ARHITECTURA SISTEMULUI IERARHIZAT DE MEMORIE

MEMORII CACHE

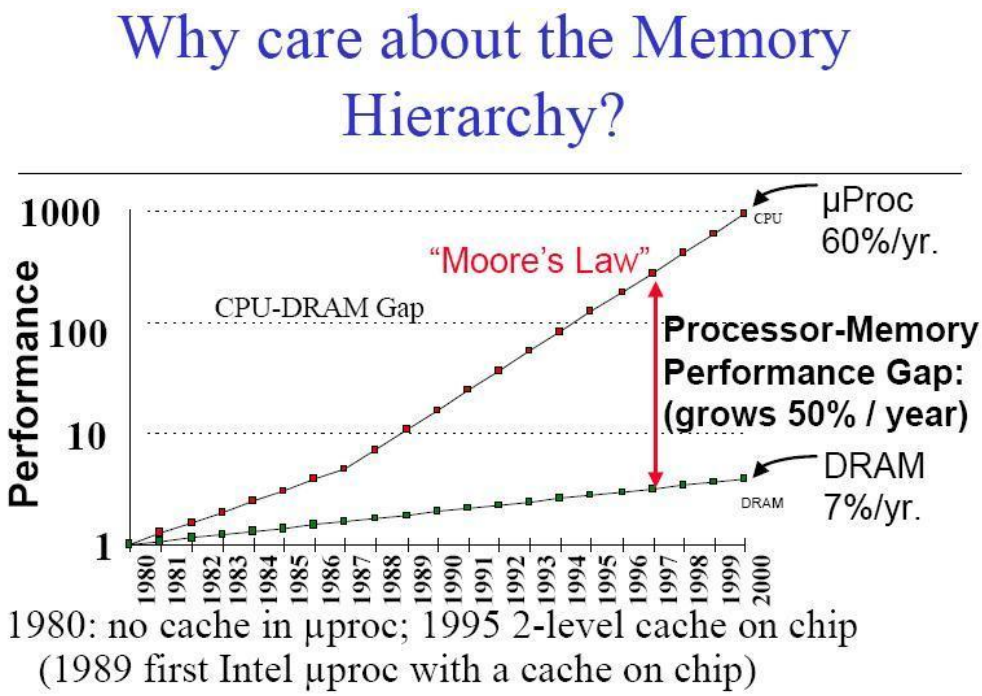
Cache: *a safe place for hiding or storing things. (Webster's New World Dictionary of the American Language)*

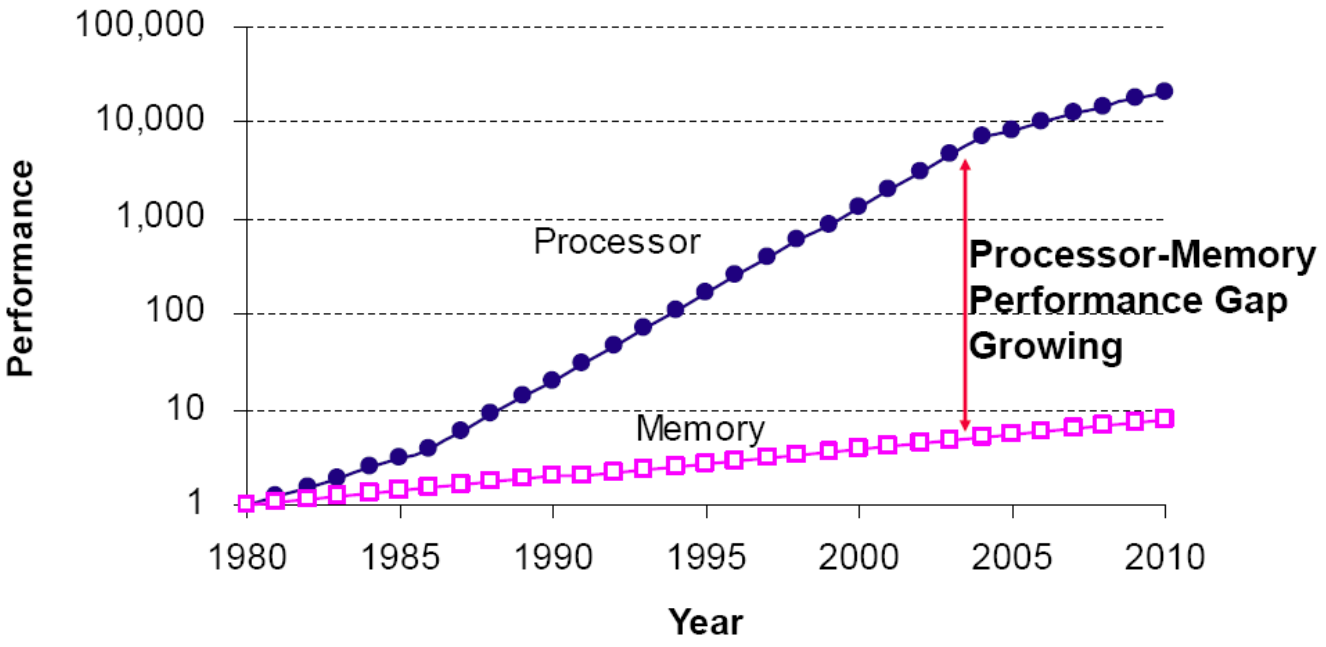
http://www.ecs.umass.edu/ece/koren/architecture/

Memoriile cache reprezintă un **mecanism omniprezent în microprocesoarele curente, dedicat mascării latenţei ridicate a memoriei principale**. Datorită importanţei lor, acestea sunt considerate elemente fundamentale în programa specifică arhitecturii calculatoarelor.

