Master SAR - ASTRE Année 2021–2022
E. Encrenaz

Projet - Modélisation et Vérification de systèmes concurrents

Architecture multiprocesseur à mémoire partagée

On considère un système multiprocesseur à mémoire partagée, organisé autour d'un bus. Le système est muni d'une hiérarchie mémoire à deux niveaux : mémoire centrale, partagée et caches privés associés à chaque processeur. La mémoire stocke les instructions, données, pile des programmes en cours d'exécution et du système d'exploitation. Le cache privé comprend deux parties distinctes, le cache d'instructions et le cache de données. On ne modélisera que les transferts induits par les mouvements de données (et non pas les flux instructions). Les processeurs+caches et la mémoire sont connectés autour d'un bus, dont l'accès est régi par un arbitre de bus. Les différents éléments et leurs connexions sont présentés sur la figure 1.

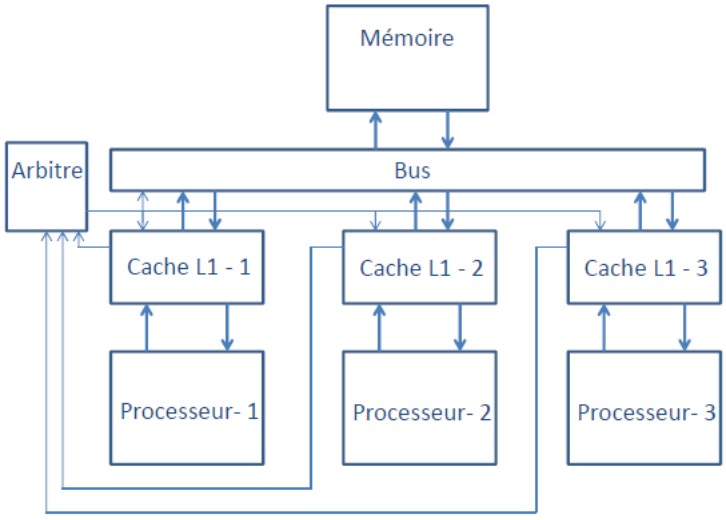

FIGURE 1 – Architecture de la plate-forme multiprocesseur.

1 Composants constitutifs

1.1 Processeur

Le processeur exécute des instructions du programme en cours ; il émet des requêtes de lecture et écriture de données vers son cache privé L1.

— `CPU_RD(ad)` : demande le mot d'adresse `ad` qui lui sera transmis (au mieux au même cycle) par le cache L1. Tant que la donnée n'est pas transmise par le cache L1, le processeur est gelé : il n'effectue aucune action.

— `CPU_WR(ad,val)` : requête d'écriture de la valeur `val` à l'adresse `ad` ; la requête est transmise au cache L1 ; tant que la requête n'a pas été validée par le cache L1, le processeur est gelé. (on ignore les mécanismes d'écriture non bloquante)

1.2 Mémoire

La mémoire est composée de n mots de m bits, adressable par mot. Dans un premier temps, on suppose que $n = 2$ et $m = 1$. De plus, on considérera que les mots d'adresses 0 et 1 correspondent à des données cachables. La mémoire est interfacée sur le bus et agit sur sollicitation de requêtes d'un des caches.

— `READ(ad)` : demande de lecture du mot d'adresse `ad`, transitant sur le bus, issue d'un cache L1. La mémoire produit le contenu du mot d'adresse `ad` sur le bus (au cycle suivant pour simplifier).

— `WRITE(ad,val)` : demande d'écriture de la valeur `val` à l'adresse `ad`, transitant sur le bus, issue d'un cache L1. La mémoire modifie le contenu du mot d'adresse `ad` en y affectant la valeur `val` et produit un signal d'acquiescement sur le bus au cycle suivant.

1.3 Bus

Le bus interconnecte la mémoire et les caches L1 des processeurs. Il est composé de trois nappes de fils :

— AD : adresse du mot à transmettre

— DT : valeur du mot transmis (en lecture ou en écriture)

— CTRL : sens du transfert (lecture ou écriture)

De plus, le champ VALID indique que les données présentes sur le bus sont significatives.

Le bus est une ressource partagée et un point de synchronisation du système.

