INFORMATICA 2/1984

UPORABA PROGRAMSKIH GRAFOV PRI UGOTAVLJANJU VZPOREDNOSTI V RAČUNALNIŠKIH ALGORITMIH I.

B. DŽONOVA–JERMAN,
J. ŽEROVNIK

UDK: 519.698

INSTITUT JOŽEF STEFAN, LJUBLJANA

POVZETEK. Prikazane so metode za merjenje vzporednosti v računalniških programih, ki za analizo algoritmov uporabljajo programske grafe. Časovne enačbe za vzporedno in zaporedno izvajanje računalniških ukazov so konstruirane za dva modela računalniške arhitekture krmiljene s pretokom podatkov. Razmerje med enačbami je uporabljeno kot merilo in ocena o obstoječih vzporednostih v računalniškem algoritmu. V kratkem so podane osnovne značilnosti računalnikov krmiljenih s pretokom podatkov.

ABSTRACT. "Application of program graph analysis for measurement of parallelism in computer algorithms".
Techniques of program graph analysis are used to measure the parallelism in computer programs. For a given semantic model of architectural support, characteristic timing equations are first constructed from the high level program to describe the sequential and parallel execution times. The ratio of these equations is then used as a measure of the inherent parallelism in the program. Graph analysis techniques are illustrated using two data flow models of architectural support. The basic feature of data flow computers are described very briefly.

1. UVOD

Sodobne smeri razvoja računalniške arhitekture imajo za cilj: povečati računalniško moč sodobnih računalnikov, kar pomeni povečati njihovo zmogljivost in hitrost računanja ter predlagati rešitve, ki odpravljajo ozka grla v procesu računanja in so rezultat klasične (Von Neumannove) organiziranosti oziroma klasične zgradbe obstoječih računalniških sistemov.

Večjo računalniško zmogljivost potrebujemo zaradi naraslih potreb po reševanju problemov s področja umetne inteligence, obdelave slik, razpoznavanja govora, napovedovanja vremenskih situacij, avtomatskega prevajanja jezikov, analize seizmografskih podatkov, obdelavah multidimenzionalnih sistemov in podobno. Ocenjeno je (1), da napoved 24 urne vremenske situacije zahteva 100 milijard operacij na sek. Obdelava slike, ki ima 512 x 512 slikovnih elementov in potrebuje do 100 ukazov za slikovni element, zahteva 656 milijonov operacij na sec. (1).

Obdelava in reševanje takšnih in podobnih problemov z zadovoljivo hitrostjo je mogoče le ob uporabi računalnikov, ki omogočajo vzporedno izvajanje računalniških ukazov. Dosedanje izkušnje so pokazale, da so možnosti vzporednega izvajanja računalniških ukazov na računalnikih z Von Neumannovo arhitekturo dokaj izčrpane. Von Neumannov model računalnika ima dve bistveni značilnosti: spomin za hranjenje programov in podatkov in števec ukazov, čigar vsebina določa ukaz, ki se bo izvajal. Von Neumannov model računalnika je imel velik vpliv na razvoj programskih jezikov. Jeziki pri katerih je vpliv te arhitekture najbolj opazen tvorijo razred Von Neumannovih jezikov. V ta razred programskih jezikov so uvrščeni FORTRAN, ALGOL 68, BASIC ipd. Med sodobnimi programskimi jeziki obstajajo jeziki, ki so bolj funkcionalno zastavljeni in ki so visoko uvrščeni v hierarhijski lestvici programskih jezikov. V programih, pisanih v teh jezikih, je zaradi dovoljenih sintaksnih konstruktov jezika značilna prisotnost notranje oziroma implicitne vzporednosti (inherent parallelism).

Namen našega prispevka je predstaviti dve metodi za ugotavljanje vzporednosti v obstoječih računalniških programih. Obe metodi uporabljata tehniko predstavitve programov s programskimi grafi. Programski grafi so model s katerim predstavljamo potek obdelav v računalnikih krmiljenih s pretokom podatkov (data flow computer). Pri analizi s pomočjo programskih grafov uporabljamo dva semantična modela arhitekture računalnika krmiljenega s pretokom podatkov. Prvi model je model arhitekture s povratno zanko krmilnih signalov, ki opravlja operacije le nad elementarnimi tipi podatkov in individualnih strukturnih komponent. Drugi model se razlikuje od prvega v tem, da so dodane operacije z vektorji.

2. OSNOVNE LASTNOSTI RAČUNALNIKOV KRMILJENIH S TOKOM PODATKOV

Računalniki krmiljeni s tokom podatkov predstavljajo temeljito spremembo v arhitekturi glede na Von Neumannovo arhitekturo računalnika. Pri Von Neumannovem računalniku potek obdelave poteka po natančno določenem zaporedju ukazov, ki tvorijo program. Spomin je enodimenzionalen in naslavljanje je zaporedno. Med ukazi in podatki ni bistvenih razlik. Semantika podatkov tudi nima nobenega pomena. Krmilni tok je opredeljen s števcem ukazov, ki skrbi za tekočo oskrbo procesne enote z ukazi. Povezava med spominom in procesno enoto po kateri potujejo ukazi je v obliki zaporedja je največja omejitev Von Neumannovega modela. Uvajanje novih organizacijskih rešitev v računalniški arhitekturi so pripeljale do izredno velike zmogljivosti sodobnih računalniških sistemov. Med te rešitve, ki bistveno niso spremenile osnove Von Neumannovega modela računanja so se zlasti pokazale za uspešne sledeče:

- cevasta organizacija aritmetično - logičnih enot
- sistemi z več aritmetično - logičnimi enotami
- matrični procesorji,
- sistemi z elementi za povezovanje večkratnih funkcionalnih enot, kot so vodila za porazdeljeno obdelavo, mreže s strukturo obroča ipd.

- priročni spomini (caché memory)
- navidezni spomin (virtual memory).

