Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Universitatea Politehnica Bucuresti

**COERENTA MEMORIEI CACHE PENTRU MULTIPROCESOARE**

Diaconescu Oana-Ramona

Ion Alexandru

Grupa 434A

**Cuprins**

**Diaconescu Oana Ramona**

1. Memoria Cache –Introducere………………………………….3
2. Coerenta memoriei…………………………………………….5
3. Coerenta memoriei cache……………………………………...5
	1. Generalitati………………………………………….5
	2. Definitie…………………………………...…….......6

**Ion Alexandru**

* 1. Mecanisme…………………………………………..7
	2. Protocoale………………………………………… ...8
1. Bibliografie…………………………………………………….12
2. **MEMORIA CACHE - INTRODUCERE**

Memoria cache este o “memorie tampon” dintre memoria de lucru si microprocesor , mult mai rapida decat memoria de lucru, formata din celule SRAM rapide. Aparitia acesteia se datoreaza faptului ca unele componente din lantul de transmisie a datelor (memoria RAM si magistralele calculatorului), iar, in aceste conditii, fluxul de date cerut de la microprocesor este mai putin rapid decat ar fi necesar.

Ca atare, între posibilităţile de prelucrare a datelor de către microprocesor şi capacitatea RAM-ului şi a magistralelor de a pune la dispoziţia microprocesorului aceste date a apărut un decalaj foarte mare. Aceasta a dus în procesul de lucru al unui microprocesor la introducerea stărilor de pauză (*wait states*), astfel încât, prin execuţia a unuia, a două sau chiar a patru cicluri de aşteptare, datele solicitate să aibă timp să ajungă la procesor. Practic, se întâmplă următorul lucru: microprocesorul lucrează fie cu datele necesare calculelor, fie cu comenzi. Cum datele şi informaţiile necesare sunt cuprinse în memoria de lucru, microprocesorul citeşte memoria RAM, preluând anumite date, şi realizează calculele şi procesele cerute.[1]

Cum memoria este mult mai lentă în livrarea datelor faţă de viteza cu care le poate prelucra microprocesorul, în funcţionarea procesorului se introduc ciclurile de aşteptare despre care s-a vorbit anterior; cât timp aceste cicluri se execută, se permite datelor solicitate de procesor să fie localizate şi „extrase” din RAM, depuse pe magistrala de date şi „transportate” către microprocesor. În final, dacă se va înmulţi timpul cât durează un ciclu de aşteptare cu milioanele de operaţii pe secundă ale unui microprocesor, va rezulta o pierdere de timp substanţială. [1]

Deci era nevoie de o memorie de lucru ceva „mai aproape de microprocesor” şi mult mai rapidă (ideal fără să necesite operaţii de reîmprospătare). [1]

Principiul de functionare al memoriei cache [1]



Schema logica de citire a memoriei chace [1]



1. **COERENTA MEMORIEI**

Coerența de memorie este o problemă care afectează proiectarea de sisteme informatice, în care două sau mai multe procesoare sau core-uri de împărtășesc o zonă comună de memorie.

 Într-un sistem uniprocesor, există un singur element de procesare care face toata munca și, prin urmare, doar un element de procesare care poate citi sau scrie de la / până la o locație de memorie data. Ca urmare, atunci când o valoare este schimbata, toate operațiunile de citire ulterioare ale locației de memorie corespunzătoare vor vedea valoarea actualizată, chiar dacă este în cache.

Invers, în sistemele multiprocesor (sau mai multe conductoare), există două sau mai multe elemente de procesare care lucrează în același timp, și de aceea este posibil ca acestea sa acceseze simultan aceeași locație de memorie. Nici unul dintre ele nu modifică datele din aceasta locatie, dar pot împărtăși pe termen nelimitat memoria cache in functie de necesitati. Dar de îndată ce unul dintre ele actualizează locația, celelalte ar putea să funcționeze pe o copie „expirata” (out-of-date), care, de exemplu, se află în cache-ul local. Prin urmare, unele sisteme trebuie sa „anunte” toate elementele de prelucrare ale modificărilor față de valorile comune; un astfel de sistem este cunoscut ca un "protocol de coerență de memorie", iar în cazul în care un astfel de protocol este folosit, se spune că sistemul are o "memorie coerentă ".