— Il est partagé par les différents caches L1 qui initient des transferts de et vers la mémoire, et par la mémoire qui répond aux sollicitations des caches. L'architecture du bus considérée est *multiplexée* : le bus est composé de 4 nappes de fils AD/DT/CTRL (une par cache L1 et mémoire pouvant écrire sur le bus, nommées `b_in`), aiguillées sur une unique nappe de fils AD/DT/CTRL (nommée `b_out`) qui peut-être lue par tous les composants connectés au bus. L'aiguillage est réalisé par un multiplexeur 4×1, commandé par le signal généré par l'arbitre de bus (cf. Figure 2).

— C'est un point de synchronisation : à tout instant, les données transitant sur le bus `b_out` peuvent être lues simultanément par tous les caches L1 et la mémoire. Tous les composants connectés voient les requêtes transitant sur le bus dans le même ordre : la séquence de requêtes `WRITE(0,0);...; WRITE(0,1)` sera interprétée par tous de la même façon (écriture de la valeur 0 à l'adresse 0, puis écriture de la valeur 1 à l'adresse 0), ils peuvent tous réagir au même instant en connaissant les mêmes informations.

1.4 Arbitre de bus

L'arbitre définit, lorsqu'il y a plusieurs requêtes simultanées vers la mémoire, celle qui a la priorité d'accès sur le bus. Une requête qui accède au bus conserve l'exclusivité d'accès jusqu'à ce que cette requête ait été acquittée par la mémoire. Lorsqu'il sélectionne la requête i, l'arbitre positionne le signal `arb_gnt` à `i` pour un cycle, puis donne l'accès du bus à la mémoire dès le cycle suivant, jusqu'à ce que la mémoire valide la requête i. Un autre arbitrage peut avoir lieu ensuite.

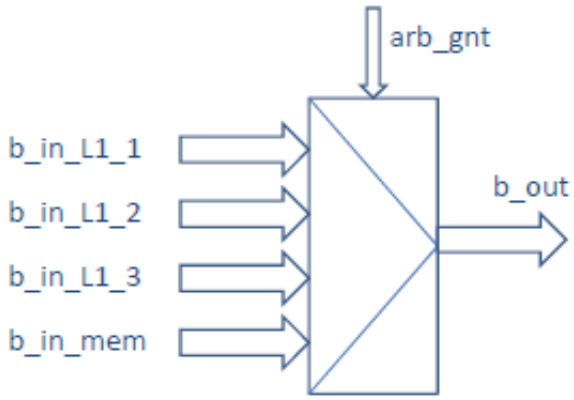

FIGURE 2 – Architecture du bus multiplexé

Le chronogramme de la figure 3 montre l'incidence de l'arbitrage sur les données transitant sur le bus. Sur la figure, une requête i survient (signal `L1_i_req` positionné et maintenu à 1) ; plus tard, (lorsque la requête j en cours sera terminée), la requête i sera choisie par l'arbitre (le signal `arb_gnt` a la valeur `i` pour un cycle) ; la transaction passant sur le bus `b_out` est celle positionnée par le cache *L1_ i*, sur son interface `b_in_L1_i` (les données associées à cette requête doivent être positionnées et valides) ; au cycle suivant, l'arbitre donne l'accès à la mémoire, qui répond sur le bus dès que les données associées à la requête i sont disponibles (en positionnant la donnée et le signal VALID) sur son interface `b_in_mem`. Le cache *L1_ i* ayant reçu la réponse à sa requête, il désactive le signal de requête `L1_i_req` vers l'arbitre ; ce dernier peut alors donner l'accès au bus à une autre requête s'il y en a au moins une ; sinon, il laisse l'accès à la mémoire (qui a désactivé son signal VALID...).

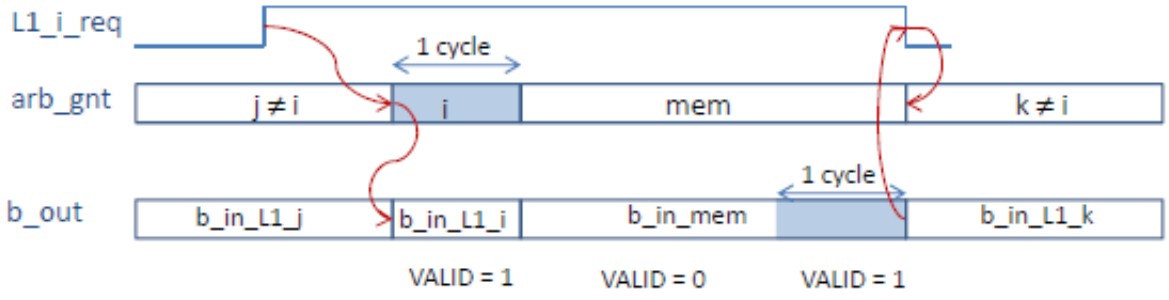

FIGURE 3 – Chronogramme de transfert sur le bus multiplexé

1.5 Cache L1

Le cache L1 stocke le mot préalablement demandé en lecture par le processeur auquel il est associé, jusqu'à ce que ce mot soit mis à jour ou évincé. On suppose que le cache ne peut contenir qu'un mot à la fois. Lorsqu'un mot en remplace un autre, le mot évincé est simplement écrasé.

