**10.6.2009 - sk.C**

```
* SPEC benchmarky
* RICS a CISC - proc takovy rozdil mezi poctem pregistru
* vzajemne vylouceni / kde a jak resit
* profilery - na co, kde, jak, co ukazuji
* optimalizace cyklu pri pocitani matic / cykly do bloku
* PVM- co,bvlastnosti, komponenty
* limitovane adresare - jaky vliv na virtualni pamet
*
```

---

3.6.2009 -- procesory, paralelni iritace, srovnani pvm a mpi, jedna optimalizace, vsechno otazky z minula (naucit se procesory, paralelni pocitace, mpi, pvm – a mohlo by to stacit)

---

**26.5.2009**

**1) CISC, RISC - rozdíly + co má větší podporu překladačů**

-- CISC=malá a drahá paměť, přímá podpora překladačů, složité adresování, složité instrukce, mikroprogramování-makro-instrukce, zpomalení za cenu kompatibility

-- RISC-redukovaná sada instrukcí, dramatický pokles ceny a vzrůst velikosti hlavních pamětí, zavedení Cache-vyrovnávací paměť (mezistupeň mezi procesorem a pamětí), vyřešen nedostatek paměti, odložené skoky, kvalitně optimalizující překladače, roste význam překladačů a kompilátorů, jednotná délka instrukcí, definovaný čas každé instrukce ve fázi execute, 2 instrukce na přístup do paměti – Load/Store, více registrů, superpipelines (rozbití částí procesoru na více- efektivnější využití, ale více devastující/nákladnější skoky), superskalarni jednotky (více ALU, FPU=floating point unit), dnes VLIW=paralelizace na úrovni instrukcí (30 instrukcí za 1 tik hodin)

-- přímou podporu mají CISC

**2) pipelining**

-- překrývání instrukcí v různých fázích rozpracovanosti, dekomponování instrukce na několik částí – Instruction Fetch – načtení z paměti, Instruction Decode-rozeznání, Operand Fetch-připravení operandů, Execute-vykonání, WriteBack-zapsání výsledku zpět(registry, paměť), instrukce jsou zpracovány paralelně s posunem o jednu fázi, máme dva typy:
Neviditelný-aktuálně používané, předsunutí čtení/zápis před vlastní práci s daty,
Viditelný-explicitní instrukce s přesně definovaným počtem cyklů do dokončení
--paralelizace instrukcí jedné úlohy, standardně 5ti úrovňový pipelining

**3) ANDES**

-- Architecture with Non-sequential Dynamic Executing Scheduling
Pro každý typ instrukce je fronta s piperine(analýza HW), fronty se obsluhují nezávisle, spekulativní výpočet, instrukce se zpracovávají podle toho, jak jsou připraveny(nedodržení pořadí z kodu programu), dokončení instrukcí zajišťuje správné uspořádání - graduování, dovolení spekulativních skoků-výpočet pokračuje předpovězenou větví, neblokující Load/Store – Load dává adresu v paměti a registr – nedefinována rychlost, Andes Load- se nastartuje ve fázi Issue, podívá se do cache, pokud jsou data, když nejsou spusti se Load a instrukce se přesune na konec fronty, přejmenování registrů je aplikováno ve velké míře-rozpoznání závislostí

**4) SIMD, MIMD - popsat, rozdíly + kam zařadit cluster**

-- SIMD a MIMD jsou architektury pro paralelní počítače (jednotlivé instrukce se mohou vykonávat paralelně)

-- SIMD =single instruction multiple data, silně synchronizované, všechny procesory vždy vykonávají stejnou instrukci, analogie vektorových počítačů-shodné výhody a nevýhody, dnes na úrovni jednoho prostoru, jednodušší programovací model

-- MIMD=multiple instruction multiple data, každý procesor může zpracovávat data nezávisle na ostatních procesorech-plně asynchronní, architektura dnešních superpočítačů, obtížné programování, potkávání ve výpočtu – bariers(explicitní synchronizace), flexibilita, efektivita
-- rozdíly—SIMD je synchronizované a MIMD – je nesynchronizované
-- cluster -- MIMD

