PASCAL により記述された H1 PASCAL コンパイラ．何の改良もされていない．

V1： V0 における手続き NEXTCH を図1(a)から(b)に変更して高速化したコンパイラ．

V2： V1 における識別子変数の初期化を，図2のように多重代入命令により高速化したコンパイラ．

V3： V2 における予約語の判定を完全ハッシュ関数により高速化したコンパイラ．

V4： V3 における識別子の探索をハッシュ化したコンパイラ．

V5： V4 からコンパイル時エラー検出機能を取り去ったコンパイラ．

V6： V5 コンパイラを実行時エラー検査指定 OFF ($D－) でコンパイルして得られたコンパイラ．

V7： V6 における手続き NEXTCH・INSYMBOL をさらに改良したコンパイラ（4.7 節に示した改良）．

各バージョンで，V0 コンパイラをコンパイルしたときの命令実行回数・IPC・実行時間・1 H1 CODE 当りの実行時間を**表1**に示す．実行時間が長いのは，μCPU が遅いためである．1 H1 CODE 当りの実行時間のなかには，実行回数の計測(19.2μ秒)や命令デコード処理(約24μ秒)も含まれている．1 H1 CODE 当りの実行時間が，改良のたびに増えているのは，多重代入命令等の割合が増えたことによる．

V4 においては前述したように，メインレベル用とローカルレベル用のハッシュ表の大きさを変えて実験を行った．その結果を**表2**に示す．

各バージョン V_i の改良点に関する部分 IPC (IPC(V_{i-1})－IPC(V_i)) を，それぞれ NEXTCH 1，IDCLEAR，RESHASH，IDHASH，NOERRORCHK，

表1 各バージョンの命令実行回数
Table 1 Instructions executed for each version.

バージョン	V0	V1	V2	V3	V4	V5	V6	V7
命令実行回数 （百万命令）	20.10	8.13	8.00	7.41	6.50	5.82	5.49	2.65
IPC	233.0	94.3	92.8	85.9	75.4	67.5	63.7	30.7
実行時間 （秒）	1790	744	734	682	613	553	527	276
1 H1 CODE 当り実行時間 (μ秒)	89.0	91.5	91.8	92.0	94.3	95.0	96.0	104.2

表 2 ハッシュ表の大きさと，命令実行回数
(NEXTCH を除く)
Table 2 Hash table length and executed instructions (except NEXTCH).

ローカル用ハッシュ表の大きさ ＼ メイン用	100	300	500	1000	3000
1	3.76M				
5	3.17M				
10	3.09M				
20	3.06M				
30	3.09M	3.05M	3.05M	3.05M	3.06M
60	3.04M				

M＝百万命令

NOCHECK, NOLIST とする．NOCHECK はさらに，NEXTCH 2 と NOCHK に分解できる．すなわち，PASCAL コンパイラのコンパイル時オプションである実行時エラー検出機能を OFF にしたことによる手続き NEXTCH (図1(b)) の部分 IPC の減少分が，2であることが，手続き NEXTCH の字面上から得られる（ハンドコンパイルにより得られる）．この部分を NEXTCH 2 とし，残りを NOCHK とする．NOCHK は，実行時エラー検査をやめたことにより，手続き NEXTCH 以外で得られた高速化分である．さらに NOLIST も，NEXTCH 3 と INSY に分解できる．すなわち，図1の手続き NEXTCH は，図中に示したようにその部分 IPC をプログラムの字面上より求めることができる．これより，NEXTCH 3 と INSY は下記のように定める．

NEXTCH 3＝図1(b) の IPC－NEXTCH 2
－図1(c) の IPC

INSY＝NOLIST－NEXTCH 3

NEXTCH 3 は，NEXTCH の改良（図1(b) から (c)）により得られた高速化分，INSY は，INSYMBOL の改良により得られた高速化分である．IPC (V7) は NEXTCH 4, INSYMBL, OTHERS に分けられる．図1(c) の手続き NEXTCH の字面から計算される部分 IPC が NEXTCH 4 である．また，手続き INSYMBOL だけをループさせるプログラムを作って INSYMBOL の部分 IPC を求め，その値から NEXTCH 4 を引き去って，手続き INSYMBOL だけ (NEXTCH を含まない) の部分 IPC（これを INSYMBL とする）を求めた．残りの IPC を OTHERS とした．OTHERS は，コンパイラの本質的な仕事をしている部分と，コード生成部分より成っている．

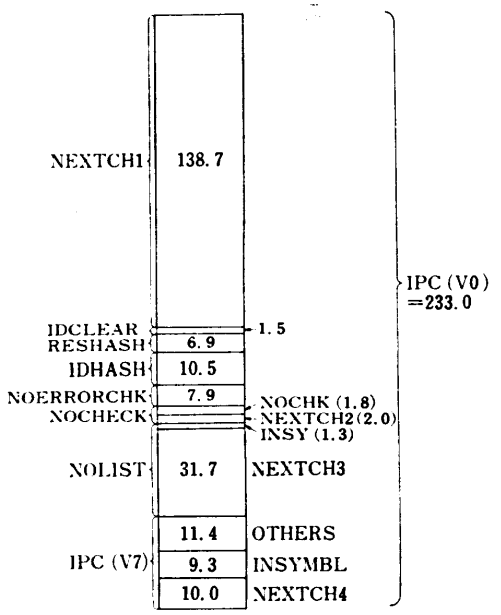

図 5 H 1 PASCAL における部分 IPC の構成
Fig. 5 Proportion of partial IPCs in H 1 PASCAL.

