# PODATKOVNO VODENE RAČUNALNIŠKE ARHITEKTURE

Jurij Šilc, Borut Robič in Branko Mihovilović
Institut 'Jožef Stefan', Ljubljana

UDK: 681.3.014

Računalniške arhitekture lahko razvrstimo glede na podatkovni in krmilni mehanizem. Podatkovni mehanizem določa način, kako se ukazom dodeljujejo argumenti, krmilni mehanizem pa določa vpliv izršitve enega ukaza na izvršitev ostalih ukazov. Glede na podatkovni in krmilni mehanizem razvrstimo računalniške arhitekture v pet kategorij. V podatkovnopretokovnih računalnikih je izvajanje instrukcij podatkovno vodeno; instrukcije postanejo izvršljive takoj, ko so na voljo vrednosti vseh vhodnih argumentov. Podatkovno vodenim arhitekturam pripadajo visoki programirni jeziki z enkratno prireditvijo (Id, Val, Lucid, Valid, SISAL). Materialno opremo podatkovnopretokonih računalnikov sestavljajo procesni, pomnilniški in komunikacijski del, ki so med seboj krožno povezani. Končno je prikazan tudi način, kako lahko uporabimo podatkovno vodeno arhitekturo kot podporo logičnemu programiranju.

DATA-DRIVEN COMPUTER ARHITECTURES - Basically, computer architectures share two fundamental mechanisms, these are data and control mechanisms. The data mechanism defines the way a particular argument is communicated by a number of instructions. The control mechanism defines how one instruction causes the execution of one or more others. In each category of computer, data mechanisms and control mechanisms are largely incompatible sets of alternative concepts. In data flow computer, instruction execution is data driven; the availability of input operands triggers the execution of the instruction that consumes the inputs. Data flow computers are programmed in very high level single-assignment languages (Id, Val, Lucid, Valid, SISAL). The packet communication machine organization is the most obvious for supporting the data flow concepts. It consists of a circular instruction execution pipeline of resources in which processors, communications, and memories are interspersed with 'pools of work' containing different types of information packets. In this paper we also consider a data-driven model for interpreting logic program and investigate the architectural requirements necessary to support its implementation. It will be shown that the model is capable of exploiting the capabilities of highly - parallel dataflow architectures.

## § 1. UVOD

Razvoj računalništva v bodoče je odvisen predvsem od uspešnosti raziskav na naslednjih področjih:

* Umetna inteligenca;
  - metodologije, ki izražajo in povzemajo znanje,
  - uporabniku prirejeni I/O v obliki naravnih jezikov, govora in slik.
* Programirni inženiring;
  - novi visoki programirni jeziki in računski postopki,
  - programska orodja zgrajena na sistemih, kot npr. Unix.
* Računalniške arhitekture;
  - razpršene arhitekture, kot podpora računalniškim mrežam,
  - paralelne arhitekture za zelo hitre računalnike,
  - VLSI arhitekture z največjo izrabo potenciala VLSI tehnologije.
* VLSI tehnologija;
  - VLSI CAD sistemi,
  - nove tehnologije, npr. Ga-As, Josephsonov spoj,...

Tako tehnološko, kakor tudi sociološko smo pripravljeni na novo - peto računalniško generacijo. Računalniki so prešli od znanstveno usmerjenih aplikacij v 50-tih letih, preko komercialno industrijskih aplikacij v 60-tih in 70-tih letih, v inteligentno široko potrošniško elektroniko v 80-tih in 90-tih letih.

## § 2. PREGLED ARHITEKTUR

Sodobne arhitekture imajo za cilj: povečati računalniško moč sodobnih računalniških sistemov, kar pomeni, povečati njihovo zmogljivost in hitrost procesiranja in odpraviti ozka grla v procesu računanja, ki so posledica klasične arhitekture. V novi generaciji splošno namenskih računalnikov bo procesiranje prešlo iz sekvenčnega, centraliziranega sveta v paralelni, decentralizirani svet procesiranja. To je zanesljivo, toda kakšni računalniki bodo prevladovali ni popolnoma jasno. Raziskovalci prisegajo na več mogočih realizacij nove generacije:

* 'Peta generacija' računalnikov;
  - knowledge information processing sistemi,
  - podpora ekspertnim sistemom,
  - uporabniku prilagojeni I/O.
* Superračunalniki;
  - velika izraba paralelizma,
  - uporabnost za obsežna numerična izračunavanja.
* Arhitekture z VLSI (ULSI) procesorji;
  - multi mikroprocesorski sistemi,
  - posebno namenski sistemi z maksimalno izkoriščenim potencialom VLSI tehnologije (1M tranzistorjev/čip).
* Integracija komunikacij in računalnikov;
  - popolnoma integrirana računalniško komunikacijska mreža,
  - velika omrežja, lokalne mreže, paralelne arhitekture.

