PREPOJOVACIA SIEŤ V ARCHITEKTÚRE DATA FLOW

(INTERCONNECTION NETWARE IN DATA FLOW ARCHITECTURE)

Liberios VOKOROKOS
Katedra počítačov a informatiky, Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach,
Letná 9, 042 00 Košice, E-mail: Liberios.Vokorokos@tuke.sk

SUMMARY

Dataflow (DF) computer architectures are based on the DF computing model where program instructions are executed when corresponding operands (data) are enabled. From several DF computing models dynamic models belong to the most significant ones. The dynamic DF models enable parallel execution of some operators in dependence on the number of operands, which are available for the execution. The nodes of the corresponding DF graph, by means of which the computing process is represented, can be created dynamically during the execution.

Structural organisation of the DF computer architecture model consists of the five base components like there are Coordinating Processors, Data Queue Unit, Instruction Store, Frame Store and Interconnection Network. The paper deals with the Interconnection Network witch is designed for switching of individual Coordinating Processors together for their data transmission. This research is supported by VEGA grand project No. 1/9027/2002.

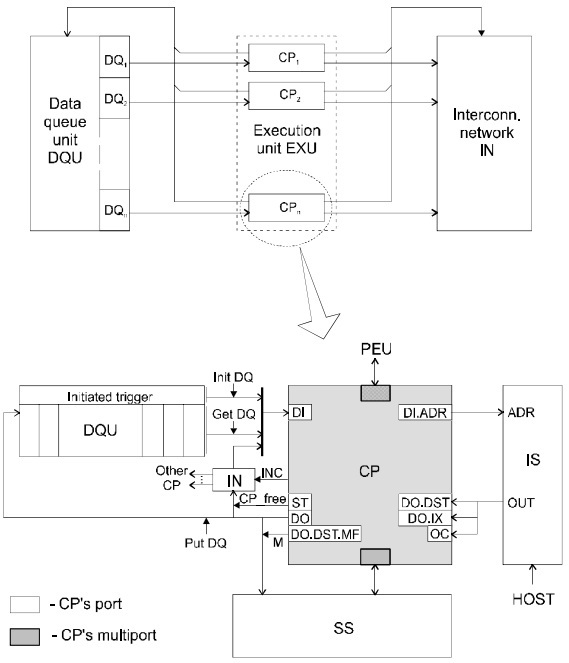
Keywords: *data flow, interconnection netware, simulation, coordinating processor, hypercube*

1. ÚVOD

V rámci riešenia grantového projektu: No: 1/9027/2002 „Výskum paralelných architektúr špecializovaných vysoko výkonných počítačových systémov: architektonické riešenie, metódy hodnotenia, simulácie a aplikácie“, je vyvinutý model data flow (DF) architektúry vychádzajúci z reprezentácie data flow programu pomocou funkcionálneho jazyka. Systém DF KPI [8,14] vyvinutý na Katedre Počítačov a Informatiky Fakulty elektrotechniky a informatiky Technickej univerzity v Košiciach je navrhovaný ako dynamický systém s priamym spájaním operandov. Kombinácia lokálneho control flow modelu s globálnym data flow modelom umožňuje efektívne organizovať paralelnú implementáciu funkcionálneho programu. Predpokladá sa, že implementácia tejto architektúry sa môže použiť ako špecializovaný akcelerátor v problémovo orientovaných počítačových systémoch s vysokými požiadavkami na operačnú rýchlosť.

Hlavné komponenty dynamickej architektúry DF KPI, zobrazené na obrázku 1 sú:

- vykonávacia jednotka (EXU), ktorá obsahuje skupinu koordinačných procesorov (CP) a jednotku procesorových elementov,
- jednotka dátového frontu (DQU) navrhnutá na uloženie operandov produkovaných v priebehu výpočtu,
- prepojovacia sieť (IN), ktorá umožňuje prenos údajov medzi procesormi,
- jednotka IS (instruction store) – v nej sú umiestnené inštrukcie data flow programu,
- jednotka SS (structure store) – pamäť určená na uloženie dátových štruktúr a spájacích vektorov programu.



Obr. 1 Komponenty architektúry DF KPI
Fig. 1 Components of DF KPI architecture

Pri spracovaní operandov a vykonávaní inštrukcií koordinačný procesor sa môže nachádzať v jednom z nasledujúcich stavov: *LOAD, FETCH, OPERATE, MATCHING a PUTTING*. Týmto stavom zodpovedá jeho architektúra, pozostávajúca zo štruktúry zreťazených segmentov [7,14], ktoré umožňujú realizovať výpočet operandov systémom data flow. Základom architektúry CP je prúdová jednotka stavov, ktorá pozostáva zo segmentov, ktoré reprezentujú jednotlivé stavy [7,8].

