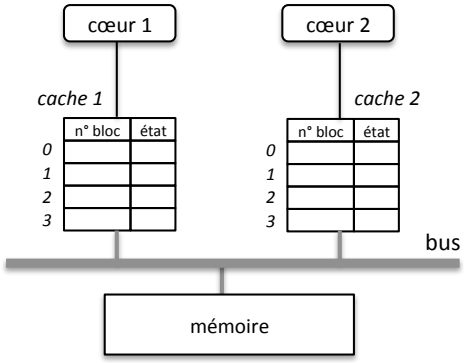
Sujets de TD n° 4

Exercice 1.

On considère un processeur multicoeur à mémoire partagée comportant 2 coeurs, chacun muni d'un cache de 4 lignes, à correspondance directe et à écriture différée.
La mémoire partagée contient 8 blocs (numérotés de 0 à 7).
La cohérence des données dans les caches est maintenue par un protocole de type invalidation à 3 états :



- ***invalide*** : la ligne de cache ne contient pas de bloc, ou contient un bloc qui a été invalidé
- ***partagé*** : la ligne de cache contient un bloc qui peut être lu seulement, et dont les autres caches sont susceptibles d'avoir une copie également
- ***modifié*** : la ligne de cache contient un bloc qui peut être lu et écrit, et dont aucun autre cache ne possède une copie

Question 1.

Construire un diagramme de transitions entre états pour une ligne de cache. Faire apparaître en trait plein les transitions générées par un accès du cœur local (lecture ou écriture dans le cache, PrRd ou PrWr), et en trait pointillé les transitions générées par une requête à la mémoire réalisée par un autre cœur et observée sur le bus commun (accès mémoire en vue d'une lecture ou d'une écriture, BusRd ou BusRdX). Etiqueter les transitions en indiquant le résultat de l'accès au cache (pour les transitions liées à une action du cœur local, read/write hit/miss) et les requêtes générées sur le bus (BusRd, BusRdX, Flush, entre crochets).

Question 2.

Remplir le tableau ci-joint en représentant, pour chacune des opérations mentionnées, l'évolution de l'état des lignes dans les deux caches, et en précisant, dans la colonne "commentaires" ce qui se passe dans les deux caches et sur le bus.

Exercice 2.

On considère un processeur multicoeur à mémoire partagée et à bus commun. Chaque processeur dispose d'un cache et la cohérence des données est gérée par un protocole espion de bus à 4 états, de type MESI :

- ***invalide*** : la ligne de cache ne contient pas de bloc, ou contient un bloc qui a été invalidé
- ***exclusif*** : la ligne de cache contient un bloc qui peut être lu seulement et dont il est garanti qu'il n'est contenu dans aucun autre cache. Une ligne ne peut être mise dans cet état que lors de son chargement, si aucun autre cache ne signale qu'il en a une copie, ce que l'on représentera par le signal SH=0.
- ***partagé*** : la ligne de cache contient un bloc qui peut être lu seulement, et dont les autres caches sont susceptibles d'avoir une copie également
- ***modifié*** : la ligne de cache contient un bloc qui peut être lu et écrit, et dont aucun autre cache ne possède une copie

Question 1.

Construire le diagramme de transitions du protocole MESI.

Question 2.

On veut étudier l'évolution de l'état de 3 blocs a, b et c dans chacun des caches, au cours de l'exécution d'une série de références mémoire. On suppose qu'au départ les blocs ne sont présents dans aucun des caches. Compléter le tableau ci-joint.

Exercice 3.

On considère un multiprocesseur (16 processeurs) à bus commun dans lequel la cohérence des données dans les caches (64 KO, associatifs de degré 4, avec des blocs de 64 octets) est gérée par un protocole espion de bus MESI.
Lors de l'exécution d'une application (Ocean) sur ce multiprocesseur, on a observé les différentes transitions possibles entre état et on a compté le nombre de ces transitions. Les résultats moyens pour 1000 références mémoire sont présentés dans le tableau ci-dessous. L'état *invalid* a été décomposé en deux états (I = *invalid* et NP = *not present*) pour distinguer un bloc non présent parce que jamais chargé dans le cache ou remplacé par un autre bloc, d'un bloc invalidé suite à une opération nécessitant le maintien de la cohérence des données. Notons par ailleurs que la somme des compteurs est supérieure à 1000 : ceci est dû au fait qu'une référence mémoire peut donner lieu à plusieurs transitions. Par exemple, un défaut en écriture peut générer deux transitions dans le cache local (par exemple, S→NP pour le bloc remplacé, et NP→M pour le bloc chargé) et des transitions dans d'autres caches qui invalident leur copie.
Par ailleurs, on a observé 99,7 millions de références mémoire pour 376,5 millions d'instructions exécutées.

De / Vers	NP	I	E	S	M
NP	0	0	26,25	2,6	15,14
I	1,33	0	0	0,3	0,0008
E	21,18	0,3	452,6	0,45	4,32
S	2,46	1,33	0	113,25	1,11
M	19,02	0,001	0	1,55	387,78

Question 1.

Dans quelles circonstances chacune de ces transitions peut-elle être observée, et par quelle transaction sur le bus est-elle déclenchée ?

Question 2.

On suppose que toute transaction sur le bus implique le transfert de 6 octets d'adresse et de commande. Quel est le trafic sur le bus (en nombre d'octets) pour 1000 références mémoire ?

