HIERARHIČNI VEČPROCESORSKI SISTEM

INFORMATICA 1/91

Keywords: Parallel processor system, Token data flow, Transputer, Topology, Communication message, Static scheduling, Processor allocation, Speedup.

Peter Kolbezen,
Institut Jožef Stefan, Ljubljana
Peter Zaveršek, Gorenje Servis,
Velenje

POVZETEK. V članku je predstavljen večnivojski rekonfigurabilni večtransputerski sistem. Izdelan je koncept izmenjave sporočil med transputerji na nivoju procesiranja uporabniških programov. Nivo procesiranja elementarnih uporabniških programov je mogoče s pomočjo spremenljive topologije povezav med transputerji povezati v hierarhično vgnezdene zanke. Razvrščanje procesov je prilagojeno tej topologiji in je zasnovano na matematičnem zapisu hierarhičnega acikličnega grafa pretoka podatkov. Predhodna analiza uporabniškega programa na sistemskem nivoju računalnika omogoča odkriti sočasne procese, določiti aciklične dele programa in jih transformirati v ekvivalentne dele programa, ki so predstavljivi s hierarhičnimi grafi pretoka podatkov. Komunikacija med transputerji, ki so povezani v zankah, in avtomatično dodeljevanje procesov potekata asinhrono s podajanjem žetonov. Predlagana topologija omogoča časovno optimirano razvrščanje, ki je pogojeno z usklajenostjo topologije in hierahičnimi acikličnimi grafi pretoka podatkov.

A HIERARCHICAL MULTIMICROPROCESSOR SYSTEM. The architecture of a multiprocessor transputer system and a communications shell for a multiple ring topology network of transputers for efficient execution of parallel programs are proposed in the paper. The system consists of three layers dedicated to elementary user program process execution, global control in programs and system functions. The elementary user program process execution layer network ring topology is hierarchically arranged. The process allocation is dinamically adjusted to the topology of intreprocess communications. It is based on the analysis of user program and on the description of the program with the hierarchical acyclic data flow graph (HADFG). The prime requirement is that any processor in the network must be capable of passing a message to its successor, with the communications shell automatically. The communications shell passes each message packet through the ring network asynchronously. Proposed topology enables time-optimized scheduling by harmony of the network topology.

1. UVOD

Procesiranje v realnem času na področjih, kot so vodenje zahtevnih procesov, robotizacija, znanstveno - raziskovalno delo, obdelava slik in signalov ter podobno, zahteva vse večje računalniške zmogljivosti, ki jih s klasičnimi računalniki ni več mogoče doseči.

Znano je, da je edina pot, ki vodi k večjim zmogljivostim, možna preko bistveno drugačnih računalniških organizacij, kot so se uporabljale v preteklosti. Med uspešnejše organizacije sodijo predvsem takšne, ki izkoriščajo paralelizem. Zato zasluži paralelno računanje danes še posebno pozornost. Že sorazmeroma cenena in hitra tehnologija zelo visoke integracije (VLSI) omogoča učinkovitejše računalniško podprto načrtovanje in uporabo paralelnih VLSI računalnikov. Zato so dosežki v mikroelektronski tehnologiji, in bodo še bolj v bodoče, pobudniki številnih inovacij v paralelnih računalniških arhitekturah in v uporabi t.im. polj procesorjev. V svetu je ta trend deležen velike pozornosti v vladnih, industrijskih in univerzitetnih okoljih, saj je v zadnjem desetletju zaslediti viden vzpon raziskav in vlaganj v razvoj tovrstne računalniške tehnologije.

2. PARALELNO PROCESIRANJE

Med najbolj preprostimi in najbolj obetavnimi tipi paralelnih računalnikov je dobro poznana večprocesorska arhitektura. Od vseh tehnik, ki večajo moč procesiranja, je paralelno procesiranje v pogledu načrtovanja aparaturne opreme ena od najpreprostejših tehnik, hkrati pa tudi najbolj izzivalna v pogledu načrtovanja programske opreme.

Paralelizem je lahko drobno ali grobo zrnat. Prvi se pojavlja pri vektorskih procesorjih, drugi pa v računalniških mrežah. Obstaja pa tudi srednje zrnat paralelizem, kakršnega srečujemo v transputerskih sistemih.

Večprocesorskimi sistemi imajo običajno centralni mikroprocesor, obdan s procesorji, pomnilniki in

komunikacijskimi potmi. Ti viri se centralno dodeljujejo in preko njih potekajo komunikacije med procesi. V sklop sistema mora biti vključena tudi posebna aparaturna oprema, ki odloča o večkratnih sočasnih zahtevah za dostop do dodeljenih virov, in programska oprema, ki mora pri dodeljevanju virov zagotavljati vzajemno izključevanje izvajanj kritičnih področij programa, v katerih se viri uporabljajo. Tako se v zvezi z uporabo večprocesorskih sistemov pojavlja vrsta problemov, ki so težje narave. Znano je, da sta ob večjem številu procesorjev večja tudi cena upravljanja in zmogljivost sistema. Ta odvisnost je pri manjšem številu procesorjev skoraj linearna in spočetka dopušča tudi dovolj veliko prepustnost sistema. Z nadaljnim večanjem števila procesov pa prične prepustnost sistema kljub večanju števila procesorjev vse hitreje upadati. Zato je zmogljivost takega sistema omejena.

V večprocesorskih sistemih, ki so osnovani na transputerski tehnologiji, se dodeljevanje virov bistveno poenostavi. Problemi spornih točk in vzajemnega izključevanja skoraj v celoti odpadejo. Komunikacije med procesorji in procesi zamenjajo medprocesorske zanke. Zlasti pa je pomembno, da v primerjavi s klasičnimi sistemi prepustnost transputerskih sistemov neomejeno narašča skoraj sorazmerno s številom transputerjev. Komunikacija med procesorji poteka zaporedno preko dveh enosmernih linij na asinhroni način. Vsaka zanka predstavlja enosmerni kanal, preko katerega dva procesa med seboj komunicirata. S takšno komunikacijo je omogočena tudi sinhronizacija med posameznimi procesi. Transputerski večprocesorski sistem predstavlja torej skupek procesorskih vozlišč s ponori in izvori, ki si v naključnostnih intervalih izmenjujejo sporočila med seboj.

Dasi razvoj in gradnja večprocesorskih sistemov napredujeta z dramatično naglico, to ne velja za razvoj dovolj učinkovitih postopkov programiranja. Tudi programiranje na nivoju srednje zrnatosti je še vedno povezano s prekomernim naporom in često zahteva povsem nove postopke. Prenašanje sporočil v transputerskem sistemu ne predstavlja tolikšnih režijskih stroškov kot sistem, ki zahteva dekompozicijo programa v drobno zrnat program. Po drugi strani pa program ne sme biti preveč grobo zrnat, saj kot tak preprečuje, da se njegovo izvajanje porazdeli na pametno število procesorjev.

3. PROGRAMIRANJE VEČPROCESORSKIH SISTEMOV

Programer se pri programiranju večprocesorskih sistemov sooča s tremi glavnimi problemi:

1. Kako v danem programu ugotoviti dele programa, ki se lahko izvajajo sočasno, in kako originalni program modificirati tako, da se zmanjša ali čas računanja ali število potrebnih virov?

2. Kako dodeljevati zgoraj omenjene različne dele kode v večprocesorskem sistemu?

3. Kako za dano arhitekturo določiti optimalno število procesorjev, ki so potrebni za kar se da učinkovito izvajanje programa?

Pričujoče delo je posvečeno predvsem vprašanju dodeljevanja procesov neregularnih algoritmov na večnivojskem transputerskem sistemu. Za reševanje problemov iz tč.1 je potrebno orodje, ki omogoča skrbno analizo in dekompozicijo programa za paralelno procesiranje na večtransputerskem sistemu. Sistem naj bi programerju dopuščal uporabo dovolj splošnih metod za reševanje zgoraj omenjenih problemov.

