Mémoires Transactionnelles : étude du protocole de cohérence de cache et de la politique de résolution des conflits

Quentin Meunier

Laboratoire d'Informatique de Paris 6
Équipe Alsoc
4 Place Jussieu, 75252 Paris, France

15 Novembre 2012

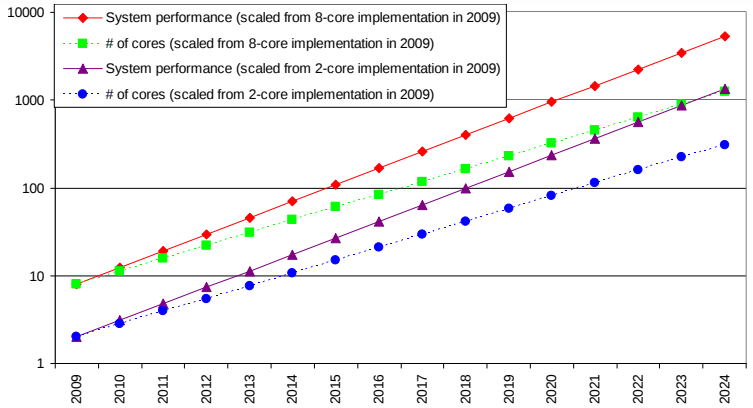
Introduction

- Relation Puissance - Fréquence pour une machine ?
- Fin de l'évolution exponentielle des performances séquentielles d'un processeur en 2004
- Nouvelle dimension : parallélisme au sein d'une puce via l'intégration de plusieurs coeurs

Introduction

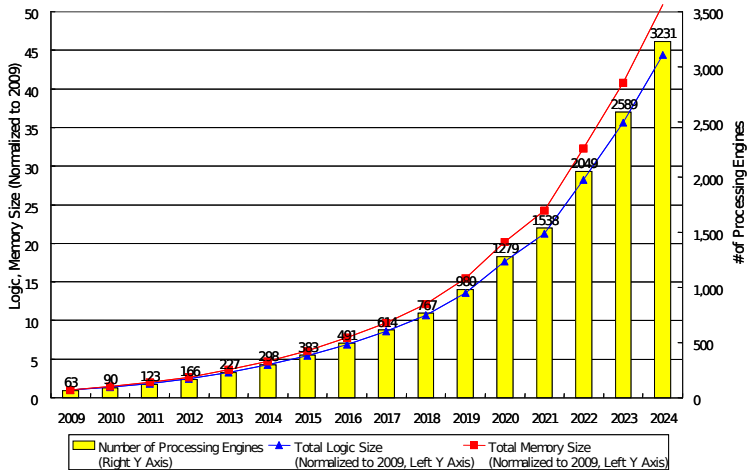
Tendances ITRS

- Augmentation importante du nombre de coeurs par puce
- Plus de 1000 coeurs dans les SoCs en 2020



Caractéristiques des puces multicoeurs

- Plusieurs processeurs sur une puce
- Communication à travers de la mémoire partagée



Architectures multicoeurs

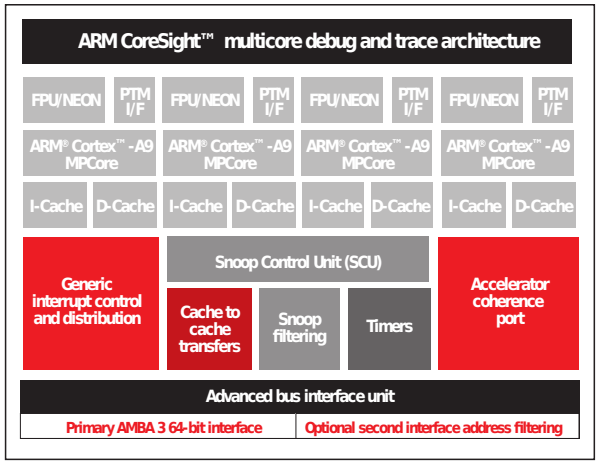
Défis des architectures multicoeurs

- Posent de nombreuses nouvelles problématiques (ex : support OS)
- Adaptation des applications
- Adéquation du modèle de programmation au matériel
- Nécessité d'écrire des programmes et des applications parallèles

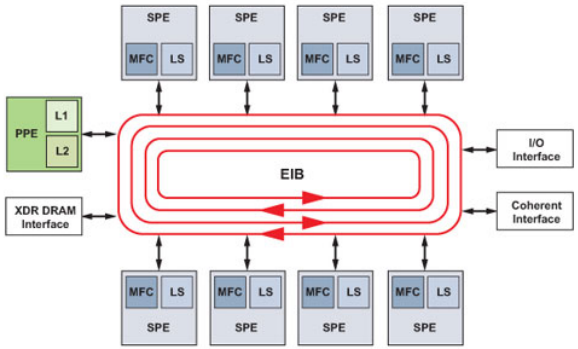
Programmation parallèle

- Écrire un programme parallèle *est* difficile
- Problématique : exploiter le parallélisme disponible (en général, compromis efficacité/complexité)
- Restriction à des modèles connus (larges charges de travail parallèles, pipeline, parallélisme de données), mais qui ne répondent pas à tous les besoins
- Nécessité d'exploiter le parallélisme entre tâches à grain fin

Exemples :



ARM Cortex A9



IBM Cell

Outils et abstractions pour l'écriture des programmes parallèles

Solutions actuelles

- Thread : fil d'exécution séquentiel
- Synchronisation des exécutions
  - Abstractions niveau système d'exploitation : sémaphores, moniteurs, barrières
  - Abstractions plus bas niveau : verrous
- Section critique : portion de code exécutée à un instant donné par au plus un processeur

Limites

- Sémaphores, moniteurs et barrières ne permettent pas toujours un parallélisme fin
- Programmes à base de verrous difficiles à concevoir, implémenter, déboguer et maintenir
- Souvent, mauvaises performances lors du passage à l'échelle
- Non composable (principe de composition) : exemple plus tard

Besoin d'autres constructions de programmation

- Programmes lock-free
- Transactions, *via* les systèmes de mémoire transactionnelle (TM)

Plan

1. Introduction
2. Problématique et cadre général du travail
3. Comparaison de 2 systèmes HTM basés sur des PCC différents
4. Étude de différentes politiques de résolution des conflits
5. Conclusion et perspectives

Plan

1 Introduction

2 **Problématique et cadre général du travail**

3 Comparaison de 2 systèmes HTM basés sur des PCC différents

4 Étude de différentes politiques de résolution des conflits

5 Conclusion et perspectives

Problèmes & Contributions

Problèmes

- Adéquation entre matériel et programmes parallèles ?
- Quelles politiques pour la résolution des conflits dans les systèmes TM ?

Contributions

- Comparaison de deux systèmes TM matériels basés sur des protocoles de cohérence de cache différents
- Étude de politiques de résolution des conflits, comparaison de performances et des garanties

Alternatives aux verrous pour la synchronisation bas-niveau

Lock-free

- Basé sur le Compare-And-Swap (CAS) ou les instructions LL/SC
- Avantages
  - Pas d'interblocage
  - Support matériel faible
- Inconvénients
  - Complexité des algorithmes
  - Performances dans le cas général

Transactions

- Ensemble d'instructions de taille arbitraire qui s'exécute de manière atomique

Sémantique de l'instruction CAS

```
bool CAS(T *address, T old_val, T new_val){
     <atomic>
     if (*address == old_val){
          *address = new_val;
          return true;
     }
     else {
          return false;
     }
     <end atomic>
}
```

Alternatives aux verrous pour la synchronisation bas-niveau : exemple

Incrémenter une variable partagée

Avec des verrous

```
pthread_spin_lock(&var_lock);
var++;
pthread_spin_unlock(&var_lock);
```

Avec un CAS

```
do {
   loc_var1 = var;
   loc_var2 = loc_var1 + 1;
} while (!cas(&var,loc_var1,loc_var2));
```

Avec des transactions

(idéalement)

```
begin_transaction();
var++;
end_transaction();
```

- Et avec deux variables ?

Avec des verrous

- Utiliser un verrou pour les 2 variables: limitation possible du parallélisme

- Utiliser 2 verrous : garantir l'ordre

Avec un CAS

- Utiliser un DCAS ou un CAS double mot (avec une structure)

- Utiliser un CAS de pointeur (problème ABA)

Avec des transactions

```
begin_transaction();
var1++;
var2++;
end_transaction();
```

- Et avec n variables ?

