Program de testare a performantelor de transfer pentru memoria cache

Contents

1.Introducere

1.1 Context

În lumea tehnologiei moderne, performanța este o componentă vitală pentru asigurarea funcționării eficiente a sistemelor informatice.

Memoriile cache reprezintă o componentă esențială a arhitecturii sistemelor de calcul, fiind responsabile pentru accelerarea accesului la datele critice. O evaluare exhaustivă a performanțelor acestor memori își are rădăcinile în necesitatea optimizării acestor sisteme pentru a asigura o experiență utilizator fluidă și o execuție rapidă a operațiilor critice.

În acest context, prezentul proiect propune dezvoltarea unui program de testare a performanțelor de transfer pentru memoriile cache, cu scopul de a evalua și optimiza rata de transfer a datelor în cadrul sistemelor de calcul.

Acest program se adresează în special arhitecturilor de tip cache utilizate în procesoarele moderne, având ca scop principal identificarea și analizarea posibilelor puncte slabe ale acestora, în vederea maximizării performanțelor sistemului.

1.2 Specificatii

Acest proiect va fi realizat in limbajul de programare C++ cu ajutorul mediului integrat de dezvoltare (IDE) Visual Studio pe un laptop Asus cu specificatiile :

- Procesor Intel Core i3 1005G1 (Ice Lake) 2 cores, 4 threads;
- 8GB memorie RAM DDR4
- Placa video Intel(R) UHD Graphics
- Cache
    - L1 Data – 2 x 48 KBytes 12-way
    - L1 Instruction – 2 32 KBytes 8-way
    - L2 – 2 x 512 KBytes 8-way
    - L3 – 4 MBytes 16-way

1.3 Obiective

Prin intermediul acestui proiect, se urmărește îmbunătățirea înțelegerii asupra modului în care funcționează memoriile cache, precum și optimizarea acestora pentru a asigura performanțe superioare în cadrul sistemelor de calcul moderne.

Cu ajutorul acestei documentații, utilizatorii vor fi capabili să înțeleagă mai bine complexitatea testării performanțelor de transfer pentru memoriile cache, precum și să aplice cunoștințele acumulate în optimizarea sistemelor lor de calcul pentru o performanță superioară și o experiență utilizator îmbunătățită.

2.Studiu Bibliografic

În informatică, memoria cache (sau simplu un cache) este o colecție de date ce sunt o "copie la indigo" a valorilor originale stocate altundeva sau calculate mai devreme, unde operația de aducere din memorie a datelor originale este costisitoare (datorită timpilor mari de acces la memorie) sau costul recalculării acestora este mare, în comparație cu costul citirii acestora din cache.

Cu alte cuvinte, un cache este o arie temporară de stocare unde datele utilizate în mod frecvent pot fi depozitate pentru un acces rapid la acestea. Odată ce datele sunt stocate în cache, în viitor vor fi luate de aici și utilizate decât să se încerce readucerea datelor originale sau recalcularea acestora, astfel încât timpul mediu de acces este mai mic.

Organizarea memorie cache:

Memoria cache poate fi organizată în mai multe zone de memorie, cu dimensiuni și funcționalități diferite. Aceste zone (de date, de instrucțiuni) se regăsesc la nivelul microprocesorului,în exteriorul UCP-ului. La un nivel superior,zonele sunt unificate și rezultată cache-ul unificat, care este cel ce interacționează cu memoria principală.

Cache de instrucțiuni:

Cache-ul de instrucțiuni este folosit pentru memorarea instrucțiunilor care sunt folosite frecvent, ceea ce duce la mărirea vitezei de funcționare a sistemului. Această zonă poate chiar să facă operații limitate, sau să "prezică" datele ce vor fi folosite ulterior, prin memorarea instrucțiunilor accesate cu frecvență.

Cache-ul de Date:

Cache-ul de date este un buffer foarte rapid, care poate prelua datele necesare unor instrucțiuni din memoria principală și să le transmită regiștrilor. Odată ajunse datele în regiștrii, acestea pot fi folosite de către procesor în instrucțiuni. După terminarea execuției instrucțiunii, rezultatul reținut în regiștrii este returnat pentru memorare în cache-ul de date și apoi transmis memoriei principale.

