Písemka 1

Popsat RISC a proč se přejmenovali registry

RISC - reduction instruction set computer
Vznikl jako důsledek existence konceptu pipeliningu
Snaha o stejnou délku výpočtu všech instrukcí (kromě load a store, které pracují s pamětí)

- z CISC se vybrali často používané a snad implementované instrukce (kvůli tomu první RISC neměli výpočty s pohyblivou desetinnou čárkou)
- Z toho vyplývá delší kód a náročnější práce pro překladače (což vedlo k jejich vylepšení)
- Všechny instrukce pracují pouze s registry (kromě load a store, které přistupovali k paměti)
- Přidaná instrukce odložený skok - po instrukci JUMP byli načteny ještě 2 instrukce, aby se vyplnil prostor v pipeline

Přejmenování registrů:
RISC procesory používají 2x tolik registrů než ví překladač. Má tak možnost lépe optimalizovat na úrovni HW a používat paralelismus (Může číst a zároveň zapisovat do stejné paměti tím, že pro si zvolí stínovou paměť, do které zapisuje. Následně pracuje s novou pamětí místo staré).

Cache koherence metody

Cache koherence - ve vyrovnávací paměti je stará (neplatná) kopie dat

Řešení:
Broadcastová vyrovnávací paměť (**snoopy cache**) - paměť používající sběrnici a sledující komunikaci na sběrnici (zápis dat do hlavní paměti). Kontroluje, zda nejsou přepisována data ve vyrovnávací paměti, pokud ano, tak jedno ze dvou řešení:

- Zneplatnění - data jsou prohlášena za neplatné a v případě opětovného využití jsou znovu nahrané z hlavní paměti
- Aktualizace - data se při změně v hlavní paměti okamžitě aktualizují

Zde existuje možnost falešného sdílení (dva procesory pracují nad stejným řádkem, ale jiné části)
Nepoužitelné na velkých systémech, jelikož by museli být všechny procesory připojené na jednu sběrnici.

Existují alternativy pro přímé sledování sběrnice:

- **Adresářový přístup** -
    - Položka u každého bloku paměti
    - Odkazy na vyrovnávací kopii tohoto bloku
    - Označení excluzivity
    - Tři způsoby provedení:
        - **Plně mapované adresáře**
            - Každá adresářová položka má tolik údajů, kolik je cache
            - Bitový vektor - nastavený bit znamená, že příslušná cache má kopii
            - Příznak exkluzivity - pro všechny kteří pracovali nad daty se pošle zpráva o zneplatnění
            - paměťově náročné
        - **Částečně mapované adresáře**
            - Místo bitového vektoru ukazatele, ukazující na procesory s kopií dat .
        - **Omezené adresáře**
            - Místo bitového vektoru ukazatele, ukazující na procesory s kopií dat .
            - Je omezený počet procesorů které mohou paralelně přistupovat k jednomu řádku.
            - Může nastat přetečení:
                - zneplatnění všech dat a uvolnění
                - zneplatnění pro jeden procesor
        - **Provázaná schémata**
            - centrálně pouze jeden ukazatel ukazující pro paměť (+příznak exkluzivity) ukazující na procesor mající aktuální data
            - ostatní ukazatele svázány s vyrovnávací pamětí
                - Pokud chce procesor pracovat s daty, tak se ptá přímo procesoru, který má aktuální kopii dat
            - Výhody - v hl. paměti jenom jeden odkaz nezávisle na počet procesorů
            - Nevýhody - složitý protokol + zvýšená komunikace + delší zápis
        - **Hierarchické adresáře**
            - Použité v systémech s výcenásobnými sběrnicemi
            - V principu hierarchie snoopy protokolů

Co měříme ohledně vlastnosti sítí a vysvětlit co je bisection bandwidth a nakreslit příklad

Měříme:

