Tema 1 - Cache Memory

Responsabili:

- Andrei Vasiliu [mailto:mailto:andrei.vasiliu2211@gmail.com]
- Mihai Popescu [mailto:mailto:mh.popescu12@gmail.com]

Deadline: 10 aprilie, ora 23:55

Modificări și actualizări

- **19 martie**
  - adaugare enunt
- **20 martie**
  - adaugare precizare genericitate Hashtable
- **21 martie**
  - adaugare informatii despre punctaj
  - adaugare informatii noi in datele de iesire
- **25 martie**
  - adaugare arhiva de teste
  - schimbare nume fisier de iesire din cores.out in cache.out
  - modificare greseala la titlul politicii de inlocuire, din Least Recently Used in Least Recently Added (cum bine a observat cineva la curs)
- **26 martie**
  - modificare teste 9 si 10

Obiective

În urma realizării acestei teme:

- veți învaţa să definiți și să folosiți tipuri de date proprii
- vă veți acomoda cu sintaxa de C++ (genericizare, management de memorie, operatori)
- veți fi capabili să implementați un hashtable, similar cu cel din STL
- veti aplica o structura de date intr-o problema reala
- veti intelege cum functionaza memoria cache a unui procesor

Intro - Cache Memory

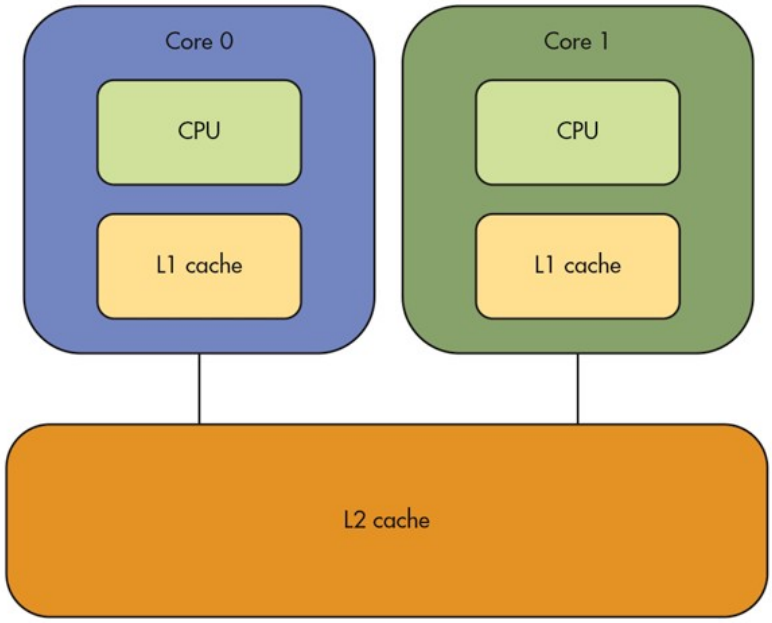
Instructiunile si datele programelor sunt stocate in RAM [https://en.wikipedia.org/wiki/Random-access_memory]. Aceasta este o memorie rapida, dar mult mai inceata fata de viteza cu care prelucreaza datele procesoarele. Astfel, pentru a reduce mult din timp a fost introdusa memoria cache [https://en.wikipedia.org/wiki/CPU_cache], o memorie specifica procesorului, de mici dimensiuni, foarte rapida(intermediara intre viteza procesorului si a RAM-ului). Cele mai des utilizate date sunt stocate in cache, pentru a fi accesate mult mai rapid. (why computers need cache [https://www.reference.com/technology/computers-need-cache-memory-ec2205edbd1c29e0]).

Functionare cache

In cache se incarca datele accesate de nucleu, pentru a putea fi accesate mai rapid data viitoare cand va fi nevoie.

Fie un procesor cu 2 nuclee si 2 nivele de cache (un L1 privat fiecarui nucleu si un L2 comun). Pentru fiecare acces la memorie, se face in prealabil o cautare a acelei adrese in memoria cache (intai in L1 si apoi L2). Daca adresa nu este gasita nicaieri, se va cauta in RAM si se va copia in L2 si in L1 al respectivului nucleu care a facut accesul. Daca adresa este insa gasita in L1, sa va folosi, iar daca e in L2, se va copia inainte in L1 si apoi se va folosi. Astfel se micsoreaza timpul de acces pentru valorile de la adresele care au mai fost accesate anterior.

