1. Je výrok pravdivý?
Pri načítaní operandov vo fáze Výberu (angl. Issue) tak pamäť registrov ako aj P-zásobník obsahuje informáciu o platnosti prislušného zdrojového registra.
Vyber jednu:
Pravda
Nepravda

2. Definujte a opíšte:
1. Definujte granularitu
2. Čo je ROB a na čo slúži?
Granularita špecifikuje rozmer údajových položiek a programových modulov výpočtu. V tomto zmysle sa algoritmy delia na

- jemnozrné
- stredozrné
- hrubozrné

ROB je ReOredering Buffer, ktorý slúži na preusporiadanie kompletizácie inštrukcii, slúži na udržanie procesorovej konzistencie pri superskárnom vykonaní ištrukcii.

3. Je tento výrok pravdivý?
Ak zvýšime taktovaciu frekvenciu CPU na dvojnásobok, CPU bude generovať 2-krát viac tepla.
Vyber jednu:
Pravda
Nepravda

4. Majme 3 vlákna, vlánko s identifikátorom ID0, vlákno s idnetifikátorom ID1, vlákno s identifikátorom ID2. Potom pri vykonaní nasledujúceho regmentu kódu:

```
sum = 0.0
#pragma omp parallel for num_threads(3) reducation(+;sum)schedule(static,1)
for(i=0;i<12;i++)
    sum += f(i)
```

vlákno s ID0 spracuváva funkcie? Označte správnu odpoveď:
f(0), f(1), f(2), f(3)
f(0), f(3), f(6), f(9)
f(1), f(4), f(7), f(10)
f(2), f(3), f(8), f(9)

5. Ktoré výroky sú pravdivé o DF architektúrach? Označte jednu alebo viac odpovedí.
Model toku dát nedisponuje spoločnými prepisovateľnými pamäťovými elementmi.
Poradie inštrukcii v programe nemá vplyv na poradie vykonania týchto inštruckcií.
Model toku dát nedisponuje spoločnými prepisovateľnými pamôťovými elementami.
Poradie inštrukcii v program má vplyv na poradie vykonania týchto inštruckii.

6. Ktorý výrok je pravdivý ohľadom SISD architekúr? Označte jednú správnu odpoveď:
Má 1 riadiacu jednotku a M výkonových jednotiek
Má N riadiacu jednot a M výkonových jednotiek
Má N riadiacich jednotiek a 1 výkonovú jednotku
Má 1 riadiacu jednotku a 2 výkonovu jednotku

7. Je výrok pravdivý?
Speedup(p processors) = Time(1 processor) / Time (p processors)
vyber jednu
pravda
nepravda

8. Je výrok pravdivý?
Speedup(p processors) = Time (p processors) / Time(1 processor)
vyber jednu
pravda
nepravda

8. Prepojovacia funkcia prepojovacej siete N-dimenzionálna kocka je definovaná zápisom
$f(p_{n-1}, \dots, p_1, p_0) = p_0, p_{n-1}, \dots, p_2, p_1$
vyberte jednu
pravda
nepravda

9. Ktoré výroky sú pravdivé. Označte jednu alebo viac odpovedí:
Po cykloch prekrývaný VLIW Model je možné uplatniť v jednovlaknových TLP procesoroch.
Po cykloch prekrývaný superskalárny model je možné uplatniť v jednovláknových TLP procesroch.
Simultánny superskalárny model je možné uplatniť v multivláknových ILP procesoroch.
Simultánny superskalárny model je možné uplatniť v jednovláknových ILP procesoroch.

10. V súvislosti s počítačmi riadeným tokom dát je alebo nie je nasledujúci výtok pravdivý?
Inštrukcia je vykonateľná, ak všetky jej operandy sú prístupné.
vyberte jednu
pravda
nepravda

11. Koeficient účinnosti vektorizácie pre nasledujúcu skladu programu:
Nevektoritovateľná časť programu: 5 skalárnych operácii
Doba vykonania jednej skalárnej operácie: 6 strojových cyklov (SC)
Vektorizovateľná časť programu: 1 operácia 10-násobného logického cyklu
Doba vykonania tela cyklu: 25 SC
Účinnosť vektorového spracovania 3.06
sa rovná: Označte jednú odpoveď:
3.23
2.50
3.11
2.62

12. Prepojovacia funkcia prepojovacej siete N-dimenzionálna kocka je zápisom:
$f_1(p_{n-1} \dots p_1 p_0) = p_{n-1} \dots p_{i+1} p_i p_{i-1} \dots p_0$
$f_1(p_{n-1} \dots p_1 p_0) = p_{n-2} p_{n-3} \dots p_{i+1} p_i p_{i-1} \dots p_0 p_{n-1}$
$f_0(p_{n-1} \dots p_1 p_0) = p_{n-1} \dots p_{i+1} p_i p_{i-1} \dots p_0 p_{n-1} \dots p_0$
$f(p_{n-1} \dots p_1 p_0) = p_0 p_{n-1} \dots p_{i+1} p_i p_{i-1} \dots p_2 p_1$

13. Je výrok pravdivý:
Medzi nárastom počtu tranzistorov tvoriacih CPU a nárastom taktovacej frekvencie je nepriama úmera.
Pravda
Nepravda

14. Ktorý výrok je pravdivý ohľadom MP Programovacieho modelu? Označte jednu odpoveď:
Komunikácia medzi procesmu je zabezpečená výlučne pomocou spoločného pamäťového priestoru.

Riadenie je založené na interpretácii toku dát.
Procesy komunikujú prostredníctvom sprav.
Tvorí základ pre PC riadené tokom dát.

15. Ktorý výrok je pravdivý ohľadom SAS programovacieho modelu? Označte jednu odpoveď:
Riadenie je založené na interpretácii toku dát.
Komunikácia medzi procesmi (vláknami) je zabezpečená pomocou spoločného pamäťového priestoru.
Procesy komunikujú prostredníctvom správ.
Tvorí základ pre počítače riadené tokom dát.

15. Ktorý výrok je pravdivý ohľadom SAS programovacieho modelu? Označte jednu odpoveď:
Tvorí základ pre počítače riadené požiadavkami
V tomto modeli neexistujú synchronizačné metódy
Procesy komunikuju medzi sebou pomocou spoločného pamäťového priestoru
Radenie je zalo%zené na interpretácii toku dát

16. Ktoré výtoky sú pravdivé? Označte jednu alebo viac odpovedí:
Paralelné spracovanie “dát” v ILP procesoroch sa uskupočňuje aplikovaním princípu zvyšovanie počtu funkčných jednotiek.
Paralelné spracovanie “dát” v ILP procesoroch sa uskupočňuje aplikovaním princípu prudového spracovania.
Paralelné spracovanie “dát” v ILP procesoroch sa uskupočňuje prostredníctvom nanovlákien.
Paralelné spracovanie “dát” v ILP procesoroch sa uskupočňuje prostredníctvom vytokových slotov.

17. Medzi inštrukciami i1 a i2 je závislosť typu?

#	Inštrukcia	Význam
i1	load RG1, M[A]	RG1:=M[A]
i2	store M[B], RG1	M[B]:=RG1

Označte jednu odpoveď:
WAR
RAW
riadiaca závislosť
WAW

18. V súvislosti s počítačmi riadenými tokom dát je alebo nieje nasledujúci výrok pravdivý?
Inštrukcia je vykonateľná, ak sú všetky operandy prístupné:
Vyberte jednu:
Pravda
Nepravda

19. Je tento výrok pravdivý?
Pri načítaní operandov vo fáze Výberu (angl. Issue) tak pamäť registrov ako aj P-zásobníj obsahuje informáciu o platnosti obsahu príslušného zdrojového registra.
Vyber jednu:
Pravda
Nepravda

20. Je tento výrok pravdivý?

Ak zvýšime taktovaciu frekvenciu CPU na dvojnásobok, CPU bude generovať cca 3-krát viac tepla.
Vyber jednu:
Pravda
Nepravda

21. Údajová závislosť RAW znamená Označte jednu odpoveď:
Čítanie premennej po jej zápise
Zápis premennej po jej čítaní
Čítanie a zápis sú V/V inštrukcie

22. Ktoré výtoky sú pravdivé? Označte jednu alebo viac odpovedí:
Po cykloch prekrývaný superskalárny model je možné uplatniť v jednovláknových TLP procesoroch.
Simultánny supersklárnych model je možné uplatniť v multivláknových ILP procesoroch.
Simultánny supersklárnych model je možné uplatniť v jednovláknových TLP procesoroch.
Po cykloch prekrývaný VLIW model je možné uplatniť v jednovláknových TLP procesoroch.

23. Je výrok pravdivý?
Počet tranzistorov tvoriacich CPU medziročne rastie 6-násobne.
Vyberte jednu
Pravda
Nepravda

24. Ktorý výrok pravdivý ohľadom MP programovacieho modelu? Označte jednu odpoveď:
Riadenie je založené na interpretácii toku dát.
Tvorí základ pre počítače riadené tokom dát.
Procesy komunikujú prostredníctvom správ.
Komunikácia medzi procesmi je zabezpečená výlučne pomocou spoločného pamäťového priestoru.

25. Definujte a opíšte
1. Definujte komunikačnú latentnosť
2. Čo je DRIS a na čo slúži?
Komunikačná latentnosť – časová miera komunikáčnej réžie medzi jednotlivými procesmi, teda komponentami PC systému počas paralelného spracovani úloh. Ovplyvňuje zložitosť paralelného systému a paralelné spracovanie.
DRIS – Deffered scheduling, Register renaming. Instruction Shelving. Je kombinovaný zásobník.

26. Granularita je definovaná ako:
Označte jednu odpoveď:
komunikačná latentnosť
miera množstva výpočtov v programových segmentoch
časova miera v programových segementoch
paralelizmus výpočtového procesu

Čo je to paralelná architektúra?
Paralelný počítač je zbierka procesných elementov, ktoré spolupracujú na vyriešení nejakého veľkého problému v čo najkratšom čase.
Niektore príklady:
- alokácia zdrojov
  - ako veľké sú zdroje?

    - ako výkonné sú vykonávacie elementy?
    - koľko pamäte zaberajú?
- prístup k zdrojom, komunikácia a sychronizácia
    - ako jednotlivé elemeny pc achitektúry navzájom spolupracujú a komunikujú?
    - ako sú dáta prenášané medzi procesormi?
    - Čo predstavuje abstrakcia a primitíva komunikácie?
- výkon a škálovateľnosť
    - ako je toto všetko preložiteľné do vykonateľného programu?
    - Ako to je škálovateľné?

Úlohou PC architekta (úloha PC architekta, uloha pc architekra) je navrhnúť a zostrojiť rôzne verzie PC systémov a maximalizovať ich výkon, zabezbečiť schopnosť programovať na týchto systémoch bez obmedzenia limitmi a zabezpečiť čo najnižšiu cenu.

Paralelizmus

- poskytuje alternatívu k rýchlemu vykonaniu výpočtov
- dá sa aplikovať na všetký levely systemového dizajnu
- je fascinujúcou perspektívou, ktorá ponúka pohľad na PC architektúru
- zabezpečuje vyššiu rýchlosť spracovania informácii

Cieľom aplikácii, ktoré používajú paralelné zariadenia je dostiahnuť čo najväčšie zrýchlenie (speedup). To je vyjadrené pomocou vzorca:

$$speedup\ (p\ processors) = Perfomence\ (p\ processors) / Performence\ (1\ processor)$$

Pre problém fixnej veľkosti (fixna veľkosť datasetu) je výkon = 1 / čas

$$speedup_{fixed\ problem}(p\ processors) = Time\ (1\ processor) / Time\ (p\ processors)$$

Paralelizmus

- zlepšuje technológiu semikonduktorov
    - menšia veľkosť, výššia rýchlosť (clock speed)ň
- zlepšuje PC architektúru
    - zabudované HLL kompaileri
    - RISC architektúry
- je zabudovaný do
    - „ľahkých pc“
    - programovacích jazykov závislýchj na manažovaní

Veľké paralelné zariadaenia / paralelné počítače sa stali najviac používané vo viacerých odvetviach priemyslu

- v oblasti pohonych hmôt (analýza)
- v automobilovom priemysle (simulácie havárii, analýza jazdy, slokúmanie efektivity spaľovania)
- v leteckom priemysle (analýza letu, analýza výkonosti motorov, štrukturálna mechanika, elektromagnetizmus)
- v počitačovo závislom dizajne
- v farmaceutickom priemysle (molekulárne modelovanie)
- vo vizualizácii
    - zábavný a fimový priemysel
    - architektúra
- vo finančníctve (modelovanie)

Gartnerov hype cyklus / Gartner hype cycle

Gratnerov hype cyklus poskytuje grafickú reprezentáciu vyspelosti a schopnosti adaptácie v technológiach a aplikáciaich a snaží sa odhadnúť ako sú potencionálne vhodné na riešenie skutočných biznis provlémov a ako využívajú nové možnosti.

Spúšťač technológie (technology trigger)

- potencionálna technologia / novinka v technologickej oblasti, ktorá dá vecei do pohybu
- skorý kocept príbehov a medialny záujem o tieto novinky spôsobuje vyšší zaujem verejnosti

- často ide o produkt, ktorý namá žiadnu inú komerčnú verziu, preto sa teší veľkej obľube

Vrchol pumpovania očakávaní (peak of inflated expectations)

- skorá publicita produktu, vysoký počet úspechov v počiatku produktu môže viesť k neskorším chybám
- dramatické prestrelenie reality tak vysoko ako to len je možné
- niektoré spoločnosti vzbudia zaújem, ale neskôr sa príde na to, že ich produkty neboli až také inovatívne

Korýto dezilúzie (trought of disillusionment)

- zaujem o produkt klesá s tím ako prichádzajú chyby v experimentálnom návrhu
- produkt alebo technológia sa otriasa v základoch alebo zliháva
- vývoj a zdokonaľovanie pokračuje iba ak sa vývojári ďalej rozhodnú vylepšovať ich produkt

Sklon osvetenia (slope of enlightement)

- viacero inštancii danej technológie môže byť prínosné na počiatočný vývoj technológie a môže sa ními docieliť širšie porozumenie technológii
- viacero inštitúcii vstupuje do pilotného testovania projektov, niektoré konzvervatívne spoločnosti v tom zostávajú opatrné

Plató produktivity (plateau of productivity)

- najelpšie pre produktu je ako je používaný mainstreamom v počiatočnej fáze
- critéria pre prístup k produktu by mali byť definované čo najjasnejšie

Súhrn k trendom v aplikáciach

- spočiatku len vo vedeckých pc a inžinierskom počítačovom výskume
- neskôr sposobili paralelné PC aj rapídny progres v komerčných PC
  - lepšie vykonávanie v oblasti databáz, transakcii a finančníctva
  - lepšiee vykonávanie v oblasti menej-škálovateľných systémov ale aj veľkých-škálovateľných systémov
- základom je znižovanie buducej veľkosti obvodu (λ) – obvody sa stávaju rýchlejšie alebo spotrebujú menej energie
- stúpa životnosť obvodou
  - clockrate raste približne rovanko ako veľkosť obvod (λ)
  - počet tranzistorov sa zvyšuje podľa vzorca λ^2 (alebo rýchlejšie)
- ako používať viac tranzistorov?
  - Paralelizmus v procesingu – viac operáci v jednom cykle spracovania redukuje záťaž na CPI
  - Lokalizovania prístupu dát - vyhnúť sa latencii a redukovať CPI, zlepšiť vykonávanie procesora

Nárast frekvencie tykov (Clock Frequency Growth Rate)

$P \sim C_{dyn} * V^2 * f$

$f \sim V$

$P \sim C_{dyn} * V^3$

P = výkon

C_{dyn} = dynamická kapacita

V = napätie

f = frekvecia

Lineárny nárast frekvencie je spôsobený rozptýlením výkonu, ktorý rastie kubicky. Ak sa frekvencia zvýší čo i len dvojnásobne, vypordukované teplo bude 8 kráte väčšie, preto musia byť procesory chladené alebo vypnuté.

Tredny v architektúre

Achitektúra procesora sa zohľadní na jeho výkone a spchopnostiach. Porozumenie architekúte mikroprocesorov umožňuje intuitívne budovať ich dizajn a paralelné mechanizmi, V histórii PC rozlyšujeme 4 typy archtektúr

- rúra (tube)

- tranzistory
- IC (Integrated chip)
- VLSI (Very Large Scale Integration)
  - až u VLSI bola načrtnutá myšlienka paralelizmu

Programovací model
- všetko to, čo programátor použije počas programovania aplikácie
- špecifikuje komunikáciu a synchronizáciu
- príklad
  - multiprogramovanie: žiadnna komunikácia alebo synchronizácia na programovom levely
  - zdielaný adresný priestor
  - tok sprav (ako list alebo telefonný hovor, explicitne od bodu k bodu)
  - paralelné dáta – striktnejšie orgranozované, viac su na nich vykonávané globálne akcie, skôr implementované pomocou zdielaného adresného priestoru alebo toku sprav

Abstrakcia komunikácie

Primitíva uživateľského levelu komunikácie
- realizujú programovací model
- mapovanie medzi primitívami jazyka programovacieho modelu a primitívami

Je podporovaná
- priamo HW
- prostredníctvom OS
- prostredníctvom SW

Kritická vrstva abstrakcie leží medzi programovou aplikáciou a HW. V tom zhrávajú rolu kompailery a sw.

Komunikácia architektúry

= uživateľské/systémové rozhranie + implementácia

prejavuje sa na užívateľskom leveli, hw, ale aj systémovom leveli

Implementácia
- orgranizačná štruktúra, kt implementuje primitíva: HW alebo OS
- štruktúra siete

Ovplyvňuje
- výkon
- aplikovateľnosť
- programovateľnosť
- škalovateľnosť
- cenu

Evolúcia archtiktonických modelov / evolúcia modelov atchitektúry

V histórii sa podporovali tieto programátorske modely
- zdielaný adresný priestor
- odovdzávanie sprav
- paralelné dáta
- tok dát
- systolické polia (systolic arrays)

Architektúra zdielaného adresného pristoru (SAS architecture)

Každý procesor dokáže priamo referovať na hocijakú lokáciu v pamäti. Komunikacia sa realizuje implicitne a má za následok načítanie a ukladanie do pamäte. Podobný programovací model je zdielanie času (time-sharing) – očakáva, že procesy bežia na rôznych procesoroch.

Prirodze bola SAS architektúra implementovaná na rôznych platformách. Populárne je známa ako zdielaná pamäť alebo model.

Proces: virtuálny adresny priestor plus jeden alebo viac vlakien na kontrolu.

Komunikácia HW

- komunikácie uniprocesorov
- kapacita pamäte vzrastá s pridaním modulov alebo I/O kontrolerov

Hlavný prístup
- motivovaný multiprogramovaním

Prístup minipočítačov
- skoro každý mikrocporcesorový systém má zberunicu
- motivovaný muliprogramovaním
- ťažko použiteľný pre paralelne programy
- nazývaný aj symetrický multiprocesing
- vyššia latencia ako u uniprocesorov
- zbernica je úzke hrdlo (bottleneck)
- nízka cena inkrementácie

SMP stroje / SMP procesory
- všetky procesory vidia všetko (pamäť, I/O, prerušenia)
- plánovač vypočítava, ktorá aplikácia bude priradená ktorému procesoru
- aplikácia môže migorvať medzi procesormi
- zvyčajne nezdieľajú cache (mulijadrové architektúry naopak zdieľajú L3 alebo L4 cache medzi jadrami)
- problémy
  - SMP systémy môže byť komplexné (náročne) nastaviť a udržiavať, preto lebo sa v nich duplikuje HW ako napr chladenie procesorov (kvôli tomu majú tendenciu byť hlučné)
  - migrovanie procesov medzi procesormi môže viesť k preťaženiu cache pamäte
  - viacero procesorov môže spôsobovať race conditions (potrebujeme zabezpečiť multiprocesorovu synchronizáciu)

Škálovanie
- problém je propojenie (jeho cena a šírka pásma zbernice)
- príklady
  - Cray T3E (obsahuje 2048 škálovateľných procesorov, 480 MB/s)
    - 3D torous prepojenie
    - 2 levely cachingu
  - NUMA
    - každý čip má vlastný zdroj pamäte a nemusí súperiť o zdielanu pamäť

Srdce problémov (Heart of the trouble)

Pamäťová stena
- rozdiel medzi tým ako rýchlo dokáže CPU operovať s dátami a ako rýchlo dokážu dáta prichádzať do CPU
- počer jadier na procesore sa zvyšuje, počet pripojený z čipu na zvyšok PC nie
- neexistuje fyzický vzťah medzi tým čo môže procesor vykonávať a kde je ďalší dataset uložený

Architektúra predávania správ (message passing architecture)
- programovaci model: priamy prístup cez súkromny adresný priestor (lokálnu pamäť), komunikácia cez explicitné správy (posielanie a prijímanie)
- vysokoúrovňový diagram blokovania sprav je podobný prístupu NUMA zdielaniu sprav
  - komunikácia je integrovaná na IO levely, nemusí byť načitávaná na pamäťový systém

Abstrakcia predávania správ (message-passing abstraction)
- správy sú posielané špecifickému bufferu, ktorý prenáša a prijíma správy
- *Recv* špecifiku preposielaci proces a predanie aplikácii
- *Memory to memory* kopíruje úseky pamäte a premenuváva procesy, ak to je potrebné
- v najjednoduchšej forme posielanie a prijímanie dat je zhodné len v prevratenom poradí

- pamäť najviac preťažujú

Vývoj predávania správ (evolution message-passing)

- pôvodné mechanizmi- FIFO pre každý prepoj
  - HW blítko programového modelu – synchroné operácie
  - nahradené DMA, zabudovanými neblokujucími operacíjami – bolokované systémom kym nieje dosiahnutý cieľ
- dôležitosť topológie siete
  - store-and-forward routing → topológia je dôležitá
  - pipeline routing → topológia je menej dôležitá
  - zjednodušuje programovanie
- evolúcia a role softvéru majú „rozmazané hranice“
  - tradičné MP operácie (posielanie a prímanie)
    - podporované bufrom
    - posielanie zahŕňa aj zápis dát do bufru
    - prijatie zahŕňa čítanie dat zo zdieľanej pamäte
    - flagy a zámky (slúžia na kontrolu prístupu k bufru a indukujú udalosti ako je napr. príchod správy)
  - na MP strojoch, užívavateľ môže konštruovať globálny adresný priestor
    - prístup cez trasakcie implicitných sprav
    - väčšina MP knižníc dovoľuje spracovať a akceptovať správy pre hocijaký proces
    - čítanie je realizovaný pomocou procesu posielania a židania sprav procesom
    - aktuálna trasakcia môže byť zchovaná pred užívateľom
  - zdielay virtuálny adresny priestor môže byť ustanovený pomocou mechanizmu prenášania sprav na stránkovom levely (page level)
    - zbierka procesov ma vymedzený region zdielanej pamäte (len jedna strana môže ku nemu pristupovať)

MPI

- štandardizovaný a prenosteľný štandard prenášania sprav
- MPI Je špecifikované pre developerov a používateľov MPI knižnic
- nie je to knižnica
- často používaný pre prenosteľné a škálovateľné aplikácie
- pôvodne navrhnutý pre distribované systémy, masívne paralelné systémy, siete zložené z viacerých pracovných staníc
- dokáže sa prispôsobyť narastajúcemu výkonu na ostatných typoch systémov → zdielaná pamať symetrického multiprocesingu, systémy úzko zoskúpene s SMP uzlamy

Systémy parlelných dát (Data parallel system)

Programovací model

- operácie sú vykonávane na každom elemente datovej štruktúry
- logicke vlákno pre riadenie → vykonáva sekvenčne alebo paralélne kroky
- koncepčne je procesor asociovaný s každým datovým segmentom

Model architektúry

- pole má mnoho jednoduchých, lacných procesorov s malým množstvom pamäte (procesor nepreusporiadáva inštrukcie za ich behu)
- špecifikované a zovšeobecňované jednodchou globálnou synchronizáciou

Pôvodné využitie na

- zjednodušené riešenie diferenciálnych rovnic
- centralizuje vyokú cenu inštrukcie fetch (sekventovanie)

Architektúra roku dát

Reprezentuje postuponosť výpočtu pomocou grafu závislostí

- každý procesor je reprezentovaný ako uzol, aktivovaný pomocou operandov
- správy (tokeny) sa prenášajú medzi uzlami, ak sa vykonanie jednej správy skončí je

preposlaná na ďalší procesor
- tokeny sú porovanné navzájom, tie, ktoré sa zhodujú sú vykonané

Systolická architekúra
- nahradzuje jeden procesor s poľom regulárnych procesných elementov
- organizuje tok dát pre vysoký výkon s čo najmenšími počtom prístupov do pamäte
- rozdiel od pipelingu → nelineárna štruktúra poľa, multismerový tok dát, každá jednotka môže mať malý zásobní inštrukcii a datovej pamäte
- rodziel od SIMP → vďaka VLSI sú dostupné lacné špecializované čipy
- povoľuje vykonávanie algoritmov priamo na čipoch pripojených k regulárnym vzorom

Porozumenie paralelej architektúre

Paralelné programy pozostávajú z jedného alebo viacerých vláken, ktoré riadia operácie nad datami. Model paralelného programovanie špecifikuje
- ako majú byť data pomenované vlánom
- aké operácie môžu byť vykonané nad dátami
- ako majú byť usporiadané operácie, ktoré sú vykonavané nad datmi

Pomenovanie a operácie

Pomenovanie a operácie v progamovacom modely môžu byť priamo podporované nízkolevelovo alebo môžu byť preložené kompajlerom, knižnicou alebo OS.
HW podporuje zdielaný fyzický adresný priestor. Ten môže vykonávať SAS prostredníctvom OS v systémovom/ používateľsko rozhrania.