| KRMILNI MEHANIZEM | PODATKOVNI MEHANIZEM: dodeljevanje pomnilnika | PODATKOVNI MEHANIZEM: prehajanje sporočil |
|---|---|---|
| ukazno vodenje | von Neumannova arhitektura<br>proceduralni jeziki | |
| podatkovno vodenje | | podatkovno pretokovna arhitektura<br>jeziki z enkratno prireditvijo |
| vodenje na zahtevo | redukcijska arhitektura<br>aplikacijski oz. funkcionalni jeziki | |
| vzorčno vodenje | logična arhitektura<br>logični jeziki | objektno orientirana arhitektura<br>objektno usmerjeni jeziki |

Slika 1. - Pregled računalniških arhitektur.

Računalniške arhitekture delimo glede na podatkovni in krmilni mehanizem.

* Podatkovni mehanizem določa način, kako se dodeljujejo ukazom zahtevani argumenti. Ločimo dva osnovna načina:
  - Dodeljevanje pomnilnika; vsakemu ukazu se dodelijo deli pomnilnika, v katerih so argumenti.
  - Prehajanje sporočil; argumenti se prenašajo v obliki sporočila (paketa) od ukaza do ukaza.
* Krmilni mehanizem določa vpliv izvršitve enega ukaza na izvrševanje ostalih ukazov. Ločimo:
  - Ukazno vodenje; ukaz se izvrši, ko je selektiran s krmilno strukturo.
  - Podatkovno vodenje; ukaz se izvrši natanko tedaj, ko so prisotni vsi vhodni argumenti.
  - Vodenje na zahtevo; ukaz se izvrši, ko zazna zahtevo drugega ukaza po njegovem rezultatu.
  - Vzorčno vodenje; vzorec, ki omogoča izvršitev ukaza se primerja z nekim vzorcem in v primeru ujemanja dovoli izvršitev.

Glede na podatkovni in krmilni mehanizem razvrstimo računalniške arhitekture v pet kategorij, katerim pripadajo tudi ustrezni programirni jeziki [1] (slika 1):

* Von Neumannove arhitekture, ki so ukazno vodene in uporabljajo dodeljevanje pomnilnika; pripadajo jim proceduralni jeziki, kot so Basic, Pascal, Fortran, ...
* Redukcijske arhitekture, katerih krmilni mehanizem temelji na načelih vodenja na zahtevo, podatkovni mehanizem pa je bodisi dodeljevanje pomnilnika ali prehajanje sporočil; takšne arhitekture uporabljajo aplikacijske oz. funkcionalne jezike, kot so čisti Lisp, SASL in FP.
* Podatkovno pretokovne arhitekture, ki so podatkovno vodene in imajo dodeljevanje argumentov s prehajanjem sporočil; uporabljajo jezike z enkratno prireditvijo, kakršni so Val, Id, Valid, Lucid, SISAL.
* Objektno orientirane arhitekture, ki so vzorčno vodene in uporabljajo mehanizem prehajanja sporočil; tem arhitekturam pripadajo objektno usmerjeni jeziki, kakršen je npr. Small-talk.
* Logične arhitekture, ki so tudi vzorčno vodene in temeljijo na dodeljevanju pomnilnika; pripadajo jim predikatno logični programirni jeziki, kakršen je Prolog.

Najenostavnejša je von Neumannova arhitektura, kjer natančno opišemo kako dobimo želeni rezultat. Pri kompleksnejših, kot je npr. logična arhitektura, pa navedemo le, kaj je želeni rezultat.

## § 3. PODATKOVNO VODENE ARHITEKTURE

V von Neumannovih arhitekturah je potek izvajanja ukazov (vnaprej) podan eksplicitno. Takšne arhitekture imajo dve značilnosti:

- Prvič, imajo globalni naslovljivi pomnilnik, ki pomni programe ter podatke in katerega vsebina se v skladu s programskimi instrukcijami pogosto ažurira.
- In drugič, vsebujejo programski števec, katerega vsebina je naslov naslednje instrukcije, ki naj se izvrši, torej je s tem določena sekvenca instrukcij, ki se izvajajo (ukazno vodenje).

Takšno vodenje poteka programa je temeljna omejitev teh arhitektur, še posebno pri vzporednem procesiranju.

jezik z enkratno prireditvijo
(Val, Id, Valid, Lucid, SISAL ... )
↓
podatkovno pretokovni programski graf
↓
podatkovno vodena arhitektura
(VLSI imlementacija)

Slika 2. - Na jeziku zasnovana podatkovno vodena arhitektura.