Principe de composition : exemple

Programme avec une Hashtable concurrente

- Existence de méthodes `Insert` et `Remove` *Thread-safe* (exécutables en concurrence)

Principe de composition : exemple

Programme avec une Hashtable concurrente

- Existence de méthodes `Insert` et `Remove` *Thread-safe* (exécutables en concurrence)
- Nouvelle opération `Move`, qui déplace un élément d'une table à une autre
  - Contrainte : l'état intermédiaire ne doit pas être visible (consistence)
  - Pas de moyen de créer `Move` en composant les codes de `Insert` et `Remove`
  - Obligation de changer d'abstraction en exposant des détails d'implémentation (ex : méthode `LockTable`)
  - Risques d'interblocages

Principe de composition : exemple

Programme avec une Hashtable concurrente

- Existence de méthodes `Insert` et `Remove` *Thread-safe* (exécutables en concurrence)
- Nouvelle opération `Move`, qui déplace un élément d'une table à une autre
  - Contrainte : l'état intermédiaire ne doit pas être visible (consistence)
  - Pas de moyen de créer `Move` en composant les codes de `Insert` et `Remove`
  - Obligation de changer d'abstraction en exposant des détails d'implémentation (ex : méthode `LockTable`)
  - Risques d'interblocages

Avec des transactions

- Code de `Move` :

```
begin_transaction();
    Elem e = hashTable1.remove(...)
    hashTable2.insert(e)
end_transaction();
```

Principe de composition : exemple

Programme avec une Hashtable concurrente

- Existence de méthodes `Insert` et `Remove` *Thread-safe* (exécutables en concurrence)
- Nouvelle opération `Move`, qui déplace un élément d'une table à une autre
  - Contrainte : l'état intermédiaire ne doit pas être visible (consistence)
  - Pas de moyen de créer `Move` en composant les codes de `Insert` et `Remove`
  - Obligation de changer d'abstraction en exposant des détails d'implémentation (ex : méthode `LockTable`)
  - Risques d'interblocages

Avec des transactions

- Code de `Move` :

```
begin_transaction();
    Elem e = hashTable1.remove(...)
    hashTable2.insert(e)
end_transaction();
```

- Aller plus loin : code de `insert` :

```
begin_transaction();
    hashTable.insertSequential(e);
end_transaction();
```

Le modèle transactionnel

Caractéristiques

- Modèle conceptuellement simple car abstraction haut-niveau : la complexité est en-dessous
- Respecte le principe de composition
- Provient des bases de données : propriétés ACID
  - *Atomicité d'échec* ou *Atomicité* : toutes les modifications de la transaction ou aucune
  - *Consistence* : dépendant du contexte (exemple : 2 tables contiennent le même nombre d'éntrées non nulles)
  - *Isolation* : la transaction doit avoir l'impression de s'exécuter toute seule
  - *Durabilité* : les effets de la transaction doivent être durables dans le temps, même lorsque le programme se termine (ex : écriture disque)
- Pour l'exécution d'un programme, les deux propriétés essentielles sont Atomicité et Isolation
- Pas de notion de durabilité : les données sont uniquement en mémoire – d'où le nom de *Mémoire Transactionnelle*
- La consistence est (devrait être) garantie si Atomicité et Isolation sont vérifiées

Types de systèmes TM

Systèmes TM logiciels (STM)

- Pas de matériel spécifique requis : en général implémenté avec un CAS
- Surcoûts temporels importants
- API complexe

Systèmes TM matériels (HTM)

- Support matériel spécifique aux transactions
- Meilleures performances que les STM
- API simple (vue précédemment)

Exemple de programme : insertion dans une liste chaînée

```
typedef struct {int key; struct node* next;} node;
typedef struct {node *head;} list;

void insert_in_list(list *l, int k){
    node *n := new node(k);



        node *prev := l→head;
        node *curr := prev→next;
        while (curr→key < k){
            prev := curr;
            curr := curr→next;
        }
        n→next := curr;
        prev→next := n;

}
```

Types de systèmes TM

Systèmes TM logiciels (STM)

- Pas de matériel spécifique requis : en général implémenté avec un CAS
- Surcoûts temporels importants
- API complexe

Systèmes TM matériels (HTM)

- Support matériel spécifique aux transactions
- Meilleures performances que les STM
- API simple (vue précédemment)

Exemple de programme : insertion dans une liste chaînée

```
typedef struct {int key; struct node* next;} node;
typedef struct {node *head;} list;

void insert_in_list(list *l, int k){
     node *n := new node(k);
     wstm_transaction *tx;
     do {
          tx := WSTMStartTransaction();
          node *prev :=  WSTMRead(tx, &(l→head));
          node *curr :=  WSTMRead(tx, &(prev→next));
          while (curr→key < k){
               prev := curr;
               curr :=  WSTMRead(tx, &(curr→next));
          }
          n→next := curr;
          WSTMWrite(tx, &(prev→next),n);
     } while (not(WSTMCommitTransaction(tx)));
}
```

Types de systèmes TM

Systèmes TM logiciels (STM)

- Pas de matériel spécifique requis : en général implémenté avec un CAS
- Surcoûts temporels importants
- API complexe

Systèmes TM matériels (HTM)

- Support matériel spécifique aux transactions
- Meilleures performances que les STM
- API simple (vue précédemment)

Exemple de programme : insertion dans une liste chaînée

```
typedef struct {int key; struct node* next;} node;
typedef struct {node *head;} list;

void insert_in_list(list *l, int k){
    node *n := new node(k);


    begin_transaction();
        node *prev := l→head;
        node *curr := prev→next;
        while (curr→key < k){
            prev := curr;
            curr := curr→next;
        }
        n→next := curr;
        prev→next := n;
    end_transaction();
}
```

Avantages et limitations des systèmes HTM

Avantages

- Modèle de programmation simple
- Évite les problèmes d'interblocages
- Tire partie des ressources disponibles :
  - Si plusieurs transactions concurrentes sur des données différentes : exploite le parallélisme
  - Si plusieurs transactions concurrentes en conflit : résolu par le système TM
- Passage à l'échelle

Limitations

- Les transactions ne peuvent pas remplacer tous les mécanismes de synchronisation de manière efficace (ex : barrière)
- Coûts matériels élevés
- Fortement lié au matériel et au système d'exploitation
- Actions irréversibles (E/S) : pas de vraie solution
- Mauvaise interaction avec les verrous (ex : bibliothèques existantes)

Classification des systèmes HTM

Transaction

- Ensemble d'instructions atomiques
- Propriétés d'atomicité et d'isolation

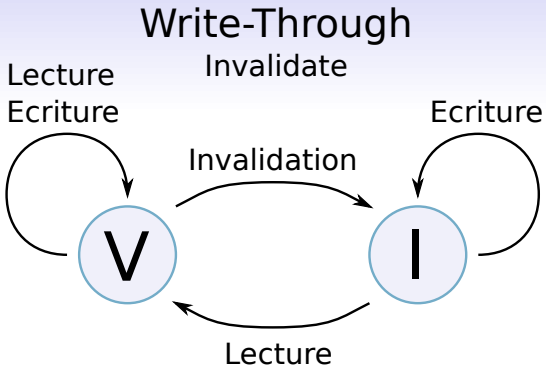
Quatre principaux critères

- Détection des conflits (CD) : Quand détecter que deux transactions sont en conflit ?
  - *Eager* ou *Lazy*
- Gestion de versions (VM) : Où mettre les anciennes et les nouvelles valeurs ?
  - Nécessité de stocker à la fois les anciennes et les nouvelles valeurs (aborts)
  - *Eager* ou *Lazy*
- Résolution des conflits : Que faire lorsqu'un conflit est détecté ?
  - De nombreuses possibilités : mettre en attente, recommencer la requête/transaction, etc.
- Gestion des débordements
  - Beaucoup de systèmes HTM utilisent les caches pour les données spéculées $\Rightarrow$ Problèmes de débordement
- $\Rightarrow$ Critères dépendants du protocole de cohérence de cache

Protocoles de Cohérence de Cache (PCC)

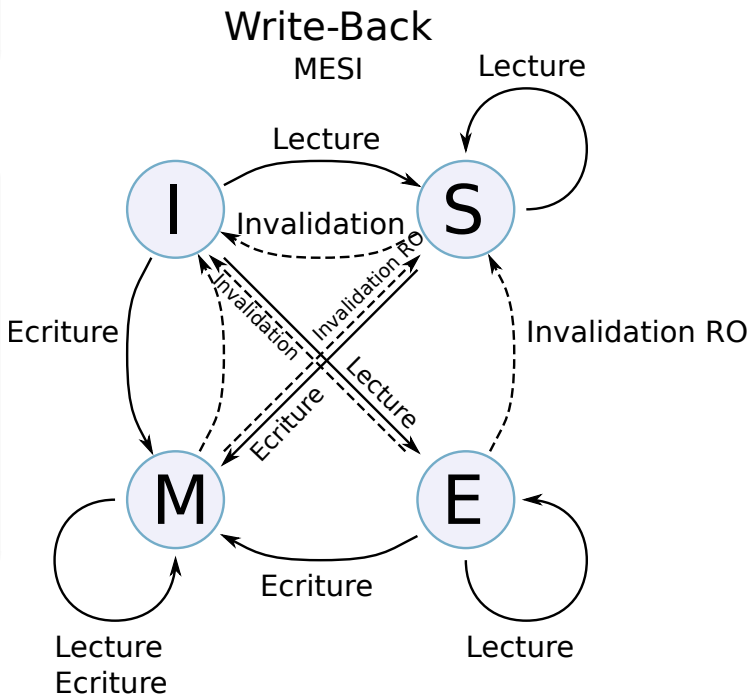
Protocole de type write-through (WT)

- Peu de logique nécessaire, protocole peu complexe
- Propagation de toutes les écritures en mémoire
- Généralement coûteux en consommation
- Performances acceptables sur un réseau-sur-puce



Protocole de type write-back (WB)

- Relativement complexe, en particulier sur un NoC
- 2 bits par ligne pour encoder l'état (MSI, MESI)
- Moins coûteux en terme de consommation : peu d'écritures propagées
- Bonnes performances sur un réseau-sur-puce