2. PREPOJOVACIA SIEŤ

Prepojovacia sieť je jednou z piatich hlavných komponentov navrhovanej architektúry DF KPI (obr. 1). Táto sieť prepája navzájom koordinačné procesory, čím sa urýchli vykonanie programu zapísaného vo forme DF grafu [6,9]. Z potreby komunikovať v multiprocesorových systémoch a paralelne distribuovaných pamätiach vznikla požiadavka na vývoj komunikačných prepojovacích sietí. Pri tomto vývoji sa vyčlenil konflikt medzi požiadavkami na spoľahlivé rýchle prepojenie a nízkou cenou. Na základe navrhovanej architektúry sa článok zaoberá prepojovacou sieťou typu hyperkocka.

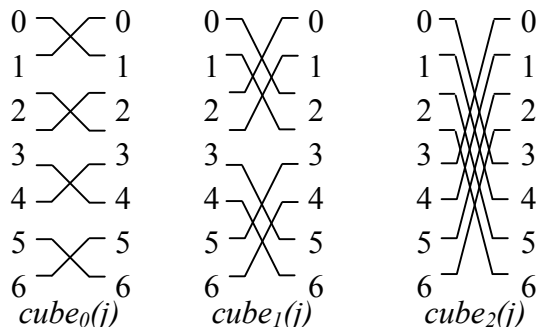
Vlastnosti hyperkocky. N dimenzionálna hyperkocka Q_n ponúka množstvo výhod [19]:
- má 2^n uzlov (procesorov),
- relatívne malá priemerná vzdialenosť uzlov,
- prepojenie medzi uzlami je zabezpečené $n2^{n-1}$ hranami (linkami).

Každý vrchol n-dimenzionálnej hyperkocky má svoju jedinečnú adresu. Prepojovacia funkcia n-dimenzionálnej hyperkocky je definovaná v tvare:

$$cube_i(p_{n-1}\dots p_{i+1}p_ip_{i-1}\dots p_0) = p_{n-1}\dots p_{i+1}p_ip_{i-1}\dots p_0,$$

kde $j = p_{n-1}\dots p_1p_0$ je binárna reprezentácia adresy uzla, $p_i \in \{0,1\}$, $i = 0,1,\dots,n-1$.

Takto definovaná prepojovacia sieť predpokladá vytvoriť medzi procesormi usporiadanými v štruktúre n-rozmernej kocky vzájomné medziprocesorové komunikácie vždy iba medzi dvoma susednými vrcholmi kocky [12]. Teda dva susedné uzly, ktoré sú prepojené linkou (hranou), majú adresy rozdielne iba v jedinom bite. Počet bitov, v ktorých sa adresy dvoch uzlov j_r a j_s líšia, je označený ako $d(p_r, p_s)$. Tento rozdiel adries sa nazýva Hemmingova vzdialenosť medzi uzlami. Linka spájajúca dva susedné uzly má index i, ak sú adresy týchto dvoch uzlov rozdielne práve v i-tom bite adresy.



Obr. 2 Realizácia prepojovacích funkcií $cube_i(j)$
Fig. 2 Realization of interconnection functions

Možné prepojenia v prepojovacej sieti typu hyperkocka sú inicializované prepojovacími funkciami $cube_0(j)$, $cube_1(j)$ a $cube_2(j)$ (obr. 2), kde j je adresa vstupu prepojovacej siete $(j=p_2,p_1,p_0)$. Vo všeobecnosti sa adresami j indexujú procesory P_j, ktoré sú prostredníctvom prepojovacej siete prepojené.

Teda uzly hyperkocky reprezentujú procesory, ktoré komunikujú medzi sebou prostredníctvom liniek. Predpokladá sa, že správy sú prenášané metódou "store and forward", t.j. "ulož a pošli ďalej". Každá linka medzi dvoma susednými uzlami predstavuje obojsmerný komunikačný kanál alebo dva jednosmerné kanály pre prenos [1,2].

Potreba prepojovacej siete v architektúre DF KPI. V takto navrhnutom systéme, v prípade, že v stavoch OPERATE alebo MATCHING bude operand DT zapísaný do vstupného portu stavu FETCH a ten je obsadený (uvažuje sa o prúdovom spracovaní), tento operand DT sa zapíše do frontu DQU príkazom *Put DQ*. Aby sa operand DT mohol opäť spracovať v koordinačnom procesore v stave OPERATE, tak musí prejsť nasledovným procesom:
1. DT sa zapíše do DQU,
2. z DQU sa DT zapíše do vstupného portu stavu LOAD,
3. zo stavu LOAD prejde DT do stavu FETCH,
4. zo stavu FETCH prejde DT do stavu OPERATE.