Kakšne so alternative? Ena od mogočih je koncept podatkovnega vodenja, torej podatkovno pretokovna arhitektura. Podatkovno pretokovni računalniki nimajo nobene od prej omenjenih von Neumannovih značilnosti. Njihova struktura je zasnovana tako, da poteka procesiranje na osnovi vrednosti in ne naslovov spremenljivk. Ti sistemi temeljijo na asinhronosti in funkcionalnosti. Vse operacije oz. ukazi so funkcije, ki postanejo izvršljive takoj, ko so na voljo vrednosti vseh vhodnih argumentov. Ukazi, ki postanejo izvršljivi, se lahko izvršijo sočasno [2].

S pravilno izbiro programske (programirni jeziki) in materialne opreme, je mogoče s takšno arhitekturo izkoristiti inherentni paralelizem v največji mogoči meri. Razen maksimalnega izkoriščanja inherentne paralelnosti v algoritmu, imajo računalniki vodeni s pretokom podatkov sposobnost dinamičnega prilagajanja svoje strukture strukturi algoritma. V bistvu smatramo te arhitekture kot arhitekture, zasnovane na jeziku, kjer programski graf - 'strojni jezik' - predstavlja medstopnjo med visokim programirnim jezikom in arhitekturo (slika 2). Od teh arhitektur se zahteva učinkovita implementacija programskega grafa.

### § 3.1. Jeziki z enkratno prireditvijo

Hkrati z izboljšavami von Neumannovih arhitektur so se pojavili tudi nekateri novi visoki programirni jeziki, ki so vsebovali sintaksne konstrukte, s pomočjo katerih so postale izboljšane arhitekturne lastnosti vidne programerju. Večina teh jezikov je nenaravnih v smislu, da v preveliki meri odražajo arhitekturo, manj pa način na katerega mora programer razmišljati pri reševanju problema.

Tako kot ostale oblike paralelnih računalnikov zahtevajo tudi podatkovno pretokovni računalniki (zaradi čimboljše izrabe potencialnih arhitekturnih lastnosti) posebne visoke programirne jezike, t.i. jezike z enkratno prireditvijo ali podatkovno pretokovne jezike. Ti jeziki, za razliko od konvencionalnih jezikov, ne poznajo stranskih učinkov. Naslednja lastnost teh jezikov je lokalnost učinka, kar pomeni, da ukazi nimajo nepotrebnih , daleč segajočih podatkovnih odvisnosti. Nadalje, ti jeziki uporabljajo tudi pravilo o enkratni prireditvi [2].

Jeziki z enkratno prireditvijo imajo v sintaksnem smislu veliko skupnih lastnosti s proceduralnimi jeziki, saj uporabljajo podobne programske konstrukte. Za razliko od proceduralnih jezikov, pri katerih identifikator predstavlja naslovljivi del pomnilnika, pa pri jezikih z enkratno prireditvijo predstavlja povezavo v nekem grafu. Operacija pa predstavlja točko tega grafa. Bistveno pa se ti jeziki razlikujejo po semantiki. Prireditev v proceduralnih jezikih pomeni prenos vrednosti argumentov iz ustreznih delov pomnilnika v registre, izvršitev operacije in prenos rezultatov v ustrezno lokacijo v pomnilnik. Pri jezikih z enkratno prireditvijo pa prireditev pomeni izvršitev operacije, ki se nahaja v točki grafa, nad vhodnimi podatki, ki gredo v to točko in vpis rezultata v izhodno povezavo, ki gre iz te točke grafa.

Oglejmo si sedaj kaj je in zakaj je potrebna enkratna prireditev. Za ilustracijo vzemimo prireditev x := a + b v nekem proceduralnem jeziku. Stavek moramo razumeti kot:
- vzemi vrednosti iz pomnilniških lokacij z naslovoma a in b,
- izvrši operacijo + in
- prenesi rezultat te operacije v pomnilniško lokacijo z naslovom x.

Vidimo, da zgornjemu stavku lahko sledi poljubno mnogo stavkov, ki imajo na levi strani x; stavki pač spreminjajo vsebino lokacije z naslovom x. Sedaj pa si oglejmo sintaktično enako prireditev x := a + b v jeziku z enkratno prireditvijo. Stavek moramo tu razumeti takole:
- obstaja točka grafa, ki je nosilec operacije + in v katero se stekata povezavi z imenoma a in b (v grafu ima lahko samo ena povezava ime a oz. b!),
- obstaja natanko ena povezava z imenom x in to je ravno povezava, ki gre iz te točke,
- po izvrsitvi operacije +, v povezavah a in b ni več vrednosti, v povezavi x pa je vrednost vsote.