Conceptul de memorie cache a fost introdus de prof. ***Maurice Wilkes*** (Univ. Cambridge, Anglia) – un pionier al calculatoarelor. **Primul sistem comercial** care utiliza cache-urile a fost **IBM 360/85** (**1968**). Conceptul de cache s-a dovedit a fi **foarte fecund nu numai în hardware dar şi în software** prin aplicaţii dintre cele mai diverse **în sistemele de operare** (memoria virtuală), **reţele de calculatoare** (de îndată ce a aflat o pereche de adrese simbolică-numerică, un calculator o păstrează într-un cache local)**, baze de date** (*MySQL va ţine în cache rezultatul unei interogări, iar la primirea aceleiaşi interogări va returna rezultatul din cache, fără a interoga tabelele în sine*), **browser**-e (*Firefox face cache la imagini, ca la un refresh să nu fie nevoit să le preia de pe server din nou*), chiar şi *Google deţine propriul cache din care poate furniza conţinutul paginilor web*.

Necesitatea memoriilor Cache





- **Microprocesorul** este responsabil cu aducerea din memorie, decodificarea şi execuţia instrucţiunilor maşină, codificate binar.
- Începând cu decada anterioară sistemelor multicore şi până la nivelul anilor 2005, performanţa relativă a microprocesoarelor a crescut cu cca. **60% pe an**.
- În cazul memoriilor DRAM timpul de acces aferent scade cu 33% pe decadă. $\mathbf{T_{accesDRAM}}$ **= 5÷40ns (2008)**.

Microprocessor	16-bit address/ bus, microcoded	32-bit address/ bus, microcoded	5-stage pipeline, on-chip I & D caches, FPU	2-way superscalar, 64-bit bus	Out-of-order 3-way superscalar	Out-of-order superpipelined, on-chip L2 cache	Multicore OOO 4-way on chip L3 cache, Turbo
Product	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4	Intel Core i7
Year	1982	1985	1989	1993	1997	2001	2010
Die size (mm^2)	47	43	81	90	308	217	240
Transistors	134,000	275,000	1,200,000	3,100,000	5,500,000	42,000,000	1,170,000,000
Processors/chip	1	1	1	1	1	1	4
Pins	68	132	168	273	387	423	1366
Latency (clocks)	6	5	5	5	10	22	14
Bus width (bits)	16	32	32	64	64	64	196
Clock rate (MHz)	12.5	16	25	66	200	1500	3333
Bandwidth (MIPS)	2	6	25	132	600	4500	50,000
Latency (ns)	320	313	200	76	50	15	4
Memory module	DRAM	Page mode DRAM	Fast page mode DRAM	Fast page mode DRAM	Synchronous DRAM	Double data rate SDRAM	DDR3 SDRAM
Module width (bits)	16	16	32	64	64	64	64
Year	1980	1983	1986	1993	1997	2000	2010
Mbits/DRAM chip	0.06	0.25	1	16	64	256	2048
Die size (mm^2)	35	45	70	130	170	204	50
Pins/DRAM chip	16	16	18	20	54	66	134
Bandwidth (MBytes/s)	13	40	160	267	640	1600	16,000
Latency (ns)	225	170	125	75	62	52	37

Performance milestones over 25 to 40 years for microprocessors, memory

Memoria cache este o **memorie situată din punct de vedere logic între CPU şi memoria principală** (uzual DRAM), **mai mică, mai rapidă şi mai scumpă** (per byte – de tip SRAM) decât aceasta şi **gestionată** – în general **prin hardware** – **astfel încât să existe o cât mai mare probabilitate statistică de găsire a datei accesate de către CPU, în cache**.

Cache-ul este adresat de către CPU în paralel cu memoria principală (MP): dacă data dorită a fi accesată se găseşte în cache, accesul la MP se abortează, dacă nu, se accesează MP cu penalizările de timp impuse de latenţa mai mare a acesteia, relativ ridicată în comparaţie cu frecvenţa de tact a CPU. Data accesată din MP se va introduce şi în cache.

Timpul de acces DRAM – 20 ÷ 50 (chiar şi **500** în anumite lucrări de cercetare) de impulsuri de tact (T_{CLK} = perioada ceasului microprocesorului), în schimb **accesarea cache-ului de nivel 1 (L1)** se face în doar 1 – 3 T_{CLK}, iar accesarea cache-ului de nivel 2 **(L2)** se face în doar 12 – 20 T_{CLK} ⇒ **memoria cache reduce timpul mediu de acces al CPU la MP**, ceea ce este foarte util.

Se defineşte un acces al CPU cu **hit** în cache, *un acces care găseşte o copie în cache a datei accesate*. Un acces cu **miss** în cache este unul care *nu găseşte* o copie în cache a datei accesate de către CPU şi care, prin urmare, adresează MP cu toate penalizările de timp care derivă din accesarea acesteia.