Tipuri de memorie cache:

(A) cache de nivel 1 (Level1 sau L1) - este memoria cache construită în Unitatea Centrală de Procesare (UCP); este cel mai rapid tip de memorie, pentru că poate funcționa la aceeași viteză cu cea a microprocesorului

(B) cache de nivel 2 (Level2 sau L2) - este memoria de pe un chip separat față de UCP; poate ajunge să funcționeze la o viteză de aproape 2 ori mai mare decât RAM-ul.
(C) cache de nivel 3 (Level3 sau L3) - folosite de anumite UCP-uri conțin atât memorie de nivel 1 cât și memorie de nivel 2 atașate sistemului

Atunci când vorbim de îmbunatățirile de performanță rezultate din folosirea cache-ului, vom folosi următorii termeni:

- **Cache Hit** – generat atunci cand pentru a accesa date din memoria principala le luam din cache
- **Cache Miss –** generat atunci cand datele nu sunt in cache, fiind aduse procesorului direct din memoria principala
    - **Cache miss obligatoriu** – cauzat de primul acces la o zona de memorie
    - **Cache miss de capacitate** – cauzat de capacitatea limitata a cache-ului
    - **Cache miss conflictual** – cauzat de inlocuirea unui element din cache cu unul nou adus
- **Fetch** – aducerea unui bloc de cuvinte din memoria principala intr-o linie de cache

Spatiu de adrese virtuale vs fizice :

- Fiecare proces are propriul spatiu de adrese, ex : procesul P1 are la adr. a alte date decat are P2 la aceeasi adresa a
- Adresele folosite intr-un program se numesc adrese virtuale si formeaza spatiul de adrese virtuale
- Adresele din memoria RAM formeaza spatiul de adrese fizice
- Procesorul are o componenta numita MMU (Memory Management Unit), ce mapeaza adresele virtuale la cele fizice

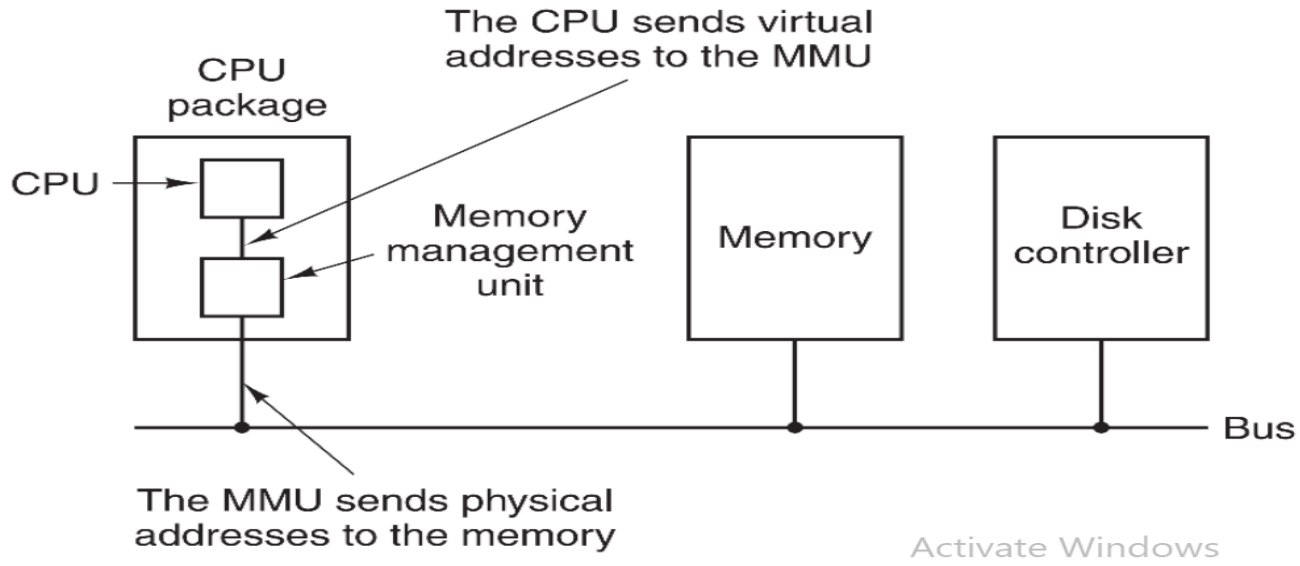

Figura 1.

