Vypracované otázky z PPS 2019

1.T1: Úvod do paralelných počítačových systémov (5)

- **Definícia paralelnej architektury**

Paralelný počítač (paralelná architektúra) **je skupina výpočtových prvkov**, ktoré spolupracujú za účelom **rýchleho riešenia rozsiahlych problémov**.

Rozvoj VT (**výpočtová technika**) v súčasnosti charakterizuje celý rad faktorov, spomedzi ktorých medzi najvýznamnejšie patria:

- **hromadné využívanie VT**, vývoj informačných technológií, vývoj nových výpočtových prostriedkov.

- **Atuálne trendy v IT v kontexte**

Hromadné využívanie VT v súčasnosti ovplyvnilo jej extrémne prenikanie do širokého spektra výrobných ale i nevýrobných oblastí, štátneho a privátneho sektora spoločnosti.

- spracovanie **nenumerických údajov** (signály, symboly, znalosti, databázy),
- tvorbu **informačných a riadiacich systémov**, s osobitným dôrazom na systémy reálneho času, pre širokú triedu aplikačného určenia, automatizáciu inžinierskych prác (AIP), do ktorej patria najmä:

o počítačom podporované navrhovanie - **CAD (Computer Aided Design)**

o počítačom podporovaná výroba - **CAM (Computer Aided Manufacturing)**

o počítačom integrované riadenie výroby - **CIM (Computer Integrated Manufacturing)**

o počítačové navrhovanie softvérových produktov - **CASE (Computer Aided Software Engineering)** a pod.

o aplikácie komunikačných procesov (elektronické vzdelávanie, elektronické obchodovanie, ...),

o aplikácie počítačovej diagnostiky a riadenie experimentov

- **zber, prenos a ukladanie** informácií, **zabezpečenie** procesov ich spracovania a **riešenie komunikačných procesov** pre potreby aplikácií **a prípravy** ich **implementačného prostredia**. Najmä:

- počítačové siete: LAN, MAN, WAN
- Komunikačné a sieťové prostredia (Internet, Intranet)
- Počítačová grafika a virtuálne realita
- Tvorba virtuálnych prostredí
- E-learning a e-business
- **Technologické trendy**
  - **Internet of things,** Sieťové počítanie - **Gridcomputing**
  - Vysoký dôraz na - **cenu, škálovateľnosť, spoľahlivosť**

- uplatnenie nových fyzikálnych princípov(**nové polovodičové materiály**, optoelektronika, laserova technika, supravodičová technika, **kvantová fyzika**),

- technológia výroby stavebných komponentov(medené a molekulové technológie, **molekulárne nanotechnológie**, kvantové technológie umožňujúce dosiahnuť **vysoký počet prvkov na čipe**),

masívne paralelné architektúry počítačových systémov, počítačové a telekomunikačné sieťové prostredia, **náhrada** drahých **superpočítačov** menej nákladnými a flexibilnejšími **clustrami**).

2. T2: Význam paralelných architektúr(5)

- **Programovacie modely**

Definujú, čo použije programátor pri programovaní aplikácie. Špecifikujú komunikáciu a synchronizáciu.

- **Multiprogramming** – žiadna komunikácia/synchronizácia na úrovni programovania
- **SAS** - Shared Address Space – ako súhrnná tabuľa
- **MP** - Message Passing – ako listy alebo volania cez telefón, explicitne z bodu do bodu
- **Data parallel** – viac usporiadaná, globálne akcie na dáta, implementovaný pomocou SAS alebo MP
- **Charakteristika SMP (Symmetric MultiProcessing) (SAS) architektúr**
- Ktorýkoľvek procesor môže **priamo** pristupovať k ľubovoľnému pamäťovému miestu. - Implicitná komunikácia – ako výsledok LOADov a STOREov.
- Časť adresného priestoru procesov je zdieľaná.
- OS používa zdieľanú pamäť na koordináciu procesov - tiež známy ako **shared memory model**.

Výhody:

- Priehľadnosť pamäťových miest
- Podobný programovací model ako time-sharing na uniprocesoroch. Výnimkou sú procesy bežiace na rôznych procesoroch. Dobrá priepustnosť na multiprogramovanej záťaži.
- Poskytovaná mnohými platformami

Príklad: Intel Pentium Pro Quad, Sun Enterprise

SMP: **Procesory - každý procesor vidí všetko** – pamäť I/O, prerušenia atď., sú identické, zdieľajú len jednu pamäť. Vlákno môže byť alokované na ľubovoľnom procesore.

Existuje **iba jedno jadro** – to rozhoduje o tom, **ktorá aplikácia** bude **pridelená ktorému procesoru**, nezdieľajú cache, ale hybridné cache architektúry sú už na ceste.

SMP systémy môžu byť **zložité na** nastavenie a **údržbu** kvôli množstvu HW (ako napr. chladiče pre procesory) - **hlučné**, zla škálovateľnosť, **slabá** využiteľnosť **cache** (potreba vyprázdnenia pri migrácii procesu).

Výskyt predbiehania procesorov – race conditions => nutnosť riešiť multiprocesorovú synchronizáciu.

- **Charakteristika MP architektúr**

Komunikácia explicitnými **I/O operáciami**. Programovací model: **priamy prístup iba na lokálnu pamäť**, komunikácia explicitnými **správami** (send/receive)

Príklady: IBM SP-2, Intel Paragon

- **Charakteristika Data Parallel architektúr**

operácie vykonané **paralelne** pre **každý element** dátovej **štruktúry**, logicky jedno riadiace vlákno vykonáva sekvenčné alebo paralelné kroky Model architektúry:

- pole viacerých jednoduchých a lacných procesorov s malou pamäťou pripojených k riadiacemu procesoru, ktorý riadi inštrukcie
- všeobecná komunikácia, lacná **globálna synchronizácia**

3. T2: Kľúčové aspekty návrhu paralelných architektúr (5)

Paralelný program sa skladá z jedného alebo viacerých riadiacich vlákien, ktoré riadia dáta. Paralelný programovací model špecifikuje:

- Ktoré dáta môžu byť pomenované vláknami
- Aké operácie môžu byť vykonané na týchto dátach
- Poradie týchto operácií
- **Kľúčové aspekty:**
- ***naming*** – ako sú zdieľané dáta a procesy označované
- ***operations*** – aké operácie sú vykonané na týchto dátach
- ***ordering*** – poradie prístupov k dátam
- ***replication*** – ako sú dáta replikované na redukovanie komunikácie
- ***communication cost*** – oneskorenie, priepustnosť, šírka pásma atď.
- **Sekvenčný, SAS a MPP model programovania**

Sekvenčný model programovania

- Naming: ľubovoľná premenná môže byť pomenovaná vo virtuálnom adresnom priestore; HW a/alebo prekladače prekladajú tieto mená na adresy
- Operácie – LOAD, STORE
- Ordering – sekvenčné vykonávanie
    - spolieha sa na závislosti na jednom mieste - **závislé poradie**
    - compiler - preusporiadanie a alokácia registrov
    - **compiler/hardvér môže zmeniť poradie**

SAS (Shared Address Space) model programovania

- Naming – ľubovoľný proces môže pomenovať ľubovoľnú premennú v zdieľanom priestore
- Operations – LOAD, STORE, a tie, ktoré sú potrebné k **ordering**
- **Ordering modely:**
    - V rámci procesu/vlákna : sekvenčné vykonávanie
    - Medzi vláknami: prelínanie
    - Ďalšie poradie cez synchronizáciu

MP (Message Passing) model programovania

- Naming – procesy môžu **pomenovať privátne dáta** priamo (nie je zdieľaný priestor)
- Operations – explicitná komunikácia cez **send a receive**, posielanie dát zo súkromného priestoru iným procesom
- Ordering:
    - Program **order v rámci procesu**
    - Send a receive môže poskytnúť z bodu do bodu synchronizáciu medzi procesmi

Je možné vytvoriť globálny zdieľaný priestor – číslo procesu a adresa z adresného priestoru procesu.

4. T3: Klasifikácia paralelných architektúr (2)

Klasifikácia paralelných a počítačových systémov (Flynn, Kuck, Shore, Feng, Erlangen, Giloi, Ungerer, Skillicorn)

Flynnova *klasifikácia* vychádza z počtu súčasne spracúvaných tokov inštrukcií a tokov údajov v počítači.

1. ***SISD*** *(Single Insíruction stream Single Data stream)* - Jeden tok inštrukcií, jeden tok údajov. Táto architektúra predstavuje architektúru Von Neumannovho počítača
2. ***SIMD -*** Táto architektúra má významné použitie pri tvorbe paralelných počítačov.
3. ***MISD -*** Niekoľko tokov inštrukcií, jeden tok údajov.
4. ***MIMD -*** *napr. multiprocesorové* a *multipočítačové* systémy s paralelným spracovaním

Kuckova klasifikacia:

1. **SISSES** - Single Instruction Scalar, Single Execution Scalar
2. **SISSEA** - -||- , Single Execution Array
3. **SIASEA** - Sing instruction Array, -||-
4. **MISMES** - Mult. Instr. Scalar, Mult. Exec. Scalar
5. **MISMEA** - Mult.In.Sc., Mult. Ex. Array

Fengova klasifikácia - Systémy popisuje pomocou dvojice čísel. Prvé číslo reprezentuje veľkosť slova a druhé počet slov, ktoré je možné spracovať v jednom čase.