- Velikost sítě N - počet uzlů
- Stupeň uzlu d - s kolika procesory daný procesor může přímo komunikovat
- poloměr sítě D - nejdelší nejkratší cesta
- redundance sítě A - kolik hran je potřeba odstranit, aby se síť rozdělila na dvě
- cena C - počet linek v síti
- Bisection width - kolik linek je potřeba odstranit, aby se sí rozdělila na dvě stejné
- Bisection bandwidth - celková kapacita odstraněných linek v bisection width (ideál *bisection_width/N*

Příklad:
Linka o------o------o-----o

- Velikost sítě: 4
- Stupeň uzlu: 1. resp 2
- poloměr sítě: 1
- redundance sítě: 1
- cena: 3
- Bisection width: 1
- Bisection bandwidth: kapacita linky/1

Optimalizace přístupu do paměti

- Cache
- Paralelizace paměti - rozdělení paměti na menší bloky, které se chovají automomně
- Přenášení dat po řádcích

Benchmark SPEC a co to je kernel

SPEC - Standard Performance Evaluation Corporation
Organizace testující výkon procesorů při specifických podmínkách za pomocí kernel kódů

kernel kódy - množina programů s jejich pomocí jsou testovány procesory - vybrány takové, aby reprezentovali reálné využití uživatelů procesorů

Různé skupiny, např:

- High performance group
- Graphic performance characterization group
- open system group

Test_and_set a test_and_test_and_set a algoritmus, jak se dá vyřešit problém těchto instrukcí (asi tím byla myšlena fronta procesů)

Test and set:

- Atomická operace zjišťující stav kritického místa v paměti.
- Pokud je paměť volná, nastaví příznak a vstupuje
    - Čte se a rovnou se nastavuje příznak bez čekání na výsledek čtení
    - Následně se teprve vyhodnocuje, jaký byl příznak původně
- Po ukončení práce příznak odebere
- Pokud je paměť již zabraná, opakuje se v intervalech
- Problém aktivního čekání
- Problém opakovaného zápisu (a tím zneplatnění dat pro ostatní, kteří čekají

Test and test and set:

- Prvně se otestuje, zda je kritická sekce už zabrána
- Pokud ano, pokračuje se v jiném výpočtu.
- Pokud ne, proběhne test and set
    - Pokud se během test and set zjistí, že proces již jiný proces předběhnul, provede se znovu test and test and set
- Nevyžaduje tolik zápisů

Problém vyhladovění - jeden proces je pomalejší než ostatní a ke kritické sekci se tak nemusí dostat. Řešení:

- Fronta
- Prioritní fronta

Jak se z hlediska paměti liší paralelní systémy se sdílenou pamětí a s distribuovanou pamětí - výhody a nevýhody.

Sdílená pamět:

- Procesory sdílí jednu hlavní paměť
- uniformní přístup k paměti
- “Levná” komunikace
- Složité prokládání výpočtů a komunikace (aktivní čekání)
- Se zvyšujícím se počtem procesorů klesá výkon (omezená přenosová kapacita)

Distribuovaná paměť:

- Ve větších distribuovaných systémech
- Každý procesor má svoji vlastní paměť
- Procesory komunikují mezi sebou za pomocí zpráv
- Výhody: minimální aktivní čekání - pokud se čeká, spustí se jiné vlákno
- Nevýhody: Vysoká cena komunikace

Co rozumíte pod pojmem zrychlení a přínos.

Zrychlení:

- Jak se změní rychlost výpočtu na jednom procesoru vs na *n* procesorech
- ideál - při n procesorech se výpočet zkrátí n krát
- realita
    - Je nutné doplnit logiku pro zpracování více procesory
    - Je nutné počítat s kapacitou sběrnice
    - Problém musí být dobře paralelizovatelný - seriovy problém nelze urychlit přidáním více procesorů

Přínos:
???

