UDK 681.3.06.004.15
Pregledni rad

O EFIKASNOSTI RAČUNARSKIH PROGRAMA

Zdravko GALIĆ — Sarajevo*

1. UVOD

Opšte uvažena težnja u procesu programiranja je da programi sadrže što manje operacija, neophodnih za rješavanje određenog problema, i da budu napisani na što koncizniji način. Ovaj princip proizlazi iz izvjesnih ograničenja računara, ustvari iz dva najznačajnija računarska resursa: centralnog procesora i centralne memorije.

Iako procesor računara izvršava instrukcije i operacije velikom brzinom, ona je ipak konačna i zavisi od konkretnog računara. Zato treba nastojati da program ima minimalan broj operacija, tj. da bude što efikasniji u smislu njegove brzine izvršavanja. Za vrijeme izvršavanja programa, program i podaci koji se obrađuju, nalaze se u (centralnoj) memoriji. Mada moderni računari posjeduju veoma velike memorije, one su uvijek konačne, pa stoga i programi treba da su napisani tako, da podaci i instrukcije koje treba izvršiti zauzimaju što je moguće manje memorije.

U ukupnoj ocjeni o kvalitetu nekog programa, pored kriterija kao što su: jasnoća i regularnost strukture programa, adaptivnost, itd. značajnu ulogu imaju efikasnost izvršenja i korištenja memorije.

Samo pitanje efikasnosti izvršenja i korištenja memorije (što je od posebnog značenja za korisnike koji koriste usluge drugih računskih centara), je upravo u neposrednoj vezi sa ukupnim vremenom angažovanja centralnog procesora i potrebnom veličinom memorijskog prostora.

Faktori koji utiču na efikasnost programa, napisanih u nekom od programskih jezika visokog nivoa, su veoma raznovrsni. Zbog toga, ako programer nastoji da kreira program koji će imati epitet najefikasnijeg, ili ukoliko namjerava izvršiti analizu i poređenje efikasnosti, trebalo bi u prvom redu da ima u vidu kompleksnost algoritma*, programski jezik i vrstu i performanse jezičnog procesora. Pored ovih faktora, na brzinu izvršavanja programa i zahtjeve za memorijskim prostorom, utiče u određenoj mjeri i računar na kome se program izvršava.

Međutim, u oblastima i disciplinama koje nisu neposredno vezane za računarsku tehniku i informatiku, kod analize efikasnosti programa za rješavanje određenog problema, uglavnom se razmatraju algoritmi, tipovi i strukture po-

Adresa autora: Zdravko Galić, dipl. ing. geod. Građevinski fakultet, Sarajevo, Hasana Brkića 24.

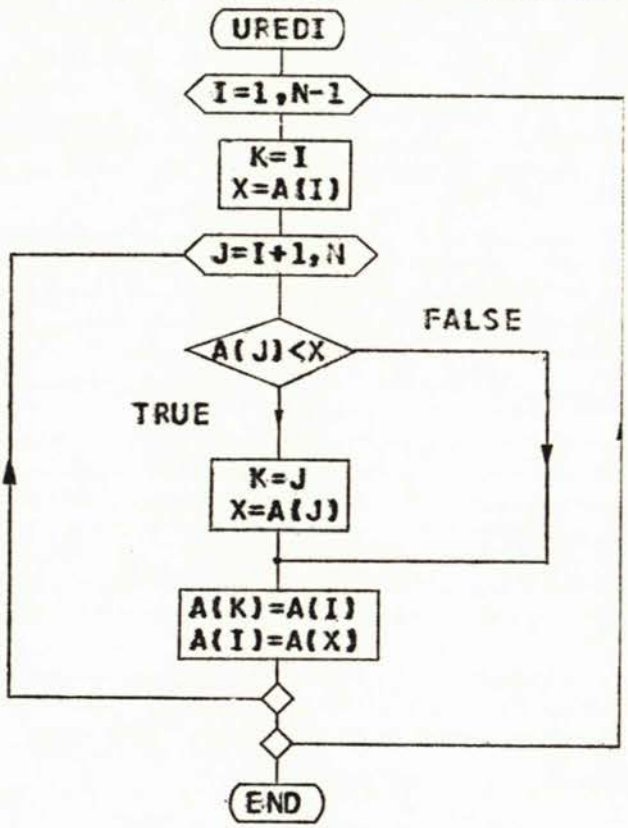
* Kompleksnost algoritma — računski napor, izražen brojem operacija.

