TP 4 – Caches directes et associatives

1 Étude du temps d'accès

Les cases d'un tableau à dimension 2 déclaré en langage C sont stockées en mémoire ligne à ligne.

1 – Pour vérifier cela, écrivez un programme C qui déclare un tableau de taille 2x3 et qui affiche les adresses (accompagnées des indices) de toutes les cases de ce tableau.

Quelque soit l'ordre de rangement des cases en mémoire, selon le type de parcours des cases (ligne par ligne ou colonne par colonne), les temps du parcours risquent de changer considérablement à cause de l'utilisation du cache.

2 – Réalisez deux programmes C déclarant chacun un tableau d'entiers de taille 1024x1024. Le premier programme initialise (avec les valeurs que vous voudrez) le tableau ligne par ligne, alors que le second programme l'initialise colonne par colonne. Pour confirmer les propos précédents, mesurez le temps de ces deux programme en utilisant la commande `/usr/bin/time NomduProgramme`.

2 Cache directe

Le but de la suite du TP est de simuler et de comparer le fonctionnement d'une mémoire cache directe et d'une mémoire cache associative à n voies. On supposera que la mémoire principale est adressée sur 32 bits.

Téléchargez l'archive `tp4src.tar.gz`. Elle contient des fichiers `.h` et `.c` que vous complèterez au long du TP, ainsi qu'un `makefile` permettant de tout compiler. Les fichiers `bits.h` et `bits.c` contiennent, notamment :

- la macro `#define SMOT(A,N,T) ((A & (((0x1«T)-1)«N))»N)` donnant la valeur entière du sous-mot de `T` bits commençant au bit `N` du mot `A`
- la fonction `void afficheBits(int n)` affichant les bits de `n`, de la gauche (bit de poids le plus fort) vers la droite (bit de poids le plus faible).

3 – Quelles sont les deux caractéristiques d'une mémoire cache directe ?

On veux simuler une mémoire cache dont les lignes sont des mots de 32 bits.

4 – Pour un entier `taille` représentant la puissance de 2 égale au nombre de ligne, détaillez la décomposition en

- tag
- index
- offset

d'une adresse mémoire.

Les fichiers `directCache.h` et `directCache.c` implémentent une cache directe.

La structure `struct Ligne` implémente une ligne du cache et stocke, dans des entiers pour simplifier, le tag, un bit de validité et un mot mémoire de 32 bits. La structure `struct DCache` implémente une mémoire cache directe. Son champ `taille` représente la puissance de 2 égale au nombre de lignes de la mémoire, et son champ `table` est un tableau de 2 puissance `taille` éléments `struct Ligne`. La fonction `void DCache_Disp(struct DCache *c)`, déjà implémentée, permet d'afficher une cache directe.

5 – Complétez les fonctions :

- `struct DCache * DCache_Create (int)`,
- `void DCache_Delete (struct DCache)`,

permettant respectivement d'initialiser une mémoire cache directe (l'entier passé en paramètre est la puissance de 2 du nombre de ligne souhaité), et de la libérer. À la création, tous les tags seront à 0 ainsi que tous les indicateurs de validité. Testez vos fonctions dans `testDirectCache.c`.

6 – Complétez les fonctions :

- `int getOffset(int adresse, struct DCache *c)`,
- `int getIndex(int adresse, struct DCache *c)`,
- `int getTag(int adresse, struct DCache *c)`,

réalisant la décomposition d'une adresse en tag, index, offset, et testez-les dans `testDirectCache.c`.

La fonction `int setDCache(int adresse, int mot, struct DCache *c)` permet de charger dans la cache `c` le mot `mot` qui se trouve à l'adresse `adresse` ; elle permet de simuler la recopie d'une partie de la mémoire dans la cache. La fonction `int lwDC (int *registre, int adresse, struct DCache *c)` est l'équivalent de l'instruction `lw` en MIPS. Sont but est de remplir le contenu du registre `registre` par le contenu de la mémoire à l'adresse `adresse`. Le mot est cherché prioritairement en cache. S'il s'y trouve (hit), `registre` est renseigné

et la fonction retourne 1. Sinon (miss), le mot est cherché dans la mémoire, et la cache est renseignée avec ce mot (ce qui peut être simulé avec la fonction `setDCache`, avec n'importe quelle valeur pour `mot`). La fonction renvoie alors 0.

7 – Implémentez les fonctions `setDCache` et `lwDC`. Comme le but est ici de caractériser les performances, on passera n'importe quelles valeur pour les mots mémoire, et `registre` pourra être `NULL`, dans quel cas on n'y touchera pas. Testez vos fonctions dans `testDirectCache.c`.

On souhaite simuler l'utilisation d'une mémoire cache. Les paramètres sont :

— le nombre de lignes de la cache,

— la plage d'adresses mémoire disponible,

— le nombre de requêtes en écriture.

Le fichier `DirectCacheSimulation.c` contient la fonction `int rand_Adresse(int inf, int supp)` renvoyant une adresse aléatoire se trouvant dans l'intervalle d'adresses [`inf`, `supp`].

8 – Complétez la fonction `int main(int argc, char **argv)` pour créer un programme simlant l'utilisation de la cache. Les paramètres de la simulation seront passés en arguments du programme `DirectCacheSimulation`. Les nombres de misses et de hits seront affichés à l'écran.

3 Cache associative à n voies

On souhaite à présent simuler une mémoire cache associative à n voies, puis la comparer avec une mémoire cache directe. Dans `associativeCache.h`, une telle mémoire cache est implémentée par la structure `struct ACache` dont le champ `nbTable` contient le nombre de voies (mémoires caches directes), et le champ `DTable` est un tableau de pointeurs vers des `struct DCache`.

9 – Dans `associativeCache.c`, complétez les fonctions :

— `struct ACache * ACache_Create (int taille, int nbTable)` créant une cache associative à `nbTable` voies, chaque cache directe ayant 2 puissance `taille` lignes,

— `void ACache_Delete (struct ACache)` permettant de supprimer un élément `struct ACache`,

et testez-les dans `testDirectCache.c`.

10 – Complétez les fonctions :

— `int setACache(int adresse, int mot, struct ACache *c)`,

— `int lwAC(int *registre, int adresse, struct ACache *c)`.

Pour la fonction `setACache` : si les lignes correspondant à l'index de l'adresse `adresse` de toutes les voies sont renseignées avec un mot valide, et dont tous les tags sont différents du tag de `adresse`, il faut appliquer une politique de remplacement du cache. Quelle politique simple pouvez-vous mettre en oeuvre ? La valeur de retour de `setACache` sera l'indice de la cache directe dans laquelle le mot a été écrit. La fonction `lwAC` aura le même comportement que `lwDC`.

11 – Enfin, dans le fichier `AssociativeCacheSimulation.c`, écrivez la fonction `int main(int argc, char **argv)` réalisant un programme prenant en argument à la ligne de commande les paramètres d'une simulation d'un cache associatif à n voies, effectuera cette simulation. Les nombres de misses et de hits seront affichés à l'écran.