Acta Informatica Pragensia
2(2), 2013, 77–96, DOI: 10.18267/j.aip.26
Online: aip.vse.cz

Sekce / Section:
Recenzované stati / Peer-reviewed papers

Mikroprogram pre riadenie procesu spájania operandov v architektúre DF KPI

Norbert Ádám[1]

[1] Katedra počítačov a informatiky, Fakulta elektrotechniky a informatiky,
Technická univerzita v Košiciach, Letná 9, 040 01 Košice, Slovenská republika

norbert.adam@tuke.sk

Abstrakt: V článku uvedená architektúra je založená na výpočtovom modeli „data flow“, v ktorom riadenie výpočtu je založené na spracovaní toku dát. Základným prvkom DF architektúry je koordinačný procesor (CP), ktorý je zodpovedný za riadenie a organizáciu vykonávania inštrukcií. Štruktúrna organizácia CP je navrhnutá ako dynamický multifunkčný systém. Pri vykonávaní operácií môže CP prechádzať rôznymi stavmi, v dôsledku čoho táto jednotka predstavuje dynamický prúdový systém. Prechod a poradie prechodu medzi jednotlivými stavmi CP je podmienený typom spracovaného operátora pri interpretovaní prúdu operandov. Z možných typov operátorov, článok uvádza mikroprogramové riadenie pre jednovstupové operátory.

Klíčová slova: data flow model, mikroprogram, spájanie operandov, spracovanie, zreťazenie

Title: Microprogram for operand matching in DF KPI architecture

Abstract: The architecture described in this article is based on the data-flow computation model, in which computation is driven by processing a flow of data. The basic element of the data-flow architecture is the coordinating processor (CP), responsible for controlling and organizing the execution of instructions. As to its structure, the CP is designed as a dynamic, multi-functional system. When executing the operations, the CP may acquire various states, therefore it is a dynamic pipelining system. The transitions and the order of the states between two transitions depend on the type of the operator processed at the time of interpreting the flow of operands. This article details the microprogram control of single-input operators.

Keywords: Data flow model, Microprogramm, Pipeline, Processing, Operand matching, Processing unit

1 ÚVOD

Prudko sa zvyšujúce nároky moderných aplikácií na technické možnosti počítačových systémov, prenikanie výpočtovej techniky do nových vedných disciplín smerujú výskum a vývoj počítačových systémov do sfér, kde si už nevystačíme s klasickými informačnými technológiami typu von Neumann. Vzniká potreba vývoja nových architektúr, na ktoré sa kladú požiadavky s dôrazom na výkonnosť, spoľahlivosť, efektívnosť a cenovú reláciu. Dosiahnutie týchto cieľov je podmienené uplatnením nových fyzikálnych princípov (napr. princípov optoelektroniky), použitím modernej technológie výroby prvkov [4], [6], [7] (vysoká koncentrovanosť prvkov na čipe), nasadenie nových architektonických riešení [9], [14], [17] a vylepšením riadenia týchto systémov [10], [11]. V rámci rôznych smerov vývoja počítačov novej generácie s extrémne vysokou výkonnosťou osobitnú triedu paralelných počítačov tvoria architektúry založené na výpočtovom modeli „data flow“. Výpočtový model „data flow“ (DF) patrí medzi modely, ktoré reprezentujú paralelnú organizáciu výpočtu typu „data-driven“. Jeho charakteristickou vlastnosťou je to, že inštrukcie programu DF pasívne čakajú na príchod určitej kombinácie svojich argumentov, sprístupňovanie ktorých sa organizuje ako údajový prúd riadenia v zmysle definície „data-driven“. Interval čakania inštrukcie na príchod operandov reprezentuje jej výberovú fázu, v priebehu ktorej dochádza k alokácii výpočtových prvkov, tzv. procesných elementov.

Program, pri ktorom sa využíva výpočtový model DF, sa nazýva program DF. Jeho strojovou reprezentáciou je graf toku dát (Data Flow Graph; DFG). Významnou vlastnosťou programu DF, resp. výpočtového modelu DF je to, že umožňuje aplikovať princípy paralelného spracovania inštrukcií na všetkých troch úrovniach (jemnozrnný, strednozrnný, hrubozrnný) paralelizmu. Realizáciu paralelizmu na zvolenej úrovni umožňujú dva charakteristické princípy výpočtového modelu DF:

- Asynchrónnosť – všetky operácie sa vykonávajú vtedy, keď sú dostupné vyžadované operandy.
- Funkcionalita – všetky operácie sú funkcie, tzn., že nezávisia od operandov iných funkcií (neexistujú vedľajšie účinky), v dôsledku čoho sa môžu vykonávať paralelne.

Ďalšou významnou vlastnosťou modelu DF a jeho realizácie je to, že sa programu DF prispôsobuje štruktúra technických výpočtových prostriedkov. Výpočtový model DF sa symbolicky zobrazuje vo forme DFG, v ktorom uzly predstavujú operácie a hrany predstavujú údajové závislosti medzi operáciami. Kým na abstraktnej úrovni sa výpočtový model interpretuje putovaním dátových tokenov (Data Token, DT) v grafe, ako výsledku konzumovania (odpálenia) a generovania operandov uzlami grafu, na realizačnej úrovni sa výpočtový model DF realizuje prenosom a spracovaním údajov vo funkčných jednotkách príslušnej počítačovej architektúry. Boli vypracované rôzne štúdie zamerané na „data flow“ výpočtový model [8], [9], [16] a bolo navrhnutých niekoľko počítačových architektúr riadených tokom dát [1], [2], [3], [5], [9], [14], [15]. Tieto architektúry sa podľa spôsobu spracovania toku dát delia na statické a dynamické architektúry.

2 VÝPOČTOVÉ MODELY ARCHITEKTONICKÝCH KONCEPCIÍ PARALELNÝCH POČÍTAČOVÝCH SYSTÉMOV

Architektonické koncepcie počítačov a princíp ich činnosti sa môže odvíjať od výpočtového modelu, ktorým je definovaný spôsob riadenia procesu spracovania informácií. Existuje rad výpočtových modelov, spomedzi ktorých najvýznamnejšie sú:

Výpočtový model „control flow" (VM CF), pri ktorom sa riadenie výpočtového procesu uskutočňuje prostredníctvom interpretácie sériového prúdu inštrukcií programu.

Výpočtový model „data flow" (VM DF), pri ktorom sa riadenie výpočtového procesu uskutočňuje prostredníctvom prúdu operandov (údajov), definujúcich pripravenosť inštrukcií na vykonanie.

Výpočtový model „demand driven" (VM DD), pri ktorom sa riadenie výpočtového procesu uskutočňuje na základe požiadaviek inštrukcií programu na vyslanie operandov. Výpočtové modely založené na báze organizácie výpočtu typu DD sa nazývajú redukčné modely.

Na základe implementácie týchto výpočtových modelov (VM) sa rozlišujú:

- počítače riadené prúdom inštrukcií – založené na VM CF,
- počítače riadené prúdom údajov – založené na VM DF,
- počítače riadené požiadavkami (redukčné počítače) – založené na VM DD.

Vo všeobecnosti výpočtový model reprezentuje opis realizácie programu. V jednotlivých modeloch CF, DF a DD sa organizácia výpočtového procesu uskutočňuje:

1. *na báze príkazov.* Program sa vykoná prostredníctvom adresovateľných inštrukcií sprístupňovaných nastavením programového počítadla (PC), z ktorých každá, v zmysle von Neumannovho princípu programového riadenia, špecifikuje (control token):
   - operáciu nad operandmi alokovanými s inštrukciami v spoločnej pamäti,
   - podmienené alebo nepodmienené riadenie prechodu na vykonanie nasledujúcej inštrukcie.

   Vykonanie inštrukcií môže prebiehať sekvenčne, prúdovo alebo paralelne.