Izjemna zmogljivost nekaterih sodobnih računalnikov kot je na primer IBM 360/91, CDC 6600, CRAY-1 sloni na vzporednem delovanju različnih funkcionalnih enot (procesor-spomin, spomin-spomin, procesor-vhodno/izhodne enote, procesor-procesor z vodenjem tokov podatkov). Procesne enote teh računalnikov imajo več izvajalnih enot in mehanizmov za dekodiranje in sprejemanje ukazov, ki omogočajo neprekinjen tok ukazov oziroma zmeraj poln vod skozi katerega ukazi pojejo. Pri IBM 360/91 se ukaz s plavajočo vejico najprej delno dekodira in šele zatem pošlje enoti s plavajočo vejico. Enota dokonča dekodiranje in pošlje ukaz izvajalni enoti. Izvajanje aritmetičnih operacij se prične šele takrat ko pridejo vsi potrebni operandi (iz spomina ali iz drugih enot). Dekodiranje ukazov poteka zaporedno ne glede na to, da je izvajanje nekaterih možno brez upoštevanja tega zaporedja. Zaporedno dekodiranje ukazov zagotavlja logično pravilnost rezultatov. Računalnik CRAY-1 (6) vzporedno izvaja ukaze ob uporabi matrično organiziranih izvajalnih enot. Zmogljivost računalnika pri izvajanju vektorskih ukazov je omejena le s kapaciteto spomina. Ocenjeno je (2), da se odstotek programske opreme, ki se lahko napiše v skladu z možnostmi, ki jih nudijo matrično organizirane procesne enote giblje od 1 do 90 % na področju znanstvenih aplikacij. Ostali del programske opreme pa lahko postane ozko grlo v kolikor učinkovitost vektorskih ukazov izjemno zraste. Programi pisani v Fortranu težko izrabljajo zmogljivost računalnikov kot je CRAY-1. Pisanje programov zahteva zelo dognano analizo podatkovnih tokov. Analiza podatkovnih tokov v programih pisanih v Fortranu je izjemno težka zaradi stranskih učinkov, ki jih povzročajo GO TO stavki in povezovanje spremenljivk. Razširitve v Fortranu lahko zmanjšajo te probleme, pri tem pa programiranje postane zahtevno, programi pa neprenosljivi in preveč odvisni od stroja. Uporaba več Von Neumannovih procesorjev, ki si delijo skupni spomin je podoben pristop. Poglavitni problem pri takšni računalniški arhitekturi je zagotovitev podpore v materialni opremi, ki bo skrbela za pravilnost rezultatov. Namreč, kosi programa, ki se izvajajo vzporedno in si delijo podatke, se sinhronizirajo s testnimi procedurami, (test and end), s semaforji, sporočili ipd. Nadzor sinhronizacijskih operacij je zahteven in zadostno zmogljivost pri obdelavi dosežemo le, če je izvajanje razdeljeno na večje kose programa, ki so sinhronizirani z zelo majhnim številom operacij. Razdelitev programa v takšnih kosih, ki bodo zagotavljali tudi pravilnost rezultatov obdelave, je zelo težka naloga.

Vsa dosedanja prizadevanja za izboljšanje zmogljivosti sodobnih računalnikov zgrajenih na osnovi Von Neumannovega modela računanja so pokazala, da so možnosti precej izrabljene. Klasična organizacija računalnikov ne omogoča zadostno izkoriščanje možnosti, ki jih ponuja nova tehnologija integriranih vezij (LSI, VLSI). Izhod iz nastale situacije išče strokovna javnost v različnih smereh razvoja računalniške tehnologije med katerimi imajo najbolj pomembno mesto zaradi svojega revolucionarnega koncepta računalniki krmiljeni s pretokom podatkov.

Arhitektura računalnika krmiljenega s pretokom podatkov ponuja rešitve za večino pomanjkljivosti klasično organiziranih računalnikov. Struktura teh računalnikov je zasnovana na podlagi naslednjih konceptov:

- ni spremenljivk in obdelava poteka le s konkretnimi vrednostmi,
- potek obdelave je krmiljen s pretokom podatkov in števca ukazov,
- ni shranjevanja podatkov v klasičnem smislu oziroma ni spomina z naslovi.

Programe krmiljene s pretokom podatkov opisujemo z usmerjenimi grafi. Elementi grafa ponazarjajo pretok podatkov med posameznimi ukazi. Vozlišča grafa ponazarjajo ukaze, usmerjene povezave med vozlišči pa predstavljajo vhod in izhod podatkov pred in po izvajanju ukaza. Usmerjene povezave so nosilke dejanskih vrednosti podatkov na vhodu in na izhodu iz vozlišča. Gibanje podatkov skozi usmerjeni graf predstavlja izvajanje programa s katerim smo ponazorili določeni algoritem.

Ukazi nimajo stranskih učinkov na podatke in jezik zgrajen na podlagi koncepta krmiljenja s pretokom podatkov vsebuje omejitve, glede zaporedja izvajanja ukazov, ki izhajajo naravnost iz odvisnosti podatkov v zapisanem algoritmu. Vsak program napisan v visokonivojskem jeziku s konceptom krmiljenja s pretokom podatkov lahko zapišemo v obliko grafa. Graf je podoben grafom, ki jih dobimo iz prevajalnika za Fortran po analizi podatkovnega toka. Prednost jezika s konceptom krmiljenja s pretokom podatkov je v tem, da graf o pretoku podatkov dobimo zelo enostavno in hitro. Procesne enote krmiljene s pretokom podatkov so dejansko računalniki v katerih je shranjen graf, ki ponazarja pretok podatkov v programu. Procesna enota razpozna kateri ukazi so pripravljeni za izvajanje in te ukaze izvajalna enota obdela takoj ko dobi sporočilo o tem, da so določeni viri razpoložljivi. V primerih, ko je razpoložljivih računalniških virov zadosti, računalnik izrabi vse možnosti vzporedne obdelave v programu.

Za implementacijo računalnikov krmiljenih s pretokom podatkov je bilo predlaganih več različnih inačic materialne opreme. Vsi predlogi uporabljajo kot osnovo model usmerjenega grafa za predstavljanje računalniških programov. Zato je to vrsta računalnika, ki ima jezikovno zasnovan koncept arhitekture in kot programski jezik programske grafe. Razlike med posameznimi arhitekturami računalnikov obstajajo v implementaciji konceptov programskih grafov na strojnem nivoju.

3. MODEL RAČUNALNIKA KRMILJENEGA S PRETOKOM PODATKOV

V metodah za merjenje vzporednosti v računalniških programih, ki jih bomo predstavili v nadaljevanju, je uporabljen model, ki sta ga predložila Martin in Estrin (4,5). V tem modelu ima vsako vozlišče usmerjenega grafa sledečo strukturo:

- polje ukazne kode, ki opredeljuje ukaz,
- dvoje ali več polj za sprejem operandov,
- eno ali več polj, ki opredeljujejo kam rezultati potujejo po izvršitvi ukaza.

