

## ***Rapport de TP Architecture***

### *Évaluation des performances de mémoire CACHE*



## Table des matières

Introduction.....	3
I. Exercice 1.....	3
I. 1. Question 1.....	3
I. 2. Question 2.....	5
II. Exercice 2.....	6
II. 1. Question 1.....	6
III. Exercice 3.....	9
III. 1. Question 1.....	9
III. 2. Question 2.....	9
III. 3. Question 3.....	10
III. 4. Question 4.....	11
III. 5. Question 5.....	12
Conclusion.....	12
Annexe Bibliographique.....	13



## Introduction

Au cours de ce TP, nous allons tenter d'apprendre à gérer une optimisation de programmation par utilisation du cache. Nous chercherons ainsi à comprendre comment interviennent les différents paramètres du cache au sein de notre programmation ainsi que la manière d'optimiser ceux-ci.

## I. Exercice 1

### I. 1. Question 1

Nous avons codé l'allocation des matrices et leurs libérations via un malloc, le code est joint en annexe. On se propose de remonter les traces d'exécution de Valgrind. Pour chaque implémentation.

*Fig. 1. Retour de Valgrind pour l'allocation colonne*

```
lince@Atmosphere:~/Divers/Cours/Esiee/I4/ARCH/TP2  valgrind  --tool=cachegrind  ./prog
colonne
==4600== Cachegrind, a cache and branch-prediction profiler
==4600== Copyright (C) 2002-2010, and GNU GPL'd, by Nicholas Nethercote et al.
==4600== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==4600== Command: ./prog colonne
==4600==
--4600-- warning: Unknown Intel cache config value (0xdd), ignoring
==4600==
==4600== I  refs:    19,226,155
==4600== I1 misses:    736
==4600== L2i misses:    710
==4600== I1 miss rate:    0.00%
==4600== L2i miss rate:    0.00%
==4600==
==4600== D  refs:    12,119,714 (11,089,759 rd + 1,029,955 wr)
==4600== D1 misses:    2,128,088 ( 1,127,770 rd + 1,000,318 wr)
==4600== L2d misses:    128,836 (  65,027 rd +  63,809 wr)
==4600== D1 miss rate:    17.5% (  10.1% +  97.1% )
==4600== L2d miss rate:    1.0% (   0.5% +   6.1% )
==4600==
==4600== L2 refs:    2,128,824 ( 1,128,506 rd + 1,000,318 wr)
==4600== L2 misses:    129,546 (  65,737 rd +  63,809 wr)
==4600== L2 miss rate:    0.4% (   0.2% +   6.1% )
```

On observe ici une importante utilisation des différents caches notamment du cache D1 qui est beaucoup sollicité, retransmettant un nombre important de références. On observe d'autre part que le nombre défauts de ce cache est assez important (miss) avec 17,5% de défauts. Les autres caches étant eux aussi relativement sollicités ne présentent que de faibles taux de défaut.



Fig. 2. Retour de Valgrind pour l'allocation ligne.

```

lince@Atmosphere:~/Divers/Cours/Esiee/I4/ARCH/TP2 valgrind --tool=cachegrind ./prog ligne
==4604== Cachegrind, a cache and branch-prediction profiler
==4604== Copyright (C) 2002-2010, and GNU GPL'd, by Nicholas Nethercote et al.
==4604== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==4604== Command: ./prog ligne
==4604==
--4604-- warning: Unknown Intel cache config value (0xdd), ignoring
==4604==
==4604== I refs:    19,226,172
==4604== I1 misses:    738
==4604== L2i misses:    712
==4604== I1 miss rate:    0.00%
==4604== L2i miss rate:    0.00%
==4604==
==4604== D refs:    12,119,720 (11,089,762 rd + 1,029,958 wr)
==4604== D1 misses:    127,256 ( 64,430 rd + 62,826 wr)
==4604== L2d misses:    126,961 ( 64,149 rd + 62,812 wr)
==4604== D1 miss rate:    1.0% ( 0.5% + 6.0% )
==4604== L2d miss rate:    1.0% ( 0.5% + 6.0% )
==4604==
==4604== L2 refs:    127,994 ( 65,168 rd + 62,826 wr)
==4604== L2 misses:    127,673 ( 64,861 rd + 62,812 wr)
==4604== L2 miss rate:    0.4% ( 0.2% + 6.0% )

```

On peut observer à quel point la copie par ligne est plus effective que celle par colonne : le taux de défaut du cache D1 passe de 2M à 127K, soit réduit de plus de ....