Program, ki naj bi se kolikor je mogoče optimalno izvajal na večprocesorski arhitekturi tako, da bi izkoriščal ves možni paralelizem, mora biti predhodno skrbno analiziran in razstavljen na množico procesov. Seveda je eden prvih pogojev za paralelno procesiranje, da program vsebuje dovolj paralelnih poti, ki so programerju pogosto tudi prikrite. Pristop k reševanju problema določanja sočasnosti pa lahko poteka po dveh poteh. Po prvi poti je določanje sočasnosti neposredno prepuščeno programerju. Znano je namreč, kako določi programer sočasnost pri pisanju programov v jezikih, ki podpirajo paralelno programiranje. Po drugi poti pa je določanje sočasnosti implicitno, neodvisno od programerja na osnovi analize izvornega programa. Pri tem je odločilnega pomena podatkovna odvisnost med procesi. Ker so sekvenčni programi razbiti na različne nivoje, je osnovni korak k paralelnemu procesiranju razpoznavanje procesov znotraj posameznih nivojev, predvsem takšnih, ki se lahko že po svoji naravi izvajajo paralelno, pa tudi takšnih, katerih paralelnost je več ali manj prikrita programerju. Vsekakor je druga pot zanimiva tudi v primeru, da programer v prvi fazi programira v paralelnem programskem jeziku, v drugi fazi pa izdelani program še dodatno časovno analizira in ga na osnovi te analize optimira in testira.

4. TRANSPUTERSKI VEČPROCESORSKI SISTEM

Sistem je zasnovan na podobnem konceptu, kakršnega sta predlagala M.Szturmowicz in M.Tudruj v delu /1/. Vsebuje tri nivoje transputerskega vezja: krmilni, sistemski in delovni nivo. Karakteristike načrtovanega sistema so:

- hierarhična struktura, ki je zgrajena iz več procesorskih nivojev, od katerih je vsak namenjen izvajanju posebne vrste opravil
- namenska komunikacijska struktura na vsakem nivoju procesorjev
- rekonfigurabilne procesorske povezave delovnega nivoja, ki omogočajo dinamično preslikavo tekočega programa na polje procesorjev z ustreznimi komunikacijskimi povezavami
- procesiranje podatkov, globalno krmiljenje v programih in izvajanje sistemskih funkcij poteka paralelno

- transputerji imajo možnost medsebojnega povezavanja s pomočjo posebne materialne opreme (C004)
- med seboj neodvisni procesi istega nivoja se izvajajo sočasno
- sistem je učinkovit za srednje zrnato paralelno procesiranje
- omogoča tudi dovolj učinkovito drobno zrnato procesiranje

Procesorsko polje na delovnem nivoju je razdeljeno v grupe (t.im. grozde) transputerjev. Transputerje v vsaki grupi je mogoče povezati v mrežo poljubne topologije. Sistem med izvajanjem nekega posla, ki je sestavljen tako iz regularnih kot neregularnih postopkov, ustrezno prilagaja polje procesorjev vsakokratnim potrebam izvajanja z rekonfiguracijo posameznih grup procesorjev delovnega nivoja. Tako je lahko v nekem trenutku izvajanja ena grupa procesorjev povezana v hiper-kocko, na kateri se izvaja nek acikličen paralelni postopek, druga grupa pa v valovnofrontno polje (wavefront array), na katerem se izvaja nek matrični račun /8/.

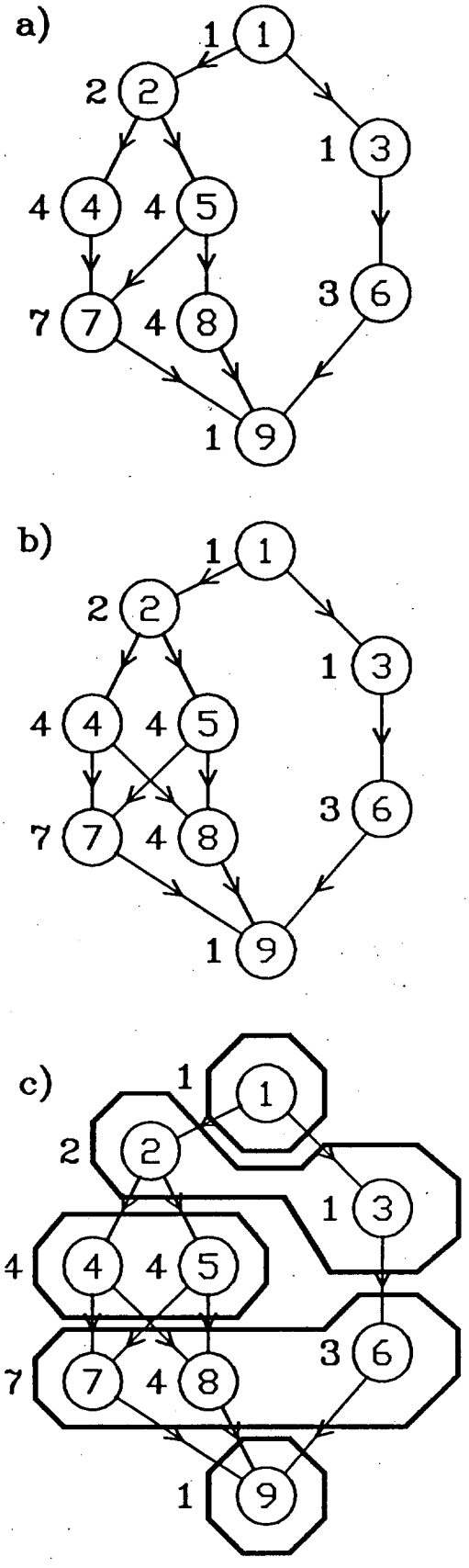
5. TOPOLOGIJA SISTEMA IN PRESLIKAVA ALGORITMA

V izdelanem konceptu rekonfigurabilnega večprocesorskega sistema topologija procesorskega polja kar najbolje ustreza zapisu algoritmov, ki določajo izvajanje opravila. Koncept kaže na dobro usklajenost topologije procesorskega polja s preslikavo algoritma.

Pri snovanju sistema je bilo upoštevano dejstvo, da je mogoče poljuben neregularen algoritem, ki ne vsebuje ciklov, imenovan "**blok**" programa, predstaviti z acikličnim grafom pretoka podatkov (ADFG). Primer takšnega grafa kaže slika 1a.

Vozlišče grafa predstavlja proces, povezave med vozlišči pa tokove podatkov. V splošnem je mogoče poljubni ADFG transformirati v hierearhični DFG (HDFG), ki predstavlja drevo (Slika 2b). ADFG je lahko preprost ali ne. Definicijo preprostega ADFG najdemo v naslednjem razdelku 5.1 in v delu Hwanga /2/. Transformacija ADFG v HADFG lahko poteka na dva načina. Prvi način transformacije ohranja podatkovno vodeno računanje in vodi v računanje, ki smo ga imenovali "**mehko sinhronizirano podatkovno vodeno računanje**". V postopku transformacije se neprepreosti grafi postopoma transformirajo v preproste le z dodajanjem novih (namišljenih) podatkovnih odvisnosti. Takšen primer je prikazan na grafu (Slika 1b), ki ga dobimo iz grafa na sliki 1a s povezavo med vozliščema 4 in 8.

Drugi način transformacije sicer ohranja princip podatkovno vodenega računanja, vendar odstopa od vodenja, ki je določeno z izvirnim ADFG. Računanje, določeno s takšno transformacijo, smo imenovali "**trdo sinhrozirano podatkovno vodeno računanje**". V postopku te transformacije se



Slika 1. ADFG in drevo grafa
a) Aciklični graf pretoka podatkov (ADFG)
b) Preprost ADFG
c) Možni sočasni procesi v ADFG

na osnovi skrbne analize programa, ki sloni na časovnih karakteristikah izvajanj posameznih procesov, v izvirnem ADFG odvzamejo nekatere povezave med vozlišči grafa. Primer takšne preslikave predstavlja preslikava grafa na sliki 1b v graf na sliki 1a. Pri uporabi takšnega načina transformacije morajo biti brezpogojno na voljo natančni podatki o dolžini ali možnih časovnih intrevalih trajanja vsakega procesa posebej.

Implementacija v tem prispevku zasnovanega transputerskega sistema in razvrščanja procesov je omejena na mehko sinhronizirano podatkovno vodeno računanje. Omejitev je pogojena z organizacijo komuniciranja in razvrščanjem procesov pri izbrani večzančni arhitekturi sistema.

Zaradi mehko sinhroniziranega podatkovnega vodenja na večzančni arhitekturi procesorskega polja, v katerem so vsi procesni elementi med seboj enakopravni, je omogočeno:

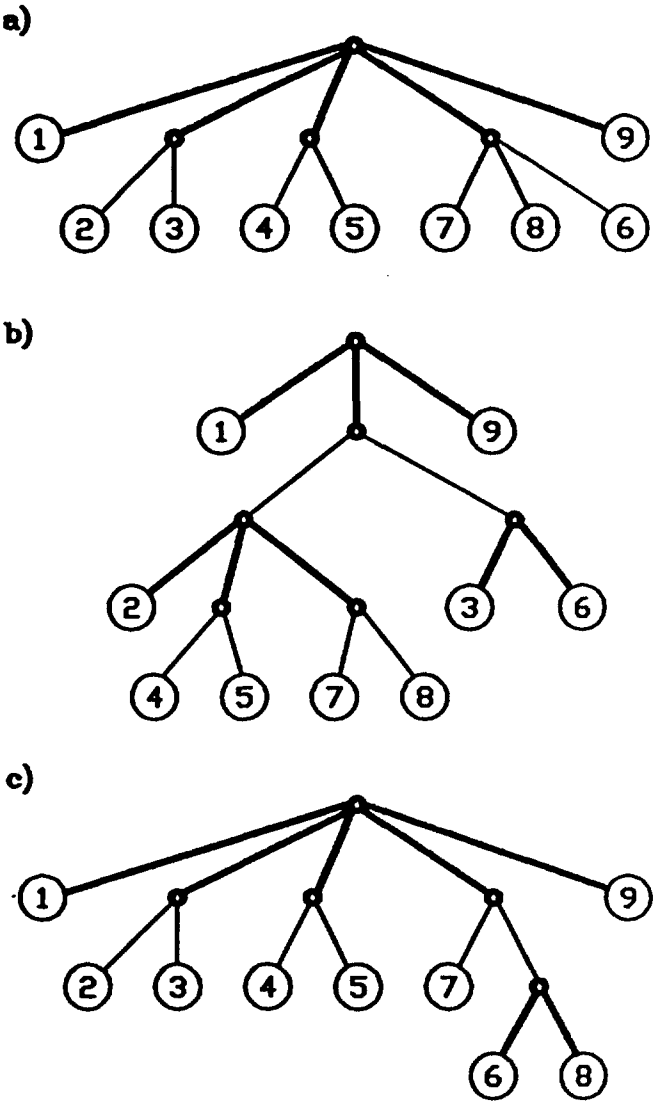
- dinamično dodeljevanje procesov
- sočasno izvajanje med seboj neodvisnih algoritmov
- medsebojno prepletanje različnih HDFG na polju procesorjev

Procesi na isti stopnji paralelnosti med seboj ne komunicirajo. Programerju ni treba skrbeti, na katerem procesorju se bo proces izvajal. Procesi se dodelujejo dinamično. Da ni del procesorskega polja preobremenjen, drugi del pa neizkoriščen, je topologija simetrična za več možnih osnovnih vstopnih mest. To pomeni, da se lahko različni algoritmi (ADFG) začnejo izvajati na različnih delih sistema. Ti deli sistema so, gledani posamično, topološko identični. Ker sistem omogoča medsebojno prepletanje različnih HDFG v procesorskem polju, je lahko bolje izkoriščen.

5.1. ADFG in HADFG

Graf pretoka podatkov opisuje izvajanje nekega programa (algoritma), ki je sestavljen iz medsebojno povezanih procesov. V nadaljevanju bomo predpostavili, da program nima povratnih zank. Graf, ki ustreza takšnemu programu, je acikličen in je prikazan na sliki 1a. Utež ob vozlišču grafa pomeni čas izvajanja procesa, ki ga vozlišče predstavlja. Ta čas je celoštevilčen večkratnik neke poljubne realne časovne enote.

Proces (vozlišče grafa) je zaključena celota. Začetek procesa je pogojen z določitvijo nabora vseh vhodnih podatkov, ki so potrebni za izvajanje procesa. Rezultat, kot izhodni podatek procesa, je navadno vhodni podatek za naslednji proces. Proces se lahko začne izvajati šele, ko ima na voljo vse potrebne podatke. Procesi, ki so med seboj neodvisni, se lahko izvajajo sočasno tako, kot kaže slika 1c. Izvajanje algoritma, ki je določen z grafom na tej sliki, je mogoče nazorno pokazati tudi z drevesno strukturiranim grafom na sliki 2a. Poudarjene razvejitve označujejo zaporedje, manj poudarjene pa sočasnost procesov.



Slika 2. Drevo grafa
a) Drevo grafa iz slike 1c
b) HADFG grafa iz slike 1b
c) Optimirano drevo grafa iz slike 1b,c

Aciklični usmerjeni graf, ki je DFG postopka, je mogoče predstaviti tudi z matematičnim zapisom. Ta predstavlja vsoto vseh možnih nizov vozlišč, po katerih je mogoče priti iz posameznih začetnih vozlišč do končnega vozlišča. Matematični zapis grafa na sliki 1b je potem:

$$X_1X_2X_4X_7X_9 + X_1X_2X_4X_8X_9 + X_1X_2X_5X_7X_9 + \\ + X_1X_2X_5X_8X_9 + X_1X_3X_6X_9 = \\ = X_1(X_2(X_4 + X_5)(X_7 + X_8) + X_3X_6)X_9$$

Graf je **preprost**, če ga je mogoče zapisati tako, da vsaka spremenljivka v njegovem matematičnem izrazu nastopa le enkrat. Iz takega matematičnega zapisa, v katerem pomeni, da se X_iX_j izvajata zaporedno, $X_k + X_m$ pa paralelno, je mogoče preprosto razbrati vse možne sočasnosti v grafu (i, j, k in m so elementi množice 1, 2, ..., 9).

Preprost acikličen DFG lahko prevedemo v **hierarhični ADFG (HADFG)**. Oblika HADFG je povsem primerljiva z

matematičnim zapisom ADFG. Preslikava ADFG na sliki 1b v odgovarjajoči HADFG na sliki 2b je določena z izpeljanim matematičnim izrazom.

Tudi tu, v HADFG je zaporedno izvajanje procesov prikazano z debelejšimi, vzporedno izvajanje pa s tanjšimi črtami. Vrstni red izvajanj je natančno določen. Poteka od leve strani drevesa proti desni strani in od zgoraj navzdol.

5.2. Analiza sočasnosti v ADFG

Algoritem, ki ga določa ADFG, je mogoče optimirati glede na čas izvajanja tako, da se v vsakem trenutku izvaja sočasno prav toliko procesov, kolikor je možno. Kot je bilo že omenjeno, tansformacija ADFG v optimirani graf (in dalje v HADFG) zahteva podatke o času trajanja posameznih procesov in splošno vodi k trdo sinhroniziranemu podatkovnemu vodenju. Pri takem vodenju lahko pride med procesiranjem algoritma do konfliktnih situacij v primerih, če se spremenijo časovne karakteristike posameznih procesov preko časovnih meja, ki so bile upoštevane v postopku transformacije.

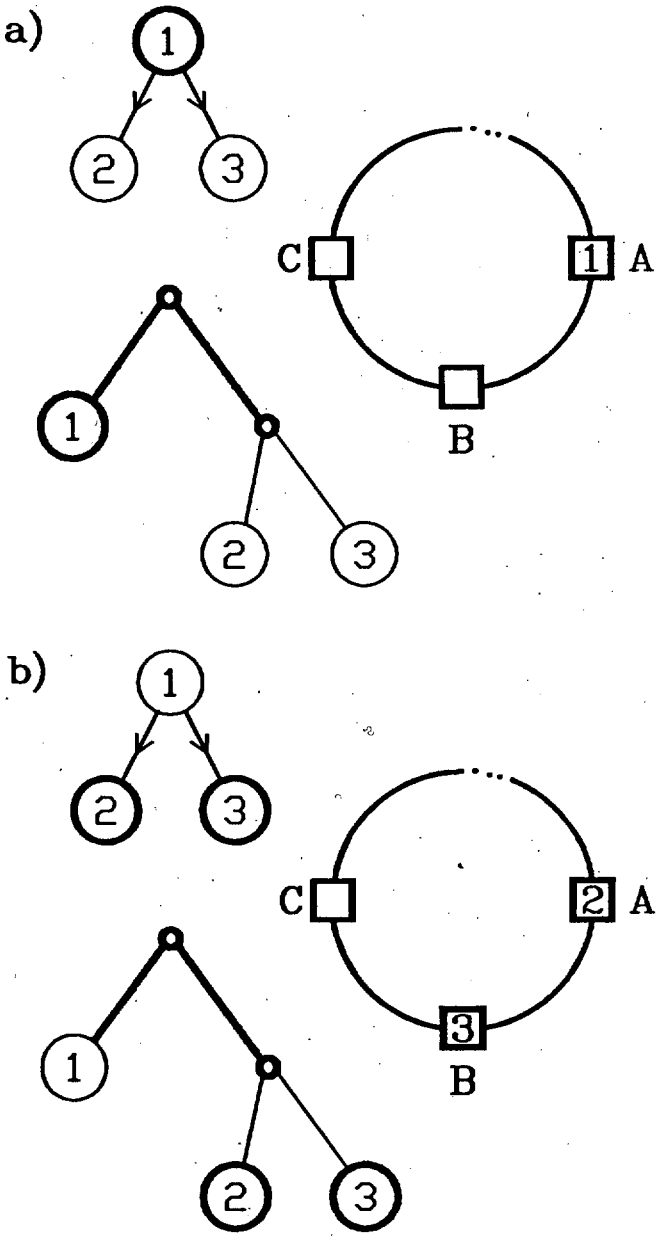
V nekaterih primerih je število procesorjev, ki je potrebno za časovno optimirano izvajanje algoritma, manjše. Včasih je tedaj mogoče z dodatno časovno analizo ugotoviti, da se lahko dva sicer vzporedna procesa izvajata sekvenčno, ne da bi se pri tem podaljšal čas izvajanja celotnega algoritma (Slika 2c). Tako optimiranje vodi k prostorski optimizaciji (tj. k manjšemu številu potrebnih procesorjev). Na primer, za izvajanje drevesa na sliki 2b so potrebni trije procesorji, za izvajanje drevesa na sliki 2c pa le dva procesorja.

5.3. Zanka in razvrščanje procesov

Ena izmed topologij, ki omogoča uresničitev prej navedenih zahtev, je zanka ali prstan. Zanka predstavlja več krožno povezanih procesnih elementov. V enostavnih zankah ima procesor povezavo le s svojima neposrednima sosedoma. Razdalje med procesorji v zanki se lahko skrajšajo z dodatnimi križnimi povezavami med procesorji, ki preskakujejo enega ali več procesorjev. Problem krajših poti v takšnih zankah sta podrobneje obravnavala Pisanski in Žerovnik v delu /3/. Z njimi se sicer poveča prepustnost zanke, žal pa postanejo mehanizmi prenašanja sporočil kompleksnejši.

Iz primera preprostega prstana na sliki 3 je razviden potek izmenjav sporočil med procesorji (procesi). Eden od treh procesorjev sprejme sporočilo o algoritmu, ki se mora izvajati. Naj bo ta procesor A. Algoritem določa dva vzporedna procesa, označena z 2 in 3, ki se pričneta izvajati, ko je proces 1 na procesorju A (Slika 3a) končan. Prvi izmed njiju (2) ostane na istem procesorju (A), drugi (3) pa se preseli na sosednji procesor (B). Slednjo situacijo kaže slika 3b.

Zanka omogoča dokaj uspešno razporejanje procesov tudi, če je procesov več kot procesorjev. Ko procesi zasedejo vse

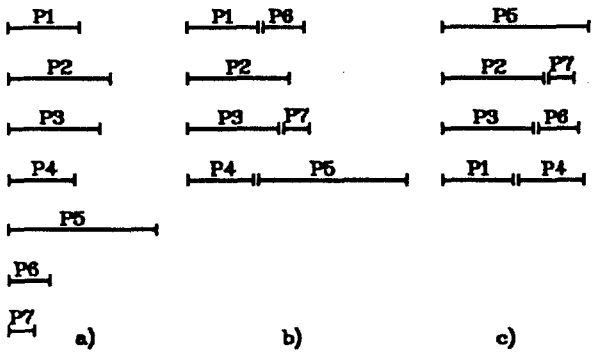


Slika 3. Izvajanje algoritma na procesorjih, ki so povezani v zanko.

razpoložljive procesorje, ostali procesi čakajo, da se eden od procesorjev sprosti. Mehanizem, ki skrbi za takšno dodeljevanje, se lahko izvede s pomočjo žetona, ki kroži po prstanu in išče prost procesor. Ko se eden od procesorjev sprosti in v vrsti za izvajanje čaka še kak proces, mu ga krožeči žeton dodeli. Na ta način je dosežen nekakšen avtomatizem razvrščanja. Vsi sodelujoči procesorji so enakopravni in za razvrščanje procesov ne potrebujejo nekega posebnega krmilnega procesorja.

Avtomatično razporejanje procesov po zgoraj opisanem načinu načelno ne zahteva podatke o trajanju posameznih

procesov. Brez predhodne skrbne analize sočasnosti in hkratne analize dolžin trajanja posameznih procesov, izvajanje algoritma ne bo dovolj časovno optimirano. Zato je potrebno vpeljati primerno strategijo razvrščanja sočasnih procesov, kar je razvidno iz primera na sliki 4. Na voljo so štirje procesorji v zanki, ki morajo izvesti 7 različno dolgih vzporednih procesov, prikazanih na sliki 4a. Optimalno izvajanje algoritma je v takem primeru možno, če obstajajo sočasni procesi, ki se lahko izvajajo zaporedno na istem procesorju, ne da bi se pri tem čas izvajanja algoritma podaljšal. Razvrščenost procesov na sliki 4b je neugodna.



Slika 4. Razvrščanje procesov

Procesi so razvrščeni po načinu FCFS (first come first served), kar ne vodi vedno v optimalno izvajanje algoritma. Procesa P4 in P5 se izvajata zaporedno na procesorju 4. Zato je čas izvajanja občutno daljši, kot v primeru na sliki 4c. V slednjem primeru sta procesa, ki se izvajata na procesorju 4, bolj ugodno izbrana. Način razvrščanja je najugodnejši, če za vsak par procesorjev velja, da sta vsoti trajanj procesov na posameznih procesorjih para čim bolj izenačeni.

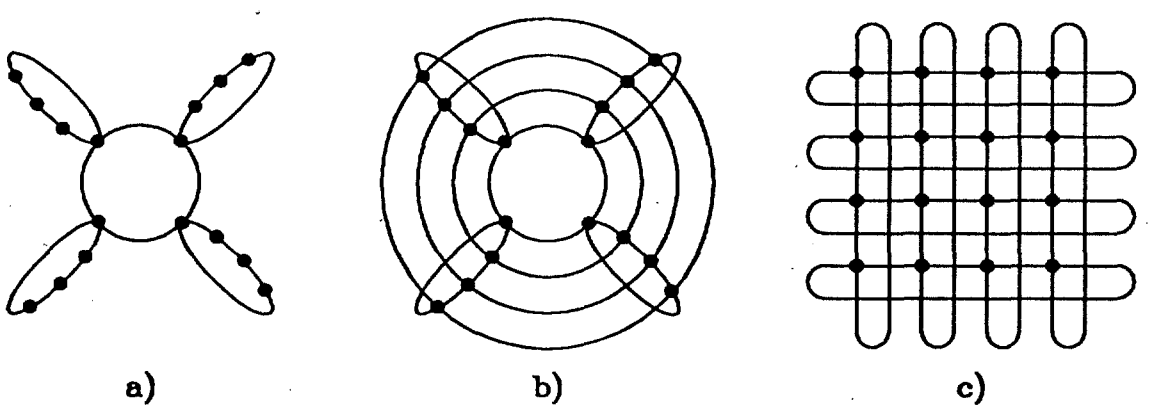
Iz gornjega je razvidno, da se pri neprimerno razporejenih procesih zmogljivost sistema zmanjša, vendar sistem pravilno deluje v obeh primerih. Tak mehanizem razvrščanja procesov je zato uporaben tudi pri izvajanju časovno nepredvidljivih procesov.

5.4. Topologija sistema

Znotraj prstana so procesorji med seboj povsem enakopravni. Procesor lahko vedno odda sporočilo le svojemu sosedu; tudi sprejemanje sporočil poteka preko sosedov. Če ima prstan velike razsežnosti pomeni, da bo čas potovanja sporočil sorazmerno dolg. Zato je v izdelanem konceptu število procesorjev v prstanu omejeno. Manj procesorjev v eni zanki je seveda premalo za zahtevnejše aplikacije, zato je vsakemu procesorju v zanki, imenovani **"osnovna zanka"**, dodeljena še ena zanka, imenovana **"sozanka"**. Osnovno zanko in sozanke, ki so vgnezdene v osnovno zanko, prikazuje slika 5a. Dobljena topologija ima prvo **"stopnjo vgnezdenosti zank"**. Z nadaljnim vgnezdevanjem je mogoče doseči poljubno stopnjo vgnezdenosti. Vsak procesni element na spojišču dveh prstanov ima štiri sosede. Pri predpostavki, da so prstani enosmerni, lahko procesor sprejema sporočila od dveh sosedov in jih tudi oddaja dvema sosedoma. To omogoča, da se v funkciji procesorja v zanki vpelje transputer. Število transputerjev v zanki je 4. To število ni izbrano naključno, ustreza številu transputerjev na komercialno dostopnih transputerskih vtičnih enotah, ki jih proizvaja INMOS /4/. Na teh vtičnih enotah so transputerji že krožno povezani.

Sistem je mogoče zaključiti tako, da se vse ostale istoležne transputerje, ki še niso križno povezani, poveže v zanko. Dobljeno vezje je prikazano na sliki 5b. To je mogoče predstaviti tudi v matrični razvrstitvi na sliki 5c ali kot trodimenzionalno hiper-kocko /5/. Procesorjev v polju je 16. Takšna topologija omogoča enostavno uporabo 2x2 procesorskih plošč s po štirimi transputerji. Povezave med transputerji omogočajo, da poteka razvrščanje procesov algoritma skladno s hierarhično strukturo HADFG. Poteka lahko od enega vstopnega prstana, ki ima vstopni procesor, do drugega. Sistem ima lahko tudi več vhodnih zank s po enim vhodnim procesorjem. Tako je možnih več neodvisnih in enakovrednih vhodov v sistem.

Transputer ima na voljo štiri dvosmerne zanke. S pomočjo dveh takšnih zank sta v enem prstanu realizirani dve med seboj neodvisni komunikacijski zanki. Vsaka izmed njih je namenjena komunikaciji v eni od obeh smeri. Smotrno je,



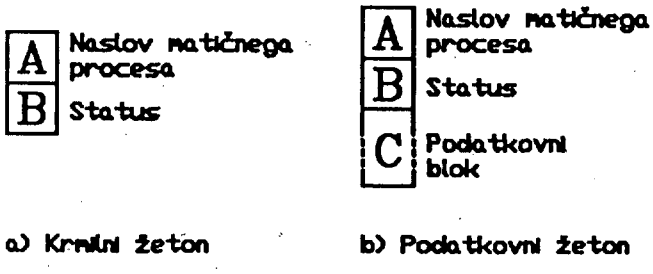
Slika 5. Topologija polja procesorjev

da je smer komunikacije določena s funkcijo komuniciranja. V eni smeri se prenašajo podatkovne, v drugi pa krmilne in statusne informacije. Enkraten prenos podatkov, ki so številnejši, traja dalj časa, prenos krmilnih in statusnih informacij pa krajši čas. Odzivi na krmilne informacije morajo biti čim hitrejši. Zaradi majhnega premera prstanov ni pričakovati, da bi čas komuniciranja med procesi (transputerji) bistveno vplival na učinkovitost sistema.

Prenašanje sporočil med procesnimi elementi je izvedeno preprosto s podajanjem žetonov, ki krožijo v zanki. Vsak žeton opiše poln krog. Procesor, ki je poslal žeton v zanko, ga mora kasneje iz nje tudi izločiti. Na tak način je zagotovljeno, da žeton obide vse procesorje v zanki. Praviloma lahko v zanki kroži več žetonov. Žeton se lahko nekje na poti po zanki ustavi pri procesorju, ki želi sprejeti od žetona neko sporočilo ali mu ga predati. Med kroženjem žeton ohranja naslov procesa (in s tem tudi naslov procesorja), ki je žeton poslal v kroženje.

5.5. Medprocesorske komunikacije

Medprocesorske komunikacije potekajo preko sporočil. Sporočila prenašajo žetoni. Teh je več vrst. Omenjeno je bilo že, da se uporabljata ločeni smeri za prenos podatkovnih in krmilnih informacij. Skladno s tem se razlikujejo žetoni, ki te prenose izvajajo. Poimenovani so kot krmilni in podatkovni žetoni. Načelno zgradbo žetonov obeh vrst prikazuje slika 6.



Slika 6. Zgradba žetonov

Možni statusi procesorja so:

- zaseden
- pripravljen
- nezaseden (prost)

Procesor je **zaseden**, kadar izvaja proces (v sklopu uporabniškega programa). Procesor ima status **pripravljen**osti od trenutka, ko sprejme zahtevo za izvajanje uporabniškega procesa, in do trenutka, ko prične proces izvajati. Kadar ni v nobenem od zgoraj opisanih statusov, je procesor **nezaseden** in opravlja le funkcijo prenašanja in razpoznavanja sporočil.

a) Krmilni žetoni. Krmilni žetoni (Slika 6a) vsebujejo le najosnovnejše informacije: naslov pošiljatelja in status oz. zahtevo, ki jo morajo posredovati ali vsem udeležencem v prstanu ali le enemu točno določenemu udeležencu.

Med krmilne žetone prištevamo žetona "**proces**" in "**konec**". Žeton "proces" išče prost procesor v zanki, žeton "konec" pa obvesti vse procesorje v zanki, da je v njej procesor, ki ima status "prost".

b) Podatkovni žeton. Sporočila, ki jih prenaša podatkovni žeton (Slika 6b), so razdeljena v dva dela. Prvi del ima enako zgradbo kot krmilni žeton in vsebuje naslov pošiljatelja in status podatka, ki ga sporočilo prenaša. Drugi del, odvisno od statusa, vsebuje: algoritme za proces in ustrezne začetne podatke ali rezulate izvršenega procesa.

Mehanizem komuniciranja z žetoni določa komunikacijski protokol, ki je podrobneje opisan v naslednjem razdelku.

5.5.1. Sporočila in komunikacijski protokol

V polju procesorjev se lahko izvaja en sam ali več vzporedno tekočih blokov. Podmnožica procesov, ki pripadajo istemu bloku in se izvajajo v isti zanki ali njeni sozanki, ima svoj "**matični proces**". Matični proces je element te podmnožice. Izvaja se na "**matičnem procesorju**", ki predstavlja koren nekega poddrevesa v HADFG.

Matični procesor pošlje po žetonu sporočilo o zahtevi po paralelnem izvajanju procesov. Vsebino žetona predstavljata naslov matičnega procesa (A) in status žetona (B). Status tega žetona je "**proces**" in žeton s takšnim statusom se imenuje žeton "proces". Ko žeton "proces" na poti po zanki naleti na prost procesor, mu ta spremeni statusni del (B) v status "**pripravljen**". Naslovni del žetona (A) ostane nespremenjen. Sporočilo, da je žeton našel prost procesor, je namenjeno matičnemu procesu, ki je poslal zahtevo po prostem procesorju. Ko žeton s statusom "pripravljen" dospe do matičnega procesorja (procesa), je ta obveščen, da je v zanki procesor, ki je nezaseden in pripravljen sprejeti podatke za izvajanje novega procesa. V primeru, da žeton na poti po zanki ne najde prostega procesorja, ostane status žetona nespremenjen. Matični procesor ga zadrži, dokler po žetonu "konec" ne sprejme sporočila, da je v zanki procesor, ki je nezaseden.

Ko matični procesor sprejme sporočilo, da obstaja procesor s statusom "pripravljen", pošlje podatkovni žeton. Ta vsebuje tri dele. Naslovni del (A) vsebuje naslov matičnega procesa. Statusni del (B) pove, da gre za vhodne podatke. V tem primeru ima žeton status "**podatek**". Podatkovni del (C) vsebuje poleg vhodnih podatkov tudi programski kod. Procesor, ki sprejme podatkovni žeton, preide v status "zaseden" in prične izvajati nalogo, ki je določena s programskim kodom, podatkovni žeton pa zadrži do konca izvajanja naloge. Po končani nalogi procesor spremeni statusni del zadržanega podatkovnega žetona v status "**rezultat**" in v podatkovnem bloku tega žetona posreduje rezultate matičnemu procesu. Potem, ko

procesor odda rezultate, generira krmilni žeton "konec". Ta žeton vsebuje status **"konec"** in z njim signalizira, da je v zanki prost procesor, ki lahko prevzame novo nalogo.

5.5.2. Preslikava algoritma na polje procesorjev

Preslikavo nekega algoritma na načrtovano polje procesorjev sestavljata dva bistvena koraka:

1. Predstavitev algoritma v paralelnem zapisu.

V poglavjih 2 in 3 je bila že nakazana problematika paralelnega zapisa algoritma. Ta največji in hkrati najpomembnejši korak k preslikavi algoritma sestavljajo naslednji postopki:

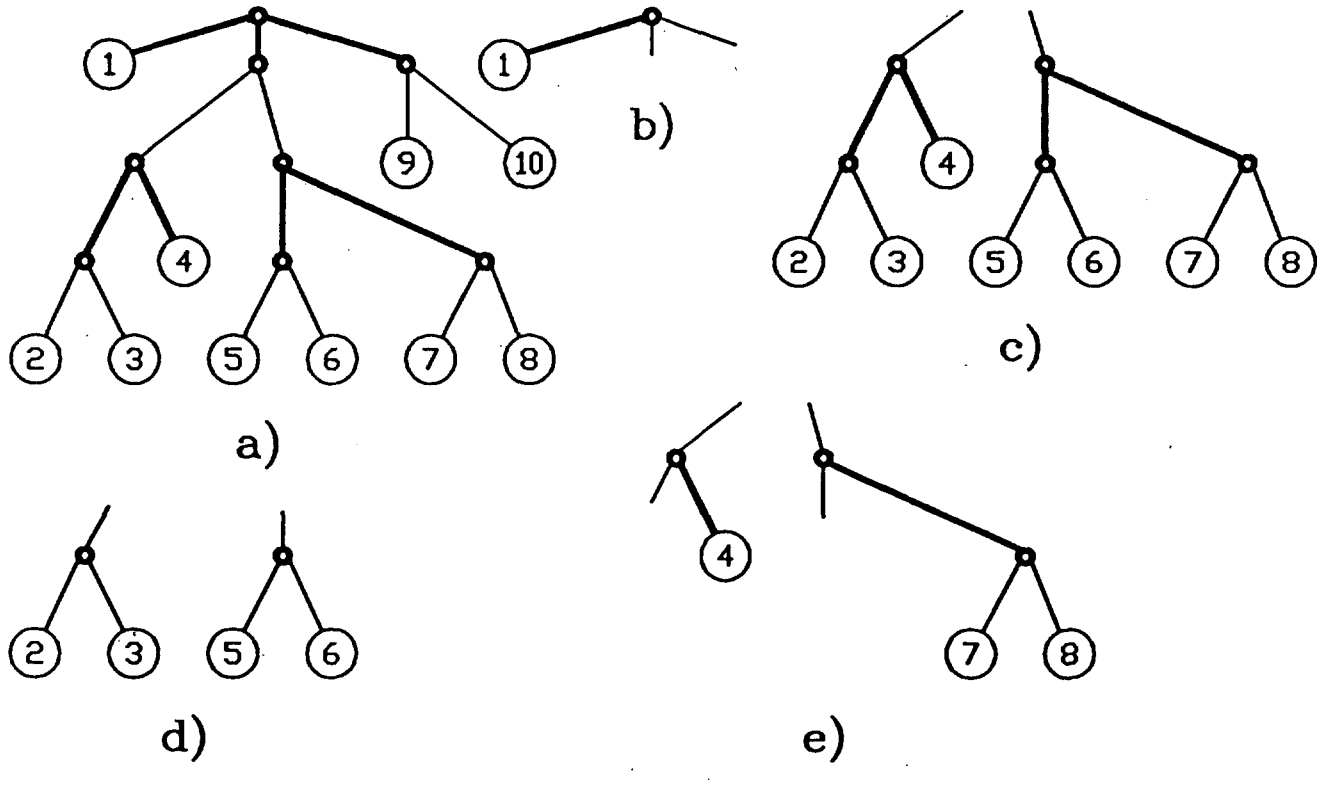
- granulacija programa (algoritma)
- dekompozicija algoritma v aciklične podalgoritme (bloke)
- odkrivanje sočasnih procesov
- paralelni zapis blokov programa v matematični obliki, ki je predstavljiva s t.im. hierarhičnim acikličnim podatkovno pretočnim grafom HADFG.
- optimalno razvrščanje procesov na osnovi časovne analize

2. Generiranje naslovnih kodov vseh poddreves v množici hierarhičnih acikličnih podatkovno pretočnih grafov.

Naslovni kod je mogoče neposredno generirati iz HADFG. Predstavlja naslov nekega poddrevesa algoritma. Med njegovim izvajanjem se naslovni kodi dinamično preslikajo na matične procesorje transputerskega polja.

Po prvem koraku preslikave celotnega algoritma, ki se izvaja na sistemskem nivoju hierarhičnega večprocesorskega sistema (HMPS) /1/, je algoritem podan v matematični obliki (računalniškem zapisu) tako, da je predstavljiv z množico drevesnih grafov HADFG, ki so bloki programa, in z osnovnim grafom, ki jih povezuje med seboj. Osnovni graf je cikličen usmerjeni graf (CDFG), ki opisuje vse zanke algoritma. Program, ki je določen z CDFG, se lahko izvaja v vstopnem transputerju procesorskega polja, ali se izvaja na nekem drugem transputerju delovnega (uporabniškega) nivoja sistema. Vhodni transputer posreduje matičnemu procesorju v izvajanje bloke algoritma zaporedno ali kvazi vzporedno, če CDFG vsebuje tudi paralelne zanke. V tem primeru je ugodno (ni pa nujno), da bloki vstopajo v polje paralelno (ali kvazi paralelno) preko večih vhodnih transputerjev.

V drugem koraku se blok, ki je določen s pripadajočim HADFG, preslika na procesorsko polje. Pri tem se posamezni procesi bloka dinamično razvrščajo po polju transputerjev. Mehanizem dodeljevanja procesov je enak



Slika 7. Ponazoritev preslikave bloka na polje transputerjev.

za vse procesorje polja. Preslikavo bloka prikazuje primer na sliki 7.

Pri razvrščanju sočasnih procesov je uporabljena strategija razvrščanja po časovni uteži posameznih procesov. Paralelni proces (P1), ki se (ali se verjetno) izvaja najdlje, ima največjo utež. Izvaja se na "prvem" procesorju v zanki. Takšen proces P1 stoji v paralelni množici HADFG pred procesom P2. Po časovnih utežeh so tako razvrščeni sočasni procesi in poddrevesa vseh paralelnih množic v HADFG. Pri tem je mišljeno, da stoji proces P1 pred procesom P2, če se v HADFG nahaja v paralelni množici prvi, gledano od leve proti desni.

Procesor polja, ki se nahaja v zanki vhodnega procesorja in prevzame podatke, ki so potrebni za izvajanje HADFG, je matični procesor tega HADFG. Ta procesor na najvišjem nivoju prevzame nadzor nad izvajanjem procesov, ki se dinamično vključujejo in izvajajo na istem nivoju, na katerem se sam nahaja. Izvajanje na nižjem nivoju preda naslednjemu matičnemu procesorju v hierarhiji izvajanja procesov, ki je določena z HADFG.

Naj bo začetni nivo grafa sekvenčen, kot kaže slika 7a. Prvi prosti procesor najprej poskrbi za izvajanje procesa 1 (Slika 7b). Ko se proces 1 konča, se mora izvršiti poddrevo, ki izvira iz druge sekvenčne veje. Ta veja se deli na dve vzporedni veji. Jasno je, da se mora poddrevo vsake od obeh vej izvajati na ločenih podmnožicah procesorjev (Slika 7c). Procesor, ki sprejme neko poddrevo, postane matični procesor tega poddrevesa in prične takoj izvajati proces, ki se na poti po drevesu nahaja na prvem mestu. Procesi, ki se v HADFG nahajajo na najnižjem nivoju posameznih vej drevesa (kar pomeni, da v smeri od zgoraj navzdol nimajo naslednika) se imenujejo "**terminalni procesi**". Med potovanjem po grafu navzdol dobi vsako poddrevo ali terminalno vozlišče, ki izhaja iz vzporedne veje, svoj matični procesor (proces). V splošnem ni nujno, da so matični procesorji posameznih delov grafa različni procesorji. Vsak procesor je lahko hkrati matični procesor za več podgrafov. Vsak HADFG se tako postopoma deli na manjše dele, ki se razvrščajo po procesorjih. Razvrščanje se ustavi na terminalnih procesih (Slika 7d in 7e), ki so, ali

- zaporedje procesov, določeno s sekvenčnimi vejami, ki se ne delijo več v vzporedne veje, ali
- procesi vzporednih vej, ki se ne delijo več v sekvenčne veje.

Ko se terminalni procesi, ki se izvajajo na matičnem procesorju ali na njemu pripadajočih procesorjih, končajo, predajo rezultate svojemu matičnemu procesorju. Ta nato izvrši naslednji korak v podgrafu (Slika 7e). Prične izvajati proces 4. Postopek se tako nadaljuje, dokler niso izvršeni vsi procesi, ki jih HADFG določa.

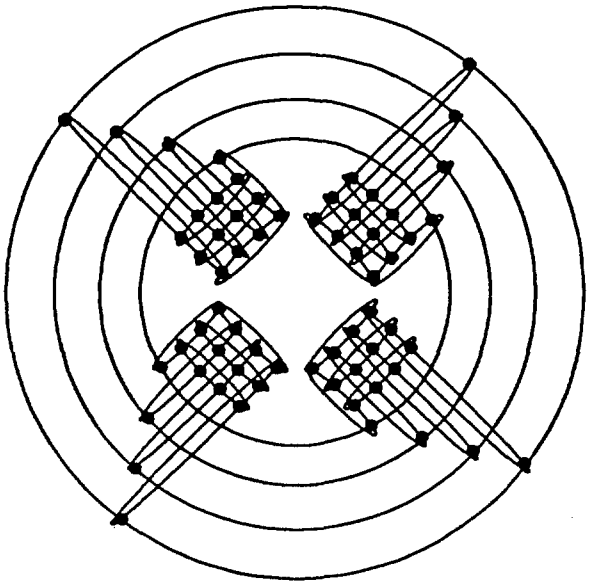
Vsak procesor ima tabelo, v kateri hrani informacijo o želenem procesu. Informacija je predstavljena v obliki HADFG. Prvi matičen procesor ima v tej tabeli zapisan celoten algoritem. Poleg algoritma shranjuje tudi podatke, kot so vhodni podatki in vmesni rezultati, ki so potrebni za njegovo izvajanje.

Preslikava algoritma na več zank, ki so vgnezdene druga v drugo, poteka po pravilu "prioritetne smeri", ki prispeva k boljši izkoriščenosti procesorskega polja. V zvezi s tem pravilom sta vpeljana pojma "**primarne**" in "**sekundarne**" zanke. Matični procesor sprejme podatke za izvajanje poddrevesa po eni od dveh zank polja, ki jima pripada. Smer, po kateri matični procesor sprejme podatke, je primarna smer. Matični procesor poskuša predati podatke za izvajanje preostalega dela drevesa, ki ga ne izvaja sam, najpreje procesorju v sozanki (tj. v sekundarni smeri) in šele, če mu to ne uspe, v primarni smeri (po osnovni zanki). Mehanizem predaje deluje takole: Kot je bilo rečeno, pošlje matični procesor žeton "proces" najpreje v sekundarno zanko. Če so vsi procesorji v tej zanki zasedeni, se žeton "proces" z nespremenjeno vsebino vrne k matičnemu procesu, ta pa ga nato pošlje v sozanko, tj. v primarno smer. V primeru, da je tudi ta zanka zasedena, matični procesor čaka, dokler se ne sprosti procesor v eni ali drugi zanki. O tem je matični procesor obveščen po sproščeni zanki z žetonom "konec" , nakar se celoten ciklus z žetonom "proces" v sproščeni zanki ponovi.

Do naslednje akcije pride, ko prispe žeton "pripravljen" do matičnega procesorja. Ta vzame iz tabele poddrevo (ali proces), ki čaka na izvajanje, in ga z žetonom "podatki" pošlje v prstan. Žeton "podatki " vsebuje kodo izvornega (matičnega) procesa in izvora podatkov. Matičnemu procesorju cilj potovanja podatkovnega žetona ni poznan. Na poti po zanki podatkovni žeton išče procesor, ki je pripravljen. Ko ga najde, spremeni njegovo stanje "pripravljen" v stanje "zaseden", procesor pa sprejme poddrevo, ki mu je namenjeno, in poskrbi za njegovo izvajanje. Žeton "podatki" ostane med izvajanjem poddrevesa na procesorju, ki njegovo izvajanje nadzira. Ko je poddrevo izvršeno, nadzorni (matični) procesor pošlje v oba prstana žeton "konec". Z njima obvesti procesorje v obeh prstanih, da se je procesor v zanki sprostil (da je nezaseden), s podatkovnim žetonom, ki mu dodeli status "rezultat", pa pošlje rezultat v prstan, kjer se nahaja njegov matični procesor. Ko žeton "rezultat" prispe do svojega matičnega procesorja (pošiljatelja žetona "podatki", ki se nahaja na višjem nivoju), ga ta vzame iz obtoka. Sprejete podatke razvrsti skladno z izvornim kodom, ki je bil poslan po žetonu "podatki". S tem je izvajanje podgrafa končano.

Iz opisanega lahko zaključimo naslednje:

- Hierarhično razvrščanje procesov po procesorjih v hierarhično vgnezdenih zankah zmanjšuje obremenitev komunikacijskih poti, saj potujejo sporočila preko manjšega števila posrednikov.
- Zaradi manjšega števila procesorjev je možna manjša razvejitev komunikacijskih poti in zato slabša izkoriščenost procesorskega polja.
- Izstop iz zanke v sozanko je mogoč le na matičnem procesorju. Na ta način se pri manjšem številu potrebnih komunikacij med procesorji in krajši poti



Slika 8. Razširitev procesorskega polja na 4 x 16 procesorjev.

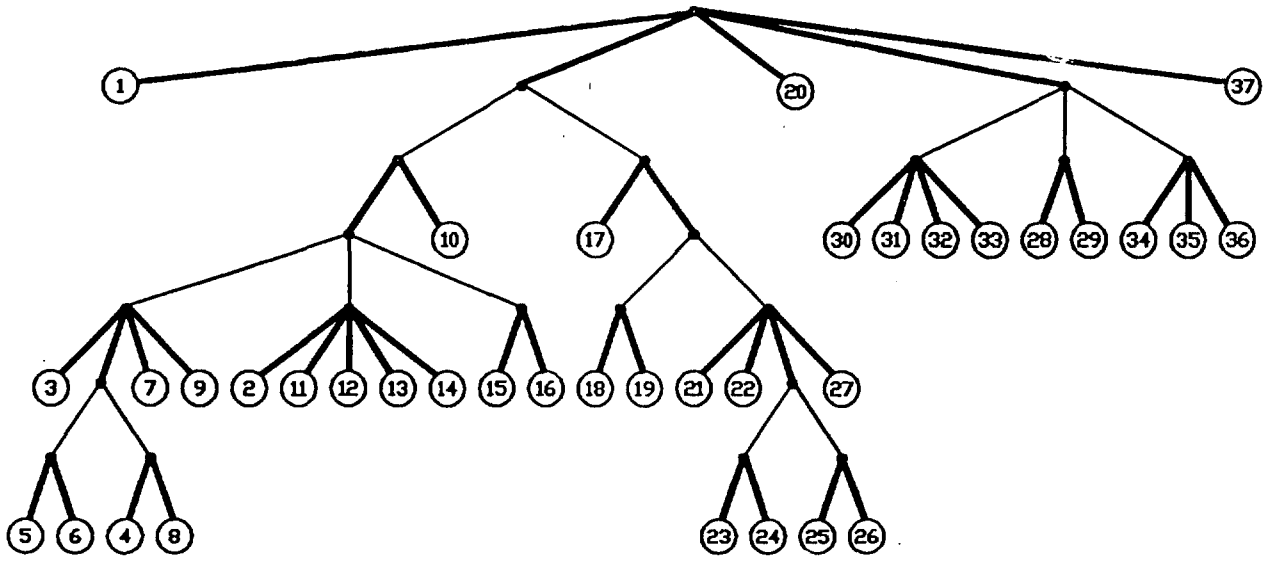
od poljubnega procesa v polju do vstopnega procesa (tj. do procesa, ki se izvaja na vstopnem procesorju), mehanizem komuniciranja poenostavi.

- Teoretično se lahko blok (ADFG) preslika na poljubno število nivojev, praktično pa je to število omejeno s številom sočasnih procesov ali/in z velikostjo procesorskega polja. Število nivojev je določeno s številom med seboj vgnezdenih zank (prstanov), na katerih se blok izvaja. Omejitev zaradi prevelikega števila sočasnih procesov istega poddrevesa, ki se izvajajo na istem nivoju, je mogoče odpraviti s takšno transformacijo ADFG v HADFG, da se v postopku transformacije uporabi postopek translacije nad presežnim delom sočasnih procesov. S postopkom translacije se "odvečne" procese prenese na nižji nivo. Druga omejitev je odpravljiva z manjšo modifikacijo tu obravnavane topologije polja. Osnovni modul transputerskega polja je namreč mogoče tako organizirati, da dopušča gradnjo bolj obsežnih polj z vgnezdevanjem zank na večih nivojih do željene stopnje. Ena od možnih razširitev polja je pokazana na sliki 8.
- Procesi se izvajajo tudi na matičnem procesorju, kar prispeva k manjšemu številu potrebnih komunikacij.
- Vsak procesor transputerskega polja mora med izvajanjem uporabniškega procesa zagotavljati razpoznavanje žetonov in nemoten prenos sporočil svojim sosedom.
- Vsi procesorji imajo enako operacijsko programsko opremo. Izjemo predstavlja le transputer na vstopu v procesorsko polje.

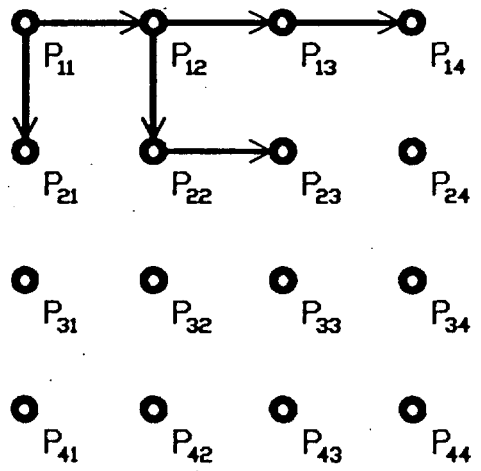
Preslikava algoritma na polje procesorjev je prikazana na naslednjem primeru: Slika 9 kaže HADFG bloka **B**, ki se preslika na polje transputerjev na sliki 10. Mehanizem razvrščanja porazdeli procese bloka **B** po transputerjih tako, kot kaže tabela na sliki 11. Transputer Pij, se nahaja na presečišču zank i, j. Preslikava bloka **B** na polje transputerjev je razvidna iz slike 10 po uporabljenih komunikacijskih poteh med sedmimi procesorji P11 do P23, ki so bili zaseženi med izvajanjem bloka.

7. ZAKLJUČEK

Sistem je prvenstveno namenjen raziskavam večprocesnih paralelnih sistemov. Pričakovati je, da bo načrtovana arhitektura večtransputerskega sistema učinkovita v sistemih,



Slika 9. HADFG bloka **B**



Slika 10. Preslikava algoritma B na procesorsko polje.

ki delajo v realnem času in so namenjeni vodenju obsežnih in zahtevnih procesov z visoko stopnjo zanesljivosti /6,7/.

Iz pričujočega dela, je mogoče strniti naslednje ugotovitve:

- Načrtovani sistem je rekonfigurabilen.
- Polje transputerjev, ki je vgrajeno v delovni (uporabniški) nivo sistema HMPS, je razširljivo. Mehanizem razvrščanja je namreč neodvisen od števila transputerjev. Z dodajanjem novih transputerjev v posamezne zanke, ni potrebno spreminjati podporne programske in materialne opreme. Zato je možno število procesorjev v zanki neomejeno večati. Smiselno pa je, da to število ni preveliko. V nasprotnem primeru daljše komunikacijske poti upočasnijo delovanje sistema.
- Pri optimiranju programa je pomembna primerna izbira sinhronizacije podatkovnega vodenja. Vodenje je lahko:

  a) asinhrono
  b) mehko sinhronizirano
  c) trdo sinhronizirano

- Postopek preslikave na koraku 1 in 2 (glej poglavje 5.5.2) je mogoče v celoti avtomatizirati in s tem bistveno olajšati delo programerja.
- Mehanizem dodeljevanja procesov je enak za vse procesorje polja, kar bistveno prispeva k prilagodljivosti sistema.
- Več vstopnih transputerjev lahko omogoča večjo izkoriščenost polja in bolj učinkovito proritetno izvajanje.

V zvezi s hierarhičnim večprocesorskim sistemom je odprtih več vprašanj, ki se nanašajo predvsem na verifikacijo predlaganega sistema. Simulacija sistema naj bi pomagala dati odgovor na številna vprašanja, med njimi:

- Kakšna je možnost zasičenosti poti za prenos podatkov?

Procesor	Procesi
P11	1, 3, 5, 6, 7, 9, 10, 20, 30, 31, 32, 33, 37
P22	17, 18, 19, 28, 29
P13	2, 11, 12, 13, 14, 34, 35, 36
P14	15, 16
P21	4, 8
P22	21, 22, 23, 24, 27
P23	25, 26

Slika 11. Tabela razvrščenosti procesov

- Kakšna je porazdelitev zasičenosti v procesorskem polju?
- Kakšna je učinkovitost sistema v odvisnosti od različnih parametrov, kot so granulacija, velikost procesorskega polja in drugi?

Nadaljne raziskave naj bi bile osredotočene v izboljšave karakteristik sistema. V ta namen naj bi obravnavale:

- dvosmerne komunikacije med procesorji z optimiranjem prenosnih poti
- komunikacije med sočasnimi procesi v primerih trde sinhronizacije podatkovnega vodenja
- prioritetno vodenje
- problematiko razvoja učinkovitih orodij za avtomatizirano načrtovanje programske opreme
- topologijo povezav med procesorji, ki omogoča preprosto modularno večanje procesorskega polja

9. LITERATURA

/1/ M.Szturmowicz and M.Tudruj, A multi-layer transputer network for efficient parallel execution of occam programs, Microprocessing and Microprogramming 28 (1989) pp.133-138.

/2/ K.Hwang and F.A.Briggs, Computer architecture and parallel processing, McGraw-Hill Book Company, 1984.

/3/ T.Pisanski and J.Žerovnik, Computing Diameter in Multiple-loop Networks, Preprint Series Dept. Math. University E.K. Ljubljana, 27(1989) No.286.

/4/ INMOS Spectrum, "Product Information, The Transputer Family", March 1988.

/5/ L.C.Waring, A general purpose communications shell for a network of transputers, Microprocessing and Microprogramming 29 (1990) pp.107-119.

/6/ Peter Kolbezen and Peter Zaveršek, Transputers for Embedded Real-Time Systems, Informatica, A Journal of Computing and Informatics, Vol.14, No 3, July 1990, pp.35-43.

/7/ Barbara Koroušić, Jim E.Cooling and Peter Kolbezen, Real-Time Executives for Embedded Microprocessor Applications, Informatica, A Journal of Computing and Informatics, Vol.14, No 4, October 1990, pp.58-63.

/8/ Peter Kolbezen, Borut Robič in Branko Mihovilović, Podatkovno vodeno računanje na polju transputerjev, MIPRO - Savjetovanje o novim generacijama računara - NG, str. 5-52 do 56, Rijeka-Opatija, maj 1989.