以上により得られた各部分 IPC を図5に示す．

7. IPC によるコンパイラの総合的評価

図1(a) の手続き NEXTCH に対する部分 IPC を NEXTCH 0 とするとき，

NEXTCH 0＝NEXTCH 1＋NEXTCH 2＋NEXTCH 3＋NEXTCH 4 となっていることがわかる．また，当然，次の式も成り立つ．

IPC(V0)＝すべての部分 IPC の和

図5からもわかるように，入出力の多いコンパイラでは，その処理の大部分が入出力とその周辺手続きに費されてしまう．コンパイラを高速化するとき，NEXTCH の高速化がいちばん大きな効果をもたらすことはいままでいわれてきたとおりである．NEXTCH 3 の大きさも見逃せない．たんに NEXTCH 中の入出力部分を高速化するだけでなく，そのなかで行っている種々の検査 (EOF, EOLN, 不正文字) を，手続き INSYMBOL 中の case 文の中に取り出せると，手続き NEXTCH をかなり高速化することができるのである．

V 7 は，コンパイル時，エラー検出・修復と，エラーリスト出力をしないので普通のコンパイラとは，性格を異にするものである．普通の意味でのコンパイラの IPC の最小値は，次の式で表される．

IPC (V 7)＋NOERRORCHK＋α

αは，コンパイルリスト出力用の行バッファに値をセットするための部分 IPC で，図1(b)の＊印の部分をつねに実行させるためのものである．図1(b)からもわかるように，$\alpha=7+3=10$ である．

このように，実際には作成していないバージョンのコンパイラの速度を定量的かつ正確に議論できるのが IPC の概念の特徴である．

8. IPC による議論

上述のコンパイラについて，IPC による議論を行う．

議論 1. コンパイラのローダ化：H1 システムでは，IPC (V7) は，30.7となっている．この値は，コンパイラをローダとして用いることを可能にする値だと考えられる．

1,000行の PASCAL プログラムの場合，

1,000行×30文字/行×30.7命令/文字×1μ秒/命令≒0.921秒

の式により，約0.9秒でコンパイル可能となる．30文字/行の値は，行末文字を含んでいる(V0の場合で29.1文字/行)．H1 CODE の実行速度を，1μ秒/命令としている点であるが，H1 システムでは，1H1CODE を実行するのに

約40μ 命令×2.4μ 秒/μ 命令＝96μ 秒

かかっている．これは，実行回数計測，命令デコード等の専用ハードウェアの使用と，μサイクルの高速化により，

20μ 命令×0.05μ 秒/μ 命令＝1μ 秒

くらいのものが，実在の技術で実現可能だと考えている．

議論 2. コンパイラの並列処理化：V7 のコンパイラにおいては，1文字読込み手続き「NEXTCH」と，1記号読込み手続き「INSYMBOL」と，「その他の部分」の IPC の比は，ほぼ，1：1：1である．また，「NEXTCH」と「INSYMBOL」と，「その他の部分」は，それぞれコルーチンとして独立に実行することが可能である．ゆえに，3マルチプロセッサによる処理方式を実現すれば，利用者から見た IPC は，10程度になるものと予想される．

以上のような議論は，IPC の定義と，その定量性から可能となるものである．

9. まとめ

今回の実験によりいくつかの所見が得られた．まず，第1に，最初に発表された PASCAL にはあった制御文字の必要性である．手続き NEXTCH は，制御文字の使用により，大幅に改良されている．行末文字やファイル末文字の使用は，コンパイラばかりでなく，一般のプログラムの効率をよくするものである．

第2に，OS の，利用者への開放の必要性である．大型計算機では，多数の利用者の互いの安全のために，OS を開放しないのは，当然の措置といえる．しかし，パーソナルコンピュータでは，すべての権利と責任を利用者に負わせることができる．もちろんこれは，すべての利用者が，OS の専用手続きを利用することを主張するものではない．OS の作成者ではないが，それに準ずる基本ソフトウェアの作成者の，OS 専用手続きの利用をいっているのである．

第3に，IPC の概念の有用性である．プログラムの効率測定のために，さまざまな手法が開発されている．しかし，IPC のような尺度は，あまり使われていないのではないだろうか．他の文字処理プログラムについても，IPC が効率測定の尺度として，利用されることを期待する．IPC の議論のためには，命令実行回数を計測する機能が計算機に要求される．しかし，命令実行回数は簡単なハードウェアの追加により，オーバヘッドなしで計測できるので，この機能が一般の計算機にも組み込まれることを期待する．

参考文献

1) 河村知行：パーソナルコンピュータの総合設計，第23回プログラミングシンポジウム報告集，pp. 1-10 (1982).
2) Myers, G. J. (福島正実訳)：コンピュータ・アーキテクチャの設計，p. 357，共立出版，東京 (1980).
3) Nageli, H. H.: PORTABLE COMPILER PROJECT, Institus for Informatik ETH, Zurich (1975).
4) 白濱律雄，前野年紀：字句解析部の高速化について，第20回プログラミングシンポジウム報告集，pp. 8-17 (1979).

(昭和58年5月9日受付)
(昭和58年11月15日採録)