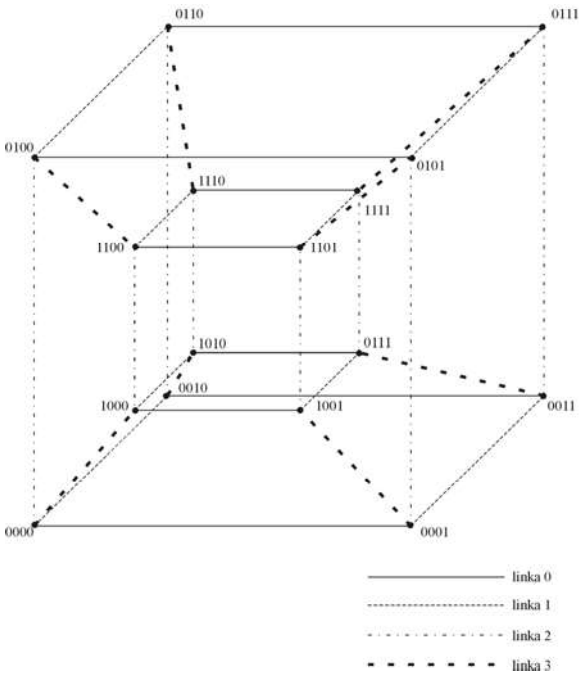
Teda proces bude trvať minimálne 4 kroky. Ak by sme využili prepojovaciu sieť, proces by trval len 2 kroky:
1. DT sa zapíše do vstupného portu stavu FETCH voľného koordinačného procesora,
2. zo stavu FETCH prejde DT do stavu OPERATE.

3. HYPERKOCKA V DF KPI

V navrhnutej architektúre DF KPI sa uvažuje o použití 16 koordinačných procesoroch. Tieto koordinačné procesory sú usporiadané v prepojovacej sieti typu hyperkocka Q_4 (obr. 3) nasledovným spôsobom [13,16]:
- koordinačné procesory sú očíslované CP_0, CP_1, ..., CP_{15},
- každému koordinačnému procesoru je priradená binárna reprezentácia jeho adresy j (napr. pre $CP_0\ j = 0000$, $CP_7\ j = 0111$, $CP_{15}\ j = 1111$).

Z navrhnutej prepojovacej siete je zrejmé, že každý koordinačný procesor je prepojený s ďalšími štyrmi koordinačnými precesormi. Adresy dvoch susedných koordinačných procesorov sú rozdielne v jenom bite a sú prepojené linkou, ktorá je označená práve číslom, ktoré odpovedá tomuto bitu.

Ako príklad je uvedení koordinačný procesor CP_0 *(j = 0000)*, ktorý je prepojený s CP_1 *(j = 0001)* cez *linku 0*, s CP_2 *(j = 0010)* cez *linku 1*, s CP_4 *(j = 0100)* cez *linku 2* a s CP_8 *(j = 1000)* cez *linku 3.*



Obr. 3 Prepojovacia sieť Q_4 pre DF KPI
Fig. 3 Interconnection netware Q_4 for DF KPI

Takže prepojovacia *linka i* reprezentuje spojenie koordinačného procesora *CPj* pomocou prepojovacej funkcie $cube_i(j)$ (obr. 4). V takto navrhnutej prepojovacej sieti môžu medzi sebou komunikovať vždy susediace koordinačné procesory, ktoré sú prepojené pomocou jednej prepojovacej linky.

4. NÁVRH KOMUNIKÁCIE V HYPERKOCKE Q_4

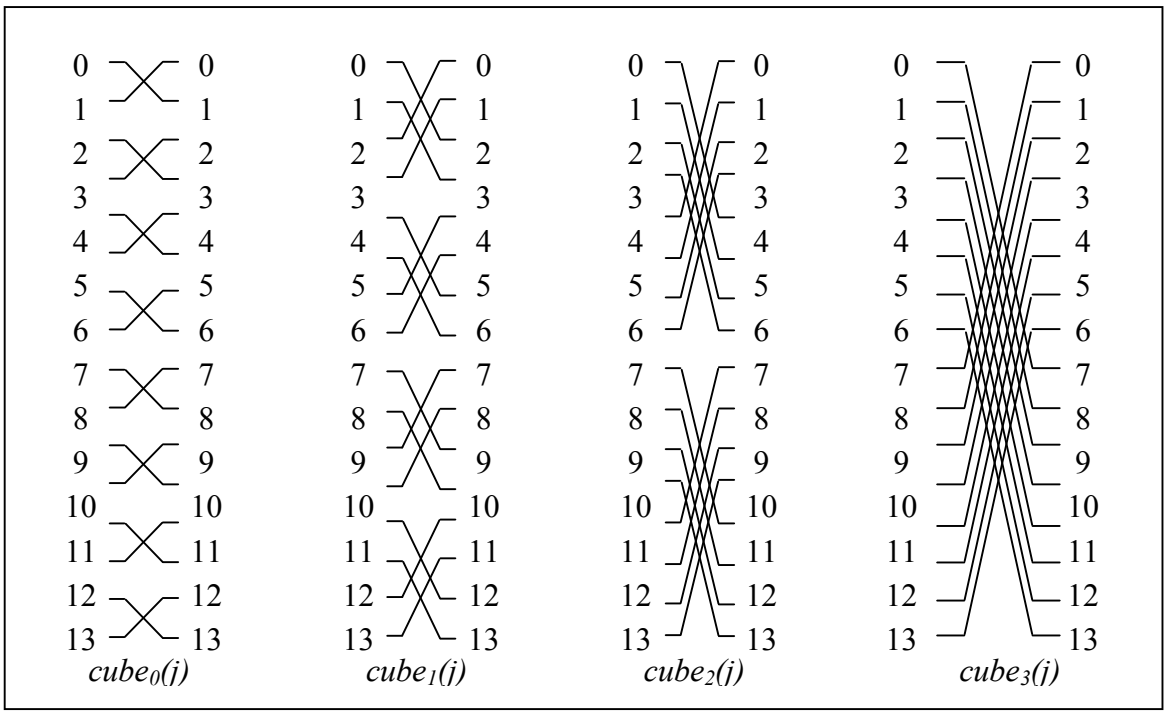
Pri návrhu komunikácie sú na výber dva spôsoby riadenia: centralizovaný a distribuovaný.