Usporiadanie

Pri message passingu niesú predpoklady na preusporiadania procesov v porovaní s očakávaním usporiadaním procesov s výnimkov preusporiadania v páre odosielateľ – prijímateľ.
SAS – Ako proces vidí poradie ostatných procesov, ktoré odkazujú na definíciu sémantiky SAS
- usporiadanie je veľmi dôležité a vhodné
- pri jednojadrových procesoroch (uniprocesor) sa pomocou preusporiadania dosahuje trik získania paralelizmu (práve pri týchto procesoroch je preusporiadavenie dôležitejšie ako pri multiprocesoroch)

Synchronizácia

Vzájomné vylúčenie (zámky)
- uisti sa, že konkrétna operácia na konkrétnych dátach môže byť vykonaná len raz v jednom čase
- ostatné procesy vtedy nemôžu pristupovať k procesu
- nieje garantované žiadne usporiadanie

Sychronizačná udalosť
- usporiadanie udalosti je závisle od vzťahu producer – konzumer
- sú 3 hlavné typy
  - point-to-point
  - globálne
  - skupinové

Replikácia
- veľmi dôležitá pre redukovanie datovej komunikácie
- závisla na pomenovaní modelu
- unipotcesory
  - replikáciu vykonávaju cache automaticky
  - redukuje komunikáciu s pamäťov
- message passing naming model
  - obdržiava repliky a dáva im nové mená, súčasne používa nové mená
  - repliky sú explucitne v softvéri rozhrania
- SAS pomenovania modelov v rozhraniach
  - prináša transparetnosť dát, takže dáta môžu byť nahradzované transparentne

    - vykonávanie cache pamäťou (zdielaným fyzickým adresným priestorom)
    - OS to môže vykonať na leveli stránky, zdielaného adresného priestoru alebo objektu
    - explicitné premenovanie, veľa kópii má rovnaké menod a dochádza k problému súdržnosti

Zdielanie adresného priestoru (programovací model)

Pomenovanie – hocijaký proces môže volať hocijakú premennú v zdieľanom priestore

Operácie – načítanie, uloženie a ďalšie operácie potrebné pre preusporiadanie

Jednoduchý model preusporiadania

- bez procesov / vláken: poradie ako v sekvenčom programe
- medzi vláknami: už zavádza nejaké vkladanie medzi behy
- pridaný výkon na uskutočnenie synchroniacie
- kompailer/HW môže vynútiť poradie

Message passing programming model

Pomenovanie – proces môže pomenovať privátne dáta priamo, nezdiela sa adresný priestor

Operácie – explicitná komunikácia pomocou posielania a prijímania

- posielanie prenáša dáta z privátneho adresného priestoru na iný processor
- obdržanie kópe dát z procesora na privátny adresný priestor
- musí byť schopné premenovať proces

Preusporiadanie

- program usporiadáva bez procesov
- posielanie a prijímane môže byť vykonávane pomocou pt-to-pt synchronizácie medzi procesmi
- dedenie vzajomneho vylúčenie

Môže konštruovať globálny adresný priestor

- číslo procesu + adresa tvoria identifikáztor v adresnom priesotre

Vykonávanie komunikácie

Fundamentálne existujú tri charakteristiky komunikácie

- oneskorenie (latencia) – čas, ktorý sa vykonáva jedna operácie
- širka pásma (bandwidth) – ohodnotenie vykonania operácie
- cena (cost) – vplyv na čas vykonania programu

Lineárny model prenostu dát

Transfer time (n) = $T_0 + n/B$

- užitočný na posielanie sprav, prístup k pamäti, vektorové operácie
- s nárastom *n* rastie šírka pásma *b* asymptoticky
- veľkosť potrebná pre polovičnú širku pásma je $n_{1/2} = T_0 / B$

Komunikačný cenový model (communication cost model)

Comm Time per message = Overhead + Assist Occupancy + Network Delay + Size/Bandwidth / Connection = $o_v + o_c + l + n/B + T_C$

Každý komponent má vlastny spôsob obsadenia (occupancy) a spomalenia

- celkové spomalenie je suma všetkých spomalení
- celkové obsadenie je najväčšie obsadenie: *Comm Cost = frequency * (Comm time – overlap)*

Recap

Paralelná architekúra je dôležité vlákno vo vývoji architektúry

- na všetkých leveloch
- viacero procesorov je momentálne mainstream v počitačov

Exotický dizajn ti dáva viacero ciest na konvergenicu

Koncept ukladania programov

- postupnosť inštruckii je vykonávaná na dátach uložených v registroch
- program = postupnosť inštukcii (operácii a operandov))
- ištrukcia => binárny kód => postupnosť mikrooperácii

- pamäť
- kontrolné registre
- kontrolná jednotka
- unikátna množina inštrukcii

Klasifikácia architektúr

Na základe účelu
- so všeobecným účelom
- so špeciálnym účelom

Na základe funkcionality
- analógové
- digitálne
- hybridné

Flynová taxonómia
- počet inštrukcii, kt môžu byť spustené a ako pracujú s datami
- SISD = Single Instruction Single Data
- SIMD = Single Instruction Multiple Data
- MIMD = Multiple Instructions Multiple Data
- MISD = Multiple Instructions Single Data

SISD (Single Instruction Single Data)
- tradičná sekvenčná architektúra
- u sériových PC – deterministický spúštaný
- na CPU sa spúšťa počas jedného taktu len len 1 inštrukcia
- len jeden datový stream je použitý ako vtip počas 1 cyklu

SIMD (Single Instruction Multiple Data)
- procerové polia, vektorové procesory
- typ paralelného PC – deterministicky spušťaný
- všetky procesné jednotky spúšťajú rovankú inštrukciu počas 1 strojového cyklu
- každá procesná jednotka môže operovať na rôznom elemente dát
- najvhodnejšie pre špecializované problémy charakterzovavané vysokým stupňom reguality (napr. grafika, spracovanie obrazkov)
- podobné vlastnosti ako u vektorových architektúr, ale paralelizmus je dosiahnutý s pipeliningom

MIMD (Multiple Instruction Multiple Data)
- najbežnejší typ paralelného PC
- každý procesor môže vykonávať rozličný prúd inštrukcii
- každý procesor môže operovať na rôznom elemente dát
- spúšťanie može byť synchrónne, asynchrónnne, deterministické, nedeterministické
- sem spadá momentálne väčšina paralelných PC a väčšina superpočítačov (clustre, MPP, datacentrá)

MISD (Multiple Instruction Single Data)
- typ paralelného PC
- nie je bežný
- každá procesná jednotka operuje na dátach nezávisle pomocou separatného prúdu inštrukcii
- samostatný tok dát je dávkovaný do viacerých procesných jendotiek

Fengová klasifikácia / Fengova klasifikacia / fengova klasifikacia
- Tse-yue Feng odporúčíl puožívať rôzne stupě paralelizmu na klasifikáciu rôznych architektúr PC
- rozlišoval sekvenčné a paralelné operácia na levely bitu a na leveli slova
  1. Word Serial Bit Serial (WSBS)
  2. Word Serial Bit Parallel (WSBP)
  3. Word Parallel Bit Serial (WPBS)

4. Word Parallel Bit Parallel (WPBP)

Erlangensová klasifikácia / Erlangensova klasifikacia / erlangensova klasifikacia

- Systém je rozdelneny do troch levelov komplexnosti
  - základný logický level obdovu (ELC) je najbežnejší – operuje na bitovom leveli
  - level aritmetických a logických jednotiek (ALU) spúšťa série operácia na ELC levely
  - programy riadené na levely jednotky (PCU) interpretujú program inštrukciu po inštrukcii

Sima-Fountain-Kacsuk klasifikácia / Sima-Fountain-Kacsuk klasifikacia / sima-fountain-kacsuk klasifikácia

- Systémy rozdeluje z hľadiska toho ako je dosiahnutý paralelizmus
  - paralelizmus dosiahnutý operovaním na viacerých dátach → datový paralelimzus
  - paralelizmus dosiahnutý vykonávaním viacerých funkcii paralelne → funkcionálny paralelizmus (riadiaci paralelizmus, úlohový paralellizmus)

Výkon paralelných architektúr

- typycké metriky na meranie výkonu
  - čas odozvy → čas behu programu
  - priepustnosť
  - MIPS
    - Million Instruction Per Second
    - MIPS = instuction count / (Execution time * 10^6)
    - GIPS = 10^3 MIPS
  - MFLOPS
    - Million Floating Point operation Per Second
    - MFLOPS = FP OPS v programe / (Execution time * 10^6)
    - GLOPS = 10^3 MFLOPS, TFLOPS = 10^3 GFLOPS
  - SPECRatio
- zrýchelnie

$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\frac{1}{\text{Performance}_Y}}{\frac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

- čas beuhu programu
  - čas nastených hodín, čas odozvy, čas uplynutia programu, CPU čas
- benchmarky
  - programy, používané na meranie výkonu PC
  - kernelové (napr. násobenie matíc)
  - hračkárske programy (napr. triedenie)
  - syntetické benchmarky (napr. Dhrystone)
  - benchmarkové sady
    - desktopové benchmarky
    - serverové benchmarky
  - mikrobenchmarkové sady (micro benchmarks suit)
    - pre numerické výpočty
      - LAPACK
      - ScaLAPACK
      - LINPACK
    - pre meranie prietorku
      - STREAM
  - jadrové benchmarky

- NPB (NAS parallel benchmark)
- PARKBENCH
- SPEC
- SPLASH

Vrchol výkonu

- meraný vo MFLOPS
- najväčší, keď systém nerobí nič len numerické výpočty
- využíva sa na meriane výkonu čistého HW alebo na indikáciu použiteľnosti systému v praxy

Trvalý výkon

- počer MFLOPS, ktorý program dosiahne na konci celého behu
- využíva sa na meranie trvalého výkonu PC Pomocou benchmarkov
- vrchol výkonu (MFLOPS) je zvyčajne oveľa väčší ako trvalý výkon
- efektívnosť = trvalý výkon [MFLOPS] / vrchol výkonu [MFLOPS]

Výhody paralelizmu

- systémový level → škálovateľnosť
- CPU level → pipelining
- digitálny dizajn → cache, vyššia výkonnosť ALU

Princíp lokality

- znovupoužiteľnosť dát a inštrukcii → cache
- dočasné umiestnenie → zabezpečuje, že nedávno použité prvky budú dostupné aj v blízkej budúcnosti
- priestorové umiestnenie → prvkz, ktorých adresa je blízko seba, maju na seba tendenciu referovať v blízkej dobre

Admahlovo pravidlo / Amdahl's Law

- zlomok výpočtového času v skutočnom počítači môže byť prekrytý výhodou vylepšenia
- $Fraction_{enhanced}$ je stále menší ako 1
- zlepšenie nadobudnuté vo výkonovom modé hovorí, ako rýchlo dokáže úloha bežať vo výkonovom móde, ktorá sa používa ako vstup do programu
- $Speedup_{enhanced}$ je stále väčší ako 1

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

Implicitný paralelitmus

- architektúra Johna von Neumanna – konvenčné archtektúry
- každý z komponentov – výkonnostné prekážyk
- zrychlenie výpočtu
  - funkčné vylepšienie
    - zvyšenie taktovacej frekvencie
    - zvyšenie počtu tranzistorov
  - viaceré funkčné jednotyk vykonávajú viaceré inštrukcie paralelne
  - nové prístupy
    - architektúra kvantových PC → kvantové časti ako elementy výpočtu
    - architektúra biopočítačov → sekvencie molekul
- rôzne aplikácie → rôzne aspekty paralelizmu
  - dátovo náročne aplikácie → pamäťová priepustnosť
  - serverové aplikácie →sieťová priepustnosť
  - vedecké výpočti →vysoké výpočtové a pamäťové nároky
- prudové spracovanie = „časový paralelizmus“ - prúdová funkčna jednotka
- paralelné spracovanie = „priestorový paralelizmus“ - N-násobné implementové komponenty

Rozparalelnenie výpočtového procesu

- paralelné spracovanie informácií predstavuje kľúčovu technológiu využívanu v moderných PC, ktorá je vyvolaná požiadavkami na vyššiu výkonnosť, nižšiu cenu a podporu produktiviy reálnych aplikácii
- súbežné vykonávanie proceso sa uskutočňuje ako

    - multiprogramovanie
    - viacprocesorové spracovanie (multiprocessing)
    - viacpočítačové spracovanie (multicomputing)
- tvorí základ paralelizácie procesov, ktoré prebiehajú pri paralelnom spracúvaní úloh, programov, inštrukcii, príkazov a pod.
- paralelizácia procesov (paralelizmus) sa vyskytuje v rôznych formách parallného spracovania, medzi ktoré patrí
    - dopredné prehľadávanie
    - prúdové spracovanie
    - vektorizácia
    - údajový paralelelizmus
    - segmentovanie
    - vkladanie
    - prekrývanie
    - rozmnožovanie
    - prideľovanie času
    - prideľovanie priestoru
    - viacúrovňové spracovanie úloh (mutitasking)
    - mutiprogramovanie
    - multivláknové spracovanie úloh (multithreading)
    - distubuované výpočty rôznej úrovne

Podmienky paralelného vykonania programu

- paralelné vykonanie inštrukcii (príkazov) programu je závislé od splnenia podmienok, kt definujú stupeň paralelizácie procesov vykonania programu
- vo všeobecnosti stupeň paralelného vykonania inštukcii je definovaný
    - údajovými závislosťami
    - štrukturálnymi (zdrojovými) závislosťami
    - riadiacimi závislosťami

Údajové závislosti

- RAW (Read After Write) – čítanie premennej po jej zápise
- WAR (Write After Read) – zápis premennej po jej čítaní
- WAW (Write After Write) – zápis premennej po jej zápise
- V/V závislosť – čítanie a zápis su V/V inštrukcie

RAW:

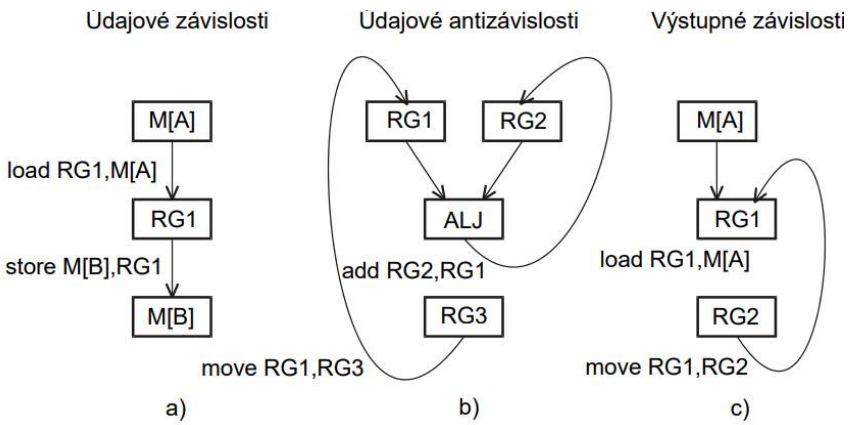
Inštr.1: **load** RG1, M [A] ⇒ RG 1 := M [A]
Inštr.2: **store** M [B], RG 1 ⇒ M [B] := RG 1

WAR:

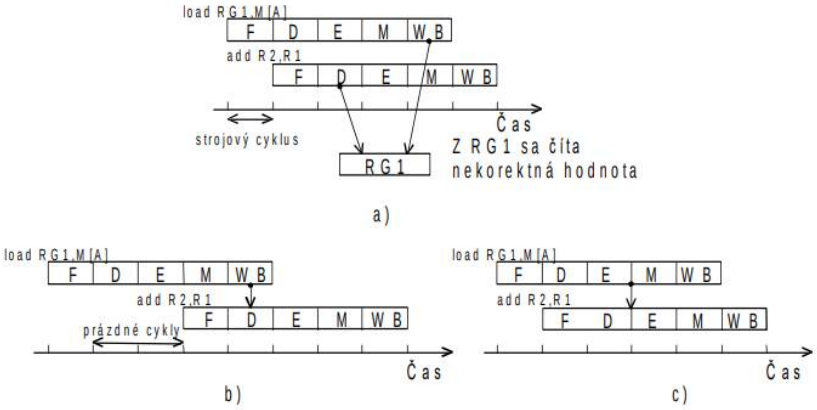
Inštr.1: **add** RG 2, RG 1 ⇒ RG 2 := RG 1 + RG 2
Inštr.2: **move** RG 1, RG 3 ⇒ RG 1 := RG 3

WAW:

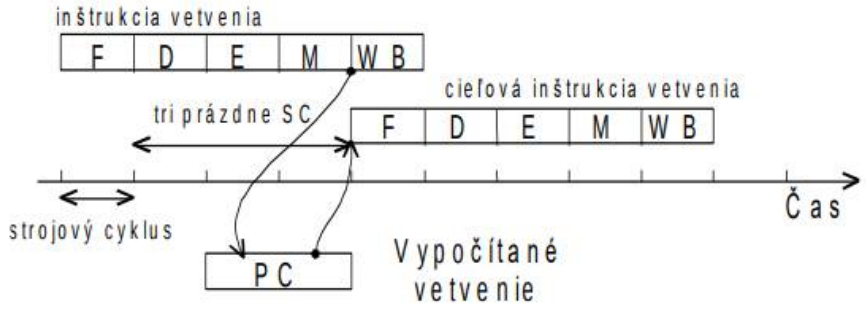
Inštr.1: **load** RG 1, M [A] ⇒ RG 1 := M [A]
Inštr.2: **move** RG 1, RG 2 ⇒ RG 1 := RG 2



- Údajové závilsosti sa osobnitne skúmajú pri paralelnom vykonávaní rozličných fáz v spracovanom prúde inštrukcií, kedy môžu vzninúť rovnako pomenované hazardy, ktorých vznik, resp. následky treba odstrániť.
- Pri jednoduchom prúdovom spracovaní inštrukcií môže vzniknúť iba hazard typu RAW. Využitie spoločného registra v dvoch rôznych fázach nasledujúcich inštrukcii.
- Hazardy WAR a WAS vznikajú iba v osobitných prípadoch prúdového spracovania inštrukcií
- Hazard RAW pri prúdovom spracovaní môžeme odstrániť
  - SW prostriedkami
    - vloženie prádznej inštrukcie po inštrukcii, kt môže spôsobovať hazard
    - preusporiadanie vykonávania inštrukcii
  - HW prostriedkami
    - blokobanie prúdového spracovania, v dôsledku ktorého sa generujú prázdne cykly v prúdovom spracovaní ionštrukcií
    - dopredné generovanie výsledku predchadzajúcej inštrukcie pre jeho použitie ako vstupného operanda nasledujúcej inštrukcie

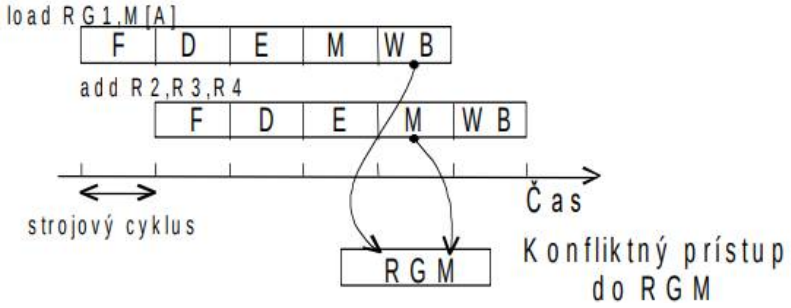


<u>Riadiace závislosti</u>



Zdrojové závislosti

- ak dve inštruckie požiadajú k ich vykonávaniu rovnakú funkčnú jednotku, nemôžu sa súčasne realizovať v dôsledku zdrojovej závislosti
- odstránenie zdrojovej závislosti sa uskutočnuje
  - vložením prázdenho SC (oneskorenie vykoanania druhej inštrukcie)
  - použitím viacnásobných zdrojov
    - viac funkčných jednotiek: *m x sčítačka, n x násobička, p x …*
    - rôznych registrov, kt časovo oddeľujú prístup do registrovej pamäte

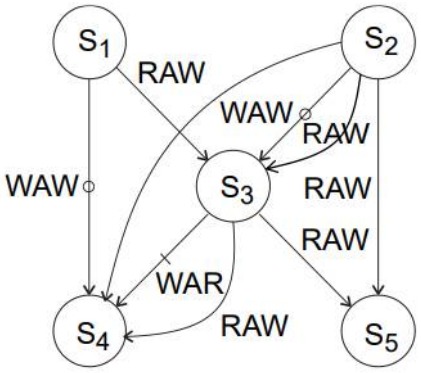


Graf závislosti

- pre účely analýzi paralelizácie procesov sa závislosti spravidlo zobrazujú grafom záislosti
  - orientovaný graf
  - uzly: programové entity
  - hrany: typy závislosti

Príklad : Zostavte graf údajových závislostí pri vykonávaní nasledujúcich inštrukcií programu v procesore DLX:

Inšt.1 : **load** R1, M[A]	R1 ← M[A]
Inšt.2 : **load** R2, M[B]	R2 ← M[B]
Inšt.3 : **add** R2, R1	R2 ← R1 + R2
Inšt.4 : **move** R1, R2	R1 ← R2
Inšt.5 : **store** M[B], R2	M[B] ← R2



Hrany v grafe závislostí reprezentujú:

⟶ údajové závislosti (RAW),
⟶| údajové antizávislosti (WAR),
⟶o výstupné závislosti (WAW)

Bernstrainove podmienky

- Procesy P_i a P_j $(i < j)$ môžu byť vykonané paralelne, ak medzi množinami vstupov I_i a I_j a množinami výstupov O_i a O_j platia

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

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

- Procesy (vlákna, príkazy, inštrukcie) $P_1, P_2, \ldots, P_n$ môžu byť vykonané paralelne, ktorá skutočnosť sa formálne vyjadruje v tvare

$$P_1 \parallel P_2 \parallel \ldots \parallel P_n$$

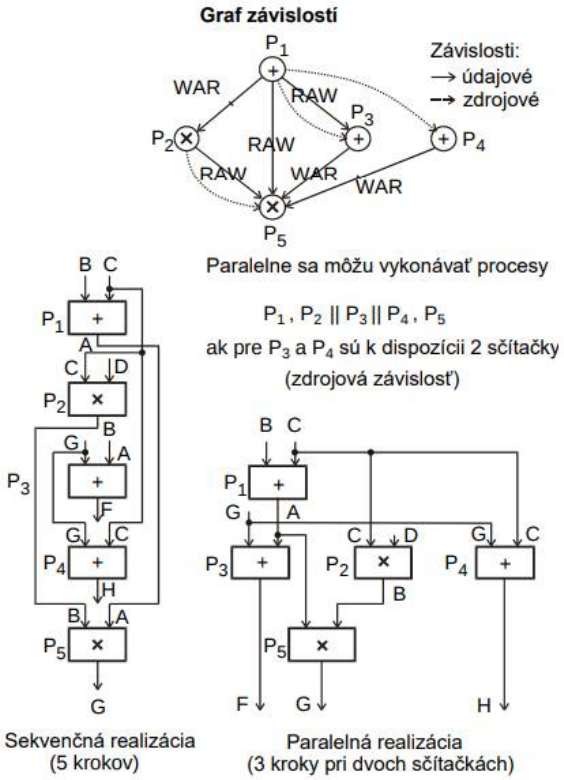
ak

$P_1 \parallel P_2$ pre všetky $i \neq j$

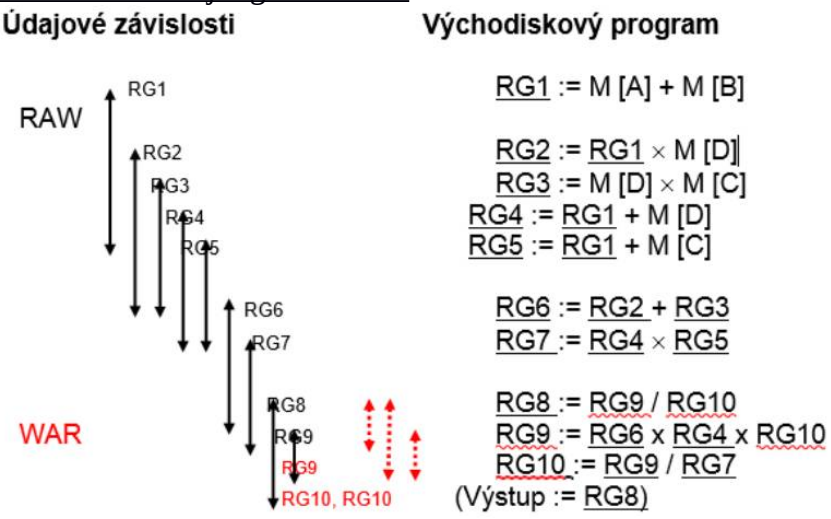
Príklad. Na základe Bernsteinových podmienok zistite možný paralelizmus medzi nasledujúcimi príkazmi.

