Chapitre 3 : Cohérence des caches

3.1 Un multi-processeur à mémoire partagée adressable par mot (de 4 octets) vient avec une mémoire cache associée à chacun des 4 processeurs. Cette mémoire cache est à correspondance directe et contient 4 blocs de deux mots. Montrez le statut de chaque bloc (MESI) après les accès suivants (processeur, P0 à P3, et accès, R ou W suivi de l'adresse et valeur) si les caches sont initialement vides. Discutez des communications dans les cas par mise à jour ou invalidation, et par répertoire ou surveillance de bus.

P0: R 120; P0: W 120, 80; P3: W 120, 80; P1: R 110; P0: W 108, 48; P0: W 130, 78; P3: W 130, 78

Cache directe, 4 blocs de 2 mots, MESI
Adresse 32 bits: étiquette 29 / bloc 2 / mot 1
Adresses décomposées: 120: 15/0/0, 110: 13/3/0, 108: 13/2/0, 130: 16/1/0

Les blocs suivants sont touchés par les accès dans chaque cache. Lorsqu'un bloc présent dans une autre cache (même étiquette) est accédé, il devient S. Si le bloc écrit se retrouve dans une autre cache, il est invalidé dans l'autre cache et modifié localement.

P0: R 120, bloc 0 E; P0: W 120, 80, bloc 0 M; P3: W 120, 80, bloc 0 M, P0 bloc 0 I; P1: R 110, bloc 3 E; P0: W 108, 48, bloc 2 M; P0: W 130, 78, bloc 1 M; P3: W 130, 78, bloc 1 M, P0 bloc 1 I.

3.2 Un programme réinitialise un vecteur de 128 entiers de 4 octets à 0. Si la mémoire cache associée au processeur contient des blocs de 64 octets, décrivez les communications nécessaires sur le bus pour une cohérence par surveillance et mise à jour versus par répertoire et invalidation.

Un programme réinitialise un vecteur de 128 entiers de 4 octets à 0, cache avec blocs de 64 octets. Il y a donc 16 éléments entiers du vecteur dans chaque ligne de cache. Avec surveillance et mise à jour, un message est envoyé sur le bus pour chacun des 128 accès. Avec invalidation, une invalidation est envoyée à chaque 16 accès au répertoire; si aucune autre cache ne contient le bloc, cette invalidation n'est pas envoyée aux autres processeurs.

3.3 Deux variables mises à jour par des processeurs différents se retrouvent dans la même ligne de cache. Ces variables servent à calculer une somme dans une boucle et sont donc accédées une fois en lecture et une fois en écriture à chaque fois (4 instructions à 1 cycle/instruction hormis les fautes de cache de 16 cycles de pénalité). Comment peut-on se rendre compte d'un tel problème? Que peut-on faire pour y remédier? Quel facteur d'amélioration pourrait en résulter?

Accès une fois en lecture et une fois en écriture à chaque fois (4 instructions à 1 cycle/instruction hormis les fautes de cache de 16 cycles de pénalité). Le

scénario probable est une faute de cache par tour de boucle pour 4+16=20 cycles versus 4 cycles. Un tel problème peut être détecté par des outils spécifiques ou avec un bon outil de profilage des fautes de cache. Il suffit d'aligner chaque variable sur une nouvelle ligne de cache pour y remédier.

3.4 Le processeur 0 exécute la fonction foo alors que le processeur 1 exécute la fonction bar sur un ordinateur multi-processeur à mémoire partagée avec cache, queues d'écriture et queues d'invalidation et le protocole de cohérence MOESI. Les variables a et b sont globales et initialement à 0. Est-ce qu'il pourrait arriver que l'assertion a==1 ne soit pas vérifiée? Expliquez?

```
void foo(void)
{ a = 1;
  smp_wmb();
  b = 1;
}

void bar(void)
{ while (b == 0) continue;
  assert(a == 1);
}
```

La barrière mémoire en écriture assure que b=1 arrivera après a=1 dans la copie maîtresse de ce bloc de mémoire (en mémoire centrale ou en cache avec le statut modified). Il faut toutefois s'assurer que les mises à jour (invalidations) arrivent dans l'ordre sur le processeur 1, avec une barrière mémoire de lecture, smp_rmb(), entre les deux énoncés de la fonction bar.