MPI a blokující/neblokující volání a závislost s využitím bufferů

MPI - Message passing interface:
Obecně:

- Komunikační prostředí pro paralelní systémy (sada instrukcí)
- Definované API s různými implementacemi (openMPI, SGI MPI)
- Určení pro SPMD / MPMD

Cíle:

- přenositelnost - definice standardu nezávislá na implementaci a jazyku
- výkon - optimalizace pro HW
- funkcionalita - snaha pokrýt všechny aspekty meziprocesové komunikace
- Spuštěn jeden proces který se na základě dalších dat rozhodne jakou větví výpočtu se vydat

Jádro funkcionality:

- MPI_init - vytvoření MPI prostředí
- MPI_Comm_Size - zjištění počtu procesů
- MPI_Comm_Rank - zjištění svého identifikátoru
- MPI_Send - zasílání zpráv
- MPI_Recv - přijetí zprávy
- MPI_Finalize - ukončení MPI

Funkce:

- komunikace pomocí komunikátoru
    - Definuje skupinu a kontext zpráv
    - Lze měnit/vytvořit komunikátor pro specifický kontext
    - Obsahuje Tagy pro upřesnění příjemce zprávy
- Datové typy
    - Datový typ je popsán trojicí (adresa, počet, datový typ)
    - Dovoluje složité typy
    - Umožňuje vytvoření nových typů
- Point to point komunikace
    - Odesílání

            - Blokující - proces čeká na receive
            - Neblokující
            - Bez bufferu
            - S bufferem
        - Příjem
            - Blokující
            - Neblokující
- Kolektivní komunikace
    - broadcast - odešle zprávu všem procesům
    - redukce - požaduje zprávu od všech ostatních procesů
- Virtuální topologie
    - umožňuje vytvořit SW topologii nezávisle na skutečném zapojení
- paralelní I/O

Závislost s využitím bufferů:
- Programátor si může vybrat, zda buffery využije nebo ne
- ????

Písemka 2

Hlavní rozdíly mezi CISC a RICS, kdo potřebuje větší podporu překladačů

CISC vs RISC:
- Komplexní sada instrukcí vs. malé instrukce s důrazem na stejnou délku výpočtu
- Využívané když rychlost procesoru cca stejná jako přístup k paměti
  vs procesor řádově rychlejší než přístup k paměti
- Obtížná paralelizace vs pipelines
- Instrukce přistupovali k paměti jak potřebovali
  vs pouze store a load mohou přistupovat k paměti
- programování za pomocí assembleru vs vyšší jazyky
- instrukce pracují s hlavní pamětí vs většina instrukcí pracuje s registry (a taky existují registry)
- využití málo externí paměti vs mnoho externí paměti

Potřeba větší podpory překladačů:
- RICS potřebuje větší podporu překladačů - je náročnější programovat na nízké úrovni v RICS vzhledem k velkému počtu instrukcí které jsou potřeba pro program (v porovnání s CISC, kde instrukce vykonávaly mnoho operací implicitně). Proto se přešlo na vyšší programovací jazyky a vznikl požadavek na lepší překladače

Pipeline

- Zvyšuje výkon procesoru pomocí paralelizace
- Uvědomění, že procesy prochází několika fázemi, kdy jednu fázy zpracovává jenom jedna část procesoru - ostatní části nevyužity
- Rozvržení instrukcí tak, že zatímco jedna část procesoru zpracovává určitou fázi procesu 1, druhá část může zpracovat jinou fázi procesu2
- Pro každou fázi se zpracovává jiný proces - vzhledem k jednotné rychlosti provedení lze takto rozplánovat instrukce na zpracování
- Tři základní oblasti: zpracování instrukcí, přístupy k paměti, výpočty v pohyblivé desetinné čárce
- Problém s instrukcí JUMP, které rozhazuje proces pipeliningu (jelikož se neví, jaká instrukce se bude volat)

ANDES

ANDES - Architecture with Nonsequentional Dynamic Execution Scheduling

- Rošíření RISC o další architekturu
- Procesory na úrovni superskalárů
- rozšiřuje tzv. okno instrukcí - pokud je následující instrukce blokovaná, hledá se další vhodná)
- Program se vykonává podle množství nezávislých instrukcí v daném okně
- má vícenásobnou frontu instrukcí, aritm. operace, FP operace a load/store (řadí dle příznaku instrukce)
- Využívá odložených load and store, instrukce se vybírají s fronty podle toho, zda už mají připravené data
- využívá spekulativních výpočtů - předvypočítávání větve při podmínce. Pokud se nakonec má provést, tak je použita, jinak zahozena
- Instrukce skoku se rozeznávají primárně.

