TP2: Topologie Mémoire

Récupération d'information sur le système

Récupération d'info sur le sysfs `/sys/devices/system/cpu/cpu0/cache/index0` ou alors avec `lscpu`.

Mesure de la taille de la ligne de cache

Quel est l'intérêt d'initialiser la nouvelle zone mémoire

La mémoire est allouée de façon pareuseuse lors du **first touch**, donc à l'initialisation lorsque l'on écrit / lit dans une valeur. Cette allocation est faîte par page, il faut donc aller **touch** toutes les pages pour qu'elles soient vraiment alloué.

La boucle d´ecriture génère-t-elle des cache hit ? Des cache miss ? Comment varient le nombre de cache hit et de cache miss en fonction de PARAM

```
char *mem = allocate_and_touch_memory(MEMORY_SIZE);
for (i = 0; i < MEMORY_SIZE; i += PARAM)
  mem[i] = 1;
```

Cette boucle génère des caches miss lors du premier accès / invalidation d'une ligne de cache. Si la ligne est contenu dans le cache, on génèrera des cache hit. La proportion de cache hit / miss dépend des paramètres `MEMORY_SIZE` et `PARAM`.

Plus on augmente `PARAM`, plus on réduit les accès mémoires, et donc plus on augmente le nombre de miss. En fonction de la taille d'une ligne de cache, on a: $\frac{\text{PARAM}}{\text{LINE\_SIZE}}$ miss et $\frac{\text{LINE\_SIZE} - \text{PARAM}}{\text{LINE\_SIZE}}$ hit.

- `PARAM` très inférieur à la taille d'une ligne de cache:
  - 1 miss lors de la récupération de la ligne, puis ensuite que des hits pour le reste de la ligne.
  - Les hit prennent la majorité du temps d'exécution.
- `PARAM` légèrement inférieur à la taille d'une ligne de cache:
  - 1 miss lors de la récupération de la ligne, puis un hit si 2 accès overlap sur la même ligne.
  - Les miss prennent la majorité du temps d'exécution.
- `PARAM` légèrement supérieur à la taille d'une ligne de cache:
  - 1 miss lors de la récupération de la ligne, aucun hit.
  - Il n'y a plus de hit, donc que les miss prennent du temps d'exécution

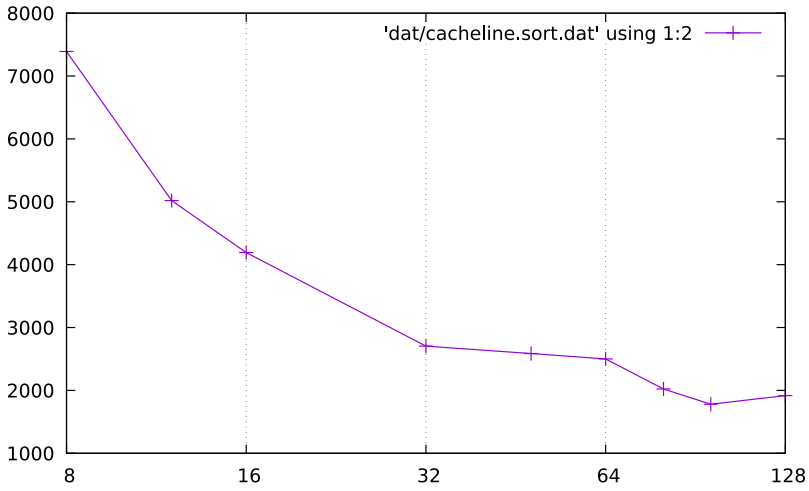
Pourquoi ajouter une boucle de warmup et répéter la boucle principale plusieurs fois

On rajoute une étape de warmup qui va apprendre au prefetcher le pattern d'accès mémoire en exécutant le code 10000 fois. Le but est de calibrer le matériel afin de minimiser son influence dans nos résultats.