**5) zneplatnění x update – rozdíly**
--zneplatnění=reakce na změnu dat ve vzdálené vyrovnávací paměti, řádka v aktuální naslouchací vyrovnávací paměti je zneplatněna, v případě opětovného přístupu je přehrána ze vzdálené paměti
--update=řádka je okamžitě obnovena, při opětovném pokusu je již k dispozici, nevýhody-falešné sdílení(nepracují na stejných datech), přílišné zatížení sběrnice
--nelze rozhodnout, který z přístupů je obecně lepší

**6) MPI - vlastnosti (zasílání zpráv - možnosti); skupinová komunikace - popsat + řešení v MPI**
--message passing interface, umožňuje paralelní programování, standard-komunikační rozhraní pro paralelní programy, knihovny + skripty, nezávislé implementace – optimalizace na daný HW, určena pro počítače, clustery, gridy, především podpora meziprocesorové komunikace,
--zasílání zpráv=operace send, receive, s buffery, bez bufferu, blokovací, neblokovací, tagy a kontext u zpráv (snadnější rozpoznání), nutná synchronizace, data jsou posílána jako proud bytů, musíme znát id příjemce/odesílajícího,
--módy zasílání zpráv=standardní(blokující, může být bufferován), synchronní (blokující), asynchronní (neblokující), Ready (receive musí předcházet send – připravení cílového bufferu)
-- skupinová komunikace=zasílání zpráv, skupina procesů, kontext, tag zrávy, id příjemce/vysílajícího, synchronizace, identifikace procesu vždy uvnitř kontextu
--zpráva=(adresa, počet, datovýTyp), možnost definice vlastního dat.typu, komunikační módy, plně duplexní komunikace, asynchronní, trvalé komunikační kanály

**7) optimalizace cyklů**
--redukce režie, zlepšení přístupu k paměti (cache), zvýšení paralelismu, snaha minimalizace skoků, při výpočtu více hodnot v cyklu stačí zapsání pouze té poslední, rozvoj cyklů-změna velikosti iteračního kroku, vyvarovat se nevhodným cyklům (=malý počet iterací, tlusté, cykly s voláním procedur, cykly s podmíněnými výrazy, rekurzivní cykly), co nejvíce věcí z cyklů ven (invariant), součet rychlejší než násobení

**8) schování pomalé paměti pomocí konzistence**
--zaměstnání procesoru, když se čeká na paměť, ANDES-realizuje ukrytí i přeskládání instrukcí,
--ukrytí zpoždění s pomocí slabé konzistence – nepožaduje striktní uspořádání ke sdíleným proměnným vyjma synchronizačních, Release Consistency-zavedení operací acquire a repase – synchronizace a kritická sekce, Fence operace-vynucené dokončení rozpracovaných instrukcí, Prefetch-stav, kdy sáhneme do paměti s dostatečným předstihem než budou potřeba data – binding (load do registru a předpokládáme, že tam už jsou, nepočítáme s přepsáním někým jiným-snížení efektivity nebo porušení konzistence), non-binding (data přesunuta pouze do vyrovnávací paměti)
-- HW (vyrovnávací paměť, ANDES), SW prefetch (podpora kompilátorů)

Slabá konzistence:
-- nepožaduje striktní uspořádání přístupů ke sdíleným proměnným vyjma synchronizačních
-- Release consistency – zavedení operací acquire a repase
-- používám a udržuji jen aktuální používané hodnoty, v jiných tyto (nepoužívané) hodnoty už neaktualizuji, musí si o ně explicitně říct
-- Fence operace – vynucené dokončení rozpracovaných operací (zarážky a sdílení hodnot v tom okamžiku, pak zase release)
-- Silná konzistence = při každé změně všichni vidí aktuální stav

**9) profilery - druhy + co očekávat**
--procedurově orientované (prof, gprof) – instrumentace – (nový překlad programu, zpravidla dostupný přes přepínač překladače), výpočet - (instrumentovaný program vytváří záznam o výpočtu, soubor mon.out), vytvoření zprávy – (vlastní běh programu prof), pokročilé procesory a operační systémy nepotřebují zvláštní překlad, instrumentace za běhu – SpeedShop (s nebo bez opětovného překladu)
--blokově orientované (pixie, tcov, lpvof)-využívají HW podpory – poskytují informace o průchodech základními bloky, TCO a lpvof – počet průchodů každým řádkem, pixie – počet cyklů procesoru strávených v každém příkazu
--přesnost výsledku závislá na vzorkování (sampling interval)
-- získání informací o částech programu, doba strávená v jednotlivých blocích, jednotlivými příkazy, počet opakování bloků/příkazů, ostrý profil-píky grafu odpovídají dominujícím procedurám, snadné nalezení pomalého místa, tupý profil – program tráví čas rovnoměrně ve všech procedurách (DB, IS), obtížně optimalizovatelné

