Compilation et Optimisation de code

ENSSAT - Systèmes Numériques 3

Camille Oudot <camille.oudot@gmail.com>

Francois Charot, charot@irisa.fr

TP 2 : Mémoires caches et temps d'accès à la mémoire

Objectif : concevoir un programme qui permet de mesurer le temps d'accès à des données en mémoire. Ce programme servira à construire une courbe montrant l'influence de la "distance" de la donnée (niveaux de cache) sur son temps d'accès.

1. Observations

Le noyau Linux (et la philosophie Unix en général) représente la plupart des concepts qu'il manipule sous forme de fichiers. On trouve notamment dans `/dev/` les périphériques d'entrées/sorties (disques, réseau, imprimantes, ...), dans `/proc/` les procesus en cours d'exécution et dans `/sys/` les caractérisques et paramètres de votre système.

En observant l'arborescence `/sys/devices/system/cpu/cpu0/cache/`, donner toutes les caractéristques de tous les caches des processeurs de la machine : quel est leur niveau, type, taille, type d'associativité, taille d'une ligne de cache, est-il commun ou partagé entre les cœurs ?

Les commentaires dans le code source du noyau Linux peuvent vous aider à clarifier la signification de ces fichiers :

https://elixir.free-electrons.com/linux/v4.20.17/source/include/linux/cacheinfo.h#L21

2. Memwalk

Dans toute la suite, nous allons utiliser un programme en C, **memwalk**, qui permettra de parcourir des zones mémoire de la taille souhaitée, d'y lire et d'y écrire des données, et de mesurer le temps nécessaire pour l'accès à une adresse particulière ce dette zone.

Cette zone mémoire sera découpée en petites cellules, qui sont des structures à deux champs :

- **next** (8 octets), pointeur vers une aurtre cellule
- **value** (8 octets), un nombre quelconque, valeur de la cellule

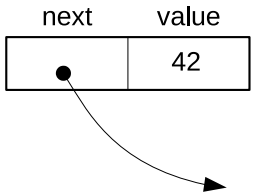

Figure 1: structure d'une cellule de **memwalk**

Ces cellules seront organisée en une liste chaînée circulaire (la dernière cellule pointe vers la première de la liste)

Utilisation de memwalk

Quand il sera complété complété, vous pourrez appeler **memwalk** en spécifiant les paramètres suivants :

- `-m TAILLE` : taille, en octets, de la zone mémoire à allouer

- `-n NOMBRE` : nombre d'accès mémoire (dans la fonction `walk()`) à effectuer
- `-r` : si l'option est présente, les cellules de la liste chaînée seront chaînées en ordre aléatoire (pour la partie **4.**)

memwalk affiche avant de se terminer la taille de la zone mémoire, et le nombre de cycles processeur nécessaire en moyenne pour accéder à une cellule.

Exemple :

```
$ ./memwalk -m 128 -n 100000000
128 7.027021
```

> **Interprétation :** La visite de 100 millions de cellules d'une liste chaînée circulaire de 128 cellules a pris en moyenne 7 cycles processeurs par cellule.

Il faut choisir des valeurs élevées de `NOMBRE` (option `-n`), pour réduire la marge d'erreur de la mesure.

3. Parcours séquentiel de la mémoire

Récupérer le code de **memwalk** sur l'ENT (**tp2_assets.tgz**).

Q1 : Compléter **memwalk.c** pour allouer la mémoire nécessaire à la liste chaînée, et initialiser les pointeurs `next` pour un parcours séquentiel, comme sur le schéma suivant :

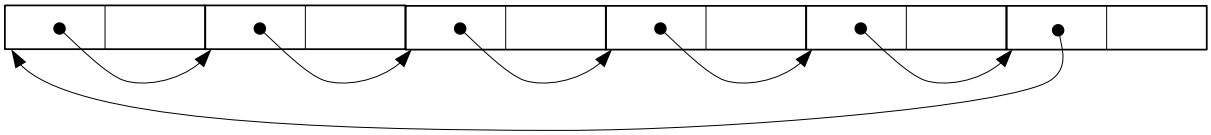

Figure 2: Pointeurs de liste permettant un parcours séquentiel

Q2 : Dans le même fichier, compléter la fonction **walk()** pour effectuer le nombre demandé (**steps**) de "visites" de cellules avec incrément de `value`. Il faut incrémenter le champ `value` de la cellule courante, puis passer à la cellule suivante.

Q3 : Mesurer le nombre de cycles nécessaires en moyenne pour accéder à une cellule, pour 100 millions d'accès, avec des listes occupant 1 kio, 10 kio, 100 kio, 1 Mio, 10 Mio et 100 Mio. Conclure.

4. Parcours aléatoire de la mémoire

Q1 : Compléter **memwalk.c** pour proposer le parcours de listes chaîneés composées de cellules chaînées en ordre aléatoire quand l'option `-r` est passée (compléter et utiliser **shuffle.c**)

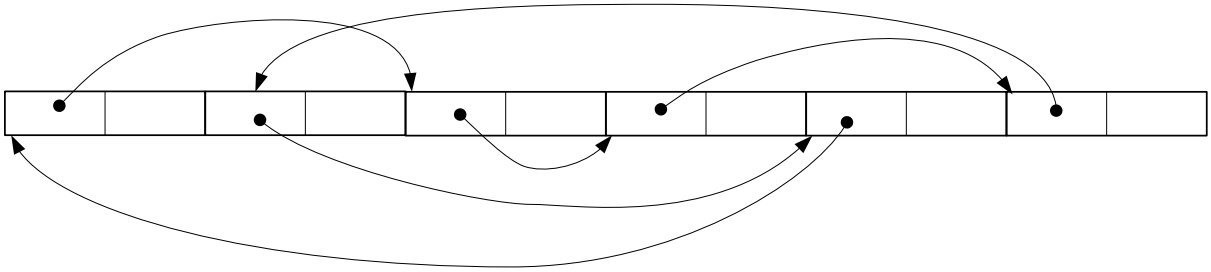

Figure 3: Pointeurs de liste permettant un parcours aléatoire

Q2 : Mesurer le nombre de cycles nécessaires en moyenne pour accéder à une cellule, pour 100 millions d'accès, avec quelques tailles de listes différentes. Quelle différence note-t-on par rapport à la partie précédente ?

Q3 : Le script **run.py** permet de lancer 100 fois le programme **memwalk** avec chaînage aléatroire en faisant varier la taille des listes uniformément sur une échelle logarithmique entre 2^{10} et 2^{25}. Placer les 100 résultats dans un fichier **results.dat**.

> Utiliser la redirection de la sortie standard (`>`) de votre commande vers un fichier :

```
$ python run.py > results.dat
```

Q4 : Assurez-vous que **caches.cfg** contient les bonnes valeurs concernant la taille des différents niveaux de cache de votre machine.

Grâce à la commande **gnuplot** au script **mem_access_time.plt**, tracer la courbe des mesures prises précédemment :

```
$ gnuplot mem_access_time.plt > results.pdf
```

Interpréter la courbe.