UDK 681.324

Branko Mihovilovič, Peter Kolbezen, Jurij Šilc
Institut »Jožef Stefan«, Ljubljana

Komunikacijske povezave med transputerji predstavljajo zaradi cenenosti, enostavnosti in preproste uporabe nov standard v povezovanju računalnikov. V prispevku so na kratko opisani occamski konstrukti, ki so osnova pri načrtovanju transputerskih sistemov. Na primeru smo pokazali kako pomembno vlogo igrajo komunikacijski procesi pri načrtovanju transputerskih sistemov, ter kako pomembna je njihova preureditev v programu, če želimo varno in v polni meri izkoristiti sočasno izvrševanje opravil v sistemu.

Communicating processes in transputer systems - The transputer communication link is a new standard for computer system interconnection, since it is a cheap, simple and easy-to-use. This paper describes the occam constructs that are fundamental for transputer systems design. Their use is given by an example where some program transformations are performed for the sake of achieving the advantages of concurrent operating transpur system.

1. Uvod

Transputerji oziroma transputerski sistemi temeljijo na krmilno vodenem računanju in jih uvrščamo v skupino arhitektur, katerih model računanja je bodisi zaporedni ali sočasni krmilni tok [1]. Ti sistemi dovolj pogumno in verjetno tudi uspešno začrtujejo novo smer v razvoju računalnikov pete generacije. Organizacija stroja je sicer centralizirana, toda izrabljajoč VLSI tehnologijo je mikro računalnik s pomnilnikom, procesorjem in komunikacijskim delom narejen kot ena komponenta (čip). S preprostim povezovanjem takšnih komponent pa je možno zgraditi visoko sposobne računalniške sisteme.

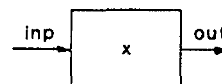
Vzporedno z razvojem materialne opreme, se je pojavila potreba po visokem programskem jeziku, ki pa ne bi bil namenjen samo programiranju računalniškega sistema, temveč bi bil primeren tudi za njegovo načrtovanje. Takšne zahteve danes izpolnjuje samo jezik OCCAM. Jezik je enostaven, sloni pa na dveh konceptih: sočasnosti in komunikaciji. Uporablja se enako dobro za opisovanje tako sistema medseboj povezanih mikroračunalnikov, kot za programiranje enega mikroračunalnika.

Semantično gledano, imamo na voljo množico pravil, ki nam služijo pri ti. transformaciji, oziroma optimizaciji programa. Pri slednji načrtovalec eksperimentalno izbira med različnimi implementacijami programa, ki so enakovredne. S tem doseže najboljše lastnosti načrtovanega sistema v pogledu sočasnosti, velikosti sistema in podobno.

V OCCAM modelu je proces zaporedje akcij, ki jih imenujemo osnovni procesi. Ločimo tri osnovne procese. Vhodni in izhodni proces skrbita za prenos vrednosti spremenljivk preko occamskih kanalov (c ? x, c ! x), s pomočjo prireditvenega procesa pa spreminjamo vrednosti spremenljivk (v := e). Osnovne procese kombiniramo tako, da se le-ti v konstruktih lahko izvajajo sekvenčno, paralelno ali pa se v množici procesov izvaja tisti proces, ki je prvi pripravljen.

2. Sekvenčni konstrukt

Element



opravlja dva sekvenčna procesa - vhodni in izhodni. Konstrukt

P = (μX.inp ? x -> out ! x -> X)

zagotavlja, da se bo vhodni proces končal prej, ko bo začel izhodni proces. V occamu zapisan program ima naslednjo obliko:

```
WHILE TRUE
VAR x:
SEQ
  inp ? x
  out ! x
```

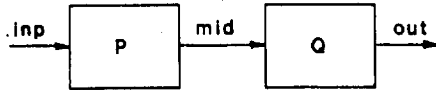
Tu velja opozoriti, da imamo v occamu na voljo WHILE stavek za opisovanje ponavljajočih procesov. To je povsem razumljivo, če vemo, da GO TO in podobni stavki ne sodijo v družino stavkov s katerimi popisujemo paralelne procese.