če temu prireditvenemu stavku sledi prireditev, ki ima na levi x, npr x := c - d, potem bi morala povezava x izhajati tudi iz točke, ki nosi operacijo - in ima vhodni povezavi c in d. Torej bi povezava potekala iz dveh točk - taki grafi pa niso dopustni.

Primer jezika z enkratno prireditvijo naj nam ilustrira program za izračun integrala pod krivuljo y=x**2, na intervalu [0,1], ki ga izračunamo s pomočjo trapezoidne aproksimacije s konstantnim korakom 0.02; program je zapisan v jeziku SISAL [3] (slika 3).

```
export Integrate
function Integrate (returns real)
for initial
  int := 0.0;
  y := 0.0;
  x := 0.02
while
  x < 1.0
repeat
  int := 0.01 * (old y + y);
  y := old x * old x;
  x := old x + 0.02
returns
  value of sum int
end for
end function
```

Slika 3. - Numerični izračun integrala v jeziku SISAL.

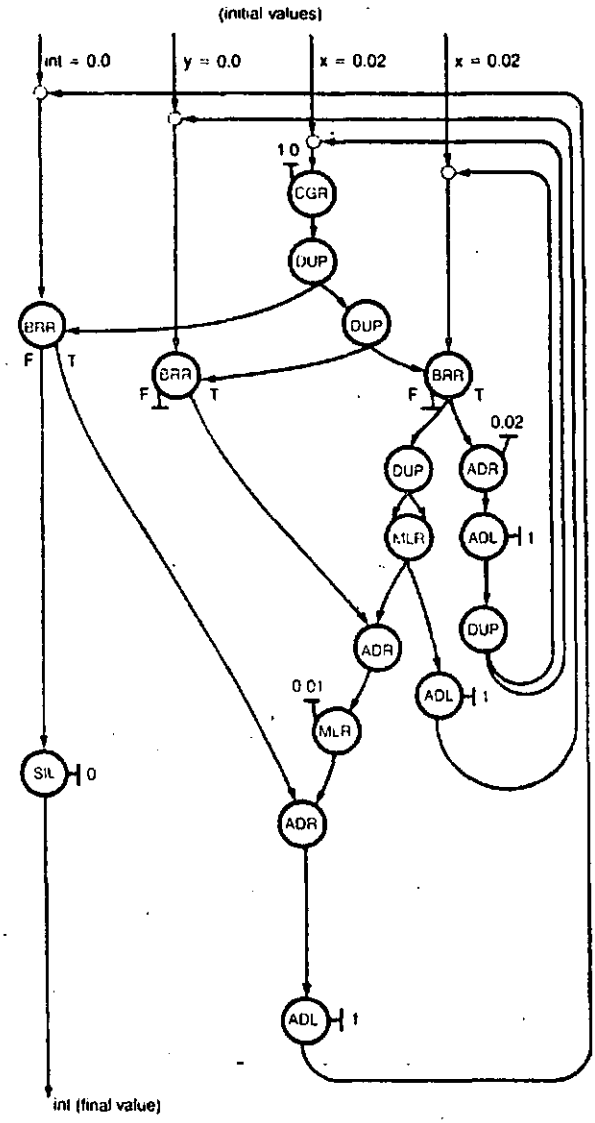
### § 3.2. Programski grafi

Programe krmiljene s pretokom podatkov opisujemo z usmerjenimi grafi, ki jih imenujemo podatkovno pretokovni programski grafi ali krajše programski grafi. Programski grafi so rezultat prevajanja programa zapisanega v enem od visokih podatkovno pretokovnih programirnih jezikov in so torej strojni jezik podatkovno vodenih arhitektur. Prevajanje običajno poteka preko vmesnega zbirnega jezika. Primer takšnega zbirnika prikazuje slika 4.

```
! initialize the loop variables
int = (Data "R 0.0");
x   = (Data "R 0.0");
y   = (Data "R 0.02");
! merge the initial values with the loop
! output values
int_mrg  = (Mer int new_int);
y_mrg    = (Mer y new_y);
x_mrg    = (Mer x new_x);
! test the termination loop
test     = (CGR "R 1.0" x_mrg);
! gate the loop variables into new loop
! instance or direct result to output
gate_int = (BRR int_mrg test);
old_int  = gate_int.R;
old_y    = (BRR y_mrg test).R;
old_x    = (BRR x_mrg test).R;
result   = (SIL gate_int.L "0 0").L;
! loop body: fom new values for loop
!            variables
incr_x   = (ADR old_x "R 0.02");
x_sq     = (MLR old_x old_x);
height_2 = (ADR old_y x_sq);
area     = (MLR "R 0.001" height_2);
cum_area = (ADR old_int area);
!          : increment iterational level
!            for new loop variables
new_int  = (ADL cum_area "I 1").L;
new_y    = (ADL x_sq "I 1").L;
new_x    = (ADL incr_x "I 1").L;
! output the final value of int
           (OPT result "G 0");
```

Slika 4. - Program preveden iz jezika SISAL v zbirnik TASS.