Centralizovaný systém je charakteristický jednou centrálnou riadiacou jednotkou, ktorá by ovládala tok údajov medzi jednotlivými koordinačnými procesormi [4,5]. Sledovala by obsadenosť vstupných portov jednotlivých koordinačných procesorov a určovala by ciele pre operandy DT vstupujúce do prepojovacej siete. Nevýhodou tohto systému je veľká zložitosť rastúca s počtom koordinačných procesorov.

Distribuovaný systém je charakteristický vlastnosťou, že každý koordinačný procesor určuje tok údajov medzi koordinačnými procesormi [3,18]. Teda, každý koordinačný procesor si určí, do ktorého koordinačného procesora pošle operand DT. Výhodou je jednoduchosť riešenia a možnosť paralelizmu z pohľadu celej hyperkocky. Činnosť je založená na riešení konfliktov v prípade viacerých voľných koordinačných procesorov alebo v prípade viacerých žiadostí o zápis do koordinačného procesora.

Správnym predpokladom návrhu je paralelný výpočet, preto je potrebné zamerať sa práve na distribuovaný systém.

Celý presun vyprodukovaného operanda DT v koordinačnom procese CP_i (ak je jeho vstupný port v stave FETCH obsadený) do susedného koordinačného procesora CP_j sa udeje v troch etapách:



Obr. 4 Prepojovacie funkcie hyperkocky Q_4
Fig. 4 Interconnection functions of hypercube Q_4

1. výber cieľa presunu operanda DT a vyslanie žiadosti o zápis,
2. odpoveď prijímacieho CP_j na žiadosť vysielacieho CP_i,
3. reakcia na odpoveď a výber ďalšej cesty operanda DT.

Ak nastane situácia, že operand DT ďalej putuje v komunikačnej sieti a všetky susediace koordinačné procesory sú obsadené, bude pre každý koordinačný procesor CP_{zdroj} pevne určený jeden koordinačný procesor $CP_{cieľ}$, do ktorého sa tento operand DT presunie. Každý koordinačný procesor bude mať pomocný register, ktorý bude slúžiť na dočasné uschovanie tohoto operanda DT. V ďalšom kroku by sa algoritmom, ktorý je tu navrhnutý, hľadal voľný sused, t.j. koordinačný procesor, ktorý by mal vstupný port v stave FETCH.

Mohol by nastať prípad, kedy by koordinačný procesor mal vlastný operand DT, ktorý by chcel poslať do siete a ešte aj operand DT v pomocnom registri, ktorý má tiež ísť do siete. Nastal by tu konflikt, ktorý by sa obslúžil. Opäť by sa jeden z nich nedostal ďalej. V prípade, že by väčšina koordinačných procesorov bola plne obsadená, začala by sa sieť preplňovať a operandy DT by "zbytočne" putovali len do pomocných registrov. Kvôli tomuto problému sa táto možnosť vylučuje.

Výber cieľa presunu. Na výber koordinačného procesora, do ktorého má byť poslaný operand DT, je potrebné, aby boli splnené tieto podmienky[15]:

1. každý koordinačný procesor vysiela do svojich susedov (koordinačných procesorov, s ktorými je prepojený cez *linku 0, linku 1, linku 2* a *linku 3*) signál o stave vstupného portu (t.j., či je voľný alebo nie v nasledujúcom takte) v segmente FETCH,
2. každý koordinačný procesor prijíma signály od svojich susedov o stave vstupných portov a je schopný vyhodnotiť aktuálnu situáciu.

Pri takto definovaných podmienkach môžu nastať tri základné situácie:

1. Všetky susediace koordinačné procesory sú obsadené. Vtedy operand DT bude poslaný do frontu DQU.
2. Voľný je práve jeden susediaci koordinačný procesor. Vyšle sa žiadosť o zápis operanda DT do tohto voľného koordinačného procesora.
3. Všetky susediace koordinačné procesory sú obsadené. Vyberie sa podľa priority jeden koordinačný procesor, do ktorého sa vyšle žiadosť o zápis opranda DT. Priorita susedov je daná poradím liniek, cez ktoré je vysielací koordinačný procesor CP_i prepojený s koordinačným procesorom CP_j.