3. Paralelni konstrukt

Paralelni konstrukt

P = (μX.inp ? x -> mid ! (a*x) -> X)
Q = (μY.mid ? y -> out ! (b*y) -> Y)
R = P || Q

/ponazorjen z naslednjim elementom



povezuje sekvenčne procese. Vsak proces uporablja svoje spremenljivke, med seboj pa procesa komunicirata preko komunikacijskega kanala. V occamu zapisan program ima naslednjo obliko:

```
CHAN mid:
PAR
  WHILE TRUE
  VAR x, a, r:
  SEQ
    inp ? x
    r := a*x
    mid ! r
  WHILE TRUE
  VAR y, b, p:
  SEQ
    mid ? y
    p := b*y
    out ! p
```

Vobče sta procesa P in Q paralelna, ni pa nujno, da tečeta sočasno. Sočasen proces uporablja svoje spremenljivke, katerih ne more dodeljevati drugemu sočasnemu procesu. Sočasna procesa lahko komunicirata samo preko kanalov. Po enem kanalu poteka samo ena enosmerna, sinhronizirana komunikacija med dvema procesoma. V kanalski komunikaciji ne poznamo vmesnih pomnilnikov (bufferiranja).

4. Sočasni procesi

Bodita dana procesa P in Q, ki imata obliko:

P = (μX.inp ? x -> outp ! x -> X)
Q = (μY.inp ? y -> outp ! y -> Y).

Poglejmo če obstaja možnost sočasnega izvajanja teh dveh procesov. Ta možnost je dana, saj lahko vhodni in izhodni komunikacijski proces tečeta sočasno: inp? x in outp! y, ter outp! x in inp? y. Vendar po pravilih o sočasnih procesih obstaja med sočasnima procesoma potencialno dead lock stanje, kar pomeni, da bi tudi procesa P in Q zašla v to stanje. Da preprečimo pojav dead lock stanja, moramo vsaj v enem paru sočasnih procesov en notranji proces začeti izvrševati predno se začne časovno prekrivanje procesnega para (premaknitev procesa po levi strani časovne osi). Zapišimo to še enkrat:

$$(a \to P) \underset{c}{\|} (b \to Q) = (\text{dead-lock})$$

$$(a \to P) \underset{c}{\|} (c \to d \to Q) = c \to ((a \to P) \underset{c}{\|} (d \to Q)).$$

Zgornji izrazi nam povedo, da lahko nek proces delimo (če je to mogoče) na podprocese, ki (če so med seboj povezani s kanali) tečejo sočasno z ostalimi podprocesi (tudi s tistimi, ki pripadajo drugemu procesu). Temu pravimo tudi dekompozicija procesa. Želimo, da je dekompozicija hierarhična (varovanje pred dead lock stanji), da omogoča uporabo paralelnih konstruktov in da so pri delitvi tisti končni procesi kar se da preprosti. Tako lahko ustvarimo programe, ki uporabljajo visoko stopnjo sočasnosti; vsak enostaven proces pa se izvaja na lastnem procesorju, ki je lahko enostaven zato pa izredno hiter (RISC).

V sekvenčnem procesu W

W = (μX.inp?x -> out!x -> inp?y -> out!y -> X)

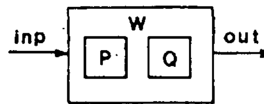
poiščemo tiste podprocese, ki se lahko izvršujejo sočasno (dekompozicija). Najdemo dva para sočasnih procesov

$$P = P1 \underset{c}{\|} P2 = (\text{inp ? x} \to \text{out ! y} \to P)$$

$$Q = Q1 \underset{c}{\|} Q2 = (\text{out ! y} \to \text{inp ? x} \to Q)$$

in dobimo

W = inp ? y -> (μX.P -> Q -> X)



5. Primer

Oglejmo si iterativno polje (kvadratna mreža transputerjev), ki je načrtano tako, da z njim lahko izračunamo m skalarnih produktov vektorjev v in w_i, $1 \le i \le m$. Na primeru bomo razložili nekatere posebnosti komunikacij med procesi oziroma transputerji v takšnem in podobnih sistemih. Naš primer zapišemo z naslednjim izrazom:

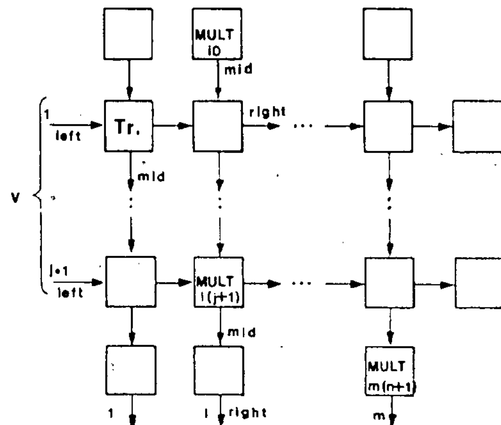
$$s_i = \sum_{j=0}^{n-1} w_{ij} \times v_j \quad .$$