dataka. Tako je npr. u [5] prezentiran jedan algoritam i program, koji je »... pogodan za primjenu u geodeziji«. (Vidi [5] str. 37.) Kritički se osvrćući na uobičajeni postupak za rješavanje matričnih jednačina ($X = A^{-1}B$, autori rada [7] Jovičić D., Lapaine M. i Petrović S., daju prihvatljive primjedbe na korišteni algoritam i strukturu podataka u [5]. Upuštajući se međutim, na određen način u ocjenu efikasnosti programa, pomenuti autori pored ostalog kažu da »cijena koju korisnik plaća prilikom izvođenja programa ovisi o trajanju izvođenja, a ovo pak o broju računskih i drugih operacija«, i nude program za rješavanje sistema linearnih jednačina napisan u BASIC-u.

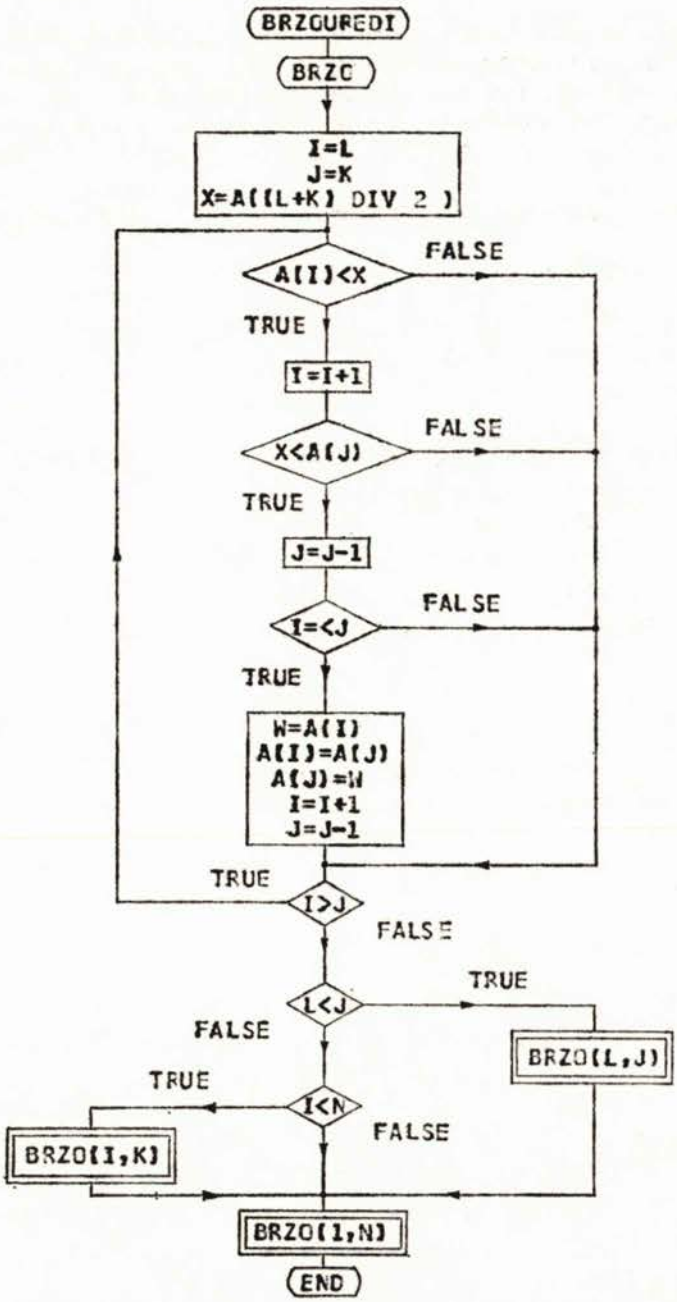
Ovakvi pristupi ocjeni efikasnosti, manje upućenog čitaoca mogu da navedu na pogrešan zaključak, da isključivo strukture podataka i kompleksnost korištenog algoritma utiču na efikasnost programa i cijenu plaćanja korištenja računskih resursa. Zbog toga, cilj ovog rada i jeste da istakne značaj i djelomično ilustruje relevantne faktore koji utiču na efikasnost programa, napisanih u nekom od programskih jezika visokog nivoa.