Parametru de performanţă al unei memorii cache **rata de hit**, ca fiind raportul statistic între numărul acceselor cu hit în cache respectiv numărul total al acceselor CPU la memorie. Măsurat pe benchmark-uri (programe de test) reprezentative, la ora actuală sunt **frecvente rate de hit de peste 90 %**. **Rata de miss** (RM) este complementara ratei de hit (RH), astfel că: **RH [%] + RM [%] = 100 %**. În realitate, în cazul unei citiri cu miss **în cache se aduce din MP nu doar data (cuvântul) dorită de către CPU ci un întreg bloc (4 – 16 cuvinte) care** evident **conţine data respectivă**. O citire cu miss presupune aducerea blocului din MP dar înainte de aceasta se impune evacuarea în MP a unui bloc din cache. Transferul din cache în MP se face tot la nivel de bloc şi nu de cuvânt ⇒ **Optimizarea traficului dintre cache şi MP**.

Metrici:

- **Rata de Hit** = raportul statistic între numărul acceselor cu hit (gasesc informatia cautata) în cache respectiv numărul total al acceselor CPU la memorie.

  Rata de Hit=1 – Rata de miss
- **Timpul mediu de acces al CPU la sistemul ierarhic de memorie (MP) =**

  = T_acces_cache*R_hit + T_acces_$_{DRAM}$*(1-R_hit)
- **Rata de procesare** (viteza de executie) = Nr. instructiuni executate / Timp total executie [IPC] (*instruction per cycle*)

Eficienţa memoriilor cache se bazează pe **2 principii de natură statistică** şi **care caracterizează intrinsec noţiunea de program**: principiile de **localitate temporală** şi **spaţială**. Conform principiului de localitate (vecinătate) temporală, există o mare probabilitate ca o dată (**instrucţiune**) accesată acum de către CPU să fie accesată din nou, în viitorul imediat.

C		*MIPS*	
suma=0;			
for (j=1000; j>0; j--)		la $2, Array_A	# $2 – adresa de inceput a vectorului *a*
suma+=a[j];		li $6, 0	# $6 – *suma*
		li $4, 1000	# $4 – contorul *j*
	loop:	**lw $5, 0($2)**	# $5 – elementul curent al vectorului a[*j*]
		add $6, $6, $5	# suma+=a[j]
		add $2, $2, 4	
		addi $4, $4, -1	# j--
		bgt $4, $0, loop	# j>0

Conform principiului de localitate spaţială, există o mare probabilitate ca o dată situată în imediata vecinătate a unei date accesate curent de către CPU, să fie şi ea accesată în viitorul apropiat. Ex: **bucle de program, structuri regulate de date** – tablouri, matrici. Pe baza acestui principiu se aduce din **MP** un bloc de date (**a[0], a[1]**) si nu doar data (**a[0]**) care cuprinde cuvantul solicitat de CPU.

O combinare a celor 2 principii conduce la celebra "**regulă 90/10**": „**cca. 90% din timpul de rulare al unui program se execută doar 10% din codul acestuia**".

Memorii Harvard – spatii si busuri separate pentru instructiuni si date **vs. Memorii Princeton** (unificate). Avantaje Harvard – (reduce coliziunile in proc. pipeline – IF/MEM). Dezavantaje Harvard (rata de hit mai mica).

D.p.d.v. **arhitectural**, **există 3 tipuri distincte de memorii cache** în conformitate cu **gradul de asociativitate: cu mapare directă, semiasociative şi total asociative**.

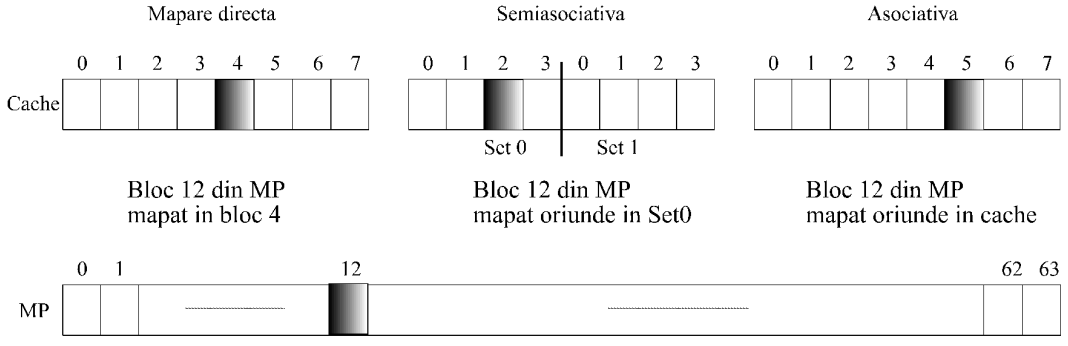


Figura 2.1. Scheme de mapare în cache

La **cache-urile cu mapare directă** – un bloc din MP poate fi găsit în cache (hit) într-un bloc unic determinat. **Regula de mapare a unui bloc din MP în cache** este:

Index_bloc_in cache = [(**Adresa bloc MP**) div **dimensiunea_bloc_in_bytes**[1]] modulo (**Nr. blocuri din cache**)

/ **%**

Tag_Emis **= (Adresa bloc MP / dimensiunea_bloc_in_bytes) / Nr. blocuri din cache** sau

Tag_Emis **= Adresa bloc MP / Dimensiunea cache-ului =>**

Stricteţea regulii de mapare conduce la o simplitate constructivă (accesul se face folosind ultimii biti de adresa) a acestor memorii **dar şi la fenomenul de interferenţă** al blocurilor din MP în cache. **Exemplu**, blocurile 12, 20, 28, 36 determină **interferenţe în cache**. Accesarea alternativă, în mod repetat a două blocuri din MP conflictuale în cache, determină o rată de hit egală cu **0**.