Primer torej rešimo z iterativnim poljem transputerjev, ki ga ponazarja slika 1. Vidimo, da gre v bistvu za izračun i skalarnih produktov med enim vhodnim vektorjem v in m vektorji w_i, katerih komponente (matrika W) so zapisane v posameznih transputerjih. Zapišimo sekvenco procesov, ki jih izvajajo posamezni transputerji v iterativnem polju:

$$MULT_{i0} = (\mu X.P_0 \to X)$$

$$MULT_{i(j+1)} = (\mu X.P_1 \to P_2 \to P_3 \to P_4 \to P_5 \to X)$$

$$MULT_{m(n+1)} = (\mu X.P_6 \to P_7 \to X)$$



slika 1

kjer so

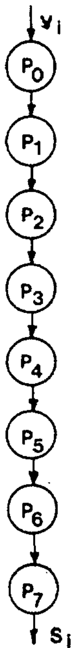
$P_0 = mid_0 \,!\, 0$ $\qquad$ $P_1 = left_i \,?\, v$

$P_2 = mid_{ij} \,?\, y$ $\qquad$ $P_3 = z := (w_{ij} * v) + y$

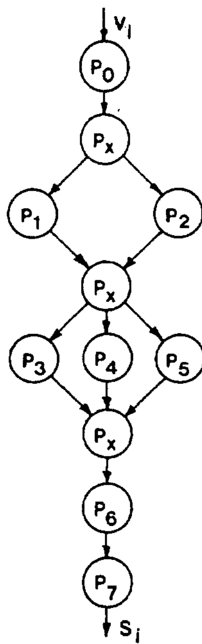
$P_4 = mid_{i(j+1)} \,!\, z$ $\qquad$ $P_5 = right_j \,!\, v$

$P_6 = mid_{mn} \,?\, x$ $\qquad$ $P_7 = right_{i(n+1)} \,!\, s$

Na sliki 2a je podan preprost grafni model programa za eno transputersko celico. Postavlja se vprašanje, kako v najboljši meri izkoristiti prednosti iterativnih polj, tj. možnosti pohitritev pri reševanju takšnih in podobnih problemov. Ali povedano drugače, zagotoviti, da kar največ celic lahko deluje sočasno.



slika 2a



slika 2b

V našem primeru najdemo dve skupini procesov, ki se lahko izvršujejo sočasno. V prvi skupini sta procesa P1 in P2, v drugi pa so procesi P3, P4 in P5. Grafni model na sliki 2b prikazuje takšno strukturo sočasnih procesov. Procesi Px združujejo v sebi vse potrebne nadzorne strukture paralelnih oziroma sočasnih procesov (sinhronizacija). V occamu zapisan program za n x m celic ima sedaj naslednjo obliko:

```
PROC MULT (CHAN mid, left, right)
   VAR y, v, m, z
   SEQ i = [1 FOR m]
     SEQ
       PAR
         left ? v
         mid ? y
       PAR
         z := W * v + y
         mid ! z
         right ! v
```

Z resnejšim premislekom ob sliki 2b in zgornjem programu ugotovimo, da lahko le v n transputerjih tečejo procesi sočasno in da je zakasnitev pri izvajanju procesa P5 kriva za to, da procesi v ostalih m-1 verigah transputerjev ne tečejo sočasno (m skalarnih produktov). Slednje bi bilo možno ob pogoju, da procesa P1 in P5 tečeta sočasno, vendar na enem transputerju takšna procesa ne moreta teči sočasno. Rešitev je v časovnem zamiku med sekvenčnima procesoma vsaj za en obhod zanke. Takšno transformacijo bi zapisali v naslednji obliki:

```
SEQ i=[0 FOR m]            SEQ
  SEQ                        P(0)
    P(i)          =>         SEQ i=[0 FOR m-1]
    Q(i)                       SEQ
                                 Q(i)
                                 P(i+1)
                             Q(m)
```