Maparea Memoriei Cache :

Memoria cache continua sa evolueze, dar traditional ea functioneaza sub 3 configuratii :

1. Memorii cache mapate direct :

Cache-ul cu mapare directă are fiecare bloc mapat către exact o locație de memorie cache. Din punct de vedere conceptual, un cache cu mapare directă este similar cu rândurile dintr-o tabelă având trei coloane: blocul de cache care conține datele efectiv preluate și stocate, o etichetă cu adresa completă sau parțială a datelor care au fost preluate, și un bit de validitate care arată prezența în intrarea din rând a unui bit valid al datelor.

2. Memorii cache complet asociative :

Maparea complet asociativă a cache-ului este similară cu maparea directă în structură, dar permite ca un bloc de memorie să fie mapat către orice locație de cache, în loc să fie mapat către o locație specificată în memoria cache, așa cum se întâmplă în cazul mapării directe.

3. Memorii cache asociative pe multimi (mapare set asociativa) :

Maparea set asociativă a cache-ului poate fi văzută ca un compromis între maparea directă și maparea complet asociativă, în care fiecare bloc este mapat către un subset de locații de cache.

Uneori este numită mapare în N moduri, care permite ca o locație din memoria principală să fie cache-uită la oricare dintre "N" locații din memoria cache L1.

Politica de scriere a datelor :

Datele pot fi scrise in memorie folosind o varietate de tehnici, dar cele doua principale implicate in memoria cace sunt :

1. Scriere Directa – datele sunt scrise atat in memoria cache, cat si in memoria principala in acelasi timp;

2. Scriere Amanata – datele sunt scrise doar in memoria cache initial, datele pot fi apoi scrise in memoria principala, dar acest lucru nu este obligatoriu si nu impiedica interactiunea sa aiba loc;

3.Analiza

Pentru testarea vitezei de transfer a memoriei cache, aplicatia foloseste mai multe array-uri generate random si aplica niste operatii simple (adunare,scadere) pentru fiecare element.

În contextul analizei noastre, este important să subliniem că, deși simulăm performanțele de transfer pentru memoriile cache, controlul direct al stocării datelor în diverse niveluri ale cache-ului este practic imposibil.

Gestionarea precisă a locației de stocare a datelor sau a unui array specific într-un anumit nivel de cache nu este posibilă din cauza restricțiilor hardware și a modului complex de gestionare a cache-ului de către sistemul de operare.

Totusi putem presupune locul de stocare al arrayului in functie de dimensiunea pe care acesta o are. Precizat la capitolul Specificatii fiecare level al memoriei cache are o anumita dimensiunea, asadar trebuie sa calculam nr maxim de elementele ale unui array pentru a incapea pe un anumit nivel, folosind relatia nr maxim de elemente din vector = nr maxim de blocuri cache.

L1 Data Cache :

- Dimensiunea totala a L1 Data Cache = 2 x 48Kbytes = 96Kbytes
- Stim ca L1 Data Cache este 12-way asa ca dimensiunea unui set de cache este = 48Kbytes / 12 = 4Kbytes
- Stiind ca un bloc de cache are 48Kbytes ca sa putem face calculul trebuie sa ii transformam in bytes adica inmultim cu 1024, rezultand astfel 48x1024=49,152
- Pentru a afla cat intra intr-un set trebuie sa impartim la cat se scrie pe un rand adica 64bytes si la 12 pentru ca avem 12 seturi intr-un bloc 49,152/64/12 = 64
- Deci practic pentru un set putem avea un array de 64 de elemente, un bloc de cache are 12 seturi, iar L1 in total are 2 blocuri , deci in L1 incape un array de 1,536 de elemente

L2 Cache :

- Dimensiunea totală a L2 Cache = 2 x 512 KBytes = 1024 KBytes
- Stim ca L2 Cache este 8-way asa ca dimensiunea unui dry de cache = 512 KBytes / 8 = 64 KBytes
- Stiind ca un bloc de cache are 512Kbytes ca sa putem face calculul trebuie sa ii transformam in bytes adica inmultim cu 1024, rezultand astfel 512x1024=524,288
- Pentru a afla cat intra intr-un set trebuie sa impartim la cat se scrie pe un rand adica 64bytes si la 8 pentru ca avem 8 seturi intr-un bloc 524,288/64/8 = 1024
- Deci practic pentru un set putem avea un array de 1024 de elemente, un bloc de cache are 8 seturi, iar L2 in total are 2 blocuri , deci in L2 incape un array de 16,384 de elemente

L3 Cache :