Priorita liniek je vyznačená v tabuľke 1, pričom najvyššiu prioritu má linka 0 a najnižšiu linka 3.

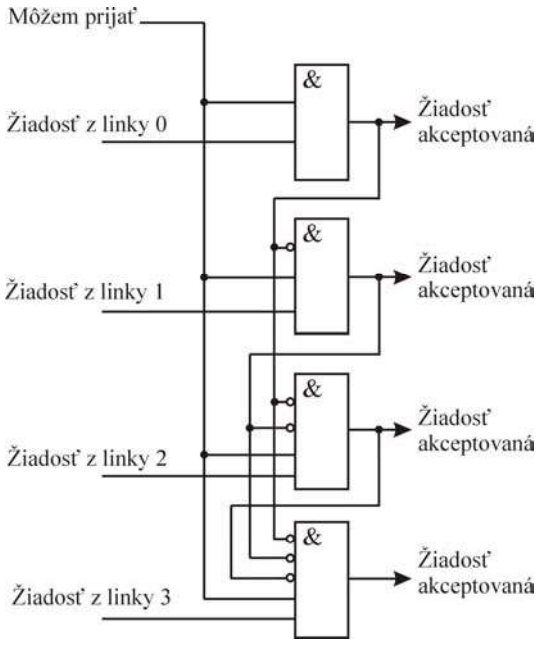
Priorita linky	linka
1.	*linka 0*
2.	*linka 1*
3.	*linka 2*
4.	*linka 3*

Tab. 1 Priorita liniek
Tab. 1 Priority of lines

Odpoveď na žiadosť o zápis. Koordinačný procesor, ktorý má voľný vstupný port v stave FETCH, je schopný prijať operand DT od suseda, takže vysiela signál o tom, že je "voľný". Ako je zrejmé z predchádzajúcich častí, niektoré koordinačné procesory si ho môžu vybrať ako cieľ a pošlú mu signál žiadosti o zápis. Takže tento koordinačný procesor musí mať schopnosť vybrať zo žiadostí o zápis jednu, na ktorú odpovie kladne a ostatné zamietne.

A vtedy môžu nastať tieto dva prípady:

1. O zápis bude žiadať práve jeden koordinačný procesor a koordinačný procesor jednoducho odpovie, že akceptuje žiadosť a je pripravený prijať operand DT.
2. O zápis bude žiadať viac koordinačných procesorov. Koordinačný procesor vyberie z viacerých žiadostí jednu, ktorá má najväčšiu prioritu a akceptuje túto žiadosť. Ostatné žiadosti zamietne.



Obr. 5 Obvod pre paralelné rozhodovanie
Fig. 5 Circuit for parallel decision

Tento výber je možné realizovať cez jednoduchý logický obvod pre paralelné rozhodovanie [17] pomocou pevne stanovených priorít (obr. 5). V systéme DF je použité riešenie, ktoré je bežné pri

riešení obvodov, známych z navrhovania prerušovacích systémov.

Reakcia na odpoveď. Vysielací koordinačný procesor po vysielaní žiadosti do cieľa presunu DT čaká na odpoveď. Môžu nastať dva prípady:
1. Koordinačný procesor dostane kladnú odpoveď na svoju žiadosť a začne presun operanda DT do cieľa.
2. Koordinačný procesor dostane zápornú odpoveď na svoju žiadosť. Oprand DT sa pošle do frontu DQU.

5. SIMULÁCIA PREPOJOVACEJ SIETE

Model zachytáva medziregistrové prenosy v jednotlivých segmentoch. Nezohľadňuje časové hľadisko pri komunikácii koordinačných procesorov s ďalšími komponentmi systému. Činnosť celého modelu je opísaná stavmi vstupných a výstupných portov v jednotlivých segmentoch koordinačných procesorov stavovými signálmi.

Podľa [10,12] má výpočet DF programu na základe DF grafu tvar "lievika", t.j. na začiatku výpočtu vo fáze programu LOADING sa deje veľké rozmnožovanie operandov DT. Postupne sa znižuje počet operandov DT, tak ako sa jednotlivé operandy "konzumujú" rôznymi funkciami, až kým sa nedospeje ku konečnému výsledku. Tejto schéme výpočtu je prispôsobený i navrhovaný model.

Na začiatku je pripustené veľké rozmnožovanie operandov a neskôr sa postupne zvyšuje "konzumácia" operandov. Pod "konzumáciou" sa rozumie spájanie operandov, ak sú vstupom viacoperandovej funkcie, alebo ak operand prislúcha k operátorom KIL alebo GATE [8,11].

Ďalej táto simulácia nezohľadňuje všetky druhy operátorov, koncentruje sa na tri druhy:
1. výstupom je viacero operandov DT (rozmnožovací operátor),
2. výstupom je jeden operand DT,
3. výstupom nie je žiaden operand DT.