Question 3.

Si un processeur exécute 200 millions d'instructions par seconde, quel est le trafic généré par processeur ? Quelle est alors la bande passante bus nécessaire pour 16 processeurs ?

Exercice 4.

On considère le programme OpenMP suivant, qui simule l'évolution des températures sur une tige métallique dont une extrémité est forcée à 0°C et l'autre à 100°C :

```
#define N 514
float temp[N] = {100.0, 0.0, 0.0, 0.0, …, 0.0};
/* calcul de d'évolution des températures */
#pragma omp parallel num_threads(8)
int t, s ;
for (t = 0 ; t < TMAX ; t++){
    #pragma omp for
    for (s = 1 ; s < N-1 ; s++)
        temp[s] = (temp[s-1] + temp[s+1]) / 2.0;
}
```

Ce programme est exécuté sur une plateforme à 8 cœurs munis de caches, à bus commun et à mémoire partagée. La cohérence des données dans les caches est assurée par un protocole espion de bus de type MESI.
En mémoire, une température est codée sur 4 octets. Un bloc mémoire a une taille de 64 octets, et l'élément `temp[1]` est rangé à l'adresse 64.

Question 1.

Quel est le nombre de blocs mémoire utilisés par un thread ? Indiquer combien de blocs sont seulement lus par le processus, et combien sont modifiés.
Application numérique : quels sont les blocs lus et modifiés par le thread 1 (qui calcule les segments 65 à 128 inclus) ?

Question 2.

On s'intéresse maintenant à ce qui se passe dans les caches et sur le bus commun au cours du calcul des températures. On examine le cas de trois calculs suivants :

(A) calcul de `temp[65]` à l'itération `t=0` (en supposant que le thread 0 n'a pas encore calculé `temp[47]`)
(B) calcul de `temp[66]` à l'itération `t=0` (en supposant que le thread 0 n'a pas encore calculé `temp[47]`)
(C) calcul de `temp[65]` à l'itération `t=1` (en supposant que le processus 0 a déjà calculé `temp[49]`)

Pour chacun de ces calculs, décrire très précisément l'évolution des blocs dans les caches et les transactions observées sur le bus commun (tableau en annexe).

Exercice 5.

On considère un quadri-cœur à mémoire partagée et à bus commun, au sein duquel la cohérence des données dans les caches est assurée par un protocole espion de bus de type MESI.

Le programme ci-dessous calcule la distribution des niveaux de gris d'une image (représentée par un vecteur de 65536 pixels, `img`, un pixel étant codé sur 1 octet, ce qui permet 256 niveaux de gris). Pour éviter des sections critiques, le calcul se fait en deux temps :

- d'abord, chaque thread calcule un histogramme partiel, `hp[#thread][256]`, sur une portion de l'image qui lui est allouée (16384 pixels consécutifs dans le vecteur `img`). Il n'y a pas d'accès concurrent dans cette partie.
- ensuite, chaque thread calcule une partie de l'histogramme complet (64 éléments) à partir des histogrammes partiels (le sien et celui des autres threads). Il n'y a pas non plus d'accès concurrent dans cette phase.

```
char img [65536] ;
int hp[4][256] ;
      // histogrammes partiels – chaque élément est codé sur 4 octets
int hc[256] ;
    // histogrammes complets – chaque élément est codé sur 4 octets

#pragma omp parallel num_threads(4)
{
    int i,pix ;
    int me = omp_get_thread_num() ;
    #pragma omp for schedule(static, 16384)
    for (pix=0 ; pix<65536; pix++)
        hp[me][img[pix]] ++;
    #pragma omp for schedule(static, 64)
    for (i=0 ; i<256 ; i++)
        hc[i]=hp[0][i]+hp[1][i]+hp[2][i]+hp[3][i];
}
```

On suppose que les variables partagées sont rangées en mémoire en séquence, à partir de l'adresse 0.
Le tableau `hp[4][256]` est rangé ligne par ligne :

`hp[0][0] – hp[0][1]- … - hp[0][255] – hp[1][0] - … - hp[1][255] - …`

Les blocs ont une taille de 32 octets. La taille du cache est supposée supérieure à celle de l'ensemble des variables.

Question 1.

On s'intéresse à des accès mémoire réalisés par le thread 0 :

- dans la première boucle (ces accès dépendent des valeurs des pixels traités). Par exemple, on suppose que le thread 0 traite un premier pixel qui vaut 0 (mise à jour de `hp[0][0]`), puis un second pixel qui vaut 12 (mise à jour de `hp[0][12]`), et plus tard un 33ème pixel qui vaut 6 (mise à jour de `hp[0][6]`).
- dans la seconde boucle quand, par exemple, il calcule l'histogramme complet de 0 à partir des histogrammes partiels de l'ensemble des threads (`hc[0] = hp[0][0] + hp[1][0] + hp[2][0] + hp[3][0]`)

Compléter le tableau sur la feuille ci-jointe.

Question 2.

Calculer le nombre de transactions sur le bus (pour les 4 threads) lors de l'exécution de ce programme. On suppose que chaque thread rencontre au moins un pixel de chaque niveau de gris.
Si chaque transaction nécessite le transfert d'une adresse sur 4 octets en plus du bloc de données (on ignore les bits de contrôle), quel est le trafic généré (en octets) ? Quel aurait été le trafic généré avec un protocole de cohérence MSI ?