Contexte de l'étude

Systèmes multiprocesseurs

- Importance du protocole de cohérence de cache
- Contrainte : pas de *snooping* possible (utilisation d'un NoC)

Systèmes HTM

- Dans les systèmes existants, hypothèse d'un PCC de type write-back
- Conception de deux systèmes HTM : un basé sur un PCC de type write-through et un sur un PCC de type write-back
- Comparaison de leurs performances

Politiques de résolution des conflits

- Paramètre de conception important dans un système HTM
- Conception, implémentation et comparaison de plusieurs politiques
- Approche du point de vue de la garantie d'absence de *livelock*

Plan

1 Introduction

2 Problématique et cadre général du travail

3 **Comparaison de 2 systèmes HTM basés sur des PCC différents**

4 Étude de différentes politiques de résolution des conflits

5 Conclusion et perspectives

Choix de conception & Restrictions

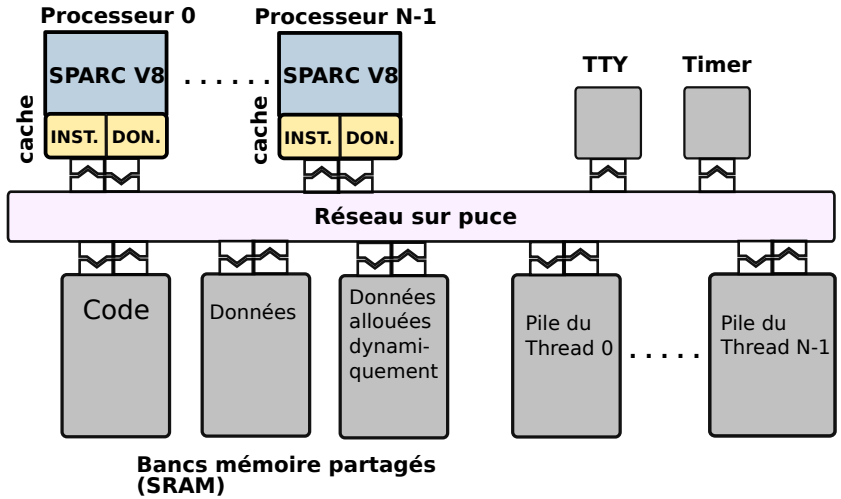
Choix de conception architecturaux

- Cohérence mémoire
- Utilisation de réseaux-sur-puce (NoCs) : cohérence plus complexe
- Processeur avec registres libres (ex : SPARC)
- Cache données et instructions séparés
- Deux configurations pour le nombre de bancs mémoire : unique ou multiples

Restrictions

- Adresses physiques, pas de MMU
- Caches L1 seulement, à correspondance directe

Architecture typique :



Configuration typique :

Nombre de processeurs	$p = 16$ ou 32
Nombre de bancs mémoire	$p + 3$
Modèle du processeur	SPARC-V8 avec FPU
Taille du cache (D)	$16Ko$
Taille des lignes du cache (D)	8 mots (32 octets)
Taille du cache (I)	$16Ko$
Taille des lignes du cache (I)	8 mots
Associativité du cache	Correspondance directe
Taille du tampon d'écritures	8 mots
Topologie du NoC	Maillage 2D

Choix de conception & Restrictions

Caractéristiques des systèmes HTM

- En write-back
  - Approche “optimiste” avec CD eager/VM eager
  - Abort de la transaction qui n’écrit pas (priorité aux écritures), ou si W/W de la transaction ayant fait la requête
- En write-through
  - Approche “pessimiste” (cf. exemples)
- Sémantique d’entrelacement à plat

Autres caractéristiques et restrictions

- Pas d’opérations d’E/S ni d’appels systèmes au sein d’une transaction (autre que malloc/sbrk : ré-écriture)
- Un thread par processeur, chaque thread fixé à un processeur – pas de *context-switch*
- Isolation forte : résistance aux accès mixtes
- Granularité = ligne de cache (faux conflits possibles)
- Pas d’acquisition de lock dans une transaction

Interface du système TM

Primitives

- Implémentées à l'aide de macros assembleur ou dans le ldscript
- `begin_transaction()`
  - Sauvegarde des registres du processeur si pas déjà dans une transaction
  - Le processeur "transforme" toutes les requêtes suivantes en requêtes transactionnelles équivalentes
- `end_transaction()`
  - termine la transaction et effectue le commit
  - pas d'effet si transaction interne
- `abort_transaction()`
  - annule la transaction en cours
  - utile ? (utilisé par certains benchmarks)
- `store_log_address(uint *)`
  - adresse du log en mémoire
- `store_log_size(uint)`
  - taille du log pour vérification de débordement

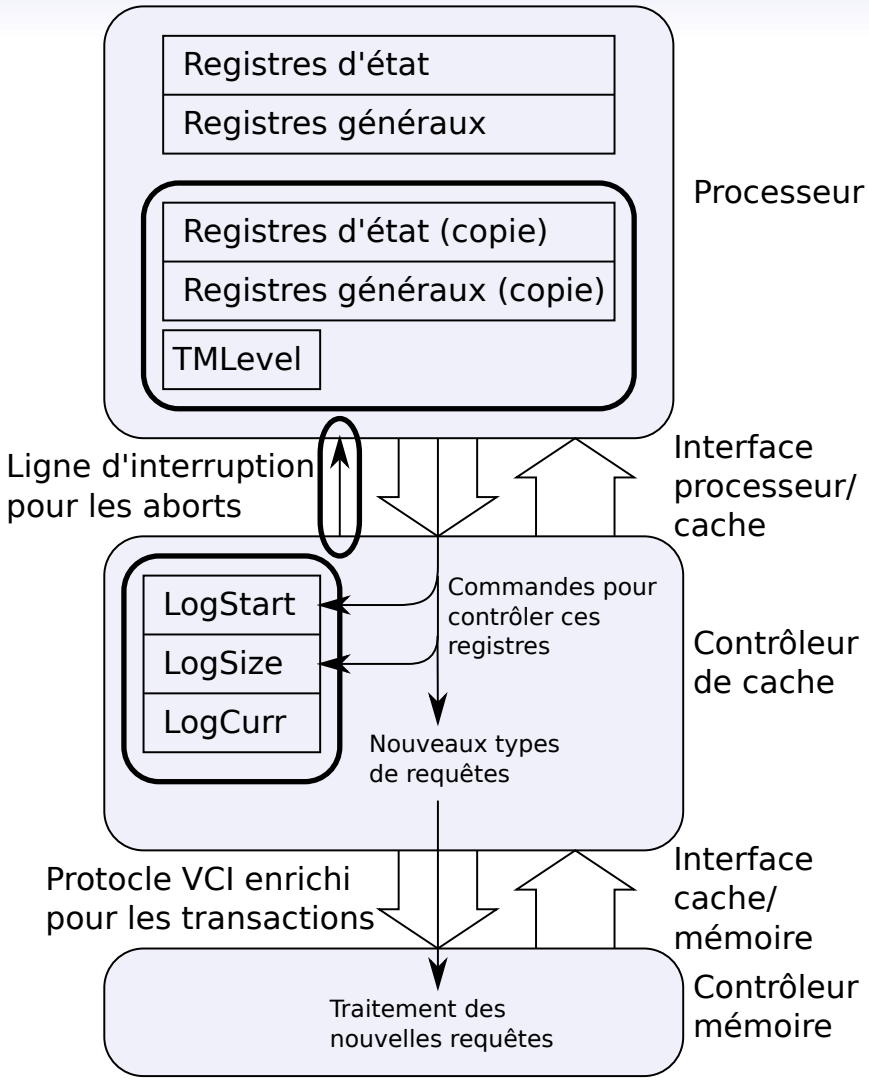
Modifications architecturales communes aux deux systèmes

Sur les processeurs

- Ajout d'un banc de registres supplémentaire dans le processeur
- Besoin de décoder de nouvelles instructions
- Ligne d'interruption pour les aborts

Sur les caches et la mémoire

- Enrichissement du protocole cache/processeur et du protocole VCI
- Ajout de registres spécifiques aux transactions
- Modification des différentes machines d'état afin d'implémenter les protocoles étendus avec des transactions



Autres modifications architecturales

Contrôleur de cache

- Ajout de 2 (WT) ou 3 (WB) bits/ligne pour l'état transactionnel de la ligne (R/W ou R/W/Ov)

Contrôleur mémoire (WT)

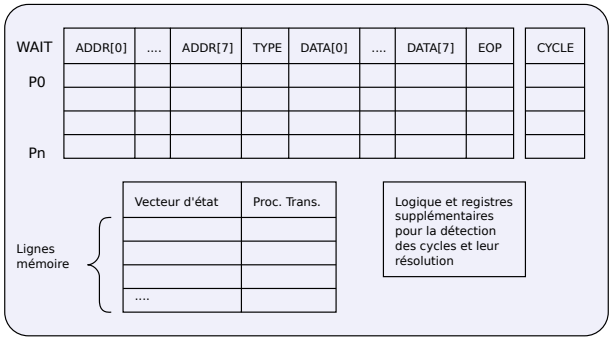
- Ajout de $\lceil \log_2(p+1) \rceil$ bits/ligne (p processeurs) pour les données transactionnelles
- Ajout pour chaque processeur d'une liste de requêtes en attente
- Logique et registres pour la détection et la correction des cycles

Contrôleur mémoire (WB)

- Peu de modifications, support des nouveaux types de requêtes

Etat (1)	Etat Trans. (2)		TAG (ex : 18 avec 512 lignes de 8 mots)
Valide	R	W	

Contrôleur de cache (WT)



Répertoire et contrôleur mémoire (WT)

Etat (2)	Etat Trans. (3)			TAG (ex : 18 avec 512 lignes de 8 mots)
M,E,S,I	R	W	Ov.	