Ukaz v vozlišču se izvrši šele takrat, ko so vsi potrebni podatki prisotni na vhodu v vozlišče. V času izvajanja vzame operator podatke iz vhoda, jih obdela ter jih postavi na izhod iz vozlišča. Za ta model obdelave je značilno, da se vozliščem direktno posredujejo parcialni rezultati obdelave. Izvajanje ukaza lahko vpliva le na ukaze, ki mu sledijo. Vsi ukazi imajo pomen funkcij. Po uporabi podatka se njegova vrednost izgubi ker ta podatek ni več na razpolago. Koncepta delitve podatkov v spominu v tem modelu ni. Tudi krmilnega toka, ki šteje in nadzoruje izvajanje ukazov ni; ukazi se izvajajo takrat ko so na vhodih v vozlišča prisotni vsi zahtevani podatki. To pomeni, da se ukazi izvajajo neodvisno in da obdelava poteka vzporedno. Seveda je pri tem nujno, da je na razpolago zadostno število virov, drugače vzporedna obdelava ne bo mogoča. Krmilni tok poteka obdelave je opredeljen s tokom podatkov, oziroma z odvisnostmi, ki obstajajo med podatki. Izvajanje programa se lahko konča asinhrono in porazdeljeno. Prednosti tega pristopa lahko strnemo v naslednjem:

- mogoča je vzporedna obdelava
- verifikacija programov je enostavnejša in lažja,
- mogoča je večja modularnost programov,
- širjenje strojne opreme je relativno enostavno
- problemi v zvezi z zaščito podatkov so manjši in
- lažji je nadzor nad napakami v programu.

Poglavitna razlika med klasičnim načinom izvajanja obdelav in med tem novim pristopom je v organizaciji poteka obdelav: Pri sistemih, ki delujejo po principu pretoka podatkov, poteka obdelava v treh fazah: izbor ukaza, pregled pogojev potrebnih za izvajanje ukaza (prisotnost operandov) in samo izvajanje ukaza. V prvi fazi poteka izbor ukaza po nekem v naprej opredeljenem pravilu. Sam izbor ukaza še ne pomeni, da bo prišlo takoj tudi do izvajanja ukaza, saj se šele v drugi fazi odloča o tem ali bo ukaz sprožen ali ne. V kolikor na vhodu v določeno vozlišče ni potrebnih operandov, se ukaz ne izvede. Pri klasični obdelavi je izbor ukaza izvršen na podlagi števca programskih ukazov. Izbran ukaz se avtomatsko tudi izvede, zato faze v kateri se operandi pregledujejo tukaj ni. Prednost obdelave, ki jo nudi koncept pretoka podatkov, je v možnosti visoke stopnje vzporednosti obdelave. Pomanjkljivost je v tem, da lahko prihaja do pogostega čakanja na vhodu v vozlišče zaradi neprisotnosti potrebnih argumentov, če ni dobro izbran algoritem glede na arhitekturo računalnika.

4. MODEL RAČUNALNIKA KRMILJENEGA S PRETOKOM PODATKOV IN USMERJENI PROGRAMSKI GRAFI S POVRATNO ZANKO

Usmerjeni programski grafi so modeli s katerimi predstavljamo izvajanje programov v računalnikih krmiljenih s pretokom podatkov. Graf je sestavljen iz vozlišč in usmerjenih povezav. Vsako vozlišče predstavlja določen korak v izvajanju programa. Najbolj razširjen model programskega grafa je t.i. model s povratno zanko. Ta model je zasnovan na vozliščih v katera prihaja najmanj ena usmerjena povezava in izhaja najmanj ena usmerjena povezava. Vhodne povezave hranijo operande za ukaz, ki ga vozlišče predstavlja, izhodne povezave pa hranijo parcialne rezultate. Semantika podatka je dana s povezavo, kar pomeni, da je v vsaki povezavi prisoten le en podatek oziroma operand. Model usmerjenega programskega grafa je asinhron in deterministične. Več vozlišč se lahko izvaja simultano in časovno neodvisno, torej asinhrono, prisotnost le enega podatka v usmerjeni povezavi pri vhodu in izhodu iz vozlišča pa govori o njegovi deterministični naravi. Poleg povezav s podatki so na vhodu v vozlišče lahko prisotni tudi drugi tipi povezav. Te povezave vsebujejo krmilne signale povratne zanke (feedback signals). Krmilne povezave imajo lahko le boolove vrednosti, ki kažejo na to ali je vozlišče prosto ali ne. Vrednost podatkov v vhodni povezavi se hrani toliko časa, da vozlišče postane prosto, kar pomeni da je krmilna povezava dobila status "prosta". Model usmerjenih programskih grafov omogoča vizualno ugotavljanje vzporednosti v računalniških algoritmih. Za ugotavljanje t.i. začasne vzporednosti, ki kaže potovanje podatkov skozi več vozlišč, si pomagamo s časovnimi enačbami. Kako to poteka bomo pokazali v podpoglavjih, ki sledijo.

5. UPORABA USMERJENIH PROGRAMSKIH GRAFOV ZA UGOTAVLJANJE VZPOREDNOSTI V RAČUNALNIŠKIH ALGORITMIH

Za znano arhitekturo procesne enote in podani algoritem konstruiramo programski graf na podlagi modela, katerega sta prva razvila Martin in Estrin. Model opredeljujejo množica vozlišč N (množica osnovnih operacij) in množica usmerjenih povezav A (ponazarja podatkovne odvisnosti). Preslikava T iz N v množico realnih števil podaja čas izvajanja vsake operacije v vozlišču n, ki pripada množici N. Funkcija B podaja oceno verjetnosti o prisotnosti rezultata na enem od izhodov iz vozlišča n. Če je n vozlišče z več izhodi, to pomeni, da tekom obdelave v tem vozlišču prihaja do razvejitev poteka obdelave. Če sta znani preslikava T in funkcija B, potem je graf določenega algoritma časovno ovrednoten. Časovno ovrednoteni graf pišemo kot urejeno četvorko: $G = (N, A, T, B)$. Graf G uporabimo za konstrukcijo karakterističnih časovnih enačb za vzporedno in zaporedno izvajanje algoritma. Vzporedno izvajanje je mogoče, če ni podatkovne odvisnosti med posameznimi vozlišči, zaporedno izvajanje pa se nanaša na izvajanje z več medsebojno povezanih generacij vozlišč. Enačbe konstruiramo kot funkcije števila iteracij v zankah in ocene verjetnosti realizacije posameznih vej v algoritmu. Pri konstrukciji enačb si pomagamo z algoritmom, ki prehodi vse poti v programskem grafu (7). V kolikor so nam razpoložljivi računalniški viri znani, potem nam ulomek med enačbo za zaporedno in vzporedno izvajanje da oceno o notranji vzporednosti v analiziranem programu. Natančnost ocene je odvisna od tega, kako natančno lahko v modelu predstavimo obnašanje osnovne računalniške arhitekture procesne enote in do katere mere je bila pri konstrukciji programa upoštevana možnost vzporedne obdelave.