1. **WSBS** – word-serial, bit-serial (bit-serial)
2. **WSBP** – word-serial, bit-parallel (word-slice)
3. **WPBS** – word-parallel, bit-serial (bit-slice)
4. **WPBP** – word-parallel, bit-parallel (fully paralel)

Ungererova klasifikácia

- **Vonkajšia architektúra** - Popisuje správanie sa počítačového systému so špecifikáciou softvérového prepojenia.
- **Vnútorná architektúra** - Abstraktný popis vnútornej štruktúry a funkcionality pozostávajúci z
- **počítačová štruktúra** – statická topológia hw zdrojov a ich prepojenie
- **informačná štruktúra** – interná reprezentacia kodu a dat na hw zdrojoch
- **operačné princípy** – reprezentacia manipulacie s informačnou strukturou sposobenou pocitacovou strukturou

Shoreova klasifikacia - Control Unit (CU), Processing Unit (PU), Instruction Memory (IM), Data Memory (DM)

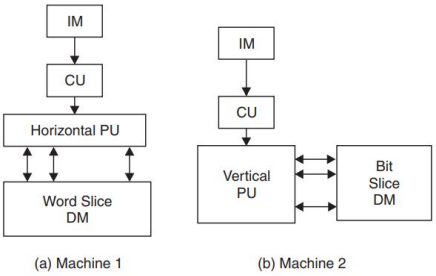


Fig. 1.15 Horizontal and Vertical Processing

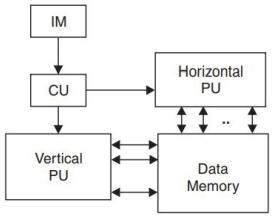


Fig. 1.16 Machine 3 Architecture

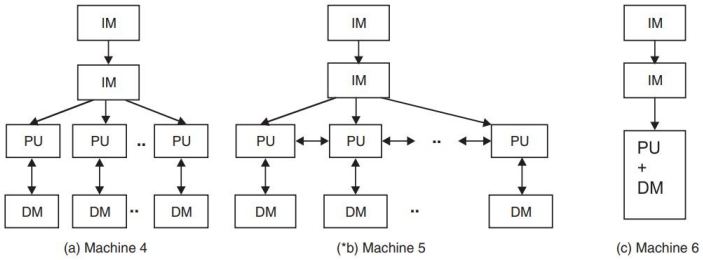
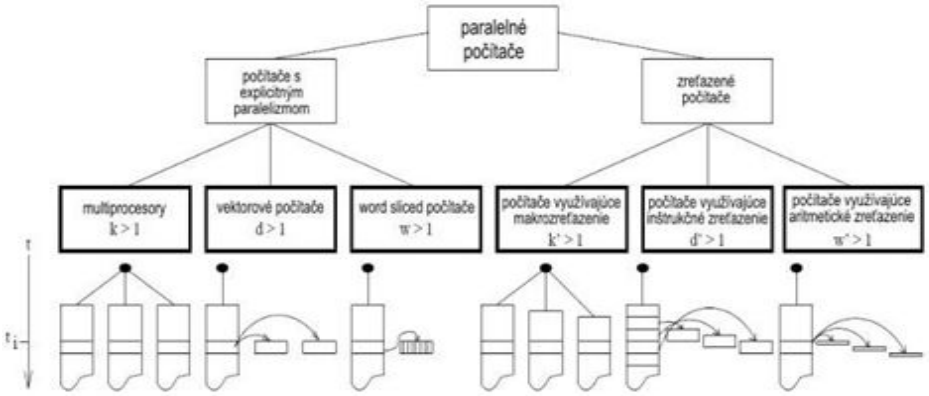


Fig. 1.17 Parallel and Associative Architectures

Erlangenova klasifikacia



Systém je popísaný trojicou (*k*k'*, *d*d'*, *w*w'*), kde:

- · *k* počet riadiacich jednotiek
- · *d* počet výpočtových jednotiek na jednu riadiacu jednotku
- · *w* veľkosť jedného dátového slova

- · *k'* počet špecializovaných riadiacich jednotiek
- · *d'* počet špecializovaných výpočtových jednotiek
- · *w´* počet zreťazených stupňov

Paralelné počítače:

1. počítače s explicitným paralelizmom:

    1. *k* > 1 – *multiprocesory* – MIMD different programs
    2. *d* > 1 – vektorové počítače – SIMD single program
    3. *w* > 1 – word sliced počítače – SI several bits of SD stream

2. zreťazené počítače

    1. *k'* > 1 – počítače využívajúce makrozreťazenie – MI of different programs SD
    2. *d'* > 1 – počítače využívajúce inštrukčné zreťazenie – MISD single program
    3. *w'* > 1 – počítače využívajúce aritmetické zreťazenie – MI parts of one program SD

Skillicorn-ova klasifikácia

- Skillicorn vypracoval počítačový systém pozostávajúci zo štyroch funkčných jednotiek: inštrukčný procesor, dátový procesor, hierarchia pamäti a prepínač.
- Existujú tri možné prepojenia – žiadne; n -n prepojenie medzi n párov (po dvojiciach) a n x n: prepojenie medzi n jednotkami na jednej strane a n jednotkami na druhej strane

Giloiova klasifikácia

Počítačové architektúry rozdelené podľa operačných princípov hardvéru a jeho štruktúry pozostávajúcej z niekoľkých hardvérových modulov

Operačné princípy definujú funkčné správanie sa architektúry viazanej na informačnú aj riadiacu štruktúru.

Triedenie podľa operačných princípov a počítačových architektúr.

Abstraktný model je špecifikovaný:

- počtom inštrukčných procesorov (IP)
- počtom inštrukčných pamätí
- typom prepojenia medzi IP a IM

- počtom dátových procesorov (DP)
- počtom dátových pamätí (DM)
- typom prepojenia medzi DP a DM
- typom prepojenia medzi IP a DP

Vypočtové modely (CF, DF, DD)

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:

- **CF** - riadenie sa výpočtového procesu uskutočňuje prostredníctvom interpretácie **sériového prúdu inštrukcií** programu.4. Kvantifikátory a kva
- **DF** - pri ktorom sa riadenie výpočtového procesu uskutočňuje prostredníctvom **prúdu operandov** (údajov), definujúcich pripravenosť inštrukcií na vykonanie
- **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

5. T3: Modely paralelných architektúr (1)

Teoretické modely (RAM, PRAM)

RAM (Random Access Machine)

Je počítač s jedným akumulátorom, v ktorom inštrukcie majú zakázané modifikovať sa.

Model RAM pozostáva: zo **vstupnej pásky *iba z možnosťou čítania***, jedného **procesora**, **výstupnej pásky *iba s možnosťou zápisu***, **programu** a **pamäťovej jednotky**. Program tvorí konečná postupnosť inštrukcií s návestím, ktorá však nie je uložená v pamäti, ale (rovnako ako aj vstup) sa načítava zo vstupnej pásky. Programy modelu RAM môžu simulovať deterministické aj nedeterministické Turingove stroje s časovou polynomiálnou zložitosťou a ohraničenosťou priestoru v rámci určitej konštanty.

PRAM (Parallel Random-Access Machine)

Teoretický model paralelného počítača. Pozostáva z **N deterministických RAM procesorov**, ktoré môžu pracovať paralelne a komunikovať cez spoločnú pamäť. Počet procesorov N je funkciou veľkosti vstupu *n.* Každý procesor má **nekonečný počet** univerzálnych **registrov** (viď. RAM), **jednotnú pamäť typu ROM** s označením **X(i)** a **spoločné pamäťové bunky C(i)**.

Program pozostáva z konečnej množiny **príkazov**, pričom každá inštrukcia je v tvare:

1. **Vykonáva sa vnútorný výpočet**
2. **Prenos riadenia**
3. **Čítanie hodnoty zo spoločnej pamäti**
4. **4. Zapísanie hodnoty do spoločnej pamäti.**

1 a 2 = lokálne inštrukcie, 3 a 4 = komunikačné príkazy medzi rozličnými procesormi V každom kroku každý

procesor je v určitom stave.Činnosti a nasledujúci stav každého procesora závisí od jeho súčasného stavu a

hodnôt, ktoré sa prečítajú z ROM X(i) a zo spoločnej pamäti. *Súbežné čítanie* nie je potrebné špeciálne riešiť, pri súbežnom zapisovaní vzniká potreba riadiť prístup.

Rozličné varianty PRAM sa líšia v tom ako ošetruje kolízie pri zápisoch:

- **CREW (Concurrent Read Exclusive Write):** paralelné čítanie, kolízie zápisu sú vylúčené mechanizmom vzájomného vylúčenia
- **COMMON:** všetky procesory súčasne zapisujú tú istú hodnotu do tej istej bunky
- **ARBITRARY:** vybraný procesor realizuje operáciu zápisu
- **PRIORITY:** procesor s najvyššou prioritou realizuje operáciu zápisu

EREW PRAM - Exclusive-read, exclusive-write, CREW PRAM - Concurrent-read, exclusive-write,

- ERCW PRAM - Exclusive-read, concurrent-write, CRCW PRAM-Concurrent-read, concurrent-write

Modely tesne viazaných multiprocesorov: UMA, NUMA, COMA, SUMO

- Zdieľajú spoločnú pamäť (Shared Memory Multiprocessors). Sú označované aj ako **tesne viazané multiprocesory.**

UMA (Uniform Memory Access) – Modely s rovnomerným prístupom Fyzická **pamäť** je **rovnomerne využívaná** všetkými procesormi.

Prístupový čas k jednotlivým slovám pamäti je rovnaký (jednotlivé procesory môžu mať k dispozícii privátne vyrovnávacie pamäte typu cache). Využíva sa na **zrýchlenie vykonania jediného rozsiahleho programu** pre časovo náročné aplikácie (aplikácie reálneho času) alebo na multiprogramový režim práce.

Na koordináciu paralelného spracovania úloh (synchronizácia a komunikácia medzi procesormi) sa využívajú **zdieľané premenné v spoločnej pamäti**.

V závislosti od využívania pamäte všetkými procesormi alebo iba ich časťou sa rozlišujú: symetrické modely UMA a nesymetrické modely UMA.

NUMA (Non-uniform Memory Access) – Modely **s nerovnomerným prístupom**

Zdieľaná pamäť je fyzicky distribuovaná procesorom prostredníctvom lokálnych pamäti (LM).