1.5.1 Interface CPU/L1 :

- Sur réception d'une requête `CPU_RD(ad)`, le cache détermine s'il dispose du mot d'adresse `ad`. S'il dispose de la donnée valide, il la transmet au CPU dans le même cycle (sur les signaux de réponse prévus entre le processeur et son cache L1). Sinon, il gèle le processeur et demande l'accès au bus à l'arbitre puis transmet une requête de lecture vers la mémoire (`READ(ad)`) et recevra la donnée par le bus (sur `b_out`, lorsque `arb_gnt = mem` et `VALID=1`) ; le cache stocke alors le mot reçu (association adresse/valeur du mot), et transmet la valeur au processeur.

- Sur réception d'une requête `CPU_WR(ad,val)`, le cache détermine s'il dispose du mot d'adresse `ad`. S'il en dispose, il le met à jour, acquitte le processeur et transmet la requête vers la mémoire ; sinon, il transmet la requête vers la mémoire et dégelera le processeur à réception de l'acquittement de la mémoire (il ne rappatrie pas le mot à modifier).

1.5.2 Interface L1/arbitre/bus :

- Lorsque le cache L1 doit transmettre une requête vers la mémoire, il demande l'accès au bus à l'arbitre. La demande du cache *L1_ i* active le signal `req_L1_i` jusqu'à ce que l'arbitre donne l'accès au bus (signal `arb_gnt=i`). Lorsque l'accès est donné, le cache écrit ses données sur le bus dans le même cycle (les données ont été préchargées sur l'interface `b_in_L1_i`, et le signal `arb_gnt` positionné à *i* aiguille ce flot vers le bus `b_out`).

- Lorsque le cache L1 a transmis une requête sur le bus, il attend la réponse provenant du bus (les transactions sur le bus sont insécables) à l'exclusion de toute autre action.

- (mécanisme de SNOOP) Lorsqu'une requête d'écriture circule sur le bus, chaque cache connecté doit déterminer s'il dispose du mot concerné. Dans l'affirmative, il met à jour sa copie locale avec la donnée à écrire, véhiculée sur le bus.

2 Travail à réaliser

2.1 Etude du protocole et des accès aux données partagées

Cas monoprocesseur

Illustrez les transferts d'informations induits par l'exécution des programmes P1a puis P1b sur le processeur 1 (les processeurs 2 et 3 sont inactifs) ; le cache est vidé entre l'exécution des deux programmes. Dans ces programmes, les mnémoniques désignent les actions décrites dans la table 1. Les symboles R1, R2, R3 désignent des registres du processeur, les symboles 0 et 1 désignent les constantes de mêmes valeurs, le symboles [0] et [1] désignent les adresses 0 et 1.

`ld`	load	`dec`	decrement by 1
`st`	store	`beq`	branch if equal
`add`	addition	`cmp`	comparaison

Effet cache

```
P1a :
      ld  R1, [0]
      add   R1, R1, 1
      st   R1, [1]
      ld   R2, [0]
      add  R2, R2,1
      st   R2, [0]
```

Eviction du cache / écriture write-trough

```
P1b :
     ld  R1, [0]
     ld  R2, [1]
     add R3, R1, R2
     st  R3, [0]
```

Cas multiprocesseur

Illustrez les flux d'informations induits par l'exécution des programmes P1 sur le processeur 1, P2 sur le processeur 2 et P3 sur le processeur 3. Vous considérerez différents ordonnancements possibles d'accès au bus.