2. KOMPLEKSNOST ALGORITAMA

Veoma često, za jedan problem postoji više metoda i postupaka koji ga korektno rješavaju. Stoga je u procesu programiranja potrebno izvršiti pažljiv



Sl. 1. Dijagram toka procedure UREDI — algoritam (1)



Sl. 2. Dijagram toka procedure BRZOUREDI — algoritam (2)

izbor metode i nastojati da se u sintezi algoritma dobije algoritam sa što manjom kompleksnošću, jer je kompleksnost algoritma od velikog značaja za ukupno vrijeme izvršavanja programa, odnosno angažovanja centralnog procesora računara.

Idealan primjer za ilustraciju neophodnosti analize algoritama, i njihovih velikih razlika, su algoritmi za uređenje nizova. Pokazaćemo dva algoritma za uređenje nizova, čiji su članovi cjelobroznog tipa i nalaze se u slučajnom poretku. Za opis ovih algoritama, pored dijagrama toka prikazanih na slikama 1. i 2., koristićemo formu procedura i njihovu notaciju u programskom jeziku PASCAL.

Uvedimo tip podataka (čiji je identifikator tipa »indeks«) sljedećom definicijom:

```
  type indeks:0..n;
procedure uredi;
  var i,j,k:indeks; x:integer;
begin
    for i:=1 to n-1 do
      begin k:=i; x:=a[i];
        for j:=i+1 to n do
          if a[j]<x then                                    (1)
            begin k:=j; x:=a[j]
            end
        a[k]:=a[i]; a[i]:=x;
    end
end;
procedure brzouredi;
    procedure brzo(l,k:indeks);
      var i,j:indeks; x,w:integer;
    begin i:=l; j:=k;
        x:=a[(l+k) div 2];
      repeat
            while a[i]<x do i:=i+1;
            while x<a[j] do j:=j-1;
              if i=<j then
              begin w:=a[i]; a[i]:=a[j]; a[j]:=w;           (2)
                    i:=i+1; j:=j-1
              end
      until i>j;
        if l<j then brzo(l,j);
          if i<n then brzo (i,k)
    end;
    begin brzo(1,n)
end;
```

Detaljna analiza kompleksnosti ovih algoritama može se naći u [11], a mi ćemo samo ovdje istaći da algoritmi (1) i (2) spadaju u klase algoritama sa kompleksnošću $c_i n^2$ i $c_j n \log(n)$, respektivno. (c_i i c_j — koeficijenti zavise od konkretnog algoritma; n- broj članova niza).

Očigledno, kompleksnost algoritma (2) je izrazito manja u odnosu na (1), i što je veći niz (veće »n«), razlika u efikasnosti će se povećavati u korist algoritma (2).

Interesantno je vidjeti i kakvi su zahtjevi za angažovanjem centralnog procesora prilikom korištenja prikazanih algoritama, odnosno procedura, u PASCAL-u.

Tabela 1. prikazuje potrebna CPU* vremena i njihov relativni odnos, za uređenje dva niza različite dužine, na računaru CDC 6400.

Tabela 1. CPU vremena (u milisekundama) i njihov relativni odnos

algoritam	broj članova niza			
	216		512	
	CPU v.	r. o.	CPU v.	r. o.
uredi (1)	509	8.5	1956	13.4
brzouredi (2)	60	1	146	1

3. PROGRAMSKI JEZICI

Izbor programskog jezika, kojim će biti opisan određeni algoritam, je od posebnog značaja za efikasnost programa. Primaran kriterij za izbor programskog jezik je priroda (klasa) problema kojeg je potrebno riješiti. Ukoliko je problem numeričkog karaktera, logično je izabrati jedan od jezika namijenjenih za rješavanje naučno-tehničkih problema (FORTRAN, ALGOL, itd.), ili neki iz grupe tzv. univerzalnih jezika (PL/I, PASCAL, itd.). Ovdje treba napomenuti da su, nažalost, kriteriji za ukupnu ocjenu kvaliteta programa kontradiktorni, pa u težnji za efikasnošću obično stradaju druge karakteristike, kao što su: jasnoća i regularnost strukture programa, prihvatljivost za korisnika, fleksibilnost, adaptivnost i t. sl.