---
více jak 10 otázek

**- je vhodné MPI pro paralelní počítače?**

-- ano, komunikace zasíláním zpráv, standard – pro komunikační rozhraní pro paralelními programy, umožňuje paralelní programování, zavedení datových typů pro data zpráv, tagy zpráv, kontext a skupiny procesů

**- popište rozdíly PVM a MPI**
-- PVM – síť démonů, primárně pro distribuované prostředí, tvorba paralelního počítače, zajištění převodu dat pomocí vnitřního mezijazyka
-- MPI – message paging určeno pro SPMD, MIMD, primárně jako podpora meziprocesorové komunikace, používanější, u 2.0 manipulace procesy, jednosměrné put, get, paralelní I/O, menší zátěž na programátora, převod dat pomocí nadefinovaných datových typů, lepší komunikace procesů než v PVM,

**- jaké jsou případy výpadku cache paměti; mají něco rozdílného jedno a víceprocesorové systémy?**
-- Compulsory miss (při prvním přístupu k datům – nutno načíst z jiné paměti), Conflict miss (různé adresy mapovány do stejného místa vyrovnávací paměti), Capacity miss (nedostatečná kapacita, něco se musí smazat), Coherence miss (různá data v různých vyrovnávacích pamětech – musí se sdělit ostatním změna –adresářová metoda, broadcasting, snoopy cache)

-- něco rozdílného jedno a víceprocesorové systémy = Coherence miss je **jen** u víceprocesorových systémů

---
Superpocitace 20.6.2006
skupina F, 10 otazek
- hlavni pamet - vlastnosti, parametry, struktura
**- RISC - vlastnosti, proc ma tolik registru**
--tolik registrů má, aby mohl využít paralelizaci na úrovni HW, sděluje menší počet registrů než doopravdy vlastní, přejmenovává je – obcházení (závislosti), musí se čekat na to, až si nějaká instrukce vyzvedne data z registru, aby mohl být znovu použit

**- SIMD/MIMD - vlastnosti, jake programy**
--programovací modely – SMPD (Single program multiple data – rozdělení výpočtu na více jader), MPMD (komunikace mezi jádry, kde běží odlišné programy)

- kriticka sekce - proc, co resi, jak je resena
--

**- klasicke optimalizace - ne optimalizace cyklu**

-- flow graph – snížení počtu procházených stavů/adres, odstranění opakovaných podvýrazů, přebytečných proměnných, zvolení vhodného překladače, přístupu k paměti (indexování C-po řádcích, fortran- po sloupcích), propagace kopírováním, zjednodušování výrazů konstantami, odstranění mrtvého kódu (nedosažitelný, nepoužitý výsledek), strength reduction – bitové posunutí, přejmenování proměnných, odstranění smetí, datové závislosti

**- Andes - vlastnosti, proc se zavadel**
-- zaváděl se, protože zpomalení bylo způsobeno čekáním na data, dynamický paralelizmus
--proto vícenásobné fronty instrukcí (celočíselné instrukce, adresní fronta pro operace Load/Store, fronta pohyblivé řádové čárky), nezávislá piperine pro každou frontu, celkové zrychlení – instrukce vybírány podle připravenosti, dokončovací fáze zajišťuje správné uspořádání instrukcí, spekulativní výpočet – (Fetch, Decode, Issue, exekute, Complete, Graduate), neblokující Load/Store, spekulativní skoky – výpočet pokračuje předpovězenou větví, nečeká na výsledek instrukce, přejmenování registrů – rozpoznání závislostí (určování dynamicky za běhu)

- zneplatneni / update
**- plny adresar**
--Plně mapovaný adresář = každá adresářová položka má tolik údajů, kolik je vyrovnávacích pamětí (procesorů), bitový vektor přítomnosti (nastavený bit znamená, že příslušná vyrovnávací data má kopii bloku paměti), příznak exklusivity (stačí jeden na blok, jen jedna vyrovnávací paměť), příznak v každé vyrovnávací paměti (každý blok, příznak platnosti – validity, příznak exklusivity)