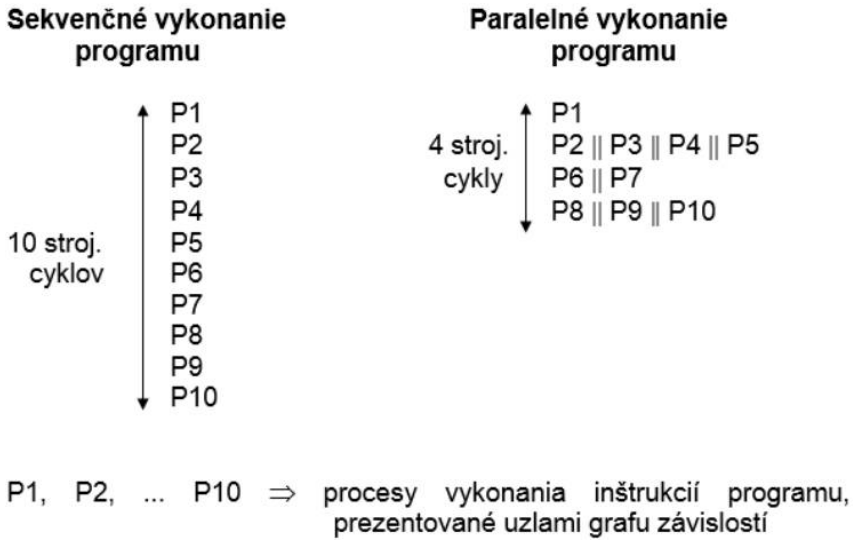
$P_1 : A = B + C$ $P_2 : B = C \times D$ $P_3 : F = G + A$

$P_4 : H = G + C$ $P_5 : G = A \times B$



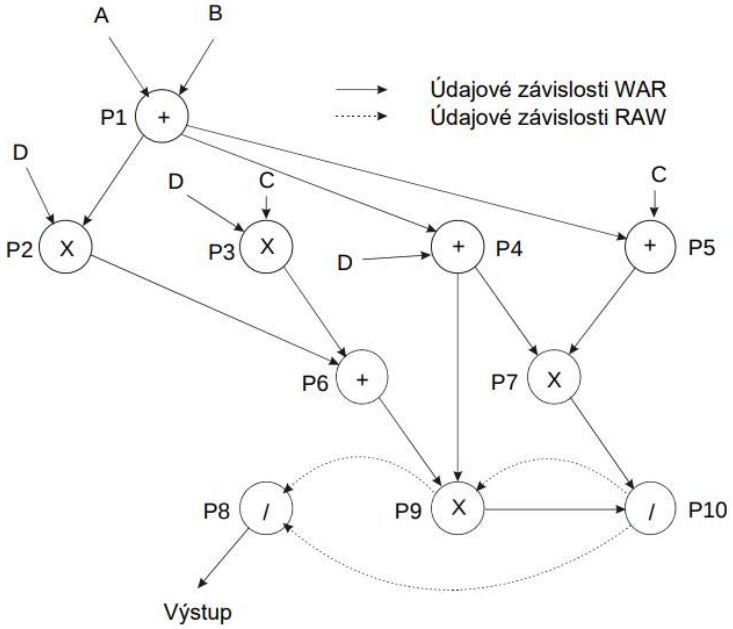
Návrh štrukturálnej organizácie FJ





P1, P2, ... P10 ⇒ procesy vykonania inštrukcií programu, prezentované uzlami grafu závislostí

Graf údajových závislostí

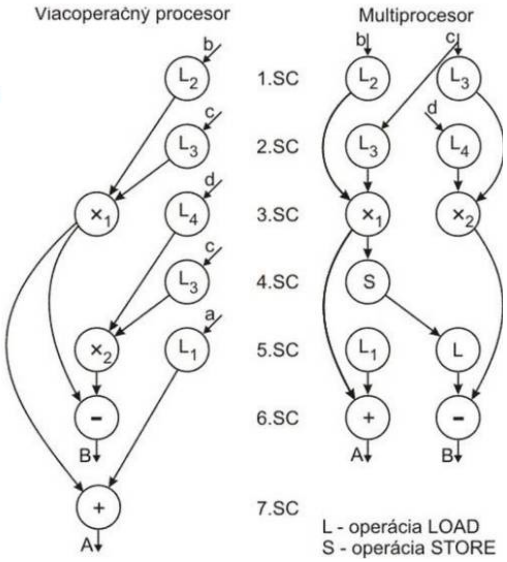


Implementáci paralelizmu

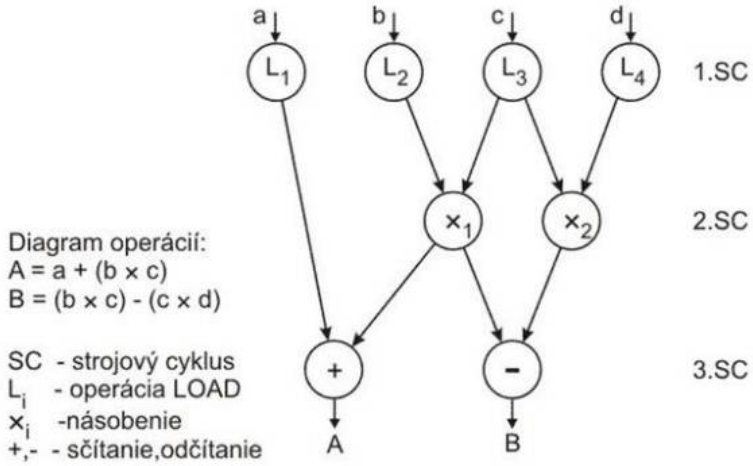
- impelementácia paralelizmu vyžaduje príslušnú SW a HW podporu, ktorou je reprezentovaný
  - HW paralelizmus
    - je def. Zdrojovými závislosťami v programe
    - základným faktorom HW paralelizmu je počet inštrukcii za jeden inštrukčný cyklus
      - jednooperačný porcesor (inštrukcia sa vykoná za jeden, alebo niekoľko strojových cyklov – konvenčný procesor)
      - viacoperačný procesor (najmenej dve inštrukcie sa vykonajú za jeden strojový cyklus – RISC, prúdové procesory)
      - viac prúdový procesor (multiprocesor)

- Viacoperačný procesor
  - 1× Load/Store jednotka
  - 1× ALJ

$P_{HWP} = N_i / N_{SC} =$
$9/7 =$
1,28 inštrukcií/SC

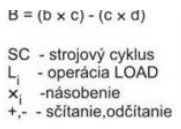


- SW paralelizmus
  - je def. pomocou riadiacich a údajových závislostí programu
  - stupeň sw paralelizmu je viditeľný v profile programu alebo vo vývojovom grafe programu
  - je funkciou
    - algotimu
    - štýlu programovania
    - oprimalizujúceho kompilátora



- inštrukčný SW paralelizmus je definovaný
  - štyrmi operáciami v 1. SC
  - dvoma operáciami v 2. SC
  - dvoma operáciami v 3. SC

$P_{SWP} = N_i / N_{SC} = 8/3 =$
2,67 inštrukcií/SC

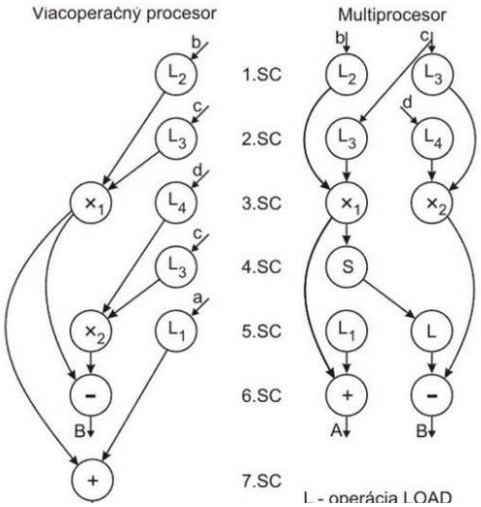


kde

P_{SWP} – priemerný SW paralelizmus,
N_i – počet inštrukcií spracovaných paralelne
N_{SC} – počet SC paralelného výpočtu.

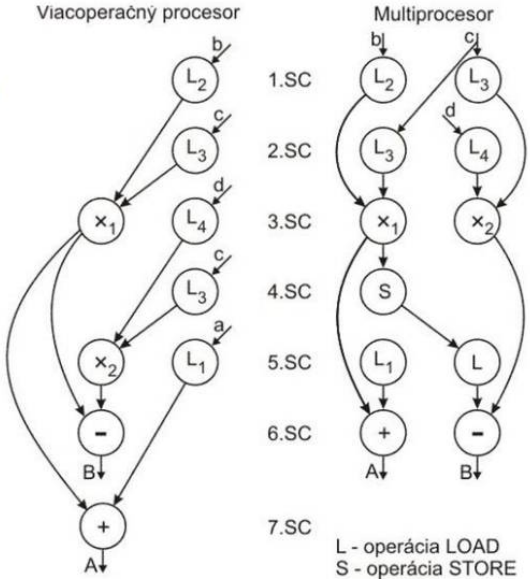
- Viacoperačný procesor
  - 1× Load/Store jednotka
  - 1× ALJ

$P_{HWP} = N_i / N_{SC} =$
$9/7 =$
1,28 inštrukcií/SC



- Multiprocesor
  - 2× Load/Store jednotku
  - 2× ALJ

$P_{HWP} = N_i / N_{SC} =$
$11/6 =$
1,83 inštrukcií/SC



Rozklad programu

Rozklad programu na programové segmenty a plánovanie poradia ich vykanannia je závisle na dvoch základných atribútoch paralelizácie výpočtového procesu, ktorými sú:

- granulartia výpočtu
- komunkačná latentnosť

Na základe programu sa na definovanej úrovni môže zúčastňovať

- tvorca algoritmu
- programátor
- kompilátor
- OS

Granularita je miera množstva výpočtov v programových segmentoch výpočtového procesu pri jeho paralelnom spracovaní. Počet inštrukcii, resp cyklov (v procesore a pamäti), potrebných pre vykonanie všetkých operácii v sedmente sa nazůy rozmer granularity

- adresovanie inštrukcie: 1 SC
- prenos informácie z pamäte do registra procesora: 4 SC
- prenos obsahu do registra (prenos medzi registrami): 2 SC
- vykonanie operácie v procesore: 3 SC

Komunikačná latentnosť je časová miera komunikačnej réžie medzi procesormi, resp. komponentami PC systému pri paralelnom spracovaní úlohy. Ovplyvňuje

- zložitosť paralelného systému
  - napr. pre komunikáciu *n* úloh je potrebných *n x (n – 1) / 2* komunikačných liniek
- paralelné spracovanie

- je potreba vyváženie rozmeru výpočtovej granularity a komunikačnej latentnosti

Paralelizmus výpočtového procesu môže predstavovať rôzne úrovne paralelného spracovania, zohľadňujúce:

- rozlyčný rozmer výpočtovej zložitosti
- zmeny komunkačnej latentnosti
- požiadavky riadenia

V závislosti na úrovni paralelného spracovania sa rozlišujú 3 úrovne granularity reprezentujúce 5 úrovní paralelizácie:

- jemná granularita
  - využívaná pomocou paralelizujúcich a vektorizujúcich kompilátorov

1. inštrukčná úroveň (inštrukčná alebo príkazová)
   - rozmer granularity: 2 – 1000 inštrukcií
   - priemerný rozmer granularity: cca 7 inštrukcií

2. úroveň operácie cyklu
   - nerekurzívny cyklus
   - rozvinutie cyklu tpicky obsahujúceho do 500 inštrukcii

- stredná granularita
  - využívaná za pomoci programátora a čiastočne I kompilátora

3. úroveň procedúr
   - využíva sa pri spracovaní úloh (multitasking)
   - bežné subritiny, úlohy a spoločné subrutiny (korutiny)
   - typický rozmer granularity: 2000 inštrukcii

4. úroveň podprogramov
   - pracovný krok -job step, príbuzná časť programu
   - využíva sa v režimoch SPMD, MPMD, pre spracovanie v multiprocesoroxh s prenosom správ
   - pri multirpogramovaní monoprocesorov a multiprocesorov (multuprograming)

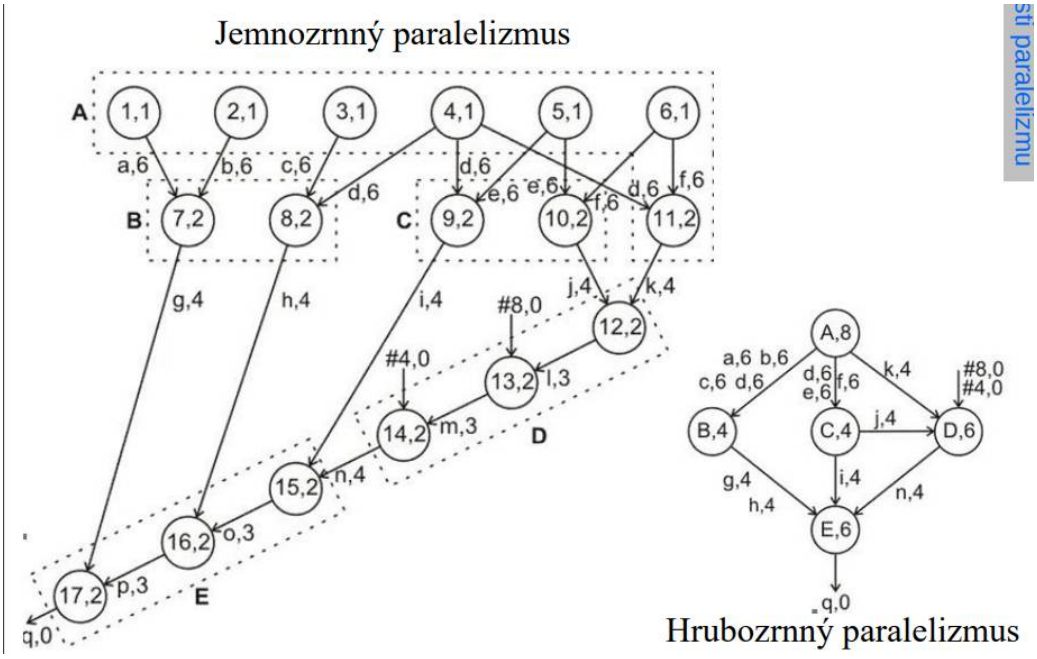
- hrubá granularita
  - využívaná pri spracovaní nezávislých hlavnáych (zḱaladných) programov a prác

5. úroveň prác alebo programov (výnimočne i 4. úroveň)
   - využíva sa pri programovaní paralelných PC, systémov s prideľovaním času a pristoru v prostredí architektúr MIMD, resp paradigmi MIMD
   - typicky rozmer granularity: 10000 a viac inštrukcií (multiprocesing, multiprograming)

Kľučové problémy rozkladu programu

- optimalizácia rozmeru granularity / optimalizácia granularity
  - ako rozložiť program do paralélnych vetiev?
  - aký je oprimálny rozmer granularity?
  - Navrhnúť plánovanie rýchleho vykonania programových segmentov rozloženého programu?
- minimalizácia komunikačnej latentnosti
- zabránenie uviaznutiu programu

Graf paralelného programu (GPP)

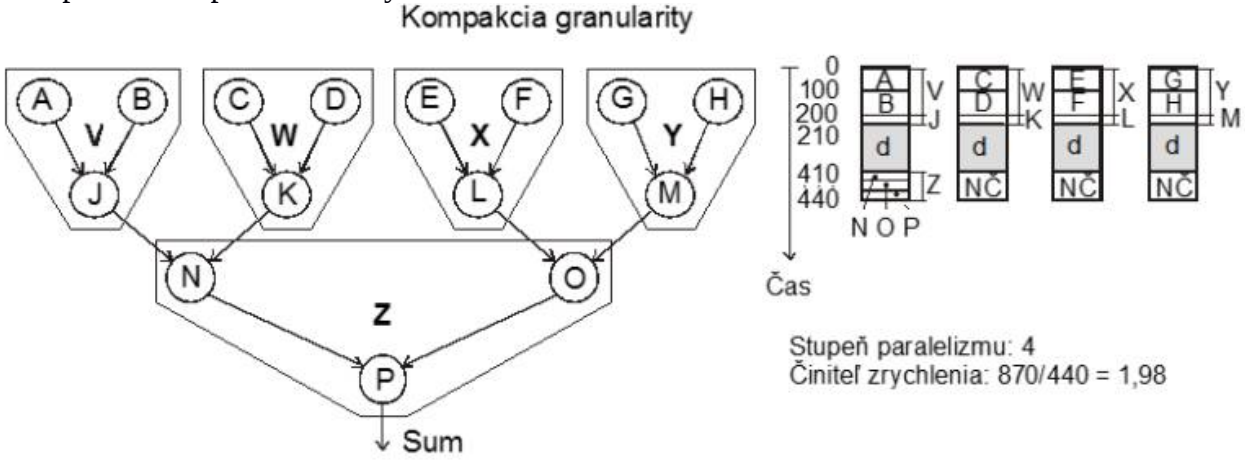


- v oboch grafoch paralelných programov
  - označenia uzlov (m,s) špecifikujú
    - meno (m)
    - rozmer (s) ich granularity
  - označenia hrán (v,d) špecifikujú
    - výstupnú/vstupnú premennú (v) z/do zdrojového/cieľového uzla
    - komunikačnú dobu (d) medzi nimi
  - uzly 1 – 6 sú pamäťové referencie
    - adresovanie: s = 1 SC
    - výber z pamäte: d = 6 SC
  - uzly 7 – 17 sú
    - operácie CPU: s = 2 SC
  - výstupné hrany uzlov 1 – 6 reprezentujú komunikačné doby d = 6 SC
  - výstupné hrany uzlov 7 – 11 a 14 reprezentujú komunikačné doby d = 4 SC
  - výstupné hrany uzlov 12, 13, 15, 16 reprezentujú komunikačné doby vo vnútri hrubozrných programových segmentov, d = 3 SC
- kompakcia (zhusťovanie) granularity GPP ponúka
  - dosiahnutie kompromisu medzi stupňom paralelizmu a
  - režiou komunikácií medzi uzlami grafu
- operácie zahrnuté do jedného uzla GPP sa vykonávajú spravidla jedných procesorom
- časy komunikácii vnútri hrubozrných uzlov GPP sú zanedbateľné oproti časom komunikácii medzi týmito uzlami
- jemnozrný paralelizmus charakterizuje väčší počet medziprocesorových komunikácii – pri kratšej dobe vykonania operácii definovyných v uzloch GPP
- hrubozrný paralelizmus charakterizuje menší počet medziprocesorových komunikácii – pri dlhšej dobe operácií definovaných v uzle GPP
- voľba rozmeru granularity rozhodujúcim spôsobom ovplyvňuje najkratšiu ežiu spracovania GPP, ktorá je priamo závisla na čase medzirpocesorových komunikácii

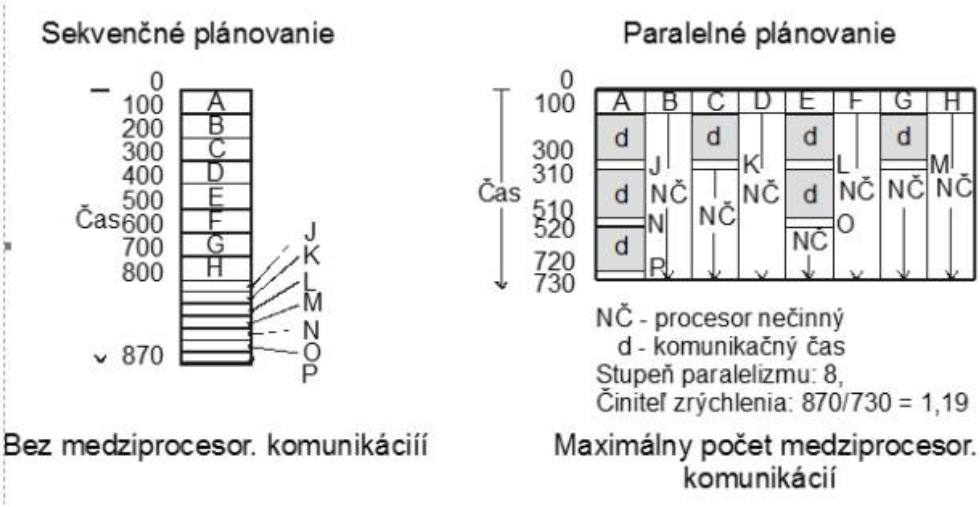
Plánovanie spracovania GPP

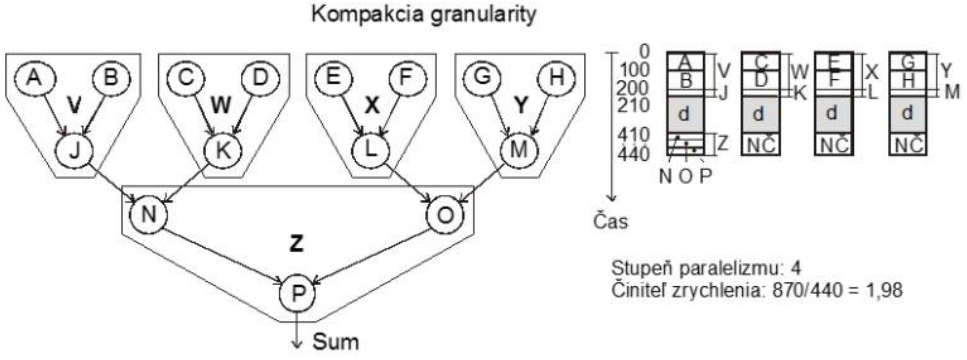
Postup pri určení rozmeru granularity a optimalizácie procesu paralelného spracovanioa sa uskutočňuje v nasledujúcich krokoch:

1. návrh jednorozmerného GPP na základe údajových závislostí
2. plánovanier paralelného vykonania GPP
3. kompakcia GPP za účelo zväčšenia jeho granularity
4. plánovanie paralelného vykonania hrubozrného GPP



Granularita:
V = W =X = Y = 2x100 + 10 = 210 SC
Z = 3x10 = 30 SC





Granularita:
V = W =X = Y = 2x100 + 10 = 210 SC
Z = 3x10 = 30 SC

Škálovateľnosť paralelných PS

Škálovateľnosť

- v PC prostredí môžeeme chápať ako schopnosť systému pružne reagovať na rastúce alebo klesajúce nároky uživažeľov na výkon systému
- jej def nieje jednoduché, lebo v konkrétnych prípadoch je vždy nutné presne definovať parametre systému, ktoré sú pre daný systém kľčové
- *Počítačový sysem alebo program je škálovateľný, ak účinnosť systému ostáva konštantná pri rastúcom počte procesných elementov*
- alayza škálovsteľnosti určuje, či paralelné spracovanie daného problému poskytuje zlepšenie výkonnosti PC
- paralelné alg implementované v reálnom prostredí sú závusle na konkrétnej realizácii tohto prostredie. Sú preto vždy závislé na architektonickom riešení.
- V ideálnom prípafde sú paralelné algortimy definované pre modely PRAM
- analýza škálovateľnosti paralelného alg vychádza z jeho rovnakej (konštantnej) účinnosti (izoúčinnosti) v danom implementačnom prostredí
- Koncepcia izoučinnosti (E) umožňuje vytvoriť vzťah medzi pracovnou záťažou definovanou rozmerom problému *s* a strojovýnm rozmerom *n* tak, aby sa udržala konštantná účinnosťi pri implementácii paralelného algoritmu v paralenom prostredí
- Izoúčinnosť je definovaná
- ak

$$E(s,n) = \frac{P(s)}{P(s) + h(s,n)}$$

kde

$P(s)$ je pracovná záťaž interpretovaná ako funkcia argumentu s, t. j. $P \equiv P(s)$,

$h(s, n)$ je súhrnná komunikačná réžia paralelného spracovania algoritmu, interpretovaná ako funkcia argumentov s a n, t. j.

$$h \equiv h(s, n).$$

Z úpravy predchádzajúceho vzťahu vyplýva

$$P(s) = \frac{E}{1-E} h(s,n) = K \times h(s,n)$$

kde $K = E/(1-E)$ je konštanta za predpokladu, že účinnosť má nemennú hodnotu.

