# 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

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.

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?

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);
}
```