Partage en lecture / une écriture

```
P1a :                 | P2a :               | P3a :
     ld      R1, [0]  |      ld   R1, [0]   |      <inactif>
     add  R1, R1, 1   |      ....           |
     st      R1, [0]  |      ....           |
     ....             |      add R2, R1, 1  |
                      |      st R2, [1]     |
```

Boucle d'attente active / partage en lecture - écriture

```
P1b :              | P2b :             | P3b :
dbt:ld    R1, [0]  | dbt: ld R1, [0]   |      ....
     cmp  R1,0     |      cmp R1,0     |
     beq  dbt      |      beq  dbt     |
     dec R1        |      dec  R1      |      ld R1, [1]
     st  R1, [0]   |      st  R1, [0]  |
     <sc>          |       <sc>        |
```

Quel est l'intérêt du mécanisme de SNOOP dans cette architecture ? Quel mécanisme doit encore être ajouté pour garantir un accès exclusif aux données partagées ? (ce dernier point pourra être l'objet de vos propositions en fin de projet).

2.2 Modélisation des composants

On s'attachera à décrire le comportement des différents éléments constitutifs de la plate-forme avec l'outil de modélisation et vérification NuSMV. On adoptera une démarche progressive en complexifiant petit à petit la plate-forme et les composants associés.

On privilégiera une description booléenne, intégrant quelques constantes entières (minimiser les calculs sur entiers). On utilisera à profit les tableaux ou les types énumérés pour décrire des collections de données (nappes de fils du bus, signaux d'interface des composants). Vous pourrez consulter les exemples *gigamax.smv* et *msi.smv*, disponibles dans la collection d'exemples proposés avec la distribution NuSMV, qui décrivent des mécanismes de maintien de la cohérence de caches dans des architectures multiprocesseurs (mais attention, le fonctionnement des caches décrits dans ces exemples est différent et bien plus complexe que celui présenté ici).

Étape 1 :

Définir les signaux d'interface des éléments de la plate-forme : bus, arbitre, mémoire, cache L1, processeur. Pour chaque composant, vous identifierez ses interactions avec les autres composants, ainsi que la nature des donnés à transférer (dans le sens des requêtes et dans le sens des réponses) ; le cas échéant, vous préciserez le codage que vous utilisez. Vous proposerez un schéma récapitulatif de l'architecture focalisé sur les interfaces de communication.

Étape 2 :

Proposez et simulez une implantation des différents noeuds. Dans cette étape, on supposera
- que la mémoire dispose de deux adresses uniquement : 0 et 1.
- que le cache L1 ne fait que transmettre les requêtes lui provenant vers la mémoire et récupère sa donnée propre (pas de mémorisation du mot en transit, pas de SNOOP des requêtes sur le bus) ; il joue uniquement un rôle d'accès au bus/récupération de la donnée.
- que l'arbitre multiplexe les écritures sur le bus d'un unique cache L1 et de la mémoire.

Il est suggéré de modéliser dans l'ordre : la mémoire, le processeur, l'arbitre, le bus, le cache L1.

Étape 3 :

Décrivez les propriétés pertinentes pour chaque élément (considéré isolément) et vérifiez-les à l'aide de NuSMV.

2.3 Réalisation d'une plate-forme monoprocesseur

Les éléments décrits précédemment sont suffisants pour réaliser une plate-forme monoprocesseur (sans effet de cache). Assemblez-les et vérifiez le bon fonctionnement du système par simulation. En supposant que la mémoire répond toujours en un cycle, proposez une propriété de sûreté montrant le bon transfert de données de la mémoire vers le processeur en cas de lecture et du processeur vers la mémoire en cas d'écriture.

Il est alors possible de complexifier le comportement du cache L1 pour introduire l'effet de mémorisation (sans le SNOOP). Proposez un nouveau modèle de cache L1 et vérifiez-le

(simulation + propriétés de sûreté). Vous pourrez vous inspirer des scenarii proposés dans l'étude du protocole.

2.4 Réalisation d'une plate-forme à trois processeurs

Modifiez l'arbitre pour qu'il puisse gérer l'accès au bus pour 3 caches L1 + la mémoire. Modifiez le comportement du cache L1 pour intégrer le mécanisme de SNOOP. Proposez un mécanisme permettant de manipuler des verrous à base de mots mémoire qui sont non cachables : les lectures et écritures sur ce verrou sont directement effectuées en mémoire, qui dispose de la seule copie valide de ce verrou, et un seul point d'accès.

Etendez l'architecture (mémoire, nappes de fils, instructions d'accès directes en mémoire) pour introduire une donnée non cachable (d'adresse 2).

Montrez le comportement conforme de la plate-forme en proposant différents scénarii d'accès aux données partagées d'adresses 0 et 1 protégés par le verrou d'adresse 2 (vous pourrez étendre les scenarii préalablement décrits pour introduire ce verrou, ou en proposer d'autres).