- škálovateľnosť má piarmu súvislosť so zrýchlením praralelného PC systému
- pre analýzu škálovateľnosti je dief 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 zákonnom

Pracovná záťaž (workload) – P

- množstvo vypočtov vykonaných v paralelnom prostredí

Rozmer problému (problem size) – S

- označuje sa aj ako zložitosť výpočtového procesu
- kvantifikuje veľkosť pracovnej záťaže

Strojový rozmer (machine size) – n

- kvantifikuje paralelné výpočtové prostredie prostredníctvom počtu jeho procesných elementov (processing element)

Determinizmus

- deterministické algoritmi sú implenetovateľné na reálnom stroji a hodnotené časovou zložitosťou výpočtového procesu

Granulartia výpočtového procesu

- granularita špecifiku rozmer údajových položiek a programových modulov výpočtum v tomto zmysle sú klasifikované algoritmy na: jemnozrné, stredozrnné a hrubozrnné

Profil paralelizmu

- rozdelenie stupňa paralelizmu v algoritme odhaľuje možnosť paralelného spracovania

Komunikačná réžia

- zahrňuje adresovanie pamäťového prístupu a medziprocesorových komunikácii, kr v závislosti na algoritme môžu byť
  - statické – vhodné pre prúdové archotektúry alebo SIMD
  - dynamické – vhodné pre MIMD

Uniformita operácii

- jednotnosť operácií špecifiku základne operácie, kt sú definovné v danom algoritme

Vplyv údajových štruktúr

- údajové štruktúry majú pri riešení rozsiahlých výpočtoch zasadný vplyv na veľkosť pamäťového priestoru, účinnosť pamäte a pohyb údajov pri riešení algoritmu

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

Asymptotycké zrýchlenie

Asymptotické zrýchlenie *S(s,n)* výkonnosti je najlepši zrýchlenie dosiahnuteľne pri danej architetúre, algoritme a rozmere problému v závislosti na premenlivom počte rezistorov *n:*

Analýza

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

kde

T(s, 1) je doba sekvenčného vykonávania algoritmu v uniprocesorovom systéme,

T(s, n) je minimálna doba paralelného vykonávania algoritmu v n-procesorovom systéme,

h(s, n) je doba celkovej komunikačnej réžie.

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

Vo všeobecnosti, max. hodnota E(s,n) = 1, vychádzajúc zo skutočnosti, že najlepšie zrýchôlenie výkonnosti je lineárne t.j. *S(s,n) = n*.
Z úvedeného vyplýna nasledujúca definícia školovateľnosti:
PPS je školaovateľný, ak hodnota systémovej učinnosti E(s,n) = 1 pre všetky algoritmy s ľubovoľným počtom procesorov n a ľubovoľným rozmerom problému s.
V prípade zanedbania komunikačnej réžie v ideálno prostredí modelu PRAM je asymptotické zrýchlenie vyjadtenie vzťahom:

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

kde $T_i(s,n)$ je doba paralelného výpočtu v modeli PRAM pri zanedbaní komunikačnej réžie.
Aritmetická priemerná výkonnosť (R_A) výpočtového procesu je def. vzťahom:
Ak
výskyt

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

kde

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

T_A je priemerná doba vykonania m programov.

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

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

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

Z definície váhovej funkcie a predchádzajúcich vzťahov vyplýva:

Na

$$R_A \neq R_A^*$$

$$R_A = R_A^* \quad \text{, ak } 1/m = f_1 = f_2 = ..., = f_m$$

kde R_A a R_A^* označujú teoretickú výkonnosť výpočtu.

$$T_H = \frac{1}{m}\sum_{i=1}^{m} T_i = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{R_i} \Rightarrow R_H = \frac{1}{T_H}$$

kde

T_i [čj] je priemerná doba vykonania i-teho programu (i = 1, 2, ..., *m*) a $T_i = 1 / R_i$

Vážená harmonická premierná doba vykonania *m* programov je definovaná v tvate

$$T_H^* = \sum_{i=1}^{m} f_i T_i = \sum_{i=1}^{m} f_i \frac{1}{R_i} \Rightarrow R_H^* = \frac{1}{T_H^*} = \frac{1}{\sum_{i=1}^{m} f_i \frac{1}{R_i}} = \frac{1}{\sum_{i=1}^{m} f_i / R_i}$$

$$R_H^* = \frac{1}{\sum_{i=1}^{m} f_i / R_i}$$

- posudzovať výkonnosťprostredníctvom *m* programových benchmarkov
- reálnejší prístup oceňovania paralelného výpočtového prostredia (jednoduchšie meranie T_i vs R_I)

Iný prístup oceňovanioa paralelného výpočtového prostredia poskytuje harmonické stredné zrýchlenie výkonnosti (S). To vyjadrije pomer zvýšenia výkonnosti pri riešení programu v n-procesorovom paralelnom prostredí (v priebehu časového intervalu T^*_H) v uniprocesorovom sekvenčom prostredí v priebehi časového intervalu T_i

$$S = \frac{T_1}{T_H^*} = \frac{1}{\sum_{i=1}^{n} f_i / R_i}$$

kde *n* je počet vykonávacích módov porgramu, kde i-*ty* mód definuje počet použitých procesorov, T_h^* je vážená harmonikcá premenná doba vykonania programu

$$T_H^* = 1/R_H^* = \sum_{i=1}^{n} f_i / R_i$$

- sformulovaný v roku 1967 nórsko americkým PC architektom Genem Myronem Amdahlom
- pri štúdiu výkonnosti paralelných aplikácií si Amdahlm všimol, že paralizáciou problému môžeme dosiahnúť len relatívne veľmi obmedzené zrýchlenie
- osobitný význam má harmonické stredné zrýchlenie výkonnosti určené pre paralelné prostredie, kt využíva iba jeden procesor v sekvenčnom móde s pravdepodobnosťou α alebo

všetky procesory v úplne paralelno móde s pravdepodobnosťou *1 - α*. Váhová funkcia je v tomto prípade *w = (α, 0, … , 0, 1 - α)*

Za predpokladu, že $R_i = i$

$$S = \frac{1}{\frac{\alpha}{1} + \frac{0}{2} + \ldots + \frac{0}{n-1} + \frac{1-\alpha}{n}} = \frac{1}{\alpha + \frac{1-\alpha}{n}} = \frac{n}{1+\alpha(n-1)} \leq \frac{1}{\alpha}$$

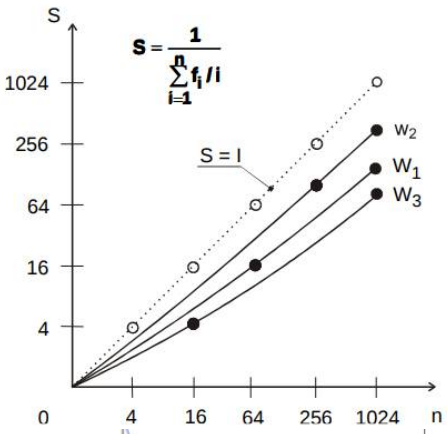
Závery vyplývajúce z predchádzajúceho vzťahu:

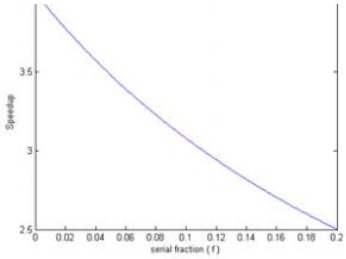
- pri neobmedzenom raste počtu procesorov ($n \to \infty$) je výkonnosť obmedzená hodnotou

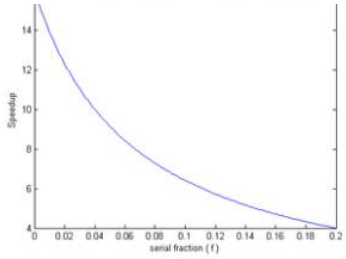
  $S = 1 / \alpha,$

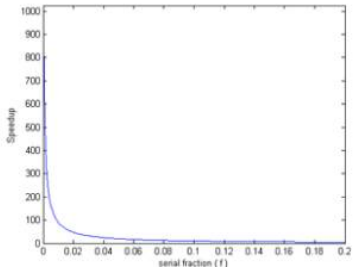
- v ideálnom prípade je $\alpha = 0$, v dôsledku čoho

  $S = n.$









- je dôležité si uvedomiť, že Amdahl rieši problém o konštantnej veľkosti -tzv s rastúcim počtom procesorov sa mení rozsah úlohy
- snaží sa vyriešiť daný problém čo najrýchlejšie

Gustafsonov zákon

- Amdahlov zákon značne obmedzuje maximálne zrýchlenie výpočtu
- pre už pomerne nízke hodnoty sekvenčnej časti nemôžeme dosiahnúť primerane zrýchlenie
- v r. 1988 John Gusrafsom prišiel s novým pohľadom na vec
- Amdahl s rastúcim počtom procesorov nemenil veľkosť problému
- Gustafson si všimol, že zrýchlenie u praktických úloh je výraznejšie vyššie než predpokladal Amdahl
- Dôležitou zmenou je, že miesto konštantného rozhasu úlohy budeme za konštantnú považovať dobu behu
- Gustafson si uvedomi, že s vyšším počtom procesorov je možné rovnkaý problém vyrisšiť v rovnakom čase presenejšie
- označil symbolom α^* sekvenčnú časť programu vykonanú na paralelnom stroji (α v Amdahlovom zákone vyjadruje pomer času stráveného výpočtom sekvenčenj časti k

celkovému času)

- Zrýchlenie potom môžeme zapísať v tvare

$$S = \alpha^* + N \times (1 - \alpha^*)$$

- s rastúcim počtom procesorov rastie aj zrýchlenie
- zrýchlenie nie je zhora ohraničné – možeme dosiahnúť ľubovoľne veľké zrýchlenie

Amdahlov zákon vs Gustafsonov zákon

Amdahlov zákon	**Gustafsonov zákon**
Zaoberá sa možným zrýchlením problému o konštatnej veľkosti s konštantnou sekvenčnou časťou. V tomto prípade je možné zrýchlenie limitované pomernou časťou sekvenčnej časti kódu k paralelnej časti	Za konštantu volí dobu behu v paralelnom systéme. Zrýchlenie nie je obmedzené za predpokladu, že sa rieši dostatočne veľký problém
	Nepopiera zákon Amdahlovho zákona

Metrika výkonnosti a jej aplikácie

Na ocenenie paralelných výpočto sa využívajú nasledujúce parametre výkonnosti paralelných systémov

- činiteľ zrýchlenie (výpočtového procesu) – miera zisku rýchlosti vykonávanie programu v paralelnom prostredí
- účinnosť – stupeň zrýchlenia výkonnosti v porotnaní s max dosiahnuteľnou hodnotou v ideálnom prípade
- redundancia – stupeň prispôsobenia HW a SW paralelizmus resp. rozsah zyšovania pracovnej záťaže
- využiteľnosť – miera využitia zdrojov pri vykonávaní paralelného programu
- kvalita – miera využitia zdrojov pri vykonaní paralelného programu

Činiteľ

- Nech O (n) je úplný počet operácií vykonaný v n-procesorovom systéme
- Nech T (n) je doba vykonania operácie v jednotkovom časovom kroku paralelného systému

$$O(n) > T(n).$$

- Napríklad

O(8) = 1000 v 8-procesorovom systéme sa vykoná za

$$T(8) < 1000 \text{ čj.}$$

Pre jednoprocesorové prostredie (n = 1) je

$$O(1) = T(1).$$

$$S(n) = T(1)/T(n)$$

kde

T(1) je doba vykonania operácie v uniprocesorovom systéme,

T (n) je doba vykonania operácie v n-procesorovom systéme.

Účinnosť E(n)

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{nT(n)}$$

kde $1/n \leq E(n) \leq 1$, pretože $1 \leq S(n) \leq n$

$$R(n) = \frac{O(n)}{O(1)}$$

kde $1 \leq R(n) \leq n$

Využiteľnosť U(n) za predpokladu O(1) = T(1)

$$U(n) = R(n)\,E(n) = \frac{O(n)}{nT(n)}$$

kde $1/n \leq E(n) \leq U(n) \leq 1$, $1 \leq R(n) \leq 1/E(n) \leq n$

$$Q(n) = \frac{S(n)\,E(n)}{R(n)} = \frac{T^3(1)}{nT^2(n)O(n)}$$

Stupeň paralelizmu (SP) vyjadruje počet procesorov, kt sa v danom časovom intervale podieľajú na vykonaní paralelného programu. Je vyjadrený diskrétnou časovou funkciou $SP(t_i)$ na intervale celého vypočtového procesu paralelného programu. V priebehu výpočtového procesu ho ovplyňuje

- štruktúra algoritmu riešenej úlohy
- oprtimalizácia programu
- využitie zdrojov
- podmienky behu programu

Priemerný paralelizmus (P) je def vzťahom

Práca

$$P = \left(\sum_{i=1}^{m} i \,.\, \Delta t_i\right) / \left(\sum_{i=1}^{m} \Delta t_i\right)$$

kde

Δti je súhrnný časový interval, v priebehu ktorého je $SP(ti) = i$,

m je maximálny stupeň paralelizmu.

Hodnota

$$\mathbf{w_i} = \Delta t_i \times r \times i \Rightarrow \mathbf{W} = \sum_{i=1}^{m} w_i$$

kde

w_i je množstvo práce vykonané pri paralelnom spracovaní programu v časovom intervale Δt_i so stupňom paralelizmu SP = i a operačnou rýchlosťou každého procesora r [op/s].

priemerného paralelizmu môže byť interpretovaná ako zvýšenie paralelitácie v reálnom paralelnom výpočtovom prostredí. Táto interpretácia je zavedená pomocou def. asymptotického zrýchlenia

výpočtového procesu. Asymptotické zrýchlenie výpočtového procesu vyjadruje účinnosť paralelného spracovania programu v paralelnom výpočtovom prostredí s neobmedzeným počtom procesorov orpoti spracovaniu v jednoprocesorovom sekvenčnom prostredí.
Východiskom pre def asymptotického zrýchlenia výpočtového procesu je výpočet doby (Tp) vykonania práce pri paralelnom spracovaní. Doba T_P vykonanej práce *W:*

$$T_p = \sum_{i=1}^{m} \Delta t_i, \quad \Delta t_i = \frac{w_i}{r \cdot i}, \quad w_i = \Delta t_i (r \cdot i)$$

Redukovaná doba (*T(j))* vykonania práce v priebehu *m* časových intervalov je Δt_0, Δt_1, … Δt_m s *j* procesormi v každom z ních je

$$T(j) = \sum_{i=1}^{m} \Delta t_i (j)$$

kde

$\Delta t_1(j) = \mathbf{W}_i(j) / r \times j$, pre $i \neq j$ platí $\Delta t_i(j) \neq \Delta t_i$

Doba vykonania práce jedným procesorom:

$$T(1) = \sum_{i=1}^{m} \Delta t_i (1) = \sum_{i=1}^{m} \frac{w_i}{r}$$

Doba vykonania práce v paralelnom prostredí s neobmedzeným počtom procesorov ($j \rightarrow \infty$)
Z

$$T(\infty) = \sum_{i=1}^{m} \Delta t_i (\infty) = \sum_{i=1}^{m} \frac{w_i}{r},$$
$$j \rightarrow \infty, \text{ resp. } 1 \leq i \leq m << j \rightarrow \infty$$

Zaťaženie

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} w_i}{\sum_{i=1}^{m} w_i / i} = \frac{\sum_{i=1}^{m} \Delta t_i \cdot i}{\sum_{i=1}^{m} \Delta_i t} = P$$

Ideálny prípad: $S_\infty = P$.

Reálny prípad: $S_\infty \leq P$, (zohľadňuje komunikačnú a systémovú réžiu paralelného spracovania).

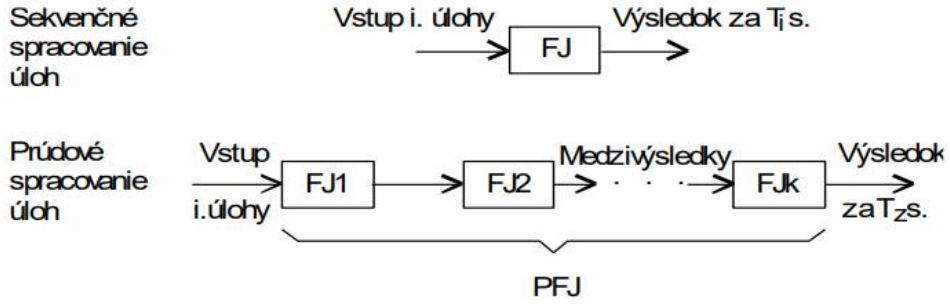
Zaťaženie je všeobecný princíp spracovania informácii založený na rozložený výpočtového procesu ľubovoľnej funkcie (úlohy, operácie a pod) na oddelené kroty, kt sa prostredníctvom samostatných funkčných modulov, označovaných ako stupne (segmenty) zaťaženia realizujú na princípe súčasného prekrývania týchto krokov v priebehu vykonávania za sebou nasledujúcih výpočtových procesov.
Podmienky prúdového spracovania

- inštrukčný cyklus je rozložený na niekoľko fáz, kt. sa môžu konkurenčne (prekrývanie) vykonať s inými fázami predchádzajúcich inštrukcii
- každá fáza sa vykonáva v priebehu jedného alebo viacerých strojových cyklov
- jednotlivým fázam vykonávania zodpovedajú na úrovni obvodového riešenie prístlušné stupe zreťazenia

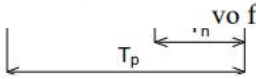
Realizácia
Spracovania *n* úloh, z kt je každá rozložiteľná na *k* podúloh (krokov), môže byť realizované:

- štandardnou funkčnou jednotkou, kompletizujúcou výsledky riešenia jednotlivých úloh vždy až po uplynutí intervalu $T_I = kT_K$ potrebého na ich vykonanie
- zreťazenou (prúdovou) funkčnou jednotkou, kompleizujúcou výsledky riešenia jednotlivých úloh už po uplynutí intervalu T_K, z čoho vyplýva, že výpočet nasledujúcej úlohy začína ešte pred ukončením úlohy prechadzajúcej

Sekvenčné vs prúdové spracovania



Organizácia sekvenčného a prúdového spracovania úloh vo funkčných jednotkách



- Stupeň F (fetch)
  - načítava inštrukcie z pamäte
- Stupeň D (decode)
  - dekóduje inštrukcie
- Stupeň E (execute)
  - vykonáva operáciu definovanú v IF KO
- Srupeň S(store) resp. WB (write back)
  - zapisuje výsledok operácie do registra / pamäte

Časový diagram

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t
1. inštrukcia	F_1	D_1	E_1	WB_1				
2. inštrukcia		F_2	D_2	E_2	WB_2			
3. inštrukcia			F_3	D_3	E_3	WB_3		
4. inštrukcia				F_4	D_4	E_4	WB_4	
5. inštrukcia					F_5	D_5	E_5	
6. inštrukcia						F_6	D_6	
7. inštrukcia							F_7	

Doba výkonania prúdu n inštrukcií

Doba vykonania prúdu **n** inštrukcií

$$T(m,p)=\left(k+\left(\frac{n}{m}-1\right)\frac{1}{p}\right)T_{SC}=\left(k+\frac{n-m}{mp}\right)T_{SC}$$

T_{SC} – perióda SC.

n – počet prúdovo spracúvaných inštrukcií.

k – počet fáz dekomponovanej inštrukcie.

m – počet súčasne spracúvaných prúdov inštrukcií.

p – počet inštrukcií v spracúvanom prúde šírky *m*, vykonanie ktorých začína postupne, v priebehu jedného SC.

Doba vykonania

V prípade skalárneho zreťazenia

$$m = p = 1$$

V prípade superskalárneho zreťazenia

$$m > 1 \text{ a } p = 1$$

Efektívnosť (zrýchlenie) prúdového spracovania je definovaná vzťahom

$$\frac{T_s}{T_p} = \lim_{n \to \infty} \frac{n}{1+(n-1)/k} = k$$

t.j. prúdové spracovanie je **k** násobne rýchlejšie ako sekvenčné spracovanie, pričom spracovanie jedinej úlohy, rozloženej na *k* vykonávacích fáz, môže pri prúdovom režime trvať dlhšiu dobu, ako pri sekvenčnom režime

($T_i \leq kT_k$).

Entity zreťazenia

Prúdové spracovanie výpočtového procesu sa aplikuje najme na

- zreťazenie operácií, ktoré sa uplatňujú v rôznych typoch ALJ
- zreťazenie inštrukcií, ktoré je charakteristické pre procesory s architektúrov RISC
- zreťazenie procesorov v PC systéme, prístupu do pamäte a pod.

Syntéza prúdového spracovania

Základné úlohy syntézy prostriedkov na zreťazené spracovanie úloh

- alnalýza procesu zreťazenia
  - rozklad procesov, úloh, operácii na čiastkové časti a z toho vyplývajúci počet štruktur segmentov zreťazenia na ich spracovanie

    - určenie času prechoduudajov segmentami zreťazenia, z ktorých je zreťazený systém vytvorený
    - vyhodnotenie výkonnosti zreťazeného systému
- stratégia riadenie procesu zreťazenia
    - organizácia prideľovania segmentov zreťazenia vstupnému prudu údajov
    - optimálne prekrytie segmentov zreăzenia s cieľom dosiahnúť ich max vyťaženosť a výkonnosť zreťazeného systému
- návrh technických prostriedkov
    - obvodové rieŠenie jednotlivých segmentov zreťazenia a oich vzájoimnej interakcie
    - synchronizácia a riešenie mechanizmov riadenia

Zreťazený systém

Cieľom syntézy zreťazenia je návrh prúdovej (zreťazenej) funkčnej jednotky (PFJ) reprezentovanej vo všeobecnosti lineárnym spôtnoväzbovým prepojením jej funkčných stupňov.
Postupnosť aktivácie jednotlivých stupňov v priebehu zreťazeného spracovania danej funkcie v PFJ sa zobrazuje v jej rezervačnej tabuľke (RTB).

Systém zreťazenia

- lineárny systém zreťazenia
    - vstupný prúd údajov postupne prechádza všetkým jeho k stupňami počas n synchronizačných intervalov (krokov) zreťazenia
- Celkový čas ($\mathbf{T_p}$) zreťazenia

$$T_p=\sum_{j=1}^{k} m_j T=\sum_{j=1}^{k} T_{pj}, m_j \in \{1,2,...\}$$

kde

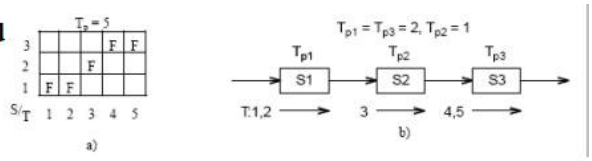
$\mathbf{T_{pj}}$ - je čas spracovania údajov v **j.**-tom stupni zreťazenia,

$\mathbf{m_j}$ - násobnosť taktovacej periódy **T** zreťazenia v jeho **j.**-tom stupni.

    - sa charekterizuje spätnoväzobným prepojení jeho jednotlivých stupňoc
    - procesa zreťazenia ne je def jednoznačne, tj môže byť zobrazený viacerými RTB
    - takéto zreťazené systémy sa nazývajú dynamické
- inicializácia zrežtazenia

Vstupný prúd údajov na obrázku (a) prechádza nasledujúcou postupnosťou stupňov

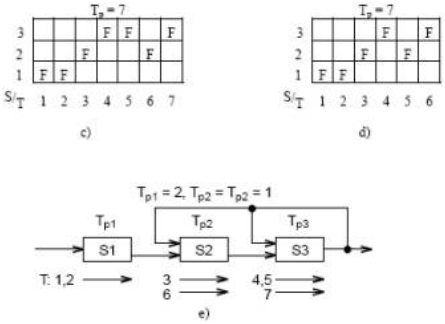
$S_j(T_{pj}):S_1(2),S_2(1),S_3(2)$



Vstupný prúd údajov na obrázku (b) prechádza nasledujúcou postupnosťou stupňov

$S_j(T_{pj}):S_1(2),S_2(1),S_3(2),S_2(1),S_3(1)$

$S_j(T_{pj}):S_1(2),S_2(1),S_3(1),S_2(1),S_3(1)$



Rezervačné tabuľky a systémy zreťazenia
(a - lineárne, b -nelineárne)

  - začiatok procesu zreťazeného spracovania vstupného prúdu údajov zobrazeného v RTB
- latentnosť
  - počet synchronizačných intervalov (krokov) medzi začiatkami dvoch za sebou nasledujúcich inicializácii zreťaznia
  - cyklicky opakovaná sekvencia prístupnych latentnosti sa nazýva cyklus latentntnosti L_s^j
  - minimálna latentonosť L_{min}

<u>Stavový graf inicializácii</u>

- vyjadruje prípustné stacy prechodov medzi postupnosťou následných inicializácii
- umožňuje vypočítať L_{min}
- vrcholy grafu zobrazujú v danom časovom okamihu všetky možné nasledujúce inicializácie, pričom jeho hrany zobrazujú počet kroko medzi dvoma in icializáciami v čase t a t + d
- vrcholy grafu sú označovné vektorom konfliktov

Vektor C^0

Ak vektor $\mathbf{S^{i-1}}$ vyjadrený v tvare

$$S^{i-1}=SL1(C^{i-1}).0$$

reprezentuje posunutý vektor konfliktov $\mathbf{C^{i-1}}$ o jeden bit vľavo, potom v **i.** kroku zreťazenia pre $\mathbf{C^i}$ platí

$$C_j^i=\begin{cases} S^{i-1} \Leftarrow S^{i-1}(1)=1 \\ S^{i-1} \Leftarrow S^{i-1}(1)=0 \ \ldots \text{ inicializácia neprípustná} \\ S^{i-1} \vee C^0 \Leftarrow S^{i-1}(1)=0 \text{ inicializácia prípustná} \end{cases}$$

kde $\mathbf{C^0}$ je začiatočný vektor konfliktov, kt. zobrazuje prípustnosť nasledujúcich inicializácií po prvej inicializácii prúdového spracovania, t.j. vyjadruje všetky predtým začaté inicializácie v procese zreťazenia.