Každý procesor má svoj „kus" pamäti

- Prístupový čas do lokálnych pamätí je rôzny podľa toho kde pamäť fyzicky je – non-uniform
- Problém ak proces „odmigruje" príliš ďaleko na iný procesor

Množina lokálnych pamätí tvorí globálny adresový priestor jednotlivých procesorov.

Multiprocesorové systémy delíme ich na

- MS so zdieľaním lokálnych pamätí, vyznačujúcich sa rýchlym prístupom do vlastnej lokálnej pamäte,
- Hierarchicky štruktúrované MS so skupinovou („klastrovou") organizáciou, ktoré vytvárajú základ pre

tvorbu kombinovaných paralelných štruktúr.

COMA (Cache Only Memory Architecture) – Modely s vyrovnávacou pamäťou

Predstavuje **špeciálny prípad modelu NUMA**, v ktorom distribuovaná hlavná pamäť je konvertovaná do modulov vyrovnávacích pamätí typu "cache„.

Moduly cache reprezentujú globálny adresový priestor. Prístup do vzdialených cache sa uskutočňuje prostredníctvom adresárov cache.

Prostredníctvom vhodného typu prepojovacej siete sa sprístupňujú údajové bloky príslušnému procesoru. Prostredníctvom smerovačov je možné vytvárať hierarchické štruktúry. Toto má význam kvôli škálovateľnosti (scalability).

SUMO (Sufficiently Uniform Memory Organization)

- AMD Opteron
- Fyzicky podobné NUMA
- Pre OS sa tvári ako SMP - **Rieši otázku ako spustiť normálne jadro OS na NUMA**

Multipočítače

Sú označované aj ako **voľne viazané multiprocesory.** Pozostávajú z **uzlových počítačov** prepojených pomocou **statickej prepojovacej siete** prenosu správ – de facto sieť pracovných staníc (procesných elementov).

Každý uzol je plnohodnotný počítač na ktorom môže bežať niekoľko procesov súčasne. Napr. cluster (počítače prepojené veľmi rýchlou lokálnou sieťou). Disponujú distribuovanou pamäťou. Dedí problémy multiprocesorov s distribuovanou pamäťou Lokálne pamäte sú privátne, t. j. prístupné iba lokálnym procesorom.

Medziprocesorové komunikacie sa uskutočňujú **prenosom správ**.

- MPP architektúry
- Rozhranie MPI

6. T4: Charakteristické vlastnosti paralelizmu (1)

Podmienky paralelného vykonania programu (hazardy)

Paralelné vykonanie inštrukcií programu je závislé od splnenia podmienok, ktoré definujú stupeň paralelizácie procesov vykonania programu.

Údajové závislosti

- údajová zavislost **RAW** - citanie premennej po jej zapise
- údajová nezavislost **WAR** - zapis premennej po jej citani
- vystupova zavislost **WAW** - zapis premennej po jej zápise
- **V/V závislosť**
- **nezistiteľná závislosť** - index premennej je sám sebe indexom, adresovanie cyklov

Riadiace závislosti

- vyplývajú z dvoch typov riadenia cyklov, ktore mozu byt nezávisle alebo závisle.

Zdrojové závislosti

- Ak dve inštrukcie požadujú k ich vykonaniu rovnakú funkčnú jednotku, nemôžu sa súčasne realizovať v dôsledku zdrojovej závislosti
- odstránenie sa uskutočňuje zväčšením počtu funkčných jednotiek Graf závislosti

1. Pre účely analýzy paralelizácie procesov sa závislosti spravidla zobrazujú grafom závislosti

Bernsteinove podmienky

- Procesy Pi a Pj môžu byť vykonané paralelne, ak medzi množinami vstupov Ii a Ij a množinami ich výstupov Oi a Oj platí:

$$O_i \cap I_j = \emptyset \quad \text{podmienka RAW}$$
$$I_i \cap O_j = \emptyset \quad \text{podmienka WAR}$$
$$O_i \cap O_j = \emptyset \quad \text{podmienka WAW}$$

- Podmienky RAW, WAR, WAW sú definované údajovými závislosťami pri vykonávaní inštrukcií programu.
- Pri paralelizácii procesov sa riadiace a zdrojové závislosti analyzujú osobitne.

7. T4: Kvantifikátory a kvalifikátory paralelných počítačových systémov(2)

- **Škálovateľnosť PPS**

1. **Pracovná záťaž – P** (workload) je množstvo výpočtov vykonaných v paralelnom výpočtovom prostredí.
2. **Rozmer problému – s ,**resp. zložitosť výpočtového procesu, kvantifikuje **veľkosť pracovnej záťaže**. **Strojový rozmer – n** kvantifikuje paralelné výpočtové prostredie prostredníctvom počtu jeho procesorov.´

Charakteristika paralelných algoritmov:

- **determinizmus** (deterministické algoritmy sú implementovateľné na reálnom stroji, hodnotené časovou zložitosťou výpočtového procesu),
- **granularita výpočtového procesu** (granularita špecifikuje rozmer údajových položiek a programových modulov výpočtu, v tomto zmysle sú klasifikované algoritmy: jemnozrnné, strednozrnné a hrubozrnné),
- **profil paralelizmu** (rozdelenie stupňa paralelizmu v algoritme odhaľuje možnosť paralelného spracovania),
- **komunikačná réžia** (komunikačná réžia zahrňuje adresovanie pamäťového prístupu a medziprocesorových komunikácií, ktoré v závislosti na algoritme môžu byť statické alebo dynamické; statické algoritmy sú vhodné pre prúdové architektúry alebo SIMD, zatiaľ čo dynamické algoritmy, pre MIMD),
- **uniformita operácií** (jednotnosť operácií špecifikuje základné operácie, ktoré sú definovaná v danom algoritme),
- **vplyv údajových štruktúr** (údajové štruktúry majú pri riešení rozsiahlych výpočtoch zásadný vplyv na veľkosť pamäťového priestoru, účinnosť pamäte a pohyb údajov pri riešení algoritmu).

Závislosť pracovnej záťaže (rozmeru problému - s) na strojovom rozmere (n) pre rôzne charakteristické situácie (vzory), zobrazuje nasledujúci **diagram charakteristík pracovnej záťaže**, z ktorého vyplýva i príslušný **diagram kriviek účinnosti**.

- **Vzory pracovnej záťaže a aplikačných modelov**

Pracovná záťaž (s)	Aplikačné modely
konštantná (K)	s konštantnou **pracovnou záťažou** (fixed–**load** model) – Mload
lineárna (L)	s konštantnou **dobou vykonania** programu (fixed–**time** model) – Mtime
sublineárna (S)	modely v oblasti kriviek K a L
exponenciálna (E)	s konštantnou **kapacitou pamäti** (fixed–**memory** model) – Mmem.

- **Model MK**
  pracovná záťaž je konštantná, model môže byť prípadne limitovaný **komunikačným ohraničením (KB)**.
- **Model MT**
  požaduje konštantnú dobu vykonania programu bez ohľadu na to, ako pracovná záťaž sa zväčšuje v závislosti na strojovom rozmere,
  model korešponduje s lineárnym nárastom pracovnej záťaže.
- **Model MM**
  je limitovaný **pamäťovým ohraničením (MB)**, korešponduje s oblastou medzi krivkami L a E.

Škálovateľnosť

- Pojem škálovateľnosť v počítačovom prostredí môžeme chápať ako **schopnosť systému pružne reagovať na rastúce alebo klesajúce nároky užívateľov na výkon systému.**
- **Definovať škálovateľnosť nie je jednoduché**, lebo v konkrétnych prípadoch je vždy nutné presne definovať parametre systému, ktoré sú pre daný systém kľúčové.
- **Počítačový systém alebo program je škálovateľný, ak účinnosť systému ostáva konštantná aj pri rastúcom počte procesných elementov.**
- Analýza škálovateľnosti určuje (stanovuje), či paralelné spracovanie daného problému poskytuje požadované zlepšenie výkonnosti.
- Paralelné algoritmy implementované v reálnom prostredí sú závislé na konkrétnej realizácii tohto prostredia. Sú preto vždy závislé na architektonickom riešení.
- V ideálnom prípade sú paralelné algoritmy definované pre modely PRAM.
- Analýza škálovateľnosti paralelného algoritmu vychádza z jeho rovnakej (konštantnej) účinnosti (izoúčinnosti) v danom implementačnom prostredí.

Koncepcia izoúčinnosti (E) umožňuje vytvoriť vzťah medzi pracovnou záťažou definovanou rozmerom problému *s* a strojovým rozmerom *n* tak, aby sa udržovala konštantná účinnosť pri implementácii paralelného algoritmu v paralelnom počítači. Izoúčinnosť E je definovaná vzťahom:

- Ak hodnota P(s) rastie tak rýchlo ako hodnota $f_E(n)$, potom je možné dosiahnuť konštantnú účinnosť danej kombinácie algoritmu a architektúry. *Uvedené paralelné výpočtové prostredie sa nazývajú škálovateľné.*
- Škálovateľnosť má priamu súvislosť so **zrýchlením výkonnosti** paralelného počítačového systému (PPS).
- Pre analýzu škálovateľnosti je definované asymptotické zrýchlenie S(s, n). Vo všeobecnosti, S(s,

n) sa zvyšuje lineárne, so zreteľom na počet procesorov používaných pre danú aplikáciu s ohraničením, špecifikovaným **Amdahlovým zákonom**.