Occamski program z zgornjo transformacijo dobi naslednjo obliko:

```
PROC MULT (CHAN mid, left, right)
     VAR y, v, m, z
     SEQ
       left ? v
       mid ? y
       SEQ i = [ 1 FOR m - 1 ]
         SEQ
           PAR
             z := M * v + y
             mid ! z
             right ! v
           PAR
             left ? v
             mid ? y
       PAR
         z := m * v + y
         mid ! z
         right ! v
```

Hkrati s transformacijo smo rešili tudi problem pojava dead-lock stanja pri sočasnem izvajanju procesov P4 in P5. Če pogledamo zančni del zgornjega programa ugotovimo, da imamo zaporedje dveh izhodnih in dveh vhodnih komunikacijskih procesov. V takšnem zaporedju sočasnih procesov kaj hitro pride do sinhronizacijskih težav in s tem do dead-lock stanja. Težavam se izognemo tako, da spremenimo ime spremenljivke vhodnima komunikacijskima procesoma in dodamo prireditvena procesa.

```
                             ---
                             PAR
---                            SEQ
PAR                              left ? a
  left ? v        =>             v := a
  mid ? y                       SEQ
---                              mid ? b
                                 y := b
                              ---
```

Prireditvena procesa želimo ločiti od komunikacijskih procesov. To bomo storili z naslednjo transformacijo. Označimo s C1 in C2 komunikacijska procesa, s P in Q pa prireditvena procesa. Glede na to, da procesa C1 in C2 ne začneta istočasno lahko zapišemo:

$$(C1 \rightarrow P) \parallel_C (C2 \rightarrow Q) = ((C1 \rightarrow P) \parallel_C C2) \rightarrow Q =$$

$$(C2 \parallel_C (C1 \rightarrow P)) \rightarrow Q = ((C2 \parallel_C C1) \rightarrow P) \rightarrow Q =$$

$$(C2 \parallel_C C1) \rightarrow P \rightarrow Q = (C1 \parallel_C C2) \rightarrow P \rightarrow Q$$

Če upoštevamo zgornjo transformacijo, dobi naš program končno obliko:

```
PROC MULT (CHAN mid, left, right)
   VAR y, v, m, z, a, b
   SEQ
     left ? v
     mid ? y
     SEQ i = [1 FOR m - 1]
       SEQ
         PAR
           z := m * v + y
           mid ! z
           right ! v
         PAR
           left ? a
           mid ? b
       v := a
       y := b
     PAR
       z := m * v + y
       mid ! z
       right ! v
```

Procesiranje podatkov v zadnjih dveh programih je popolnoma enako, razlika pa je v tem, da v zadnjem programu ne more priti do izpisov rezultata (mid ! z), predno se ne izvedeta vhodna sočasna procesa (left ? a, mid ? b).

6. Zaključek

Nedvomno je jezik occam v primerjavi s sorodnimi jeziki najprimernejši tako za opisovanje strukture, kot za programiranje transputerskega sistema in njegovih komponent. Na primeru smo pokazali pomen podrobnejše analize programskega dela transputerskega sistema; posledica neustrezne časovne razporeditve procesov je lahko neučinkovitost celotnega sistema.

Komunikacijski procesi igrajo pomembno vlogo pri izbiri tistih procesov, ki jih lahko združimo v sisteme sočasnih procesov. Pokazali smo, da lahko sočasnosti formiramo na dveh nivojih:

a) kjer več transputerjev deluje sočasno, ter

b) kjer na enem transputerju sočasno poteka več procesov.

7. Literatura

[1] Borut Robič, Jurij Šilc, Razvrstitev novogeneracijskih računalniških arhitektur, Informatica **10** (4) (1986) 18-32.

[2] Branko Mihovilović, Slavko Mavrič, Peter Kolbezen, Transputer - osnovni gradnik večprocesorskih sistemov, Informatica **10** (4) (1986) 81-84.

[3] C.A.R. Hoare, Communicating sequential processes, Prentice-Hall International (1985).

[4] Jane Curry, Occam solves classical operating system problems, Microprocessors and Microsystems **8** (6) (1984) 280-283.

[5] David May, Richard Taylor, Occam - an overview, Microprocessors and Microsystems **8** (2) (1984) 73-79.

[6] Richard Taylor, Transputer communication link, Microprocessor and Microsystems **10** (4) (1986) 211-215.

[7] David May, Roger Shepherd, The transputer implementation of occam, Proc. Int'l Conf. Fifth Generation Computer Systems 1984, OHM North-Holland (1984) 533-541.