- profilery, strmy profil
**- PVM - jak se komunikuje**
-- asynchronní zasílání zpráv, zprostředkováno démony, kteří odpovídají za spolehlivý přenos a doručení, zprávu vždy přebírá lokální pvmd (procesu příslušný), následně z TID cílového procesu určí jeho umístění a zprávu zašle jeho vzdálenému pvmd procesu, musí zajistit převod mezi architekturami

---
2006-06-14, skupina D

10 otazek

**CISC architektura**
-- nedělej programem to, co může udělat HW, zpětná kompatibilita na úkor zrychlení, mikroinstrukce, instrukce jsou emulovány - snadná záměna instrukčních sad, adresování (přístup k paměti), srovnání s rychlostí samotných procesorů – velikost a rychlost pamětí, přímá podpora překladačů

**reseni prodlev komunikace a vypoctu**
--pipelining – neviditelný (předsunutí čtení/zápis před bvlastní manipulaci s daty), využití prodlev k výpočtům (zpracování instrukcí, přístupy k paměti, výpočty v pohyblivé čárce)

Protokol update - vyhody, nevyhody, uvest priklad vhodneho pouziti
zhodnotit propojeni stylem switch vs. zvolene stromove propojeni (fat tree, omega), co pouzit pro 4 procesory
--

optimalizace cyklu - zavislosti (kdy jsou nevinne, kdy obtizne)
**cache coherence – vlastnosti**
--vyrovnávací paměti musí vědět o změně, metody založené na broadcastu, adresářové metody, snoopy cache (broadcastový přístup, sběrnice, každý procesor sleduje všechny přístupy k paměti), zneplatnění, update

**lze pouzit MIMD pro SPMD?**
-- ano

a par dalších
---
IA039 - 20.06.2006 Skupina E

**1. Jakym zpusobem se RISCove procesory vyporadaji s instrukci skoku a pripadnym vyprazdnenim pipeline. Popiste podrobneji dusledky predpovedi cile skoku. (12b)**

--vyprazdnění se snaží vyhnout, protože je drahé, skoky se snaží dynamicky předvídat, nečekají na dokončení instrukce, se skokem na adresu čtou hned i data v daném místě a okolí, nulování operace, vícenásobné předčtení z paměti, buffer potenciálních cílů skoku

**2. Co je virtualni pamet? Jakou roli hraje TLB v teto souvislosti? (10b)**

--fyzická vs. Logická adresa – více adresních prostorů- uspořádání zajišťuje OS+HW
--TLB=Translation Lookaside Buffer – tabulky se stovkami položek určité délky, slouží pro překlad logických adres na fyzické, je součástí HW, výpadky (misses), tvořeny ze stránek pevné délky (4kB až GB na stránku), HPC- nemají virtuální paměti vůbec- např. vektorové (vše se musí vejít na fyzickou paměť), někdy má virtuální paměť tak pomalé zpracování/načtení dat, že je lepší provést výpočet znovu (až o dva řády rychlejší)

3. Co rozumite pod pojmem interni paralelismus v pokrocilych procesorech? Jak je realizovan a proc se zavadi? (12b)

--

**4. Popiste a pojmenujte vypadky cache. Ktery neni u jednoprocesorovych systemu? (8b)**

--Compulsory miss (při prvním přístupu k datům – nutno načíst z jiné paměti), Conflict miss (různé adresy mapovány do stejného místa vyrovnávací paměti), Capacity miss (nedostatečná kapacita, něco se musí smazat), Coherence miss (různá data v různých vyrovnávacích pamětech – musí se sdělit ostatním změna –adresářová metoda, broadcasting, snoopy cache)

**5. Kterymi vlastnostmi budete charakterizovat propojovaci site paralelnich pocitacu? A jake konkretni vlastnosti plati pro propojovaci site typu "hyperkrychle"? (14b)**

--základní parametry: velikost sítě, stupeň uzlu, poloměr sítě, bisection width, redundance sítě, cena
-- hyperkrychle – pbecně n-rozměrná krychle, základní parametry-poloměr= log N, bisection N/2, redundance = log N, vyšší cena (N log N)/2, mřížky jsou speciálními případy hyperkrychle, snadné nalezení cesty – binární číslování uzlů - směrování