- **Asymptotické zrýchlenie výkonnosti**

je najlepšie zrýchlenie dosiahnuteľné pri danej **architektúre**, **algoritme** a **rozmere problému** v závislosti na premenlivom počte procesorov ***n***:

Asymptotické zrýchlenie výkonnosti je **najlepšie** dosiahnuteľné **zrýchlenie** pri danej **architektúre**, **algoritme** a **rozmere problému** od počtu procesorov:

$$S(s,n) = \frac{T(s,1)}{T(s,n) + h(s,n)}$$

kde:
$T(s,1)$ - doba **sekvenčného vykonávania** algoritmu v **jedno**-procesorovom systéme
$T(s,n)$ - **minimálna** doba **paralelného vykonávania** algoritmu v n-procesorovom systéme
$h(s,n)$ - doba **celkovej komunikačnej réžie**

Analýza škálovateľnosti sa robí pre riešenie **rozsiahlych** problémov, ktorých **systémová účinnosť** je:

$$E(s,n) = \frac{S(s,n)}{n}$$

Analýza škálovateľnosti sa robí pre riešenie **rozsiahlych** problémov, ktorých **systémová účinnosť** je:

$$E(s,n) = \frac{S(s,n)}{n}$$

Maximálna hodnota $E(s,n) = 1$, keďže **najlepšie** zrýchlenie výkonnosti je **lineárne**.

Paralelný počítačový systém je **škálovateľný**, ak hodnota **systémovej účinnosti** $E(s,n) = 1$ platí pre **všetky** algoritmy s počtom n **procesorov** a **rozmerom problému s**.

Pri **zanedbaní komunikačnej réžie** v prostredí modelu **PRAM** je **asymptotické zrýchlenie**:

$$S_I(s,n) = \frac{T(s,1)}{T_I(s,n)}$$

kde $T_I(s,n)$ je **doba** paralelného výpočtu v modeli **PRAM**.

Škálovateľnosť $\Phi(s,n)$ pre daný algoritmus je:

$$\Phi(s,n) = \frac{S(s,n)}{S_I(s,n)} = \frac{T_I(s,n)}{T_I(s,n)}$$

8. T4: Zákony paralelných počítačových systémov (1)

Výkonnosť paralelného prostredia

Aritmetická priemerná výkonnosť (RA) výpočtového procesu je definovaná vzťahom:

$$R_A = \frac{1}{m}\sum_{i=1}^{m} R_i \Rightarrow \frac{1}{R_A} = T_A$$

kde

R_i [op/čj] je priemerná operačná rýchlosť i -teho programu (i = 1, 2, ..., m),

T_A je priemerná doba vykonania m programov.

vážená aritmetická priemerná výkonnosť (R_A^*) definovaná vzťahom:

$$R_A^* = \sum_{i=1}^{m} (f_i R_i)$$

kde f_i sú hodnoty váhovej funkcie pravdepodobnostného rozloženia $w = \{f_i \mid i = 1, 2, ..., m\}$, pre ktoré platí:

$$\sum_{i=1}^{m} f_i = 1$$

- **AMDHALOV ZAKON**
- Sformulovaný v roku 1967 nórsko-americkým počítačovým architektom Genem Myronem Amdahlom.
- Pri štúdiu výkonnosti paralelných aplikácií si Amdahl všimol, že paralelizáciou problému môžeme dosiahnuť len relatívne veľmi obmedzené zrýchlenie.
- Osobitný **význam má stredné harmonické zrýchlenie výkonnosti** určené pre paralelné prostredie, ktoré **využíva iba jeden procesor v sekvenčnom móde** s pravdepodobnosťou **alebo všetky procesory v úplne paralelnom móde s pravdepodobnosťou 1** – . Váhová funkcia v tomto prípade je **w = (, 0, …, 0, 1 –)**
- Za predpokladu, že R_i = I

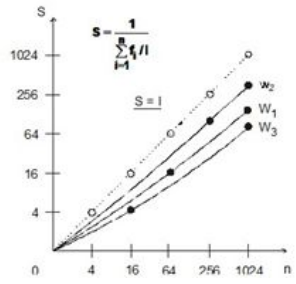
Závery vyplývajúce z predchádzajúceho vzťahu:

- pri neobmedzenom raste počtu procesorov ($n \to \infty$) je výkonnosť obmedzená hodnotou

  $S = 1 / \alpha$,

- v ideálnom prípade je α = 0, v dôsledku čoho

  $S = n$.



Ak budeme riešiť problém, kde sekvenčná časť bude tvoriť 10%, maximálne zrýchlenie bude 10.

Nezáleží na tom, či problém bude riešiť 4, 16 alebo 1024 procesorov

- Je dôležité si uvedomiť, že **Amdahl rieši problém o konštantnej veľkosti** - tzn. s rastúcim počtom procesorov sa nemení rozsah úlohy.

- Snaží sa vyriešiť daný problém čo najrýchlejšie.
- **GUSTAFSONOV ZAKON**
- Amdahlov zákon značne obmedzuje maximálne zrýchlenie výpočtu.
- Pre už pomerne nízke hodnoty sekvenčnej časti nemôžeme dosiahnuť primerané zrýchlenie.
- V roku 1988 John Gustafson prišiel s novým pohľadom na vec.
- Amdahl s rastúcim počtom procesorov nemenil veľkosť problému.
- Gustafson si všimol, že zrýchlenie u praktických úloh **je výrazne vyšší než predpovedal Amdahl.**
- **Dôležitou zmenou je, že miesto konštantného rozsahu úlohy budeme za konštantnú považovať dobu behu.**
- Gustafson si uvedomil, že **s vyšším počtom procesorov** je možné rovnaký problém vyriešiť v rovnakom čase presnejšie.

Znamená to, že pre dostatočne veľký problém môžeme dosiahnuť **ľubovoľné zrýchlenie**

9. T5: Prúdové spracovanie vektorizovaných programov v PPS (5)

- **Prístupy spracovania vektorizovaných programov**
- **Vektorizácia skalárneho programu**

1. Prístupy spracovania vektorizovaných programov

Vektorizácia

prúdového spracovania skalárnych inštrukcii programu alebo
paralelného spracovania vektorových inštrukcii programu.

Tieto prístupy sú ilustrované na príklade rozkladu programu cyklu vo verziách s aplikovaním skalárnych operácií a s aplikovaním vektorových operácií.

Príklad programu cyklu:

```
    DO 10 I = 1,N
    Z(I) = Z(I) + X(I) × Y(I)
10 continue
```

Rozklad programu s použitím skalárnych operácii. Formát 64 bitovej skalárnej inštrukcie FSI je:

⟨FSI⟩ ::= ⟨KSO(1..8)⟩ ⟨R1(1..6)⟩ ⟨M [R2(1..50)]⟩

kde je

KSO - kód skalárnej operácie
R1 – adresa prvého skalárneho operanda v registrovej pamäti definovaná obsahom registra (R1)
R2 – adresa druhého skalárneho operanda v pamäti M definovaná obsahom registra R2 v tvare:

(R2) = (B2) + (X2) + disp2

kde je
B2 - bázový register druhého operanda,
X2 – indexový register druhého operanda,
disp2 – posuv (displacement) druhého operanda.

Skalárne inštrukcie programu:	Sémantika inštrukcii:
LD R1, M [R2]	R1 ← M [R2] ≡ R1(I) ← X(I)
MUL R1, M [R2]	R1 ← R1 × M [R2] ≡ R1(I) × Y(I)
ADD R1, M [R2]	R1 ← R1 + M [R2] ≡ R1(I) + Z(I)
ST R1, M [R2]	M [R2] ← R1 ≡ Z(I) ← R1(I)
BR disp2	if I ≤ N then PC := PC + step + disp

Ak doba T_{SC} vykonania skalárnej inštrukcie je 1SC, uvedený program sa v jednoprocesorovom systéme vykoná **sekvenčne** za 5 N × T_{SC} [SC]. Uvedený postup vykonania programu so skalárnymi údajmi je charakteristický pre architektúry typu SISD (sekvenčné počítače). Efektívnejšie vykonanie programu umožňujú vektorové operácie vo viacprocesorovom systéme.

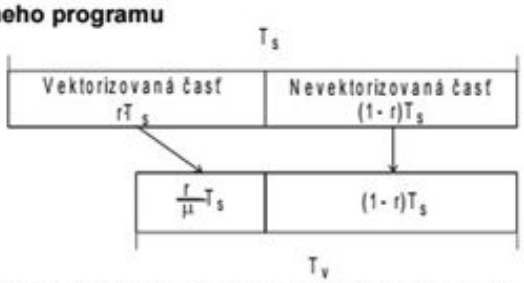
Rozklad programu s použitím vektorových operácii. Formát 64 bitovej vektorovej inštrukcie FVI je:

⟨FVI⟩ ::= ⟨KVO(1..8)⟩ ⟨VR1(1..8)⟩ ⟨M [VR2(1..50)]⟩

Kde je

KVO - kód vektorovej operácie
VR1 – adresa prvého vektorového operanda v registrovej pamäti definovaná obsahom vektorového registra (VR1)
VR2 – adresa druhého vektorového operanda v pamäti M definovaná obsahom vektorového registra VR2

Vektorizácia skalárneho programu



T_s doba vykonania programu pri skalárnom spracovaní
T_v doba vykonania programu pri vektorovom spracovaní
r - koeficient vektorizácie
μ - účinnosť vektorového spracovania

10. T6: Prepojovacie siete (3)

- **Základné vlastnosti a použitie prepojovacích sietí**

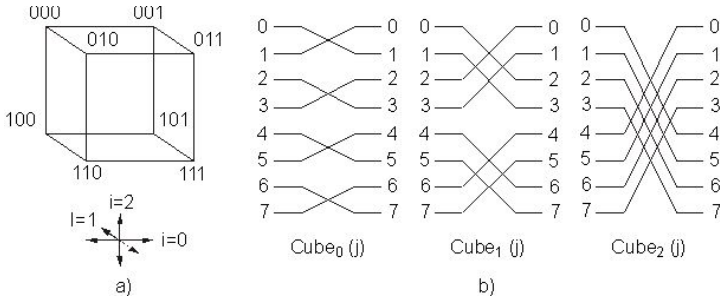
V paralelných počítačových systémoch majú významnú úlohu prostriedky na realizáciu medziprocesorových

komunikácií, ktoré sa organizujú obyčajne ako komunikačné modely typu **P-M** (medzi procesormi a pamäťovými modulmi) alebo **P-P**.