- Dimensiunea totală a L3 Cache = 4 MBytes
- Stim ca L3 Cache este 16-way asa ca dimensiunea unui set de cache = = 4 MBytes / 16 = 256 KBytes
- Stiind ca dimensiunea totala de cache are 4Mbytes ca sa putem face calculul trebuie sa ii transformam in Kbytes deci avem 4x1024 = 4096Kbytes , apoi transformam in bytes adica inmultim cu 1024, rezultand astfel 4096x1024=4,194,304, pentru a afla cat intra intr-un set trebuie sa impartim la 64(cat se scrie pe un rand) si la 16 pentru ca sunt 16 seturi 4,194,304/64/16 = 4096
- Deci practic pentru un set putem avea un array de 4096 de elemente, un bloc de cache are 16 seturi, iar L3 in total are 1 blocuri , deci in L3 incape un array de 65,536 de elemente

In realitate aceste calcule pot fi destul de complexe, deoarece depind de mai mulți factori, inclusiv arhitectura specifică a procesorului.

4. Design :

Designul aplicatiei va consta in generarea unor rapoarte cu ajutorul profilerului. Aplicatia este contrstruita modular, fiecare zona de memorie cache avand propriul cpp si header unde sunt setate valorile calculate la capitolul de mai sus.

Utilizand functia clock din libraria <chrono> vom calcula durata operatiilor aplicate array-ului, vom imparti la numarul de teste si vom face un avg.

Utilizand un vector de stringuri ‘stringVector’ vom stoca numarul de seturi pentru fiecare model de cache,dar la L3 vom pune inca 16 seturi in plus pentru a calcula si valori ce ies inafara memoriei cache.

Utilizand libraria “Profiler.h” librarie utilizata la disciplina algoritmi fundamentali, vor fi generate grafice ce evidentiaza diferentele intre vitezele de transfer.

5.Implementation :

Implementarea unei astfel de testari consta in implementarea unui cod care sa evidentieze diferentele dintre vitezele de transfer.

Pentru a indeplini acest scop ne vom defini constante aproximative pentru fiecare zona de memorie cache si pentru memoria propriu zisa, astfel vom defini constantele L1,L2,L3 si M.

Pentru a putea folosi libraria profiler trebuie sa creem un obiect de tip profiler :

```
Profiler profiler("ProiectSSC");
```

Figura 2.

Se vor declara array-urile propriu zise si folosing functia FillRandomArray din profiler ele vor fi umplute cu elemente random :

```
FillRandomArray(vL1, L1);
FillRandomArray(vL2, L2);
FillRandomArray(vL3, L3);
FillRandomArray(vM, M);
```

Figura 3.

Testarea vitezei se face prin calcularea timpului in care parcurgem fiecare element din array si ii aplicam o operatie simpla, in cazul nostru incrementarea cu 1. Cu cat arrayul se afla pe un nivel de cache mai apropiat de procesor cu atat timpul de executie va fi mai rapid, astfel vom incerca sa determinam diferenta dintre ele.

Testarea se va face de un numar mai mare de ori(in cazul nostru 20) pentru a ne asigura ca arrayul a reusit sa fie stocat in cache. Pentru testarea timpul vom folosi libraria <chrono> din C++, deoarece ne pune la dispozitie mai multe optiuni de calculare a timpului.

Vom calcula timpul in nanosecunde pentru ca este singura unitate care poate observa timpul nivelelor cele mai apropiate de procesor :

```
auto start_timeL1 = std::chrono::high_resolution_clock::now();
```

Figura 4.

```
auto end_timeL1 = std::chrono::high_resolution_clock::now();
```

Figura 5.

```
auto durationL1 = std::chrono::duration_cast<std::chrono::nanoseconds>(end_timeL1 - start_timeL1);
```

Figura 6.

Se va repeta acest procces pentru fiecare zona, masuratorile trebuie facute pe rand pentru a ne asigura ca nu se suprapun.

Cu profiler vom crea un grafic pentru fiecare masuratoare, in mod normal profiler poate fi folosit si pentru a masura timpul de executie dar in cazul vitezei memoriei cache sunt numere prea mici pentru a putea fi inregistrate.

Pentru a putea afisa pe grafic numarul nanosecundelor vom folosi profiler.countOperation pentru a afisa numarul nanosecundelor :

```
for (int i = 0; i < durationL1.count(); i++)
{
    profiler.countOperation("L1", k);
}
```

Figura 7.