Fig. 3. Retour de Valgrind avec l'utilisation de memcpy.

```

lince@Atmosphere:~/Divers/Cours/Esiee/I4/ARCH/TP2 valgrind --tool=cachegrind ./prog memcpy
==4656== Cachegrind, a cache and branch-prediction profiler
==4656== Copyright (C) 2002-2010, and GNU GPL'd, by Nicholas Nethercote et al.
==4656== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==4656== Command: ./prog memcpy
==4656==
--4656-- warning: Unknown Intel cache config value (0xdd), ignoring
==4656==
==4656== I refs:    221,099
==4656== I1 misses:    738
==4656== L2i misses:    712
==4656== I1 miss rate:    0.33%
==4656== L2i miss rate:    0.32%
==4656==
==4656== D refs:    118,210 (88,097 rd + 30,113 wr)
==4656== D1 misses:    2,115 ( 1,794 rd + 321 wr)
==4656== L2d misses:    1,732 ( 1,438 rd + 294 wr)
==4656== D1 miss rate:    1.7% ( 2.0% + 1.0% )
==4656== L2d miss rate:    1.4% ( 1.6% + 0.9% )

```



```

==4656==
==4656== L2 refs:      2,853 ( 2,532 rd +  321 wr)
==4656== L2 misses:    2,444 ( 2,150 rd +   294 wr)
==4656== L2 miss rate:  0.7% (  0.6%  +  0.9% )

```

## I. 2. Question 2

Fig. 4. Sortie CacheGrind

```

desc: I1 cache:      32768 B, 64 B, 4-way associative
desc: D1 cache:      32768 B, 64 B, 8-way associative
desc: L2 cache:      262144 B, 64 B, 8-way associative
cmd: ./prog memcpy
events: Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw

```

Fig. 5. Sortie CacheGrind Esiee – Acme1

```

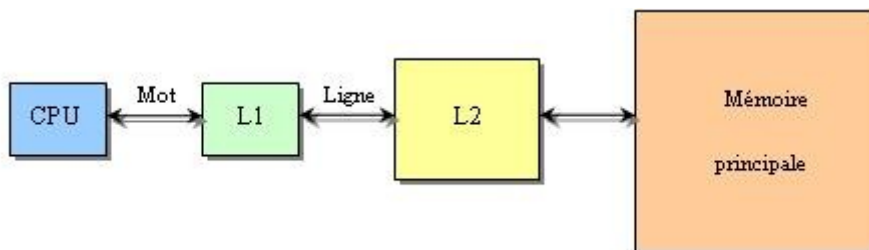
acme1:~/archtp2> cat cachegrind.out.10951 | more
desc: I1 cache:      32768 B, 64 B, 8-way associative
desc: D1 cache:      32768 B, 64 B, 8-way associative
desc: L2 cache:      6291456 B, 64 B, 24-way associative
cmd: ./prog memcpy
events: Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw

```

On peut observer que l'outil CacheGrind appelle le cache L3, L2 dans le cas de la première machine dû au fait que celui-ci ne reconnaît pas l'architecture, il assimile donc les deux caches. On observe dans les relevés deux niveaux d'efférents de cache : I1 et D1, renforcé par un second niveau de cache unifié L2. Cela correspond aux différents cache d'instructions et de données.