Asa arata memoria procesorului din exemplu:



Sincronizare intre nuclee folosind L2 si dirty bit

Sa luam un exemplu pentru simplificarea explicatiei. Fie un procesor cu 2 nuclee. Fiecare nucleu are o memorie cache privata L1, iar ambii au acces la o memorie shared L2. In L1 pentru fiecare nucleu sunt adresele si datele pe care repectivul nucleu le-a accesat, pe cand in L2 sunt o reuniune de adrese si date accesate de ambii.

Astfel, sa presupunem ca memoria RAM arata ceva de genul acesta (prima coloana e adresa, a doua e valoarea de la acea adresa):

```
20000 7
20004 10
20008 0
```

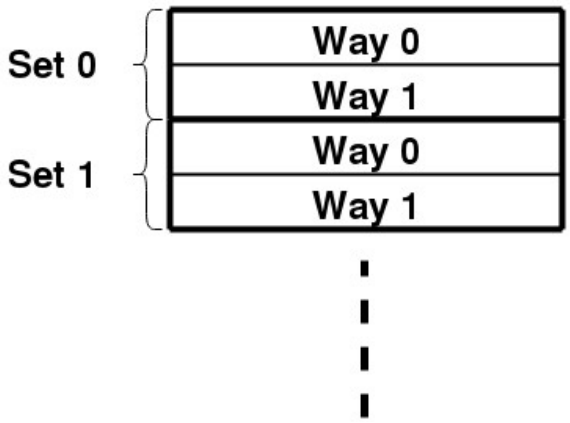
Un nucleu o sa vrea sa citeasca valoarea de la adresa 20000. Presupunand ca nu exista altceva in cache pana acum, ca sa simplificam exemplul, el va ajunge cu cautarea adresei pana in RAM (nu a gasit-o nici in L1, nici in L2). O gaseste in RAM, o copiaza in L2 si in cache-ul L1 al lui. Al doilea nucleu citeste si el valoarea de la adresa 20000, pe care o gaseste deja in L2. O copiaza in cache-ul L1 al lui.

Apoi, sa spunem ca nucleul 1 vrea sa modifice valoarea de la adresa 20000. O cauta in L1, o gaseste, o modifica in L1, apoi o modifica si in L2. Daca acum nucleul 2 vrea sa citeasca valoarea de la adresa 20000, va citi o valoare veche (asta are el in L1), insa vrem sa citeasca valoarea noua. Pentru aceasta, exista notiunea de **dirty bit**, care este setat atunci cand o intrare din cache nu are cea mai recenta valoare. Astfel, cand nucleul 1 modifica valoarea de la adresa 20000, o modifica in L1, in L2 si o pune pe dirty pe cea din L1 al celuilalt nucleu. Astfel, cand acela va vrea sa citeasca data de la adresa 20000, o va cauta in L1, vede ca e dirty, astfel ca merge la nivelul 2, in L2, unde gaseste valoarea updatata, pe care o copiaza la el in L1 si apoi o foloseste.

Mapare set asociativa pe 2 cai

Modalitatea de mapare a unei memorii cache reprezinta felul in care datele sunt organizate in interiorul memoriei pentru a facilita accesarea rapida. Fara sa intram in prea multe detalii, vom prezenta doar un tip de mapare, care va fi si de folos in rezolvarea temei.

Memoria cache are o dimensiune, sa spunem **cacheDim** bytes. Ea este formata din blocuri de cache. Vom presupune ca un bloc de cache are dimensiunea **blockDim** bytes, rezulta ca sunt **nrBlocks** = **cacheDim** / **blockDim** blocuri in acest cache. La o mapare set asociativa pe 2 cai, fiecare 2 blocuri reprezinta cai intr-un set, astfel ca sunt **nrSets** = **nrBlocks** / 2 seturi. O reprezentare vizuala a acestei mapari poate fi vazuta mai jos:



Indexarea in acest cache se face cu ajutorul indexului setului (de la **0** la **nrSets - 1**). Astfel, se folosesc anumiti biti (dupa cum vom vedea) din adresa pentru a extrage numarul setului din cache pentru aceasta.