Procesul descris se va repeta pentru fiecare dintre zonele de memorie. Profilerul va genera o pagina web unde vom putea vedea graficele. Varianta standard a aplicatiei foloseste 20 de teste.

Graficele generate de profiler au fost impartite, cate 3-4 pe un grafic pentru a putea fi mai usor de observat comportamentul

```
profiler.createGroup("CompareL1Part1", stringVector[0].c_str(), stringVector[1].c_str(),
    stringVector[2].c_str());
```

Figura 8.

6.Testing:

Testarea aplicatiei precum descris mai sus, consta in graficele generate. Am ales ca cele 3-4 elemente afisate pe grafic sa fie cat mai apropiate, astfel putem sublinia mai bine diferentele.

Pentru L1 am calculat ca o bucata din setul de memorie cuprinde 64 de elemente, memoria fiind doua 12 way sets, practic am calculat diferentele de la bucata la bucata pentru pentru fiecare set

De exemplu aici avem testul pentru primele 3 bucati din primul set :

CompareL1Part1

size	L1_Set1	L1_Set2	L1_Set3
1	300	400	600
2	500	800	900
3	400	500	600
4	400	400	700
5	300	500	800
6	300	600	1400
7	400	500	800
8	300	300	900
9	200	500	700

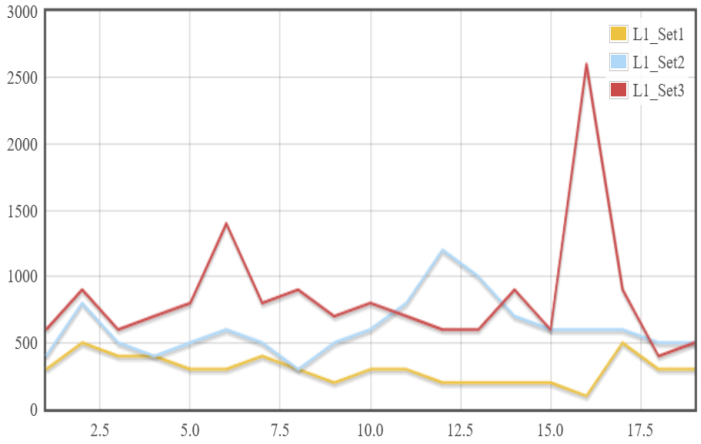

Figura 9

Pentru primele 3 bucati din set executia este foarte rapida, putem observa valorile de start ca fiind 300, 400 respectiv 600 nanosecunde, masuratoare se realizeaza in nanosecunde pentru ca avand valori mici este singura unitate de masura care poate surprinde timpii de executie

Putem observa cum in 20 de teste se pastreaza acelasi ritm constant,bucata cu capacitate mai mica va avea intotdeauna timpul mai rapid.

Pentru bucati mai mari din set se pot observa clar diferentele de timp fata de cele mai mici de ex :

CompareL1Part5

10	2100	3100	3000
11	2300	3000	2900
12	2400	3700	3200
13	2200	3000	3100
14	2100	3000	3100
15	2800	3400	3700
16	2700	11500	3300
17	2600	3200	3400
18	4100	3000	3500
19	3000	3400	4000

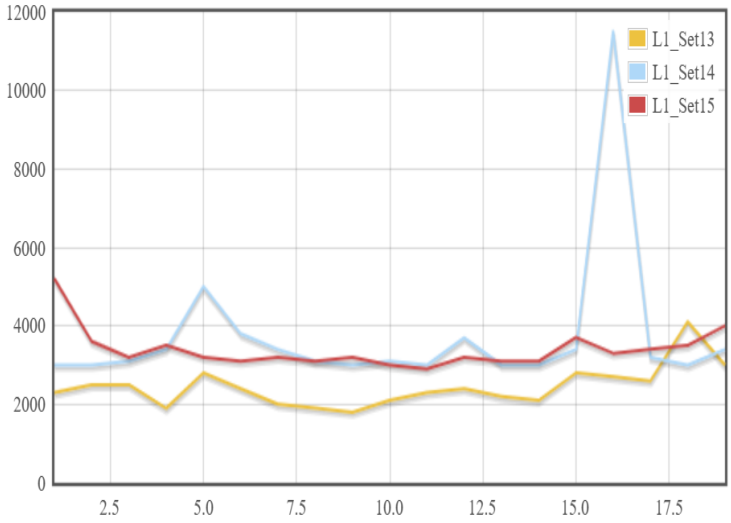

Figura 10.