Sur la première configuration et la seconde on observe une taille identique dans les deux caches de niveaux inférieurs (D et I). Cependant, on observe une différence entre les caches au niveau L2 (même si celle-ci n'est dûe qu'à la non détection de CacheGrind du niveau L3 : spécification des caches suivante L1 I cache: 32K, L1 D cache: 32K , L2 cache: 256K, L3 cache: 3072K [Obtenue par un *dmseg | grep cache*]).

Fig. 6. Représentation de la mémoire cache microprocesseur.<sup>1</sup>



Dans le schéma ci-dessus, le bloc CPU contient les caches séparés de données et d'instructions.



## II. Exercice 2

### II. 1. Question 1

On utilise le programme ex1.sh légèrement modifié pour permettre l'utilisation de paramètre d'entrée pour notre programme. On lance celui-ci via la commande

```
./exo1.sh [nom_programme] [paramètre] [fichier de sortie]
```

Fig. 7. Trace console de l'exécution en ligne du programme ex1.sh

```
1024
way_1:==15686== D1 misses: 4,109,434 ( 3,106,618 rd + 1,002,816 wr)
way_2:==15689== D1 misses: 899,662 ( 578,538 rd + 321,124 wr)
way_4:==15692== D1 misses: 540,150 ( 281,945 rd + 258,205 wr)
way_8:==15695== D1 misses: 512,911 ( 260,538 rd + 252,373 wr)
2048
way_1:==15700== D1 misses: 4,059,149 ( 3,056,953 rd + 1,002,196 wr)
way_2:==15707== D1 misses: 704,637 ( 418,329 rd + 286,308 wr)
way_4:==15711== D1 misses: 515,961 ( 263,049 rd + 252,912 wr)
way_8:==15714== D1 misses: 508,541 ( 257,161 rd + 251,380 wr)
4096
way_1:==15719== D1 misses: 4,034,095 ( 3,032,215 rd + 1,001,880 wr)
way_2:==15722== D1 misses: 606,580 ( 337,754 rd + 268,826 wr)
way_4:==15725== D1 misses: 509,650 ( 258,002 rd + 251,648 wr)
way_8:==15728== D1 misses: 507,844 ( 256,574 rd + 251,270 wr)
8192
way_1:==15733== D1 misses: 4,012,183 ( 3,010,749 rd + 1,001,434 wr)
way_2:==15736== D1 misses: 557,086 ( 297,056 rd + 260,030 wr)
way_4:==15752== D1 misses: 507,994 ( 256,680 rd + 251,314 wr)
way_8:==15755== D1 misses: 507,560 ( 256,337 rd + 251,223 wr)
```

Fig. 8. Trace console de l'exécution memcpy du programme ex1.sh

```
1024
way_1:==15848== D1 misses: 16,567 (12,716 rd + 3,851 wr)
way_2:==15851== D1 misses: 13,128 (10,553 rd + 2,575 wr)
way_4:==15854== D1 misses: 12,700 (10,121 rd + 2,579 wr)
way_8:==15857== D1 misses: 13,019 (10,367 rd + 2,652 wr)
2048
way_1:==15862== D1 misses: 13,705 (10,475 rd + 3,230 wr)
way_2:==15865== D1 misses: 9,679 ( 7,879 rd + 1,800 wr)
way_4:==15868== D1 misses: 9,113 ( 7,412 rd + 1,701 wr)
way_8:==15871== D1 misses: 8,636 ( 6,979 rd + 1,657 wr)
4096
way_1:==15876== D1 misses: 12,239 ( 9,325 rd + 2,914 wr)
way_2:==15879== D1 misses: 8,628 ( 7,016 rd + 1,612 wr)
way_4:==15882== D1 misses: 7,991 ( 6,438 rd + 1,553 wr)
```



```

way_8:==15885== D1 misses: 7,938 ( 6,393 rd + 1,545 wr)
8192
way_1:==15890== D1 misses: 10,134 ( 7,706 rd + 2,428 wr)
way_2:==15893== D1 misses: 7,701 ( 6,330 rd + 1,371 wr)
way_4:==15896== D1 misses: 7,328 ( 5,948 rd + 1,380 wr)
way_8:==15899== D1 misses: 7,391 ( 5,948 rd + 1,443 wr)
    
```

Fig. 9. Tableau des défauts de cache en fonction de la taille du cache en ligne.