Elementi programskega grafa ponazarjajo pretok podatkov med posameznimi ukazi. Točke grafa ponazarjajo ukaze, usmerjene povezave pa so nosilke vrednosti argumentov ter običajno še dodatnih informacij o delih izračunov katerim pripadajo argumenti. Torej so vhodne povezave nosilke operandov ukaza, ki ga točka predstavlja, izhodne povezave pa hranijo parcialne rezultate. Primer programskega grafa je prikazan na sliki 5.




Slika 5. - Podatkovno pretokovni programski graf.

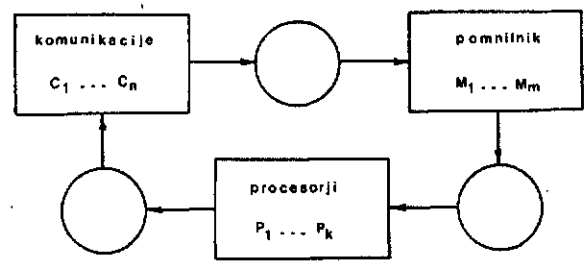
Programski graf je asinhron, kar pomeni, da postanejo ukazi izvršljivi tedaj, ko imajo na voljo vse vhodne operande. Gibanje podatkov skozi programski graf pomeni izvajanje programa. Omeniti velja še eno zelo pomembno lastnost programskih grafov in jasno s tem tudi podatkovno vodenih arhitektur; namreč, ko je operacija oz. ukaz izvršen, vhodni operandi oz. njihove vrednosti, niso več razpoložljivi. Pri cikličnih programskih grafih se pojavijo dodatne težave; v določenih vejah grafa se lahko začno akumulirati vrednosti operandov, kar pomeni, da bi postal graf nedeterminističen. Torej ni možno s prisotnostjo katerekoli vrednosti vhodnega operanda deklarirati točko kot izvršljivo, saj lahko pripadajo operandi popolnoma različnim delom izračuna. Obstaja nekaj rešitev nastalega problema:

* Ciklične programske grafe transformiramo v aciklične. Ta rešitev je zelo nerodna v primeru, ko imamo zanke, katerih pogoj za izstop iz zanke se izračuna šele v času samega izvajanja. V teh primerih je potrebno dinamično generiranje koda.
* V vejah dovoljujemo prisotnost le ene vrednosti. To izvedemo s pomočjo dodatnih povratnih povezav, ki nosijo potrditvene signale, kateri dovoljujejo ustvarjanje novih vrednosti. Z uvedbo teh povezav se promet v grafu podvoji.
* Naslednja rešitev je, da so povezave nosilke paketov, ki poleg vrednosti operandov nosijo še dodatne informacije o delih izračunov, katerim te vrednosti pripadajo. Te dodatne labele običajno imenujemo barve in točka postane izvršljiva, ko so v vseh vhodnih povezavah prisotni paketi enake barve.
* Obstaja še ena možna rešitev; povezave lahko opravljajo funkcijo vrste, kar pomeni, da so vrednosti razvrščene tako, kot so v povezavo prihajale.

Če povzamemo, programski graf je asinhron in determinističen, vse točke imajo pomen funkcij in vrednosti operandov po uporabi niso več razpoložljive [4].

## § 3.3. Materialna oprema

Materialna oprema podatkovno vodenih arhitektur mora biti zasnovana tako, da omogoča učinkovito implementacijo programskega grafa. Tak sistem sestavljajo procesni, pomnilniški in komunikacijski del, ki so med seboj krožno povezani (slika 6).




Slika 6. - Podatkovno vodena arhitektura.

Podatkovno pretokovni računalniki nimajo centralnega procesorja, temveč imajo procesni del, ki ga sestavlja množica nekaj deset, sto ali tisoč procesnih enot. Nimajo niti pomnilnika, kakršnega uporabljajo von Neumannove arhitekture, zato pa imajo pomnilniški del, ki ima potencialno veliko število pomnilniških celic, v katerih se nahajajo vsi podatki o programskem grafu. Za podatkovno pretokovne računalnike je značilno tudi to, da ne uporabljajo sinhronizacijske ure, programskega števca in registrov. Zato pa ima arbitražno vezje, ki usmerja izhode iz celic pomnilniškega dela v ustrezne procesne enote procesnega dela in distribucijsko vezje, ki povezuje procesne enote s pomnilniškimi celicami.