5. ANALIZA VZPOREDNOSTI V RAČUNALNIŠKIH ALGORITMIH

5.1. Model 1

Vozlišča usmerjenega programskega grafa v modelu 1 so vozlišča, ki imajo povratno zanko (8). Do izvajanja ukaza v vozlišču pride takrat, ko je vsebina operandov znana in ko so vozlišča, ki sprejemajo izhodne veličine, ta sprejem tudi potrdili. Osnovne operacije na strojnem nivoju v Modelu 1 so razvrščene v 5 kategorij:

- operacije nad elementarnimi podatki (unarne in binarne operacije, vključno s trigonometrijskimi funkcijami in predikati),
- zaporedne operacije nad datotekami (read, readedit, write, writeedit),
- operacije za uporabo procedur in sinhronizacijo (identity, merge),
- operacije nad strukturami (append, select),
- operacija "apply" za predstavljanje funkcij v obliki podatkov (na primer ko vozlišče n posreduje konstanto, ki je rezultat poteka obdelave skozi graf, ki je v programskem grafu ponazorjen z vozliščem n).

Zgledi algoritmov, ki jih bomo analizirali z modelom 1, vsebujejo enostavne podatkovne strukture kot npr. polja. Ne glede na dimenzijo preslikamo polja najprej v enonivojsko strukturo. Meje polja opredeljene v deklaraciji uporabljamo za izračun indeksov in za opredelitev elementov v spominu. Lokacija v spominu vsebuje bodisi odgovarjajočo vrednost elementa ali kazalec, ki kaže na shranjeno vrednost podatka. Kadar kličemo kakšen element iz polja, izračunamo najprej vrednost selektorja z uporabo standardnih aritmetičnih operacij in z uporabo operacije "append" ali "select" (8).

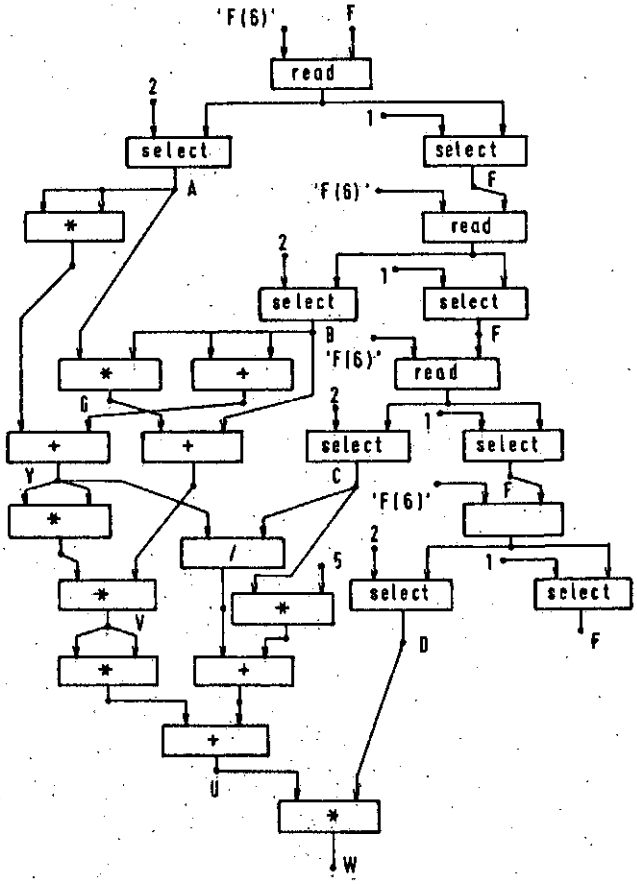
Zaporedne datotečne operacije podpirajo formatirane vhodno/izhodne ukaze. Pri ukazu "read" se najprej prebere zaporedje znakov z vrha datoteke, zatem se to zaporedje preoblikuje v notranji tip podatkov oziroma v strukturo, ki ima poleg vrednosti podatka še datotečni kazalec. Ukaz "write" poteka enako, le v obratni smeri. Ukaza "readedit" in "writeedit" podpirajo druge operacije. Ukaz "apply" zahteva za operande ime procedure in strukturo argumentov. Struktura argumenta je že vnaprej prirejena z ukazom "append". Proceduro najprej porušiti to strukturo z uporabo ukaza "select". Konec procedure pa vsebuje ukaze, ki konstruirajo strukturo rezultata. Ukaz "merge" uporablja vrednosti iz dveh virov in omogoča izvajanje konstruktov višjega nivoja le z enim vhodom in enim izhodom, kot so na primer pogojni stavki in zanke. Operacija "identity" skrbi za zamenjavo vrednosti.

Slika 1 nam ponazarja potek analize vzporednosti v algoritmih na konkretnem zgledu ob uporabi programskih grafov in ob upoštevanju računalniške arhitekture modela 1. Na sliki 1a je ponazorjen del programa v visokem programskem jeziku, na sliki 1b pa časovno ovrednoteni programski graf. Zaradi enostavnosti smo vsakem vozlišču oziroma vsaki operaciji do-