SGI má zásadný význam pri syntéze zreťazeného systému, prezože pre zadaný rozklad zreťazenia

poskytuje informáciu o všetkých kombináciách inicilizácií v procese zreťazeného spravoania, na základe ktorého sa uskutočňije výber optimálnej synchronizácie riadenia vstupného prúdu údajov v navrhovanom zreťazenom systéme.

Vektor konfliktov

Riadenie

$$C^i=\left(c_1^i c_2^i ... c_n^i\right);\ c_j^i \in \{0,1\}$$

ktorý zobrazuje informáciu o prípustnosti novej inicializácie v **i.** kroku zreťazenia (i = 1, 2, ..., n) prostredníctvom nasledujúcej interpretácie jeho prvkov

$$c_j^i = \begin{cases} 1 \ \ldots \text{ inicializácia neprípustná} \\ 0 \ \ldots \text{inicializácia prípustná} \end{cases}$$

kde $\mathbf{j \in \{1, 2, ..., n\}}$.

- vykonáva riadiaca jednotka zreťazenia, kt hlavnou úlohou je
  - riadenie a synchronizácia vkonávania funkcií jednotlivých stupňov zreťazenoa (riadenie stupňov zreťazenia)
  - riadenie a synchronizácua začiatkov inicializácii prúdového spracovania na vstupe PFJ
- vychádza zo všeobecnej schémy riadenia stupňa zreťazenia dekomponovaného na operačné a pamäťové obvody
- vnútrostupňové riadenia zahrňuje
  - výber funkcie stupňa (F)
  - aktiváciu logiky pamäťových obvodou (A)
  - synchronizáciu pamäťových obvodou (H)
  - stav stupňa zreťazenia (S) v priebehu jeho aktivácie

Riadiaca jednotka zreťazenia

- riadfenie začiatkov inicializácií vstupného prúdu údajov zreťazeneho systému uskutočňuje jeho RJZ prostredníctvom synchronizačných hodinových signálov CLK a synchronizačných signálov inicializacii SYN
- jeho základným komponentom je posuvný register ST nastavený na počiatočnú hodnotu vektora konfliktov C^0

Hodnotenie výkonnosti zreťazeného systému

Priepustnosť zreťazenia

- priepustnosť zreťazeného systému predstavuje primeraný počet inicializácii prúdového spracovania pripadajúci na periódu T_c strojového cyklu, t.j. na jeden krok zreťazenia. Perióda SC v zreťazenom systéme sa nazýva cyklus zreťazenia.
- je def počtom N úloh prúdového spracovania inicializovaných v priebehu *n* cyklov zreťazenia, z kt každy je vykonaný za periódu T_C

$$\mathbf{P = N / n = 1 / L_{min}}$$

Vyťažiteľnosť stupňov zreťazeného systému

- def pomerom počtu využití N_v jednotlivých stupňov zreťazenia pre daný cyklus pripustných latentnosí a celkového počtu cyklov zreťazenia N_Z pripadajúcich na všetky stupne N_s zrežazeného systému
- vyťažiteľnosť stupňov zreťazenia *V* je pre daný piemerný cyklus latentnosti percentuálne vyjadená vzťahom

$$V = (N_v / (n \times N_s)) \, 100\%$$

Spravidla opatrenia na zvyšovanie hodnoty priepustnosti zreťazenia spôsobujú I zvyšovania hodnoty vyťažiteľnostu stupňov zreťazeného systému.

Vektorizácia prúdového spracovania skalárnych inštrukcii programu

Tieto prístupy sú ilustrované na príklade rozkladu progeamu cyklu vo verziách s aplikovaním sklárnych operácii a s aplikovaním vektorových operácii.

Rozklad programu SI

Rozklad programu na báze skalárnych inštrukcií (SI).

- Formát 64 bitovej skalárnej inštrukcie FSI je:

$$\langle FSI \rangle ::= \langle KSO(1..8) \rangle \langle R1(1..6) \rangle \langle M[R2(1..50)] \rangle$$

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

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

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

Skalárne inštrukcie programu	Sémantika inštrukcií
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)

Rozklad programu na báze vektorových inštrukcií (VI).

- Formát 64 bitovej vektorovej inštrukcie FVI je:

$$\langle FVI \rangle ::= \langle KVO(1..8) \rangle \langle VR1(1..8) \rangle \langle M[VR2(1..50)] \rangle$$

kde
KVO je kód vektorovej operácie.
$VR1$ je adresa prvého vektorového operanda v registrovej pamäti definovaná obsahom vektorového registra (VR1).
$VR2$ je adresa druhého vektorového operanda v pamäti M definovaná obsahom vektorového registra VR2 v tvare:

$$(VR2) = (B2) + disp2$$

kde
$B2$ je bázový register druhého operanda v tvare vektora,
$disp2$ je posuv (*angl.* displacement) druhého operanda v tvare vektora.

Vektorové inštrukcie programu	Sémantika inštrukcií
VLD VR1, M [VR2]	VR1 ← M [VR2] ≡ VR1 ← M [EX]
VMUL VR1, M [VR2]	VR1 ← VR1 × M [VR2] ≡ VR1 × M [EY]
VADD VR1, M [VR2]	VR1 ← VR1 + M [VR2] ≡ VR2 + M [EZ]
VST VR1, M [VR2]	M [VR2] ← VR1 ≡ M [EZ] ← VR1

i-ta iterácia cyklu pozostáva z nasledujúcich inštrukcií:

```
C:   R(i) := M[X(i)]
     R(i) := R(i) × M [Y(i)]
     R(i) := R(i) + M [Z(i)]
     M[Z(i)] := R(i)
     IF i < N THEN i := i + 1
                 goto C
     ENDIF
```

Po rozvinutí cyklu jeho inštrukcie pre i = 8 sú:

R(1) := M [X(1)]	R(2) := M [X(2)]	. . .	R(8) := M [X(8)]
R(1) := R(1) × M[Y(1)]	R(2) := R(2) × M [Y(2)]	. . .	R(8) := R(8) × M [Y(8)]
R(1) := R(1) + M[Z(1)]	R(2) := R(2) + M [Z(2)]	. . .	R(8) := R(8) + M [Z(8)]
M [Z(1)] := R(1)	M [Z(2)] := R(2)	. . .	M [Z(8)] := R(8)

Po rozvinutí cyklu nie je potrebné použiť inštrukciu podmieneného skoku. Každý riadok definuje jedna vektorová inštrukcia, ktorá pozostáva z N = 8 skalárnych inštrukcií. V danom príklade sú to vektorové inštrukcie typu:

VLD zavedenie vektora operandov z pamäte

VMUL vektorové násobenie

VADD vektorové sčítanie

VST odpamätanie výsledku vektorovej operácie do pamäte

Koeficient účinnosti vektorizácie je definovaný v tvare

$$\frac{T_S}{T_V} = \frac{T_S}{\frac{r}{\mu} T_S + (1 - r)\, T_S} = \frac{1}{1 + \frac{r}{\mu} - r} = \frac{1}{1 - r\left(1 - \frac{1}{\mu}\right)}$$

Príklad

5 skalárnych operácií (6SC / operácia)

- nevektorizovateľná časť programu,

1 operácia 10-násobného cyklu typu DO (25SC / telo cyklu)

- vektorizovateľná časť programu,

Riešenie

Doba vykonania programu

$$T_s = 5 \cdot 6 + 10 \cdot 25 = 280\ [SC]$$

odkiaľ koeficient vektorizácie $r = (280 - 5 \cdot 6) / 280 = 0{,}89$.

Princíp

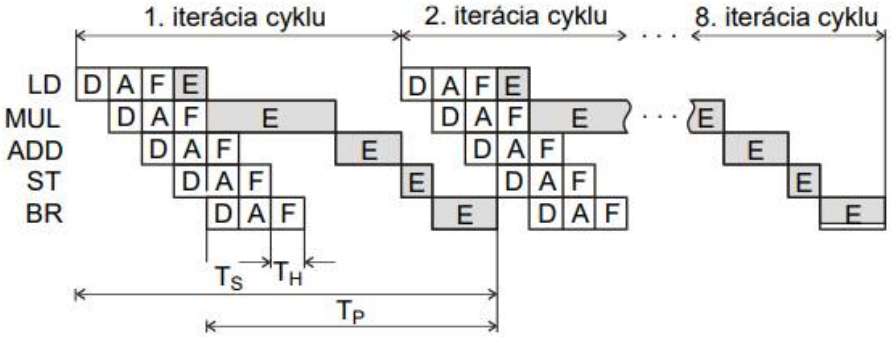
Výpočet koeficientu účinnosti vektorizácie pre dve hodnoty účinnosti vektorového spracovania:

Pre $\mu = 3{,}06$

$$\frac{T_S}{T_V} = \frac{1}{1 - 0{,}89\left(1 - \frac{1}{3{,}06}\right)} \cong 2{,}49$$

Pre $\mu = 4{,}20$

$$\frac{T_S}{T_V} = \frac{1}{1 - 0{,}89\left(1 - \frac{1}{4{,}20}\right)} \cong 3{,}10$$



T_H - hodinová perióda
T_S - perióda jednotlivých iterácií cyklu programu (5 inštrukcií) pri ich sériovom vykonávaní (vnútorné zreťazenie)
T_P - perióda jednotlivých iterácií cyklu programu (5 inštrukcií) pri ich prúdovom vykonávaní (vonkajšie zreťazenie)

Účinnosť prúdového spracovania cyklu

Doba vykonania *m* iterácii cyklu pri prúdovom spracovaní jeho inštrukcii je ak

$$T_{PS} = m \cdot T_S \qquad (1)$$

jednotlivé iterácie sa výkonávajú sériovo a

$$T_{PP} = T_S + (m - 1) \cdot T_P \qquad (2)$$

Zo vzťahov (1) a (2) vyplýva nasledujúci vzťah pre účinnosť prúdového spracovania

$$\mu = \lim_{m \to \infty} \frac{mT_S}{T_S + (m-1)T_P} = \frac{T_S}{T_P} \quad \text{(teoretická hodnota)}$$

Príklad stavový graf inicializácii

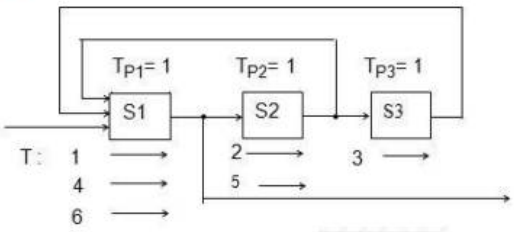
Nech $m = 8, T_S = 13T_H, T_P = 10T_H$

Riešenie

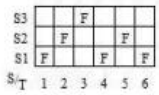
Reálna účinnosť sa rovná

$$\mu = \lim_{m \to \infty} \frac{mT_S}{T_S + (m-1)T_P} = \frac{8 \cdot 13T_H}{13T_H + (8-1) \cdot 10T_H} \cong 1{,}25 < \frac{13T_H}{10T_H} = 1{,}3$$

*Zostavte **stavový graf inicializácií** (SGI) zreťazeného spracovania výpočtového procesu v PFJ s tromi stupňami zreťazenia S1, S2, S3, definovaného nasledujúcim prepojením funkčných stupňov a určte optimálnu postupnosť prípustných latencií.*

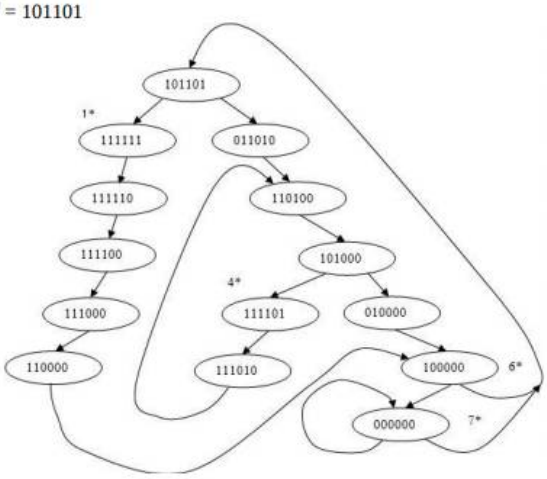


Stupne zreťazenia
S ∈ {S1, S2, S3}
k = 3



Riešenie:

Vektor konfliktov $C^0 = 101101$



Obr.1. Stavový graf inicializácií s tromi stupňami zreťazenia

Prípustné latentnosti $L_1 = 1$, $L_2 = 4$, $L_3 = 6$.

Cykly prípustných latentností, vyplývajúcich z hodnôt prípustných latentností L_1, L_2, L_3 analyzovaného grafu inicializácií a ich priemerné hodnoty sú:

$L^1 = (1,6)$ $L^1_S = 3,5$
$L^2 = (4,6)$ $L^2_S = 5$
$L^3 = (4,6)$ $L^2_S = 5$
$L^3 = (4,\{4\},6)$ $L^3_S = 4.$

Optimálna postupnosť latencií, zabezpečujúca maximálnu priepustnosť systému je definovaná minimálnou priemernou hodnotou a pre tento prípad je $L^1 = (1,6)$.

Prepojovacie siete – základné vlastnosti a použitie

Komunikačné modely

- P-M (Processor – Memory)
- P-P (Processor – Processor)

Prepojovacie systémy

- jednoduchá časovo prideľovaná zbernica / jednoduchá časovo zdeľaná zbernica / jednoduchá zbernica
  - v nenáročných zostavách paralelných PC systémoch s malým počtom prepájaných jednotiek, najmä vtedy, keď medzi mimi nepožaduje súčasné vykonanie komunikácií
- viacnásobné zbernice
  - v konvenčným multiprocesorových systémoch, v kt sa prepojenie z viacerých zberníc uskutočňuje prostredníctvom viacprístupových jednotiek
  - zbernicoe systémy sa obyčajne používaju v modeloch typ P-M
- prepojovacie siete / prepojovacia sieť
  - obvodový prostriedok PS[NxM]
    - N vstupov $X_0, X_1, \dots, X_{N-1}$

        - M výstupov $Y_0, Y_1, \dots, Y_{M-1}$
        - prepojenie N komponentov s M komponentami – komunikácia typu P – M
        - vzájomné prepojenie N komponentov – komunikácia typu P - P
    - delenie poďla typu prepojenia
        - jendostranné PS
        - obojstranné PS
    - delenie podľa smerovania komunikácii
        - jednosmerné PS
        - obojsmerné PS
    - prepojovacia funkcia

$$F: X \rightarrow Y$$

kde

X – množina vstupov $\{X_0, X_1, \dots, X_{N-1}\}$

Y – množina výstupov $\{Y_0, Y_1, \dots, Y_{N-1}\}$

- štvorcové prepojovacie siete
    - prepojovacie siete v ktorých platí $N = M$
- vyžadovaný stav siete
    - súhrn dvojíc vstupov a výstupv v PS, medzi ktorými majú byť vytvorené vodivé cesty
- skutočný stav siete
    - vyjadruje reálnu nastaviteľnosŤ PS v danom okamihu, kt ale nemusí zodpovedať vyžadovaný stav siete
- podľa vzájomného vzťahu týchtvo dvoch stavov sa rozlišújú
    - blokujúce prepojovacie siete
        - existuje v nich aspoň jedno prepojenie medzi vstupom a výstupom, kt nemôže byť realitzované, pretože je blokované iným vyžýadovaným prepojením
    - neblokujúce prepojovacie siete
        - môžu sa v nich vytvoriť prepojenia medzi ľubovoľnými dvojičami vstupov a výstupov
    - prestaviteľné siete
        - reprezentujú osobitný prípad blokovacích sieti, v kt sa dočasne zablokuje vyžadované prepojenie
        - prestavením niektorého z prepojení podľa inej vodivej cesty je možné dočasne zablokovaé prepojenie uskutočniť
        - umožňuje pre ľubovoľnú množinu realizovaných prepojení uskutočniť prepojenie každej dvojice voľného vstupu a voľného výstupu
    - údajový manipulator
        - osobitná skupina PS uplatňovaných v architekúrach typu MIMD
        - umožńujú jeden vstup prepojiť s viacerými výsupmi
    - permutačne siete
        - jednostupňové
        - viacstupňové

Prepojovacia funkcia

$$f_i = t_i(p) = (p_{n-1} \dots p_1 p_0)$$

$$p_i \in \{0,1\}; p \in \{(j)_2, (k)_2\}$$

$$n = log_2(N); i = 0,1, \dots, n-1$$

kde

t_i – typ prepojovacej funkcie

p – binárna reprezentácia vstupnej adresy (j) a výstupnej adresy (k)

i – počet funkcií, definujúcich PS

Základné typy permutačných operácii

- dokonalé premiešanie

$$shuffle(p_{n-1}p_{n-2} \dots p_1 p_0) = p_{n-2}p_{n-3} \dots p_0 p_{n-1}$$

- inverzné premiešanie

$$shuffle^{-1}(p_{n-1}p_{n-2} \dots p_1 p_0) = p_0 p_{n-1} \dots p_2 p_1$$

- výmena

$$exchange(p_{n-1}p_{n-2} \dots p_1 p_0) = p_{n-1}p_{n-2} \dots p_1 \overline{p_0}$$
$$exchange = cube_0$$

- preklopenie

$$reversal(p_{n-1}p_{n-2} \dots p_1 p_0) = p_0 p_1 \dots p_{n-2}p_{n-1}$$

- motýliková operácia

$$butterfly^k(p_{n-1}p_{n-2} \dots p_1 p_0) = p_{n-1} \dots p_{k+1} p_0 p_{k-1} \dots p_1 p_k$$

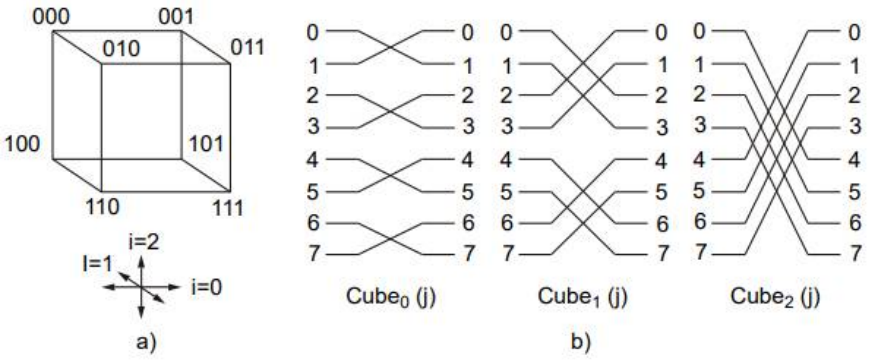
pre

$$1 \le k < n$$

Jednostupňová prepojovacia sieť

- N-dimenzionálna kocka

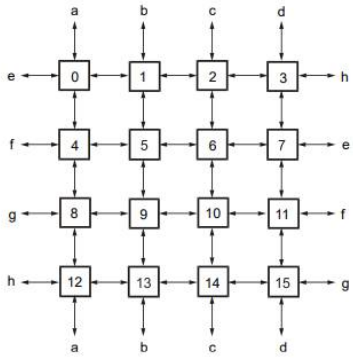
$$cube_i(p_{n-1} \dots p_1 p_0) = p_{n-1} \dots p_{i+1} \overline{p_i} p_{i-1} \dots p_0$$



a – kubická sieť,
b – realizácia prepojovacích funkcií $cube_0(j)$,

- štvorcová mreža

$$grid_{+1}(j) = (j+1)\ mod\ N$$
$$grid_{-1}(j) = (j-1)\ mod\ N$$
$$grid_{+n}(j) = (j+n)\ mod\ N$$
$$grid_{-n}(j) = (j-n)\ mod\ N$$



<u>Viacsutpňové prepojovacie siete</u>
Realizácia ako PMS

$$P_N = \alpha E \alpha E \dots \alpha E = (\alpha E)^m$$

kde

P_N množina permutácií pre PS[N×N]
α typ permutácie
E riadená sieť typu výmena
m počet stupňov

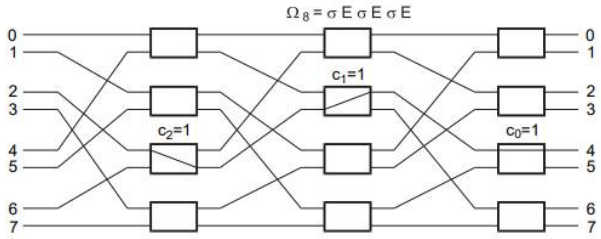
<u>Omega sieť</u>

Omega (Ω_N) sieť

$$\Omega_N = \sigma_{n-1} E \sigma_{n-1} E \dots \sigma_{n-1} E = (\sigma_{n-1} E)^n$$

kde

σ_{n-1} je dokonalé premiešanie
E riadená sieť typu výmena



Kubická sieť

Kubická (C_N) sieť

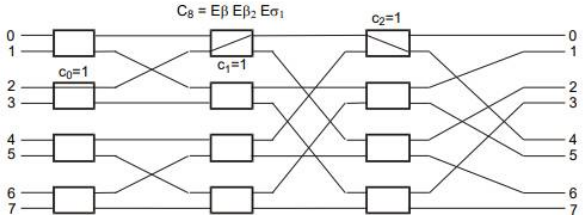
$$C_N = E\beta_1 E\beta_2 \dots E\beta_{n-2} E\sigma_1$$

kde

β_i je i-ta motýliková operácia

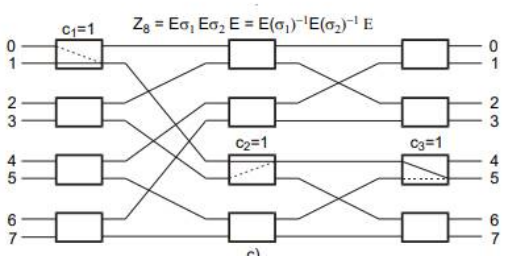
σ_1 je inverzné dokonalé premiešanie

E riadená sieť typu výmena



Zakladná sieť

$$Z_N = E\sigma_1 E\sigma_2 \dots E\sigma_{n-1} = E(\sigma_{n-1})^{-1} E(\sigma_{n-2})^{-1} \dots E(\sigma_1)^{-1} E$$



$$F_N = E\sigma_1 E\sigma_1 \dots E\sigma_1 = (E\sigma_1)^n$$

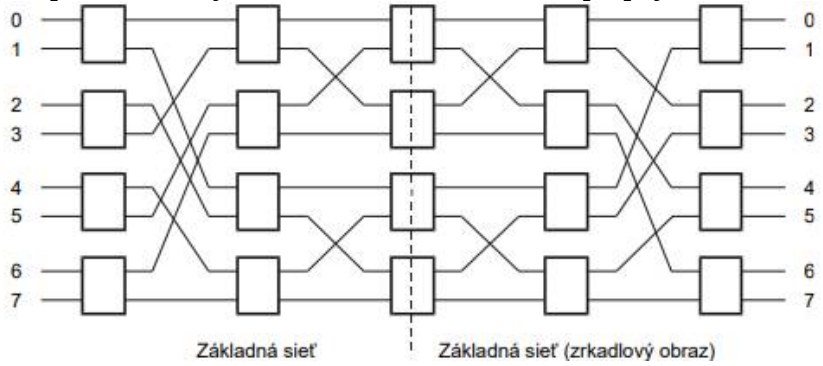
R sieť kde

$$R_N = Z_N \times \rho$$

σ_1 je inverzné dokonalé premiešanie

E riadená sieť typu výmena

- úplná SE (suffle/exchange) sieť
- má schopnosť realizovať všetky permutácie z N prvkov
- z jedného vstupu Benešovej siete možno definovať N/2 prepojovacích sieti



Údajové manipulátory / údajový manipulator

- požiadavky na SIMD a MIMD sú rôzne
- SIMD
  - synchrónne architektúry
  - manipulovanie s údajmi
  - permutačné siete
- MIMD
  - dôraz sa kladie na optimálne prepojenie bez ohľadu na momentálny stav siete
  - údajové manipulátory
    - vo všeobecnosti ide o operácie pre manipuláciu s údajmi def manipulačnými funkciami, ktoré sa rozdeľujú do nasledujúcich tried
      - permutácie
      - kopírovania
      - rozmiestnenie
      - maskovanie
    - sieťový prostriedok na realizáciu manipulačných funkcií sa nazvýja udajový manipulátor
    - udajový manipulátor je m stupňová prepojovacia sieť M[NxN] vytvorená na báze prepínacích elementov s troma výstupnými linkami

Údajový manipulátor

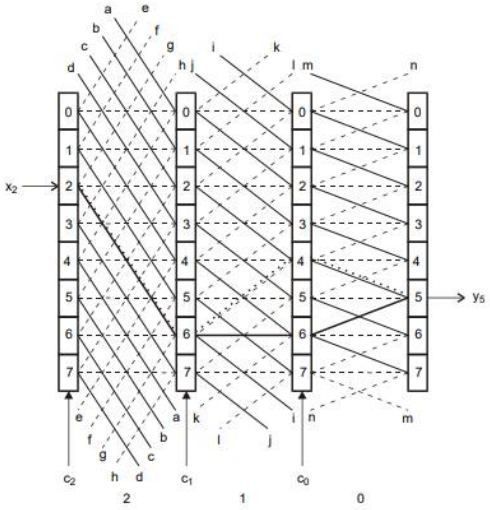
$$M2_{-1}(j) = (j - 2^i) \bmod N$$

$$I(j) = j$$

Údajový manipulátor

(... pokračovanie)

- Prepínač vyberá jednu zo svojich troch vstupných liniek a v súlade s prepínacou funkciou ju prepojí s jednou zo svojich výstupných liniek alebo súčasne s dvoma, resp. tromi výstupnými linkami vtedy, keď vykonáva funkciu kopírovania.