**6. Jake vlastnosti ocekavate u architektury NUMA? (8b)**

-- ne-uniformní, každý procesor má svoji lokální paměť, o kterou musí jiný požádat o přístup do paměti, hybridní systém, přístup k různým fyzickým adresám trvá různou dobu, umožňuje vyšší škálovatelnost, potenciálně nižší propustnost, koherence vyrovnávacích pamětí – ccNUMA – může způsobit cache na procesoru, vždy když je změna stránky, tak ji musím změnit/zneplatnit všude jinde, stránka je „doma“ jen na jednom místě, každé logické adrese odpovídá konkrétní fyzická adresa - snížení zpoždění paměti

**7. Co je to slaba konzistence? Znamena, ze procesory nemuzou pristupovat paralelne ke stejnemu useku pameti? Jak byste to osetrili? (12b)**

Slaba konzistence:
-- nepožaduje striktní uspořádání přístupů ke sdíleným proměnným vyjma synchronizačních
-- Release consistency – zavedení operací acquire a repase
-- používám a udržuji jen aktuální používané hodnoty, v jiných tyto (nepoužívané) hodnoty už neaktualizuji, musí si o ně explicitně říct
-- Fence operace – vynucené dokončení rozpracovaných operací (zarážky a sdílení hodnot v tom okamžiku, pak zase release)

--ošetření – něco jako ve vyšších jazycích semafory … fronta požadavků na zdroj, první vejde, zamkne, vykoná, odejde, odemkne

**8. Co je to loop unrolling? Jak prispiva k optimalizaci cyklu? (10b)**

--tělo cyklu se několikrát zkopíruje, sníží se režie – snížení průchodů cyklem, zvýšení paralelizace – i v rámci jednoho procesoru (sw pipelining), pre- a postconditioning loops (adaptace na skutečný počet průchodů)

9. Zkuste strucne popsat zakladni vlastnosti MPI, zejmena v oblasti podpory vlastni komunikace. Jak se lisi model pouzity pro zpravy v MPI od starsiho modelu, pouziteho v PVM? (14b)
--

10. Jak merit vykon? Jake nastroje ci prostredky byste pouzili? (10b)
-- benchmarking, profilig, MIPS, MFLOPS,

---
IA039 - Architektury superpočítačů - 29.5.2007
skupina B

**1. Srovnejte způsoby adresování a přístupu k paměti v RISC a CISC procesorech. Diskutujte důvody, které vedly k řešení použitém v RISC. (12b)**
RISC:
--rychlost přístupu přestala být hlavním úzkým místem, velikost rozsahu programu přestala být podstatná (i rozsáhlé programy se vejdou do paměti), problém-zadržení při čekání na výsledek předchozí instrukce,
--zavedení vyrovnávací paměti Cache u RISC, přestalo být reálné, aby si procesor bral data přímo z paměti – toto byl jistý mezistupeň mezi procesorem a pamětí
--obcházení/přejmenování registrů- snaha o odstranění závislostí (registr nemůže být smazán dokud si jeho obsah nevyzvedne jiná instrukce), vytváření stínových registrů, vytvoří se prostor pro paralelizaci (řešenou stínovými instrukcemi)
--řízení skoků - skoky nevedou daleko, vyhýbáme se drahému vyprazdňování pipelines, při skoku se čte cílová adresa a hned i data, která se případně velmi rychle provedou, předpověď skoku
CISC:
--adresování (přístup k paměti), velmi rychlé paměti, nebylo časově drahé sahat do paměti pro data, umožňovala složité adresování
--paměti měly vyšší rychlost než procesory, paměť byla malá a drahá

**2. Jaké problémy u RISC procesorů dělá zpracování skoků a proč? Byl tento problém i u CISC? Zdůvodněte. (10b)**
--předpovídání (statistické, dynamické, za běhu) adresy, kam se skočí je obtížné, sekvenční zpracováni instrukci ve frontách s pipelines, buffer potenciálních cílů skoku
-- problémy = je to drahé, když se skočí blbě
-- u CISC to nedělalo, přístup do paměti byl levný, předpovídání skoků nebylo,