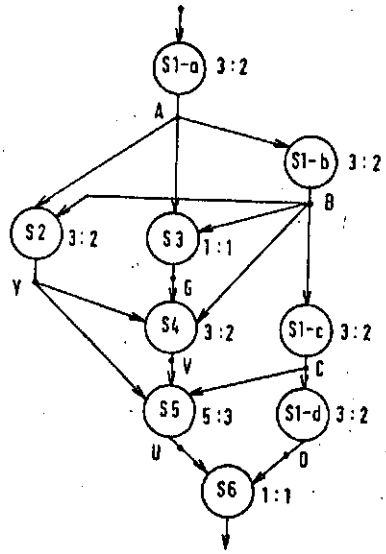
delili le eno časovno enoto. Čas vzporednega izvajanja je enak maksimalni dolžini poti v grafu pod pogojem, da imamo odgovarjajoče procesne enote. Graf iz slike 1b ima 32 podatkovno odvisnih povezav, ki so povezane s 25 vozlišči. Čas zaporednega izvajanja programa je tsec.(G) = 25, ker imamo 25 operacij. Čas vzporednega izvajanja je tpar(G) = 11, ker je globina grafa enaka 11. Oceno vzporednosti dobimo z ulomkom tsec/tpar in znaša za naš primer 2,3. Takšna analiza je precej naporna, zlasti če želimo analizirati velike programe ali kompleksne algoritme.

```
S1  input A, B, C, D, file = F
                format = 4F (6);
S2  Y:=A*A+B+B;
S3  G:=A*B;
S4  V:=Y*Y*(G+B);
S5  U:=V*V+Y/C+C*5;
S6  W:=U*D;
```

Sl. 1a. Del programa v visokonivojskem jeziku



Sl. 1b. Programski graf na strojnem nivoju
$\tau_s(G) = 25$, $\tau_p(G) = 11$, $\tau_s(G)/\tau_p(G) = 2,3$



Sl. 1c. Poenostavljena predstavitev programskega grafa
$\tau_s(G') = 25$, $\tau_p(G') = 12$, $\tau_s(G')/\tau_p(G') = 2,1$

5.2. Modifikacija modela 1

Analiza algoritmov z modelom 1 je zelo zamudna, zato model modificiramo tako, da zmanjšamo število potrebnih vozlišč. Najprej definiramo novo množico vozlišč N'. Množico N' uporabimo za predstavitev izrazov, prireditvenih stavkov, klica in izvajanje procedur (vključno s konstrukcijo argumentov), vhodno/izhodnih ukazov ipd., tako kot se ti pišejo v visokih programskih jezikih. Množica povezav A' pa nam ponazarja vrednosti podatkov med vozlišči iz množice N'. Na sliki 1c je prikazan usmerjeni programski graf G' = (N', A', T, B) z modificiranim modelom 1 za del programa iz slike 1a. Izračun časov vzporednega in zaporednega izvajanja zahteva analizo 15 povezav in 9 vozlišč. Časovno funkcijo določimo na enak način, torej kot ulomek tsec/tpar. Čas zaporednega izvajanja je enak vsoti časov vseh vozlišč. Čas potreben za izvajanje operacije v vozlišču dobimo kot vsoto vseh potrebnih operacij na nivoju stroja, ki se izvajajo v vozlišču. Tako izračunani čas ne prinaša nobene napake. Čas vzporednega izvajanja je enak globini drevesa, ki ga zgradi prevajalnik za operacije, ki se izvajajo v posameznem vozlišču. S to metodo naredimo napako pri oceni vzporednega izvajanja glede na celotni programski graf, ker poenostavljeni graf ne prikaže delnih prekrivanj, do katerih prihaja v podatkovno odvisnih vozliščih, kot so S3, S4 in S5.

REFERENCE

S. Ribarič, "Računari upravljani tokom podataka", Informatica, 6, No 4, 3 (1982).

T. Agervala, M. Arvind, Data flow systems computer, 15, 2 1982, 15.

A. Davis, R. Keller, Data flow program graphs, computer, 15, 2, 1982, 26.

D. Martin, G. Estrin, Models of computational systems, IEEE Trans. on Computers, 16, 1, 1967, 70.

D. Martin, G. Estrin, Path length computations on graph models of computation, IEEE Trans. on Comp., 18, 6, 1969, 530.

R. Russel, The Cray-1 Computer system, CACM, 21, 1, 1978, 63.

(II. del dela bo objavljen v eni od naslednjih številk časopisa)

INFORMATICA 3/1984

UPORABA PROGRAMSKIH GRAFOV PRI UGOTAVLJANJU VZPOREDNOSTI V RAČUNALNIŠKIH ALGORITMIH II.

B. DŽONOVA–JERMAN,
J. ŽEROVNIK

UDK: 519.698

INSTITUT JOŽEF STEFAN

POVZETEK. Prikazane so metode za merjenje vzporednosti v računalniških programih, ki za analizo algoritmov uporabljajo programske grafe. Časovne enačbe za vzporedno in zaporedno izvajanje računalniških ukazov so konstruirane za dva modela računalniške arhitekture krmiljene s pretokom podatkov. Razmerje med enačbami je uporabljeno kot merilo in ocena o obstoječih vzporednostih v računalniškem algoritmu. V kratkem so podane osnovne značilnosti računalnikov krmiljenih s pretokom podatkov.

ABSTRACT. "Application of program graph analysis for measurement of parallelism in computer algorithms".
Techniques of program graph analysis are used to measure the parallelism in computer programs. For a given semantic model of architectural support, characteristic timing equations are first constructed from the high level program to describe the sequential and parallel execution times. The ratio of these equations is then used as a measure of the inherent parallelism in the program. Graph analysis techniques are illustrated using two data flow models of architectural support. The basic feature of data flow computers are described very briefly.

5.3. Uporaba modela 1 pri vrednotenju konstruktov višjega nivoja

Model 1 in metoda časovnega vrednotenja programskega grafa omogočata vrednotenje konstruktov višjega nivoja kot so zanke in pogojni stavki. Pri tem je najpomembnejše, da so ti konstrukti primerno strukturirani. Časovne enačbe za določene operacije so ponazorjene skupaj z vozlišči, ki te operacije predstavljajo v usmerjenem grafu na sliki 2. Aciklično strukturo grafov zgradimo tako, da vozlišča, v katerih prihaja do zaprtih krogov, transformiramo v tranzitivno obliko (4). Oblika karakteristične enačbe je enako ne glede na to ali gre za vzporedno ali zaporedno izvajanje programa. Oznaka t se nanaša na čas potreben za izvajanje strojnega ukaza. Konstrukti s slike 2 omogočajo izračun karakteristične časovne enačbe zaporednega izvajanja za celotni program z enostavnim seštevanjem časov posameznih vozlišč. Karakteristično enačbo vzporednega izvajanja konstruiramo s pomočjo konstruktov iz slike 2 in sledečega pravila: če so vozlišča n1, n2, ..., nk podatkovno neodvisna potem je njihov zbirni vzporedni čas izvajanja enak max (tpar1, tpar2, ..., tpark).