Úvod do ILP procesorov

Výkonnosť

- zvyšovanie pracovnej frekvencie
- funkcionálne vylepšenie komponentov

Funkcionálne vyplepšenie

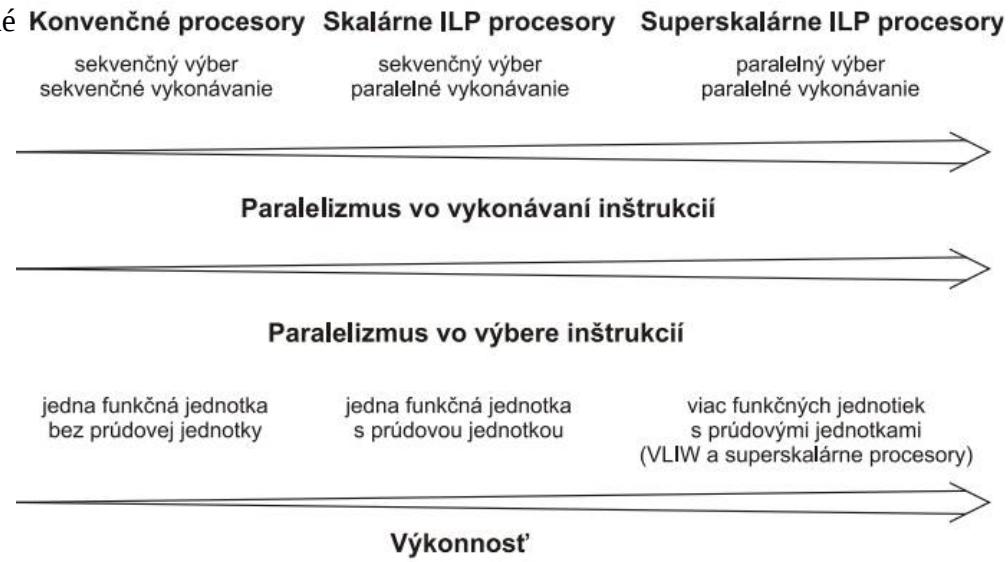
- výber inštrukcie
- vykonanie inštrukcie

Vývoj

- konvenčné
- skalárne ILP procesory
- superskalárne ILP procesory
- VLIW

História

Sekvenčné



Obr. 7.1 Vývoj ILP procesorov

- výber inštrukcie
- vykonávanie inštrukcie

Paralelné vykonávanie

- zvyšovanie poštu funkčných jednotiek
- prúdové spracovanie

Paralelizmy v ILP procesoroch

- „priestorový paralelizmus
  - zvyšovanie počtu funkčných jednotiek
- „časový“ paralelizmus
  - prúdové spracovanie

Časový vs priestorový paralelizmus

ILP

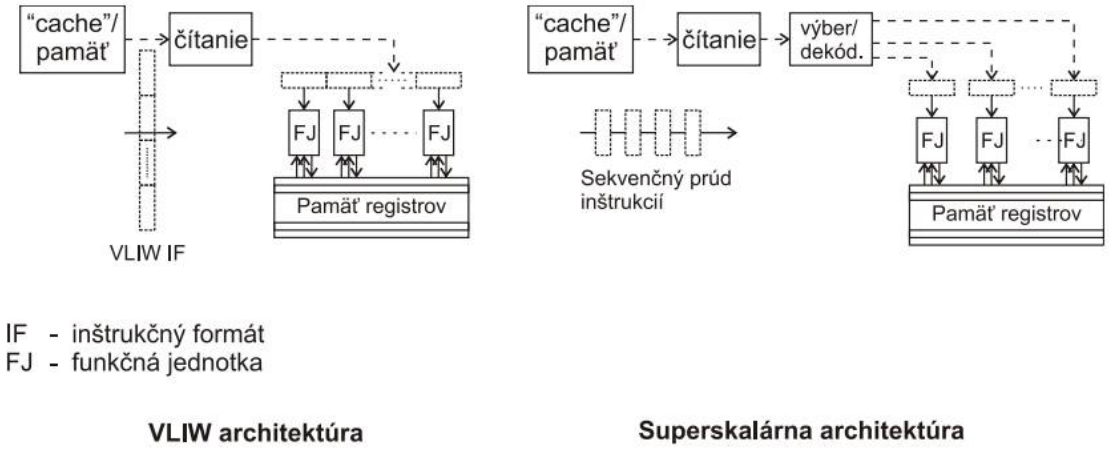
	Prúdové spracovanie	Zvyšovanie počtu funkčných jednotiek
Vektorové procesory	+	
Systolické polia	+	+
SIMD procesory		+
Asociatívne procesory		+
ILP procesory	+	
VLIW procesory	+	+
Superskalárne procesory	+	+
Viacvláknové architektúry	+	+
Multiprocesory		+
Multipočítače	+	+

- skalárne procesory
  - vo všeobecnosti každú jednoduchú inštukciu dekomponovanujú na jednotlivé fázy (strojové cykly) prúdového spracovania
- superskalárne procesory
  - v jednom strojovom cykle prúdového spracovania vykonávajú viacnásobné inštrukcie
  - charakteristickou črtou superskalárneho prístupu vykonani inštrukcii je súbežné

spracovanie a ukončenie viacerých prúdov inštrukcií, v dôsledku čoho sa architektúra superskalárneho procesora nazýva multiprúdová

- VLIW procesory
  - predstavujú kombináciu dvoch koncepcii spracovania inštrukcii, ktorými sú
    - horizontálne mikroprogramovanie
    - supersklárne spracovanie inštrukcii

VLIW vs SSP architektúra



Obr. 7.4 Rozdiely medzi VLIW a superskalárnymi procesormi

- v riadiacej časti – na úrovni inštrukcií
- vo vykonávacej časti – na úrovni operácii

Inštručný paralelizmus

- prúdové spracovanie inštrukcií – dekompozícia spracovania inštrukcií na dielčie fázy
- prúdové spracoanie operácií – dekompozícia spracovania operácii na atomické operácie

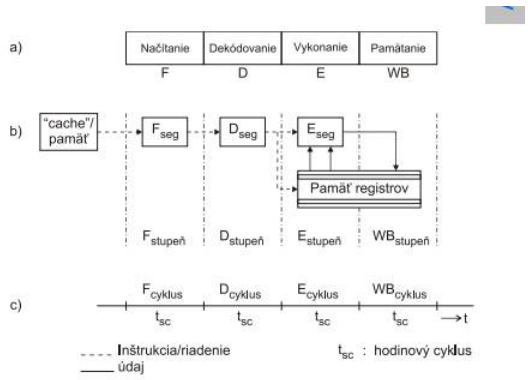
Paralelizmus v skalárnych procesoroch

Predpoklady

- inštrukčný cyklus je rozdelený na dielčie fázy, kt sa môžu kokurenčne vykonať s inými fázami predchadzajúcich inštrukcii
- každa fáza je vykonateľná v priebehu jedného alebo viacerých strojových cyklov
- jednotlivým fázam vykonávania zodpovedajú na úrovni obvodového riešenia prístlušne stupe zreťazenie

Systém zreťazenia

- Stupeň F
  - Načítava inštrukcie z pamäte
- Stupeň D
  - Dekóduje inštrukcie
- Stupeň E
  - Vykonáva operáciu definovanú v IF KO
- Stupeň WB
  - Zapisuje výsledok operácie do registra/pamäte



Obr. 7.5 Stupne systému zreťazenia

a – Stupne systému zreťazenia pre register-register architektúru,
b – Radenie segmentov prúdovej funkčnej jednotky,
c – Časovanie

Skalárne procesory

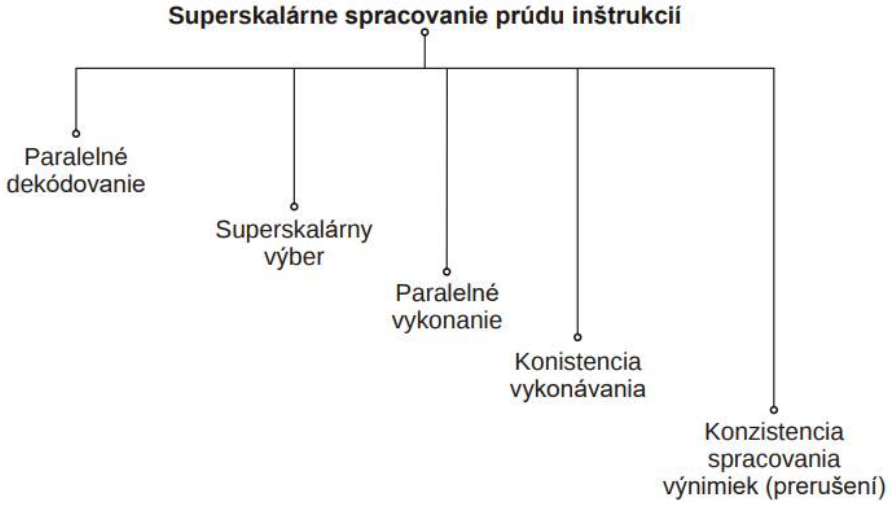
Architektonické riešenie

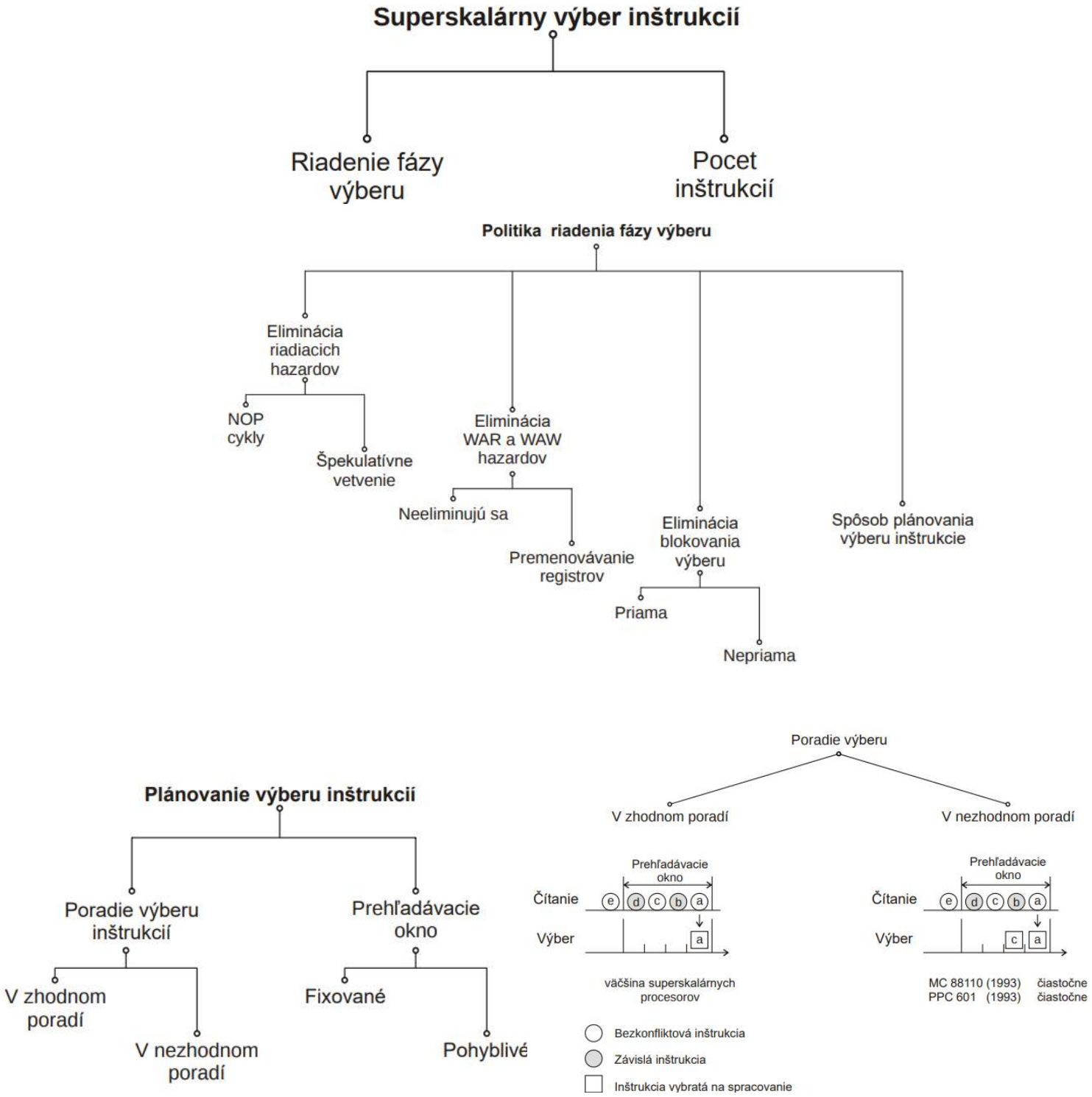
- CISC
  - funkčne jednotky na vykonávanie operácií s pervnou rádovou čiarkou, resp, iných typov neskalárnych operácií sú realizované osobitým koprocesorom pripojeným ku čipu procesora
  - implementovaný do mikroprocesorov od firiem: DEC, Intel (4004, 8080, 8086, 80386, DX4) , Motorola, Zilog, National Semiconductor
  - implementovaný do minipočítačov: DEC PDP-11, LSI-11, VAX 8600
- RISC
  - počítač s redukovanou sadou inštrukcii
  - viacnásobné funkčné jednotky na vykonanie operácii s pevnou radovou čiarkou, celočíselne operácie a podobne
  - integroavné na čipe procesora
  - vyvíjajú ako výpočtové prostriedky na vykonácanie programov napísaných v jazykoch vykoskej úrovne HLL
  - RISC I a RISC II
    - vývoj na Kalinfornskej univerzite v Berkley
    - char. črty architektonického riešenia
      - aplikácia registrovaných okien
      - 3-stupňová PFJ
      - vyrovnávacia pamäť pre skokové inštrukcie
      - oneskorovanie realizácie vetvenia
  - MIPS (Microprocessor without Inerlocked Piped Stages)
    - vývoj na Standfordskej univerzite
    - char. črty architektonického riešenia
      - 5-stupňová PFJ
      - 31 inštrukcii
      - obmedzený počet registov
      - kompilátor rieši otázku hazardov
  - SPARC (Scalable Procesor ARChitecture)
    - vývoj v Sun Microsystems
    - char. Črty
      - nahradenie viaccyklových inštrukcií jendocyklovými
      - použitie registrovaných okien
      - pr:
        - SPARC version 7
        - UltraSPARC-II
        - SPARC M8
  - JavaChips
    - picoJava-I
      - 4-stupňová PFJ
      - stupne PFJ sú F – D – E - WB
    - picoJava-II
      - optimalizované hw prostredie pre JVM
      - hw implementácia nietorých JVM Inštrukcii
      - podpora iných jazykov vyššej úrovne
      - zásobníková architektúra
      - inštukčná sada: cca 300 inštrukcii
    - microJava 701

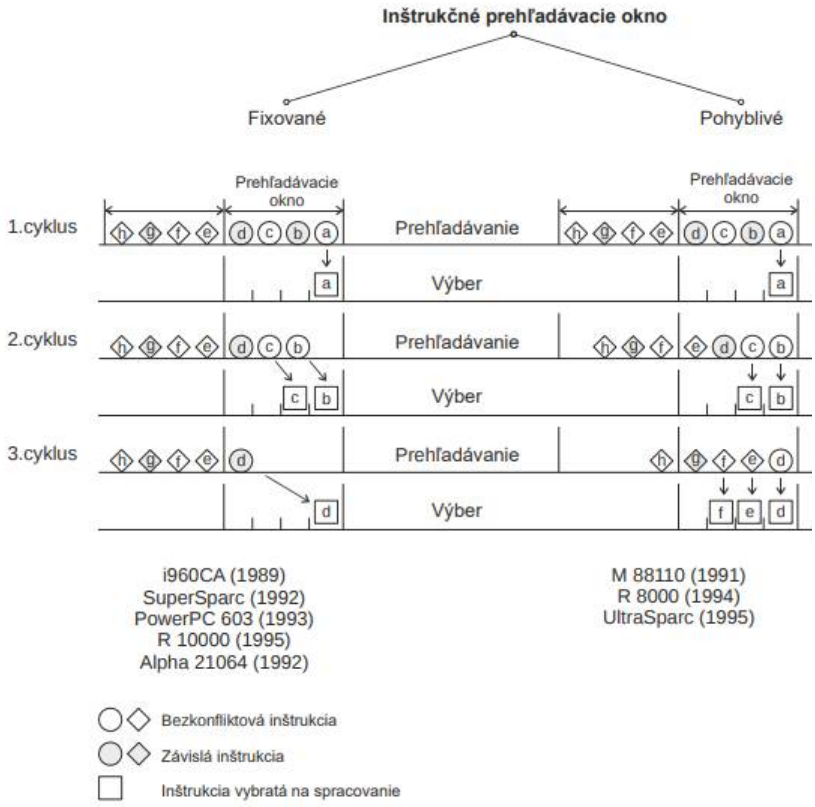
- implementácia 32b picoJava-II architektúry
- 200MHz, 2.8 m tranzistors
- 6-stupňová PFJ
- logika prekrývania 4 inštrukcie
- 0-16KB I-cahce pamäť
- 0-16KB D-cahce pamäť
- 2 x 32 b zbernica medzi D-cache a zásobníkovou pamäťou
- 3 x 32 b zbernica medzi jednotku celočisených operácií a FPU
- komponenty
  - I-cache
  - D-cache
  - Logika prekrývania (IFU)
  - Jednotka celočísených operácii (IU)
  - jednotha pre práciu s PHRČ (FPU)
  - zásobníková pamäť (Stack Cache)
  - V/V zbernicka a jednotka pamäťového rozhrania (Bus Interface Unit)
- prudová dunkčna jednotka – 6 stupňová
  - Fetch – čítanie inštrukcii z I cahce alebo z pamäte
  - Decode – prekrývanie inštrukcie a ich dekódovanie
  - Register – načítanie operandov zo zásobníka
  - Execute – vykonávanie inštrukcií
  - Cache – čítanie / zápis výsledkov z / do D cache
  - WriteBack – zápis výsledkov do zásobníka
  -
    - UltraJava
- zásobníkovo orientované architektúry

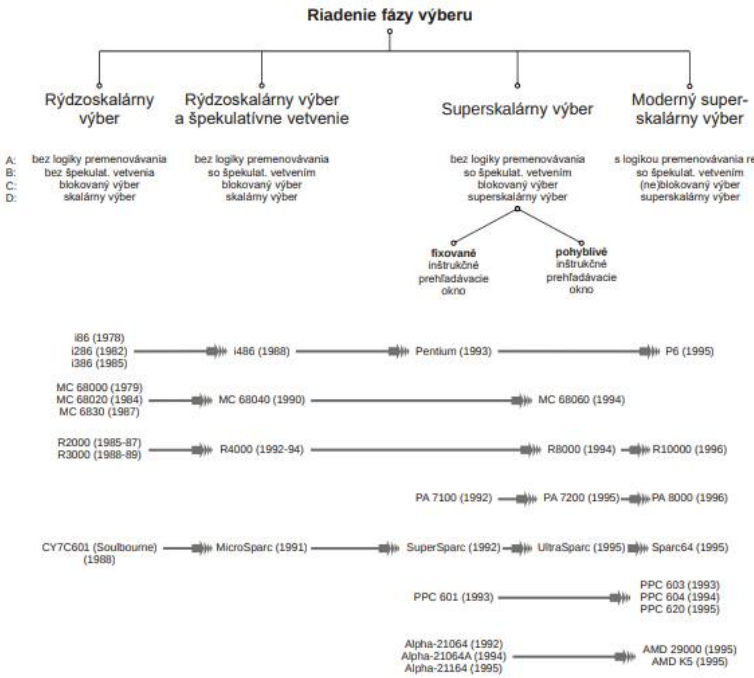
<u>Superskalárne procesory / superskalarne procesory</u>

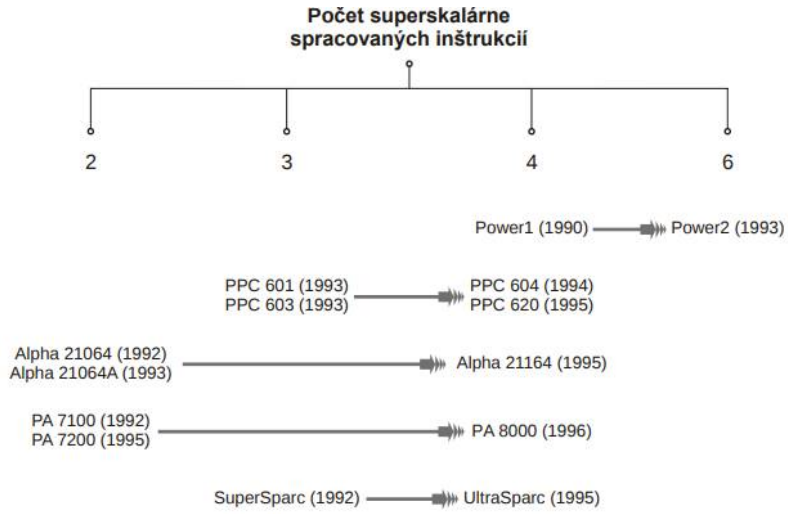
- Intel – Intel i960 → Pentium III
- IBM – PowerPC 601 → Power4
- AMD – AMD 2900 → AMD-K6
- DEC – Alpha 21x64
- SUN – UltraSPARC
- MIPS – R8000 → R16000




Superskalárny výber inštrukcií
Riadenie fázy výberu
Pocet inštrukcií
Politika riadenia fázy výberu
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 inštrukcií
V zhodnom poradí
V nezhodnom poradí
Prehľadávacie okno
Fixované
Pohyblivé
Poradie výberu
V zhodnom poradí
V nezhodnom poradí
Prehľadávacie okno
Čítanie
Výber
e d c b a
a
väčšina superskalárnych procesorov
Prehľadávacie okno
Čítanie
Výber
e d c b a
c a
MC 88110 (1993) čiastočne
PPC 601 (1993) čiastočne
Bezkonfliktová inštrukcia
Závislá inštrukcia
Inštrukcia vybratá na spracovanie







<u>Oddelenie fázy výberu inštukcii</u>

Prečo?