**3. Jaký význam má přejmenování registrů u RISC? (8b)**
-- zvýšení paralelizace resp. HW paralelizace, z registru si musí data instrukce vyzvednout než může být znovu použit pro data

**4. Jaké vlastnosti běžně sledujeme u hlavní paměti? Zkuste popsat základní strukturu a operace, které se nad ní provádí. (10b)**
--přístupová doba (vystavení řádku, sloupce, dat), cyklus paměti (jak často lze data číst), statická a dynamická paměť, organizace paměti na řádky a sloupce (matice), page mode = naráz je čtena skupina souvisejících bytů
--

**5. Rozdíl mezi sdílenou a distribuovanou pamětí. Mají něco společného nebo nějaké rozdíly? (10b)**
--sdílená – paměť oddělená od procesorů, uniformní přístup k paměti, nejsnazší propojení je směrnice, levná komunikace, složité prokládání výpočtu a komunikace (aktivní čekání)
-- distribuovaná – distribuovaný cluster (lokální paměť u každého uzlu, vzdálená paměť ostatních uzlů), fikce jedné rozsáhlé paměti, HW řešení (zpravidla využívá principů virtuální paměti, transparentní), Sw řešení (knihovna, dnes již není rozdíl mezi přístupem k datům lokálně a přes síť, netransparentní – programátor musí program explicitně přizpůsobit)

6. Jaké jsou příčiny výpadků vyrovnávacích pamětí? Je rozdíl mezi jednoprocesorovými a víceprocesorovými systémy? (10b)
**7. Zvolte nějakou metodu zajištění koherence cache v paralelních systémech pomocí plných adresářů a diskutujte její vlastnosti. (12b)**
--snoopy schéma založené na broadcastu– nepoužitelné u složitějších propojovacích sítí, není rozšiřitelné,

**8. Diskutujte, jak je možné skrýt zpoždění při přístupu k paměti pomocí PREFETCH a PRODUCENTEM INICIOVANÉ KOMUNIKACE. (14b)**
--prefetch – přesun dat k procesoru s předstihem (binding – data až k procesoru, možné porušení konzistence, non-binding – data pouze do vyrovnávací paměti), HW a SW prefetch
--producentem iniciovaná komunikace – analogie invalidce a update při cache koherenci, specifické použití pro message-passing nebo blosk-copy, vhodné např. pro přesun velkých bloků dat či při synchronizaci – zámky, nemá smysl pro NUMA, ccNUMA, zajímavé pro distribuované systémy (síťové karty přímo přes DMA do vzdálené paměti)

9. Uveďte a stručně diskutujte jednoduché optimalizace (ne cyklů). (10b)
**10. Srovnejte PVM a MPI. Očekávali byste jejich použití u SPMD nebo MPMD? Má MPI smysl i v paralelních systémech? (14b)**
--MPI má smysl v paralelních systémech
-- použiti SMPD – není synchronizován na úrovni jednotlivých instrukci, ekvivalentní MIMD

---

IA039 Architektura superpocitacu.
Zkouska 23.5.2007. Skupina A
110 bodu. Kazda otazka za 8-14 bodu.

1) CISC, RISC, rozdily. Ktery znich vyzaduje vetsi podporu prekladace?
2) Pipelining, co to je, vlastnosti.
3) Andes - co to je. Jaky problem resi?
4) SIMD MIMD - popsat, rozdil mezi nimi. Cluster je SIMD nebo MIMD?
5) Zneplatneni x Update diskutujte.
6) Prime a neprime propojeni siti. Porovnat. Jeden priklad ke kazdemu.
7) Jak schovat pomalou pamet pomoci konzistence?
8) Jake jsou moznosti optimalizace cyklu? Jake problemy mohou nastat? A jeste neco o optimalizaci cyklu :)
9)Profilery. uvest jejich druhy a co byste od kazdeho z techto druhu ocekavali (heslovite)?
10)MPI jeho vlastnosti hlavne ohledne zasilani zprav. Jake moznosti nabizi pro zasilani zprav. Skupinova komunikace - co to je a jak je v MPI resena.
-- skupinova komunikace – procesy se přihlašuji do skupin, každý proces ma id, zprava se zasila broadcastem do skupiny, muze se ji pridat tag pro rychlejsi rozpoznani, zasilani pouze v danem kontextu