Keywords: tree, network, allocation, network construction

Peter Zaveršek*, Peter Kolbezen**
* VELCOM d.o.o., Foitova 8, Velenje
** Institut Jožef Stefan, Jamova 39, Ljubljana

POVZETEK. Delo obravnava preslikavo algoritma, ki je določen s hierarhičnim acikličnim grafom pretoka podatkov (HADFG), na drevesno avtomatno strukturo (TAS). Predlagani postopek preslikave upošteva prostorsko optimizacijo avtomata. Podana je primerjalna analiza med avtomatom z drevesno in avtomatom s toroidno strukturo. Na osnovi te analize so pokazane dobre in slabe strani obeh struktur avtomata.

ALLOCATION OF HADF GRAPHS ONTO A TREE STRUCTURE. Hierarchical acyclic data flow graph (HADFG) tree-shaping form encouraged us in investigating a possibility of their allocation onto a tree-like automat structure (TAS). Various approaches can be used in order to obtain an appropriate structure. The paper proposes an analytical space-optimizing approach towards the network construction. Further, execution results on such a network are compared to the ones obtained by a torus-like network and the behaviour of each of them is discussed.

1. Uvod

V delih [1, 2] je opisan večprocesorski sistem za učinkovito izvajanje hierarhičnih acikličnih grafov pretoka podatkov (HADFG). Sistem ima toroidno avtomatno strukturo. Preslikava HADFG na takšno strukturo je posredna. Mehanizem posredne preslikave s svojimi neželenimi vplivi podaljšuje čas izvajanja HADF grafov, ker zahteva zase del procesorskih zmogljivosti. Tako si lahko upravičeno zastavimo vprašanje uspešnosti neposredne preslikave HADFG na avtomatno strukturo. Postopek preslikave, če naj bo uspešen, mora v povezavi z avtomatno strukturo čim manj obremenjevati procesorje. Preslikava naj bo zato neposredna, procesorji v strukturi pa statično povezani. Zaželena je avtomatna struktura, ki se povsem ujema s programskim grafom v času izvajanja grafa. Procesorji v dinamično rekonfigurabilnem polju, kot npr. [3] ali sistemi ParsyTec-a [6], bi poleg kompleksnejše strojne opreme zahtevali dodatne procesorske zmogljivosti za povezovanje v času izvajanja. Omenjene sisteme bi lahko uporabili kot osnovo statičnim sistemom, če bi procesorje v njih ustrezno povezali pred začetkom izvajanja programskih grafov. Procesorji v statični strukturi, ki jo zahtevamo, bodo predvidoma manj izkoriščena kot tisti v toroidnem sistemu, ker se statična struktura ne more prilagajati trenutnim razmeram v sistemu (zasedenost, razpoložljivost procesorjev). S transformacijo DFG v HADFG dobimo drevesni programski graf s korenom zgoraj, in vejami, ki se širijo od korena navzdol [1]. Kot je splošno znano, razlikujemo pri drevesnih grafih več tipov vozlišč. *Koren* grafa je vozlišče, ki nima predhodnika, in ima enega ali več naslednikov. *Razvejišče* je vozlišče, ki ima enega predhodnika in enega ali več naslednikov. Vozlišče grafa, ki ima le predhodnika in nima nobenega naslednika, imenujemo *list grafa*. Če poskušamo takšen graf čimbolj neposredno preslikati v avtomat, lahko sklepamo da bo imela tudi avtomatna struktura drevesno obliko. To nas je vzpodbudilo k primerjalni raziskavi drevesne avtomatne stukture in že v delih [1, 2, 4] obravnavane toroidne strukture. Pričakujemo, da bo neposredna preslikava na drevesno avtomatno strukturo (TAS) ponudila boljše rezultate glede na čas izvajanja vsaj za neko družino HADF grafov. Družino HADFG, ki se bo uspešneje izvajala, želimo tudi identificirati. Zato poskušamo v tem delu odgovoriti, ali je za izvajanje alogritmov, ki so predstavljeni v obliki HADF grafov, učinkovitejši toroidni sistem ali TAS.

2. Drevo in torus

Predlagana toroidna struktura v delih [1, 2] je sestavljena iz množice prepletenih zank (prstanov) transputerjev. Vsak transputer v avtomatni strukturi je fizično vezan na štiri sosede. Komunikacija med njimi je krožna po zankah, ki sestavljajo torus, in poteka eno-

smerno. Vsak procesor ima zato le dva logična soseda. Komunikacijske zanke ponujajo možnosti dvostranskih prenosov, ki jih izkoriščamo tako, da znotraj ene fizične zanke vzpostavimo dve logični komunikacijski zanki. Medtem, ko se po eni prenašajo podatki, ima druga funkcijo prenosa krmilnih sporočil in ukazov. Zasnova sistema podpira dinamično, "run-time" razporejanje procesov glede na proste vire v polju. Izbrana arhitektura, podprta s preprostimi mehanizmi optimizacije razporejanja, omogoča skupaj z uporabljeno arhitekturo sorazmerno učinkovito preslikavo programskega grafa na statično povezano polje procesorjev.