Snaha o co největší výkon - využití procesoru
Dnes se ustupuje - vysoké zatížení procesoru - problém s chlazením

- Jedno z řešení je do pipeline přidávat prázdné instrukce

SIMD MIMD, clustry

SIMD - Single instruction multiple data

- Odpovídá vektorovému programování, nevhodné pro skalární výpočty
- Všechny procesory provádí stejnou instrukci nad různými daty
- Jednodušší procesy a programovací model
- Složitější programování - vektorové paradigma
- Použití v GPU jako akcelerátory

MIMD - multiple instruction multiple data

- Plně asynchronní model
- Samostatné procesory
- Větší flexibilita

- Složitější programovací model

Cluster - distribuovaný systém

- Lokální paměť každého uzlu
- Vzdálená paměť ostatních uzlů
- “Fikce” jedné rozsáhlé paměti
- Je potřeba HW nebo SW podpora

Zneplatnění, update

viz. Cache koherence metody

Parametry u sítí + dvourozměrné sítě

Parametry:

- Velikost sítě - počet procesorů
- Stupeň uzlu - počet sousedících procesorů
- Poloměr sítě - nejkratší nejdelší cesta
- Redundance sítě - počet linek, které musíme odstranit aby se síť rozdělila ve dvě
- Cena - počet linek
- Bisection width - počet linek které musíme odstranit aby se síť rozdělila na polovinu
- Bisection banwidth - kapacita linek * bisection width

Dvourozměrné sítě:

- Mřížka
- Torus
- Strom (+ fat tree)
- Hvězda
- Kruh

Jak ukryjeme zpoždění u propojovacích sítí, nějaká změna modelu nebo co…

- Přepínání okruhů
    - 3 fáze spojení
        - probe - ustanoví cestu
        - přenos dat
        - zrušení spojení
    - Buffery nemusí ukládat pakety a rovnou přeposílají
- Virtuální propojení
    - zpráva rozdělená na menší bloky - flits
    - první flits obsahuje informaci o cestě, další zprávu a poslední o ukončení cesty
    - flitsy posílány kontinuálně - zaplnění kapacity linky
- Směrování černou dírou
    - flits je velký jako buffer sítě

    - podporuje multicast a broadcast
- Virtuální kanály
    - Podporuje sdílení fyzických kanálů
        - Zábrana deadlocku
    - Více bufferů na uzlu
        - více procesů může využít přenosový kanál
    - Speciální systémový buffer pro systémové instrukce

Optimalizace cyklů + nějaké problémy s tím

Optimalizace cyklů:

- Nejdůležitější část na optimalizaci - nejvíce času procesor pravděpodobně stráví prací nad cykly
- Při optimalizaci:
    - Redukce režie
    - Zlepšení paralelizace
    - Zlepšení přístupu k paměti
- Optimalizace nad datovými závislostmi:
    - Flow dependenci - výpočet závisí na předcházejícím výsledku - cyklus se provádí v blocích pro paralelizaci - loop enroling
    - Anti-Dependencies - složitější závislost v cyklech - přejmenování proměnných
    - Output dependencies - počítají se hodnoty, které nejsou nakonec využity - pre a post conditioning
- Loop enroling - snížení počtu iterací cyklu tak, že se v jedné iteraci spočítá to, co se počítalo ve více iteracích
- Spojování cyklů - opakované použití dat
- Optimalizace přistupu do paměti - přeskládání instrukcí tak, aby se při jedno čtení načetlo co nejvíce dat pro cyklus

Nevhodné cykly pro optimalizaci:

- Malé cykly
- Cykly s podmínkou
- Cykly s voláním procedur
- Velké cykly

Problémy při optimalizaci cyklů:

- zahlcení registrů
- rozvoj špatným počtem iterací
- numerické nepřesnosti

Co je to komunikator v MPI a co řeší za problém z PVM