Totodata pentru date atat de mici nu putem masura cu exactitate ce se intampla de fapt, putem observa clar ca apar si abateri de la timpurile asteptate de noi, de ex L1_Set14 la al 16-lea test are valoarea 11500, L1_Set14 teoretic trebuind sa fie mai mic decat L1_Set15, genul acesta de valori demonstreaza impredictibilitatea pentru bucati de date atat de mici

Trecand la L2 putem observa ca pe grafice incepe sa apara comportamentul asteptat de catre noi :

CompareL2Part5

size	L2_Set13	L2_Set14	L2_Set15	L2_Set16
1	20800	33300	23100	24800
2	29800	22100	23900	26200
3	34900	23000	23700	25400
4	24100	23300	23400	60100
5	49000	22500	42900	26100
6	25900	22500	24200	25800
7	22500	28800	23600	25600
8	20600	22500	23400	26500
9	20500	23100	24500	25700

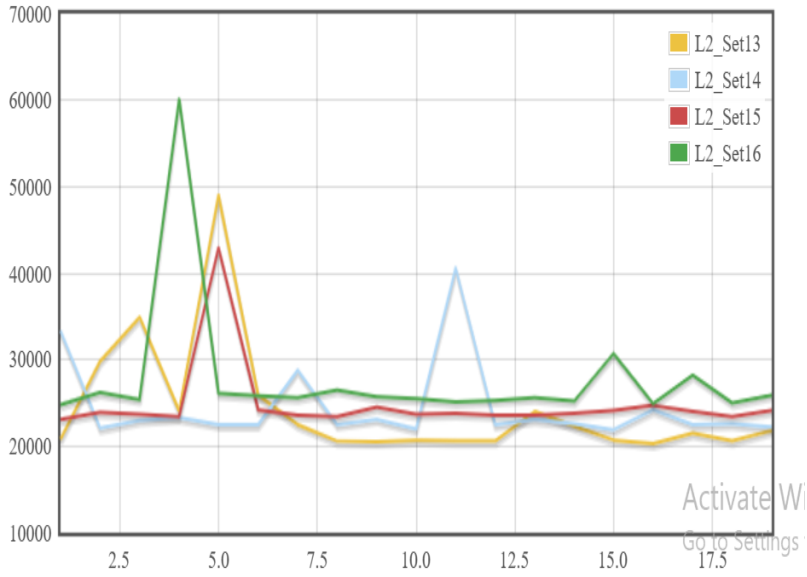

Figura 11.

Pe exemplul acesta putem sa observam cel mai bine cum se comporta memoria cache, observam ca la inceput fiecare bucata are un spike destul de mare si apoi incepe sa scada, acest comportament este firesc deoarece ne asteptam sa treaca cel putin 5 teste pana ca o valoare sa ajunga sa fie in cache, dupa care timpul de executie scade pentru fiecare pastrand un comportament aproape perfect asteptat

Deja se poate observa diferenta considerabila a timpului de executie fata de bucatile de date de la L1, motivul pentru care am lasat nanaosecunde peste tot si nu am transformat in altceva pentru a avea valori mai mici este exact acesta, de a sublinia diferentele mari intre diferitele nivele de memorie.

Putem observa ca si la L3 unde valorile sunt mult mai mari tot apar imprefectiuni dar in general fiecare grafic arata rezultatul asteptat si respecta regula :

CompareL3Part4

8	64300	89000	75200
9	63400	69300	75400
10	63800	102900	76800
11	63100	70600	75300
12	63300	69200	95300
13	64600	70600	75500
14	62700	71200	77400
15	63700	86900	76200
16	64900	75400	78300
17	62700	89400	77200

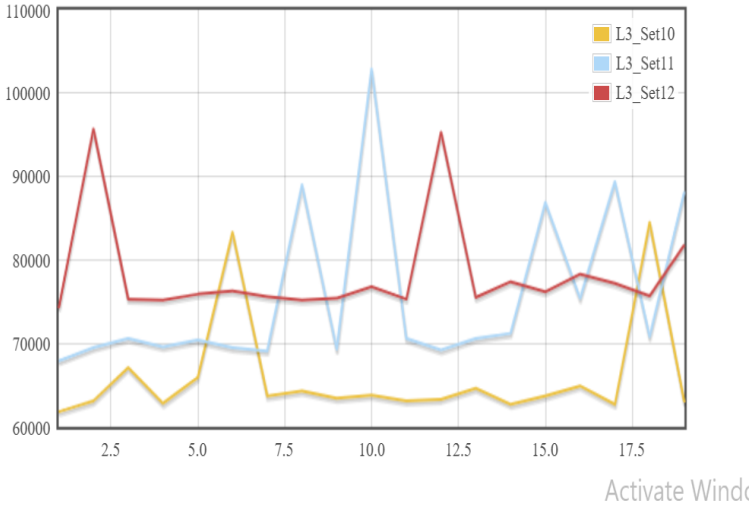

Figura 12.