Druh operátora, prislúchajúci vstupujúcemu operandu DT sa určuje generovaním náhodného čísla z intervalu <0,8> v stave OPERATE. Podľa jeho hodnoty sa určí jeho druh. Ak patrí intervalu <0,5>, výsledok nie je určený na rozmnožovanie. Ak je z intervalu <5,7>, je určený na rozmnožovanie. Počet rozmnožovaní je určený generovaním náhodného čísla z intervalu <2,5>. Generovaním náhodného čísla sa tiež určuje, či je tento operátor určený na spájanie alebo nie. Dĺžka výpočtu, t.j. počet taktov, ako dlho bude trvať výpočet v stave OPERATE sa určuje generovaním náhodného čísla z intervalu <1,3>.

V stave MATCHING sa tiež náhodne určí, či už daný operand DT má svojho partnera v pamäti štruktúr SS.

Vytvorený program simuluje činnosť koordinačných procesorov prepojovacej siete medzi nimi a frontom údajov DQU. Keďže model paralelnej činnosti je vytvorený na sekvenčnom počítači, je nutné upraviť prácu jednotlivých segmentov tak, aby sa vykonávali paralelne.

5.1. Priebeh simulácie

Na účel simulácie boli vytvorené dva programy, ktoré simulujú činnosť navrhnutého modelu DF KPI. Tieto programy sa líšia typom výstupov.

Po spustení prvého z nich (správny zápis je program 1.exe), dáva na výstup stavy všetkých koordinačných procesorov po ukončení každého taktu. Výstupom je číslo taktu a hodnoty nasledujúcich stavových premenných a hodnôt počítadiel: F.DI_free, signály o stave vstupných portov susediacich koordinačných počítačov, počet operandov DT prijatých od segmentu OPERATE, MATCHING, od prepojovacej siete, od segmentu LOAD, celkový počet vyprodukovaných operandov DT, počet odoslaných do segmentu MATCHING, do siete, do frontu údajov DQ, počet načítaných z DQ a aktuálny počet operandov DT vo fronte DT. Každý riadok odpovedá jednotlivému koordinačnému procesoru. Koordinačné procesory sú usporiadané v poradí od CP_0 do CP_{15}.

Po spustení druhého programu (správny zápis je program2.exe) dáva na výstup počet operandov, ktoré chcú ísť do prepojovacej siete a počet, ktorý sa dostal do prepojovacej siete po každom takte. Rozdiel medzi týmito číslami je počet operandov DT, ktoré boli zapísané do DQU, lebo neboli akceptované ich žiadosti pre nižšiu prioritu alebo vstupné porty F.DI susediacich koordinačných portov boli tiež obsadené. Tento program podáva informácie o zaťaženosti koordinačných procesorov a prepojovacej siete. Podľa počtu žiadostí operandov o vstup do prepojovacej siete a počtu operandov vstupujúcich do siete je možné zistiť, v ktorej fáze výpočtu sa nachádza celý model.

	DT	CP_NET	IN	DQ	%
1.	9897	6241	2626	3615	26,53
2.	10390	6546	2811	3735	27,05
3.	10232	6407	2679	3728	26,18
4.	9560	6013	2576	3437	26,94
5.	10169	6394	2670	3724	26,25
6.	10050	6328	2591	3737	25,78
7.	10224	6378	2612	3766	25,54
8.	10040	6218	2628	3590	26,17
9.	9729	6104	2546	3558	26,16
10.	9849	6172	2594	3578	26,33
Priemer	10014	6280,1	2633,3	3646,8	26,29

Tab. 2 Výsledky simulácie
Tab. 2 Results of simulation

V závere sa vypíšu celkové súčty vyprodukovaných operandov, žiadostí o sieť, vstupujúcich do siete a počet zapísaných do DQU. Posledným

číslom je percentuálne vyjadrenie počtu zaslaných operandov DT do prepojovacej siete k celkovému počtu vyprodukovaných operandov (tab. 2).

5.2. Výsledky simulácie

Simulovaním tohto modelu sa získal obraz o činnosti koordinačných procesorov, frontu DQU a samotnej prepojovacej siete. Program2.exe bol spustený 10 krát a výsledky sú zhrnuté v tabuľke 2.

Z priemerov výsledkov je zrejmé, že až 26,29 % celkového počtu vyprodukovaných DT prešlo cez sieť a tým sa výrazne znížil čas potrebný na výpočet oproti času, keby nebola použitá prepojovacia sieť.

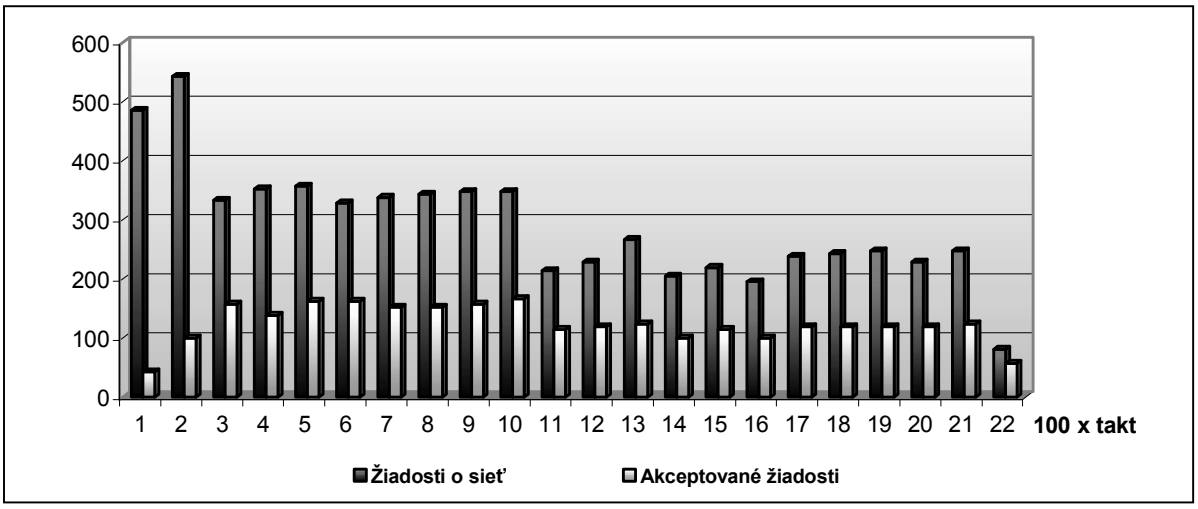
Program2.exe bol použitý na sledovanie zaťaženosti siete v priebehu výpočtu. Rozdelením výsledkov po intervaloch 100 taktov sa dosiahli výsledky, ktoré hovoria o tom, že využitie siete vzrastá s dobou výpočtu (graf 1). Pri veľkom rozmnožovaní operandov na začiatku programu vo fáze LOADING je väčšina koordinačných procesorov obsadená a preto sa nemôže využívať prepojovacia sieť. Väčšina operandov sa zapisuje do frontu DQ.

V grafe 2 je znázornené percentuálne vyjadrenie počtu operandov DT, ktoré sa dostali do siete k počtu operandov DT, ktoré žiadali o sieť.

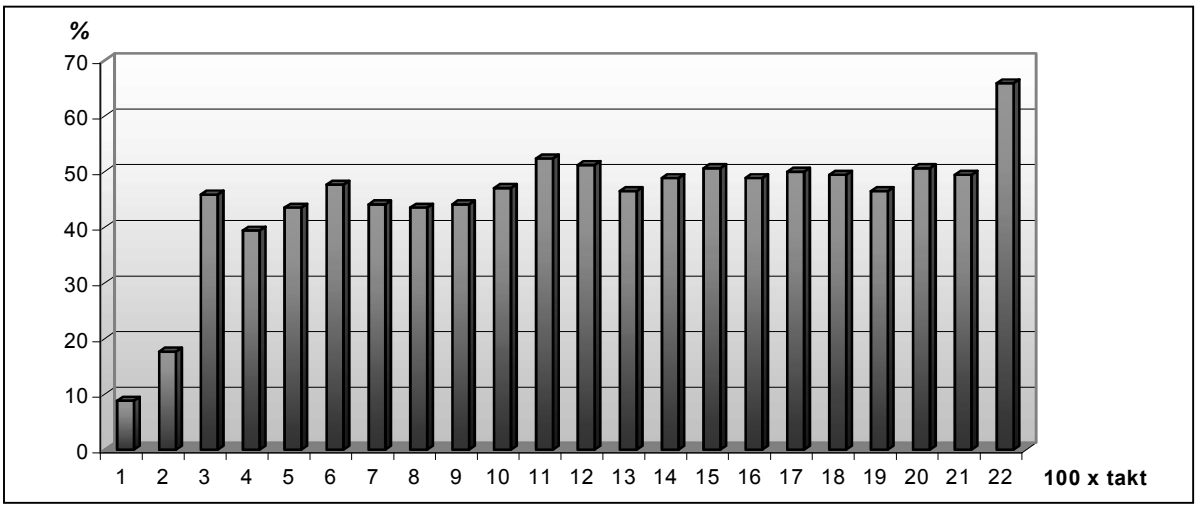
6. ZÁVER

Overenie správnosti návrhu sa dá uskutočniť programovou simuláciou alebo technickou realizáciou a štatistickým vyhodnotením. Simulácia môže byť na úrovni spojovacej pamäti a tabuliek alebo na úrovni celého počítača. Realizácia vyžaduje zrealizovať aspoň jeden DF procesorový element. Základným problémom je, ako overiť simulačný model.

Pre simuláciu na úrovni spojovacej jednotky sú potrebné štatistické údaje z programu o rozložení jednotlivých riadiacich operátorov. Pri simulácii celého počítača je potrebný prekladač z programového jazyka a pomocou skutočných úloh štatisticky overiť navrhnuté predpoklady.



Graf 1 Sledovanie zaťaženosti siete
Graf 1 Monitoring of netware load



Graf 2 Percentuálne vyjadrenie počtu operandov DT
Graf 2 Per cently expression of operands counts DT

LITERATÚRA

[1] Abram, G., Treinish, L.: An Extended Data-flow Architecture for data Analysis and Visualisation. Proc. of Conf. on Visualisation ´95 (Cat. No. 95CB35835), Atlanta, Ga, USA 1995, 263 – 270.

[2] Duncan, R.: A Survey of Parallel Computer Architectures. Computer, 1990, 2 pp. 5 - 16.

[3] Hudec, L., Lesko, J.: Parallel Computing Recovery by Rollback Point Insertion. In: Proc. Of Scientific Conference with Intern. Participation. Electronic Computers and Informatics, Košice-Herlany 26-27.9.96, pp. 2-12 (in Slovak).

[4] Hwang, K.: Advanced Computer architecture. Mc-Graw Hill, Inc., New York 1993, 770 p.

[5] Jamil, T., Deshmukh, R. G.: Design of a Tokenless Architecture for Parallel Computations Using Associative Dataflow Processor, Proc. Of Conf. On IEEE SOUTHEASTCON ´96, Briging Together Education, Science and Technology (Cat. No. 96CH35880), Tampa, FL, USA 1996, 649 - 656.

[6] Jelšina, M.: Computer System Architecture. ELFA s.r.o., Košice 2002, 467.

[7] Jelšina, M., Krahulík, P., Legnavský, M.: Data Flow Architecture of the Functional Language, Proc. of International Conf. on Information, Communications and Signal Processing, IEEE Singapore Section, Singapore 1997, Vol. 3 of 3, 1452-1456.

[8] Jelšina, M., Legnavský, M.: Dynamic Pipeline Architecture of Data Flow System. Journal of Electrical Enginerinrg, Vol. 50, No. 7-8, 1999, pp. 206-210.

[9] Jelšina, M., Vokorokos, L., Sobota, B.: Parallel computer architecture of the MIMD paradigm. III. Interná vedecká konferencia, FEI TU v Košiciach, 28. máj 2003, pp. 35-36. ISBN 80-89066-65-8.

[10] Jelšina, M.: Parallel Processing Control at the Dataflow Computer System. In: Zborník konferencie Electronic Computers and Informatics, FEI TU Košice-Herľany, 1996, s. 120-128.

[11] Kollár, J.: Functional languages for Problem Solving at the Parallel Environment of High Performance Computer Systems, Proc. of Int. Scientific Conference MICROCAD – System´ 93, Technical University Košice, November 1993, 92 - 96.

[12] Plander, I.: Mapping Strategies for Reconfigurable Massively Parallel Computers. Third Workshop on Compilers for Parallel Computers, Vienna, Austria, July 6-9, 1992, pp. 359-375.

[13] Plander, I.: Paralelné architekúry počítačov. Im: Zborník seminára: Paralelné systémy. ÚTK SAV - DT ČSVTS, Tatranská Lomnica 1990.

[14] Vokorokos, L.: Princípy architektúr počítačov riadených tokom údajov. Copycenter, spol. s r.o., Košice, 2002, p. 147. ISBN 80-7099-824-5.

[15] Vokorokos, L.: Diagnostika mechanických zariadení pomocou paralelného počítačového systému. Monografia. Východoslovenské tlačiarne, a.s. 2000. p. 152. ISBN 80-7099-619-6.

[16] Vokorokos, L.: Data Flow Computing Model: Application for Parallel Computer Systems Diagnosis. Computing and informatics, formerly: Computers and artificial inteligence, Vol. 20, No. 4/2001, SAP, pp. 411-428. ISSN 1335-9150.

[17] Vokorokos, L.: Data flow computing model for fault tolerant systems. Fascicola Matematica - Informatica, Buletinul Stiintific al Universitatii din Baia Mare, Seria B, vol. XVI (2000), Nr. 2, 2000, pp. 319-326.

[18] Wadler, P., Miller, Q.: An Introduction to Orwell 5.00. Programming Research Group, University of Oxford, 1990.

[19] Wirth, N.: MODULA – 2. Institute für Informatik, ETH, Zürich, Mar. 1980.

BIOGRAPHY

Liberios Vokorokos (doc., Ing., PhD.) was born on 17.11.1966 in Greece. In 1991 he graduated (MSc.) with honours at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He defended his PhD. in the field of programming device and systems in 2000; his thesis title was "Diagnosis of compound systems using the Data Flow applications". He was appointed Associative Professor and extraordinary professor for Computers Science and Informatics in 2001 and 2003, respectively. Since 1995 he is working as an educationist at the Department of Computers and Informatics. His scientific research is focusing on parallel computers of the Data Flow type. In addition to this, he also investigates the questions related to the diagnostics of complex systems. Currently he is a member of the State Examination Committee in the field Computing engineering and Informatics. His other professional interests include the membership on the Advisory Committee for Informatization at the faculty and Advisory Board for the Development and Informatization at Technical University of Košice.