Uskutočňuje sa prostredníctvom rôznych druhov prepojovacích systémov, najvýznamnejšie:

- **Jednoduché zbernice**: časovo prideľované (zdieľané) zbernice - používajú sa **v nenáročných zostavách** paralelných počítačových systémov **s malým počtom prepájaných jednotiek** a najmä vtedy, **keď sa** medzi nimi **nepožaduje súčasné vykonávanie komunikácií**
- **Viacnásobné zbernice**: umožňujú **súčasné vykonávanie medziprocesorových komunikácií** - multiprocesorové systémy. Uplatňujú sa najmä **v konvenčných multiprocesorových systémoch**, v ktorých prepojenie z viacerých zberníc sa uskutočňuje **prostredníctvom viacprístupových jednotiek**
- **Prepojovacie siete**: je obvodový **prostriedok**, ktorý umožňuje prepojiť **ľubovoľný** zo svojich N **vstupov** $X_0, X_1, \ldots, X_{N-1}$ s **ľubovoľným** zo svojich M **výstupov** $Y_0, Y_1, \ldots, Y_{M-1}$. - Táto vlastnosť sa využíva na realizáciu prepojenia N s M komponentmi **pri** organizovaní **komunikácií typu P-M alebo** na realizáciu vzájomného prepojenia N komponentov pri **P-P**. V súvislosti s uvedeným sa rozlišujú **obojstranné a jednostranné PS**, ktoré podľa smerovania komunikácií v použitých aplikáciách môžu byť jednosmerné a obojsmerné.
- **Jednostupňové prepojovacie siete**

Vychádzajúc z definície **prepojovacích funkcií** sú v ďalšom uvedené niektoré najvýznamnejšie typy **jednostupňových prepojovacích sietí**, ktoré vytvárajú **základ** aj **na tvorbu viacstupňových PS**. Jednou z najvýznamnejších PS je prepojovacia sieť typu **n–dimenzionálnej kocky**, ktorej PF:



- **Permutačné siete**

Prepojovacia funkcia v tvare $Y_k = f(X_j)$, ktorou sú v PS definované vyžadované **prepojenia vstupov *j* s jej výstupmi *k*** pre všetky $j, k = 0, 1, \ldots, N-1$, vyjadruje operáciu **permutácií.** Medzi základné typy permutačných operácií patria operácie **dokonalého premiešania**, **inverzného premiešania** a **výmeny**. PS, ktoré realizujú uvedené operácie sa nazývajú **permutačné siete** (PMS).

Funkcie premiešania vyjadrujú **rotačný *k*–násobný posuv vľavo alebo vpravo** adries vstupov j zobrazených ich lineárnou reprezentáciou $\rho = \rho n \ldots \rho 1\ \rho 0$, takže pre ich základné tvary (k = 1) platí:

***shuffle* (*pn* _ 1 *pn* _ 2 ... *p*1 *p*0) = *pn* _ 2 *pn* _ 3 ... *p*1 *p*0 *pn* _ 1**

pri posuve adries **vľavo** (**dokonalé premiešanie**), resp.

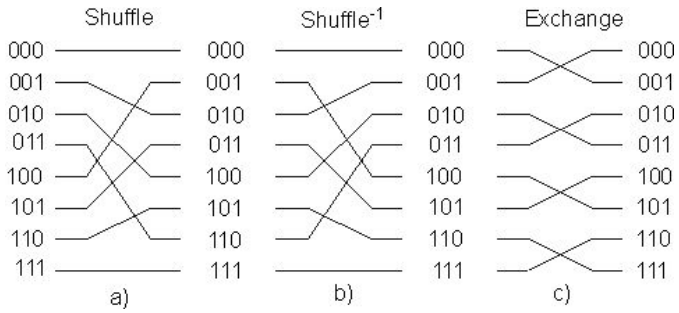
***shuffle* – 1(*pn* – 1 *pn* – 2 ... *p*1 *p*0) = *p*0 *pn* – 1 *pn* – 2 ... *p*2 *p*1**

pri posuve adries **vpravo** (**dokonalé inverzné premiešanie**).

Pri tvorbe PMS sa **permutácia dokonalého premiešania** využíva v spojení **s permutáciou typu výmena**. Permutácia typu výmena sa realizuje prostredníctvom vytvárania **komplementu** v najnižšom ráde adresy *Xi* , takže príslušná permutačná funkcia je vyjadrená v tvare:

***exchange* (*pn* – 1 *pn* – 2 ... *p*1 *p*0) = *pn* – 1 *pn* – 2 ... *p*1 *p*0**

pre ktorú platí *exchange* = *cube*0 . PMS, ktoré realizujú permutácie dokonalého premiešania a výmeny sú uvedené na nasledujúcom obrázku.



11. T6: Prepojovacie siete (3)

- **Permutačné siete (z otazky 10.)**
- **Viacstupňové prepojovacie siete**

Viacstupňové PS sa najčastejšie navrhujú ako PMS, ktoré realizujú množinu permutácií P v tvare:

$P_N = EE \dots E = (E)^m$, kde

- ***PN*** je množina permutácií realizovaná rovnako označovaným typom štvorcovej siete *N N* (P_N sieť),
- typ permutácie realizovanej v jednom stupni *PN* siete, ktorou môže byť **dokonalé premiešanie**: = , resp. = $^{-1}$ definované príslušnými vzťahmi, **preklápanie**, **motýliková permutácia** (= ^{k}) a mnoho ďalších, • E nastavovaná (riadená) sieť typu výmena,
- **M** počet stupňov.

Každý stupeň v *PN* sieti môže byť rozdelený na **segmenty**, v ktorých sa autonómne vykonávajú permutácie nad počtom vstupov *s* (*s* = $N / 2^0$, $N / 2^1$, ..., $N / 2^{n-1}$), čo sa vo vzťahu (1) formálne vyjadruje symbolom (s). Ako príklad sú v ďalšom uvedené niektoré najvýznamnejšie typy viacstupňových *PN* (*N*) sieť, medzi ktoré patria:

Omega sieť (Ω_N sieť) definovaná permutáciou:
$\Omega_N = \sigma_{n-1} E\, \sigma_{n-1} E \ \ldots \sigma_{n-1} E = (\sigma_{n-1} E)^n = (\sigma E)^n$
kde σ prestavuje dokonalé premiešanie a σ_{n-1} predstavuje
(n –1) -„ručné" premiešanie.

Preklápacie sieť (F_N sieť) definovaná permutáciami:
$F_N = E\, \sigma_1 E\, \sigma_1 \ldots E\, \sigma_1 = (E\, \sigma_1)^n$ - predstavuje inverziu siete omega.

Kubická sieť (σ_N sieť) definovaná permutáciou:
$C_N = E\, \beta_1 E\, \beta_2 \ldots E\, \beta_{n-2}\, E\, \sigma_1$ - predstavuje sieť typu n–rozmerovej kocky.

Základná sieť (Z_N sieť) definovaná permutáciou:
$Z_N = E\sigma_1\, E\, \sigma_2 \ldots E\, \sigma_{n-1}\, E = E\, (\sigma_{n-1})^{-1}\, E\, (\sigma_{n-2})^{-1} \ldots E\, (\sigma_1)^{-1} E$ - predstavuje porovnávací normál pri analýze n–stupňových sietí typu SE.

Každá z uvedených PS sa vyznačuje jednoduchým riadiacim mechanizmom. Keď sa použijú prepínače s výberom výstupu, môže sa prepojenie medzi vstupom X_j a Y_k nastaviť riadiacim slovom:

$$C = C_{n-1}\, C_{n-2} \ldots \ C_0 = p_{n-1}\, p_{n-2} \ldots p_0 = (k)_2$$

kde C_r , $r = n-1, n-2, \ldots, 0$ je riadiaca premenná prepínačov r–tého stupňa v PMS. Znamená to, že riadiace slovo reprezentuje lineárne zobrazenie výstupu, ku ktorému sa má zo zadaného vstupu vytvoriť prepojovacia cesta.

- **Údajový manipulátor**
  - Požiadavky na SIMD a MIMD sú rôzne.
  - **SIMD** architektúry:
    - synchrónna architektúry
    - manipulovanie s údajmi
    - permutačna sieť
  - **MIMD** architektúry:
    - Optimálne prepojenie • Údajové manipulátory

    manipulačné funkcie :

    - permutácie
    - kopírovanie
    - rozmiestňovanie
    - maskovanie

12. T7: ILP procesory (4)

- **Úvod do ILP procesorov**

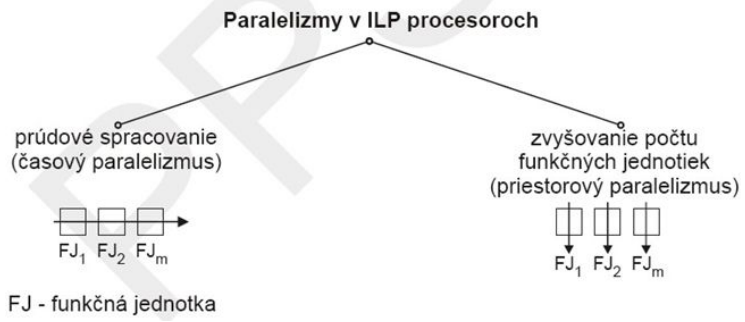
Instruction level parallelism

Vykonnost
Zvysovanie pracovnej frekvencie
Funkcionalne vylepsenie komponentov
Funkcionalne vylepsenie
Vyber instrukcie } Paralelizmus
Vykonanie instrukcie }