- Eliminácia vlokovania výberu inštrukcií v dôsledku údajových a zdrojových hazardov

Aspetky oddelenia fázy výberu

- trieda inštrukcii
  - planovací zásobník (P-zásobník) je definovaný len pre niektoré typy inštrukcii
    - Power 1
    - Powe 2
  - planaovací zásobník je definovaný pre všetky typy inštrukci
    - PPC 603
    - Pentiu Pro
- spôsob realizácie plánovacieho zásobníka
  - v závislosti od typu

        - planovací zásobník
            - špecialitovaný
                - Power 1
                - AMD K5
            - skupinový
                - Power 2
                - R10000
            - centralizovaný
                - Pentium Pro
        - Kombinovaný zasobník
    - v závislosti od kapacity
        - špecializovaný
            - CDC66000
        - skupinový
            - R10000
        - centralizovaný
            - Pentium Pro
    - v závislosti od počtu vstupných a výstupných hradiel
    - v závislosti od typu P zásobníka
- politika riadenia fázy načítania operandov
    - vo fáze výberu inštrukcie
        - planovací zásobnik obsahuje hodnoty zdrojových registrov
        - Power PC 620, AMD K5, Pentium Pro
    - vo fáze odoslania inštrukcie
        - plánovací zásobník obsahuje indexy zdrojových registrov
        - CDC6600, Nx586, PA8000, R10000
- načítanie operandov vo fáze V / načítanie operandov vo fáze výberu
    - tak pamäť registrov ako aj zásobník obsahuje informáciu platnosti obsahu príslušného zdrojového registra
- načítanie operandov vo fáze O / načítanie operandov vo fáze odoslania
    - len pamäť registrov obsahuje informáciu o platnosti obsahu prislúšneho zdrojového registra

<u>Superskalárna architektúra</u>

- viac PFJ a viac VJ
- spracovanie inštrukcii v zhodnom alebo nezhodnom poradí
    - usporiadaná kompletizácia
    - preusporiadaná kompletizácia

<u>Konzistencia vykonávania / konzistencia superskalárneho vykonávania inštrukcií</u>

Procesorová konzistencia

- poradie kompetizácie inštrukcií
- slabá procesorová konzistencia
    - preusporiadanie poradai inštrukcii
    - preusporiadaná kompletizácia inŠtrukcii (len pre bezkonfliktový prúd inštrukcií)
    - je nutné detegovať výskyt údajových hazardov
    - pr
        - Power1
        - Power2
        - R8000
- silná procesorová konzistencia
    - usporiadané poradie (v zhodnom poradí)

    - usporiadaná kompletizácia inštrukcií
    - použirie ROB (ReOrderingBuffer) a DRIS (Deffered Scheduling Register Renaming Instruction Sheving)
    - pr
        - Pentiom Pro
        - AMD 29000
        - R10000

Pamäťová konzistencia

- prístup do pamäte
- slabá pamäťová konzistencia
    - preusporiadané poradie inštrukcií
    - peusporiadaný prístup do pamäte (len bez bezkonfliktový prúd inštrukcií)
    - je nutné detegovať výskyt údajových hazardov
    - pr
        - PPC 602-620
        - UltraSparc
        - R10000
- silná pamäťová konzistencia
    - usporiadané poradie (v zhodnom poradí)
    - usporiadaný prístup do pamäte
    - použitie ROBu

<u>Logika preusporiadania</u>

Pamäť preusporiadania (ROB)

- funkcia
    - len preusporadanie
    - preusporiadanie a premenovávanie
    - preusporiadanie a premenovávanie a blokovanie
- veľkosť
    - 2 až 64 slov
- počet naraz zapisovaných / odstránených inštrukcií do / z ROB-u
    - 1 až 4 slová

<u>Konzistencia spracovania výnimiek / konzistencia spracovania prerušenú</u>

Slabá konzostenicia

- nadefinované poradie spracovania výnimiek
- pr: Power 1, Alpha procesory

Silná konzistenica

- def poradie spracovania výnimiek
- pr: Pentium, R10000

<u>Procesory Alpha</u>

DEC (Digital Equipment Corpotation)

- 64 bitový RISC procesor

Alpha 21464

- známy aj ako EV8 alebo Araňa
- charakteristické črty
    - 0.125 mikrom technológia
    - cca 250M tranzistorov
    - integrované pamäťové rozhranie
    - integrocané systémy
    - multiprocesorové rozhanie
    - veľká L2 cahce

- podpora multivláknového spracovania úloh
  - spracovanie 8 inštrukčných prudov
  - spracovanie 4 vlákien súčasnie
  - zdokonalená prerusporiadaná kompletizácia
- zabudovaný smerovač prepojovacej siete
- ccNUMA

Procesory Pentium

Pentium PRO

- charakteristické črty
  - optimalizované 32 bitové vykonávanie operácií
  - integrovaná L2 cache pamäť
  - podpora štvorprocesorocých systémov
  - 15 stupňová prúdvoá jednotka
  - preusporiadanie inštrukcii
  - špekulatívne vetvenie
  - premenovávanie registrov
  - 64 bitová adreova zbernica

Pentum 4

- architektúra sieťovej prerušovanej komunikácie (NetBurst)
- predradená procesorová sekcia
  - sledovacia pamäť cache
    - nekonvenčný prístup riešenia pamäte inštrukcií I-cache L1, kt na úrovni inštrukcii vysiela do procesorovej sekcie na ich preusrporiadané vykonanie
  - pamäť ROM mikrokódu
    - je využívaná na vykonanie zložitých inštrukcií IA-32, každu z kt reprezentuje príslušný mikrokód a realizáciu ktorého inicializuje slodovacia pamôať po identifikácii týchto inštrukcii
  - vyrovnávacia pamäť preložených adries inštrukcii (I-TLB)
    - riadí sprístupňovanie lineárnej postupnosti inštrukcii programu: ak čítanie inštrukcie zo sledovacej pamäte nie je úspešne čitanie inŠtrukcie sa uskutočňuje z pamäte L2
  - prediktor vetvenia (BTB – Branch Target Buffer)
    - určený na predvýber inštrukcií vetvenia z chache L2 po predchádzajúcej predikcii korektonosti vetvenia na základe vyhodnotenia jeho histórie
  - dekóder inštrukcií
    - IA-32 dekóduje 64 bitové inštrukcie sprístupňované z cache L2, pričom jednoduché inštrukcie dekóduje a následne konvertuje do úrovne mikroinštrukcií a zložité do úrovne mikrokódu prostredníctvom pamäte ROM
  - stará sa o
    - výber inštrukcií
    - dekódovnaie inštrukcií
    - jednotka predvídania vetvenia
      - špekulatívne vetvenie (na základe histórie vetvenia)
      - vyrovnávacia pamäť cieľov vetvenia BTV (Branch Target Buffer)
    - generovanie mikrooperácií
- sekcia preusporiadaného vykonania inštrukcií
  - alokátor
    - prostredníctvom jednoúčelových vyrovnávacích pamätí (ROB) uskutočňuje preusporiadanie, sledovanie do postupností oeprácie, prideľuje zdroje na vykonanie požadovaných mikrooperácii a riadí optimálne vykonanie zreťazenia inštrukčných prúdov, kt vďaka jeho 20 fázovej hĺbe umožňuje súbežne spracúvavať 126

mikrooperácii
    - logika premenovania
        - priradí logické inštukcie IA-32 fyzickej 128-záznamoje registrovej pamäti
    - plánovač mikrooperácii
        - určuje pirpavenosť mikrooperácie k vykonaniu na základe pripravenosti jej vstupnych operandov a prostredníctvom štytoch portov priraďuje vykonávacej jednotke k vykonaniu až šesť mikrooperácii v každom SC zreťazenia
- sekcia celočísených ALJ a FP vykonávacích jednotiek
    - rýchla ALJ → jendnoduché coločiselné operácie (používa sa polovičný strojový cyklus)
    - pomalá ALJ → zložité celočíselné operácie (násobenie: 14SC, delenie: 60SC)
    - ALJ na výpočet adries → LOAD a STORE
    - jednotka PHRČ operácii
        - MMX (Multi Media Extension) operácia
            - SIMD operácie – 64 b registre
        - SSE (Streaming SIMD Extension) operácia
            - SIMD operácie – 128 b registre
        - SSE2 (Streaming SIMD Extension 2) operácia
            - SIMD operácie s dvojnásobnou presnosťou
- sekcia pamäťového podsystému
    - vyrovnávacia pamäť
        - cache L2 s kapacitou 256KB
        - 8-stupňová asociatívna pamäť
        - 128 bajtové bloky
    - systémová zbernica
        - Quad Data Rate 4 x 266 MHZ
        - frekvenčné pásmo: 8.5 GB/s
- prúdová funkčia jednotka procesorov „P“ piatej, šiestej a siedmej generácie
    - Pentium MMX (P5) – 5 stupňov
    - Pentium Pro (P6) – 10 stupňov
    - Pentium III (P6) – 11 stupňov
    - Pentium 4 (NetBurts, P68) – 20 stupňov

<u>AMD Opteron</u>

Charakteristické črty architekúry AMD Opteron

- na báze procesora Athelon
- RISC jadro
- 64 bitová architekúra
- preusporiadanie prúdu inštrukcií
- viacnásobné FJ Pre PHRČ operácie
- podpora odosielania 9 inštrukcii naraz do FJ

<u>VLIW architektúry / VLIW architektury / VLIW architektura</u>

Veľmi dlhé inštrukčné slová (formáty) – 512 b / 256 b / 128 b

Zvyšovanie počtu funkčných jednotiek

Realizácia pamäte registrov

- centralizovaná
- decentralizovaná (charakteristická pre dnešné superskalárne procesory)

Veľký počet vykonávacích jednotiek (5-30)

Dlhé inštrukčné formáty vyžadujú

- viac pamäte
- širšiu pamäťovú zbernicu

Na úrnovni asembleru nie sú „manulálne“ programovateľné

Preferujú statické plánovanie vykonávania inštrukcií
Intel Itanium (P7)

- EPIC (Explicit Parallel Instruction Computing)
  - paralelné plánovanie inštrukcií má na starosti prekladač
  - statické plánovanie → odľahčený HW od komplexného plávnovania
- statické plánovanie
  - HW premenovávanie registroc
  - špekulatívne vetvenie
- charakteristické črty architektúry procesora
  - 128 celočiselných (64 b) registrov
  - 128 registroc pre čísla PHRČ (82 b)
  - 64 jednobitových predikátov
  - 8 „branch“ registov
- pamäťová architektúra
  - L1 I-cache: 16 KB
  - L1 D-cache: 16 KB
  - spoločná L2: 256 KB
  - spoločná L3: 1.5 MB – 24 MB
- zbernicová architektúra
  - McKinley zbernica
  - Monzecito zbernica
  - 2x128 zbernicový cyklus

ILP architektúry

- paralelizmus na úrnovni inštrukcií
- jednorozmernný paralelizmus

TLP architekúty

- paralelizmzus na úrovni vláken a procesov
- strrednozrný a hrubozrnný paralelizmus
- multivláknocvé architektúry procesorov, kt predstavujú významnú implementáciu hrubozrného paraleloizmu, sú reprezentované nasledujúcimi modelmi
  - 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 superskárna verzia
    - simultánna superskárna verzia
  - model VLIW
  - model multuprocesorového čipu

Hrubozrný paralelizmus

- multiprocesorové čipy
  - integruje niekoľko procesorov na ploche čipu
  - symetrické multiprocesory – SMS
  - distribuované multiprocesory so zdieľanou pamäťou – DSM
  - multiprocesory s odovzdávaním sprav - MPP
- multivláknové procesory
  - prekrývy vykonávanie inŠtrukcii rozličných zariadení (vláken) toho istého prúdu inštrukci

Multivláknové architektúry

- paralelné architektúry, kde s ainštrukcie vykonávajú vo vláknach sakalárne, superskalárne, resp podľa princípu VLIW arch

- motiváciou pre ich vznik bola príčina dlhej latencie v pamäti, kde sa rýchlostný rozdeil medzi pamäťovým systémom a procesorom postupne zvyšoval
- viacvláknový prístup ponúkol stratégiu architektur pte toleranciu odozvyx v multiprocesorových systémoch
- modely multivláknových architektur (HW aspekt / hardwearový aspekt / hardwearovy aspekt)
  - po cykloch prekrývaný model (saklárna a superskalárna verzia)
  - blokovo prekrývaný model (skalárna a superskalárna verzia)
  - simultánny multivláknovy model (superskalárna verzia)
  - nanovláknový a mikrovláknový model (blokovo prekrývaná superskalárna a simultánna superskalárna verzia)
  - model VLIW
  - model multiprocesorového čipu
- koncepcia spracovania prúdu podľa inštrukcii v multivláknových architektúrach

Koncepcia spracovania prúdu inštrukcií v multivláknových architektúrach

$$
\begin{aligned}
&(I_{11}, I_{12}, \dots, I_{1k}, \dots, I_{1q_1})_1 \\
&(I_{21}, I_{22}, \dots, I_{2k}, \dots, I_{2q_2})_2 \\
&\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
&(I_{n1}, I_{n2}, \dots, I_{nk}, \dots, I_{nq_n})_m
\end{aligned}
$$

kde

I_{jk} je *k*-ta inštrukcia *j*-ho vlákna, $j \in \{1, 2, \dots, n\}$, $k \in \{1, 2, \dots, n_j\}$,

n je počet vlákien,

q_j je počet inštrukcií v *j*-tom vlákne

m je *m*-ty hodinový cyklus.

- skalárne procesory s cyklickým prekrývaním

Skalárne procesory s cyklickým prekrývaním

$$
\begin{aligned}
&(I_{11})_1, (I_{21})_2, \dots, (I_{n1})_m \\
&(I_{12})_{m+1}, (I_{22})_{m+2}, \dots, (I_{n2})_{2m} \\
&\dots\dots\dots\dots\dots\dots\dots\dots \\
&(I_{1k})_{m\times(k-1)+1}, (I_{2k})_{m\times(k-1)+2}, \dots, (I_{nk})_{mk}
\end{aligned}
$$

kde

I_{jk} je *k*-ta inštrukcia *j*-ho vlákna, $j \in \{1, 2, \dots, n\}$, $k \in \{1, 2, \dots, n_j\}$,

n je počet vlákien,

m je *m*-ty hodinový cyklus.

- skalárne procesory s cyklickým prekrývaním

Skalárne procesory s cyklickým prekrývaním

$$(I_{11})_1, (I_{21})_2, \dots, (I_{n1})_m$$
$$(I_{12})_{m+1}, (I_{22})_{m+2}, \dots, (I_{n2})_{2m}$$
$$\dots\dots\dots\dots\dots\dots\dots\dots$$
$$(I_{1k})_{m\times(k-1)+1}, (I_{2k})_{m\times(k-1)+2}, \dots, (I_{nk})_{mk}$$

kde

I_{jk} je *k*-ta inštrukcia *j*-ho vlákna, $j \in \{1, 2, \dots, n\}$, $k \in \{1, 2, \dots, n_j\}$,

n je počet vlákien,

m je *m*-ty hodinový cyklus.

Skalárne procesory s cyklickým prekrývaním (prvé tri inštrukcie)

$$(I_{11}), (I_{21}), \dots, (I_{n1}), (I_{12}), \dots, (I_{n2}), \dots, (I_{13}), \dots, (I_{n3}), \dots$$

Skalárne procesory s blokovým prekrývaním (prvé tri inštrukcie)

$$(I_{11}), (I_{12}), \dots, (I_{1n_1}), (I_{21}), \dots, (I_{2n_2}), \dots, (I_{31}), \dots, (I_{3n_3}), \dots$$

Clearwater Network CNP810SP
- start-up+, CNP810SP bol implementovaný ale do výroby sa nedostal
- sieť procesorov s podporou pre simultánny multithreading (SMT)
- 8 vláken

MLX1
- start-up+, x86 arch v ARM tele
- 586 (SMT) jadier pre smartfony

Alpha 21464
- podora pre SMT
- jedno jadro
- 4 vlánka

UltraSPARC-II
- viacjadrový/viacvláknový procesor (2 jadra / 2 čipy)
- referenčný procesor pre SPEC CPU 2006

SPARC M8
- viacjadrový / viacjadrový procesor (32 jadier, 8 vláken/jadro)

Intel HyperThreading
- 2 vlákna

Intel Atom
- 2 cestný SMT procesro

Intel Itanium
- Montecito: hrubozrnný paralelizlus
- Tukwila: 2 cestný SMT procesor

Intel Xeon Phi
- 4 cestný SMT procesor (časovo multiplexovaný)

Intel mikroarhitektúra

Atom

- nízka spotreba
- určený pre NB, embedead systemy a IOT

Core

- i3, i5, i7, i9

Xeon

- určený len pre servery

Itanium

- určený pre podnikové servery a vysokovýkonné systémy
- nie je to x86

„TICK-TOcK“ vyrobný model / tick-tock model / tick – tock

Najväčší výrobca procesorov mal každý rok od vzniku priniesť novú architektúru procesorov založenú na starom výrobnom procese s tým, že na ďalší rok ju previedol na zmiešaný proces.

- TICK: zmena výrobného procesu, rovnaká architektúra
- TOCK: nová architektúra, rovnaký výrobný proces

Modely multivláknových architektúr

- Neumann model – CF Model / sériové riadenie / seriove riadenie
  - princíp programového riadenia
  - intrepretácia sériového prúdu inštrukcií
  - programové počítadlo
  - inštrukcie sú uložené v inštrukčnej pamäti
  - údaje uložené v pamäti alebo v registroch
- model toku dát – DF Model
  - princíp toku dát
  - poradie inštrukcií v programe nemá vplyv na poradie vykonávanie týchto inštrukcií
  - ištrukcia sa vykoná vtedy ak sú dostupné všetky infoprácie na jej vykonanie
  - model toku dát nedisponuje spoločnými prepisovateľnými pamäťovými elementami
  - graf toku dát (DFG)
    - zoskúpenie uzlov DFG do takých podgrafou, kt vykazujú nízky stupeň paralelizmu
    - vopred def podgraf uzlov je tranformovaný do reťazca, kt sa vykoná sekvenčne
- hybridný CF/DF Model
  - prienik klasického prístupu spracovania inštrukcii a prsitupu riadenia výpočtu na základe toku dát
  - je založeny na riadiacich operátoroch (RO) ma bližšie ku CF Modelu a vykazuje aj črty DF modelu
  - paralelné riadiace operátory
    - majú za úlohu riešiť sled vykonávania inštrukcií
  - paralelné riadiace tokeny
  - existencia špeciálnych riadiacich inštrukcii
    - FORK – slúži na vytvorenie nového vlákna a na špecifikovanie prvej inštrukcie určenej na vykonanie na tomto vlákne
    - JOIN – má za úlohu znížiť počet vlákien, čo docielili tým, že špecifikuje v progr bodm kde sa dve vlákna stretnú, z nich pokračuje vo vykonaní len jedno a druhé sa zruší
  - komunikácia medzi vláknami je založená na výmene dát cez spoloČné registre alebo cez spoločnú pamôťovú oblasť, pričom rézia prístupu je vyriešená napr. pomocou explicitnej synchronizačnej techniky ako je semafor
  - inštrukčný rámec a štruktúra pamäte údajov je takmer taká istá ako v CF modeli. Jediný rozdiel je len v tom, že v systéme môže v danom okamihu existovať ciac riadiaich vláken vykonávných parallne

- v takýchto sys, rôzne inštrukcie, kt sa nachádzajú v rozličných vláknach, kt môžu behať na rôznych procesoroch, môžu sa odvolávať v danom okamihu na rovnaký údaj, čo môže zapríčíniť synchronizačné problémy
- v multiprocesorových sys, na odstánenie konfliktov vyplývajúcich z prístupu k zdieľaným dátam, je potrebné použiť explicitné synchronizačne techniky
- problém optimaliacie registrov v prípade hrubozrnných arch je možné vyriešiť pre každé vlákni
- v prípade jemnozrných paralelných arch, kde v danom okamihu na jednom stroji môžu bežať až stovky vlákien súčase, efetívne riešenie optimalizácie registrov vyžaduje veľký súbor registrov
- v paralelných výpočtových modeloch založených na riadiacich tokenoch je spracovanie prúdu údajov zabezpečené na základe iterpretácie grafu údajových závislostí – na rozdiel od DFG sa na hranách tohto grafu pohybujú riadaice a nie datové tokény
- postupnosť vykonávania inštrukcii je jednoznačne definovaná DDG
- inštrukcia reprezentovaná uzlom DDG, je pripravená na vykonanie vtedy a len vtedy, keď sú dostupné všetky riadiace tokeny na jej vykonanie (na každej vstupnej hrane sa nachádza riadiaci token)
- existuje silná analógia medzi dataflow modelom a modelom založeným na paralelnych riadiciach tokénoch, aj keď v prípade druuhého modelu riadenie prúdu inštrukcií prebieha nezávisle od prúdu dát
- inštrukčný rámec CF je rozšírený o ďalšie dva rámce
  - rámec riadiacich tokenov
    - uchováva riadiace tokeny potrebné na vykonanie inštrukcii
  - ráme uchovania adries operátorov
    - slúži na určenie adresy pamäťového miesta, kam bude treba poslať nový riadici token po vykonaní danej inštrukcie
  - tieto dva rámce nahradzujú programové počítadlo CF modelu

Procesy a vlákna

- Proces je inštanciou bežiaceho programu
- Vlákno je anlógiou ku procesu
- ku zdielanej pamätipogramu môžu pristupovať viaceré vlákna a kontrolovať ju

POSIX Threads

- Ptherads
- štandard pre unixové operačné systémy
- knižnicia môže byť pripojená k C programom
- špecifické API pre multitredové programovanie
- dostupná len na POSIXT systémoch (Linux, MacOS X, Solaris, HPUX…)
- pri kompilácii je potrebné linknuť knižnicu *-lpthread*

Globálne premenné / globalne premenne

- môže zaviesť zmetočné chyby
- programátor musí limitovať počet globálnych premenných len na tie, ktoré potrebuje

Začiatok vlákien v Pthreads

- vlákna sú odštartované, keď je spustený program
- zavádzajú pridanú komplexnosť, keď pootrebujeme zaviešt kód do nášho programu, potrebujeme totiž explicitne zaviesťmiesto, kde vlákno začína a potrebujeme datovu štruktúru, do ktorej vlákno ukladá informácie

pthread_t

- premenná obsahujúca aktuálne dáta uložené v systéme
- jej členy niesu priamo prístupné z používateľského kódu
- Pthread standard garantuje, že pthrad objek ukladá dostatok informácii na unikátne informovanie o vlákne, ktoré je dostupné

- funkcia na začatie vlákna: *void* thread_function(void * args_p)*
- zastavenie vlákna: *pthread_join*
  - po zavolaní *pthread_join* sa čaká na dokončenie behu vlákna asociovaného s objektom typu *pthread_t*

busy-waiting
- vlákno opakovane testuje podmienku, ale efektívne nepracuje na žiadnej užitočnej úlohe pokiaľ nenadobudne podmienka určitú hodnotu
- často optimalizované kompilátorom

mutex / mutexy (mutual exclusion)
- vlákno môže „robiť“ busy-waiting opakovane a CPU nerobí nič
- mutex je špeciálny typ premennej, kt môže byť používaná na obmedzenie prístupu do kritickej sekcie jedného vlákna v jeden časový okamih
- využíva sa na to, že vláknu je garantované, že ak pristúpi do kritickej sekcie, všetký ostatné vlákna do nej už nepristúpia
- Pthreads definuje špeciálnu premennú pre mutexy: *pthread_mutex_t*
- ak mutex dokončí svoju prácu musíme zavolať:

*int pthread_mutex_destroy(pthread_mutex_t * mutex_p, FILE *in, FILE *out)*
- ak chceme aby vlákno získalo prístup do kritickej sekcie, musíme zavolať

int pthread_mutex_lock(pthread_mutex_t mutex_p, FILE *in, FILE *out)*
- ak vlákno dokončí beh kódu v kritickej sekcii musíme zavolať

int pthread_mutex_unlock(pthread_mutex_t mutex_p, FILE *in, FILE *out)*

bariery
- zosynchronizovanie vláken nás uisťuje, že vlákna sú pri zavolaní bariery, na rovnakom bode vo svojom behu
- žiadne vlákno nemôže prejsť cez barieru dovtedy, pokiaľ ju nedostiahnú všetky vlákna
- môžene nimi priamočiaro implementovať busy-waiting aj mutex
- ak counter indikuje, že každé vlákno vstúpilo do kritickej sekcie, ostatným vláknam je povolené kritickú sekciu opustiť

podmienené premenné
- datový objekt, kt môže vláknu pozastaviť jeho v činnosť do vtedy, kým nenanstane určitá podmienka – ak podmienka nastane iné vlákno môže čakajúce vlákno zobudiť
- podienená premenná je stále asociovaná s mutexom

cache
- blokujúca, relatívne rýchla pamäť, ktorá môže byť volaná procesorom
- má vysoký vplyv na používanie zdielanej pamäte
- L1, L2
  - súčasť čipu procesora (on-chip cache)
- L3
  - pôvodne implementované mimo čipu procesora (off-chip cache)
  - dnes súčasť matrice
- L4
  - mimo čipu procesora v eDRAM
- často potrebované dáta sú uložené v L1 cache, ak sa nenájdu v L1 cache hľadajú sa v L2 cache, potom v L3 cache a nakoniec v operačnej pamäti
- nájdenie dát je označované ako *cache hit*, nenájdenie dát je označované ako *cache miss*
- *read-miss* je stav, kedy sa jadro pokúča prečítať premennú, kt nieje dostupná v cache a musí ju zavolať z operačnej pamäte
- *write-miss* je stav, kedy sa jadro pokúša prepísať premennú, kt nieje dostupná v cache a musí ju zavolať z operačnej pamäte

umiestnenie jadra / core locatily

- s paralelným spracovaním sa objavuje potencionálny konflikt, že jednen alebo viac procesorov musí mať skopírované tie isté dáta

súdržnosť cache / case coherence
- ak jeden procesor vloží niekde kopiu svojích dát a iná kópia dát je updatovaná
- má dramatický efekt na výkonnosť systémov so zdielanou pamäťou

chybné zdielanie / nesprávne zdielanie / false sharing
- ak jedno jadro updatuje premennú *x* a iné jadro updetuje *y*, cahce sa bude plynulo presúvať medzi týmito jadrami

bezpečnosť jadier
- úsek kódu sa považuje za bezpečný ak môže byť naraz spúšťaný viacerými vlaknami bez toho aby spôsobil problémy

nebezpečné funkcie pre prácu s vláknami
- strtok() zo strstr.h
- random() zo stdlib.h
- localtime() z time.h
- niekedy majú thread-safe verziu
  - strtok() → strtok_r()