La **cache-urile semiasociative** există mai multe seturi, fiecare set având mai multe blocuri componente. **Blocul dorit se poate mapa oriunde în setul respectiv**. **Regula de mapare precizează strict doar setul** în care se poate afla blocul dorit:

Index_bloc_in cache = (**Adresa bloc MP / dimensiunea_bloc_in_bytes**) modulo (**Nr. seturi din cache**)

Tag_Emis **= (Adresa bloc MP / dimensiunea_bloc_in_bytes) / Nr. seturi din cache**

Mai precis, **la un miss în cache, înainte de încărcarea noului bloc din MP, trebuie evacuat un anumit bloc din setul respectiv**. Există implementate două-trei tipuri de algoritmi de evacuare: **pseudorandom** (cvasialeator), **FIFO şi LRU** ("***Least Recently Used***"). **Algoritmul LRU evacuează blocul din cache cel mai demult neaccesat, în baza principiului de localitate temporală**.

Dacă un set din cache-ul semiasociativ conţine N blocuri atunci cache-ul se mai numeşte "**tip N-way set associative**". Într-un astfel de cache **rata de interferenţă se reduce odată cu creşterea gradului de asociativitate** (N). Exemplu, blocurile 12, 20, 28 şi 36 pot coexista în setul 0.

- (N) ⇒ interferenţele blocurilor ⇒ performanţa globală (IR) .

[1] In bytes sau locatii

- asociativitatea impune **căutarea după conţinut** (se caută deci într-un set dacă există memorat blocul respectiv) (N) ⇒ complexitatea structurală ⇒ Timpul de acces la cache ⇒ performanţa globală (IR) .

Optimizarea gradului de asociativitate, a capacităţii cache-ului, a lungimii blocului din cache se face prin **laborioase simulări software**, **variind** toţi aceşti **parametrii** în vederea **maximizării ratei globale de procesare** a instrucţiunilor [instr./cicli].

În principiu blocul dorit se poate mapa oriunde în setul respectiv. Mai precis, la un miss în cache, înainte de încărcarea noului bloc din MP, trebuie evacuat un anumit bloc din setul respectiv. În principiu, în mod uzual, există implementate două-trei tipuri de algoritmi de evacuare: pseudorandom (cvasialeator, uşor de implementat), FIFO (sau *round-robin*, se evacuează blocul cel mai vechi din cache. Contorul aferent se încarcă doar la încărcarea blocului în cache şi nu la fiecare hit per bloc, ca la algoritmul LRU) şi LRU ("*Least Recently Used*"). **Algoritmul LRU evacuează blocul din cache cel mai de demult neaccesat, în baza principiului de localitate temporală (aflat oarecum în contradicţie cu o probabilistică markoviană care ar sugera să fie păstrat!).** În practică, implementările FIFO, LRU sunt simplificate şi deci aproximative. Deşi acest model pare intuitiv corect, el poate genera şi rezultate eronate uneori. De exemplu, numărul total de accese cu miss poate uneori să crească când creşte asociativitatea, iar politica de înlocuire LRU este departe de a fi cea optimă pentru unele din programe. Această anomalie poate fi evitată prin folosirea algoritmului "optim" (OPT) în loc de LRU ca bază pentru clasificarea miss-urilor în cache. **Algoritmul OPT, înlocuieşte întotdeauna blocul care va fi adresat cel mai târziu în viitor (eventual nu va mai fi adresat deloc).** Un astfel de algoritm s-a dovedit a fi cvasi-optimal pentru toate pattern-urile de program, ratele de miss fiind cele mai mici în acest caz, dintre mai toate politicile de înlocuire folosite. Politica se dovedeşte optimă doar pentru fluxuri de instrucţiuni *read-only*. Pentru cache-urile cu modalitate de scriere *write-back* algoritmul de înlocuire nu este întotdeauna optim (spre exemplu poate fi mai costisitor să se înlocuiască blocul cel mai târziu referit în viitor dacă blocul trebuie scris şi în memoria principală, fiind "*murdar*", faţă de un bloc "*curat*" referit în viitor puţin mai devreme decât blocul "*murdar*" anterior; în plus, blocul curat nu mai trebuie evacuat, ci doar supra-scris). Este evident un **algoritm speculativ**, **practic imposibil de implementat** în practică. Totuşi el are **două calităţi majore**: (1) reprezintă o metrică de evaluare teoretică a eficenţei algoritmului de evacuare implementat în realitate, absolut necesară şi (2) induce ideea predictibilităţii valorilor de folosinţă ale blocurilor din cache, conducând astfel la **algoritmi predictivi de evacuare** (memoria SVC implementează un astfel de algoritm – care anticipează pe baze euristice utilitatea de viitor a blocurilor memorate în cache, evacuându-l pe cel mai puţin valoros).