Konvencne procesory	**Skalarne ILP procesory**	**Superskalarne ILP procesory**
Sekvencny vyber	sekvencny vyber	paralelny vyber
Sekvencne vykonavanie	paralelne vykonavanie	paralelne vykonavanie
----------→		
	Paralelizmus vo vykonavani instrukcii	
----------→		
	Paralelizmus vo vybere instrukcii	
Jedna funkcna jednotka Bez prudoveh jednotky	jedna funkcna jednotka s prudovou jednotkou	viac funkcnych jednotiek s prudovymi jednotkami (VLIW a superskalarne procesory)
----------→		
	Vykonnost	

- **Paralelizmy v ILP procesoroch (zreťazenie výpočtového procesu)**

Sekvencne kontra paralelne vykonanie
Sekvencne vykonanie } Sekvencia
Vyber instrukcie}
Vykonanie instrukcie
Paralelne vykonanie
Zvysovanie poctu funkncych jednotiek
Prudove spracovanie



Zretazenie vypoctoveho procesu

V riadiacej casti - Na urovni inštrukcie

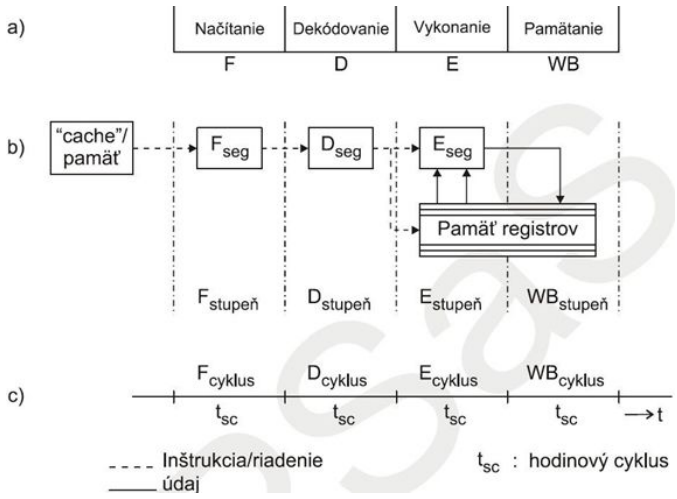
Vo vykonavacej casti - Na urovni operacii

Paralelizmus v skalarnych procesoroch

Predpoklad - Inštrukčný cyklus je rozlozeny na dielcie fazy, ktore sa mozu konkurentne (prekryvane) vykonavat s inými fazami predchadzajucich instrukcii.

Kazda faza sa vykonava v priebehu jedneho alebo viacerych strojovych cyklov.

- System zreťazenia
- Stupne systemu zreťazenia
- Stupeň F - Nacitava instrukcie z pamate
- Stupeň D - Dekoduje instrukcie
- Stupeň E - Vykonáva operaciu definovanú v IF KO
- Stupen WB – Write Back - Zapisuje vysledok operacie do registra/pamate

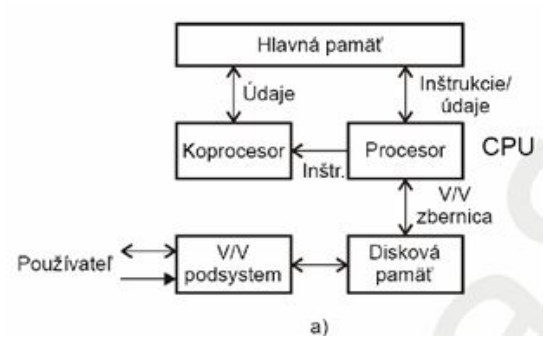


- **Skalárne procesory**

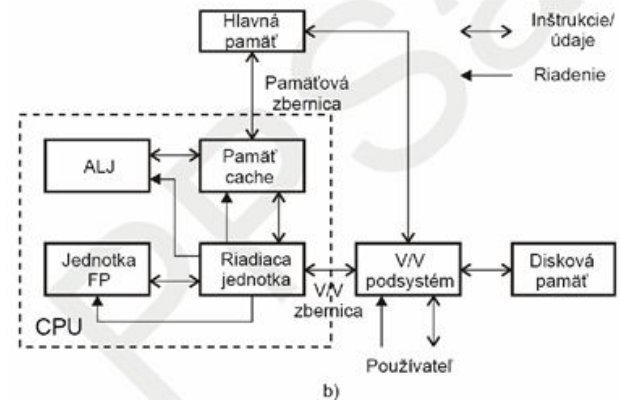
Architektonicke riesenie - **CISC** a **RICS** a zasobnikovo orientovanie architektury

RISC • Prudova Funkcna Jednotka je integrovana
na cipe CISC • Koprocesor

Štruktúrna organizácia komponentov skalárnych procesorov CISC



Typicka strukturna organizacia komponentov skalarnych procesorov CISC



- **PicoJava**

Procesory JavaChips - picoJava I microJava I UltraJava

Programovaci jazyk Java - Virtualny procesor JVM picoJava I

- 4 stupnova PFJ
- F - D - E - WB picoJava II – microJava 701
- Optimalizovane HW prostredie pre JVM
- HW implementacia niektorych JVM instrukcii
- Podpora inych jazykov vyssej urovne
- Instrukčná sada – 300 instrukcii
- 6 stupnova PFJ
- Logika prekrývania styroch instrukcii
- 0-16 KB I-cache pamat
- 0-16 KB D-cache
- 2x32 bitová zbernica medzi D cache a zasobnikovou pamäťou
- Komponenty
    - I-cache
    - D-cache
    - Logika prekryvania
    - Jednotka celociselnych operacii a PHRC operacii
- Prudova funkcna jednotka
    - 6 stupnova F D R E C WB
    - Fetch – citanie instrukcii z I cache alebo z pamate
    - Decode – prekrývanie inštrukcie a ich dekodovanie
    - Register – nacitanie operandov zo zasobnika
    - Execute – vykonavanie instrukcii
    - Cache – citanie/zapis výsledkov do/z zasobnika
    - WriteBack

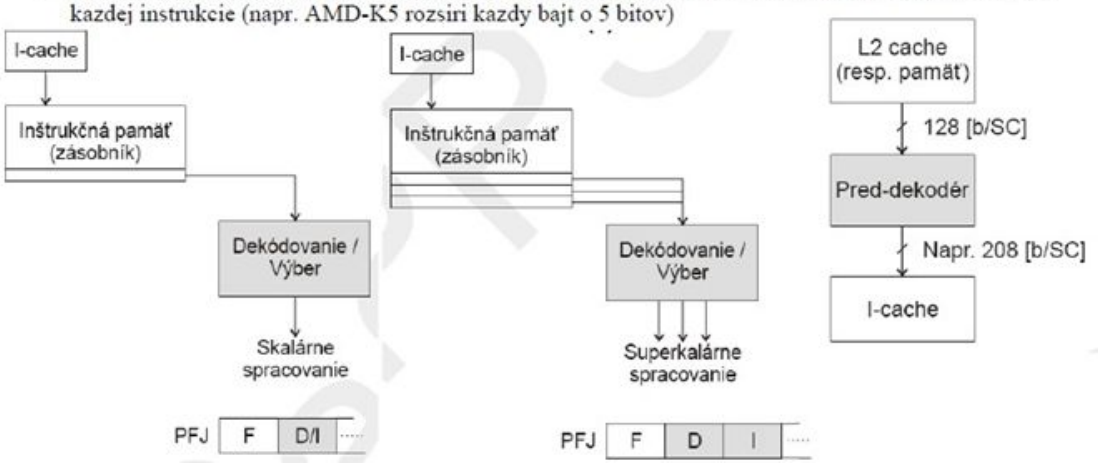
13. T7: Aspekty superskalárneho spracovania prúdu inštrukcií (5)

- **Aspekty superskalárneho spracovania prúdu inštrukcií (vymenovať)**

- **- Paralelné dekodovanie**
- **- superskalárny výber**
- **- paralelné vykonanie**
- **- konzistencia vykonávania**
- **- konzistencia spracovania výnimiek**

- **Paralelné dekódovanie**

- Viac PFJ
- Kontrola vyskytu hazardov
- Porovnanie skalarneho a superskalarneho spracovania prudu instrukcii (Obr. 7.12)
- Pred-dekodovanie (Obr 7.13) – po nacitani instrukcii do I-cache, procesor pripoji priznakove bity ku kazdej instrukcie (napr. AMD-K5 rozsiri kazdy bajt o 5 bitov)



- **Superskalárny výber**

- Aspekty superskalárneho výberu inštruckií
  - Riadenie fázy výberu
  - Počet inštrukcií
- Politika riadenia výberu inštruckií
  - Eliminácia riadiacich hazardov
    - NOP cykly
    - špekulatívne vetvenie
  - Eliminácia WAR a WAW hazardov
    - neeliminujú sa
    - premenovávanie registrov
  - Eliminácia blokovania výberu
    - Priama
    - nepriama
  - Spôsob plánovania výberu inštrukcie
- Plánovanie výberu inštrukcií
  - poradie výberu inštruckií
    - v zhodnom poradí
    - v nezhodnom poradí
  - prehľadávacie okno
    - Fixované
    - pohybové
- Poradie výberu inštrukcií
  - zhodné
  - nezhodné

14. T7: Aspekty superskalárneho spracovania prúdu inštrukcií (5)

- **Paralelné vykonanie**

- Viac PFJ a viac VJ
- Spracovanie inštrukcií v zhodnom/nezhodnom poradí:

    - Usporiadaná kompletizácia
    - Preusporiadaná kompletizácia
- **Konzistencia superskalárneho vykonávania inštrukcií:**
- Procesorová konzistencia
    - poradie kompletizácie inštrukcií
    - slabá procesorová konzistencia
        - preusporiadané poradie inštrukcií
        - preusporiadaná kompletizácia inštrukcií (len bezkonfliktný prúd)
        - detekovanie údajových hazardov
        - napr.: Power1, Powerr2, R8000 ◦ silná procesorová konzistencia
    - usporiadané poradie (v zhodnom poradí)
    - usporiadaná kompletizácia inštrukcií
    - použitie ROB (ReOrdering Buffer) alebo DRIS (Deffered scheduling, Register renaming, Instruction Shelving)
    - napr.: Pentium Pro, AMD 29000, R10000

Pamäťová konzistencia

- prístup do pamäte
- **slabá**
    - preusporiadané poradie inštrukcií
    - preusporiadaný prístup do pamäte (len bezkonfliktný prúd inštrukcií)
    - detekovanie údajových hazardov
    - napr.: PPC 602-620, UltraSparc, R10000
- **silná**
    - usporiadané poradie (v zhodnom poradí)
    - usporiadaný prístup do pamäte
    - použitie ROB ; - napr.: PPC 601
- **Konzistencia spracovania výnimiek (prerušení)**
- Slabá konzistencia
    - Nedefinované poradie spracovania výnimiek (Power1, Alpha procesory)
- Silná konzistencia
- Definované poradie spracovania výnimiek (Pentium, R10000)

15. T7: Superskalárny výber (5)

- **Príklad na výber a kompletizáciu v zhodnom poradí pre definovaný prúd inštrukcií (praktická úloha)**
- **Príklad na výber v nezhodnom poradí a preusporiadanú kompletizáciu pre definovaný prúd inštrukcií (praktická úloha)**

16. T8: TLP architektúry (4)

- **Úvod do TLP architektúr**

ILP architektúry

- Paralelizmus na úrovni inštrukcií
- Jemnozrnný paralelizmus TLP architektúry
- Paralelizmus na úrovni vlákien a procesov
- Stredno a hrubozrnný paralelizmus

Využitie hrubozrnného paralelizmu

- Multiprocesorové čipy o Symetrické multiprocesory – SMS (model UMA)
    - Distribuované multiprocesory so zdieľanou pamäťou – DSM (model NUMA) o Multiprocesory s

odovzdávaním správ – MPP
- Multivláknové procesory
- **Výpočtové modely TLP architektúr**

Modely multivláknových architektúr

- Neumann model – CF model
- Model toku dát – DF model
- Hybridný CF/DF model: Paralelné riadiace operátory, Paralelné riadiace tokeny

Príklady: Tera, P-RISC, EM-4

Sériové riadenie (CF model)

- Princíp programového riadenia
- Interpretácia sériového prúdu inštrukcií
- Programové počítadlo
- Inštrukcie sú uložené v inštrukčnej pamäti
- Údaje sú uložené v pamäti alebo v registroch

Riadenie tokom dát (DF model)

- Princíp toku dát
- Poradie inštrukcií v programe nemá vplyv na poradie vykonávania týchto inštrukcií
- Inštrukcia sa vykoná vtedy ak sú dostupné všetky informácie (všetky operandy) na jej vykonanie.
- Model toku dát nedisponuje spoločnými prepisovateľnými pamäťovými elementmi
- Graf toku dát (Data Flow Graph; DFG)
- Zoskupovanie uzlov DFG do takých podgrafov, ktoré vykazujú nízky stupeň paralelizmu.
- Vopred definovaný podgraf uzlov je transformovaný do reťazca, kt. sa vykonáva sekvenčne.

Hybridný CF/DF model

Hybridný Neumann/dataflow model predstavuje prienik klasického prístupu spracovania inštrukcií a prístupu riadenia výpočtu na základe toku dát.

Hybridný model založený na **riadiacich operátoroch** (RO) má bližšie ku CF modelu a hybridný model založený na **riadiacich tokenoch** (RT) vykazuje črty DF modelu.

Hybridný CF/DF model: RO

Paralelné riadiace operátory podobne ako riadiace inštrukcie v CF modeli majú za úlohu riadiť sled vykonávania inštrukcií.

Rozdiel medzi riadiacimi inštrukciami CF modelu a paralelnými riadiacimi operátormi spočíva v tom, že kým riadiace inštrukcie určujú len sled vykonávania inštrukcií, riadiace operátory sú schopné distribuovať sériové vykonávania inštrukcií do viacerých vlákien. Existencia špeciálnych riadiacich inštrukcií

- **FORK** - Slúži na vytvorenie nového vlákna a na špecifikovanie prvej inštrukcie určenej na vykonanie v tomto vlákne.
- **JOIN**. - Má za úlohu znížiť počet vlákien, čo docieli tým, že špecifikuje v programe bod, kde sa dve vlákna stretnú; z nich pokračuje vo vykonávaní len jedno a druhé vlákno sa zruší.

V rámci vlákien, vykonávanie inštrukcií je založené na CF modeli.

Hybridný CF/DF model : RT

V paralelných výpočtových modeloch založených na riadiacich tokenoch, spracovanie prúdu údajov je zabezpečené na základe interpretácie grafu údajových závislostí (Data Dependency Graph; DDG). Na rozdiel od DFG použitého v DF modeli na hranách tohto grafu sa pohybujú nie dátové ale riadiace tokeny.

Postupnosť vykonávania inštrukcií je jednoznačne definované DDG.Je zrejmé že existuje silná analógia medzi dataflow modelom a modelom založenom na paralelných riadiacich tokenoch, aj keď v prípade druhého modelu riadenie prúdu inštrukcií prebieha nezávisle od prúdu dát.

Inštrukčný rámec CF modelu je rozšírený o ďalšie dva rámce

- Rámec riadiacich tokenov - Uchováva riadiace tokeny potrebné na vykonanie inštrukcie.
- Rámec na uchovávanie adries operátorov - Slúži na určenie adresy pamäťového miesta kam bude treba poslať nový riadiaci token po vykonaní danej inštrukcie.

Tieto dva rámce nahrádzajú programové počítadlo CF modelu.

- **Multivláknové architektúry**

Multivláknové architektúry sú paralelné architektúry, kde sa inštrukcie vykonávajú vo vláknach skalárne, superskalárne, resp. podľa princípu VLIW architektúr.

Motiváciou pre vznik viacvláknových počítačových architektúr bola hlavne príčina dlhej latencie v pamäti, kde rýchlostný rozdiel medzi pamäťovým systémom a procesorom sa postupne zvyšoval.

Viacvláknový prístup ponúkol stratégiu architektúry pre toleranciu odozvy v multiprocesorových systémoch.

Modely multivláknových architektúr (HW aspekt)

- Po cykloch prekrývaný model (skalárna a superskalárna verzia)
- Blokovo prekrývaný model (skalárna a superskalárna verzia)
- Simultánny multivláknový model (superskalárna verzia)
- Nanovláknový a mikrovláknový model (blokovo prekrývaná superskalárna a simultánna superskalárna verzia)
- Model VLIW
- Model multiprocesorového čipu **Multivláknové architektúry**
- Supeskalárne procesory s cyklickým prekrývaním
- VLIW procesory s cyklickým prekrývaním
- Multivláknové procesory so súbežným vykonávaním multivlákien
- Jednočipové multiprocesory so štyrmi multivláknovými procesormi

17. T9: Programovanie TLP architektúr - Pthreads (3)

- **Definícia procesu a vlákna**

Proces:

- je bežiaca inštancia programu vrátane všetkých hodnôt premenných a stavu. (je úloha, ktorú vykonáva počítač, často súčasne s inými.)
- *Multitasking operačného systému prepína medzi procesmi, čím vzniká dojem súčasného behu viacerých procesov*

Vlákno

- je jednotný prúd (sekvencia) existujúca v rámci procesu.
- Vlákna majú niektoré spoločné vlastnosti ako majú procesy
- vlákna sú využívané na zrýchlenie vykonávania aplikácie pomocou paralelizmu Objekty a funkcie knižnice Pthreads API

- **Objekty a funkcie knižnice Pthreads API**

  Objekty

  pthread_t - uchovava informacie identifikujuce vlakno s ktorym je asociovany

  Funkcie

  int pthread_create(pthread_t* thread, pthread_attr_t* attr, void* (*start_routine)(void*), void * arg);

  vytvorenie vlakna: parametre: 1. moze byt NULL, 2. alokujeme pred tymto volanim, 3. funkcia ktoru bude vlakno vykonavat, 4.smernik na argumenty pre fciu *start_routine*

  int pthread_join(pthread_t thread, void ** thread_return);

  čakanie na ukoncenie vlakna, volame pre kazde vlakno

  ukoncenie vlakna: int pthread_exit(void * retval);

  oddelenie vlakna: int pthread_detach(pthread_t thread);

  inicializacia attributov: pthread_attr_init(pthread_attr_t *attr);

  uvolnenie attr.: pthread_attr_destroy(pthread_attr_t attr);

  Atribúty

  pthread_attr_t attr a pthread_attr_init

- **Synchronizácia vlákien: busy-waiting, mutex**

  Busy-waiting & mutex:

  - Synchronizácia s aktívnym čakaním - prostriedky:
    - zákaz prerušení
    - inštrukcie Test a Set Lock
    - použitie premenných programu
    - Compare-and-swap mutex
  - Synchronizácia vlákien, ochrana premenných v zdieľanej pamäti
  - Zámok na ochranu zdieľaných zdrojov
  - Iba jedno vlákno môže vlastniť (uzamknúť) mutex v danom čase
  - Ak sa viaceré vlákna pokúsia obdržať vlastníctvo, iba jeden ho získa. Až keď ho vlákno uvoľní, môže ho získať iné.

- **„Producer-konzumer" synchronizácia**

  The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened. The problem can also be generalized to have multiple producers and consumers.