Vsa vozlišča iz slike 2, ki so označena z B so sestavljena iz različnih konstruktov in omogočajo konstrukcijo časovne enačbe programa z vrha navzdol. Vozlišča označena s P pa označujejo predikate. Vse operacije v vozliščih B, ki sledijo vozliščem P čakajo na ovrednotenje predikata. Tak potek izvajanja je v skladu z osnovno predpostavko, da uporabljamo procesno enoto s povratno zanko v podatkovnem toku, tako kot je ponazorjeno na slikah 2c, 2d, 2e in 2f. Zakasnitve vseh operacij, ki so odvisne od parametrov, katere posreduje "while-do" zanko po zaključenem izvajanju so v skladu z načinom, po katerem deluje procesna enota s povratno zanko. Podobne zakasnitve, ki nastajajo v "repeat-until" zankah niso v skladu z delovanjem procesorja kot je bil slučaj s predhodnimi zankami, ker ponekod prihaja do prekrivanja med iteracijami. To je lahko vir resnih napak pri uporabi modela za računanje časovnih enačb programskih grafov. Do manj resnih napak lahko pride pri vrednotenju zakasnitev v operacijah, ki so odvisne od izhodnih vrednosti v pogojnih stavkih. Tak slučaj je ponazorjen v vozlišču OR na slikah 2c in 2d.

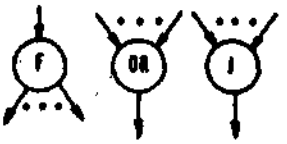
Sliki 2e in 2f ilustrirata tranzitivno predstavitev zank v programu. Časovna enačba za nerekurzivne procedure na sliki 2g vključuje čas za oblikovanje strukture argumenta oziroma izvajanje procedure B in predelavo strukture argumenta. Sliki 2h in 2i kažeta časovne enačbe za branje (ali zapis) elementarnih vrednosti in čas potreben za izvajanje operacij "readedit", "write" in "writedit" ter za ukaze "select".

V predhodnih poglavjih smo s pomočjo časovno ovrednotenega usmerjenega grafa G' = (N', A', T, B) opredelili merilo za oceno vzporednosti v računalniških programih. Merilo vzporednosti nam daje razmerje med časom vzporednega in zaporednega izvajanja algoritma. V ilustracijo metode smo na sliki 3. ponazorili analizo programa napisanega v visokem programskem jeziku. Poleg programa, slika 3, kaže še časovno ovrednoteni usmerjeni graf programa in karakteristične časovne enačbe. Čas izvajanja v posameznih vozliščih je ovrednoten s pomočjo preslikav, ki so ponazorjene na sliki 4. Kako vrednotenje poteka, bomo pokazali z izračunom časa za vozlišče 3. Vozlišče 3 ponazarja stavek v visokem programskem jeziku. Za izvajanje tega stavka sta potrebna dva ukaza "select" (2 časovni enoti), ukaz "multiply" (6 časovnih enot) in ukaz "append" (1 časovna enota), skupno 9 časovnih enot za vsako od N1 iteracij. Dejansko potrebuje vozlišče 3 le 8 časovnih enot, saj se ukaz "select" lahko izvaja v paraleli.

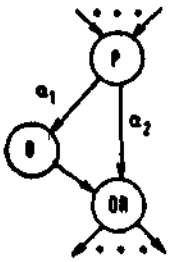
Iz slike 3 je razvidno, da je ocena vzporednosti (Vz) 13/38 dobljena pod predpostavko, da se Ni približuje neskončnosti in da najbolj globoko gnezdena zanka dominiro nad potekom obdelave

Postopek izračunavanja notranje (inherent) vzporednosti v algoritmu lahko avtomatiziramo tako, da izdelamo simulator. Za potrebe simulatorja predstavimo programski graf v obliki, ki ponazarja izvajanje na strojnem nivoju (9). Časovne vrednosti operacij podamo v celoštevilnih spominskih časovnih ciklov. Tako na primer za seštevanje lahko ugotovimo, da zahteva to opera-

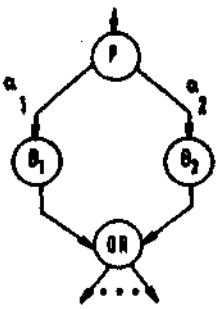
(a) assignment
$\tau_s(AS)$ = sum of machine level operations
$\tau_p(AS)$ = height of parse tree



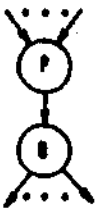
(b) fork, join and OR operators
$\tau(F) = \tau(J) = \tau(OR) = 0$



(c) If P then B
$\tau(COND) = \tau(P) + {}_1\tau(B)$

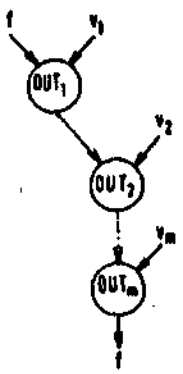


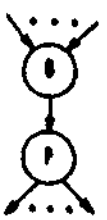
(d) if P then B_1 else B_2
$\tau(COND) = \tau(P) + a_1\tau(B_1) + a_2\tau(B_2)$



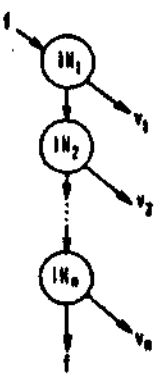
(e) while P do B
$\tau(LOOP) = (N_L + 1)\tau(P) + N_L(B)$

(g) nonrecursive procedure call
A(in $(I_1, \ldots, I_m)$, out $(J_1, \ldots, J_k)$)
$\tau_s(CALL) = m\ t(append) + t(apply) + \tau_s(A) + k\ t(select)$
$\tau_p(CALL) = m\ t(append) + t(apply) + \tau_p(A) + t(select)$