Programski jezici se i unutar određene grupe, međusobno razlikuju, kako u konstrukciji, tako i po realizaciji, a osnovne razlike se ogledaju u lakoći, elegantnosti i efikasnosti programiranja pomoću njih. Prilikom izbora jezika potrebno je kritički razmotriti sve njegove prednosti i nedostatke, imajući u vidu sve specifičnosti problema koji se rješava. Ovo vjerovatno autori rada [7] Jovičić D., Lapaine M. i Petrović S. nisu imali u vidu, obzirom da su imali amibicije da kreiraju što efikasniji program.

Jer, većina stručnjaka i teoretičara iz oblasti računarskog programiranja se kritički odnosi prema korištenju BASIC-a za rješavanje naučno-tehničkih problema, s obzirom na njegovu osnovnu namjenu (interaktivni rad), karakteristike, način realizacije, itd. i svrstavaju ga u jezike namijenjene za »amatersku okolinu«. (Vidi npr. [1]).

*Central Processing Unit — centralna procesna jedinica, centralni procesor.

Programski jezici su samo ekvivalentni u potencijalnoj računskoj moći, a razlike u efektivnoj računskoj moći zavise od prirode zadatka i nužno ih je razmatrati u kontekstu određene klase zadataka. ([9]). Detaljnija analiza pojedinih jezika na ovome mjestu ne bi imala punog opravdanja, pa ćemo izložiti samo neke najznačajnije karakteristike i načine realizacije, koji imaju utjecaja na efikasnost.

3.1 *Tipovi i strukture podataka*

Osnovne razlike između programskih jezika se ogledaju u dopuštenim tipovima podataka. Nepostojanje određenih tipova (npr. kompleksni tip i tip analogan FORTRAN-skom »double precision«), mogu u određenim primjenama da predstavljaju ozbiljno ograničenje i nedostatak.

Tipovi podataka i način njihovog strukturiranja, pored toga što mogu direktno da utiču na izbor programskog jezika, utiču i na efikasno (racionalno) korištenje memorijskog prostora. Premda je elementaran način racionalnog korištenja memorije izbor optimalne dužine podataka, u nekim jezicima mogućnosti za izbor su dosta skromne. S druge strane, FORTRAN i PL/I npr. pored razrađenih načina za izbor optimalne dužine podataka, posjeduju mogućnost tzv. višestrukog korištenja memorijskog prostora od strane više promjenljivih ili konstanti.

Pri izboru konkretnih tipova podataka, a naročito ako se želi što racionalnije iskoristiti memorijski prostor, neophodno je poznavati i forme interne reprezentacije podataka, kako bi se pravilno mogle sagledati potrebe za memorijom. U suprotnom, naravno, moguće su greške. (Tako je u [5], zbog uopštenosti i šire mogućnosti primjene korišten kompleksni tip podataka. Autori rada [7], vršeći analizu potrebnog memorijskog prostora za strukture podataka u [5], nisu vodili računa da interna reperezentacija kompleksne promjenljive obične tačnosti, u FORTRAN-u zahtijeva 2 memorijske lokacije (riječi).)

Način strukturiranja podataka također spada u jedan od osnovnih načina efikasnijeg korištenja memorijskog prostora. I dok u nekim jezicima (PASCAL) postoji širok repertoar načina strukturiranja podataka, u drugim je on znatno manji (BASIC, FORTRAN).

Često se javljaju podaci koji su tako »organizovani« da nije moguće izvršiti njihovo adektvatno strukturiranje u odgovarajućem programskom jeziku. U geodeziji, takav slučaj je kod korištenja gornjeg (ili donjeg) trokuta koeficijenata normalnih jednačina. Kako se strukturiranje podataka zasniva na određenim operacijama sa skupovima, odnosno domenima, ni jedna od tih operacija kao svoj rezultat ne daje »trokutasto polje«, pa takva struktura podataka ne postoji ni u jednom programskom jeziku. Tada je potrebno izvršiti određeno preslikavanje takve apstraktne strukture (najčešće) u jednodimenzionalno polje. Iako se u tom slučaju, pisanjem većeg broja instrukcija, povećava angažovanje programera i mogućnost grešaka, značajno se smanjuje potreba za memorijskim prostorom, nego kada se koriste dvodimenzionalna polja. (Vidi [7]).