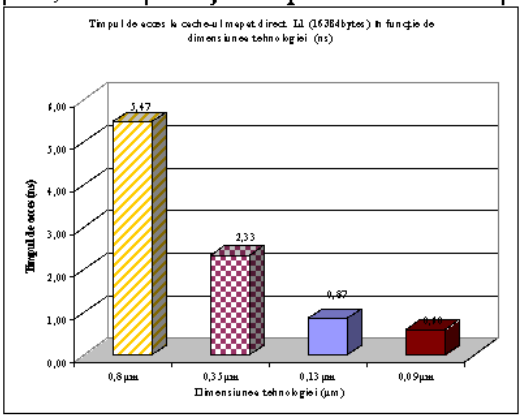
1st JILP Workshop on Computer Architecture Competitions (JWAC-1): Cache Replacement Championship (http://www.jilp.org/jwac-1/JWAC-1%20Program.htm)

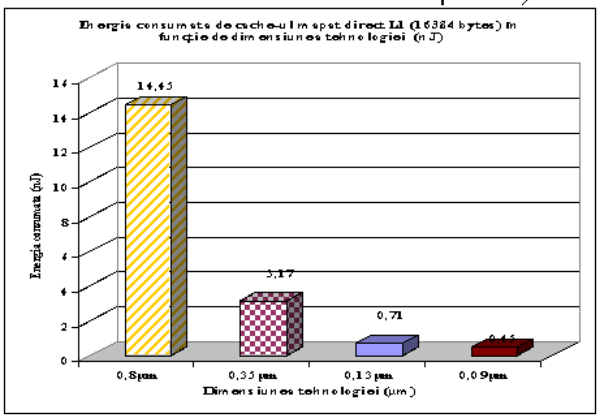
Memoriile **cache complet asociative**, implementează practic **un singur set permiţând maparea blocului practic oriunde în cache**. Ele **nu se implementează** deocamdată **în siliciu datorită complexităţii deosebite şi a timpului prohibit de căutare**. Reduc însă total interferenţele blocurilor la aceeaşi locaţie cache.

Ierarhie: ***Rata de hit*** (crescător) şi ***Complexitate*** (crescător) **Timp de acces** (crescător)	**Cache DM**	**Cache Semiasociativ**	**Cache complet asociativ**

Dimensiunea tehnologiei

- Reprezintă o caracteristică importantă în creşterea performanţei cache-ului.
- Parametrul dimensiunea tehnologiei exprimă dimensiunea unei porţi logice (diferenţa în microni dintre ieşirea şi intrarea unei porţi logice fundamentale NAND). În unele documentaţii este interpretată ca fiind *distanţa dintre două intrări aferente porţii logice.*
- Am testat CACTI 3.0 variind dimensiunea tehnologiei de la 0.8µm, la 0.35 µm, 0.13µm şi respectiv la 0.09µm (ex. cacti 16384 32 DM 0.35µm 2)





Dimensiunea tehnologiei	0,8µm	0,35µm	0,13µm	0,09µm
Timpul de acces (ns)	5,47	2,33	0,87	0,60
Procesor Intel Pentium 4	Northwood 1,6 GHz	PSATSim - Cache L1 – 16KBytes DM	Northwood Hyperthread (2GHz şi 3,4GHz)	Extrem Edition Hyperthread (2,8 - 3,4GHz)
Recovery	20 de cicli de tact		20 de cicli de tact	32 cicli de tact
Tranzistori/Cip	42 milioane		55 milioane	178 milioane

Access time estimate for 90 nm using CACTI model 4.0 – Median ratios of access time relative to the direct-mapped caches are 1.32, 1.39, and 1.43 for 2-way, 4-way, and 8-way caches

TEMĂ:

1. Utilizând simulatorul CACTI (http://quid.hpl.hp.com:9081/cacti/) ilustraţi grafic influenţa tehnologiei asupra: *tensiunii de alimentare*, *timpului de acces la cache* şi *energiei disipate*. Interpretaţi rezultatele.
2. Utilizând simulatorul CACTI ilustraţi grafic influenţa dimensiunii cache-ului asupra: *timpului de acces la cache* şi *energiei disipate*. Interpretaţi rezultatele.
3. Folosind acelaşi simulator determinaţi influenţa gradului de asociativitate al cache-ului asupra: *timpului de acces la cache* şi *energiei disipate*. Interpretaţi rezultatele.
4. Utilizând simulatorul CACTI ilustraţi grafic influenţa dimensiunii blocului din cache asupra: *timpului de acces la cache* şi *energiei disipate*. Ce observaţi.

