

Architecture des ordinateurs

L3 Informatique 2020-2021

Binôme :

TP 2 - mesure du temps d'exécution

- Chaque binôme rendra une feuille d'énoncé complétée et la déposera sur *e-learning* avec les programmes écrits pour répondre aux questions.
- Pour ce TP, écrivez les programmes **en C (et compilez sans optimisation)**.
- Ce TP a les objectifs suivants :
 - a. Découvrir certains des outils disponibles pour mesurer le temps d'exécution d'un programme C.
 - b. Observer des sources d'imprécision de ces outils de mesure, et apprendre à les corriger.
 - c. Écrire une fonction `print_timing` qui mesure le temps mis par une autre fonction (donnée en paramètre) en évitant certaines causes d'imprécision.
 - d. Utiliser cette fonction de mesure pour mettre en évidence des phénomènes liés aux mémoires cache.
- Il est **très important** que la fonction `print_timing` soit correctement écrite et facilement utilisable à la fin de cette séance. En effet, nous l'utiliserons pour **toutes** les mesures de temps faites à partir du TP 3 (c'est à dire dans la plupart des exercices des TP 3, 4 et 5).

Exercice 1. ★ Outils disponibles pour mesurer un temps d'exécution.

Nous allons tester différentes façons de mesurer la temps d'exécution d'un programme en C.

- a. Écrire un programme qui calcule la somme des carrés des entiers de 0 jusqu'à n , avec n pris en paramètre de la ligne de commande.
- b. Mesurer le temps d'exécution de votre programme pour différentes valeurs de n ,
 - en utilisant la commande linux `time`,
 - en utilisant la fonction `time()` de la bibliothèque `time.h`,
 - en utilisant `clock()` et `CLOCKS_PER_SEC` de la bibliothèque `time.h`,
 - en utilisant ¹ `__rdtscp()` de la bibliothèque `x86intrin.h`,
 - en utilisant la commande linux `perf` avec comme arguments `stat` et votre programme.

n	cmd time	time()	clock()	__rdtscp()	cmd perf
10 ³					
10 ⁶					
10 ⁸					
10 ⁹					

- c. Commenter les avantages et inconvénients de chaque méthode.

1. <https://docs.microsoft.com/en-us/cpp/intrinsics/rdtscp>

Exercice 2. ★ ★ Sources d'imprécision des outils de mesure

- Écrire un programme qui déclare un tableau `TAB` de $N = 10^9$ entiers (ou 10^8 si c'est trop lent), alloué en utilisant `malloc` (pour la suite, ça pourra être pratique de le déclarer comme une variable globale).
- Écrire une fonction `acces_seq` qui prend un entier `n` en paramètre et accède en écriture à `n` cases de `TAB` dans l'ordre (accès séquentiel). Mesurer le temps d'exécution d'un appel à `acces_seq` pour différentes valeurs de `n` (10^3 , 10^5 , 10^7 , ...) dans des exécutions différentes.
- Appeler deux fois de suite (dans la même exécution) la fonction `acces_seq` et mesurer le temps d'exécution de chacun des deux appels, toujours pour différentes valeurs de `n`.
- Faire la même chose avec 3 appels (dans la même exécution) et remplir le tableau. Compléter également la dernière case avec la valeur $E_{max} = 100 \times \frac{\text{écart max.}}{\text{temps moyen}}$ (cela correspond au pourcentage du temps moyen que représente l'écart maximum entre les temps mesurés).

n appel	1000	10^5	10^7	10^9
1er				
2eme				
3eme				
E_{max} (en %)				

Que constatez-vous ?

- Écrire une deuxième fonction `acces_alea` qui effectue `n` accès aléatoires au tableau `TAB`.
- Appeler dix fois de suite la fonction `acces_alea`, mesurer (afficher) le temps d'exécution de chacun des dix appels, toujours pour différentes valeurs de `n`, et calculer le temps moyen d'exécution d'un appel ainsi que l'écart maximal entre 2 temps d'exécution (pour la même valeur de `n`), afin d'obtenir E_{max} .

n	1000	10^4	10^6	10^8
temps moyen				
écart max.				
E_{max} (en %)				

Que constatez-vous ?

Exercice 3. ★ Écriture de la fonction `print_timing`.

En tenant compte des observations de l'exercice précédent, on va écrire une fonction "intelligente" `print_timing` de mesure du temps d'exécution d'une fonction, selon le squelette suivant :

```
1 void print_timing(int n, int (*f_to_time)(int)){
2
3     // écrivez votre code ici
4     // il peut appeler la fonction à mesurer par f_to_time(n)
5
6     // le résultat de la mesure (corrigeant les imprécision) en secondes
7     // doit être dans une variable `mesure` (de type float)
8
9     printf("n=%d : %.10f sec\n",n, mesure);
10 }
```

Comme on le voit sur le squelette, la fonction à mesurer doit prendre en paramètre un entier et renvoyer un entier. Elle est transmise à la fonction `print_timing` via un pointeur de fonction. Autrement dit, pour utiliser `print_timing` sur la fonction

```
1 int test(int a){
2
3     // ...
4
5 }
```

pour le paramètre $a = 12$ il suffira d'appeler la commande

```
1 print_timing(12,test);
```

- a. Comment résoudre le problème identifié à la question 2d ?
- b. Comment tirer avantage de ce que l'on a observé à la question 2f ?
- c. **Écrire la fonction `print_timing` en utilisant les remarques ci-dessus et la tester.**
- d. Pourquoi n'a-t-on pas fait ces premiers exercices en Python ou Java ?
- e. Écrire une fonction similaire `print_cycles` qui compte le nombre de cycles du processeur.

Exercice 4. ★ ★ Accès séquentiels VS accès aléatoires.

On se propose de faire un grand nombre N d'accès dans un tableau et de comparer le temps pris quand ces accès sont consécutifs au temps pris quand ils sont aléatoires. À partir de maintenant, les mesures de temps se feront au moyen de la fonction `print_timing`.

- a. Reprendre le programme de l'exercice 2 et comparer les temps d'exécution de `acces_seq` et `acces_alea` pour différentes valeurs de `n`.

	$n = 1000$	$n = 10^6$	$n = 10^7$	$n = 10^8$	$n = 10^9$
Séquentiel					
Aléatoire					

- b. Donner au moins deux raisons qui peuvent expliquer que ces temps sont différents.
- c. Refaire vos mesures en commençant par préparer des tableaux auxiliaires contenant la liste des accès à effectuer (dans l'ordre dans un cas, dans un ordre aléatoire dans l'autre). On exclura des mesures le temps de préparation du tableau auxiliaire.

	$n = 1000$	$n = 10^6$	$n = 10^7$	$n = 10^8$	$n = 10^9$
Séquentiel					
Aléatoire					

- d. Pour quelles raisons ces temps sont-ils différents ?
- e. Qu'est-ce que ça donne si vous compilez avec les optimisations sde gcc ?

Exercice 5. ★★★ Estimation des paramètres de cache

- a. Mesurez les temps d'accès séquentiels et aléatoires (comme à l'exercice 4) pour diverses valeurs de n et tracez les courbes des temps d'accès, de préférence en fonction de $\log_{10} n$.
- b. Utilisez les tests que vous venez de faire pour estimer la taille du cache de dernier niveau.
- c. Comparez votre réponse à la question précédente aux informations qui se trouvent dans le répertoire : `/sys/devices/system/cpu/cpu0/cache`.