3.2 *Realizacija programskog jezika*

Fundamentalan princip u projektovanju programskih jezika jeste, da jezik mora biti nezavisan od realizacije. Zbog toga se u literaturi, koja obrađuje korištenje konkretnog jezika, pažnja poklanja uglavnom tipovima i struktu-

rama podataka, operatorima, instrukcijama i načinima njehovog komponovanja, itd., dok se mogući načini realizacije malo ili skoro nikako ne razmatraju.

Korisnik, koji namjerava da studiozno i efikasno koristi određeni jezik, mora da poznaje i metode kojima je on realizovan. U protivnom neće imati realnu sliku o svim prednostima i nedostacima i mogućim uticajima na efikasnost programa. Pored ostalog, trebalo bi da poznaje vrstu jezičkog procesora i njegove značajnije performanse, način provjere kompatibilnosti podataka, vrijeme vezivanja, parametarske mehanizme i slično.

Da bismo istakli značaj naprijed izrečenog, pogledajmo samo uticaj nekih parametarskih mehanizama na angažovanje memorijskog prostora, u programskim jezicima PASCAL i FORTRAN. Sljedeća procedura, odnosno opšti potprogram*, obavlja transponovanje kvadratnih matrica:

```
type matrica=array [1..n,1..n] of real;
    procedure transp (var a:matrica; b:matrica);
        var i,j:1..n;
            begin
                for i:=1 to n do
                    for j:=1 to n do
                        a[i,j]:=b[j,i]
            end;
    end;
```

```
      SUBROUTINE TRANSP (M,B,A)
      DIMENSION A(X,X), B(X,X)
      DO 10 I=1,M
      DO 10 J=1,M
   10 A(I,J)=B(J,I)
      RETURN
      END
```

Izvršimo li transponovanje matrice Q pozivom procedure:

transp (Q,Q)

potreban memorijski prostor (zanemarujući prostor za instrukcije) iznosi $2n^2$ memorijskih lokacija (riječi). Naime, identifikator formalnog parametra »a« će se vezati za memorijsku zonu veličine n^2, alocirainu deklaracijom: var Q:matrica u okviru glavnog;

Identifikator »b« programa se vezuje za memorijsku zonu veličine n^2 memorijskih lokacija, alociranu po pozivu procedure. Nakon izvršenja tijela procedure, ova zona se oslobađa.

* Sintaksa FORTRAN-a zahtijeva da se u okviru instrukcije »DIMENSION«, umjesto simbola »X«, nalazi cijeli broj.

Isti parametarski mehanizam se koristi kod većine jezičkih procesora FORTRAN-a (izuzev oslobađanja zone), pa će i zahtjev za memorijom (za parametre procedure, odnosno potprograma), u vrijeme izvršenja biti isti.

Ukoliko se transponovanje matrice Q izvrši pozivanjem opšteg potprograma:

```
CALL TRANSP (N,Q,Q)
```

i ako jezički procesor koristi parametarski mehanizam »vrijednost-rezultat« (izvorno: »value-result« ili »copy-restore«), potreban memorijski prostor je veličine $3n^2$ memorijskih lokacija. Naime, za samu matricu Q se u okviru glavnog programa alocira zona veličine n^2. Identifikatori formalnih parametara »a« i »b« se vezuju za dvije memorijske zone, svake veličine n^2, inicijalizirane vrijednošću aktuelnog parametra poziva potprograma — Q.

Ovaj jednostavan primjer, pored toga što ilustruje prednosti nekih parametarskih mehanizama u odnosu na memorijske zahtjeve, ukazuje i na raznovrsnost pristupa u realizaciji jednog istog programskog jezika (FORTRAN).