Osnovne ideje podatkovno vodenih arhitektur segajo v pozna šestdeseta leta, ki pa so ob možnosti VLSI implementacije postale ponovno zelo aktualne. VLSI tehnologija, ki omogoča izdelavo mikrovezij s celo 100 pini, je potrebna, saj bi tisoč takšnih vezij, ki bi opravljalo funkcijo usmerjanja in povezovanja, omogočilo uporabo 512 procesorskega podatkovno pretokovnega sistema.

## § 4. LOGIČNO PROGRAMIRANJE NA PODATKOVNO VODENIH ARHITEKTURAH

Logično programiranje je matematični formalizem, s katerim lahko rešujemo probleme na neproceduralni način. Logični programi niso vezani na von Neumannove - sekvenčne arhitekture in so zato primerni za izvajanje na paralelnih arhitekturah. Zato se samo po sebi postavlja vprašanje, kako je mogoče za njihovo izvajanje uporabiti podatkovno vodene arhitekture? S tem bi namreč dosegli, da bi te arhitekture podpirale tudi Prolog, ki je izbran za jezik 'pete' generacije. Odgovor je: narediti prevajalnik, ki bi logični program prevedel v programski graf, to je strojni jezik podatkovno pretokovnega računalnika [5].

Prolog je nepostopkovni ali deklerativni jezik, ki temelji na predikatnem računu prvega reda, ki tudi tvori njegovo sintakso. Omogoča enostavno definiranje relacij med podatki in pa strukturiranje podatkov.

### § 4.1. Logični program

Logični program je množica stavkov, ki imajo obliko

$$p_0 \;:-\; p_1,\ldots,p_m.$$

Vsak $p_i$, imenovan cilj (literal), ima obliko $p(t_1,\ldots,t_n)$, kjer je p predikatni simbol, $t_1,\ldots,t_n$ pa so elementarni podatkovni objekti (termi), ki so lahko konstante, spremenljivke ali funktorji. Privzemimo omejitev, da so vsi predikati binarne oblike, torej $p(t_j,t_k)$. S to omejitvijo ne izgubimo ničesar na splošnosti, saj lahko vsako n-mestno relacijo pretvorimo v n+1 binarnih relacij. Zaradi enostavnosti se omejimo le na elementarne podatkovne oblike tipov konstanta in spremenljivka. $p_0$ se imenuje glava stavka, $p_1$ do $p_m$ pa tvorijo telo stavka. Stavek brez ciljev je enotin stavek, ki izraža eksplicitno dejstvo, to je brezpogojno trditev oz. dejstvo. V primeru programa s slike 7 so stavki, označeni z (1) do (5), ki podajajo relacije "vodja", "raziskovalec" in "programer" med osebami in skupinami, enotini stavki.

```
(1)  vodja(sk1,peter).
(2)  vodja(sk2,gorazd).
(3)  razis(sk1,borut).
(4)  razis(sk1,jure).
(5)  prog(sk2,jure).
                .
                .
                .
(6)  sodel(S,I) :- prog(S,I).
(7)  sodel(S,I) :- razis(S,I).
(8)  nad(I1,I2) :- sodel(S,I1),vodja(S,I2).
                .
                .
                .
(9)  ?- nad(jure,I2).
```

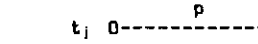
Slika 7. - Primer logičnega programa.

Stavki, ki imajo glavo in telo prinašajo implicitno informacijo in se imenujejo splošni stavki. Takšni so stavki (6) do (8) v primeru s slike 7. Stavka (6) in (7) opisujeta dejstvo, da oseba I, ki je "programer" ali "raziskovalec" skupine S, tudi "sodelavec" te skupine. Stavek (8) pravi, da je oseba I2 v relaciji "vodja" z osebo I1, če je I2 "vodja" skupine S, katere "sodelavec" - "programer" ali "raziskovalec" - je I1.