 Natura exactă și sensul coerenței de memoriei sunt determinate de modelul de coerență pe care protocolul il implementeaza. În scopul de a scrie programe concomitente corecte, programatorii trebuie să tina cont de modelul exact de coerență care este implementat de sistemele lor.

1. **COERENTA MEMORIEI CACHE**

**3.1 Generalitati**

Coerența memoriei cache este consistența/coerenta datelor partajate de resurse care sfarsesc stocate în mai multe memorii cache locale.

Atunci cand clientii dintr-un sistem mentin memorii cache ale unei resurse de memorie comună, pot apărea probleme cu datele inconsistente. Acest lucru este valabil mai ales la procesoare într-un sistem de multiprocesare. Referindu-se la imaginea de mai jos, în cazul în care top-client-ul are o copie a unui bloc de memorie de la o citire anterioară și bottom-clientul schimbă acel bloc de memorie, top-client-ul ar putea fi lasata cu o memorie cache invalida fără nici o notificare de schimbare . Coerența memoriei cache este destinata pentru a gestiona astfel de conflicte și de a menține buna legatura dintre cache și memorie.

Într-un sistem multiprocesor cu memorie partajata cu o memorie cache separata pentru fiecare procesor, este posibil să existe mai multe copii ale oricărei instrucțiuni: o copie în memoria principală și una în fiecare memorie cache. Când o copie a unui operand este schimbata, celelalte copii ale operandului trebuie de asemenea schimbate. Coerența memoriei cache este disciplina care asigură că schimbările în valorile operanzilor comuni sunt propagate în întregul sistem, în timp util. [2]

 Există trei niveluri distincte de cache coerență: [3]

* fiecare operație de scriere pare să apară instantaneu
* toate procesoarele vad exact aceeași secvență de schimbări de valori pentru fiecare operand in parte
* procesoare diferite pot vedea o operațiune și pot asuma diferite secvente de valori; acest lucru este considerat ca fiind un comportament necoerent.

 **3.2 Definitie**

 Coerența definește comportamentul citire și scriere în aceeași locație de memorie. Coerența cache se obține dacă sunt îndeplinite următoarele condiții: [4]

* Într-o citire făcută de un procesor P într-o locație X care urmează dupa o scriere de același procesor P in X, fara scriere despre X de către un alt procesor care sa intervina între instrucțiunile de scriere și citire realizate de P, X trebuie să returneze întotdeauna valoarea scrisa de P. Această condiție este legată de păstrarea ordinii de program si trebuie să fie pastrata și în arhitecturi uniprocesor.
* citire făcută de un procesor P1 la poziția X, care are loc după o scriere de un alt procesor P2 la X trebuie să returneze valoarea scrisă de P2, dacă nu apar alte scrieri in X efectuate de orice alt procesor între cele două accesele și scrierea si citirea sunt separate suficient. Această condiție definește conceptul de vedere coerenta de memorie. Dacă procesoarele pot citi aceeași valoare veche după scrierea făcută de P2, putem spune că memoria este incoerenta.
* Scrierile în aceeași locație trebuie ordonate. Cu alte cuvinte, în cazul în care locația X a primit două valori A și B diferite, în această ordine, de la oricare două procesoare, procesoare nu pot citi niciodată locație X ca B si apoi ca A. Locația X trebuie să fie văzuta cu valorile A și B în această ordine.[3]

Aceste condiții sunt definite presupunand că operatiile de citire si scriere sunt făcute instantaneu. Cu toate acestea, acest lucru nu se întâmplă în hardware-ul computerului, având în vedere latenta de memorie și alte aspecte ale arhitecturii. O scriere de la procesor P1 nu poate fi văzuta de o citire de la procesor P2 dacă citirea se face într-un timp foarte mic după ce scrierea a fost făcută.

Definiția alternativă a unui sistem coerent este prin definirea modelului de memorie cu coerenta secvențială: "sistemul coerent cache trebuie să apară ca executand sarcini și stocand toate firele de executie într-o singură locație de memorie într-o ordine totala care respectă ordinea de program a fiecarui fir " . Astfel, singura diferență dintre sistemul coerent cache și sistemul coerent secvențial este în numărul de locații de adresă despre care se vorbeste in definiție.

O altă definiție este:. "Un multiprocesor are coerenta cache dacă toate scrierile in aceeași locație de memorie sunt efectuate într-o ordine secvențială"

Multiprocesoarele cu memorie partajata au aparut cu o modalitate deosebit de eficienta, din punct de vedere al costului, de a furniza o putere si viteza de calcul ridicate, datorate folosirii unor microprocesoare low-cost interconectate cu modul de memorie comun. [5]

*Memorie*:Deoarece un modul de memorie nu poate efectua mai mult de o cerere ce necesita memorie in acelasi timp, mai multe cereri de la diferite procesoare for fi serializate. [5]

*Comunicare:*Continutul pentru link-uri individuale din reteaua interconectata poate rezulta chiar daca cererile sunt directionate catre un alt modul de memorie. [5]

*Timp de latenta:*Multiprocesoarele cu un numar mare de procesoare tind sa aibe interconexiuni de retea complexe. Timpul de latenta pentru astfel de retele (timpul necesar unei cereri sa traverseze reteaua) este mare.[5]

 **3.3 Mecansime**

 *Bazate pe director*

Într-un sistem bazat pe director, datele fiind partajate sunt plasate într-un director comun care menține coerența între memoriile cache. Directorul acționează ca un filtru prin care procesorul trebuie să ceară permisiunea pentru a încărca o intrare din memoria principală in cache. Atunci când o intrare este modificată, directorul fie actualizează sau invalidează celelalte memorii cache cu acea intrare.

 *Snooping*

Introdus pentru prima dată în 1983, Snooping este un proces în care memoriile cache individuale monitorizeaza linii de adresa pentru căi de acces la locații de memorie pe care le-au trimis in cache. Este apelat un protocol de invalidare a scrierii atunci când se observă o operație de scriere într-o locație a careia o memorie cache are o copie și controlerul cache invalidează propria copie a locației de memorie.

 *Snarfing*

Este un mecanism in care un controler cache urmareste atât adresa cat și datele într-o încercare de a actualiza propria copie a unei locații de memorie, atunci când un al doilea master modifică o locație în memoria principală. Când se observă o operație de scriere într-o locație a carei copie o are o memorie cache, operatorul cache actualizează propria copie a locației de memorie cu noile date.

Sistemele distribuite cu memorie partajată imita aceste mecanisme în încercarea de a menține coerența între blocurile de memorie în sistemele cuplate slab.

Cele două mecanisme mai comune de asigurare a coerenței sunt snooping și cel pe bază de director, fiecare având propriile sale avantaje și dezavantaje. Protocoale Snooping tind să fie mai rapide, în cazul în care este disponibila destula latime de banda, deoarece toate tranzacțiile sunt vazute de toate procesoarele ca un cuplu cerere/raspuns. Dezavantajul este că Snooping nu este scalabil. Fiecare cerere trebuie să fie difuzata la toate nodurile din sistem, ceea ce înseamnă că, sistemul devine mai mare, dimensiunea bus-ului (logic sau fizic) și lățimea de bandă pe care le oferă trebuie să crească. Directoarele, pe de altă parte, tind să aibă latențe mai lungi, dar folosesc mult mai puțina latime de banda, deoarece mesajele sunt punct la punct și nu difuze. Din acest motiv, multe dintre sistemele mai mari (> 64 procesoare) folosesc acest tip de memorie cache coerenta.

Pentru mecanismul de snooping, un filtru Snoop reduce traficul de snooping prin menținerea a o multitudine de intrări, fiecare reprezentând o linie cache care poate fi deținută de unul sau mai multe noduri. Atunci când este necesară înlocuirea uneia dintre intrari, filtrul snoop selectează pentru înlocuire intrarea care reprezinta linia cache sau liniile deținute de cele mai puține noduri, determinate printr-un vector de prezență în fiecare dintre intrăr. Un tip temporal sau alt tip de algoritm este folosit pentru a rafina selecția dacă mai mult de o linie cache este deținută de cel mai mic număr de noduri.

**3.4 Protocoale [5]**

Protocoalele bazate pe hardware includ protocoale cache Snoopy,directory schemes, si arhitecturi cache-coherent network. Toate se bazeaza pe un anumit protocol de coerenta cache.

Protocoalele de coerenta a memoriei cache sunt protocoalele bazate pe hardware pentru mentinerea coerentei cache care garanteaza coerenta memoriei sistemului fara mecanismele software implementate. Mecanismele hardware detecteaza conditii de inconsistenta si efectueaza actiuni conform unui protocol hardware implementat. Datele sunt descompuse intr-un numar egal de blocuri. Un bloc este unitatea de transfer dintre memorie si cache.Protocoalele hardware permit un numar arbitrar de copii ale unui bloc sa existe simultan. Exista doua protocoale pentru mentinerea consistentei cache ( write invalidate si write-update):

-Protocoale de cache Wriste-invalidate snoopy.

-Protocolul write-once asociaza o stare cu fiecare copie a unui bloc din cache.

Starile posibile pentru o copie sunt:

* *Invalid*. Copie este inconsistent.
* *Valid.* Exista o copie valida consistent cu copia din memorie.
* *Rezervata.* Datele au fost scrie exact o data si copia este consistent cu copia memoriei, care este singura copie.
* *Corupt*. Datele au fost modificare de mai multe ori si copia este singura din sistem. Write-once foloseste protocolul memoriei copy-back actualizata, ceea ce inseamna ca intreaga copie a blocului trebuie scrisa inapoi in memorie cand este inlocuita, cu conditia ca a fost modificata in timpul de stocare in cache. Pentru a mentine consistent, protocolul necesita urmatoarele comenzi de consistent pe langa blocul normal de citire a memoriei(Read-Blk) si blocare scriere (WriteBlk):

-Write-Inv: invalideaza toate celelalte copii.

-Read-Inv: citeste un bloc

Tranzitiile de stare rezulta ori din citirea procesorului local sau scrierea comenzilor.

(P-Read si P-Write) sau comenzile de consistent (Read-Blk, Write-Blk, WriteInv, si Read-Inv) primite de la bus-ul global.

Operatia protocolului poate fi specificata prin claritatea actiunilor luate asupra citirilor si scrierilor procesorului. Citirile reusite sunt intotdeauna realizate la nivel local in cache si nu duc la stari de tranzitie. Pentru citirile ratate, scrierea este realizata , si scrierea rateaza actiunile ce apar astfel:

* + *Citire rateaza*.Daca nu exista nicio copie corupta, atunci memoria are o copie consistent si ofera o copie cacheu-lui. Aceasta copie va fi in stare valida. Daca o copie corupta exista, atunci memoria cache corespunzatoare inhiba si trimite o copie cacheului solicitant. Ambele copii vor devein valide si memoria este actualizata.
	+ *Scriere realizata.* Daca copia este corupta sau in starile rezervate, atunci scrierea poate fi efectuata local si noua stare este corupta.Daca starea este valida, atunci o comanda de consistenta Write-Inv este distribuita la toate cache-urile, invalidand copiile lor. Copia memoriei este actualizata si starea rezultata este rezervata.
	+ *Scriere ratata.* Copia provine de la un cache cu o copie corupta, care actualizeaza apoi memoria, sau poate proveni din memorie. Acestea sunt realizate prin trimiterea unei comenzi de consistent Read-Inv, care invalideaza toate copiile din cache. Copia este actualizata local si starea rezultata este corupta.
	+ *Inlocuirea.* Daca copia este corupta, atunci trebuie scrisa in memoria principal. Altfel, nu se efectueaza nicio actiune.

 **Protocoale cache wirte-update snoopy.**

*Solutie Hardware*

Un exemplu de protocol write-update este protocolul Firefly, care a fost implementat in statia de lucru multiprocessor Firefly de la Digital Equipment. Asocia trei stari posibile cu o copie cache a blocului:

* Valid-exclusive. Singura copie din cache, este consistent cu copia memoriei.
* Shared. Copia este consistenta si sunt si alte copii consistente.
* Dirty(corrupt) Este singura copie. Memoria copiei este inconsistenta.

Protocolul Firefly foloseste protocolul actualizat copy-back pentru blocari private si write-through pentru blocuri commune. Notiune de comun si privat este determinat la timpul de rulare.

Pentru a mentine consistent, o comanda de consistent write-update acutalizeaza toate copiile. O linie “shared line” este folosita de mecanismul de cautare pentru a informa scriitorul de existent copiei.

**Probleme de implementare si performanta pentru protocoalele de cautare cache.**

(Snoopy cache protocols)Protocoalele de cautare cache sunt extrem de populare pentru usurinta implementarii.

 Principalele diferente dintre un cache de cautare si un cache uniprocesor sunt controlerele cache, informatia stocata in directorul cache si controlerul bus.

*Solutie Hardware*

 Schemele de coerenta cache software incearca sa evite necesitatea unui mecanism hardware complex.

 Protocoalele bazate pe hardware efectiv reduc traficul de retea. Insa, platim pentru acest lucru cu un mecanism hardware complex, in special pentru multiprocesoare cu un numar mare de procesoare.

*Solutie Software*

 O alternativa ar preveni existenta inconsistentei datelor cache prin limitarea structurarii datelor cache la momente sigure . Acest lucru face necesara analizarea programului pentru a marca variabilele ca cacheable sau non-cacheable, ceea ce un compilator sau un preprocesor mai sofisticat poate face. Cea mai triviala solutie ar fi marcarea tuturor variabilelor de citire-scriere ca non-cacheable. Acest lucru este prea conservativ, din moment ce structura datelor comune poate fi accesata de un proces sau este read-only pe o durata considerabila de timp. In acest timp nu exista vreun pericol in a pune o structura de date in cache.

 O abordare mai buna ar permite compilatorului sa analizeze cand este sigur sa salveze in cache o variabila de citire-scriere comuna. In acest timp variabila este marcata drept cacheable. Dupa acestea, memoria principala trebuie sa contina datele cache, si datele cache trebuie sa fie inaccesibile din cache prin invalidare.

 **4. Bibliografie**

[1] <http://141.85.107.10/Laboratoare/_Asc/L03/MemoriaCache.htm>

[2]  Michael E. Thomadakis ["The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms"](http://sc.tamu.edu/systems/eos/nehalem.pdf) Texas A&M University.

<http://sc.tamu.edu/systems/eos/nehalem.pdf>

[3] Neupane, Mahesh ["Cache Coherence"](https://web.archive.org/web/20100620091706/http%3A/cse.csusb.edu/schubert/tutorials/csci610/w04/MN_Cache_Coherence.pdf)

[https://web.archive.org/web/20100620091706/http://cse.csusb.edu/schubert/tutorials/csci610/w04/MN\_Cache\_Coherence.pdf](https://web.archive.org/web/20100620091706/http%3A/cse.csusb.edu/schubert/tutorials/csci610/w04/MN_Cache_Coherence.pdf%20)

[4Patterson & Hennessy, Computer Architecture: A Quantitative Approach Fifth Edition, 2012 Elsevier, Inc <http://www.cpp.edu/~kding/materials/Computer%20Architecture%20A%20Quantitative%20Approach%20%285th%20edition%29.pdf>

[5]Per Stenstrom **,“**A Survey of Cache Coherence- Schemes for Multiprocessors”, Lund University

<http://web.mit.edu/6.173/www/currentsemester/readings/R04-cache-coherence-survey-1990.pdf>