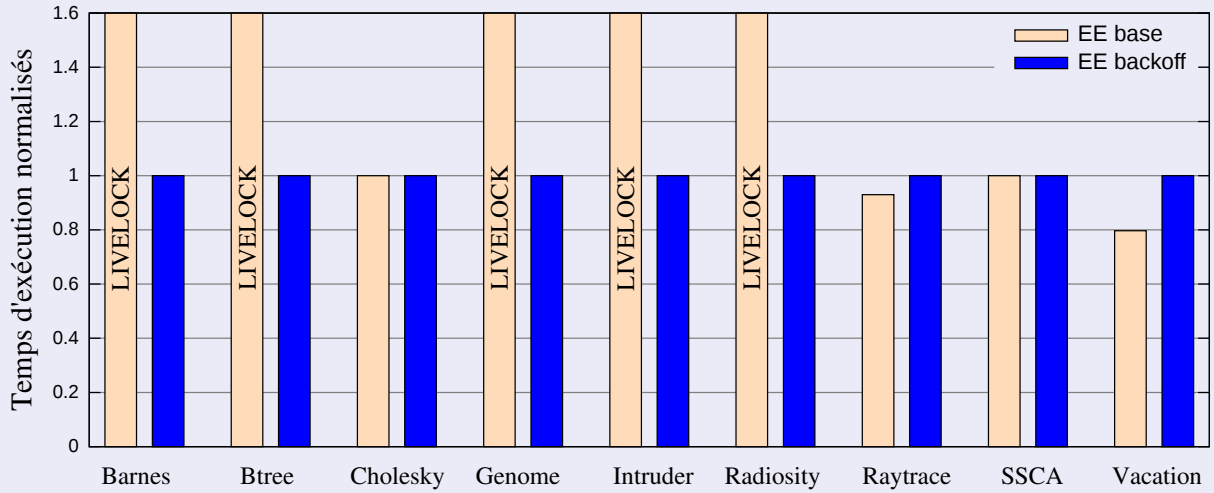
Contrôleur de cache (WB)

Impact du backoff sur le système Write-Back

Propriétés

- Temps d'attente après un abort
- Temps pseudo aléatoire, graine propre à chaque processeur
- Doublement du temps max après chaque abort, réinitialisation après un commit
- Permet d'éviter les cas pathologiques en pratique

Résultats sur les benchmarks SPLASH-2 et STAMP



- Ajouter un backoff est indispensable pour éviter les cas pathologiques

Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

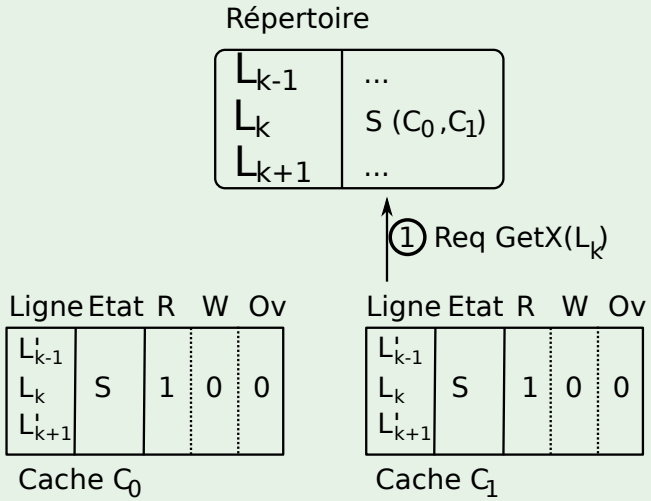
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 1



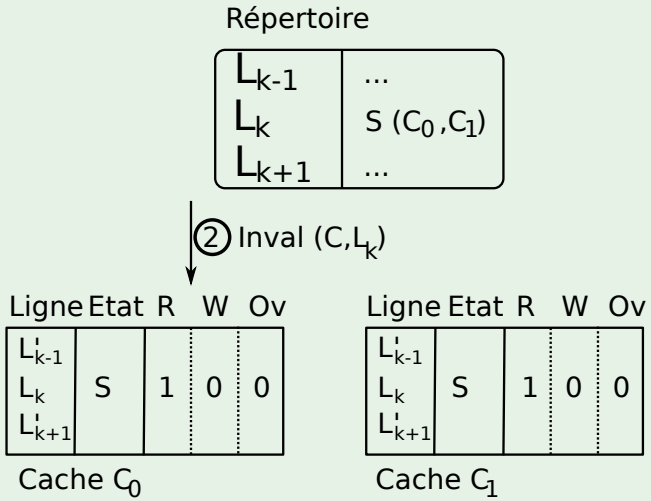
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 1



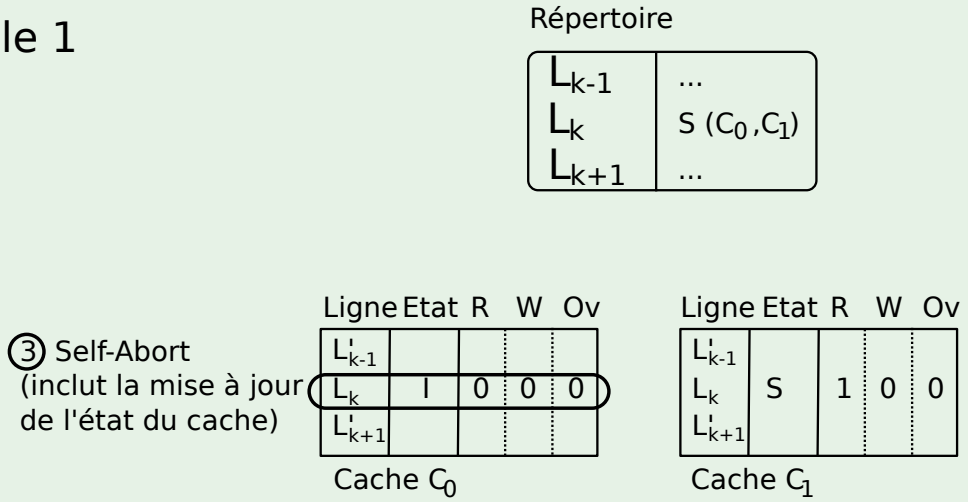
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 1



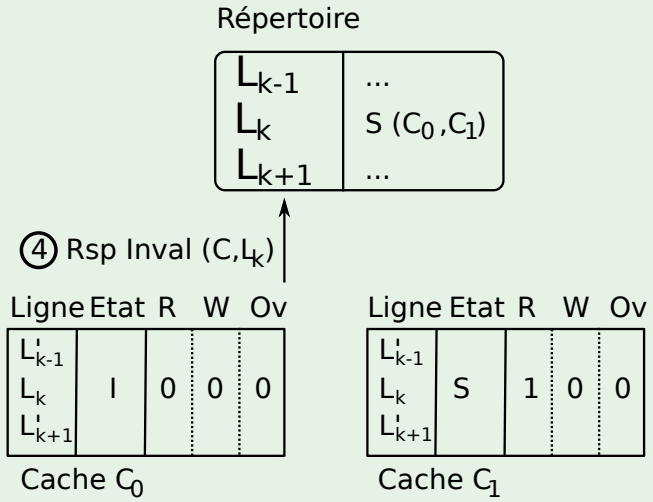
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 1



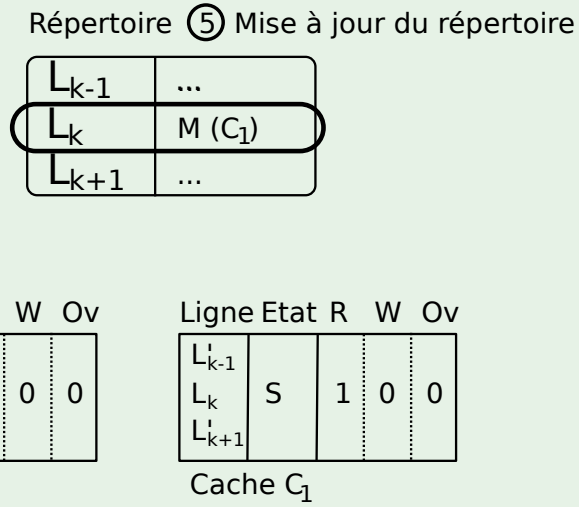
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 1