De plus, on exécute la boucle principale plusieurs fois afin de réduire le bruit est avoir plus de précision sur nos valeurs.

Déduction de la taille d'une ligne de cache en fonction du graphique

On observe une stagnation vers `PARAM=32`, puis une baisse quand `PARAM=64`. On en déduit que la taille d'une ligne est de 64 B.



Mesure de la taille du caches L1

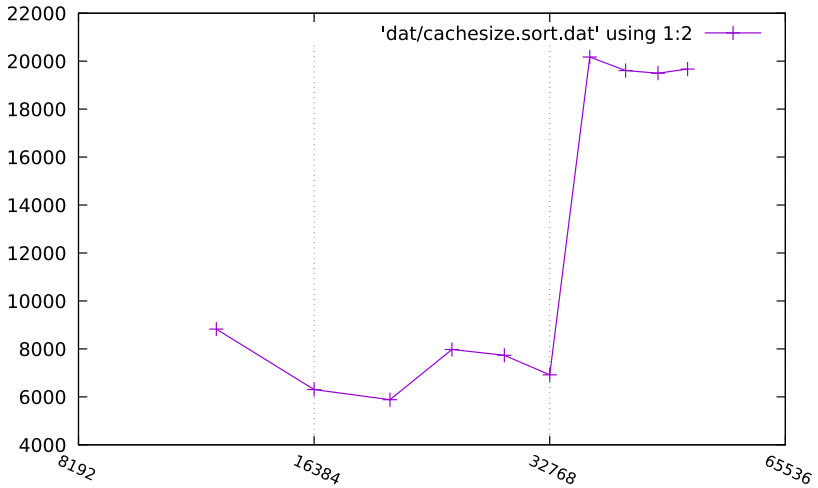
Influence de la taille d'une zone mémoire accédée par rapport au temps d'accès

`PARAM` représente la taille de la zone mémoire accédée.

- Si `PARAM` est plus petit que la taille du cache L1, alors on aura beaucoup de cache miss compulsifs lors de la première séquence d'accès mémoire puis que des hits lors des séquences suivantes.
- Si `PARAM` est plus grand que la taille du cache L1, alors on aura aussi beaucoup de miss compulsifs lors de la première séquence d'accès mémoire, puis dans les séquences suivantes, on aura des invalidations et donc plus de miss de conflits.

Donc avec un nombre de séquence d'accès très grand, le temps d'accès mémoire va dépendre du taux de miss après la première séquence. On en déduit donc que si `PARAM` est plus petit que la taille du cache L1, le temps d'accès sera faible. Puis il augmentera jusqu'à atteindre un plateau avec que des miss de conflits a mesure que l'on augmente la taille de `PARAM` après avoir dépassé la taille du cache.

Experimentation



D'après `lscpu` et `sysfs`, la machine a 4 MB de taille de cache L2. On ne peut par contre pas générer de graph car les grandes valeurs nous donnent erreur 139.

Mesure de l'associativité du cache L1

Valeur de SKIP_SIZE pour que toutes les lignes de cache modifiées appartiennent au même cache set?

`SKIP_SIZE` représente l'espace en bytes entre les séquences d'écritures. Pour que toutes les lignes de cache modifiées appartiennent au même cache set, il faut que `SKIP_SIZE` soit égal a $\text{NB\_SETS} * \text{LINE\_SIZE}$, avec $\text{NB\_SETS} = \frac{\text{CACHE\_SIZE}}{\text{NB\_WAYS}}$.

On le fait en fonction du nombre de set car dans un cache 2-associatif par exemple, un set peut contenir 2 lignes différentes dans un même emplacement. Donc si le cache fait 32K avec taille de ligne de 64B, il y a seulement 250 emplacements de lignes de cache au lieu de 500.