Taille D1		Misses
1024	Associativité	
	1	4,109,434
	2	899,662
	4	540,150
	8	512,911
2048	Associativité	
	1	4,059,149
	2	704,637
	4	515,961
	8	508,541
4096	Associativité	
	1	4,034,095
	2	606,580
	4	509,650
	8	507,844
8192	Associativité	
	1	4,012,183
	2	557,086
	4	507,994
	8	507,560



Fig. 10. Tableau des défauts de cache en fonction de la taille du cache avec memcpy.

Taille D1		Misses
1024	Associativité	
	1	16,567
	2	13,128
	4	12,700
	8	13,019
2048	Associativité	
	1	13,705
	2	9,679
	4	9,113
	8	8,636
4096	Associativité	
	1	12,239
	2	8,628
	4	7,991
	8	7,938
8192	Associativité	
	1	10,134
	2	7,701
	4	7,328
	8	7,391

Les résultats obtenus montrent bien que l'utilisation des fonctions optimisées fournis dans les bibliothèques C réduisent drastiquement le nombre de défaut de cache. D'autre part, on voit qu'en augmentant le nombre N de voies d'associativité du cache, on peut encore diminuer ces défauts. En doublant l'associativité, on réduit en moyenne de 20% la fréquence des défauts de cache tandis qu'en doublant la taille de cache, on réduit de 69% la fréquence des défauts de cache.



### III. Exercice 3

Le makefile permettant de compiler les sources n'étant pas fournies, on utilise un Makefile générique.

Au cours de notre observation des fichiers sources, on remarque ceux-ci font appelent à la librairie Pink. On se réfère donc à la documentation disponible en ligne<sup>2</sup>.

On rajoute le header « stdint.h » dans le fichier mcimage.h car la compilation des fichiers sources se solde par des erreurs de type inconnu pour int32.

Le main charge l'image passée en argument et effectue une opération dessus avant de rediriger le resultat vers une image de sortie ici égale à result.pgm.

Comme spécifié, la fonction allocimage alloue la mémoire nécessaire pour l'image de sortie. On fait ensuite appel à une fonction mean externe à la librairie, permettant de d'affaiblir le contraste de l'image en moyennant la valeur des pixels en fonction du paramètre renseigné en second.

#### III. 1. Question 1

Fig. 11. Retour de gprof

```
Each sample counts as 0.01 seconds.
% cumulative self      self   total
time seconds seconds  calls ms/call ms/call name
99.11    1.11    1.11     6 185.00 185.00 mean
0.89    1.12    0.01     6   1.67   1.67 readimage
0.00    1.12    0.00    12   0.00   0.00 allocimage
0.00    1.12    0.00     6   0.00   0.00 freeimage
0.00    1.12    0.00     6   0.00   0.00 writeimage
0.00    1.12    0.00     6   0.00   0.00 writerawimage
```

On peut observer que la fonction limitatrice du temps d'exécution est l'appel à la fonction mean. La fonction readimage nécessitant la lecture d'un fichier d'entrée, il est plutôt normale que ce soit la seconde fonction limitante.

#### III. 2. Question 2

Fig. 11. Retour de l'utilisation du cache du programme fournis

```
valgrind --tool=cachegrind ./prog im2.pgm 5
==13306== Cachegrind, a cache and branch-prediction profiler
==13306== Copyright (C) 2002-2010, and GNU GPL'd, by Nicholas Nethercote et al.
==13306== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==13306== Command: ./prog im2.pgm 5
==13306==
--13306-- warning: Unknown Intel cache config value (0xdd), ignoring
==13306==
==13306== I refs: 474,909,830
```



```

==13306== I1 misses:      2,408
==13306== L2i misses:    1,129
==13306== I1 miss rate:   0.00%
==13306== L2i miss rate:  0.00%
==13306==
==13306== D refs:  239,723,932 (182,117,459 rd + 57,606,473 wr)
==13306== D1 misses:  635,457 ( 323,133 rd + 312,324 wr)
==13306== L2d misses:  16,335 (  5,953 rd + 10,382 wr)
==13306== D1 miss rate:  0.2% (  0.1% +  0.5% )
==13306== L2d miss rate:  0.0% (  0.0% +  0.0% )
==13306==
==13306== L2 refs:      637,865 ( 325,541 rd + 312,324 wr)
==13306== L2 misses:    17,464 (  7,082 rd + 10,382 wr)

```

### III. 3. Question 3

En observant de plus près la fonction `mean`, on remarque que les opérations de celle-ci s'effectuent en colonnes et non en ligne, on décide donc d'échanger les deux boucles `for` tel que suivant :

Fig. 12. Échange des boucles `for` au sein de `function.h`

```

for (j=0; j < image->col_size; j++)
    for (i=0; i < image->row_size; i++){

```

Fig. 13. Trace d'exécution de Valgrind

```

==6334== I refs:  705,149,553
==6334== I1 misses:      2,408
==6334== L2i misses:    1,129
==6334== I1 miss rate:   0.00%
==6334== L2i miss rate:  0.00%
==6334==
==6334== D refs:  351,806,398 (281,408,829 rd + 70,397,569 wr)
==6334== D1 misses:    16,928 (  7,005 rd +  9,923 wr)
==6334== L2d misses:    16,528 (  6,628 rd +  9,900 wr)
==6334== D1 miss rate:  0.0% (  0.0% +  0.0% )
==6334== L2d miss rate:  0.0% (  0.0% +  0.0% )
==6334==
==6334== L2 refs:      19,336 (  9,413 rd +  9,923 wr)
==6334== L2 misses:    17,657 (  7,757 rd +  9,900 wr)
==6334== L2 miss rate:  0.0% (  0.0% +  0.0% )

```

On observe une baisse importante des défauts de cache.



**III. 4. Question 4***Fig. 14. Image de base avec paramètre par défaut.*

```

==7339== I refs:    705,149,553
==7339== I1 misses:    2,408
==7339== L2i misses:    1,129
==7339== I1 miss rate:    0.00%
==7339== L2i miss rate:    0.00%
==7339==
==7339== D refs:    351,806,398 (281,408,829 rd + 70,397,569 wr)
==7339== D1 misses:    16,928 ( 7,005 rd + 9,923 wr)
==7339== L2d misses:    16,528 ( 6,628 rd + 9,900 wr)
==7339== D1 miss rate:    0.0% ( 0.0% + 0.0% )
==7339== L2d miss rate:    0.0% ( 0.0% + 0.0% )
==7339==
==7339== L2 refs:    19,336 ( 9,413 rd + 9,923 wr)
==7339== L2 misses:    17,657 ( 7,757 rd + 9,900 wr)

```

Temps d'exécution, constaté : 0.307s

*Fig. 15. image de grande taille*

```

==7381== I refs:    13,982,222
==7381== I1 misses:    1,168
==7381== L2i misses:    1,111
==7381== I1 miss rate:    0.00%
==7381== L2i miss rate:    0.00%
==7381==
==7381== D refs:    6,615,268 (5,814,316 rd + 800,952 wr)
==7381== D1 misses:    2,625 ( 2,062 rd + 563 wr)
==7381== L2d misses:    2,045 ( 1,553 rd + 492 wr)
==7381== D1 miss rate:    0.0% ( 0.0% + 0.0% )
==7381== L2d miss rate:    0.0% ( 0.0% + 0.0% )
==7381==
==7381== L2 refs:    3,793 ( 3,230 rd + 563 wr)
==7381== L2 misses:    3,156 ( 2,664 rd + 492 wr)
==7381== L2 miss rate:    0.0% ( 0.0% + 0.0% )