(f) repeat B until P
$\tau(LOOP) = N_L(\tau(B) + \tau(P))$

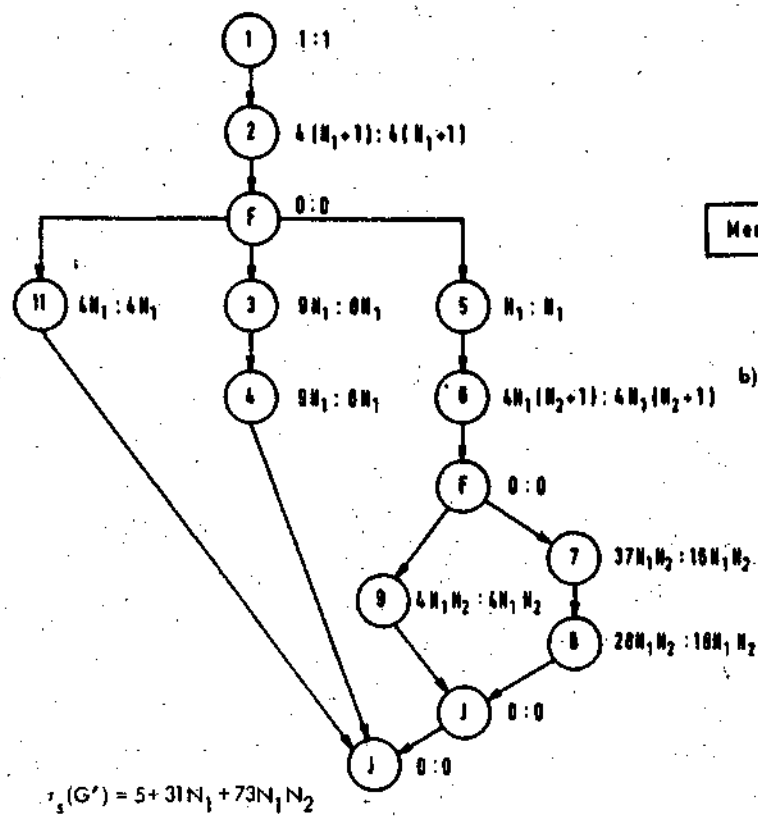


(h) output $v_1, \ldots, v_m$ file = f
format = $(f_1^1, \ldots, f_{k_1}^1, \ldots, f_1^m, \ldots, f_{k_m}^m)$
$\tau(output) = \sum_{i=1}^{m} (OUT_i)$
$\tau(OUT_i) = (k_i - 1)\ t(writeedit) + t(write)$

(i) input $v_1, \ldots, v_m$, file = f
format = $(f_1^1, \ldots, f_{k_1}^1, \ldots, f_1^m, \ldots,$
$\tau(input) = \sum_{i=1}^{n} \tau(IN_i)$
$\tau_s(IN_i) = (k_i - 1)\ t(readedit) + t(read) + 2\ t(select)$
$\tau_p(IN_i) = (k_i - 1)\ t(readedit) + t(read) + t(select)$

Slika 2. Predstavitev časovnega vrednotenja višjih konstruktov

```
1.  I:=1 ;
2.  while I < N do
3.        V(I):=A(I) * B(I) ;
4.        Q(I):=V(I) * C(I) ;
5.        J:=1 ;
6.        while J < N do
7.              E(I,J):=F(I,J) + K(I,J) ;
8.              D(I,J):=E(I,J) * 10 ;
9.              J:= J+1 ;
10.       end ;
11.       I:= I + 1 ;
12. end ;
```



$$\tau_s(G') = 5 + 31N_1 + 73N_1N_2$$

$$\tau_p(G') = 5 + 4N_1 + \max\{4N_1 + 16N_1,\ 5N_1 + 4N_1N_2\} + \max\{4N_1N_2,\ 34N_1N_2\} = 5 + 9N_1 + 38N_1N_2$$

$$\lim V_z(G') = 73/38 \text{ as } N_i \to \infty$$

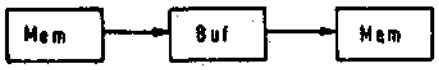
Slika 3. Analiza programa pisanega v visokonivojskem jeziku

Čas izvajanja	Operacije
1	select, append, constant
3	negate, not, readedit
4	+, −, relationals, or, and, writedit
5	abs, arc tan, $\log_e$, sqrt
6	*,/
8	tanh, cosh, sinh, cos, sin, tan
10	read, write
12	arc cos, arc sin, **
14	apply

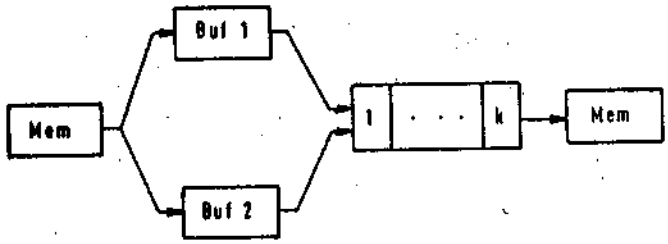
Slika 4. Časi izvajanja za osnovne strojne operacije

cija 4 spominske cikle, enega za izvajanje ukaza "Store", enega za ukaz "Addition" in dva za izvajanje ukazov "Fetch". Za oceno časa kompleksnih operatorjev se lahko usposobijo formule, ki jih je predlagal DeZugish (10).

Rezultati dobljeni z uporabo predlagane metode (12) kažejo na obstoječo vzporednost v programih pisanih v visokem programskem jeziku, in jih kot take lahko uporabljamo, kljub temu, da dobljene vrednosti v časovnem razmerju med vzporednim in zaporednim izvajanjem niso absolutne.



a) Prireditev vrednosti vektorja $t_s^{os} = 1$, $t_w^{os} = 2$



b) Operacija z vektorji $t_s^v = 1+1+k+1 = k+3$, $t_w^v = 2$

Slika 5. Model 2

6. UPORABA MODELA 2

Model 2 predstavlja nadgradnjo modela 1. Za razliko od modela 1, omogoča sprejem in obdelavo vektorsko predstavljenih in zapisanih podatkov. Vsaka vektorska operacija zahteva pred izvajanjem hkratni dostop do vseh operandov.

Pri implementaciji koncepta računalnika krmiljenega s pretokom podatkov operacije z vektorji povzročajo določene težave zaradi problemov pri dodeljevanju spomina in zaradi večkratnega kopiranja podatkov (v primerih ko več vozlišč potrebuje isti podatek).

Model 2, za razliko od modela 1, elemente polj pošilja skozi funkcionalno enoto tako kot kaže slika 5.

Ukaz na strojnem nivoju, določa začetno lokacijo vektorskih operandov v spominu s podatkovnimi strukturami, razširitev operandov ter vrednost inkrementa. Kontrola zanke, ki usmerja pretok vrednosti med spominom in med funkcionalnimi enotami je del logike vgrajene v funkcionalni enoti. Pretok vektorskih elementov iz spomina v posamezne funkcionalne enote in nazaj poteka skozi ena vrata v spominu. Pri tem zanemarjamo vse možne konflikte, ki lahko nastanejo pri vzporednem delovanju več vektorskih funkcionalnih enot.