Cache direct mapat

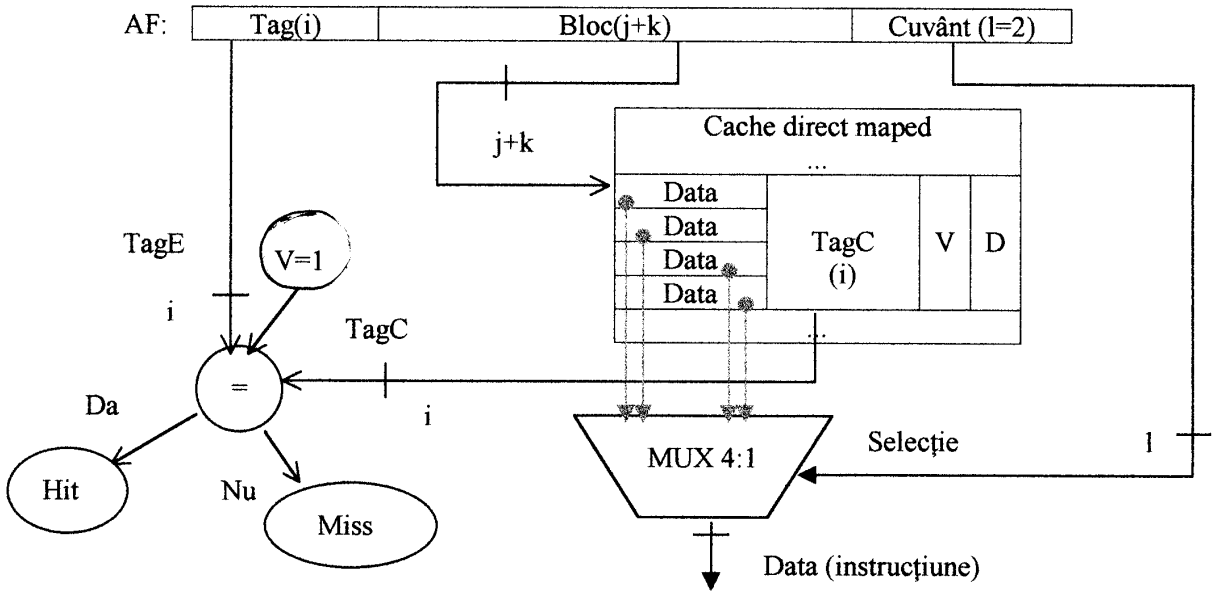


Figura 2.2. Implementarea unui cache cu mapare directă

Cache semiasociativ pe 2 căi

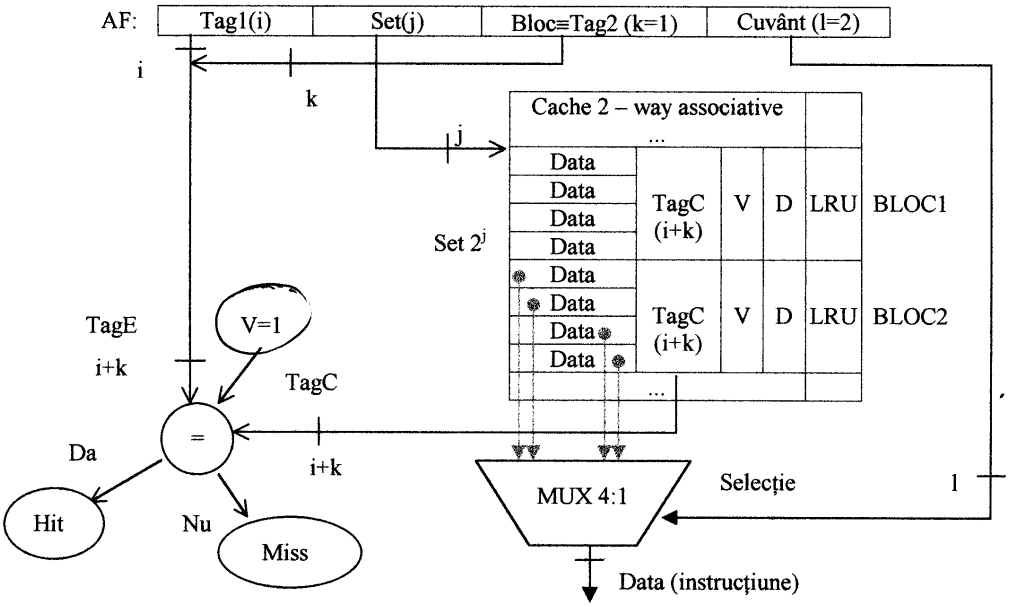


Figura 2.3. Implementarea unui cache semiasociativ (2-way)

Cache complet associativ

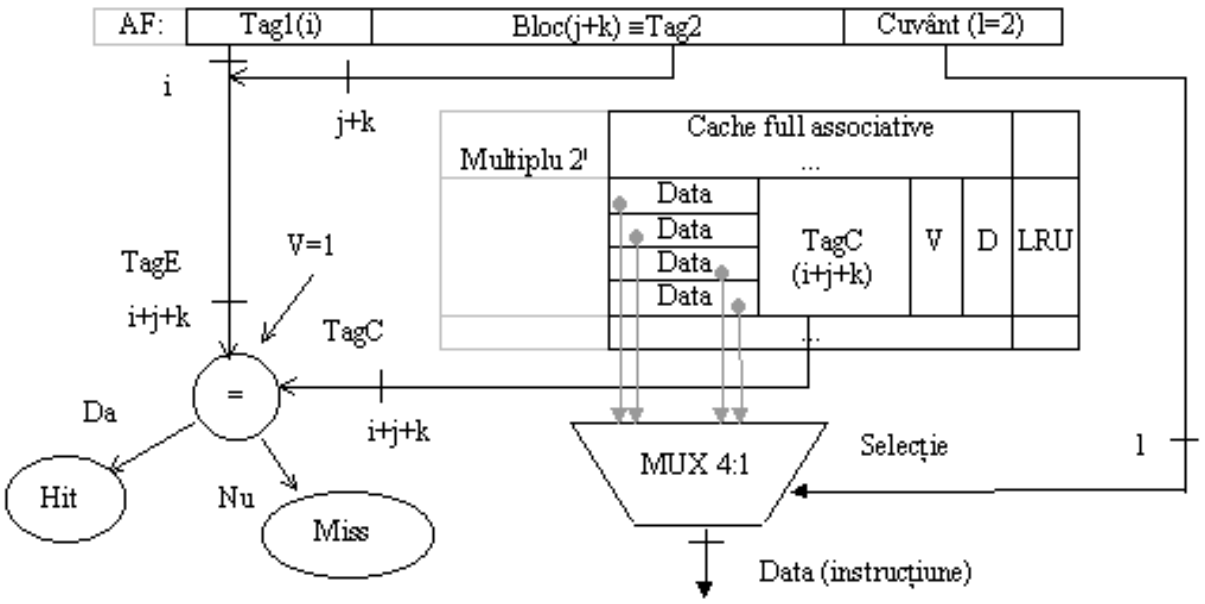


Figura 2.4. Implementarea unui cache complet asociativ

Blocul este compus din **4 cuvinte**. Bitul **V** este un **bit de validare a blocului**, V=1 fiind o condiţie necesară a obţinerii hitului. Pentru a evita false hituri la citirea programului încărcător din EPROM, se iniţializează biţii V cu zero. La prima încărcare a unei date (instrucţiuni) în cache, bitul V aferent se va seta pe '1', validând astfel hitul.

Bitul D (Dirty) este pus pe '0' la încărcarea iniţială a blocului în cache. La prima scriere a acelui bloc, bitul se pune deci pe '1'. Evacuarea propriu-zisă a blocului se face doar dacă bitul D=1. Prin acest bit **se minimizează evacuările de blocuri în MP, pe baza principiului că un bloc trebuie evacuat numai dacă a fost scris în cache**.

Consider the following C declaration which is part of a small cache simulator.

```
typedef struct {
     char valid;
     char dirty;
     int tag;
     char block[32];
     int LRU;
    } cache_line;

char main_memory[1<<16];          /* array representing main memory */
typedef cache_line cache_set[4];   /* set of cache lines */
cache_set cache[16];               /* cache, uninitialized */
```

D.p.d.v. al **acceselor de scriere a unui procesor**, există 2 posibilităţi:

a) Strategia "**Write Through" (WT)**, prin care informaţia este scrisă de către procesor atât în blocul aferent din cache cât şi în blocul corespunzător din memoria principală.
b) Strategia **"Write - Back" (WB)**, prin care informaţia este scrisă numai în cache, blocul modificat fiind transferat în MP numai la evacuarea din cache.

Coerenţa memoriilor cache este o problemă care afectează arhitecturile multiprocesor, cu memorie partajată (SMA – Shared Memory Architectures).

În cazul arhitecturilor uniprocesor această problemă nu apare deoarece există un singur procesor care să citească şi să scrie date din / în memorie. În plus, se

poate face o singură operaţie asupra memoriei la un moment dat, astfel că, atunci când o locaţie din memorie se schimbă, toate operaţiile următoare, care implică citirea acelei locaţii de memorie, vor accesa valoarea corect modificată.

În sistemele multiprocesor există două sau mai multe procesoare care lucrează în paralel, existând deci posibilitatea ca o locaţie de memorie să fie accesată în acelaşi moment de timp, de mai multe procesoare. Atât timp cât acea locaţie de memorie este accesată doar pentru citire (niciun procesor nu o modifică), partajarea ei se poate face fără probleme. Dar, atunci când valoarea este modificată de un procesor, există riscul ca celelalte procesoare să lucreze cu o copie veche, invalidă, a locaţiei de memorie partajată. **Atunci când mai multe procesoare păstrează în cache-urile lor locale (private) copii ale unor locaţii dintr-o memorie partajată, orice modificare a unei astfel de locaţii, la nivel de cache (locală deci), poate cauza o inconsistenţă la nivelul global al memoriei partajate**.

În vederea menţinerii coerenţei cache-urilor cu precădere în sistemele multimicroprocesor – există 2 posibilităţi în funcţie de ce se întâmplă la o scriere:

a) **Write invalidate** – prin care CPU care scrie determină ca toate copiile din celelalte memorii cache să fie invalidate înainte ca el să-şi modifice blocul din cache-ul propriu.
b) **Write Broadcast** – CPU care scrie pune data de scris pe busul comun spre a fi actualizate toate copiile din celelalte cache-uri.

Ambele strategii de menţinere a coerenţei pot fi asociate cu oricare dintre protocoalele de scriere (WT, WB) dar **de cele mai multe ori se preferă WB cu invalidare**.

Se va considera un exemplu care arată cum **2 procesoare pot "vedea" 2 valori diferite pentru aceeaşi locaţie (X) de memorie globală**, adică un caz tipic de **incoerenţă** a unei valori globale.

Pas	Eveniment	Conţinut cache CPU1	Conţinut cache CPU2	Conţinut Memorie globală (X)
0	_	?	?	1
1	CPU1 citeşte X	1	?	1
2	CPU2 citeşte X	1	1	1

3	CPU1 scrie 0 în X (WT)	0	1	0 (WT)
	WB	0	1	1 (WB)

Exemplificarea unei incoerențe

S-a presupus că iniţial, nici una din cele 2 cache-uri nu conţine variabila globală X şi că aceasta are valoare 1 în memoria globală. De asemenea s-au presupus scrieri de tip WT în cache-uri (o scriere de tip WB ar introduce o incoerenţă asemănătoare). În pasul 3 CPU 2 are o valoare incoerentă a variabilei X.

În continuare, se prezintă un exemplu de **protocol de coerenţă WI**, bazat pe un protocol de **scriere în cache de tip WB**.

Pas	Activitate procesor	Activitate pe bus comun	Loc.X cache CPU1	Loc.X cache CPU2	Loc. X Memorie globală
0	_	_	?	?	0
1	CPU1 citeşte X	Cache Miss (X)	0	?	0
2	CPU2 citeşte X	Cache Miss (X)	0	0	0
3	CPU1 scrie ‘1’ în X	Invalidare X	1	0 INV.	0
4	CPU2 citeşte X	Cache Miss (X)	1	1	1

Coerenţa prin protocol WI

REZUMÂND, apar posibile 4 procese distincte într-un cache ca în tabelul următor:

Tip acces	Hit / Miss	Acţiune în cache
Citire	Miss	Evacuare bloc + încărcare bloc nou
Citire	Hit	(comparare tag-uri)
Scriere	Miss	(Evacuare bloc – Dirty=1) + încărcare bloc nou + scriere data în bloc (WB)
Scriere	Hit	Scriere data în blocul din cache (WB)

Tabelul 2.1. Tipuri de acces în cache

Memoriile cache îmbunătăţesc performanţa îndeosebi pe citirile cu hit iar în cazul utilizării scrierii tip “Write Back” şi pe scrierile cu hit. Îmbunătăţirea **accesului la memorie pe citirile CPU** este normală având în vedere că acestea **sunt**

mult mai frecvente decât scrierile (orice instrucţiune implică cel puţin o citire din memorie pentru aducerea sa; **statistic, cca. 75 % din accesele la memorie sunt citiri**).

Cu privire la accesele cu miss în cache **există trei tipuri de miss**:

***Compulsor*y** - sau de start rece. Apar la primul acces al unui anumit bloc în cache.

De ***capacitate*** - apar în situaţia în care cache-ul nu poate reţine toate blocurile necesare iar cele care sunt evacuate la un moment dat vor fi necesare ulterior.

De ***conflict*** **-** sau de interferenţă. Apar dacă strategia de plasare a blocurilor în cache este mapată direct sau semi-asociativă, situaţie în care mai multe blocuri din memoria principală pot interfera (accesa) acelaşi bloc din cache.

Metode de îmbunătăţire a performanţei cache-urilor:

1. **Reducerea ratei de miss în cache**

Creşterea dimensiunii blocurilor (din păcate cresc şi penalităţile de miss la evacuare-încărcare bloc)

Creşterea capacităţii cache (mărire timp acces hit şi costuri)

Creşterea gradului de asociativitate a cache-ului (creşte timp acces la hit)

Utilizarea mecanismelor de prefetch asupra instrucţiunilor (*buffer de prefetch*) şi datelor (*data write buffer*)

Optimizarea de programe prin *compilator*

- intrarea într-un *basic-block* să reprezinte începutul unui bloc în cache
- exploatarea localităţilor spaţiale ale datelor din cache – ***loop interchange***, *loop fusion*, *merging array* etc.

Exemplu optimizare a localitatilor spatiale:

```
for (j=0; j<1000; j++)
    for (i=0; i<2000; i++)
        A[i,j]=4*A[i,j];
```

Dezavantaj: pas (A[i,j]; A[i+1, j]) este 1000 sunt in blocuri diferite in cache **scade Rhit**

```
for (i=0; i<2000; i++)
  for (j=0; j<1000; j++)
        A[i,j]=4*A[i,j];
```

Avantaj: pas (A[i,j] si A[i, j+1]) este 1 sunt in acelasi bloc in cache **creste Rhit**

2. **Reducerea penalităţii în caz de miss în cache**

Niveluri multiple de cache-uri (*multi-level inclusion*, *exclusion*, hibrid);

Utilizarea conceptului de *victim cache* respectiv *selective victim cache* (a se vedea capitolul 5 – SOAC)

L2-Cache. Aspecte calitative

- L1 Read_Miss – aduce bloc din L2 in L1si bloc L1 evacueaza in MP (**Multilevel Inclusion**) SAU Swap_L2&L1 (**Multilevel Exclusion**)
- L1 Write_Hit – Write_Back→(**Multilevel Exclusion**); Write_Through →(**Multilevel Inclusion**)
- L1 Write_Miss – (**Multilevel Exclusion**) aduce bloc din L2/MP si scrie in L1 (WB); (**Multilevel Inclusion**) aduce bloc L2/MP→L1 si scrie in ambele (WT).

•Multilevel Inclusion → L2 mare (redundanta info)
•Multilevel Exclusion → L2 mai mic

Multilevel Caches: Design of L2

- Size
  - Since everything in L1 cache is likely to be in L2 cache, L2 cache should be much bigger than L1
- Whether data in L1 is in L2
  - novice approach: design L1 and L2 independently
  - multilevel inclusion: L1 data are always present in L2
    - Advantage: easy for consistency between I/O and cache (checking L2 only)
    - Drawback: L2 must invalidate all L1 blocks that map onto the 2nd-level block to be replaced => slightly higher 1st-level miss rate
      - i.e. Intel Pentium 4: 64-byte block in L1 and 128-byte in L2
  - multilevel exclusion: L1 data is never found in L2
    - A cache miss in L1 results in a swap of blocks between L1 and L2
    - Advantage: prevent wasting space in L2
      - i.e. AMD Athlon: 64 KB L1 and 256 KB L2

Documente utile:
1 - Optimal partitioning of LLC in CAT-enabled CPUs to prevent side-channel attacks

3. **Reducerea timpului de execuţie în caz de hit în cache.**

Way prediction, Trace caches.