Vsi procesorji v toroidnem polju imajo enako število sosedov, zato so si enakovredni ne glede na položaj v vezju. Ker imajo vsi procesorji logične naslednike, se polje navidezno nikjer ne zaključi. Izvajanje programskih algoritmov, ki so predstavljeni v obliki HADFG, je, zahvaljujoč virtualni razširljivosti toroidnega polja, učinkovito in uspešno tako pri manjšem, kot pri večjem številu procesorjev.

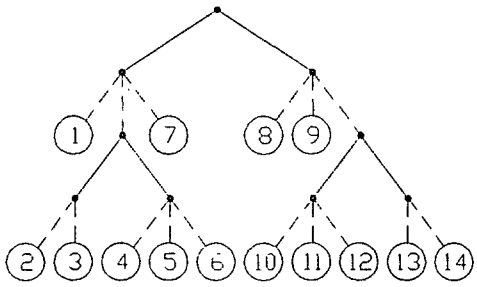
Drevesna struktura za razliko od toroidne ni virtualno razširljiva. Terminalni procesorji (listi drevesne avtomatne strukture) nimajo naslednikov. Širjenje HADF grafa po takšnem polju se zaustavi na terminalnih procesorjih, kjer se struktura zaključi. Zato drevo ne ponuja vsem procesorjem enakovrednega položaja v strukturi. Učinkovita preslikava HADF grafa na drevesno strukturo zahteva ustrezno razvejanost in globino strukture.

Drevo torej ni tako prilagodljivo, kot je torus. Če želimo izvajati programski graf na statično povezani drevesni avtomatni strukturi, moramo ustrezno strukturo praviloma zasnovati vnaprej. Struktura je lahko predvidena ali za vsak HADF graf posebej ali za neko natančno določeno družino HADF grafov.

3. Določitev drevesne strukture

Učinkovito izvajanje algoritma, ki ga predstavlja določen HADF graf, zahteva ustrezno drevesno avtomatno strukturo. Razsežnost avtomata mora biti vsaj takšna, kot je razsežnost preslikave grafa. Globina avtomatne strukture mora biti vsaj enaka številu paralelnih nivojev HADF grafa, medtem ko mora širina na opazovanem delu ustrezati številu paralelnih vej v vozliščih grafa, ki se bodo preslikala na to avtomatno vozlišče.

Zahtevamo, da je avtomatna struktura splošna. Ustrezati mora vsakemu HADF grafu dane oblike in mora zagotavljati neomejeno širjenje procesorskih aktivnosti v njej, neodvisno od časov izvajanja posameznih procesov. Posledica pogoja takšne časovne neodvisnosti je, da je tudi rezultat optimizacije skromnejši. Primer optimiranja avtomatne strukture je npr. sekvenčno izvajanje dveh kratkih paralelnih procesov ob tretjem zelo dolgem, s čemer lahko prihranimo kakšen procesor. V nadaljevanju bomo nakazali možne poti, ki nas pripeljejo do ustreznega avtomata.



Slika 1: Izhodiščni HADF graf

3.1 Prekrivanje

HADF graf natančno predpisuje način izvajanja posameznih vej v skladu s tipom veje (paralelna, sekvenčna). Možnost, ki se sama ponuja, je neposredna preslikava grafa v avtomatno strukturo. Glede na tip veje lahko zaključimo naslednje:

- **Paralelne veje:** Vse paralelne veje, ki izhajajo iz istega (paralelnega) vozlišča, se izvajajo sočasno. Izvajanje ostalih paralelnih vej grafa glede na opazovano vejo je določeno z njihovo lego v grafu. Zaželeno je, da se paralelne veje znotraj enega paralelnega vozlišča, kot tudi vse njim paralelne veje drugod po grafu, izvajajo sočasno. Zajeti želimo ves ponujeni paralelizem in tako doseči največjo stopnjo paralelnosti izvajanja. S tem bo tudi hitrost izvajanja največja. Med izvajanjem moramo imeti vsak trenutek na razpolago zadostno število procesorjev, odgovarjajoče številu procesov, ki jih je v danem trenutku možno izvajati sočasno. Procesorji morajo biti medsebojno povezani tako, kot zahteva HADF graf, saj v nasprotnem primeru preslikava procesov nanje ne bi bila možna.