Neosporno je međutim, da u okviru realizacije jezika, u širem smislu, najznačajniju ulogu ima jezički procesor (njegova vrsta i performanse), obzirom da može drastično da utiče na efikasnost programa. U principu, postoje dvije sasvim suprotne vrste jezičkih procesora: prevodioci (kompajleri) i interpreteri. Njihove prednosti i nedostaci su u direktnoj vezi sa brzinom izvršenja i racionalnog korištenja memorijskog prostora. Prevodioci vrše prevođenje izvornog programa (napisanog u nekom od programskih jezika visokog nivoa) u njegovu objektnu formu (mašinski jezik konkretnog računara). Takav način realizacije zahtijeva više memorijskog prostora, ali je brzina izvršavanje izrazito veća, nego kod programske interpretacije. S druge strane, korištenjem interpretera se smanjuju zahtjevi za memorijskim prostorom.

Kako bismo, prilikom izbora programskog jezika i cjelokupne ocjene efikasnosti programa, istakli neophodnost razmatranja načina realizacije i performansi jezičkih procesora, poslužićemo se (prema [4]) analizom implementacije algoritma (1), odnosno procedure »uredi« u programskim jezicima FORTRAN, PASCAL i BASIC. Naredni programski segmenti prikazuju opis tog algoritma* u navedenim jezicima, a komentari uz svaki segment se odnose na vrstu i verziju jezičkog procesora i memorijski zahtjev.

```
C VRSTA JEZIČKOG PROCESORA: KOMPAJLER
C FORTRAN IV V02.04
C POTREBAN MEMORIJSKI PROSTOR: 86 B
  DO 5 I=1, N—1
  DO 5 J=I+1,N
  IF(A(J).GE.A(I)) GO TO 5
  T=A(I)
  A(I)=A(J)
  A(J)=T
5 CONTINUE
```

* Koristise, ustvari, jedna verzija algoritma (1).

{vrsta jezičkog procesora: kompajler}
{OMSI PASCAL V1.2G}
{potreban memorijski prostor: 164 B}

```
for i:=1 to n-1 do
    for j:=i+1 to n do
        if a[j] < a[i] then
            begin
                t:=a[i];
                a[i]:=a[j];
                a[j]:=t
            end
```

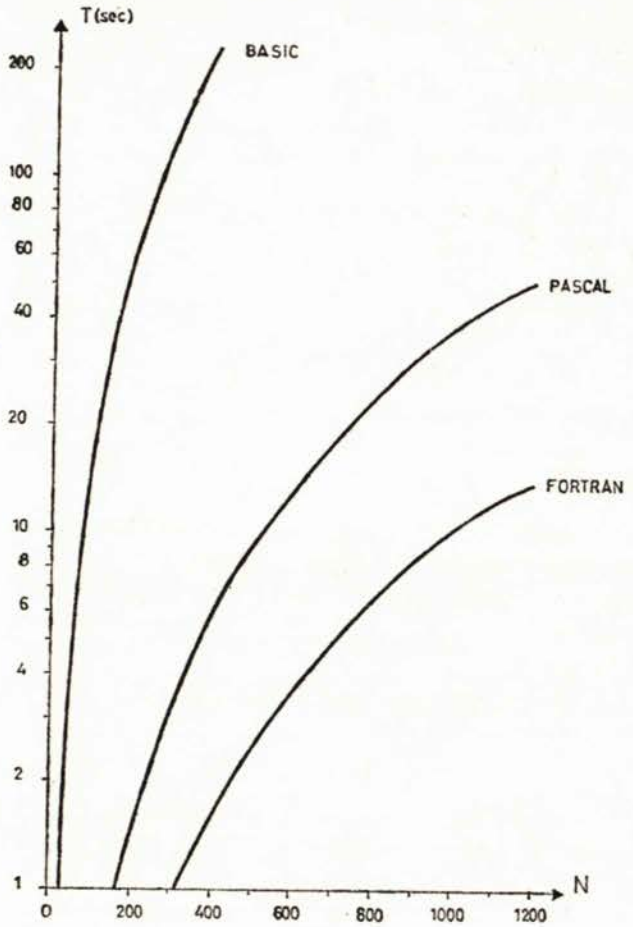
```
10 REM VRSTA JEZIČKOG PROCESORA: INTERPRETER
20 REM MICROSOFT BASIC IN ROM V.1 REV 3.2
30 REM POTREBAN MEMORIJSKI PROSTOR: 86 B
40 FOR I=1 TO N-1
50 FOR J=I+1 TO N
60 IF A(J) < A(I) THEN T=A(I): A(I)=A(J): A(J)=T
70 NEXT J,I
```

Slika (3) prikazuje CPU vremena za uređenje nizova sa »n« cijelih brojeva, koristeći prikazane programe, za računar PDP-11/34 sa opertivnim sistemom RT-11.