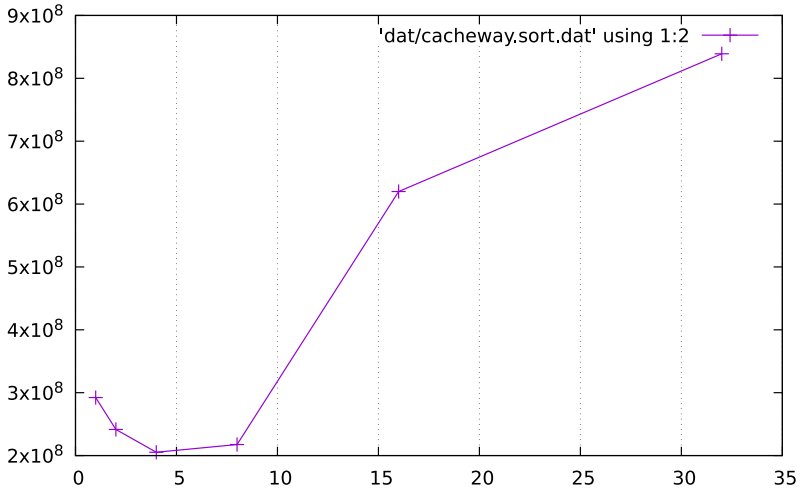
Soit X séquences de N écritures dans un même cache set, quelle est la valeur minimum de N pour générer seulement des caches miss

Si N est strictement supérieur au degrée d'associativité du cache, alors chaque écriture dans le même set au bout de N séquence génèrerons que des cache miss de conflit car la donnée sera évictée en boucle faute de place.

Donc la valeur minimum de N est $\text{NB\_WAYS} + 1$.

Expérimentation

On peut voir que le nombre d'associativité est 8.



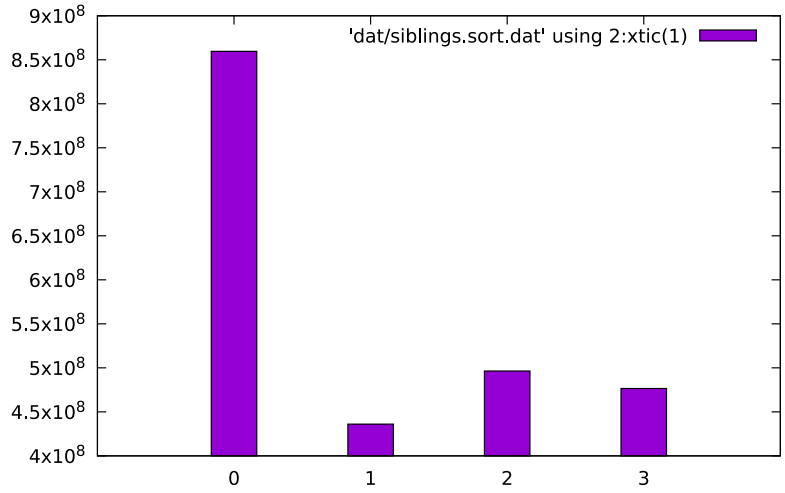
Mesure de l'hyperthreading

Pour tester si il y a de l'hyperthreading, on exécute deux programmes en même temps, un sur le coeur 0, et l'autre sur chaque coeur logique.

Chaque processeur remplit complétement le cache (remplit chaque set 8 fois), ils écrivent tout les 2 dans un espace mémoire différents (pour éviter de faire que des hits sur le 2eme programme). Si les 2 processeurs tournent sur le même coeur physique mais des coeurs logiques différents, alors les performances seront moins bonnes, car il y aura beaucoup de miss de conflit.

De plus, le warmup est effectué séparement avant le lancement des threads afin d'éviter des problèmes de synchronisation et d'obtenir des résultats plus consistent.

Experimentation



On peut voir sur le graph ci-dessus que lorsque les 2 programmes s'exécutent sur le ceur logique 0 en même temps, les performances sont nettement moins bonnes. Finalement, on peut observer que la machine n'a pas d'hyperthreading car les autres coeurs logiques n'ont pas de problème visible de performance, ce qui est cohérent avec `lscpu`.