TP3 : Simulation de cache de données et TLB

Compte rendu de TP à rendre au plus tard le 11 décembre par mail à `alain.merigot@u-psud.fr`, sous forme de fichier pdf TP3-nom-prenom.pdf)

1 Objectifs du TP

Ce TP a pour objectifs de d'étudier les performances d'un cache de données de premier niveau en fonction de ses caractéristiques. Le cache est simulé par un programme en C. Avant d'utiliser une variable `var` en lecture ou en écriture, on appelle une fonction qui simule l'accès au cache et qui prend en paramètre l'adresse mémoire de la variable obtenue par `&var`. On peut ainsi comptabiliser les succès ou les échecs en fonctions des paramètres du cache.

Nous considèrerons un cache avec les caractéristiques suivantes :

- Taille du cache : 4 ko, 8 ko, 16 ko, 32 ko ou 64 ko
- Taille des lignes : 16 octets, 32 octets ou 64 octets
- Associativité : 1 ligne/ensemble (correspondance directe), 2 lignes/ensemble ou 4 lignes/ensemble

Le cache est à réécriture. En cas de défaut en écriture, le bloc est chargé dans le cache avant l'écriture (écriture allouée).

Pour les caches associatifs par ensemble, la politique de remplacement est le LRU.

Les programmes fournis permettent d'obtenir le taux d'échec pour les différentes configurations du cache (taille de cache, taille de ligne et associativité).

La programme indique également le nombre de défauts de cache par type :

défauts obligatoires Ces défauts arrivent quand la ligne n'a *jamais* été chargée dans le cache. Pratiquement, on marque dans un tableau quand une ligne est lue. Une défaut sur une ligne non lue sera considéré comme un défaut obligatoire.

défauts de capacité Dès que toutes les lignes du cache sont occupées, tous les défauts sur une ligne qui a déjà été lue par le cache sont considérés comme des défauts de capacité.

défauts de conflit Un défaut de conflit arrive quand la ligne a déjà été lue par le cache et qu'elle a été enlevée du cache alors qu'il reste des emplacements libres. Avec une associativité totale, le défaut n'arriverait pas.

En pratique, tous les défauts non obligatoires sont considérés comme des défauts de capacité quand le cache est plein ou des défauts de conflit dans le cas contraire.

La taille des vecteurs ou des matrices est donnée par `#define N`. Pour utiliser une autre taille de vecteurs ou de matrices, il faut recompiler le programme en modifiant `N` dans le programme ou en ajoutant l'option `-DN=...` à la compilation.

Le répertoire comprend un certain nombre de programme avec des algorithmes d'algèbre linéaire programmés. A l'exécution, le programme parcourra les différentes configuration de caches et affichera les résultats (taux d'échec) correspondants. Un programme de nom prog-xxx.c génèrera à l'exécution un ficher res-xxx comprenant la description des caractéristiques des accès cache.

Ce programme sera étendu pour simuler et étudier le comportement d'un TLB.

2 Produit-scalaire

Le programme `prog-dotproduct.c` calcule le produit scalaire de deux vecteurs de taille `N`.

Pour les différentes configurations de cache données, quel est le taux d'échecs ? Essayer avec différentes valeurs de `N`.

Commentez et expliquez les résultats obtenus. Quel type de défaut a-t-on ? Pourquoi ?

3 Produit Matrice Vecteur

Le programme `prog-matvec.c` calcule le produit d'une matrice de taille `NxN` par un vecteur de taille `N`.

Pour les 27 configurations de cache données, quel est le taux d'échecs pour les valeurs suivantes de `N` : 64, 100, 512, 1024

Expliquez les résultats obtenus.

4 Produit de matrices

Lr programme `prog-matmult-ijk.c` calcule le le produit de deux matrices carrées de taille `NxN` $Z = X \times Y$ dans l'ordre ijk.

Pour toutes les configurations de cache données, donner et commenter le taux d'échecs pour les valeurs suivantes de `N` : 16, 32, 64, 128, 256.

Essayer également des valeurs proches, mais non puissance de 2 et expliquer la différence de comportement.

Même question en utilisant `prog-matmult-ikj.c` qui calcule le produit de deux matrices en utilisant l'ordre d'itération ikj.

Commentez les résultats et comparez au cas ijk.

5 Produit de matrices par blocs (optionnel)

On utilise l'algorithme de multiplication de matrices par blocs suivant :

```
1  for (jj=0;jj<N;jj+=B)
2    for (kk=0;kk<N;kk+=B)
3      for (i=0;i<N;i++)
4      {
5        for (j=jj;j<min(jj+B-1,N);j++)
6        {
7          s=0;
8          for (k=kk;k<min(kk+B-1,N);k++)
9          {
10           s = s + x[i][k]*y[k][j];
11         }
12         z[i][j]=s+z[i][j];
13       }
14     }
```

Avec un cache de 16 ko, le programme `prog-matmul-bloc.c` donne le taux d'échecs en fonction de la taille des lignes et de l'associativité pour différentes valeurs du facteur de blocage `B`.

Quels sont les taux d'échec pour N=64, 128 et 256 ?

Expliquer les résultats obtenus. Comment peut-on déterminer la taille optimale du facteur de blocage `B` ?

6 Simulation de TLB

En s'inspirant du programme de simulation de caches, il s'agit maintenant d'écrire un programme pour simuler le comportement d'un TLB.

Expliquez pourquoi un TLB de e entrées, avec une associativité a et gérant des pages de taille p, aura le même taux d'échec qu'un cache ayant la même associativité a, e/a ensembles de a lignes et des lignes de taille p.

Nous supposerons que les pages sont de 1ko. Nous considèrerons des TLB de 8 ou 16 entrées, et avec soit une correspondance directe, soit une associativité de 2 ou 4.

Le progamme `prog-matmult-tlb.c` contient deux produits de matrices (ijk et ikj).

En partant d'une version simplifiée du simulateur de caches (sans traitement des différents types de défauts), modifier le fichier `tlb.h` pour décrire le comportement d'un TLB à partir de l'adresse de la page accédée et déterminer les taux d'échec obtenus pour les différentes configurations du TLB. Les fonctions à écrire sont indiquées.

Déterminer en exécutant `prog-matmult-tlb` les taux d'échec du TLB pour ses différentes configurations dans les deux versions du produit de matrices. Conclusions ?

7 Préacquisition de données [optionnel]

En définissant `PREFETCH` dans le fichier `cache.h`, on simule une préaquisition simple : toute lecture d'une ligne est suivie par un chargement de la ligne suivante.

Regarder le comportement de ce cache avec *prefetch* pour le produit scalaire et le produit de matrice et expliquer les différences avec le cas sans préacquisition.