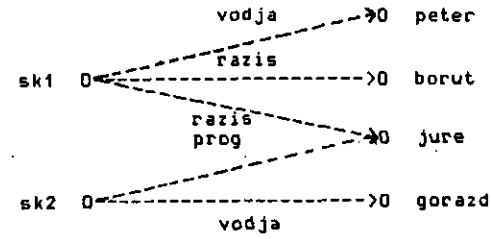
Stavki, ki nimajo glave, so vprašalni stavki. Sistem poišče odgovor nanje tako, da cilj vprašalnega stavka prilagodi na glavo enega izmed splošnih stavkov. Na sliki 7 je vprašalni stavek označen z (9), ki sprašuje "Kdo je Juretu nadrejen?". Cilj tega stavka se prilagodi z glavo stavka (8). Spremenljivka I1 v celem stavku pri tem zavzame vrednost jure. Telo stavka (8) sestavljata cilja sodel(S,jure) in vodja(S,I2). V naslednjem koraku se sodel(S,jure) prilagodi eni od glav stavkov (6) ali (7), npr.(6). Postopek se nadaljuje dokler ni cilj vprašalnega stavka izpolnjen. Kadar cilja ni več mogoče prilagoditi nobeni glavi, pride do vračanja in ponovnega prilagajanja z drugimi glavami ; npr. cilj sodel(S, jure) se prilagodi glavi stavka (7). Če niti prilagajanje niti vračanje ni več mogoče, potem cilja ni mogoče izpolniti.

### § 4.2. Predstavitev logičnega programa s podatkovno pretokovnim grafom

Kot rečeno, logični program sestavljajo dejstva in splošni stavki. Vsako dejstvo vsebuje ime predikata p, ki mu sledita dva elementarna podatkovna objekta $t_j$ in $t_k$. Množici vseh dejstev logičnega programa priredimo graf tako, da predstavimo vsako dejstvo $p(t_j,t_k)$ s podgrafom oblike

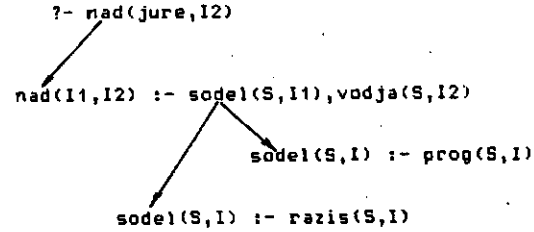



Vozlišča grafa so elementarni podatkovni objekti. Usmerjena povezava iz vozlišča t v t obstaja in nosi oznako p natanko tedaj, ko je p(t ,t ) dejstvo. Dobljeni graf imenujemo graf dejstev. Dejstva opisujejo relacije med elementarnimi podatkovnimi objekti, ki v splošnem niso simetrične. Zato je graf dejstev usmerjen. Za primer s slike 7 je graf dejstev prikazan na sliki 8.



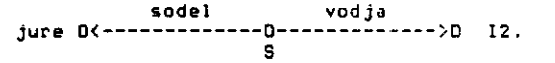

Slika 8. - Graf dejstev.

Splošni stavki, ki imajo glavo in telo, se med seboj povezujejo s kazalci v usmerjeno strukturo na sledeč način: cilj v telesu nekega stavka kaže na vse stavke, katerih glave se z njim ujemajo. Takšno, včasih celo ciklično, strukturo imenujemo strukturo ciljev. Nek cilj L te strukture rešimo tako, da v grafu dejstev poiščemo dejstvo, ki se ujema s tem ciljem ali pa (če to ni možno) rešimo vsaj enega od stavkov, na katere kaže ta cilj. Slika 9 prikazuje strukturo ciljev za primer logičnega programa s slike 7.

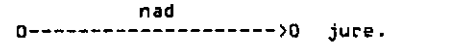



Slika 9. - Struktura ciljev.

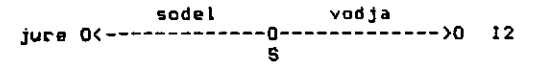
Telesa stavkov lahko predstavimo z grafi na podoben način kot predstavimo dejstva, le da so tu elemetarni podatkovni objekti lahko tudi proste spremenljivke. Tako predstavljene stavke imenujemo graf vzorcev ali kar vzorec. Na primer, v strukturi ciljev na sliki 9 je v stavku v drugi vrsti spremenljivka I1 zaradi prilagoditve na cilj v prvi vrstici vezana na vrednost jure, toda spremenljivki S in I2 sta prosti; temu stavku zato ustreza naslednji vzorec



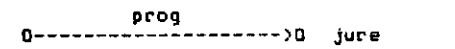

S tem, ko smo logični program prevedli v grafno obliko, postane izvajanje programa ekvivalentno problemu prilagajanja grafa. Potek je naslednji: vzorec se reši tako, da se v grafu dejstev poišče dejstvo, ki se ujema z vzorcem. Npr. vprašalni stavek v našem primeru je vzorec