Premda je ovaj primjer nedovoljan za jedno opšte rangiranje jezika i njihovih procesora, očigledno je da korištenje jezika, koji se u principu realizuju programskom interpretacijom, jako povećava vrijeme potrebno za izvršenje programa. Također je očigledno da u ovom slučaju, korištenje BASIC-a predstavlja najlošiji mogući izbor, s obzirom da je u tom slučaju potrebno procesorsko vrijeme oko 140 puta (!) veće u odnosu na FORTRAN, odnosno 40 puta u odnosu na PASCAL. Stoga su i moguće, na prvi pogled apsurdne situacije, da je jedan program koji koristi algoritam sa većom kompleksnošću, efikasniji u pogledu vremena izvršenja, u odnosu na neki drugi program, sa algoritmom manje kompleksnosti.

I zahtjevi za memorijom se bitno razlikuju, pa je za program u PASCAL-u potrebno skoro dva puta više memorijskog prostora u odnosu na FORTRAN.

Sve ovo, međutim jasno ukazuje da vrijeme angažovanja centralnog procesora ne zavisi samo od »broja računskih i drugih operacija«, nego da je nužno poznavati i načine realizacije programskih jezika i performanse njihovih procesora, kako bi se pravilno izabrao programski jezik i postigla veća efikasnost konkretnog programa, a tako i izbjegla neopravdano visoka cijena korištenja računarskih resursa.



Sl. 3.

4. RAČUNARI

Razmatranje uticaja karakteristika i performansi računara, odnosno računarskih sistema, i pored toga što ovaj faktor najčešće ne zavisi od programera, zaslužuje pažnju iz više razloga. Navedimo dva, koja smo i do sada imali na umu:

— korisnici, koji plaćaju korištenje računara su zainteresovani za što manje angažovanje procesora i memorijskog prostora,
— prilikom poređenja i ocjene efikasnosti treba imati u vidu i sam računar na kome se programi izvršavaju

Slično, kao i kod programskih jezika, detaljnija analiza svih mogućih načina organizcije i arhitekture računara, performansi i tehnologije njegovih os-

novnih dijelova, i njihov uticaj na efikasnost programa, uveliko bi prevazišla okvire ovakvog rada.

Nesumnjivo je međutim, da svako projektantsko rješenje određenog računara na određen način utiče na efikasnost programa koje taj računar izvršava. (Npr. hardverski rješenja realizacije osnovnih aritmetičkih operacija doprinose bržem izvršavanju programa; način pristupa, vrsta memorijskog medijuma, kapacitet i brzina centralne memorije su također od izuzetnog značaja za efikasnost izvršenja, itd.) Od svih mogućih izvora »računarskih« uticaja, dva su posebno interesantna: arhitektura računara i operativni sistem.

Arhitektura računara, između ostalog, neposredno utiče na složenost i prevođenja (kompilacije), što se opet održava na efikasnost objektnog kôda, odnosno brzinu izvršenja i korištenje memorijskog prostora ([12]. Kod multiprogramskih, odnosno multiprocesorskih sistema, posebna pažnja se pridaje operativnom sistemu. Pored izvanredno važnog uticaja na cjelokupne performanse računarskog sistema, strategija zajedničkog korištenja njegovih resursa, ugrađena u operativni sistem, ima određene reprekusije i na efikasnost svakog pojedinačnog programa. Zbog toga se npr. u [6], kod analize multipleksiranja memorije, pored ostalog kaže: »Najvažnije pravilo dodjeljivanja memorije je da računanjima, da bi se obavljala efikasno, treba dodijeliti dovoljnu količinu centralne memorije. Manje su značajni efikasni algoritmi zamjene i prenosa kojima se smanjuje učestalost i vrijeme čekanja na prenos stranice*. Programer može da doprinese tom smanjenju pažljivim strukturiranjem programa i podataka.«

Zbog ovakvih, za programere uglavnom nevidljivih uticaja, razmatranje i proučavanje uticaja računara, u jednoj detaljnoj analizi, nameće i analizu njihovih operativnih sistema. Svakako treba očekivati, da se kod većih i složenijih programa, ovi uticaji mogu ispoljavati u većoj mjeri. Zaključimo ovo kratko razmatranje jednim primjerom, koji ilustruje uticaj računara na efikasnost (izvršenja i korištenja memorije).