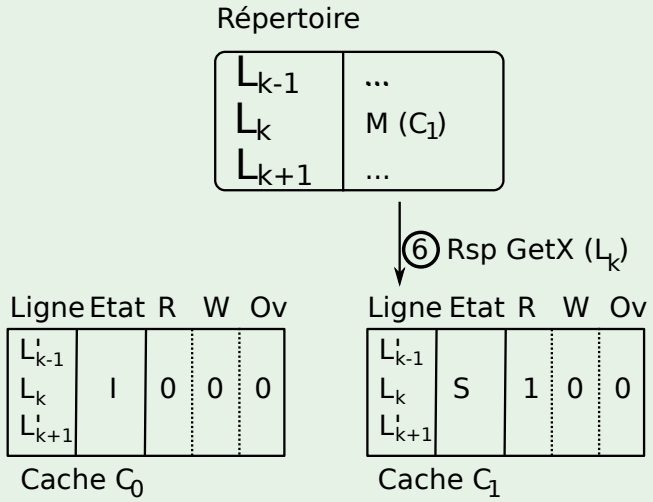
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 1



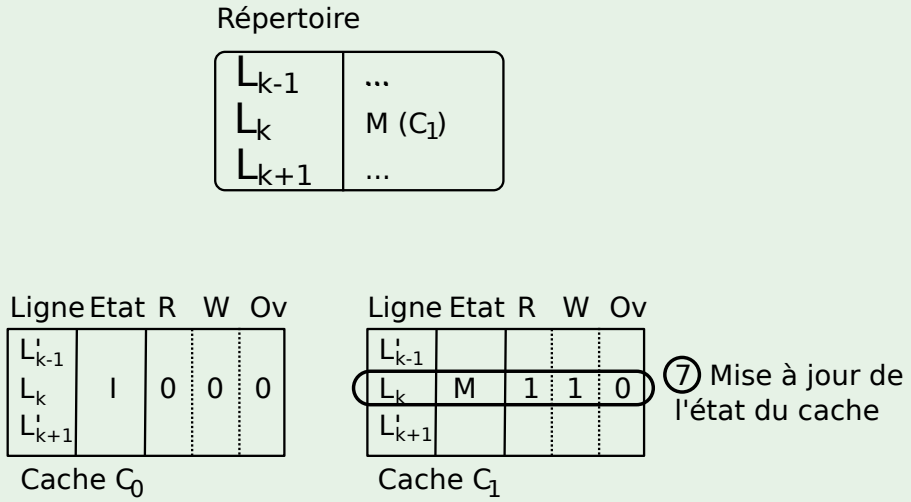
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 1



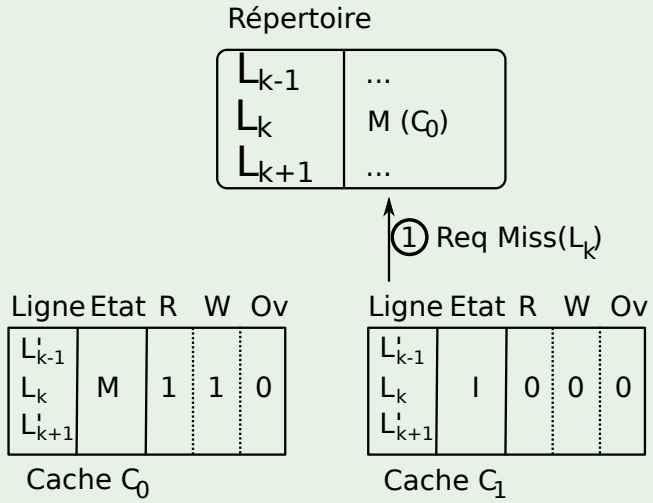
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 2



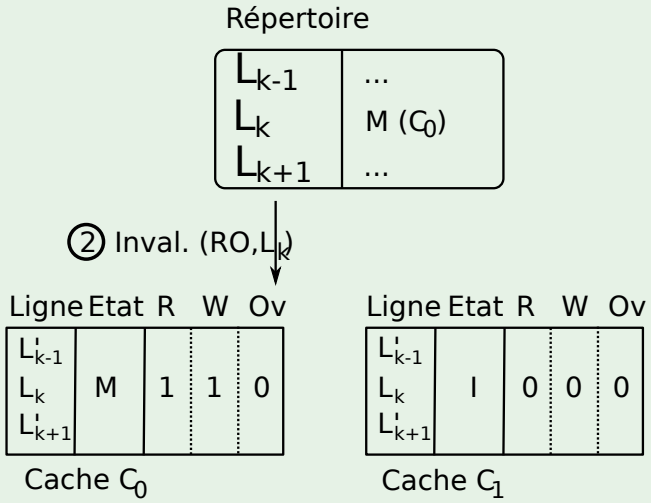
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 2



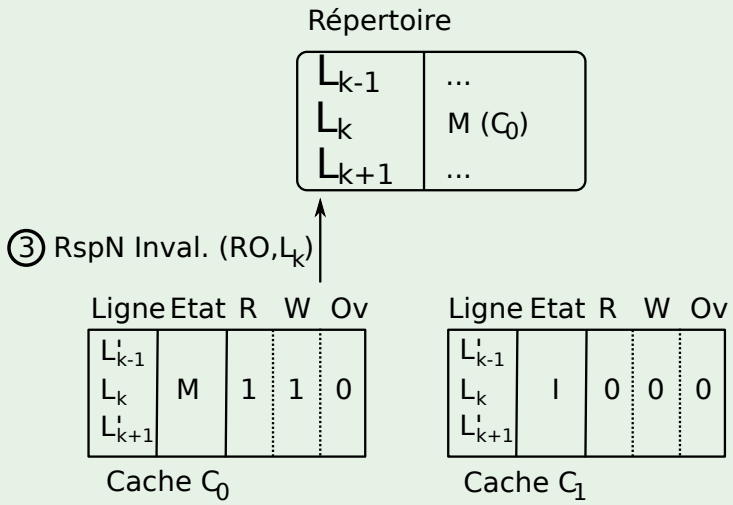
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 2



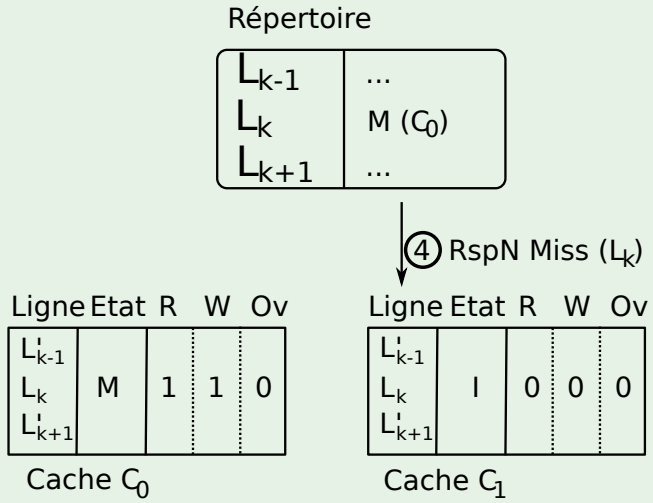
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 2



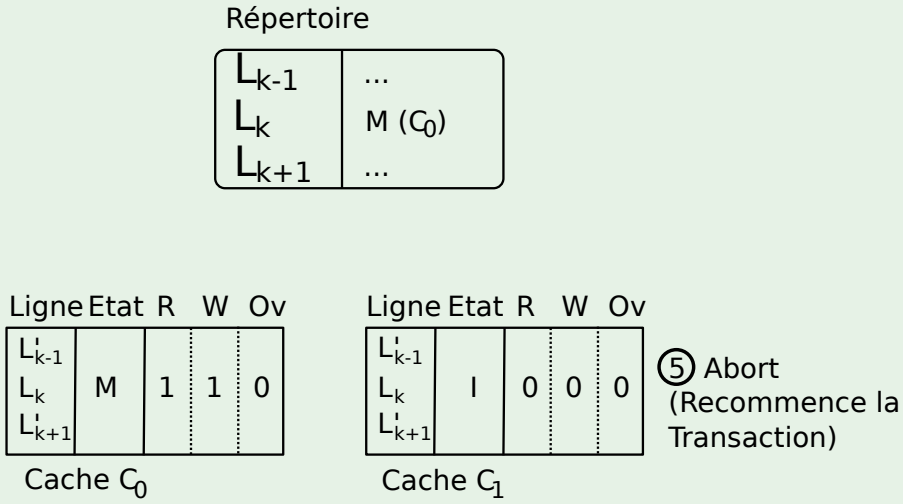
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 2



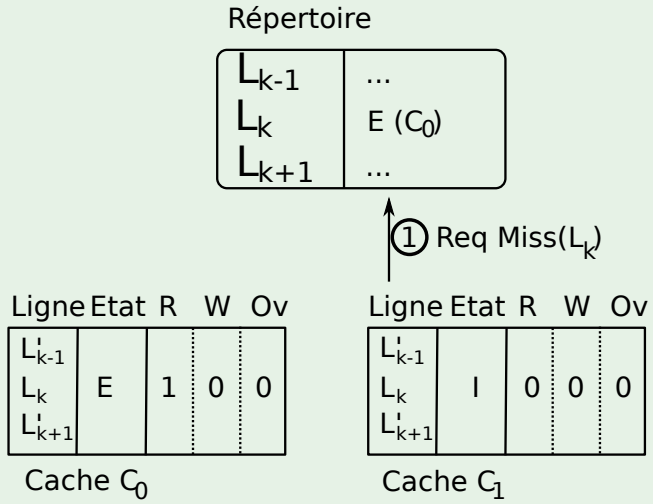
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 3