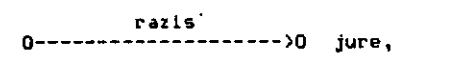

V grafu dejstev poizkusimo poiskati povezavo z oznako "nadrejeni" in točko "jure". Skratka vzorec poizkušamo prilagoditi dejstvu. To prilagajanje ne uspe, ker v strukturi ciljev ni povezave "nadrejeni". Istočasno se pričnе prilagajati tudi vzorec



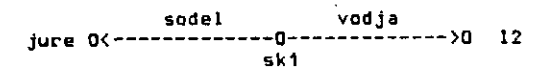

na katerega kaže ciljni vzorec. Cilj "sodelavec" kaže na naslednja dva vzorca



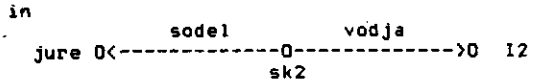

in




ki se tudi sočasno prišneta prilagajati grafu dejstev. Tu prilagajanje uspe. Spremenljivka S se veže na vrednosti "sk1" in "sk2". Po ponovnem prilagajanju vzorcev




in




dobimo iskani rešitvi, ki sta "peter" in "gorazd".

Graf dejstev torej ustreza podatkovno pretokovnemu grafu oz. strojnemu jeziku podatkovno vodene arhitekture. Vsaka točka grafa je logično gledano ativni element, sposoben izmenjave paketov podatkov med točkami. Paketi vsebujejo vzorce (vključno s kazalci), ki se poizkušajo prilagoditi grafu dejstev.

## § 5. ZAKLJUČEK

Večji projekti na področju podakovno vodenih sistemov potekajo:

* Na MIT, kjer sta dve skupini; prva pod vodstvom J. Dennisa razvija sistem z rezinastimi procesorji in druga po vodstvom Arvida, ki gradi VLSI 64 procesorski sistem.
* Na univerzi Utah deluje skupina, ki jo vodi Davis.
* Na CERT v Toulousu dela skupina pod vodstvom J. C. Syra na projektu LAU.
* Na univerzi v Manchestru gradijo eksperimentalni podatkovno vodeni multiprocesorski sistem pod vodstvom J. Gurda in I. Watsuna.
* Na univerzi Tokyo, kjer gradijo računalnik Topstar pod vodstvom T. Suzukija in J. Mot... oke.

Analize učinkovitosti, ki so jih izvršili na realiziranih sistemih so pokazale občutno časovno izboljšanje. Ti sistemi so šele v razvoju, zato obstaja tudi nekaj nerešenih problemov, kot npr. nerešen problem vhodno/izhodnih aktivnosti. Vsekakor pa postajajo podatkovno vodene arhitekture tudi komercialno dosegljive. Pri firmi NEC so izdelali, tako lahko beremo v lanski oktoberski številki revije Computer Design, prvi VLSI čip µPD7281, ki uporablja podatkovno vodeno arhitekturo in je namenjen procesiranju slik [6].

V referatu smo predstavili eno od mogočih arhitektur pete generacije, ki bo morda prevladala na področju superračunalnikov. Takšne arhitekture imajo za cilj povečati računalniško moč sistemov, kar pomeni povečati njihovo zmogljivost in hitrost procesiranja ter odpraviti ozka grla v procesu računanja, ki so posledica von Neumannovega ozkega grla. Večje računalniške zmogljivosti potrebujemo zaradi naraslih potreb pri reševanju problemov na področju umetne inteligence, obdelave slik, razpoznavanja govora, napovedovanja vremenskih situacij, avtomatskega prevajanja jezika, ipd. Obdelava in reševanje takšnih in podobnih problemov s sprejemljivo hitrostjo je mogoča le ob uporabi novih paralelnih - decentraliziranih računalniških arhitektur.

## § 6. LITERATURA

[1] Treleaven P.C., Isabel Gouveira Lima: 'Future Computers: Logic, Data Flow, ..., Control Flow?', Computer, Vol.17, No.4, Mar. 1984, pp.47-57

[2] Computer, Special Issue on Data Flow Systems, Vol.15, No.2, Feb. 1982

[3] Gurd J.R., C.C. Kirkham & I. Watson: 'The Manchester Prototype Dataflow Computer', Comm. of the ACM, Vol.28, No.1, Jan. 1985, pp.34-52

[4] Silc J., B. Robič: 'Osnovna načela DF sistemov', Informatica, št.2, 1985, str.10-15

[5] Bic L.: 'Execution of Logic Programs on a Dataflow Architecture', The 11th Annual International Symposium on Computer Architecture, Ann Arbor, Michigan, June 1984, pp.290-296

[6] Chong Y. M.: 'Data Flow Chip Optimizes Image Processing', Computer Design, Oct.15 1984, pp.97-103