Po programu**, prikazanom u [5], riješen je sistem sa 15 linearnih jednačina, na dva različita računara. Tabela 3. prikazuje angažovanje memorije i CPU vremena (za prevođenje i izvršenje, uključujući ulazno-izlazne operacije), i njihov relativni odnos.

Tabela 2.

računar	prevodilac	operativni sistem	dužina riječi [bit]	vrijeme [sec]	r. o.	memorija [bajt]	r. o.
IBM 1130 model 2B	FORTRAN IV	DMS V2	16	118.8	6.5	3216	0.46
IBM 4331 model J1	FORTRAN IV	DOS/VSE	32	18.3	1.0	7066	1.0

* Jedan od načina upravljanja memorijom. (Vidi [2] ili [6]).

** Nije korišten kompleksni tip podataka, iz razloga navedenog u odjeljku 3.1 ovog rada.

5. ZAKLJUČAK

Pitanje ocjene efikasnosti programa, napisanih u nekom od programskih jezika visokog (višeg) nivoa je kompleksno i ne može se svesti isključivo na analizu kompleksnosti korištenog algoritma, tipova i struktura podataka.

Veoma važan značaj ima programski jezik (performanse njegovog procesora, s obzirom da može drastično da utiče na brzinu izvršenja programa i korištenje memorijskog prostora. Ukoliko postoje ambicije da se kreiraju programi sa visokom efikasnošću, naročitu pažnju treba posvetiti izboru algoritma i programskog jezika. Korištenje »interpretatorskih« jezika (kao što su npr. BASIC i APL), dovodi do velikih zahtjeva za angažovanjem (centralnog) procesora. Korisnici, koji koriste usluge drugih računskih centara, trebaju ovo imati na umu, jer je procesorsko vrijeme (CPU time), u principu izrazito skuplje u odnosu na memorijski prostor.

Pri svemu ovome, ne treba zanemariti uticaje određenih karakteristika i persormansi računara, na kome se programi izvršavaju.

LITERATURA:

[1] Barnes, J. G. P.: An Overview of ADA, Software-Practice and Experience, 1982, 10, 851—887.

[2] Denning, P. J.: Virtual memory, Computing Surveys, 2, 3, 153—189.

[3] Dijkstra, E. W.: A discipline of Programming, Prentice-Hall, Inc., Englewood Clifss, New Jersey 1976.

[4] Dujmović, J. J.: Compiler performance measurement and analysis, Informatica, 1982, 2, 45—54.

[5] Galić, Z.: Program za rješavanje sistema linearnih jednačina metodom Choleskog, Geodetski list, 1983, 1—3, 31—37.

[6] Hansen, P. B.: Operating System Principle, Prentice-Hall, Inc., Englewood Clifs, New Jersey 1973.

[7] Jovičić, D., Lapaine M., Petrović S.: Rješavanje normalnih jednadžbi i metoda Choleskog, Geodetski list, 1984, 1—3, 33—44.

[8] Hoare, C. A. R.: Proof of a recursive program: Quicksort, Computer Journal, 1971, 14, 391—395.

[9] Pritt, T. W.: Programming languages: design and implementation, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1975.

[10] Tennent, R. D.: Principles of programming languages, Prentice-Hall, Toronto 1981.

[11] Wirth, N.: Algorithms + Data structures = Programs, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1976.

[12] Wirth, N.: The Design of a PASCAL Compiler, Software — Practice and Experience, 1971, 1, 309—333.

REZIME

U radu su prikazani i razmotreni glavni faktori koji utiču na efikasnost programa napisanih u nekom od programskih jezika visokog nivoa. Posebno je istaknut značaj i uticaj programskih jezika i njihovih procesora.

ABSTRACT

In this work, the main factors which influence efficiency of the computer programs in high-level programming languages are described and mentioned. Particulary, significance of the programming languages and their processors is emphasized.

Primljeno: 1984-08-24