Čas izvajanja vektorskih operacij je izračunal Ramamoorthy (11):

$$t_s^v + (N-1)t_w^v = t \quad ,$$

kjer je t_s^v čas potreben za pripravo enote, N je dolžina vektorja in t_w^v je enako recipročni vrednosti kapacitete cevasto organiziranih enot. Tako je na primer na sliki 5b t enako:

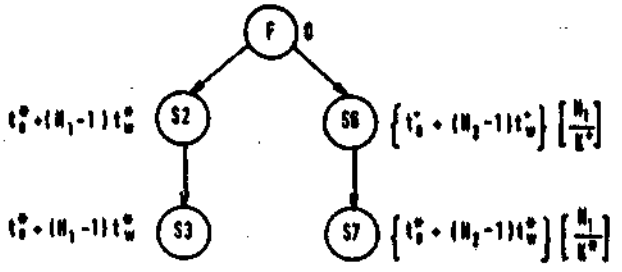
$$t_s^v = 3 + k \quad \text{in} \quad t_w^v = 2 \quad .$$

k označuje številko segmentov v enoti, številka 3 se nanaša na potrebne spominske cikle za dosego operandov in shranjevanje rezultata. Slika 5a kaže poenostavljeno strukturo delovanja in potrebni čas pri prireditvi vrednosti vektorju.

Slika 6 kaže del programa v visokem programskem jeziku (vzporedna inačica programa iz slike 3) in časovno ovrednoteni graf.

```
DO PAR I:=1 TO N1
       V(I):=A(I)*B(I);
       Q(I):=V(I)*C(I);
END PAR;
DO PAR I:=1 TO N1 , J:=1 TO N2
       E(I,J):=F(I,J)+K(I,J);
       D(I,J):=E(I,J)*10;
END PAR;
```

Vzporedna oblika programa iz slike 3



$$\tau_s(G') = 6 + 3lN_1 + 73N_1N_2$$

$$\tau_p(G'') = 2\left\{t_s^* + (N_1-1)t_w^*\right\} + \left[\frac{N_1}{k^+}\right]\left\{t_s^+ + (N_2-1)t_w^+\right\} + \left[\frac{N_1}{k^*}\right]\left\{t_s^* + (N_2-1)t_w^*\right\}$$

b) Programski graf in karakteristične časovne enačbe

Slika 6. Program v visokonivojskem jeziku in njegov programski graf v računalniškem modelu 2

V časovnih enačbah iz slike 6 je upoštevan dodatni dejavnik. Če vozlišče opravi N vektorskih operacij, potem v časovno enačbo za to vozlišče vključimo dejavnik g:

$$g = \left\lceil \frac{N}{K^v} \right\rceil ,$$

kjer K označuje število razpoložljivih oziroma dostopnih funkcionalnih vektorskih enot tipa v. Vrednost g upoštevamo le v primeru uporabe modela 2, oziroma vektorskih procesnih enot. Pri takšni računalniški arhitekturi nastopajo problemi razvrščanja uporabe procesorskih kapacitet s strani neodvisnih podatkovnih poti. Dejavnika g uporabljamo zaradi serijsko zastavljenega načina izvajanja operacij v neodvisnih vozliščih (slika 5b). Če primerjamo skalarni in vektorski model, lahko ugotovimo, da pri vektorskem modelu pridobimo na vzporednosti. Na sliki 6 je ocena te vzporednosti podana z enačbo:

$$V_z(G') = t_{sec}(G')/t_{par}(G'') = 14 \quad \text{za } N1 = N2 = 10$$

in

$$k^+ = k^* = 1 \qquad t_s^+ = t_s^* = 7 \qquad t_w^+ = t_w^* = 2 .$$

Poizkusi, da prevajalnik obdela program, tako kot, da bo obdelava potekala na vektorskem modelu so ostali omejeni na dekompozicijo zank (12).
Vsako zanko, ki vključuje računanje s poljem prevajalnik analizira posebej, in ugotovi ali obstaja neodvisnost v iteracijah. Če taka neodvisnost obstaja, potem se ta transformira v paralelni konstrukt. Če neodvisnosti ni, potem se uporabljajo dodatne tehnike, kot je substitucija, uvajanje začasnih imen polj in konverzija skalarjev v vektorsko obliko in podobno, s ciljem zagotoviti podatkovno neodvisnost. V kolikor poizkus za generiranje podatkovne neodvisnosti uspe, se zanka razdeli na neodvisne dele in vsak del se posebej analizira.

7. ZAKLJUČEK

Programske grafe in karakteristične časovne enačbe smo uporabili za oceno zaporednega in vzporednega časa izvajanja računalniških algoritmov. Pri tem so bila uporabljena dva modela računalniške arhitekture. Prvi model dovoljuje le uporabo skalarnih veličin. Pri oceni zaporednega časa izvajanja je bila upoštevana predpostavka, da je ta čas izvajanja sorazmeren z dolžino najdaljše poti v grafu in z največjo globino gnezdenja zank na tej poti.

8. LITERATURA

1. S. Ribarič, "Računari upravljani tokom podataka", Informatica, 6, No 4, 3 (1982).
2. T. Agervala, M. Arvind, Data flow systems computer, 15, 2, 1982, 15.
3. A. Davis, R. Keller, Data flow program graphs, computer, 15, 2, 1982, 26.
4. D. Martin, G. Estrin, Models of computational systems, IEEE Trans. on Computers, 16, 1, 1967, 70.
5. D. Martin, G. Estrin, Path length computations on graph models of computation, IEEE Trans. on Comp. 18, 6, 1969, 530.
6. R. Russel, The Cray-1 Computer system, CACM, 21, 1, 1978, 63.
7. V. Aho, J. Hopcroft, D. Ullman, The design and analysis of computer algorithms, Add. Wesley, P.C. 1975.
8. J.B. Denis, First version of data-flow procedure language, MAC TM 61, May 1975.
9. A.E. Oldehoeft, Translation of high level programs to data flow and their simulated execution on a feedback interpreter, Comp. Sci., TR, 78-2, Iowa State University, 1978.
10. B. De Lugish, A class of algorithms for automatic evaluation of certain elementary functions in a binary computer, Tec. R. 399, University of Illinois, 1970.
11. C. Ramamoorthy, H. Li, Pipelined architecture, ACM computing surveys, 9, 1, 1977, 61.
12. A. Oldehoeft, R. Zingg, C. Retnadhas, Measurement of parallelism in computer programs through analysis of program graphs, T.R. 78, Iowa State University, 1978.