- **Sekvenčne veje:** Sekvenčne veje se izvajajo druga za drugo. Pravimo, da so si tuje. Pri izvajanju izrabljajo isti del avtomatne strukture. Kaskado večih zaporednih sekvenčnih procesov lahko obravnavamo kot en sam proces. Vse procese, ki sestavljajo kaskado, preslikamo na isti procesor. Podobno obravnavamo podgrafe, ki si sledijo drug za drugim. Tudi te preslikamo na isti del avtomatne strukture. Zavedati se moramo pomembne razlike med procesi in podgrafi. Posamezni procesi zasedejo le en procesor, medtem ko je razsežnost in oblika zahtevanega avtomatnega polja za podgraf določena s pripadajočo globino in širino podgrafa. Potrebno globino podavtomata določa število paralelnih nivojev najglobjega podgrafa. Število procesorjev na posameznem nivoju je enako številu paralelnih vej tistega peralelnega vozlišča nivoja, ki ima največje število vej.

Gornje ugotovitve strnimo v preprost postopek, poimenovan *prekrivanje*. Z njim je že mogoče določiti

avtomat. Postopek prekrivanja na primeru grafa iz slike 1 prikazuje slika 2 in poteka takole :

1. postopek pričnemo izvajati na podgrafu, ki je določen s skrajnimi levimi sekvenčnimi vejami podgrafa (slika 2-I).
2. Preslikamo vozlišča v procesorje po naslednjem pravilu:
   - sekvenčna vejišča se preslikajo v procesorje, povezave med procesorji pa določajo paralelne povezave grafa
   - procesi (listi HADFG drevesa) se preslikajo v procesorje
   - če v avtomatni struktruri na mestu, kjer bi moral biti procesor slednjega še ni, ga dodamo (temnejši procesorji na sliki 2)
   - če ima avtomatna struktura več procesorjev, kot je potrebno, jih ne odvzemamo
3. Izberemo naslednjo sekvenčno vejo na najnižjem sekvenčnem nivoju (naslednji podgraf) in nadaljujemo od točke 2 dalje, dokler ni obdelan celoten graf (slike 2 II-IV).

Metoda prekrivanja avtomatne strukture z ustreznimi deli grafa, nam omogoča enostavno konstruiranje avtomatne strukture. Tako pridobljena avtomatna struktura popolnoma ustreza danemu grafu, vendar je v splošnem predimenzionirana.

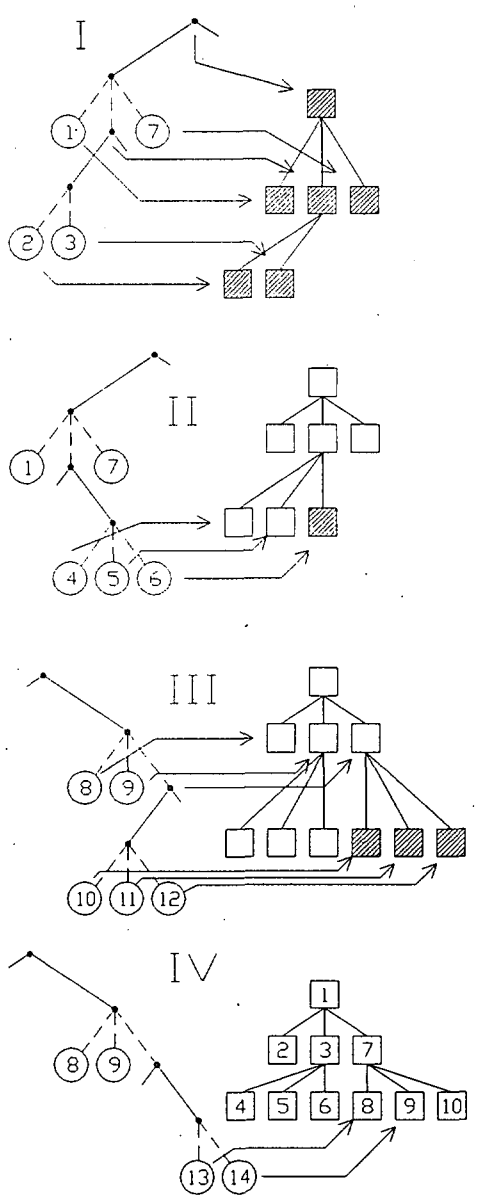
Ugotovimo lahko, da je možno graf ustrezno preoblikovati in realizirati funkcionalno enakovredno strukturo z manjšim številom procesnih elementov.

3.2 Urejeno prekrivanje

HADF graf ima v splošnem nesimetrično obliko. Nesimetričnost se odraža v različnih globinah podgrafov istega nivoja, in v različnem številu vej, ki izhajajo iz posameznih paralelnih vozlišč grafa. Vrstni red posameznih vej znotraj paralelnega vozlišča je funkcionalno nepomemben, saj se vse veje istega vozlišča izvajajo sočasno. Ta lastnost nam daje možnost, da veje med seboj kombinatorično permutiramo tako, da dosežemo kar najboljše prekrivanje nastajajoče drevesne avtomatne strukture z obravnavanim grafom. Originalni HADF najprej graf ustrezno preoblikujemo, nakar sestavimo avtomat z zgoraj opisanim prekrivanjem.