Pentru L3 seturile de la 16 la 32 sunt valori care nu mai intra in memoria cache, ci in memoria principala datorita dimensiunii lor foarte mari, putem observa diferentele dintre cele care intra in cache si cele care nu. Putem observa diferenta de timp pentru date din memoria principala in figura 12:

CompareL3Part8

size	L3_Set24	L3_Set25	L3_Set26	L3_Set27
1	148200	154300	160300	166300
2	151300	156800	164400	171200
3	152700	180900	163200	190700
4	170500	157100	163000	169500
5	150200	178000	188300	192800
6	151300	235800	163800	170600
7	150400	157300	183300	170700
8	150400	159000	164000	170000
9	168400	156900	165100	203700

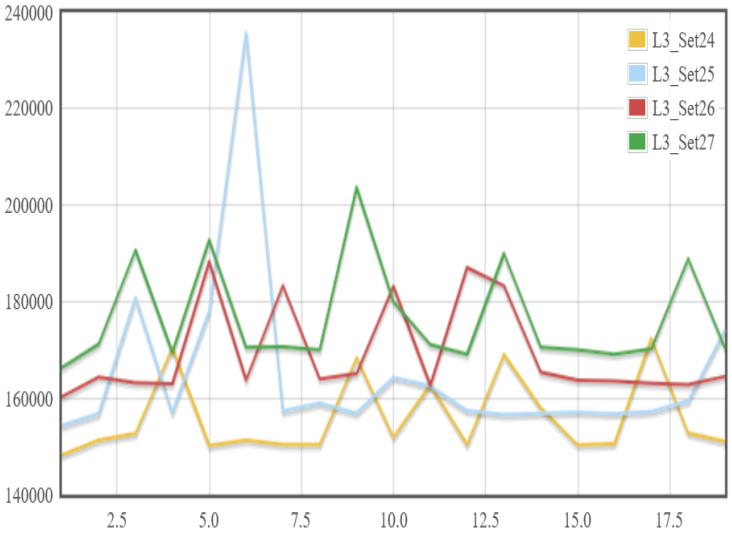

Figura 13.

7.Concluzii:

In concluzie aplicatia simuleaza viteza de transfer a diferitelor levele de memorie si sublineaza diferentele semnificative dintre nivele. Desi nu putem stii cu exactitate locatia din cache in care se afla vectorul pe care testam, testarea repetata ajuta pentru a simula cu cat mai mare exactitate.

Lucru la aceasta aplicatie a fost unul placut, am aflat mutle despre memoria cache, cum opereaza ea si avantajele acesteia.

Pentru dezvoltarea ulterioara a aplicatiei se poate incorpora o interfata grafica in care sa poti selecta un numar de teste dupa bunul plac.

8.Bibliografie :

- Memorie Cache Universitatea Politehnica din Bucuresti - https://elf.cs.pub.ro/ac/wiki/_media/arhiva/2012/laborator10.pdf
- Memorie Cache Wikipedia - https://ro.wikipedia.org/wiki/Memorie_cache
- TechTarget Cache Memory - https://www.techtarget.com/searchstorage/definition/cache-memory
- Technical University Of Berlin Caches and the Principles of Locality - https://www.youtube.com/watch?v=KmairurdiaY
- SimonDev Memory, Cache Locality, and why Arrays are fast - https://www.youtube.com/watch?v=247cXLkYt2M
- Eye on Tech L1,L2 and L3 Cache explained - https://www.youtube.com/watch?v=IA8au8Qr3lo
- Universitatea Tehnica Cluj Napoca disciplina Sisteme de operare Oprisa Ciprian, cursul 11 – Managmentul Memoriei 1
- Universitatea Tehnica Cluj Napoca profiler - https://users.utcluj.ro/~cameliav/fa/profiler_guide.pdf

Tabel de figuri: