

Architecture des ordinateurs

L3 Informatique 2020-2021

Binôme :

TP 3 - Mémoire cache

- Chaque binôme rendra une feuille d'énoncé complétée et la déposera sur *e-learning* avec les programmes écrits pour répondre aux questions.
- Les exercices de ce TP sont à programmer **en langage C**.
- Ce TP a l'objectif de comprendre le fonctionnement d'une mémoire cache et de mettre en évidence divers phénomènes qui en découlent.
- Toutes les mesures de temps doivent être faites au moyen de la fonction `print_timing` écrite au TP 2.

Exercice 1. ★ Simulation d'une hiérarchie à 2 niveaux On a construit une liste chaînée de 27 cellules, chacune occupant 12 octets en mémoire. À l'issue d'une séquence d'insertion, les cellules sont stockées comme suit en mémoire (avec la position dans la liste) :

| Adresse | Position | Adresse | Position | Adresse | Position |
|------------|----------|------------|----------|------------|----------|
| 0xAE01D500 | 1 | 0xAE01D900 | 16 | 0xAE01DD00 | 9 |
| 0xAE01D50C | 3 | 0xAE01D90C | 20 | 0xAE01DD0C | 10 |
| 0xAE01D518 | 4 | 0xAE01D918 | 2 | 0xAE01DD18 | 5 |
| 0xAE01D524 | 8 | 0xAE01D924 | 24 | 0xAE01DD24 | 26 |
| 0xAE01D530 | 12 | 0xAE01D930 | 18 | 0xAE01DD30 | 21 |
| 0xAE01D53C | 14 | 0xAE01D93C | 6 | 0xAE01DD3C | 15 |
| 0xAE01D548 | 17 | 0xAE01D948 | 27 | 0xAE01DD48 | 22 |
| 0xAE01D554 | 19 | 0xAE01D954 | 7 | 0xAE01DD54 | 13 |
| 0xAE01D560 | 25 | 0xAE01D960 | 11 | 0xAE01DD60 | 23 |

On suppose que la hiérarchie mémoire consiste en la RAM principale et un unique niveau de cache de paramètres suivants :

direct mapping, lignes de 64 octets, taille totale 1024 octets.

On appelle *bloc* une zone de la mémoire RAM qui est chargée dans une ligne de cache (par exemple les octets d'adresses 0 à 63).

a. Combien de lignes ce cache contient-il ?

b. Les cellules suivantes sont-elles contenues dans un **même bloc** en mémoire principale ?

1, 3, 4 12, 14, 17 8, 10, 12 12, 14

c. Les blocs contenant les cellules suivantes sont-ils chargés dans la **même ligne** de cache ?

8, 9 10, 11 21, 15 15, 22

d. On parcourt la liste complètement, de 1 à 27. Quelles cellules occasionnent un *cache-miss* lorsqu'on les traverse ?

- e. On suppose maintenant que la hiérarchie mémoire consiste en la RAM principale et un unique niveau de cache de paramètres suivants :

2-way associatif (LRU), lignes de 64 octets, taille totale 1024 octets.

Combien de cache-miss la traversée complète de la liste occasionne-t-elle ?

Exercice 2. ★★ Accès espacés à un tableau

On se propose de faire un grand nombre N d'accès dans un tableau, par exemple $N = 10\,000\,000$, en espaçant les accès d'un pas fixe : l'accès à la case i sera suivi de l'accès à la case $i + pas$.

- Écrivez une fonction prenant en arguments deux entiers n et pas et un tableau tab de taille n , et qui accède à N cases de tab en lisant à chaque étape la case située pas cases après la précédente case. Dans le cas où vous devriez sortir du tableau, repartez à partir du début (ie. l'indice est calculé modulo la taille du tableau).
- Mesurez les temps d'exécution de cette fonction pour les valeurs de n et de pas suivantes.

| pas | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|--------------|---|---|---|---|----|----|----|-----|
| $n = 2^{10}$ | | | | | | | | |
| $n = 2^{16}$ | | | | | | | | |
| $n = 2^{17}$ | | | | | | | | |
| $n = 2^{18}$ | | | | | | | | |
| $n = 2^{19}$ | | | | | | | | |
| $n = 2^{20}$ | | | | | | | | |
| $n = 2^{21}$ | | | | | | | | |
| $n = 2^{22}$ | | | | | | | | |
| $n = 2^{30}$ | | | | | | | | |

- À partir de quelle taille de tableau le temps d'exécution commence à augmenter significativement avec le pas ?
- Que pouvez vous en déduire sur votre cache ?

- Analysez le comportement du programme en utilisant `perf_4.9`.

Exercice 3. ★★ Accès répétitifs

Pour cette expérience, on utilise un très grand tableau d'entiers (par exemple, de taille $size = 10^9$). On se propose de répéter un grand nombre n de fois un petit nombre k d'accès à des cases séparées d'une grande puissance de 2 (par exemple $2^{20} = 1048576$). On souhaite faire l'expérience pour différentes valeurs de k , de telle façon que $N = n \times k$ soit constant. On choisit $N = 2^{28} = 268435456$.

- Pourquoi veut-on que N soit constant ?
- Écrivez une fonction mettant en place cette expérience en limitant autant que possible la mesure au temps d'accès au tableau.
- Mesurez les temps d'exécution de cette fonction pour les valeurs de k indiquées ci-dessous.

| k | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|----|----|----|-----|
| | | | | | | |

Ces mesures permettent-elles de conclure quelque chose ?

- Analysez le comportement des différentes mesures (pour chaque valeur de k) au moyen de l'outil `perf`. On s'intéressera notamment aux paramètres du cache de niveau 1. Ces analyses permettent-elles de conclure quelque chose ?
- Allez chercher dans le répertoire `/sys/devices/system/cpu/cpu0/cache` les paramètres du cache de niveau 1 (on précisera les unités) : taille totale du cache, taille d'une ligne de cache, associativité. Notez les ci-dessous.
- Analysez, au vu des paramètres de la question précédente, les mesures effectuées par `perf`.

Exercice 4. ★★ Simulateur de cache non associatif On souhaite simuler les accès à un cache non associatif de 2^{15} octets avec des lignes de 64 octets. Pour cela, on va conserver, pour chaque ligne de cache, un entier représentant l'indicateur du bloc stocké en cache lors d'un accès fictif à une adresse donnée. Pour simplifier le code, ce tableau sera global, ainsi que les autres informations nécessaires à la modélisation du cache.

- Écrire une fonction `cache_simulator` qui prend en paramètre l'adresse d'une case mémoire et simule un accès en cache pour cette case. Elle renvoie 1 si c'est un cache miss et 0 sinon. Pour tester, nous vous conseillons d'afficher les informations suivantes :
 - le numéro de ligne de cache correspondant,
 - l'identifiant en RAM correspondant,
 - si cela provoque un cache-miss ou non.
- En utilisant la fonction précédente, écrire des fonctions `acces_seq` et `acces_alea` qui prennent un entier n en paramètre et permettent de **simuler** n accès (séquentiels pour la première, aléatoires pour la seconde) à un tableau de taille n .
- Compléter le tableau suivant avec vos observations sur le nombre de cache-misses :

| | $n = 2^{10}$ | $n = 2^{12}$ | $n = 2^{13}$ | $n = 2^{14}$ | $n = 2^{15}$ |
|------------|--------------|--------------|--------------|--------------|--------------|
| Séquentiel | | | | | |
| Aléatoire | | | | | |

Exercice 5. ★★★ Transposer une matrice

On s'intéresse dans cet exercice à la manipulation de très grands tableaux 2D par l'exemple d'opérations courantes sur les matrices. L'objectif est de mettre au point un algorithme qui tire parti de la hiérarchie de mémoire.

- On commence par représenter une matrice par un simple tableau 2D. Écrire une fonction qui crée une matrice $m \times n$ d'entiers initialisée d'entiers aléatoires.
- Écrire une fonction qui prends en entrée deux entiers m, n et une matrice $m \times n$ et retourne la matrice $n \times m$ obtenue en la transposant. Mesurez ses temps d'exécution pour différentes tailles de matrice.

| $n = m$ | 1 000 | 5 000 | 10 000 | 20 000 | 50 000 |
|---------|-------|-------|--------|--------|--------|
| temps | | | | | |

- Pourquoi les petites matrices permettent de limiter (en proportion) les `cache miss` ?
- Soit $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ une matrice $m \times n$, où A, B, C et D sont des matrices (ce sont des *sous-matrices* de M). On rappelle que $M^t = \begin{pmatrix} A^t & C^t \\ B^t & D^t \end{pmatrix}$, ce qui permet de réduire la transposition d'une grande matrice à celle de petites matrices. Écrire une nouvelle fonction de transposition qui se base sur cette idée et mesurez ses temps d'exécution pour différentes tailles de matrice.

| $n = m$ | 1 000 | 5 000 | 10 000 | 20 000 | 50 000 |
|---------|-------|-------|--------|--------|--------|
| temps | | | | | |

- Étendez cette idée récursivement.