```

Temps d'exécution constaté : 0.764s

*Fig. 16. image de taille moyenne*

```

==7546== I refs:    1,576,046
==7546== I1 misses:    1,159
==7546== L2i misses:    1,107
==7546== I1 miss rate:    0.07%
==7546== L2i miss rate:    0.07%
==7546==
==7546== D refs:    752,003 (644,262 rd + 107,741 wr)
==7546== D1 misses:    2,227 ( 1,954 rd + 273 wr)
==7546== L2d misses:    1,738 ( 1,508 rd + 230 wr)

```



```

==7546== D1 miss rate:    0.2% ( 0.3% + 0.2% )
==7546== L2d miss rate:   0.2% ( 0.2% + 0.2% )
==7546==
==7546== L2 refs:        3,386 ( 3,113 rd + 273 wr)
==7546== L2 misses:      2,845 ( 2,615 rd + 230 wr)
==7546== L2 miss rate:   0.1% ( 0.1% + 0.2% )

```

Temps d'exécution constaté : 0.276s

Fig. 17. image de petite taille

```

==7604== I refs:    1,202,439,269
==7604== I1 misses:    1,174
==7604== L2i misses:    1,135
==7604== I1 miss rate:    0.00%
==7604== L2i miss rate:    0.00%
==7604==
==7604== D refs:    567,922,015 (501,310,818 rd + 66,611,197 wr)
==7604== D1 misses:    27,022 ( 14,355 rd + 12,667 wr)
==7604== L2d misses:    26,590 ( 13,954 rd + 12,636 wr)
==7604== D1 miss rate:    0.0% ( 0.0% + 0.0% )
==7604== L2d miss rate:    0.0% ( 0.0% + 0.0% )
==7604==
==7604== L2 refs:    28,196 ( 15,529 rd + 12,667 wr)
==7604== L2 misses:    27,725 ( 15,089 rd + 12,636 wr)
==7604== L2 miss rate:    0.0% ( 0.0% + 0.0% )

```

Temps d'exécution constaté : 0.041s

Fig. 18. image de très grande taille

```

==7697== I refs:    4,092,449,530
==7697== I1 misses:    1,174
==7697== L2i misses:    1,135
==7697== I1 miss rate:    0.00%
==7697== L2i miss rate:    0.00%
==7697==
==7697== D refs:    1,932,902,529 (1,706,251,840 rd + 226,650,689 wr)
==7697== D1 misses:    85,955 ( 43,802 rd + 42,153 wr)
==7697== L2d misses:    85,521 ( 43,399 rd + 42,122 wr)
==7697== D1 miss rate:    0.0% ( 0.0% + 0.0% )
==7697== L2d miss rate:    0.0% ( 0.0% + 0.0% )
==7697==
==7697== L2 refs:    87,129 ( 44,976 rd + 42,153 wr)
==7697== L2 misses:    86,656 ( 44,534 rd + 42,122 wr)
==7697== L2 miss rate:    0.0% ( 0.0% + 0.0% )

```

Temps d'exécution constaté : 1.669s

On peut en déduire que plus la taille de l'image augmente plus le nombre de références aux caches augmentent et ainsi plus il y a de risque de défaut de cache importants.



## Conclusion

Au cours de ce projet, nous avons pu apprendre à évaluer les performances de la mémoire cache processeur de à l'aide des outils fournis par le logiciel Valgrind, nous avons vu comment ceux-ci intervenaient lors de l'exécution de programme. Nous avons pu voir comment la programmation logicielle intervenait sur l'optimisation de l'utilisation de cette mémoire.

## Annexe Bibliographique

- <sup>1</sup>: Figure 6 extrait de Wikipédia à [http://fr.wikipedia.org/wiki/M%C3%A9moire\\_cache](http://fr.wikipedia.org/wiki/M%C3%A9moire_cache)  
Différents niveaux de mémoire d'un microprocesseur
- <sup>2</sup>: Site externe de la librairie Pink, disponible à l'adresse <http://pinkhq.com/>.