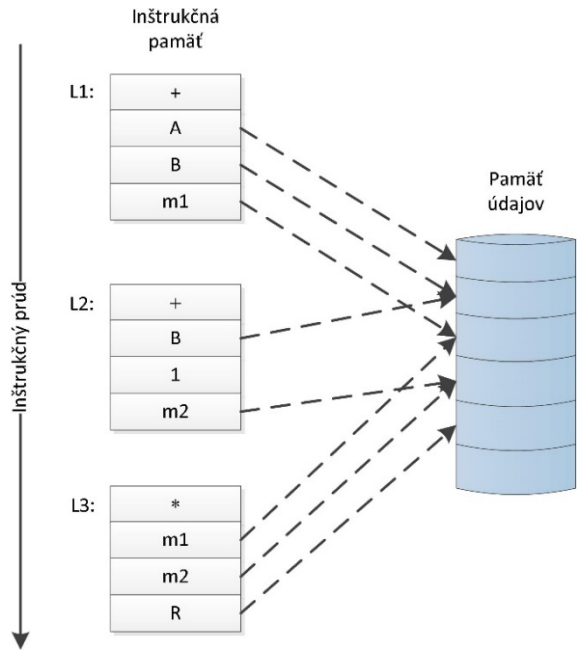
2. *na báze údajov.* Inštrukcie programu sa vykonajú v okamihu, keď majú sprístupnené operandy (data token), čo vytvára optimálne podmienky na jeho paralelnú implementáciu. Vykonanie inštrukcií sa organizuje na rôznej úrovni paralelizmu prúdovo a/alebo paralelne.

3. *na báze požiadaviek*. Inštrukcie programu sa vykonajú v okamihu, ak výsledok príslušnej operácie je požadovaný inou inštrukciou programu prostredníctvom osobitnej požiadavky (demand token). Vykonanie inštrukcií sa organizuje na rôznej úrovni inštrukčného paralelizmu prúdovo alebo paralelne. Požiadavky VM DD kladú pri návrhu počítačových architektúr príliš vysoké nároky na hardvér. Efektívne riešenie mechanizmu šírenia požiadaviek rapídne zvyšuje cenu hardvéru a preto od vývoja redukčných počítačov opustilo. Princípy VM DD sa uplatňujú v dnešných prekladačoch na softvérovej úrovni pri preklade programov do strojového jazyka.

Uvedený článok sa zaoberá opisom počítačovej architektúry s VM DF. Charakteristika ako aj hlavné prednosti a nedostatky VM DF v porovnaní s VM CF sú opísané v nasledujúcej podkapitole.

2.1 PREDNOSTI A NEDOSTATKY DF ARCHITEKTÚR

Konvenčné von Neumannove počítače, sú založené na princípe VM CF, pri ktorom riadenie výpočtového procesu sa uskutočňuje prostredníctvom interpretácie sériového prúdu inštrukcií programu. Sériové vykonávanie inštrukcií na úrovni hardvéru je podporované programovým počítadlom. Úlohou programového počítadla je určiť nasledujúcu inštrukciu v prúde inštrukcií. Obsah programového počítadla po vykonaní jednej inštrukcie sa inkrementuje alebo v prípade riadiacich inštrukcií (GOTO, JUMP a CALL) sa nastaví explicitne. Údaje sa nachádzajú buď v pamäti alebo v registroch počítača. Dátový prúd je určený odvolávaním sa na tieto pamäťové bunky, čo nemá vplyv na riadenie vykonávania inštrukcií, t.j. poradie vykonávania inštrukcií je jednoznačne určené prúdom inštrukcií. Interpretácia výpočtu jednoduchého matematického výrazu v tomto modeli a spôsob reprezentácie dát je znázornený na Obr. 1.

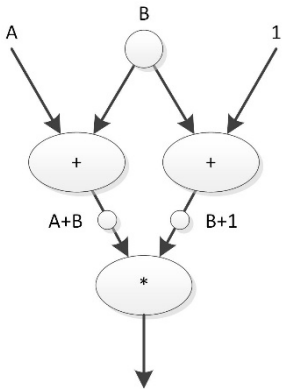


Obr. 1. Výpočet výrazu R = (A + B) x (B + 1) podľa princípu programového riadenia

Inštrukcie sa nachádzajú v pamäti inštrukcií v takom poradí v akom poradí budú vykonávané. Operandy (okrem konštánt) sú reprezentované smerníkmi, ktoré ukazujú na pamäťové miesto kde je operand uložený. Slabá stránka von Neumannovho modelu sa najviac prejavuje pri spracovaní programových cyklov (for, while, ...). Tento model neumožňuje rozvinutie jednotlivých iterácií v priestore. Tým pádom aj v prípadoch, keď telo cyklu tvoria v čase od seba nezávislé inštrukcie, inštrukcie sa vykonávajú sekvenčne za sebou. Výhoda von Neumannovho modelu spočíva v tom, že ak operandy uložíme do registrov, tak je možné vykonať efektívnu optimalizáciu (preusporiadanie) dát.

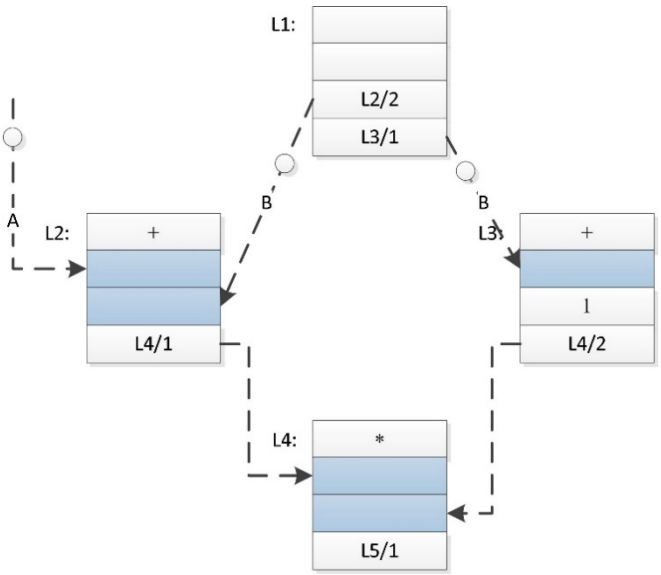
V modeli DF, riadenie výpočtu je jednoznačne určené tokom dát. Poradie inštrukcií v programe nemá vplyv na poradie vykonávania týchto inštrukcií. Inštrukcia sa vykoná v momente keď sú dostupné všetky informácie (všetky operandy) na jej vykonanie. Model DF, na rozdiel od VM CF, nedisponuje spoločnými prepisovateľnými pamäťovými elementami. Údaje sa prenášajú nezávisle medzi pamäťovými bunkami, a ich dostupnosť určuje okamih vykonania inštrukcie. Dostupnosť údajov pre

viacero inštrukcií naraz umožňuje paralelné vykonanie týchto inštrukcií. Možný paralelizmus v procese vykonávania inštrukcií je určený pomocou grafu závislosti dát (Data Dependency Graph), resp. grafu toku dát (DFG). Uzly v grafe reprezentujú inštrukcie programu a hrany medzi uzlami určujú vzťahy medzi dátami (Obr. 2).



Obr. 2. Graf toku dát pre výpočet výrazu R = (A + B) x (B + 1)

Údaje na hranách sa pohybujú v tvare dátových tokenov (DT), ktoré obsahujú stavové informácie a číselnú hodnotu údajov. Asynchrónne a paralelné vykonávanie inštrukcií je dané pravidlom odpálenia tokenu: *uzol je pripravený na odpálenie (vykonanie) ak na každom vstupe tohto uzla sa nachádza token*. Počas vykonania uzla, tokeny na vstupe sú konzumované (následne odstránené zo vstupov), vykoná sa inštrukcia priradená k tomuto uzlu a výsledok vykonania danej inštrukcie sa objaví na výstupe uzla vo forme tokenu. Výhoda DF modelu spočíva v tom, že uzly DFG sa vyhodnocujú samostatne na základe pravidla odpálenia tokenov. Na obrázku 3 sa nachádza riešenie problému z obrázka 2 použitím modelu DF. Z obrázka 3 vyplýva že v tomto modeli neexistuje samostatná údajová pamäť. Operandy inštrukcií sú súčasťou inštrukčného formátu. Inštrukcia je pripravená na vykonanie vtedy a len vtedy, keď všetky jej operandy sú dostupné na jej vykonanie. Dostupnosť operandov je indikovaná prítomnosťou operanda v inštrukčnom formáte.



Obr. 3. Výpočtu výrazu R = (A + B) x (B + 1) podľa princípu riadenia tokom dát

Miesto uloženia týchto inštrukcií vo viacprocesorových systémoch nemá vplyv na efektívnosť vykonávania inštrukcií a synchronizáciu inštrukcií je možné vykonať aj bez explicitných synchronizačných techník ako je napr. semafor. Nevýhoda oproti VM CF spočíva vo veľmi zložitom mechanizme spájania operandov a v ťažkopádnosti registrovej optimalizácie.

Hlavným prínosom VM DF oproti VM CF (v kontexte viacvláknových architektúr[1]) je to, že okamih spracovania inštrukcie programu je podmienený len prítomnosťou požadovaných operandov, t.j. umiestnenie inštrukcie v pamäti počítača nemá vplyv na poradie vykonávania inštrukcie. Tento fakt umožňuje zvýšiť koeficient vyťažiteľnosti funkčných jednotiek danej architektúry, čo priaznivo ovplyvňuje priepustnosť a tým redukuje potrebný čas na dokončenie výpočtu. V DF architektúrach sa prejavuje len hazard typu RAW (Read-after-Write). Ďalšie dva typy údajového hazardu, WAR (Write-after-Read) a WAW (Write-after-Write), sú eliminované počas prekladu výpočtového problému do tvaru DFG. Znalosť informácie o výskyte hazardu RAW, už v čase prekladu do DFG, umožňuje efektívne synchronizovať činnosť komponentov DF architektúry a tým znižovať oneskorenie v dôsledku nedostupnosti operandov pre spracovanie inštrukcií. Zároveň platí, že koeficient vyťažiteľnosti procesného elementu vo viacvláknových architektúrach vyjadrený zápisom:

$$U = \frac{P}{T} = \frac{P}{P + I + S} \qquad \textbf{(1)}$$

kde

- U - koeficient vyťažiteľnosti procesného elementu počítačovej architektúry;
- P - čas potrebný na realizáciu výpočtu;
- T - doba spracovania programu;
- I - čas čakania na vykonanie V/V operácie, čas prístupu do pamäťového podsystému, čas v priebehu ktorého procesný element vykonáva inštrukcie iného programu v rámci multiprogramovania;
- S - čas potrebný na prepínanie medzi kontextmi vlákien.

v prípade DF architektúr udáva väčšie číslo (hodnotu bližšiu k číslu 1) ako pre viacvláknové architektúry riadených tokom príkazov [13].

Ďalšou prednosťou VM DF oproti VM CF je viacrozmernosť, viacnásobné prúdové spracovanie inštrukčného paralelizmu a hardvérový paralelizmus. Základný VM DF vychádza z dvoch východiskových prístupov vytvárania modelov DF, ktoré reprezentujú statické modely a dynamické modely, opis ktorých je predmetom nasledujúcej kapitoly.

3 POČÍTAČE RIADENÉ TOKOM DÁT

Implementácia architektúry počítača DF závisí od spôsobu vykonania inštrukcií programu DF, ktorý prebieha ako proces prijímania, spracovania a vysielania dátových tokenov (Data Token - DT), reprezentujúcich údaje a príznaky na hranách DFG. V závislosti na spôsobe spracovania DT v DFG, resp. v závislosti na rozsahu architektonickej podpory jeho vykonávania, sa rozlišujú nasledujúce typy priamych architektúr DF:

[1] Porovnávajú sa paralelné architektúry.

- Statické modely (Obr. 4a);
- Dynamické modely (Obr. 4b).

Výpočtový model statického DF počítača bol navrhnutý výskumným tímom Dennisa z inštitúcie MIT [3]. Statický model reprezentovaný orientovaným grafom, pozostáva z operátorov, údajových (dátových) a riadiacich hrán a údajových (dátových) a riadiacich DT. Z priestorového hľadiska je prípustné, aby na jednej hrane bolo umiestnených niekoľko DT, v danom časovom okamihu môže byť však prítomný iba jeden. Operátor je aktivovaný (spustený), ak na jeho výstupných hranách sa nenachádza žiadny DT [1], [3].

Statický model (Obr. 4a) pozostáva z nasledujúcich funkčných blokov:

Pamäť aktivačných rámcov (Activity Store) obsahuje aktivačné rámce inštrukcií, reprezentujúce uzly v DFG. Každý aktivačný rámec obsahuje operačný kód, položky pre operandy a cieľovú adresu. Položka operandov obsahuje kontrolný bit pre určenie dostupnosti operanda.

Aktualizačná jednotka (Update Unit) vykonáva aktualizáciu DT a skontroluje či je inštrukcia vykonateľná. Ak podmienka vykonateľnosti je splnená, jednotka pošle inštrukciu cez inštrukčnú FIFO pamäť do výberovej jednotky inštrukcií.

Jednotka výberu inštrukcií (Fetch Unit) sprístupňuje adresy vykonateľných inštrukcií, na základe ktorých, z príslušných aktivačných rámcov uložených v *aktivačnej pamäti*, vyberie operačný kód, príslušné operandy a cieľové adresy výsledku operácie, vytvárajúc tak operačný balík. Tento operačný balík potom pošle do voľnej *operačnej jednotky*.

Operačná jednotka (Operation Unit) zrealizuje výpočet a výsledok vykonanej operácie pošle do *aktualizačnej jednotky*.

Nakoľko statická DF architektúra nie je schopné efektívne spracovať také programové konštrukcie ako sú cykly a rekurzie (v tomto modeli je povolený len jeden token na vstupe operátora) bol vyvinutý dynamický model DF architektúry.

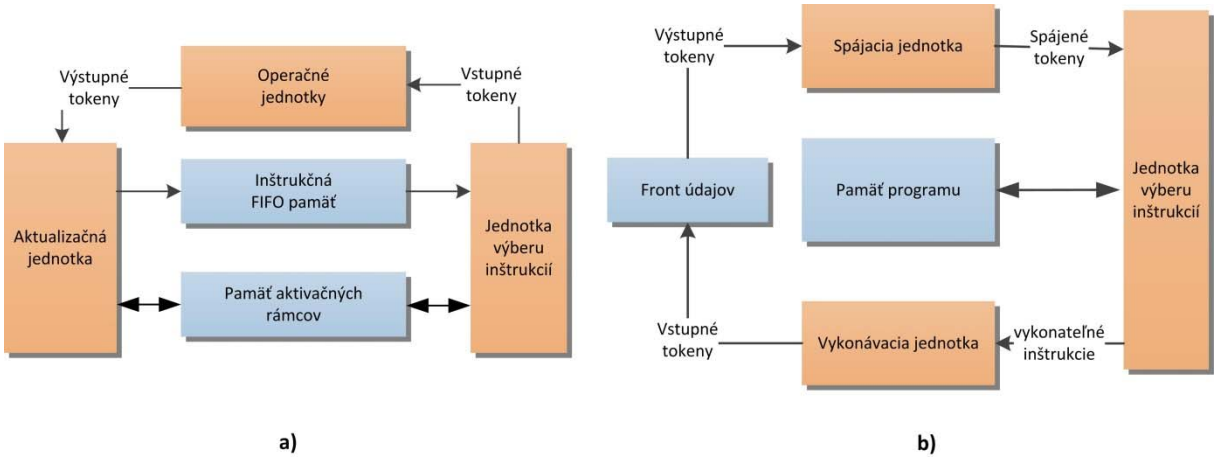
Dynamický DF model bol navrhnutý skupinou s členmi Arvind z MIT [1], Grund a Watson z University of Manchester [5].

V dynamickom DF modeli operátor spojený s uzlom je vykonateľný vtedy, keď všetky vstupné hrany obsahujú tokeny, ktorých značky sú identické [1]. V tomto modeli každá hrana môže obsahovať viac ako jeden značený token. Pri vykonaní uzla k sebe patriace tokeny sú odstránené zo vstupných hrán a na výstupnej je vygenerovaný token s príslušnou značkou. Dynamický DF model využíva tak slučkový paralelizmus ako aj rekurzívny paralelizmus, ktoré sa dynamicky objavujú počas behu programu, preto takáto architektúra musí podporovať proces spájania operandov.

Opis jednotlivých funkčných blokov (Obr. 4b) dynamického modelu počítača riadeného tokom dát je uvedený nižšie.

Spájacia jednotka (Matching Unit) predstavuje pamäť, ktorá vykonáva spájanie operandov na vstupe operátora. K spájaniu operandov dochádza len vtedy, ak DT definujúci daný operand je určený pre

dvojvstupový operátor, na základe čoho sa inicializuje proces spájania operandov. Ak spájanie bolo úspešné (z frontu tokenov bol sprístupnený token s rovnakou značkou uloženou v jednotke) jednotka pošle korešpondujúci DT do jednotky výberu inštrukcií. Ak v jednotke sa nenachádza DT korešpondujúci so vstupujúcim DT, token sa uloží do nej.



Obr. 4. (a) Statická data flow architektúra (b) Dynamická data flow architektúra

Jednotka výberu inštrukcií (Fetch Unit) uskutočňuje výber pripravených inštrukcií z pamäti programu a generuje vykonateľný balík pre *vykonávaciu jednotku*.

Pamäť programov (Instruction Store) plní funkciu pamäti inštrukcií programu DF.

Vykonávacia jednotka (Processing Unit) spracuje operácie definované v DF programe a výsledok postúpi ďalej do *spájacej jednotky* cez *front údajov*.

Front údajov (Token Queue) spĺňa funkciu prepojovacieho média medzi *vykonávacou* a *spájacou jednotkou*.

Súčasný vývoj aplikácií princípov DF pri návrhu architektonického riešenia počítačov DF je zameraný na dynamické DF výpočtové modely. Rad dynamických modelov DF architektúr tvoria:

- architektúry s *označenými aktivačnými značkami* (Tagged Token Model, TT),
- architektúry *s pamäťou explicitných aktivačných značiek* (Explicit Token Store Model),
- architektúry *s kombinovaným (hybridným) CF/DF modelom* (Hybrid Model).

Hlavnou prednosťou TT architektúr pred statickými modelmi je lepšia výkonnosť, pretože pripúšťa viacnásobné DT na hranách uzla, čo umožňuje rozvinutie väčšieho paralelizmu. Problémom modelov TT architektúr je dosiahnutie efektívnej implementácie jednotky spájania rovnako označených DT (spájacej jednotky) na hranách uzlov (operátorov). Pre tento účel bola používaná asociatívna pamäť, ktorá však pre jej vysoké náklady nepredstavovala optimálne riešenie (aplikácia asociatívnej pamäti na spájanie DT v TT architektúrach musí mať značnú kapacitu, v dôsledku čoho jej realizácia je príliš nákladná).

Nevýhodou TT architektúr je nízka účinnosť spájania DT uskutočňovaného pomocou cenovo náročnej asociatívnej pamäti so zložitou organizáciou vyhľadávania veľkého množstva operandov. Asociatívny

princíp spájania je možné eliminovať zavedením pamäti explicitných aktivačných značiek. Princíp spájania pomocou pamäti explicitných aktivačných značiek spočíva v alokovaní aktivačných rámcov v pamäti rámcov (Frame Store, FS) pre každú aktiváciu iterácie cyklu alebo podprogramu pri vykonávaní programu DF.

Pravé architektúry DF, ktoré vychádzajú z koncepcie prítomnosti jediného DT na hranách DFG (statické DF) alebo viacerých, označených DT na hranách DFG (dynamické DF), spracúvajú inštrukčný kód s relatívne nízkou výkonnosťou. Tento nedostatok vyplýva z nasledujúcich skutočností:

- nie sú využité prednosti prúdového spracovania, pretože vykonanie každej inštrukcie v definovanej postupnosti inštrukcií môže byť prúdovo spracované až po ukončení predchádzajúcej inštrukcie. Tzn., že v n-fázovom prúdovom DF systéme je jeho výkonnosť n-násobne znížená;
- réžia spájania DT znižuje efektívnosť prúdového systému DF, pretože v dôsledku aktivačného pravidla dyadická operácia môže byť vykonaná až po príchode druhého (partnerského) DT, čo vytvára podmienky na vznik oneskorení v prúdovom spracovaní reťazca inštrukcií;
- v dôsledku prepínania kontextu po vykonaní každej inštrukcie v definovanom zoskupení inštrukcií jemnozrnného systému DF, nie je možné zvýšiť jeho výkonnosť:
    - použitím registrov na optimalizáciu prístupového času k údajom,
    - vyhýbaním sa oneskorení pri prúdovom spracovaní dyadických inštrukcií,
    - znížením počtu DT pri vykonávaní programu DF.

Uvedené nedostatky je možné eliminovať aplikáciou kombinácie VM CF a VM DF pri návrhu architektúry DF. Z von Neumannovských princípov v architektúre DF sa uplatňujú tie inštrukcie, ktoré redukujú réžiu riadenia systému DF prostredníctvom:

- zvyšovania granularity DFG,
- využitia rôznych mechanizmov riadenia zdrojov známych z von Neumannovských architektúr.

Na základe uplatnenia rôznych techník kombinovania VM CF a VM DF sa rozlišujú nasledujúce modifikácie architektúr DF:

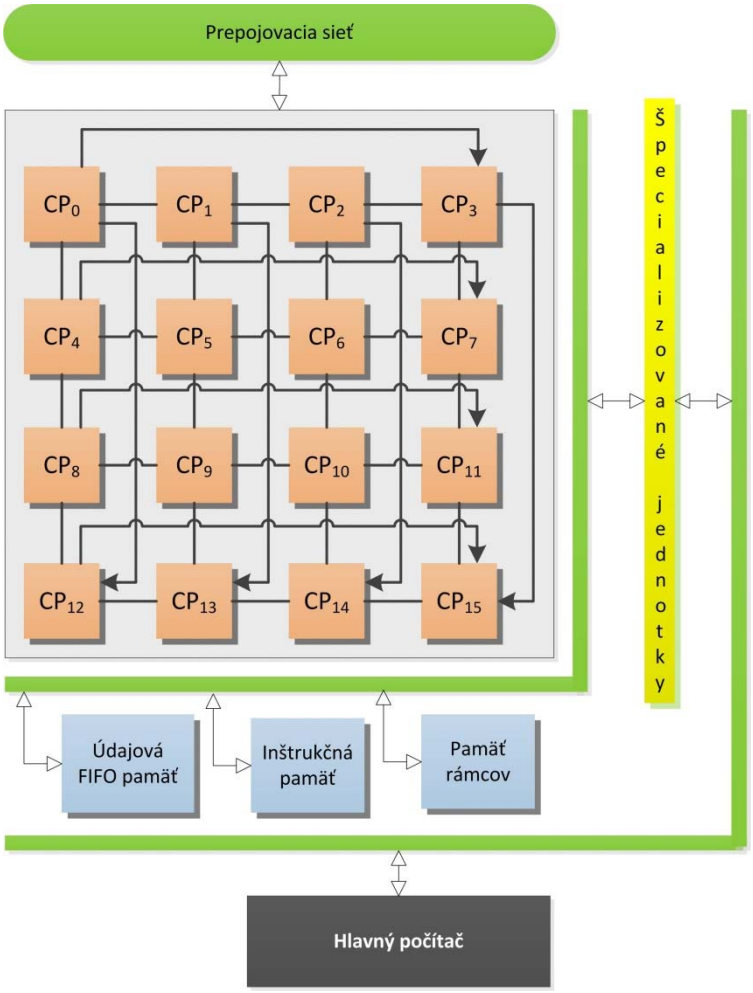
- *zreťazené* DF (Multithreaded Data Flow),
- *hrubozrnné* DF (Coarse-Grained Data Flow, Macro Data Flow),
- *RISC* DF.

Architektúra opísaná v nasledujúcich kapitolách patrí do triedy hybridných CF/DF architektúr. Opisovaná architektúra DF-KPI je jedinečná v kontexte dynamických hybridných CF/DF architektúr nakoľko zavádza N-vstupové ($N \geq 3$) operátory a spája prednosti zreťazených a hrubozrnných hybridných architektúr. Umožňuje vytvárať subgrafy v DF programe pomocou techniky farbenia grafu, čím vzniká možnosť si zvoliť optimálnu granularitu paralelizmu, aplikovať techniku maskovaného multi-mapovania subgrafov do pamäte rámcov [] a zároveň umožňuje uplatniť techniku prúdového spracovania na úrovni subgrafov.

4 SYSTÉM DF-KPI

Systém DF-KPI [9] vyvíjaný na Katedre počítačov a informatiky Fakulty elektrotechniky a informatiky Technickej univerzity v Košiciach je navrhovaný ako dynamický systém s priamym spájaním operandov. Kombinácia lokálneho CF modelu s globálnym DF modelom umožňuje efektívne organizovať paralelnú implementáciu funkcionálneho programu. Model architektúry počítača DF-KPI je súčasťou komplexného systému DF, ktorý zahŕňa aj podporné komponenty výpočtového prostredia DF pre realizáciu definovaného aplikačného určenia.

Štruktúrna organizácia (Obr. 5) modelu architektúry počítača DF-KPI pozostáva z nasledujúcich funkčných jednotiek.



Obr. 5. Systém DF-KPI

Koordinačný procesor (Coordinating Processor; CP) – je určený na riadenie, koordinovanie a spracovanie inštrukcií programu DF, na základe prítomnosti ich operandov, ktoré sa na vstupný port CP.DI koordinačného procesora sprístupňujú buď z jeho výstupného portu CP.DO, resp. z výstupných portov CP.DO iných CP prostredníctvom prepojovacej siete, alebo z *údajového frontu* a z *pamäti rámcov*. Štruktúrna organizácia CP je navrhnutá ako dynamický multifunkčný systém, ktorý pozostáva zo segmentov LOAD, FETCH, OPERATE, MATCHING a COPY.

Údajový front (Data Queue Unit; DQU), je jednotka, určená na ukladanie DT, reprezentujúcich operandy, ktoré čakajú na spájanie v priebehu vykonávania programu.

Inštrukčná pamäť (Instruction Store; IS) je pamäť inštrukcií programu DF v tvare kódu príslušného DFG.

Pamäť rámcov (Frame Store; FS) je pamäť spájacích (párovacích) vektorov, pomocou ktorých CP zisťuje prítomnosť DT na vykonanie operácie definovanej operátorom (uzlom) v DFG. Skrátená definícia formátu položky spájacieho vektora (Matching Vector; MV) v *pamäti rámcov* je <AF><V>, kde AF (Affiliation Flag) je príznak prítomnosti operanda a V (Value) je hodnota daného operanda v DT.

Podporné komponenty systému DF sú potrebné pre vytvorenie reálneho výpočtového prostredia. V danej architektúre ich tvoria:

Hlavný počítač (HOST) pre zabezpečenie štandardných funkcií počítačového systému výpočtového procesu DF.

Špecializované jednotky slúžia na vytvorenie špecializovaného aplikačného prostredia (virtuálna realita, diagnostika, e-learning).

V/V procesory (súčasťou hlavného počítača) pre rýchle priame vstupy/výstupy do modulu DF (štandardné V/V sa realizujú prostredníctvom hlavného počítača).

4.1 INŠTRUKČNÝ FORMÁT

Inštrukcie zdrojového kódu programu sú uložené v inštrukčnej pamäti. Každá inštrukcia reprezentuje operátor v DFG. Operátory DFG sa delia na jednovstupové a na N ($N \geq 2$) vstupové. V prípade jednovstupových operátorov, vstup tvorí len jeden DT, ktorého prítomnosť spĺňa podmienku vykonateľnosti, t.j. nie je potrebné spájanie operandov. Pre N vstupové operátory platí, že operátor, reprezentujúci inštrukciu DF sa vykoná vtedy a len vtedy, ak sú prítomné všetky vstupujúce DT. Tieto DT následne sú spájané v spájacom segmente na základe MV. Súbor inštrukcií architektúry DF-KPI tvoria inštrukcie reprezentované jednovstupovými operátormi (ACCEPT, IF, KILL, OUT, RET, SEL, UN_OP), dvojvstupovými (BIN_OP, CASE, DEF, GATE, LOAD, SEND, TUP) a M ($M \geq 3$) vstupovými (APPLY, CONSTR). Operátor ACCEPT je vstupným operátorom DF programu a podprogramu. Operátor IF reprezentuje programové dvojcestné a operátor CASE viaccestné vetvenie. Na konzumovanie vstupného DT bez odozvy slúži operátor KILL. Jednovstupový operátor OUT je posledným operátorom a RET návratovým operátorom (pod-)programu DF. Operátor SEL vyberá dáta z údajovej štruktúry definovanej operátorom CONSTR. Jednovstupový operátor UN_OP reprezentuje unárnu a dvojvstupový BIN_OP binárnu matematickú operáciu. Operátor DEF slúži na definovanie programovej konštanty. Na vytváranie kópie z údajovej a adresnej časti vstupného DT slúži operátor LOAD, kým operátor TUP vytvára kópie len z údajovej časti DT. Spustenie programu/podprogramu je odštartované operátorom APPLY. Podrobnejší popis týchto operátorov sa nachádza v [9].

Formát inštrukcie (Data Flow Instruction; DFI) zdrojového kódu je nasledovný:

$$\langle DFI \rangle ::= \langle OC \rangle \langle LI \rangle \langle \{DST, [IX]\}^n \rangle$$

kde

- OC - je kód operátora;
- LI - literál (napr. počet kópií výsledku);
- DST - adresa cieľového operátora výsledku operácie;
- IX - index spájania výsledku operácie.

Zápis $\langle \{DST, [IX]\}^n$ definuje možnosť výskytu tejto dvojice 1 až n-krát v DFI, pričom položka [IX] môže ale nemusí (výskyt 0 alebo 1) byť súčasťou DFI. Inštrukcia je identifikovaná a vykonaná na základe adresy obsiahnutej v časti DST dátového tokenu. Formát vstupujúceho DT je definovaný zápisom:

$$\langle DT \rangle ::= \langle P \rangle \langle T, V \rangle \langle MVB \rangle \langle \{DST, [IX]\} \rangle$$

kde

- P - je priorita spracovania DT;
- T - typ hodnoty údaja;
- V - hodnota údaja;
- MVB - bázová adresa spájacieho vektora;
- DST - cieľová adresa vstupujúceho DT.

Formát cieľovej adresy vstupujúceho DT je nasledujúci:

$$\langle DST \rangle ::= \langle MF \rangle \langle IP \rangle \langle ADR \rangle$$

kde

- MF - je spájacia funkcia (Matching Function), s definovaným príznakom z množiny {M, B}, M – spájanie (Matching) dvoch DT, B – prechod bez spájania (Bypass);
- IP - je vstupný port, s definovanou hodnotou z množiny {L(eft), R(ight)};
- ADR - je adresa operátora, ktorý sa aktivuje vstupným DT.

V navrhnutej architektúre sa používa priame spájanie operandov. Je založené na alokovaní spájacieho vektora podľa definície pamäte rámcov a aktivácie bloku kódu (procedúra, volanie funkcie). Aktivovaný spájací vektor je reprezentovaný ako aktivačný záznam v pamäti rámcov. Aktuálna bázová adresa sa označuje ako MVB = B_{ACT}. Formát spájacieho vektora v pamäti rámcov je nasledovný:

$$\langle FS[B_{ACT} + H + IX + 1] \rangle ::= \langle RC, MVS \rangle \langle B_{OLD} \rangle \langle DST_{RET} \rangle \langle D\{[B_{NEW}]\{D\}\} \rangle$$

kde

- B_{ACT} - je ukazovateľ na začiatok aktuálneho záznamu;
- H - počet slov rezervovaných pre hlavičku záznamu;
- MVS - veľkosť spájacieho vektora;
- RC - počítadlo referencií;
- B_{OLD} - smerník na starý aktivačný záznam;

DST_{RET} - cieľ návratu;
B_{NEW} - bázová adresa nového aktivačného záznamu;
D - operand definovaný hodnotou, príznak prítomnosti AF, typ hodnoty T.

K spájaniu DT dochádza pri spracovaní operandov vstupujúcich do N-vstupových operátorov. RC je nastavené podľa veľkosti spájacieho vektora v čase kompilácie. Po realizácii spájania vstupných DT sa RC dekrementuje. Ak RC = 0, tak sa spájací vektor v pamäti rámcov uvoľní.

Proces spájania spočíva v uložení hodnoty operanda v alokovanom priestore pamäti rámcov až do okamihu, kedy sú prítomné všetky vstupné operandy operátora. V takomto prípade sa všetky operandy načítajú a operátor sa vykoná, pretože je splnená podmienka vykonateľnosti.

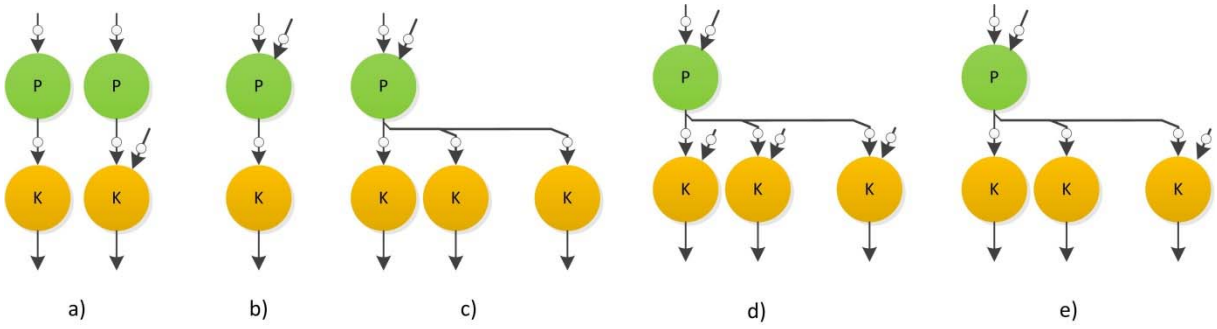
4.2 SPÁJANIE OPERANDOV

Nositeľom informácie o stave výpočtu je DT, zobrazovaný na hranách uzlov DFG v tvare plného krúžku. Jeho umiestnenie na vstupných hranách uzlov (operátorov) vyjadruje prítomnosť operandov a definuje vykonateľnú inštrukciu. Umiestnenie DT na výstupnej hrane uzla vyjadruje prítomnosť výsledku operácie definovanej operátorom vykonanej inštrukcie.

