TP : Mesure de performances sur les mémoires cache (2h)

L'objectif de cette séance de TP est de montrer l'intérêt des mémoires cache et leur influence sur les performances d'exécution d'un programme.

La bibliothèque <x86intrin.h> contient un ensemble de fonctions permettant de manipuler le matériel de votre ordinateur. Cette bibliothèque contient notamment une fonction *__rdtscp(unsigned int *)* qui permet de récupérer la valeur sur 64 bits du compteur de temps qui s'incrémente tous les cycles d'horloge du processeur (CF annexe pour la définition de cette fonction). Malheureusement, cette fonction n'est plus autorisée à l'exécution par certaines versions de Linux pour des raisons de sécurité. Nous utiliserons donc la version en langage d'assemblage x86. Les programmes fonctionnent sous Linux, sous Mac et certainement sous Windows (non testés).

1 – Principe de mesure du temps d'exécution d'un programme

Vous trouverez dans le répertoire *Partage/TP_Caches/* un fichier *pgm0.c* qui contient un ensemble d'instructions permettant de mesurer le temps d'exécution d'un programme ou bout de programme. Dans ce fichier, la fonction *__rdtscp()* a été mise en commentaires et remplacer par sa version en langage d'assemblage. En effet, il existe une instruction *rdtsc* en langage d'assemblage x86 qui permet de récupérer le nombre de cycles du processeur depuis son démarrage (https://fr.wikipedia.org/wiki/RDTSC).

- Compilez ce programme à l'aide de la commande `gcc pgm0.c –o pgm0`
- Exécutez-le plusieurs fois. Vous devriez observer quelques variations mais on mesure quelques cycles d'horloge à 2,93 GHz (fréquence du processeur des machines de la salle 216 – vous pouvez vérifier en tapant : `cat /proc/cpuinfo`).
- Utilisez une boucle en bash pour l'exécuter une dizaine de fois. Par exemple avec la commande : `for ((i=0;i<10;i++)) do ./pgm0; sleep 1; done`

2 – Intérêt des lignes de cache

Le fichier *pgm1.c* contient la déclaration d'un tableau et son initialisation. On cherche ensuite à mesurer l'influence des lignes de caches sur un calcul effectué avec les éléments de ce tableau.

- Compilez et exécutez ce programme.
- Dans la bibliothèque <x86intrin.h>, il existe une fonction `_mm_clflush(&x);` qui vide la ligne de cache contenant la variable x. Insérer cette instruction dans la boucle et comparez les performances en vidant la ligne de cache.

3 – Ordre dans les boucles

Le fichier *pgm2.c* contient la déclaration d'un tableau à 2 dimensions et son initialisation. On cherche ensuite à mesurer l'influence de l'ordre du traitement lignes-colonne. On souhaite calculer la somme des éléments de ce tableau.

- Proposer 2 programmes en inversant l'ordre de traitement ligne-colonne de ce calcul de la somme des éléments.
- Indiquez lequel est le plus rapide et pourquoi ?
- Avant l'instruction de traitement, videz la ligne de cache et observer la qualité du résultat.
- Vérifier que les résultats restent cohérents pour un tableau de float.

4 – Attaque par canal caché !

En janvier 2018, deux failles de sécurité ont été annoncés : Meltdown et Spectre. Si l'explication technique est complexe, on peut retenir que le principe est de récupérer des informations présentes dans les lignes de cache dans certaines conditions.

Frédéric Pétrot, professeur en informatique à l'ENSIMAG a proposé un ensemble de tests, de programmes et d'explications pour comprendre l'idée de ces failles de sécurité :
https://ensiwiki.ensimag.fr/images/5/5b/Sm.pdf
Il a par exemple montré qu'il est possible de repérer les lignes de cache accédées récemment. On appelle cela un canal caché. C'est le programme pgm3.c légèrement adapté à notre environnement et en supposant que les lignes des caches sont de 64 octets.

- Compilez et exécuter le programme. La commande `gcc -DINVALIDATE=0 pgm3.c`. n'exécute pas la boucle d'instructions d'invalidation. La commande `gcc -DINVALIDATE=1 pgm3.c` exécute la boucle d'instructions d'invalidation.
- Observer les résultats sur un ensemble d'exécutions.

Information inattendue accessible!

On peut savoir si une ligne de cache a déjà été lue en mesurant le temps qu'il faut pour lire une donnée qu'elle contient
C'est un canal caché qu'un attaquant peut exploiter![a]

a. Timing (Side-channel) attack.

5 – Transmission d'information par canal caché !

On peut maintenant utiliser cette notion de canal caché pour transmettre une information entre 2 fonctions. Considérons un programme principal qui crée un tableau (cf fichier *side_channel .c*). Une fonction *side_channel_write()* invalide une ligne du cache sur une valeur particulière à transmettre. La fonction *side_channel_read()* vient ensuite mesurer le temps d'accès aux différents éléments du tableau. La plus grande valeur correspond alors à la ligne de cache qui a été invalidée. C'est une méthode pour transmettre une information d'une fonction à une autre par canal caché.

- Comprendre le programme *side_channel.c* donné. Vous noterez par exemple l'utilisation d'un tableau à 2 dimensions (de taille 17 pour transmettre une information entre 1 et 16 inclus).
- Compilez et exécuter le programme en utilisant l'option -O3 de gcc.
- Observer les résultats sur un ensemble d'exécutions.

Annexe : Source : Intel®64 and IA-32 Architectures Software Developer's Manual Vol. 2,

Instruction rdtscp p. 4-547

Instruction assembleur rdtscp: Reads the current value of the processor's time-stamp counter (a 64-bitMSR) into the EDX :EAX registers and . . . The processor monotonically increments the time-stamp counter MSR every clock cycle and resets it to 0 whenever the processor is reset.

Instruction cflush p. 3-139

Insruction assembleur clflush: Invalidates from every level of the cache hierarchy in the cache coherence domain the cache line that contains the linear address specified with the memory operand.