Uspešnost postopka se kaže v prostorski optimizaciji ATS in je neposredno odvisna od preoblikovanja grafa. Iterativne metode (npr. simulirano ohlajanje) bi lahko bile zelo uspešne, vendar potrebujejo za svoje izvajanje več časa. Analitični pristop je hitrejši in zaradi drevesne strukture enostavno izvedljiv, zato smo ga uporabili tudi mi.

Prvi korak do optimizacije je torej ureditev HADF grafa. Graf uredimo tako, da znotraj vseh paralelnih vozlišč razvrstimo skrajno levo najglobje in skrajno desno najplitvejše veje. V primeru enake globine



Slika 2: Postopek prekrivanja

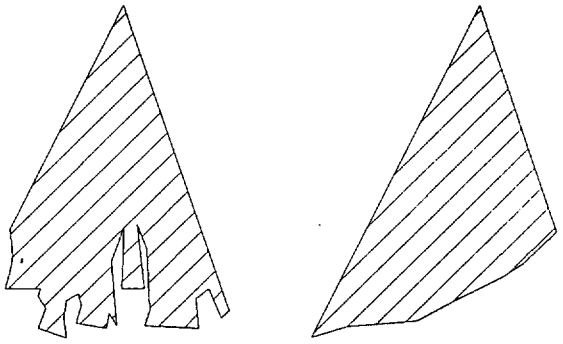
dveh vej ima prednost tista, ki ima večje število procesov na najnižjem nivoju. Opisani postopek prevede drevo iz neurejene v urejeno strukturo, kar lahko shematično prikažemo na sliki 3.

3.2.1 Algoritem urejenega prekrivanja

Algoritem deluje na principu utežitve vozlišč HADF grafa. Uteži, ki jih pripišemo vozliščem, so le računski pripomočki in nimajo nobene povezave s časovnimi utežmi grafa. Primerna utežitev nam olajša postopek urejanja grafa.

Predpostavke za HADF graf:

- začetno vozlišče HADF grafa je sekvenčno vozlišče, ki predstavlja ničti nivo v grafu ($l = 0$). V nasprotnem primeru govorimo o večih grafih, ki se lahko izvajajo paralelno.



Neurejena struktura Urejena struktura

Slika 3: Struktura grafa

- na koncih vej, ki izhajajo iz korenskega vozlišča, se nahajajo procesi ali paralelna vejišča. To je prvi nivo v grafu ($l = 1$).
- ko se spuščamo po grafu navzdol, se izmenjujejo sekvenčni in paralelni nivoji. Nivo v grafu se poveča z vsakim naslednjim vozliščem za ena ($l_{i+1} = l_i + 1$).

Utežitev listov in vozlišč grafa ter urejanje
Prvi, prioritetni kriterij pri urejanju grafa je globina. Širina predstavlja drugi kriterij. Utežitev vej mora zagotoviti globjim vejam večjo težo ne glede na širino primerjanih plitvejših vej. Uteži moramo izbirati tako, da ima utež na najnižjem nivoju večjo težo, kot je vsota vseh ostalih uteži na višjih nivojih.

$$g_n > \sum_{i=0}^{N-1} g_i$$

Predno začnemo uteževati posamezne liste, moramo določiti težo uteži za liste HADF grafa po posameznih nivojih. Največje število paralelnih vej N, ki izhajajo iz katerega koli paralelnega vozlišča v grafu, je v skladu z gornjo zahtevo in mora biti znano. Enakovredno bi ustrezalo tudi poznavanje največjega števila paralelnih vej po posameznih paralelnih nivojih, vendar zaradi poenostavitve postopka gledamo na HADF graf kot celoto.

Uteži določamo po naslednjem pravilu:

Utež q_0 ($l = 0$)
q_0 ... izberemo: $q_0 = 0$

Utež q_1 ($l = 1$)

$$q_1 > q_0$$

q_1 ... izberemo: $q_1 = 1$

Utež q_2 ($l = 2$)

$$q_2 > \sum_{i=1}^{N} q_1 = N \cdot q_1 = N$$

Najmanjša možna razlika za

$$q_2 > N \ldots q_2 = N + 1$$

Utež q_3 ($l = 3$)

$$q_3 > \sum_{i=1}^{N} q_2 = N \cdot q_2 = N(N+1) = N^2 + N = \frac{N^3 - 1}{N - 1} - 1$$

$$q_3 = q_2 + 1 = N^2 + N + 1 = \frac{N^3 - 1}{N - 1}$$

⋮

Utež q_k ($l = k$)

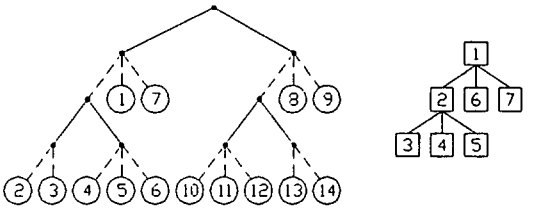
$$q_k > \sum_{i=1}^{N} q_{k-1} = N \cdot q_{k-1} =$$

$$= N(N^{k-2} + N^{k-3} + \cdots + N + 1) =$$

$$= N^{k-1} + N^{k-2} + \cdots + N^2 + N = \frac{N^k - 1}{N - 1} - 1$$

$$q_k = q_2 + 1 = N^{k-1} + N^{k-2} + N^2 + N + 1 = \frac{N^k - 1}{N - 1}$$

Pripisovanje uteži procesom HADF grafa
Ko so uteži določene, utežimo z njimi liste grafa (procese) v skladu z njihovo lego v grafu. Paralelne in sekvenčne procese obravnavamo po ločenih kriterijih.



Slika 4: Urejen HADF graf

Paralelni procesi

- obsegajo liste grafa na nivojih $l = 2k$; $k = 1, 2, \ldots$
- iz posameznega vejišča izhaja največ N takšnih procesov
- utež $q_n = q_n|_{n=\frac{1}{2}} = \frac{N^n - 1}{N - 1} \Longrightarrow$
  $\Longrightarrow q_1|_{l=2}, q_2|_{l=4}, \ldots q_k|_{4l=2k}$

Sekvenčni procesi

- obsegajo liste grafa na nivojih $l = 2k - 1 \; ; k = 1, 2, \ldots$
- število vej iz posameznega vejišča ni omejeno
- utež $q_n = q_{n|_{n=lDIV2}} = \frac{N^n - 1}{N - 1} \Longrightarrow$
  $\Longrightarrow q_{0|_{l=1}}, q_{2|_{l=3}}, \cdots q_{k|_{l=2k-1}}$

Pripisovanje uteži razvejiščem grafa
Paralelna razvejišča ($l = 2k - 1$)
Paralelnemu razvejišču pripišemo vsoto uteži vseh vej, ki iz njega izhajajo:

$$q_p = \sum_{i=1}^{m_p \leq N} q_i$$

Sekvenčna razvejišča ($l = 2k$)
Sekvenčnemu razvejišču pripišemo utež najtežje veje sekvenčnega razvejišča.

$$q_s = \max_i q_i$$

Ko vsem vozliščem grafa pripišemo ustrezne uteži, uredimo veje znotraj vseh paralelnih vozlišč po utežeh, od veje z največjo (levo) do veje z najmanjšo težo (desno).

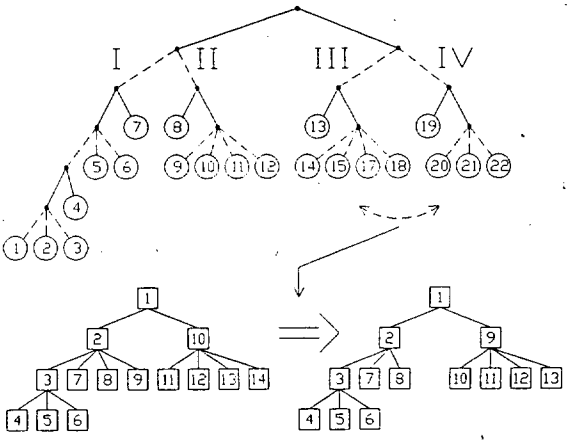
Avtomatno strukturo konstruiramo iz tako preoblikovanega grafa po enakem postopku kot prej, t.j. s prekrivanjem. Avtomatna struktura urejenega grafa je praviloma optimalnejša (manjše število procesorjev) kot tista, ki jo dobimo neposredno iz originalnega HADF grafa.

Rezultat generiranja avtomatne strukture s predhodnim urejanjem HADF grafa si oglejmo na primeru. Originalni par: neurejeni graf in pripadajoča avtomatna struktura je prikazan na sliki 2, urejeni par pa na sliki 4. Vidimo, da ima urejeni graf preprostejšo avtomatno strukturo.

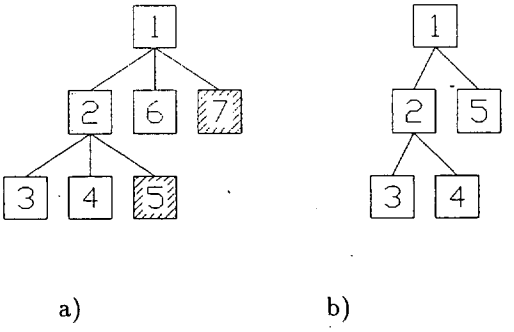
Opisani algoritem urejenega prekrivanja je dokaj enostaven, vendar vselej ne zagotavlja optimalnih rezultatov. Kljub temu menimo, da je dovolj uporaben za procesorje z omejenim številom zunanjih povezav pri postavki, da nastopa v paralelnih vozliščih HADF grafov praviloma več vej, kot je možnih povezav na procesnem elementu. Algoritem ima v primerjavi z iterativnimi algoritmi tudi manjšo kompleksnost in je zato cenejši.

Na sliki 5 si lahko ogledamo preprost primer grafa, pri katerem z opisanim postopkom urejanja ne dobimo idealne avtomatne strukture. Ugotovimo lahko, da bi bila ob zamenjavi paralelnih vej III in IV avtomatna struktura manjša za en procesor.

Funkcije posameznih procesorjev prikazuje slika 6. Iz slike je razvidno, da opravljajo vozliščni procesorji le funkcijo vmesnika in stikala, in so kot taki slabo izkoriščeni. Izkoriščenost le-teh povečamo, če enega izmed procesov (če obstaja) zadržimo v paralelnem vozlišču in ga izvajamo na vozliščnem procesorju. Takšen primer za že prej obravnavani HADF graf si oglejmo



Slika 5: Primer, kjer algoritem ni optimalen



a) b)

Slika 6: Krčenje avtomatne strukture

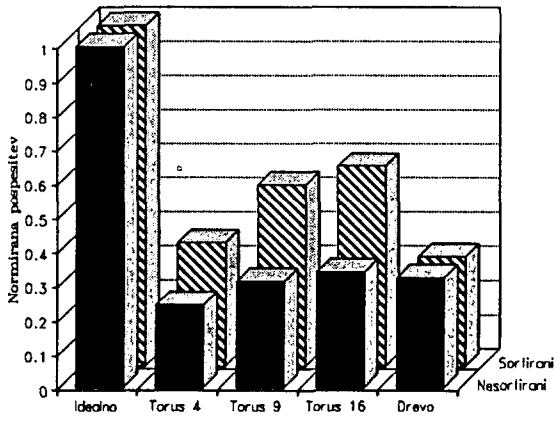
na sliki 6a. Procesorja 1 in 2 opravljata le funkcijo razvejišča. Ker sta neizkoriščena, smemo nanju preslikati procesa, ki se sicer izvajata na procesorjih 5 in 7. Rezultat je dodatno skrčenje avtomatne strukture, ki je v tem primeru manjša za 2 procesorja (slika 6b). Pridobljena avtomatna struktura še vedno ne upošteva časovnih značilnosti posameznih procesov in zadošča splošnemu primeru grafa dane oblike.

4. Izvajanje grafov

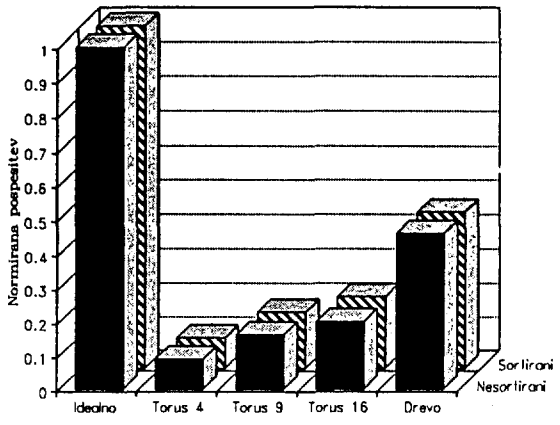
Regularna (drevesna) avtomatna struktura in neposredna preslikava HADF grafa nanjo, nam omogočata analitično oceno časa izvajanja. Ocena, ki smo jo napravili, je poenostavljena in daje boljše rezultate, kot bi jih dobili na resničnem sistemu, saj ne upošteva zmanjšanja moči procesiranja zaradi prenosov. Prispevek te napake ocenjujemo kot majhen, ker je intenzivnost prometa nizka (samo lokalni prenosi). Graf se bločno (koračno) širi po drevesni strukturi navzdol tako, da se od višjega na nižji nivo najprej prenese celotno poddrevo; delitev grafa na nižje nivoje se nadaljuje šele v naslednjem koraku. Avtomatna struktura mora biti prilagojena obravnavanemu grafu, zato pri preslikavi ne pričakujemo mrtvih točk (blokad).

5. Rezultati

Čas izvajanja. Pri opazovanju časov izvajanja posameznih grafov opazimo, da se lahko z obravnavanimi transformacijami drevesne strukture čas izvajanja HADF grafa izboljša v primerjavi s tistim časom, ki smo ga dosegli na toroidnem polju [2, 4]. Dosežena normirana povprečna pospešitev je prikazana na sliki 7. Sortirani grafi so prilagojeni za uspešnejše izvajanje na toroidnem polju [1] in ne vplivajo na potek izvajanja na drevesni strukturi avtomata. Eksperimentalno smo ugotovili, da so rezultati na drevesu boljši pri grafih, ki imajo močno poudarjen faktor širine. Pospešitev za primer takšnega grafa prikazuje slika 8. Drevesna avtomatna struktura je manj uspešna pri globokih grafih (slika 9). Največjo prednost drevesne strukture opazimo pri izvajanju popolnoma simetričnih grafov (slika 10). Grafi te vrste so simetrični tako po zgradbi, kot tudi po času prenosov in izvajanj posameznih procesov.

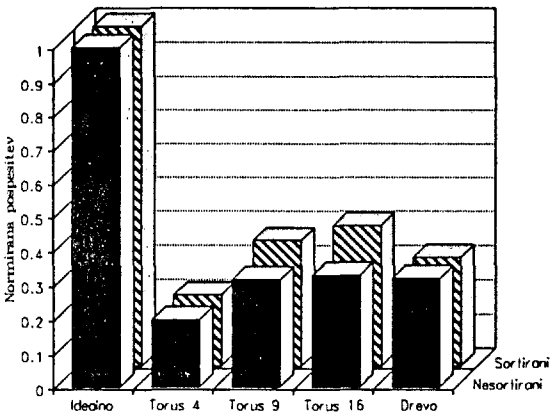


Slika 7: Primerjava drevesne in toroidne strukture

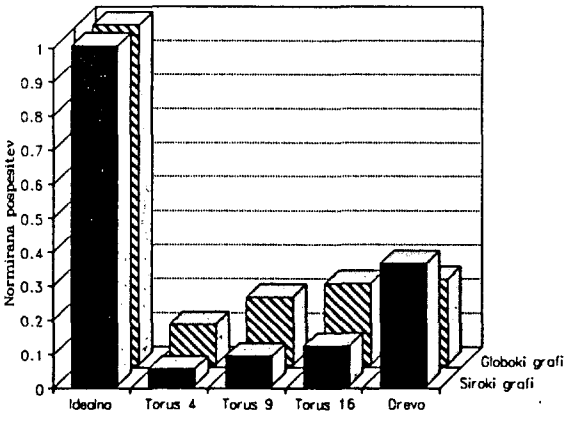


Slika 8: Primerjava širokih grafov

Količnik prenosnega časa. Vrednost potrebnega količnika prenosnega časa se pri drevesni strukturi giblje okoli vrednosti 20, kar je več, kot je veljalo za toroidno polje. Razmerje T_{izv}/T_p smo kljub temu ohranili na vrednosti 10, da bi bili rezultati, ki smo jih dobili za



Slika 9: Primerjava globokih grafov



Slika 10: Primerjava popolnoma simetričnih grafov

drevo primerljivi z rezultati, ki smo jih dobili za toroidni sistem. Ocenjujemo, da je povečanje količnika prenosnega časa posledica odsotnosti interakcije prenosov sporočil med procesorji. Značilnost drevesne avtomatne strukture je namreč lokalen prenos sporočil. Prenos se izvaja le med dvema procesorjema. Istočasne zahteve za prenos po isti poti, kot so možne pri torusu, pri drevesni strukturi niso možne.

6. Zaključek

Drevesna oblika HADF grafov nam je pomenila izziv za raziskavo, ki naj bi dala odgovor na vprašanje o upravičenosti izvajanja algoritmov, ki so predstavljeni v obliki HADFG na drevesni avtomatni strukturi. Rezultate raziskave smo primerjali z rezultati raziskave toroidnega polja [2, 4]. Ugotovili smo, da je drevesna struktura glede na čas izvajanja primernejša za razreda popolnoma simetričnih in širokih grafov.

Drevesna struktura je neprilagodljiva. Meje, ki jih predstavljajo terminalni procesorji, t.j. takšni procesorji, ki nimajo naslednikov, pomenijo meje širjenja grafa. Posledica je, da mora biti za neblokirano preslikavo grafa avtomatna struktura dovolj razvejana. Avtomat je tako lahko bodisi splošen (predimenzioniran), bodisi takšen, da ustreza družini izbranih HADF grafov

ali nekemu določenemu grafu. Splošna oblika avtomata je drevo zadostne širine in globine, da se lahko na njega preslika vsak želeni HADF graf. Povsem jasno je, da je takšen avtomat občutno predimenzioniran. Bolj upravičena je konstrukcija avtomata, ki pokriva potrebe določene izbrane družine HADF grafov. Avtomat bo najverjetneje za večino grafov še vedno predimenzioniran, vendar v znatno manjši meri. Namenski avtomat za nek določen graf je cenovno najboljša rešitev saj potrebuje najmanjše število procesorjev. Ozka omejenost na en sam tip grafa pa je cena, ki jo moramo plačati za takšno rešitev.

Problemi, na katere naletimo ob povezovanju procesorjev v drevesne strukture, so predvsem tehnične narave in se jim lahko do določene mere izognemo ob uporabi rekonfigurabilnega polja procesorjev. Statična polja so manj uporabna in bi bila upavičena kvečjemu za neprestano izvajanje istega grafa ali iste družine grafov.