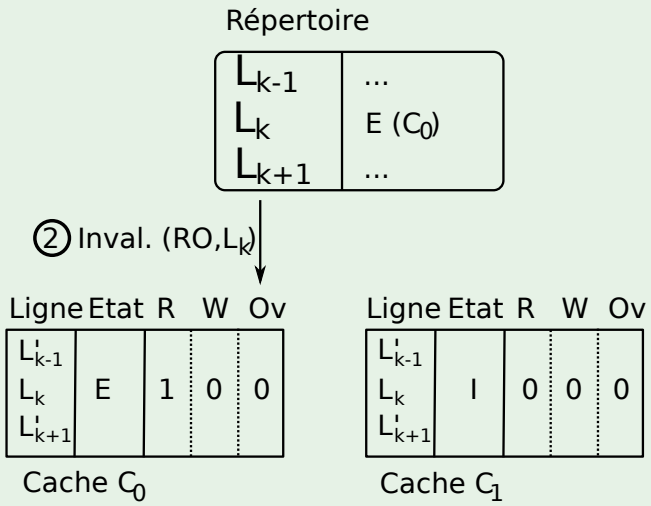
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 3



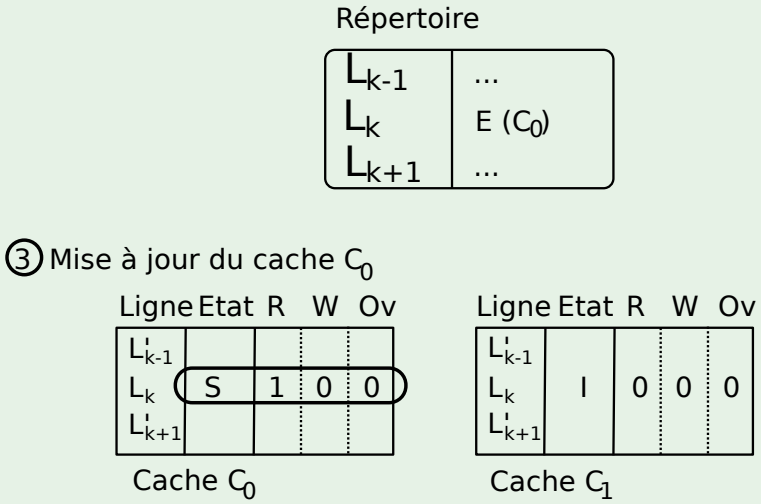
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 3



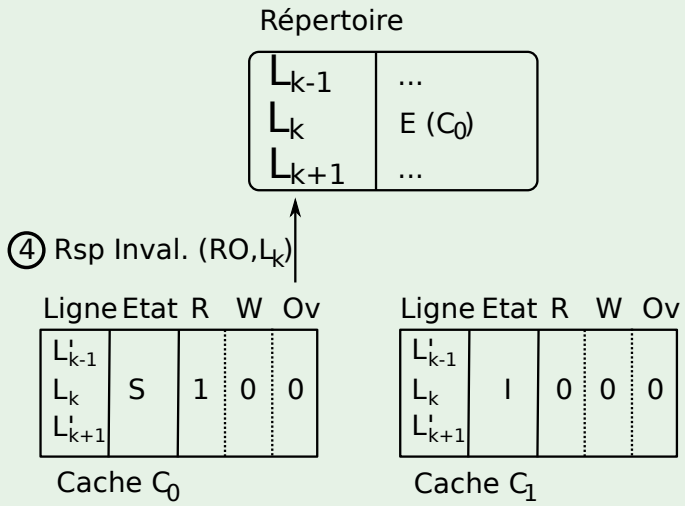
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 3



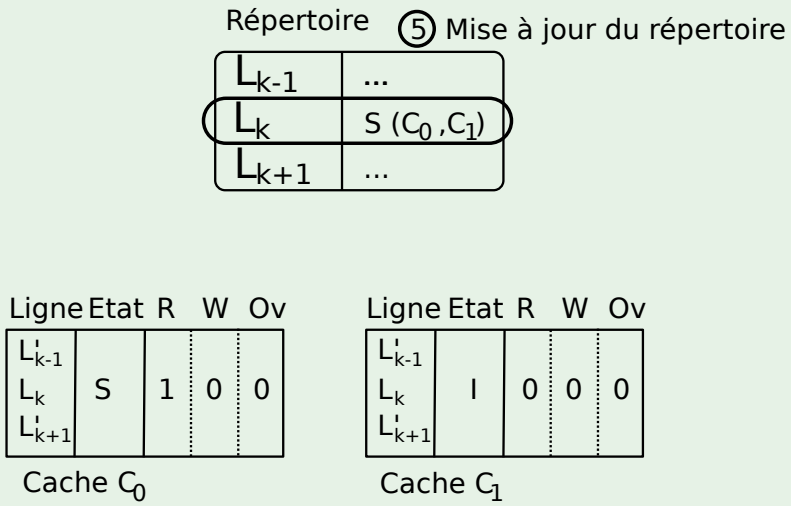
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 3



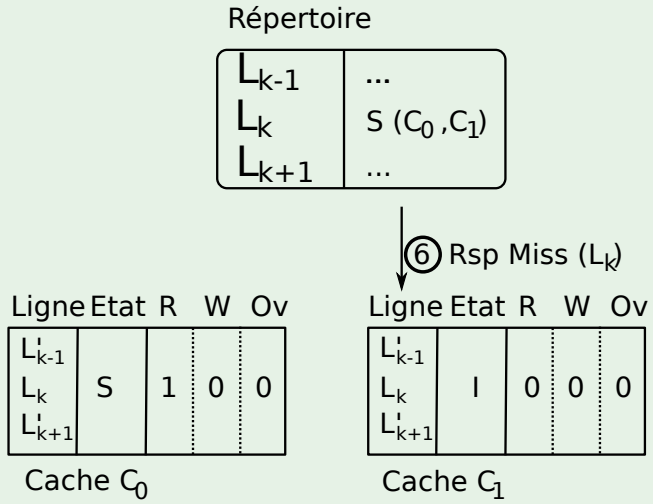
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 3



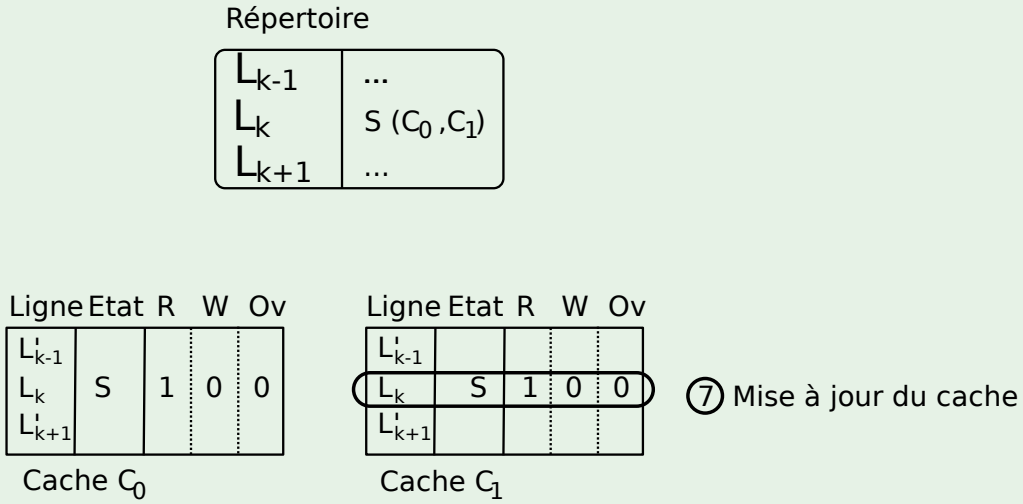
Détection des conflits en Write-Back

Principe

- Étend la détection des conflits du protocole write-back MESI
- Modifier spéculativement une ligne requiert le droit d'exclusivité
- Nombre d'états de la ligne augmenté (M,E,S,I x RT x WT x Ov)
- Réponse positive ou négative aux requêtes d'invalidation

Exemple

Exemple 3



Détection des conflits en Write-Back

Actions prises par un cache lorsqu'une invalidation est reçue

- '-' → bit à 0
- 'X' → valeur indifférente
- Les configurations non listées ne sont pas possibles