18. T9: Programovanie TLP architektúr - Pthreads (5)

o Barrier a condition variables

- Bod stretnutia
- Kritický bod
- Žiadne vlákno neprejde cez kritický bod skôr ako všetky vlákna nedosiahnu bod stretnutia

Barrier - čakanie kým sa všetky vlákna nedostanú po bariéru - bod v programe, implementácia pomocou:

- busy-waiting a mutex: zdieľané počítadlo kontrolovane mutexom
- semafor
- condition variable

Condition variables

- blokovanie vlákna po výskyt udalosti/stavu
- posielanie signálov
- vždy spojená s mutexom

o Read-write lock

Read-write lock sa používa keď čítanie dátovej štruktúry viacerými vláknami naraz je bezpečné, ale keď jedno z vlákien potrebuje modifikovať alebo zapisovať do dátovej štruktúry, vtedy iba to jedno vlákno môže pristupovať k údajom.

o Caches, Cache-Coherence, and False Sharing

Dizajnéri čipov pridali do procesorov bloky relatívne rýchlej pamäti – tie sa volajú cache pamäť.

o Thread-safety

Blok kódu sa nazýva thread-safe keď môže byť vykonávaný súčasne viacerými vláknami bez toho aby vznikli problémy.

19. T10: Programovanie TLP architektúr - OpenMP (5)

o Charakteristika OpenMP API : pragmy

Špeciálne **preprocesorove inštrukcie**

Pridané na povolenie takého správania systému, ktoré nie sú súčasťou základnej C-čkovej špecifikácie.

Taký prekladač, ktorý nepodporuje pragmy, jednoducho ich ignoruje.

#pragma omp parallel shared(A,B) private (i, j)

- shared - premenné A, B sú zdieľané medzi vláknami, všetky vlákna majú prístup, jedna inštancia
- private – každé vlákno má vlastné premenné i, j, každé vlákno má vlastnú kópiu, zmeny sú zachované len v danom vlákne

#pragma omp critical

- je vykonávaná v danej chvíli práve jedným vláknom

#pragma omp barrier

- predstavuje virtuálnu stenu ktorá zabraňuje vykonávanie programu, pokiaľ všetky vlákna nedokončili svoju úlohu

#pragma omp for

o Redukčné operátory

Redukčný operátor je binárny operátor (ako sčítanie a násobenie)

Redukcia je výpočet, ktorý opakovane používa rovnaký operátor na postupnosť operandov aby získal jeden výsledok.

Všetky medzivýsledky sa ukladajú do jednej premennej, ta sa vola redukčná premenná.

o Klauzula „schedule"

#pragma omp parallel for schedule(type [,chunk size]) typ môže byť:

- **static** (iterácie mozu byt pridelene vlaknam predtym ako sa vykoná cyklus)
- **dynamic**/guided (iterácie su pridelene vlaknam pocas vykonávania cyklu)
- **auto** (compiler alebo system urci schedule)
- **runtime** (schedule je urceny pri spusteni) chunksize je kladny integer

o Klauzula „critical"

Kritická sekcia

Zaisťuje seriove spracovanie blokov kódu.

Môžu byť rozšírené o sériové zoskupenie blokov s použitím pomenovania "name".

Pomalšie!

20. T11: GPGPU (zevraj nebude)

o CUDA : charakteristika, základná konštrukcia programu, synchronizácia

- **OpenCL : charakteristika, základná konštrukcia programu, synchronizácia**

21. T12: Architektúry MIMD, paradigmy MIMD a distribuované systémy (2)

- **Typy MIMD architektúr**

Multiprocesory

- Spojenie viacerých nezávislých procesných elementov (PE)
- Každý PE má vlastnú kópiu programu a môže pracovať s odlišným dátovým prúdom
- Implementácia prístupu do pamäti: Zdieľaná pamäť, Distribuovaná pamäť

Multiprocesory so zdieľanou pamäťou: UMA, NUMA, COMA

Multiprocesory s distribuovanou pamäťou - Multipočítače

- **Paradigmy MIMD -**

Kombinácia rôznych prístupov na báze MIMD

- Špecializované architektúry
- Hybridné architektúry o MSIMD
- multivektorové PPS,
- Multiple Instruction Stream Associative MASC procesor (r.2006) o SPMD ~ MIMD

o MIMD/SIMD

- Počítače DataFlow
- Redukčné počítače

22. T12: Architektúry MIMD, paradigmy MIMD a distribuované systémy (2)

- **Počítače riadené tokom dát**

Paralelná organizácia výpočtu typu data-driven.

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 alokácií výpočtových prvkov (PE) ku všetkým inštrukciám s dostupnými argumentmi.

Strojovou reprezentáciou programu DF je dataflow graf (DFG).

Výpočtový model DF (resp. program DF) podporuje paralelizmus na úrovni inštrukcií.

Úroveň paralelizmu v programoch závisí od granularity organizovania výpočtového procesu.

Charakteristické princípy výpočtového modelu DF • Asynchrónnosť: Operácie sa vykonajú vtedy, keď sú dostupné vyžadované operandy.

• Funkcionalita: Operácie sú funkcie, t. j. nezávisia od operandov iných funkcií (neexistujú vedľajšie účinky), v dôsledku čoho sa môžu vykonávať paralelne.

Klasifikácia DF architektúr **Statická DF architektúra**

Dynamická DF architektúra

- Čistá (pure) DF architektúra
- DF architektúra s explicitným ukladaním dátových tokenov (explicit DT store)
- DF architektúra s priamym spájaním operandov (direct matching) **Hybridná DF architektúra**
- Skorá hybridná DF architektúra
- DF architektúra s vláknovým prístupom
- Hrubozrnná DF architektúra
- RISC-ovská DF architektúra

Graf toku dát (Data Flow Graph; DFG)

- Orientovaný graf
- Uzly reprezentujú inštrukciu (resp. operátor)
- Orientované hrany indikujú tok výsledných dát vo forme aktivačných značiek (tokenov), ktoré môžu byť dátové(DT) alebo riadiace(CT).

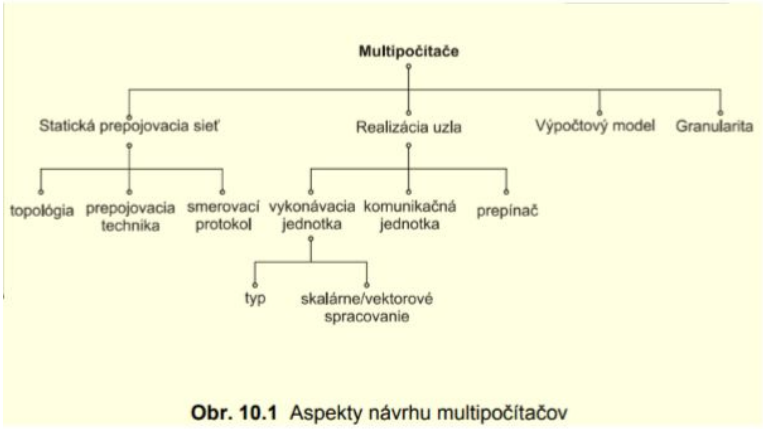
Vlastnosti DFG

- Funkcionalita: Operácie v DFG sú funkcie
- Asynchrónnosť: Uzly sa aktivujú v okamihu, keď na ich vstupných hranách sú prítomné aktivačné značky
- Komponovateľnosť: DFG je možné spájať • Vykonanie DFG:
    - Pravidlo vykonateľnosti (PV) - Inštrukcia je vykonateľná, ak všetky jej operandy sú prístupné.
    - Pravidlo aktivácie (PA) - Inštrukcia je aktivovaná, keď je vykonateľná a zdroje na jej aktiváciu sú k dispozícii

o Distribuované systémy -

Sieťové prostredie vzájomne prepojených počítačových uzlov, umožňujúcich prenos informácií medzi nimi a zdieľanie ich HW a SW prostriedkov.

Počítačová sieť: LAN, MAN, WAN



Obr. 10.1 Aspekty návrhu multipočítačov

23. T12: FPGA (2)

o FPGA

FPGA umožňuje **vytvoriť** ľubovoľný **digitálny obvod**. Narozdiel od **mikrokontroléra**, **nedokáže** vykonať žiadnu **funkciu**, ktorú definuje **programovateľná logika**.

Základná štruktúra:

- **logické bloky**:
  - CLB - **C**onfigurable **L**ogic **B**lock
  - LAB - **L**ogic **A**rray **B**lock
- **switch boxes**
- **input output block**
- **prídavné bloky**:
  - embedded processors
  - DSP bloky
  - distribuovaná pamäť
  - clock mangement

HDL - kód sa **nevykonáva**, ale je z neho **odvodená** konfigurácia **FPGA**

Výhody FPGA:

- **flexibilnejšia ako dedikované čipy** - rýchle prototypovanie, simulácia, IP cores
- **rýchlejšie ako mikrokontroléry**
- **preprogramovateľné** - vo všetkých fázach
- **vysoká hustota integrácie** - nízka cena, malý rozmer
- priaznivý **pomer počtu** logických **obvodov** za **cenu**
- **rôzne I/O sú prepojiteľné** cez napr. VGA -> HDMI

o Syntéza logických obvodov

Krok 1 - Syntéza:

Analyzuje sa HDL opis a **hľadá** zodpovedajúce hardvérové **štruktúry** v podobe **kombinačných** a **sekvenčných** logických obvodov.

Krok 2 - Mapovanie na cieľovú technológiu:

Podľa vybraného typu **FPGA** sa „**mapper**" snaží nájsť najlepšie **ekvivalenty** v syntéze nájdených komponentov. Zisťuje sa, či je návrh **realizovateľný** na zvolenom **FPGA**.

Krok 3 - Rozmiestnenie a prepojenie:

Hľadá sa vhodné **usporiadanie** a **prepojenie** komponentov FPGA v **prepojovacej matici**. Rieši sa **časovanie** čo určí či návrh bude **fungovať** na zvolenej **frekvencii**.