In acest fel, daca se doreste cautarea unei anumite adrese in cache, se va afla numarul setului in care aceasta poate fi si apoi se va cauta in interiorul setului, daca adresa cautata se afla printre cele doua valori posibile.

Pentru adaugarea unei valori noi in cache, avem nevoie de urmatoarea notiune:

Politica de inlocuire a datelor din cache - Least Recently Added (LRA)

Cand vrem sa adaugam o noua valoare in cache, si am identificat setul in care trebuie sa o adaugam, avem urmatoarele cazuri:

1. ambele locuri sunt libere
2. un singur loc e liber
3. ambele locuri sunt ocupate

Astfel, Least Recently Added spune ca:

- daca ambele locuri sunt libere, vom adauga noua adresa pe prima pozitie
- daca un singur loc e liber, vom adauga noua adresa in acel loc
- daca ambele locuri sunt ocupate, vom adauga noua adresa in locul adresei adaugate cel mai putin recent

Politica de scriere in RAM - Write-back

Pana acum am vazut cum interactioneaza procesorul cu datele citite din RAM si adaugate in cache. Apoi, am vazut cum interactioneaza cu datele care sunt deja in cache.

Datele din cache trebuie sa ajunga si inapoi in RAM, la un moment dat. Acest moment este dat de politica de scriere in RAM. Vom vorbi doar despre o politica, de interes pentru aceasta tema, **Write-back**.

Aceasta politica spune ca o valoare este scrisa in RAM atunci cand este scoasa din cache prin adaugarea unei alte intrari.

In cadul enuntului acestei teme au fost explicate foarte sumar notiunile de baza despre memorii cache (exact atat cat va va trebui pentru a rezolva tema). Informatii mai multe si mai detaliate veti invata la materiile Calculatoare Numerice 1 si 2.

Cerinta

Ne propunem sa simulam functionalitatea memoriei cache pentru un procesor folosind hashtable-uri. Vom considera un procesor cu doua nuclee, astfel incat:

- fiecare nucleu are memoria lui L1 (un hashtable de dimensiune mica)
- ambele nuclee au acces la acceasi unica zona de memorie L2 (alt hashtable de dimensiune mai mare)

Memoriile cache vor folosi maparea **set asociativa pe 2 cai** si o politica de **write-back** pentru scrierea in RAM, ambele prezentate anterior.

Ele vor avea urmatoarea dimensiune:

- L1 va avea 2^12 = 4096 intrari → 2^11 seturi
- L2 va avea 2^14 = 16384 intrari → 2^13 seturi

Implementare

Structuri de date obligatorii

Memoriile cache trebuie implementate ca hashtable-uri **generice** in care:

- **cheia** reprezinta adresa
- **valoarea** reprezinta data

Indexul setului din cadrul cache-ului va fi calculat cu ajutorul:

- pentru cache-urile L1 - bitilor 2-12 din adresa
- pentru cache-ul L2 - bitilor 2-14 din adresa

Bitii se numara de la dreapta la stanga (de la cei mai putini semnificativi la cei mai semnificativi), pornind de la 0

Functionalitate

Pentru operatia **read**:

- Cautati valoarea in L1 nucleului corespunzator
  - Nu o gasiti in L1 (sau e dirty in L1), o cautati in L2
    - O gasiti in L2, o copiati in L1, done
    - Nu o gasiti in L2, o cautati in fisierul **ram.in**
      - cititi din fisier valoarea de la acea adresa
      - veti stoca in L2 aceasta valoare
      - veti stoca in L1 nucleului corespunzator aceasta valoare
  - O gasiti in L1, done

Pentru operatia **write**:

- cautati valoarea in L1 nucleului corespunzator
  - Nu o gasiti in L1 (sau e dirty in L1), o cautati in L2
    - O gasiti in L2, o copiati in L1 si updatati si in L1 si in L2 (punand dirty in L1 celuilalt nucleu, pentru aceasta intrare)
    - Nu o gasiti in L2, o cautati in fisierul **ram.in**
      - O copiati in L1 si in L2 si o updatati in ambele
  - O gasiti in L1 (si nu e dirty), o updatati in L1, si o copiati in L2 (punand dirty in L1 celuilalt nucleu, pentru aceasta intrare)

Doar atunci cand scoateti o valoare din L2 va trebui sa faceti update in fisierul ram.out (**write-back**).

Date de intrare

Veti avea 2 fisiere de intrare:

- ram.in
- operations.in

Fisierul **ram.in** are urmatoarea structura:

- linii separate prin newline, fiecare formate din 2 intregi: **`address value`**
  - **address** este un intreg pozitiv reprezentand adresa la care se afla stocata valoarea **value**
  - **value** este un intreg pozitiv reprezentand valoarea de la adresa **address**

Fisierul **operations.in** are urmatoarea structura:

- linii separate prin newline, de tipul: **`coreNumber operation address [newData]`**
  - coreNumber reprezinta numarul nucleului pe care va fi facuta operatia; poate fi 0 sau 1 deoarece avem doar 2 nuclee
  - operation reprezinta tipul operatiei; poate fi de tipul 'r' sau 'w', unde:
    - **r** reprezinta operatie de read
    - **w** reprezinta operatie de write; inseamna ca valoarea de la acea adresa va fi modificata cu o noua valoare; in acest caz apare parametrul optional **newData**

Date de iesire

Veti avea 2 fisiere de iesire:

- cache.out - in care veti afisa date ramase in cache-ul L1 al ambelor nuclee si apoi cele ramase in L2
- ram.out - in care veti salva valorile finale din RAM

Fisierul **cache.out** trebuie sa aiba urmatoarea structura:

- valorile pentru L1 al nucleului 0
- rand liber
- valorile pentru L1 al nucleului 1
- rand liber
- valorile pentru L2

unde liniile sunt de forma:

- ```
  setIndex wayIndex address value [*]
  ```
  - **setIndex** reprezinta numarul setului
  - **wayIndex** reprezinta numarul caii din set
  - **address** reprezinta adresa care a fost adusa in cache
  - **value** reprezinta valoarea de la adresa **address**
  - **steluța "*"** se afișează daca respectiva cale din cache este dirty

Nu se printeaza seturile si caile din cache care nu au date

Fisierul **ram.out** trebuie sa pastreze structura fisierului **ram.in**.

Teste publice

Aveți la dispoziție o arhivă ce conține o suită de teste publice (singurele cu care va fi testată tema) și un script de testare automată a temei. Scriptul este de bash (Linux).

Atenție! Pentru ca testarea sa mearga, executabilul generat de tema voastră trebuie să fie în același director cu scriptul test.sh, respectiv cu directoarele input și ref din arhiva de mai jos.

Arhiva de testare
  ```

Reguli pentru trimitere

Temele vor trebui trimise pe vmchecker [https://elf.cs.pub.ro/vmchecker/ui/#SD]. **Atenție!** Temele trebuie trimise în secțiunea **Structuri de Date (CA)**.

Arhiva trebuie să conțină:

- surse
- fișier Makefile cu două reguli:
  - regula **build**: în urma căreia se generează un executabil numit **tema1**
  - regula **clean** care şterge executabilul
- fisier **README** care să conțină detalii despre implementarea temei

Punctaj

- 80 puncte obținute pe testele de pe vmchecker. Condiții pentru obținerea punctajului total:
  - fără memleak-uri
  - fără erori de valgrind
- 20 puncte: README
- **Bonus 20% din punctajul obținut** pentru coding style
  - spre exemplu: cu 60p din 100p și coding style perfect, obțineți 60*1.2 = 72p

Nu copiați! Toate soluțiile vor fi verificate folosind o unealtă de detectare a plagiatului. În cazul detectării unui astfel de caz, atât plagiatorul cât și autorul original (nu contează cine care e) vor primi punctaj 0 pe **toate temele**!

De aceea, vă sfătuim să nu vă lăsați rezolvări ale temelor pe calculatoare partajate (la laborator etc), pe mail/liste de discuții/grupuri etc.

FAQ

Q: Se poate folosi STL?

A: Nu puteți folosi structurile gata implementate în STL. Obiectivul principal al cursului de structuri de date este acela ca voi să implementați structurile respective.