Naslednji problem predstavlja stopnja povezljivosti procesnih elementov, ki je tehnološko omejena, medtem ko je število možnih naslednikov vozlišča v grafu neomejeno. Transputerji imajo npr. štiri povezave, kar ob enem predhodniku pogojuje le tri naslednike. Dinamično rekonfigurabilna polja omogočajo navidezno poljubno število naslednikov, vendar so tudi ta omejena z zmogljivostjo matričnih stikal.

Iz obravnavanega lahko zaključimo naslednje :

Toroidni sistem je najprimernejši za grafe, ki so nesimetrični po obliki in v katerih imajo procesi različne čase izvajanja. Dekompozicija grafa po korenskih procesorjih (glej[2]) praviloma zagotavlja krajši skupni čas prenosov in izvajanja ter občutno manjše zahtevano število procesorjev, kot drevesna struktura. Toroid si predstavljamo kot vase zapognjeno drevo, ki se navidezno nikoli ne konča. Vsi procesorji prvega procesorskega nivoja[1] predstavljajo primarne, procesorji drugega nivoja pa sekundarne potencialne naslednike. Tu lahko potegnemo vzporednice s kompleksnejšo rekonfigurabilno strukturo. Navidezna neskončnost strukture pomeni hkrati tudi večjo univerzalnost.

Čeprav je število navideznih naslednikov procesorja veliko, lahko sočasno dostopamo le do dveh. Pri prenosu sporočil ima namreč vsak procesor v torusu le dva logična naslednika, kar omejuje komunikacijske zmožnosti. Kot vemo, se sporočila prenašajo krožno po zankah torusa. Ker prenos ni neposreden, so pri svojem delu dodatno moteni tako vsi posredniki, kot tudi prenosne poti. Posledica je, da takšen sistem zelo dobro deluje z nesimetričnimi HADF grafi, kjer so prenosi razdeljeni neenakomerno. Izvajanje širokih in popolnoma simetričnih grafov je manj uspešno, ker slednji zahtevajo za učinkovito delovanje simetrično in sočasno razporejanje.

Predstavljena transformacija HADF grafa v avtomatno strukturo je analitična in temelji na predpostavki, da imajo procesorji neomejene zmožnosti povezovanja, in je kot taka idealizirana. Transformacija v splošnem ne zagotavlja idealnih rezultatov, vendar menimo, da je glede na nizko stopnjo kompleksnosti dovolj uporabna. Nadalje menimo, da bo potrebno večino HADF grafov ustrezno preoblikovati še predno skonstruiramo avtomatno strukturo in jih prilagoditi stopnji povezljivosti procesorjev. Preoblikovanje izvedemo tako, da odvečne paralelne veje, t.j. tiste, ki prekoračijo število razpoložljivih povezav, potisnemo na nižji nivo v grafu. Tako preoblikovan graf bo predvidoma na vseh višjih nivojih avtomatne strukture popolnoma povezan, zato se situacije, kot je podana na sliki 5, ne bodo pojavljale.

Idealizirane drevesne strukture so se pokazale kot primerne za popolnoma simetrične in široke HADF grafe. Takšni grafi so značilni za rekurzivne algoritme [5]. Naše nadalnje delo bo usmerjeno v raziskavo statičnih drevesnih struktur z realnimi procesorji, z možnostjo preslikav algoritma nanje in uspešnosti takšnih struktur. Pričakujemo, da se bodo rezultati po času izvajanja približali rezultatom, ki jih dosegamo na toroidnem polju, pri čemer dopuščamo slabšo učinkovitost drevesne strukture.

Literatura

[1] P.Kolbezen, P.Zaveršek, A hierarchical multimicroprocessor system, Informatica, A Journal of Computing and Informatics, The Slovene Society Informatika, Vol.15, Nr.1, Feb. 1991, 65-76.

[2] P.Zaveršek, P.Kolbezen, Process allocation in the multitransputer network, Informatica, A Journal of Computing and Informatics, The Slovene Society Informatika, Vol.15, Nr.4, Nov. 1991, 43-52.

[3] M.Szturmowicz, M.Tudruj, A multi-layer transputer network for efficient execution of OCCAM programs, North-Holland, Microprocessing and Microprogramming, Vol.28 (1989), 133-138.

[4] P.Zaveršek, P.Kolbezen, Dynamic allocation on the transputer network, Proceedings of the CONPAR 92-VAPP V, Lyon, France, 1992 (in print).

[5] B.Buchberger, J.Fegerl, F.Lichtenberger, Computer trees: a concept for parallel processing, IPC Business Press, Microprocessors and Microsystems, Vol.3 No.6, 1979, 244-248.

[6] Parsytec Transputer Modules, ParsyTec, Gesellschaft für Parallele Systemtechnik mbH.