OPENMP
- open multirpocessing
- API pre zdielanie pamäte v paralelnom programovaní
- štandard pre programovanie systémov zo zdielanou pamäťou
- používa špeciálne funkcie a preprocesorové direktivy, ktoré sa nazývajú *pragmy (pragma)*
- viac využíva beh viacerých vláken ako viacerých procesov
- navrhnuté pre systémy, v ktorých každé vlákno alebo proces môže potenciálne pristupovať ku všetkej dostupnej pamäti
- na systém sa pozerá ako na kolekciu jadier alebo CPU a každé jadro/CPU má prístup k operačnej pamäti

pragma
- špecialná inštrukcia preprocesora
- pridaná do systému, aby umožnila správanie, ktoré nieje bežnou súčasťou špecifikácie jazyka C
- ak compaliler nepodporuje direktívu pragma, ignoruje ju
- tvat: *# pragma omp parallel num_threads(thread_count)*

#pragma omp parallel
- najbežnejšia direktíva preprocesora
- počet vláken, ktoré bežia na nasledujúcom bloku kódu je limitovaný run-time systémom
- pridanie *num_thread(thread_count)* umožňuje programátorovi špecifikovať počet vláken, ktoré môžu byť spustené na nasledujúcom bloku
  - tento počet môže byť limitovaný systémom
  - OpenMP štandard negarantuje, že takýto počet vláken bude skutočne vykonávať daný úsek kódu
  - bežne systémy podporujú aj beh stovák alebo tisícok vláken

team – vlákno A spolu s vláknmi, ktoré vlákno A vytvorilo
master – pôvodné vlákno, kt ako prvé začalo vytvárať ďalšie vlákna
slave – vlákna, kt boli vytvorené master vláknom
#pragma omp critical
- v jeden okamih môže len jedno vlákno spúšťať nasledujúci úsek kódu

oblasť platnosti premenných / oblast platnosti premennych / scope
- v sériovom programovaní, oblasť platností premenných obsahuje tú časť programu, v kt môže byť premenná používaná

- v OpenMP, oblasť platnosti premenných
  - referuje na množinu vláken, ktoré môžu pristupovať k premennej v paralelnom bloku
  - premenná môže byť dosiahnitá všetkými vláknami, kt zdieľajú túto oblasť
  - premenná, kt môže byť dosiahniutá len jedným vláknom sa nazýva *súkromná oblasť platnosti premenných (private scope)*

redukčné operátory / redukcne operatory / redukčný operátor / redukcny operator
- binárna operácia
- výpočet, kt je aplikopvaný opakovanie na sekvenciu operandov v stanovenom poradí aby sa dosiahlo jedného výsledku
- všetky čiastkové výsledky výpočtu môžu byť uložené v jednej, rovnakej premenej – redukčnej premennej (redukcnej premennej / reduknca premenna)

paralelný for / paralelny for / paralelný cyklus / paralelny cyklus
- skupina vláken ho spúšťa v štrukturovanom bloku
- cyklus je rozdelený do iterácia a jednotlivé iterácie sú priradené vláknam
- upozornenia
  - iteračná premenná (index) musí byť integer alebo pointer (nemôže to byť float)
  - všetky tri výrazy (start, end, increment) musia byť zhodného datového typu
  - výrazy start, end, increment sa nemôžu meniť počas vykonávania cyklu
  - počas vykonávania cyklu, môže byť iteračná premenná (index) modifikpvaná iba inktementom v časti for

klauzula default
- klauzula default upozorňuje, že kompailer bude vyžadovať aby sme špecifikovali scoup pre každú premennú, ktorú používame v bloku a bola deklarovaná mimo bloku

klauzila schedule(type, chunksize)

Type môže byť
- static
  - jednotlivé iterácie môžu byť priradené ku vláknam pred samostaným vykonaním cyklu
- dynamic / guided
  - jednotlivé iterácie môžu byť priradené ku vláknam počas vykonanía cyklu
- auto
  - kompaliler a/alebo run-time systém dopredu určí rozvrhutie
- runtime
  - rozvrh sa určuje za behu

Chuncksize je stále pozitívny integer.

Stacické plánovanie / staticke planovanie / static

```
schedule(static,1)
```

Thread 0 : 0, 3, 6, 9
Thread 1 : 1, 4, 7, 10
Thread 2 : 2, 5, 8, 11

```
schedule(static,2)
```

Thread 0 : 0, 1, 6, 7
Thread 1 : 2, 3, 8, 9
Thread 2 : 4, 5, 10, 11

```
schedule(static,4)
```

Thread 0 : 0, 1, 2, 3
Thread 1 : 4, 5, 6, 7
Thread 2 : 8, 9, 10, 11

Dynamické plánovanie

- iterácia sú rozdelené do „chunkov“ o veľkosti „chunksize“
- každé vlákno spúšťa jeden chunk a keď tento chunk dokončí, vyžiadá si iný chunk od systému. Takto sa pokračuje až pokiaľ niesú dokončené všetky iterácie
- „chunksize“ je explicitne nasavený na 1

OMP_SCHEDULE

- premenná prostredia, ktorá slúži na odhad (plánovanie) doby behu cyklu

Bariery

- implicitná bariera:
  - napr. *#omp parallel for*
- explicitná bariera
  - *#pragma omp barrier*
- ak vlákno narazí na berieru, zablokuje sa dovtedy, kým ho „nedobehnú“ aj ostatné vlákna „z jeho tímu“. Ak všetký vlákna z tímu prídu na barieru, sú naraz spracované.

Direktiva atomic

- môže byť použitá na chránenie kritickej sekcie, ktorá obsahuje len príkazy jazyka C

 # pragma omp atomic

- môže byť použíta len pre príkazy, ktoré majú nasledujúci tvar
  - *x<op> = <expression>;*
  - *x++;*
  - *++x;*
  - *x--;*
  - *--x;*
- veľa procesorov poskytuje špeciálny načitaco-modifikovaco-ukladacie inštrukcie (load – modify – store instruction)

Kritická sekcia / kriticka sekcia

- OpenMP poskytuje voľbu na pridanie mena kritickej sekcii pomocou direktívy

 # pragma omp ctitical (name)

- ak to urobíme, dvoe bloky s rôznymi menami môžu byť spustené naraz
- mená kritických sekcii sa nastavbujú za prekladu

Zámky / zamky

- obsahujú datovú štruktúru a funkcie, ktoré dovoľujú programátorovi explicitne vynútiť vzájomne vylúčenie v kritickej sekcii
- jednoduché zámky
  - môžu byť nastavené len raz pred tým ako sú odobmkute
- preťažené zámky
  - môžu byť nastavené viac, tým istým vláknom, krát pred tým ako sú odomknuté

Rady

1. nemal by si miešať rôzne rypy vzajomných vylúčení v jednej kritickej sekcii
2. neexistuje žiadná garancia spravodlivosti pri vzájomnou vylúčení
3. môže byť nebespečné „zahniezdenie“ v kritickej sekcii

GPU (Graphics processing units / graficke karty / grafické karty)

- vysoko paralélne, mutivláknové, viacprocesorové
- schopné veľkej výpočtovej schopnosti a vysokého prietok
- primárne používané na manažovanie výkonu pri videách a grafike
  - rendering
  - výstup monitorov
- špecializované pre vypočtovo náročné, vysoko paralelné výpočty

GPGPI (General Purpose Parallel Processors)

- podpora pre dostupné programovacie rozhania a štandardne jazyky ako je napríklad C

VPU (Visual Processing Unit)

- vstupom je video
- výstupom je video

TPU (Tensor Processing Unit)

- zrýchlenie hlbkového učenia
- vyvynutý Googlom a NVIDOU

CPU vs GPU

CPU	**GPU**
všeobecný účel	Naučínnejšia ak su všetky komponenty PC sústredené na jednu špecifickú činnosť
Pre beh programu je vyhradené len jedno (alebo veľmi málo) CPU, dosahuje sa nízkej latencie	Používa tisíce menších a efetívnejšiech jadier na dosiahnite masivneho paralelizmu
	50 – 100 x rýchlejšie ako si vyžadujú viaceré paralelne procesy

Využitie

- hry
  - renderovanie hracieho sveta
  - moderné hry sú veľmi náročné na grafiku
- 3D vizualizácia / 3D vizualizacia
  - pri CAD
  - požiadavka je vizualizovať objekty v 3D v riealnom čase, rotovať nimi a hýbať nimi
  - pr: AutoCAD 2019

- spracovanie obrázkov (image processing)
  - využíva väčšinou mnoho PC zdojov
  - najviac používane pri hraničných kontrolách, bezpečnostných kontrolách, alebo rengene v medicíne
- big data
  - GPU su využívané na interatívne vizualizácie a integrocanie s ostatnými datasetmi
  - GPU pomáhajú vylepšiť výkon, analýzu, pochopenie vzťahov medzi dátami
- hlboké strojové učenie (hlboke strojove ucenie / deep machine lreaning)
  - GPU sa využívajú na rýchle natrénovanie neurónovej siete pri spracovaní obrázkov, videa, rozoznávaní reči, autonómnych riadeniach, visualizáciach na PC a ďalších

Programovanie GPU

- bežiaci SW musú byť schopný behu na GPU
- graficka karta je programovateľná procesná jednotak s rôznymi typmi pamäte a cache a s rôznymi typmi funkcii špecializovanými na grafické úlohy

Technológie GPU

- vlákno je asociované s každým elementom dát
- vlánka sú organizované do blokov
- blok je priradený procesoru a ten spúšta kód
- bloky sú organizované do gridov
- mechanický objekt, ktorý je HW vytvora, manažuje, plánuje a spúšťa je vlákno SIMD inštrukcie (toto vlákno beží na multivláknovom SIMD procesore)

Plánovač / planovac / scheduler

- posiela inštrukcie na beh na SIMD procesore
- 2 druhy HW plánovačov
  - plánovať vláknových blokov

        - priradttuje bloky na multithreadové SM
    - SIMD plánovač vlaknových blokov bez S
        - plánuje keď vlákno SMID inštrukcie beží

CUDA (Comuputer Unified Device Architecture)

- programovací jazyk podobný jazyku C vyvinutý formou NVIDIA
- používa heterogénny model spušťania (CPU ako host, GPU ako zariadenie)
- programovaci model SIMT= Singe Instruction Multiple Thread
- funkcie alebo deklarácia dát pre zariadenia sú zaznačené špecialnými CUDA C kľučovými slovami
- obsahuje veľa funkcii, ktoré môžu napomôcť ku dosiahnutiu paralelizmu
- keď do zdrojového kódu v jazyku C pridáme deklarácie alebo kľučové slova z CUDA, tento zdrojový kód už nieje skompilovateľný C-ečkovským prekladačom, preto musíme použiť CUDA C prekladač NVCC (NVIDIA C Compailer)
- celkový počet vláken v jednom vláknovom bloku je špecifikovaný v hostovskom zdrojovom kóde
- jadro môže byť požiadané o vykonanie rôzneho počtu vláken v rôznych častiach zdrojového kódu
- CUDA jadro (CUDA kernel) má prístup k dvom zabudovaným premenným (*threadIdx*, *blockIdx)*, ktoé dovoľujú vláknu rozlišovať oblasť dát na ktorej vlákno pracuje
- *threadIdx* určuje každému vláknu unikátny kordinátor
- všetky vlákna zdieľajú pamäťový priestor
- vlakna v bloku môžu spolupracovať medzi sebou
    - synchronizujú svoje vykonávanie
    - zdieľajú dáta s nízkou latenciou
- dve vlákna z dvoch rôznych blokov niesu schopné spolupracovať
- globálna pamäť môže byť prečítana aj prepísana zariadením
- konštantná pamať podporuje malé spomalenie, visoku širku pásma, prístup len prre čítanie
- registre a zdielaná pamäť sú príkladom „pamäte na chipe“

API funkcie, ktoré pomáhajú programatorovi manažovať data v pamäti

- *cudaMalloc()*
    - host alokuje pamäť na globálnom zariadení
    - dve parametre
        - adresa pointra na alokovaný objekt
        - veľkosť objektu v bajtoch
- *cudaFree()*
    - uvoľnenie pamäťového priestoru z globálnej pamäte
    - jeden parameter
        - pointer na alokovaný objet
- *cudaMemcpy()*
    - presun medzi pamäťami
    - 3 parametere
        - pointer na cieľ
        - pointer na zdroj
        - počet bajtov, kt. sa budú kopírovať
- transfer je asyncrhonny

Architekrúty MIMD

Mikroprocesory

- spojenie viacerých nezávislých procesných elementov
- každý procesný element
    - má vlastnú kópiu programu
    - môže pracovať s odlišným dátovým prúdom

- implementácia prístupu do pamäti
  - zdieľaná pamäť
    - procesor UMA
    - procesor NUMA
    - procesor COMA
  - distribuovaná pamäť
    - multipočítače

Kombinácia rôznych prístupov na báze MIMD

- špecializované architektúry
- hybridné architektúry
  - MSIMD
    - multivektorové PPS
    - Multiple Instruction Stream Associative MASC procesor
  - SPMD ~ MIMD
  - MIMD/SIMD
- počítače DataFlow
- redukčné PC

PC riadené tokom dát / PC riadene tokom dat

- paralelna organizácia výpočtu typu data-driven
- inštrukcie programu DF pasíne čakajú na príchod určitej kombinácie svojich argumentov, sprístupňovanie ktorých sa organizuje ako údajový prúd riadena v zmysle data-drive
- interval čakania inštrukcie na príchod operandov reprezentuje jej výberovu fázu, v priebehu ktorej dochádza k alokácii výpočtových prvkov ku všetkým inštrukciá s dostupnými argumentami
- strojovou reprezentáciou programu DF je datalflow graf
- výpočtový model DF podporuje paralelizmus na úrovni inŠtrukcii
- úroveň paralelizmu v programoch závisí od granularity organizovania výpočtového procesu

Charakteristické princípy výpočtového modelu DF

Asynchrónnosť

- operácie sa vykonajú vtedy a len vtedy, keď sú dostupné vyžadované operandy

Funkcionalika

- operácie sú funkcie, t.j. nezávisia od operandov iných funkcii (neexistujú vedľajšie účinky) v dôsledku čoho sa môžu vykonávať paralelne

Klasifikácia DF architektur

- statická DF archutektúra
- dynamicka DF architektúra s explicitným ukladaným dátových tokenov
  - DF architekúra s priamym spajaním operandov
  - hybridná DF architektúra
    - skorá hybridná DF architektúra
    - DF architektúra s vláknovým prístupom
    - hrubozrnná DF architektúra
    - RISC DF architektúra

Graf toku dát / graf toku dat / data flow graph / dataflow graph / DFG

- orientovaný graf
- uzly reprezentujú ištrukciu
- orientované hrany indikujú tok výsledných dáto vo forme aktivačných značiek (tokenov), kt môžu byť dátove alebo riadiace

Vlastnosti DFG

- funkcionalita
  - operácie v DFG sú funkcie
- asynchronnosť

    - uzly sa aktivujú v okamihu, keď sú na ich vstupných hranách pritomné aktivačné značky
- komponovateľnosť
    - DFG je možné spájať
- vykonávanie DFG
    - pravidlo vykonateľnosti
        - inštrukcia je vykonateľná, ak všetky jej operandy sú prístupné
    - pravidlo aktivácie
        - inštrukcia je aktivovaná, keď je vykonateľna a zdroje na jej aktiváciu sú k dispozícií

Statická DF architekúra

- operátor v podobe uzla je vykonateľný, keď sú tokény prítomné na všetkých vstuných hranách a na výstupnej hrane nie je žiaden token
- využíva štrukturálny paralelizmus
    - vykonanie rôznych operátorov naraz
- využíva paralelizmus zreťazenia
    - v rôznych častiach grafu sú konzumované rôzne tokeny v tom istom čase
- patrí sem
    - systém LAU
    - DDM1
    - HDFM

Dinamická DF architekúra

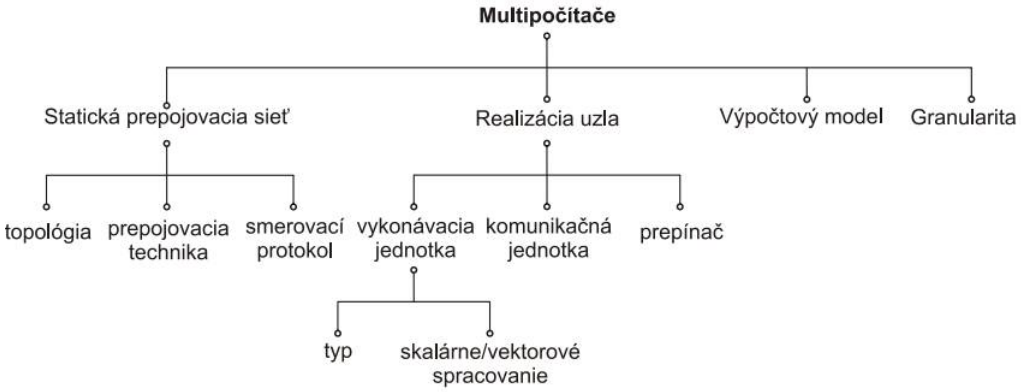
- operátor spojený s uzlom je vykonateľný, keď všetky vstupné hrany obsahujú tokény, ktorých značky sú identické
- využíva slučkový aj rekurzívny paralelizmus
- spájanie tokenov – asociatívne ukladanie údajov
- Architektúra Manchesterského PC
- EXMAN
- SIGMA-I

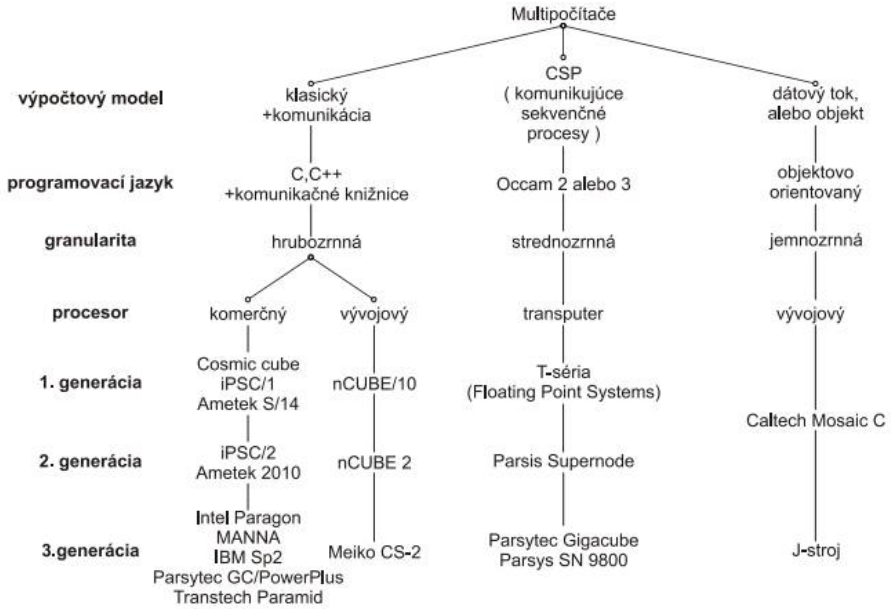
Modifikovaná DF architekúra

- EM-4
- TAM
- DF KPI

Distribuované systémy

Sieťové prostredie vzájomne prepojených pc uzlov, umožňujúcich prenos informácii medzi nimi a zdieľanie HW a SW prostriedkov.





FPGA (Filed Programmable Gate Array)

- programovateľné hradlové pole, ktoré umožňuje vytvoriť „ľubovoľný“ digitálny obvod
- sámo o sebe nedokáže vykonávať žiadnu funkciu na rozdiel od mikrokontroléra
- má FLASH pamäť
- po pripojení na zdroj stiahne konfiguráciu na FPGA
- po zapnutí nie je okamžite funkčné

Štruktúra FPGA

- vysoký počet vývodov
- najčastejšie v BGA prúdoch (preto nieje možné amaterské použitie)
- základná štruktúra
  - logické bloky
  - programovateľná prepojovacia matica
  - programovateľne I/O bloky
- moduly (prídavné bloky)
  - embadded processors (Microblaze, Nios)
  - DSP bloky
  - Distribuovaná pamäť
  - Bloky na generovanie a správu hodín (clock management)
    - PLL (Phase-locked Loop)
    - DCM (Digital Clock Management)
- logické bloky
  - konfigurovateľné logicke bloky
    - CLB Xilinx
    - LAB Altera
  - obsahujú
    - lookup table (LUT)
    - ščítačky
    - klopné obvody

Opis digirálneho obvodu

- pomocou HDL
  - na rozdiel od konvenčných jazykov sa tu kód nevykonáva (výnimkou je simulácia)
  - z kódu je odvodená konfigurácia FPGA
  - masívne paralelny priebeh, k tomu sa využíva kocept procesov a signálo

    - Low level HDL
        - VHDL
        - Verilog
        - System Verilog
    - High level HDL
        - System C
        - Catapult C
        - Vivado HDL
        - Visual HDL
        - MyHDL
- delí sa na
    - behaviorálny
    - štrukturálny
    - tok dát

Vznik konfigurácie FPGA z HDL

1. Syntéza
    - analyzuje HDL opis a hľadá zodpovedajúce HW štruktúry v podobe kombinačných a sekvenčných logický obvodov
2. Mapovanie na cieľovú technológiu
    - na základe výbraného typu FPGA sa snaží najsť najúspornejšie / najrýchlejšie ekvivalenty v syntéze najdených komponentov
    - zisťuje sa, či ja návrh realizovateľný na zvolenom FPGA (počet komponetov, časové charakteristiky)
3. Rozmiestnenie a prepojenie
    - hľadá sa vhodné usporiadanie a prepoje nie jednotlivých komponetov FPGA v prepojovacej matici
    - rieši sa hlavne časová analýza, kt určí či návrh bude fungovať na zvolenej frekcvencii (a iných časovacích obmedzeniach)

Výhody FPGA / vyhody FPGA / prečo FPGA / preco FPGA

- flexibilnejšie ako dedikované čipy
    - podporujú rýchle protypovanie
    - rýchla simulácia
    - IP Cores
- sú rýchlejšie ako mikrokontroléry, resp mikroprocesory
    - výraznejšie ako DSP
    - pracovná frevencia 10^2~10^3 MHz
- sú programovateľné vo fáze návrhu, výroby a aj prevádzky
- podporujú vysokú hustou integrácie (nízka cena, malý rozmer, energeticky nenáročné)
- výroba je cenovo dostupnejšia a menej riskantná
- priaznicý pomer Počet logických blokov / Cena
- rôzne I/O zariadenia
    - PCI Express
    - HDMI
    - Gigabit Ethernet

Technológia	Výkon / Cena	Programovanie [väzbou na čas]	Rýchlosť [reakcia na zmenu vstupu]	Flexibilita [preprogramovateľnosť]
ASIC	Výborný	Veľmi náročné	Výborná	Žiadna
FPGA	Veľmi dobrý	Náročnejšie	Veľmi dobrá	Uspokojivá ~ Dobrá
GPU	Veľmi dobrý	Náročné	Veľmi dobrá	Dobrá
DSP	Veľmi dobrý	Náročné	Dobrá	Veľmi dobrá
Generic CPU	Uspokojivý ~ Dobrý	Nenáročné	Uspokojivá ~ Dobrá	Výborná

Softvér – mentor graphics

- programový balík „HEP Design, Verification and Test“ firmy Mentor Graphics je univerzálny súbor špičkových nástrojov pre návrh čislicových systémov na báze ASIC a FPGA

Kľúčové programové nástroje pre prácu s integrovanými obvodmi		
Návrh číslicového systému	HDL Designer, Visual Elite HDL	Blokový a stavový diagram, VHDL, Verilog, TLM, ESL, ...
Syntéza	Precision RTL	Podpora širokej škály FPGA a ASIC (Xilinx, Altera, Actel, Atmel, Lattice, ...)
Simulácia, verifikácia, prototypovanie	Questa (resp. ModelSim SE)	Schopnosť pracovať s kombinovaným grafickým a HDL opisom hardvéru
• Podpora automatizovanej tvorby dokumentácie • Podpora OS: Windows (XP x86, 7 x64,); Linux (RHEL 5 x86, RHEL 5 x86-64)		

Kintex XC7K325T-2FFG900C

- frekvencia: 10 MHz – 810 MHz
- 326 080 buniek
- MicroBlaze 241-337 MHz soft core
- 1GB DDR3 pamäť
- Ethernet 10/100/1000 PHY, RJ45
- PCI Express
- AD/DA prevodník
- HDMI konektor
- LCD displej

Spartan-3 XC3S700AN-FG484 FPGA

- 500 MHz – 133 MHz oscilátor
- 700K systémových brán
- 13248 logických buniek
- MicroBalze 66.67 MHz
- DDR2 pamäť
- Ethernet 10 / 100 PHY, RJ45
- AD/DA prevodník
- VGA konektro
- RJ-232 sériový port
- 2x16 LCD displej