Vykonanie DF grafu sa uskutočňuje na základe pravidiel vykonateľnosti (inštrukcia je vykonateľná, ak všetky jej operandy sú prístupné) a aktivácie (inštrukcia je aktivovaná, keď je vykonateľná a zdroje na jej aktiváciu sú k dispozícii) inštrukcií programu DF.

Jeden z najvýznamnejších krokov, vychádzajúc z dynamického modelu DF, je priame spájanie operandov [9]. Koncepcia priameho spájania operandov spočíva v eliminácii výpočtovo (časovo) náročných procesov spojených s asociatívnym hľadaním operandov. V schéme priameho spájania operandov v navrhovanej architektúre DF-KPI, položky spájacieho vektora v pamäti rámcov sa alokujú dynamicky pre každý jeden token generovaný počas vykonania DFG. Aktuálne umiestnenie spájacieho vektora v pamäti rámcov je určené počas kompilácie programu, kým bázová adresa pamäti rámcov je určená po spustení programu DF. V schéme priameho spájania operandov, sa dá každý výpočet popísať pomocou smerníka na inštrukciu (ADR) a smerníka na spájací vektor v pamäti rámcov (MVB). Dvojica <MVB, ADR> je súčasťou hlavičky DT. Typickou akciou je hľadanie páru DT v pamäti rámcov. Po príchode operanda do koordinačného procesora, sa podľa indexu spájania zistí, či už je v pamäti rámcov prítomný spoluvstupujúci operand operátora. V prípade, že tam ešte nie je, uloží sa operand v pamäti rámcov do spájacieho vektora určeného bázovou adresou operanda do položky danej indexom IX.

Riadenie procesu spájania operandov na vstupe operátora ovplyvňuje proces spracovania a generovania výsledku na jeho výstupe. Použitím prekladača programu do DFG s dopredným prehľadávaním, ktorý umožňuje detegovať a eliminovať redundantné výpočty, zmeniť poradie spracovania tokenov, riadenie je možné definovať ako prechod DT po hranách DFG (Obr. 6) medzi operátorom „produkujúcim“ (P – producent) a operátorom „konzumujúcim“ (K - konzument) DT. Tento proces je definovaný nasledujúcimi kombináciami prechodu DT po hranách medzi operátormi P a K grafu toku dát, ktoré môžu byť jednovstupové, dvojvstupové, resp. riadiace.



Obr. 6. Variácie konfigurácie P-operátorov a K-operátorov v procese spájania operandov

Pri spracovaní programu DF nám vznikajú nasledujúce typy spojení operátorov:

PJ/KJ – P jednovstupový, K jednovstupový (Obr. 6a). Je to konfigurácia ktorá nevyžaduje spájanie operandov. Dátový token produkovaný producentom P je pohltený konzumentom K bez aktivácie procesu spájania.

PJ/KD – P jednovstupový, K dvojvstupový (Obr.6a). Vyžaduje spájanie operandov v operátori K.

PD/KJ – P dvojvstupový, K jednovstupový (Obr.6b). K spájaniu dôjde v operátori P.

PD/uKJ – P dvojvstupový, u×K jednovstupový (Obr.6c). Po vykonaní inštrukcie definovanej dvojvstupovým operátorom P, výsledok je distribuovaný medzi *u jednostopovými* operátormi K. K spájaniu dôjde len v operátori P.

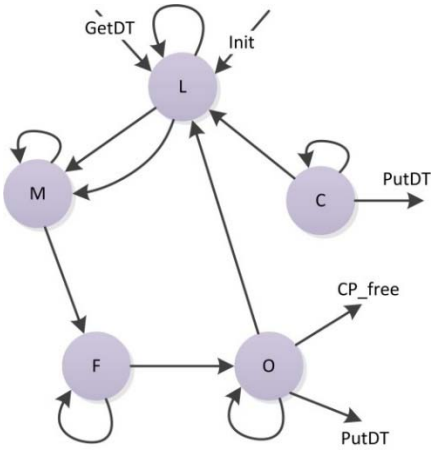
PD/vKD – P dvojvstupový, v× K dvojvstupový (Obr. 6d). Výsledok realizácie dvojstupového operátora P, je rozoslaný na *v dvojvstupový* operátor K. Proces spájania je aktivovaný tak pre operátor P ako aj pre operátor K.

PD/uKJvKD – P dvojvstupový, u×K jednovstupový, v×K dvojvstupový (Obr. 6e). Výsledok realizácie dvojstupového operátora P, je rozoslaný na *u jednovstupový* a na *v dvojvstupový* operátor K. Proces spájania je aktivovaný tak pre operátor P ako aj pre dvojvstupové operátory K.

Z uvedených možných kombinácií P a K operátorov proces riadenia spájania operandov bude opísaný vzhľadom na konfiguráciu z obrázka 6a.

4.3 SPRACOVANIE OPERÁTOROV TYPU PS/CS A PS/CD

Koordinačný procesor (CP) predstavuje dynamický prúdový systém, ktorý umožňuje prechádzať medzi stavmi (stupňami) prúdovej funkčnej jednotky (L – load, M – matching, C – copy F – fetch, O – operate) v rôznom poradí. Prechody medzi stavmi na základe riadiaceho mikroprogramu pri riadení procesu spájania operandov sú znázornené pomocou stavového diagramu (Obr. 7).

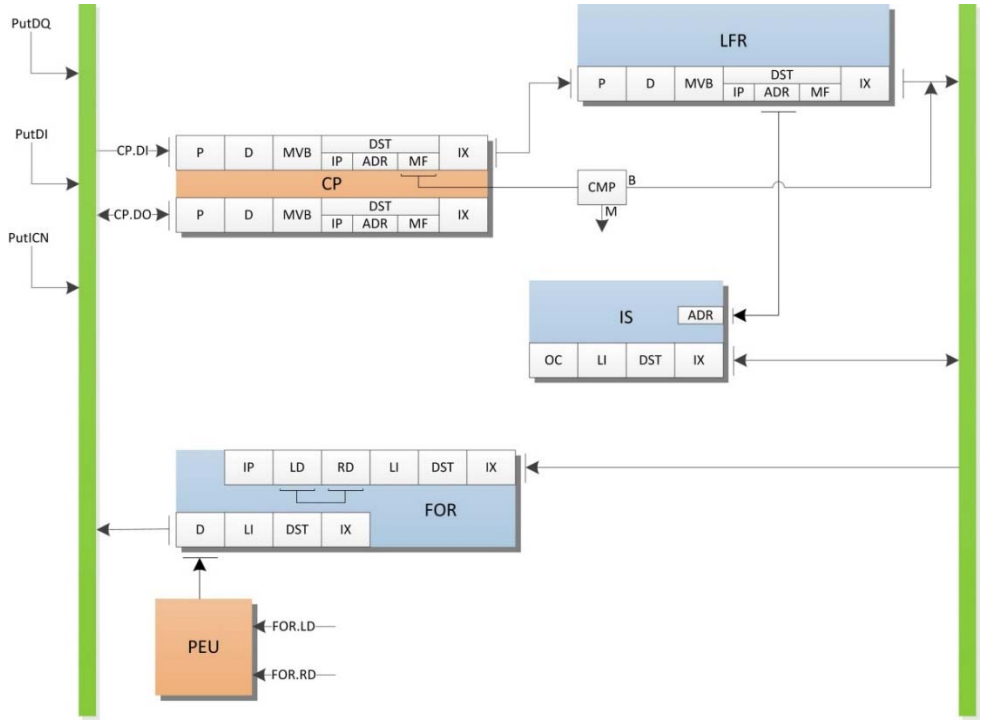


Obr. 7. Stavový diagram spracovania dátových tokenov v architektúre DF-KPI

Na obrázku 7 majú vyznačené riadiace signály nasledujúci význam: *CP_free* – indikuje obsadenosť alebo prístupnosť CP; *GetDT* – čítanie tokenu z DQU; *PutDT* – zápis tokenu do DQU; *Init* – inicializácia prúdových stupňov.

CP je v stave LOAD, ak očakáva alebo sprístupňuje operand z DQU prostredníctvom riadiaceho signálu *GetDT* alebo sprístupňuje operand z výstupného portu CP.DO koordinačného procesora, alebo z výstupných portov CP.DO iných CP, prostredníctvom prepojovacej siete (IN). Signál *Init* slúži ku počiatočnej inicializácií CP. V stave FETCH CP číta kód operácie definovaný príslušnou inštrukciou (operátorom), ktorej adresa je súčasťou vstupujúceho DT. Príslušná inštrukcia je adresovaná v IS. Potom je aktivovaný stupeň OPERATE. V stave OPERATE CP zabezpečí vykonanie inštrukcie podľa VM CF. Výsledok vykonanej inštrukcie predstavuje vstupný operand inej inštrukcie a preto sa tento výsledok posiela do spájacieho segmentu v ktorom sa vykonáva hľadanie partnerského operandu. Ak pre vstupný operand sa nenájde partnerský operand, tak tento stav je zaznamenaný v príslušnom spájacom vektore (MV) na pozícií, ktorá určuje dostupnosť operandu (Affiliation Flag; AF). V opačnom prípade, ak partnerský operand je dostupný (jeho pozícia je zaznamenaná v MV a tá určuje miesto uloženia v pamäti rámcov FS), tak sa načíta a spracujú sa v procesnej elementárnej jednotke (PEU). Výsledok spracovanej operácie (čo je opäť vyjadrený vo forme DT) je preposlaný na vstup ďalšieho operátora (, resp. na vstupy ďalších operátorov, pokiaľ bol označený viac ako jeden cieľový operátor vo formáte vstupujúceho DT), pokiaľ koordinačný procesor CP nie je zaneprázdnený (*CP_free* = 1, čo je ekvivalent zápisu *isFree*(LOAD) = 1) iným výpočtom. Ak je CP zaneprázdnený (*CP_Free* = 0), výsledný DT je prostredníctvom prepojovacej siete odoslaný na spracovanie inému nezaneprázdnenému CP. Ak nastane prípad, že všetky CP sú zaneprázdnené, potom výstupný operand (výsledný DT) sa ukladá do dátovej fronty (DQU), čo sa aktivuje pomocou kontrolného signálu *PutDT*. Ak výsledok párovania je neúspešný, operand je ukladaný do vybranej položky príslušného spájacieho vektora, na základe kontrolného signálu spájacej funkcie MF=M. Stav OPERATE môže byť rozložený na niekoľko dielčich krokov, keď výsledok operácie ukončovanej v stupni OPERATE sa odosiela viac ako jednému nasledujúcemu operátoru. K vykonaniu týchto dielčích krokov sa využíva stav COPY [9].

Mikroarchitektúra prúdovej funkčnej jednotky určenej na riadenie procesu spájania operandov pre typ operátorov PJ/KJ a PJ/KD je znázornená na obrázku 8.

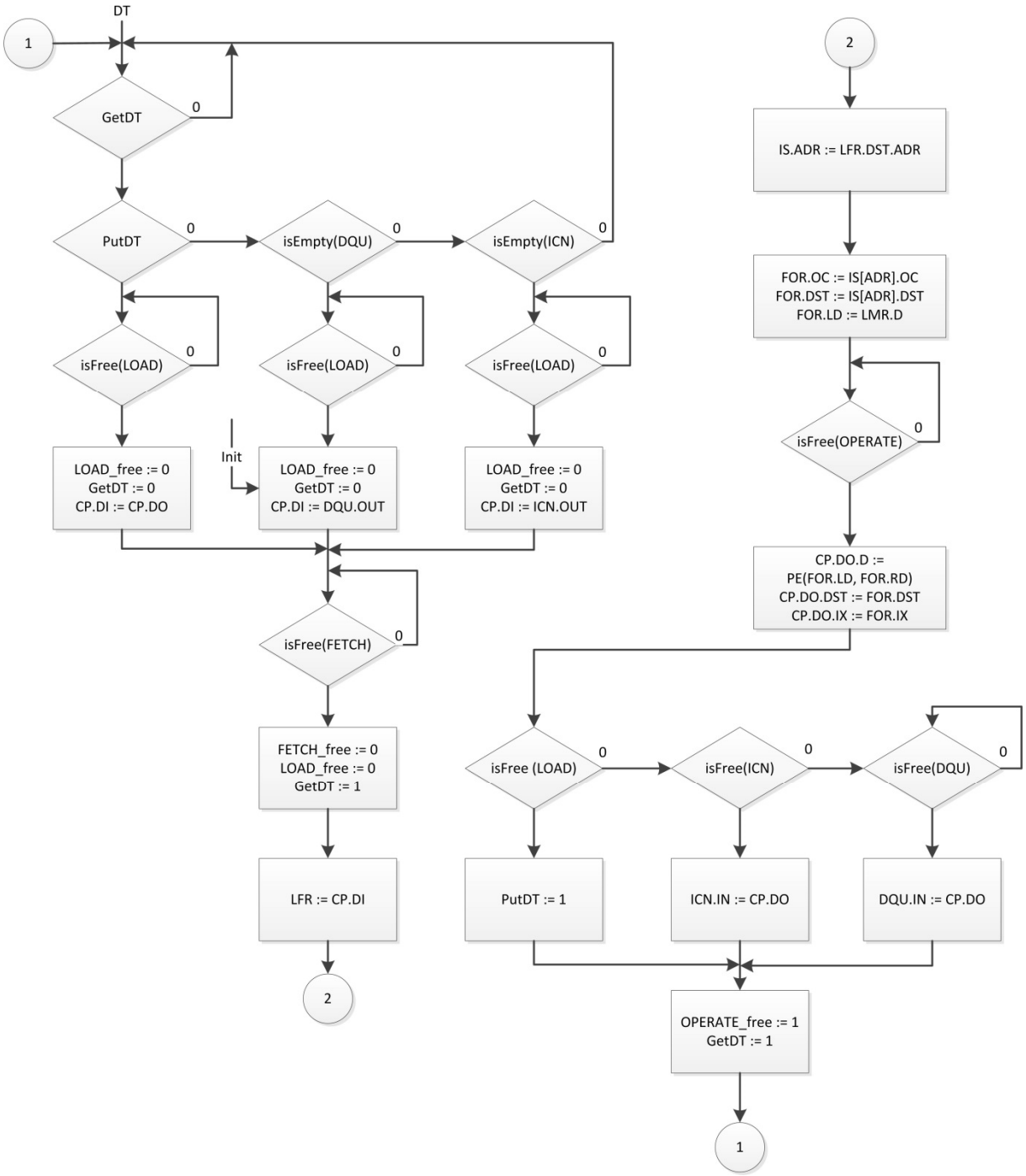


Obr. 8. Prúdová jednotka pre spracovanie operátorov typu PJ/KJ a PJ/KD

Kvôli zvýšeniu priepustnosti a koeficientu vyťažiteľnosti prúdovej funkčnej jednotky medzi jej jednotlivé stupne boli vložené medzistupňové pamäte typu FIFO s nasledujúcou špecifikáciou:

- Medzi stupne L a F → register LFR
- Medzi stupne F a O → register FOR

Mikroprogramové riadenie vykonania operátorov jednovýstupových operátorov DFG sa uskutočňuje prostredníctvom mikrooperácií (Obr. 9) pre jednotlivé stupne multifunkčnej prúdovej jednotky (Obr.8) koordinačného procesora.



Obr. 9. Mikroprogram pre spracovanie operátorov typu PJ/KJ a PJ/KD

Funkcia *isFree*(X), testuje príznak obsadenosti segmentu X. Mikrooperácie, ktoré môžu byť paralelne vykonané, sú uložené do jedného príkazového bloku vývojového diagramu. Inicializácia koordinačného procesora sa uskutočňuje nastavením príznaku *Init* := 1 v štartovacej inštrukcii programu DF. V prípade, že ďalší segment, segment FETCH, je voľný, načíta sa DT do medzipamäte LFR (load/fetch register), uvoľní sa segment LOAD a aktivuje sa príjem ďalších DT do koordinačného procesora. Na základe adresy LFR.DST.ADR, sa určí adresa DF operátora, ktorý sa načíta z inštrukčnej pamäti do medzipamäte FOR (fetch/operate register). V prípade, že segment OPERATE je

neaktívny (isFree(Operate) = 1), začne sa spracovanie operátora načítaného z FOR. Výsledok spracovanej operácie (token) je dostupný pre spracovanie operandu v ďalšom operátore, pokiaľ CP nie je zaneprázdnený iným výpočtom. Ak CP je zaneprázdnený, tak sa operátor pošle pomocou prepojovacej siete na spracovanie inému nezaneprázdnenému CP. Ak všetky CP sú zaneprázdnené, token sa ukladá do DQU.

5 ZÁVER

Snaha skonštruovať vysokovýkonné systémy vedie k neustálemu skúmaniu a využívaniu výhod tej ktorej architektúry alebo k vývoju novej, u ktorej sa dopredu predpokladá zvýšená výkonnosť, aby uspokojovala a dokonca aj prekračovala súčasné nároky na výkon. Jedným z východísk architektúry počítača a princípu jeho činnosti je spôsob organizácie a riadenia výpočtového procesu - spracovania informácií. Výpočtový proces je reprezentovaný postupnosťou stavov, k zmene ktorých dochádza v dôsledku vykonávania inštrukcií programu, na základe ktorého sa výpočtový proces v počítači organizuje. Organizácia výpočtového procesu predstavuje popis uskutočňovania týchto zmien prostredníctvom definovania podmienok pre vykonávanie inštrukcií a ich dôsledkov. Z organizácie výpočtového procesu vyplýva príslušný riadiaci mechanizmus. V článku uvedená architektúra je založená na výpočtovom modeli „data flow“, v ktorom riadenie výpočtu je založené na spracovaní toku dát. Základným prvkom navrhovaného DF-KPI architektúry je koordinačný procesor (CP), ktorý je zodpovedný za riadenie a organizáciu vykonávania inštrukcií. Štruktúrna organizácia CP je navrhnutá ako dynamický multifunkčný systém. Pri vykonávaní operácií môže CP prechádzať rôznymi stavmi, v dôsledku čoho táto jednotka predstavuje dynamický prúdový systém. Opisovaná architektúra DF-KPI je jedinečná v kontexte dynamických hybridných CF/DF architektúr nakoľko spája prednosti zreťazených a hrubozrnných hybridných architektúr. Umožňuje vytvárať subgrafy v DF programe pomocou techniky farbenia grafu, čím vzniká možnosť si zvoliť optimálnu granularitu paralelizmu a zároveň umožňuje uplatniť techniku prúdového spracovania na úrovni subgrafov. Uplatnenie princípov prúdového spracovania skomplikuje úlohu spájania operandov na vstupe operátorov. S cieľom zefektívniť proces spájania operandov sa použila koncepcia priameho spájania operandov. Priame spájanie operandov je založené na použití pamäti explicitných aktivačných značiek (dedikovaná časť pamäte CP zvaná aktivačný rámec). Myšlienka zavedenia tejto pamäti vychádza z poznania, že pri spracovaní subgrafov, ktoré sú súčasťou cyklov, resp. rekurzií, nie je nutné subgrafy označiť za každým novou farbou. Počas prúdového spracovania prechod a poradie prechodu medzi jednotlivými stavmi CP je podmienený typom spracovaného operátora pri interpretovaní prúdu operandov. Z možných typov operátorov, článok uvádza mikroprogramové riadenie pre operátory typu PJ/KJ a PJ/KD. Navrhnutý algoritmus je optimalizovaný na rýchlosť spracovania týchto operátorov. Z algoritmu vyplýva, že jeden „inštrukčný cyklus“ spracovania týchto operátorov vyžaduje 8 strojových cyklov. Počet potrebných cyklov zároveň definuje počet fáz pri prúdovom spracovaní. Z teórie prúdového spracovania vieme, že zrýchlenie spracovania inštrukcií (‚resp. programových entít) je ekvivalentné počtu zreťazených fáz, čo pre architektúru DF-KPI znamená 8-násobné zrýchlenie voči sekvenčnému spracovaniu. Prezentovaný systém DF-KPI je ešte „len“ v štádiu vývoja, preto toto číslo podľa môjho očakávania nie je ešte konečné.

POĎAKOVANIE

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe zmluvy č. APVV-0008-10“.

6 SEZNAM POUŽITÝCH ZDROJŮ

[1] ARVIND – CULLER, D. E. Dataflow Architectures. In *Annual Review of Computer Science*, Vol. 1, CA: Annual Reviews Inc. Palo Alto, 1986. pp. 225-253. ISBN 0-8243-3201-6.

[2] CARLSTRÖM, J. – BODÉN , T. Synchronous Dataflow Architecture for Network Processors. In *Micro IEEE*, Volume. 24, Issue 5, 2004. pp. 10-18. ISSN 0272-1732. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.229.1244&rep=rep1&type=pdf

[3] DENNIS, J. B. Data-Flow Supercomputers. In *Computer* , Volume 13, Issue 11, 1980, pp. 48-56.

[4] FRANKLIN, A. D at al. Sub-10 nm Carbon Nanotube Transistor, In Nano Letters, Volume 12, Issue 2, 2012. pp. 758-762.

[5] GURD, J. R. – KIRKHAM, C. C. – WATSON, I. The Manchester Prototype Data-Flow Computer. In *Communication of the ACM*, Volume 28, Number 1, 1985. pp. 34-52. Dostupné z: http://courses.cs.washington.edu/courses/csep548/05sp/gurd-cacm85-prototype.pdf

[6] GYÖRÖK, GY. – MAKÓ, M. – LAKNER, J. Combinatorics at Electronic Circuit Realization in FPAA. In *Acta Polytechnica Hungarica*, Volume 6, No. 1, 2009. pp. 151-160. ISSN 1785-8860. . Dostupné z: http://www.uni-obuda.hu/journal/Gyorok_Mako_Lakner_17.pdf

[7] GYÖRÖK, GY. The FPAA realization of analog predictive circuit. In *8th IEEE International Symposium on Applied Machine Intelligence and Informatics: SAMI 2010, 2010.* pp. 105-108. ISBN 978-1-4244-6424-1.

[8] JAMIL, T. – DESHMUKH R.G. Design of a Tokenless Architecture for Parallel Computations Using Associative Dataflow Processor. In *Proc. of Conf. on IEEE SOUTHEASTCON ´96*, Briging Together Education, Science and Technology, Tampa, FL, USA 1996. Pp. 649 - 656. ISBN 0-7803-3088-9.

[9] JELŠINA, M a kol. *Architektonické riešenie počítačového systému data flow KPI*. Košice: Elfa s.r.o., 2004. ISBN 80-89066-86-0.

[10] KOPJÁK J. – KOVÁCS, J. Event-driven control program models running on embedded systems. In *6th IEEE International Symposium on Applied Computational Intelligence and Informatics*, Romania, 2011. pp. 323-326. ISBN 978-1-4244-9109-4.

[11] KOPJÁK J. – KOVÁCS, J. Timed cooperative multitask for tiny real-time embedded systems. In *Proc. of IEEE 10th Jubilee International Symposium on Applied Machine Intelligence and Informatics, SAMI 2012*, Slovakia, 2012. pp. 377-382. ISBN 978-145770197-9.

[12] MADOŠ B. – BALÁŽ, Data FLow Graph Mapping Techniques of Computer Architecture with Data Driven Computation Model. In *Proc. of IEEE 9th International Symposium on Applied Machine Intelligence and Informatics, SAMI 2011*, Slovakia, 2011. pp. 355-359. ISBN 978-1-4244-7428-8.

[13] SIMA, D. – FONTAIN, T. – KACSUK, P. *Korszerű számítógép-architektúrák tervezéstér-megközelítésben*. Szak Kiadó Kft., Bicske, 1998. ISBN 963-9131-09-1.

[14] SWANSON, S. – MICHELSON, K. – SCHWERIN, A. – OSKIN, M.: WaveScalar. In *Proc. of the 36th International Symposium on Microarchitecture,* 2003. pp. 291-302, ISBN 0-7695-2043-X.

[15] VERDOSCIA, B. – VACARRO, R. ALFA. A Static Data Flow Architecture. In *Proceedings of Fourth Symposium on the Frontiers of Massively Parallel Computation*, McLean, VA, USA, 1992. pp. 318-325. ISBN 0-8186-2772-7.

[16] VOKOROKOS, L. *Princípy architektúr počítačov riadených tokom údajov.* Košice: Copycenter, spol. s.r.o., 2002. ISBN 80-7099-824-5.

[17] WINTZLAW, D. – GRIFFIN, P. – HOFFMANN, H. at al. On-Chip Interconnection Architecture of the Tile Processor. In *IEEE Micro*, vol. 27, no. 5, 2007. pp. 15-31.