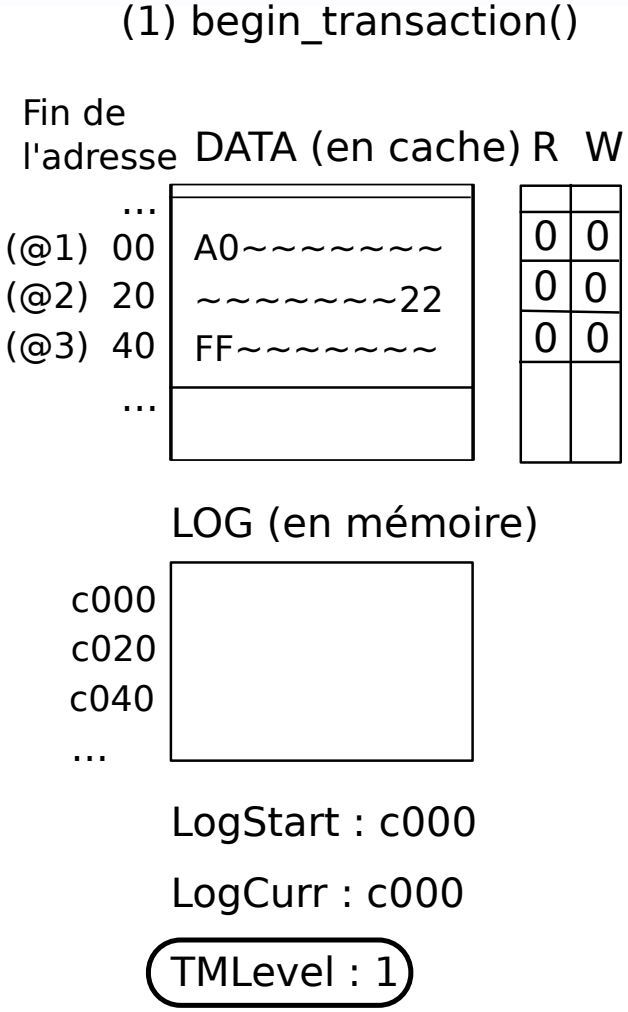
État de la ligne	**État transactionnel de la ligne**			**Type d'invalidation**	**Action(s) prise(s)**
M,E,S,I	**R**ead	**W**ritten	**Ov**erflowed (Débordé)	**C**omplète ou **R**ead-**O**nly	Ack, Nack, abort local Inval, Inval-RO
I	-	-	-	RO,C	Ack
S	-	-	-	RO	Ack (*)
S	-	-	-	C	Inval, Ack
E	-	-	-	RO	Inval-RO, Ack
E	-	-	-	C	Inval, Ack
M	-	-	-	RO	Inval-RO, Ack
M	-	-	-	C	Inval, Ack
S	R	-	-	RO	Ack (*)
S	R	-	-	C	Inval, abort local, Ack
E	R	-	-	RO	Inval-RO, Ack
E	R	-	-	C	Inval, abort local, Ack
M	X	W	-	RO,C	Nack
X	X	X	Ov	RO,C	Nack

Gestion des versions en Write-Back

Principe

- Écriture en log des lignes avant modification
- Le cache contient les nouvelles valeurs
- Log inutile en cas de commit
- En cas d'abort, parcours du log et restauration des valeurs en mémoire

Gestion des versions en Write-Back



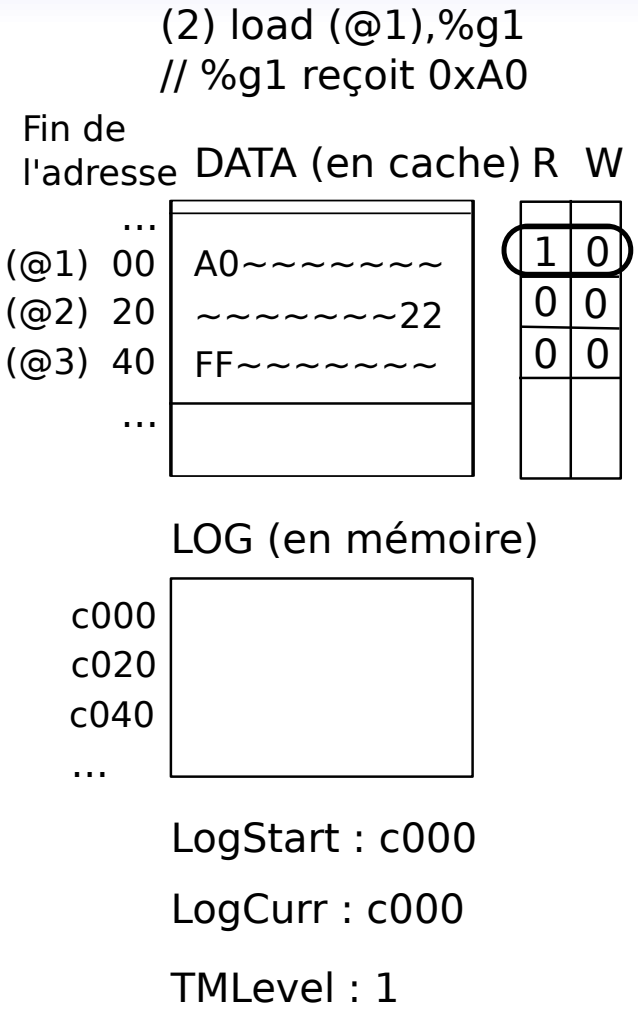
Principe

- Écriture en log des lignes avant modification
- Le cache contient les nouvelles valeurs
- Log inutile en cas de commit
- En cas d'abort, parcours du log et restauration des valeurs en mémoire

Exemple

- Début d'une transaction

Gestion des versions en Write-Back



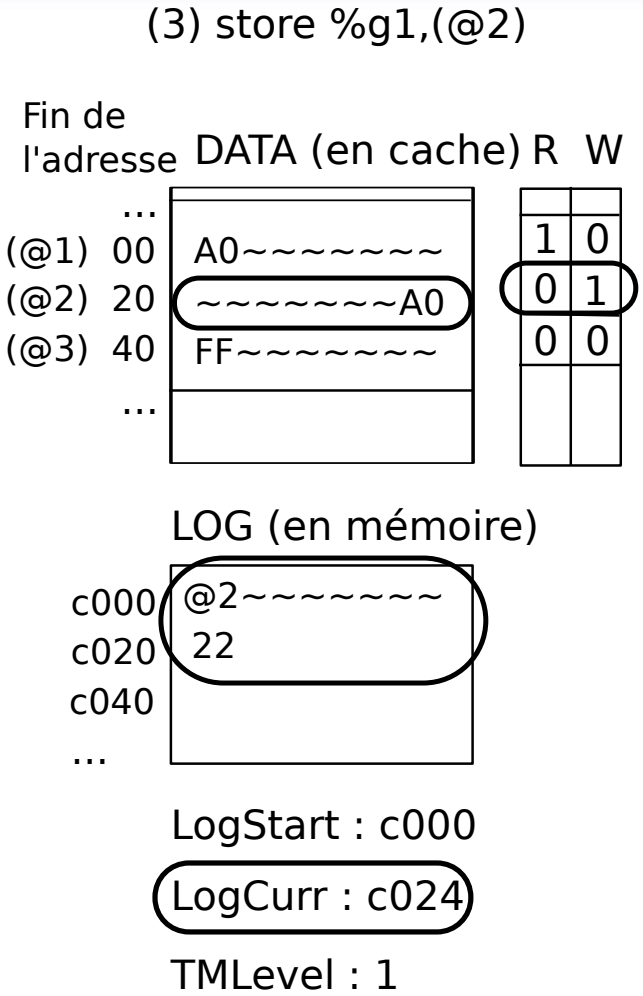
Principe

- Écriture en log des lignes avant modification
- Le cache contient les nouvelles valeurs
- Log inutile en cas de commit
- En cas d'abort, parcours du log et restauration des valeurs en mémoire

Exemple

- Début d'une transaction
- Lecture : pas d'adresse ajoutée

Gestion des versions en Write-Back



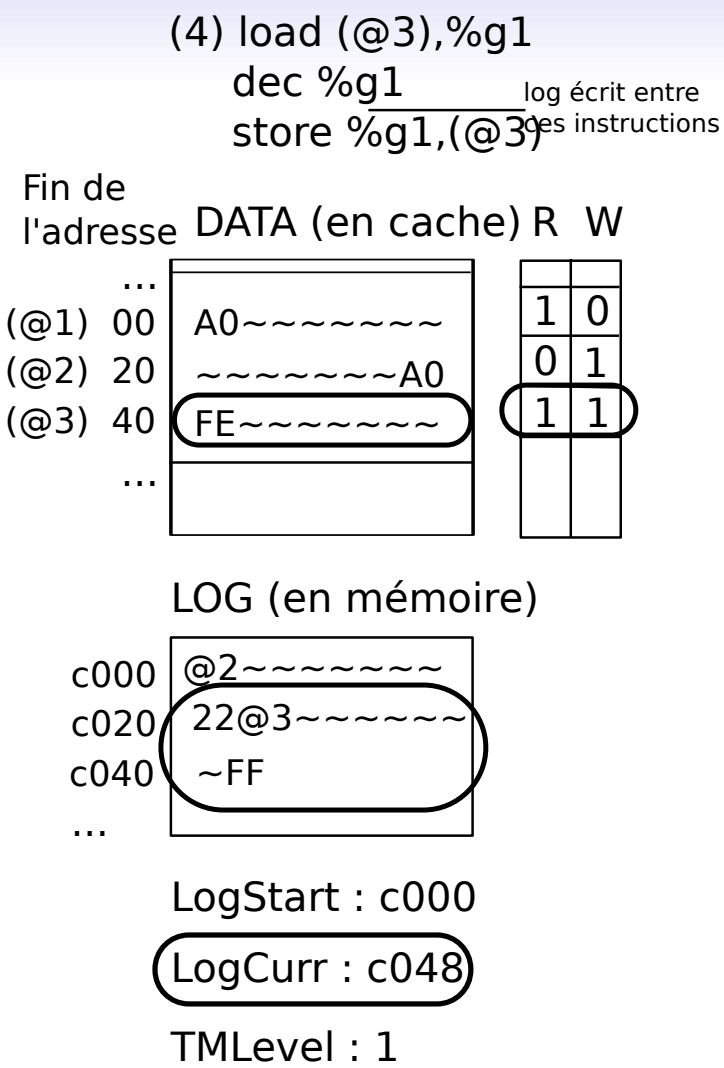
Principe

- Écriture en log des lignes avant modification
- Le cache contient les nouvelles valeurs
- Log inutile en cas de commit
- En cas d'abort, parcours du log et restauration des valeurs en mémoire

Exemple

- Début d'une transaction
- Lecture : pas d'adresse ajoutée
- Écriture : ajout de la ligne complète dans le log

Gestion des versions en Write-Back



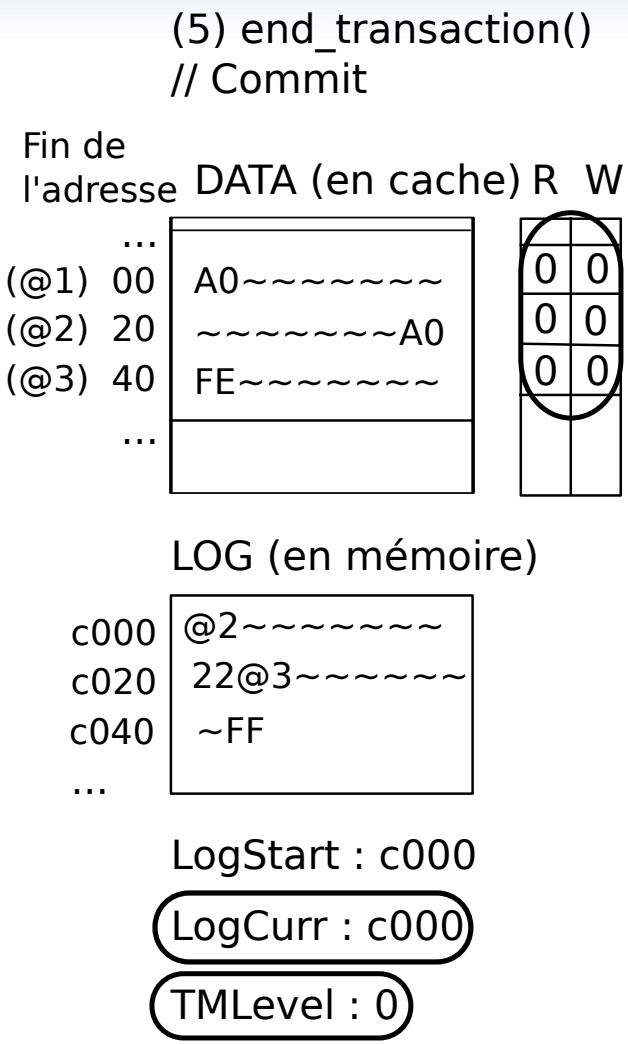
Principe

- Écriture en log des lignes avant modification
- Le cache contient les nouvelles valeurs
- Log inutile en cas de commit
- En cas d'abort, parcours du log et restauration des valeurs en mémoire

Exemple

- Début d'une transaction
- Lecture : pas d'adresse ajoutée
- Écriture : ajout de la ligne complète dans le log
- Lecture-Modification-Écriture : ajout de la ligne complète dans le log

Gestion des versions en Write-Back



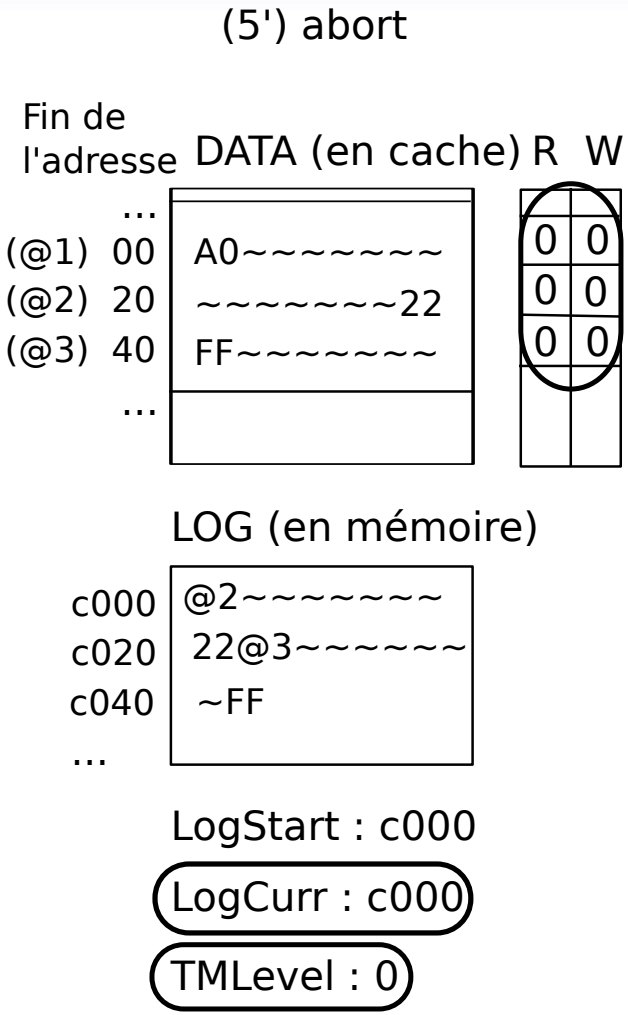
Principe

- Écriture en log des lignes avant modification
- Le cache contient les nouvelles valeurs
- Log inutile en cas de commit
- En cas d'abort, parcours du log et restauration des valeurs en mémoire

Exemple

- Début d'une transaction
- Lecture : pas d'adresse ajoutée
- Écriture : ajout de la ligne complète dans le log
- Lecture-Modification-Écriture : ajout de la ligne complète dans le log
- Commit : remise à 0 des bits R/W/Ov

Gestion des versions en Write-Back



Principe

- Écriture en log des lignes avant modification
- Le cache contient les nouvelles valeurs
- Log inutile en cas de commit
- En cas d'abort, parcours du log et restauration des valeurs en mémoire

Exemple

- Début d'une transaction
- Lecture : pas d'adresse ajoutée
- Écriture : ajout de la ligne complète dans le log
- Lecture-Modification-Écriture : ajout de la ligne complète dans le log
- Commit : remise à 0 des bits R/W/Ov
- Abort : parcours du log, restauration des valeurs initiales

Gestion des débordements en Write-Back

Principe

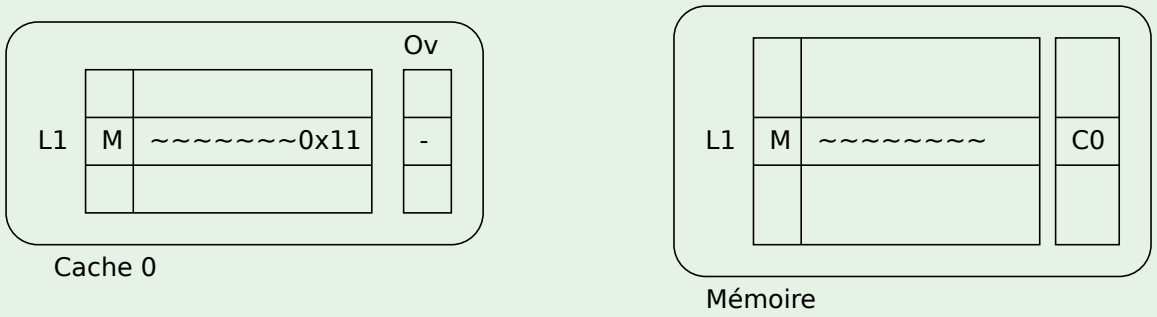
- Bit de débordement par ligne de cache (Ov)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit Ov à 1
- Faux conflits possibles

Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (Ov)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit Ov à 1
- Faux conflits possibles

Exemple

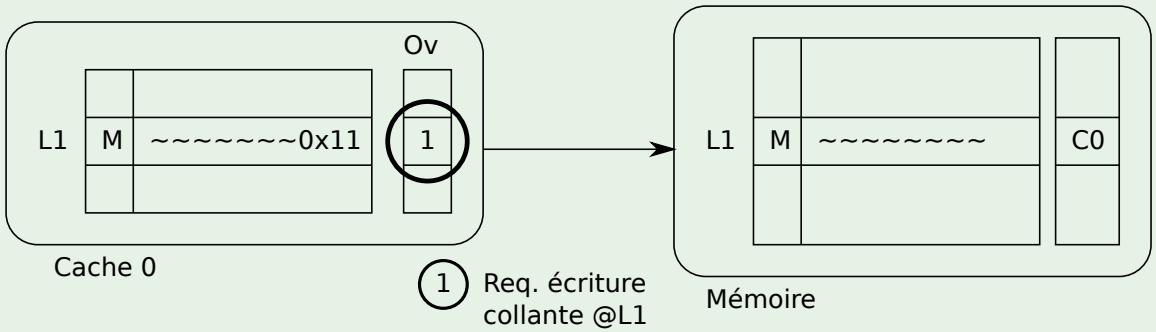


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

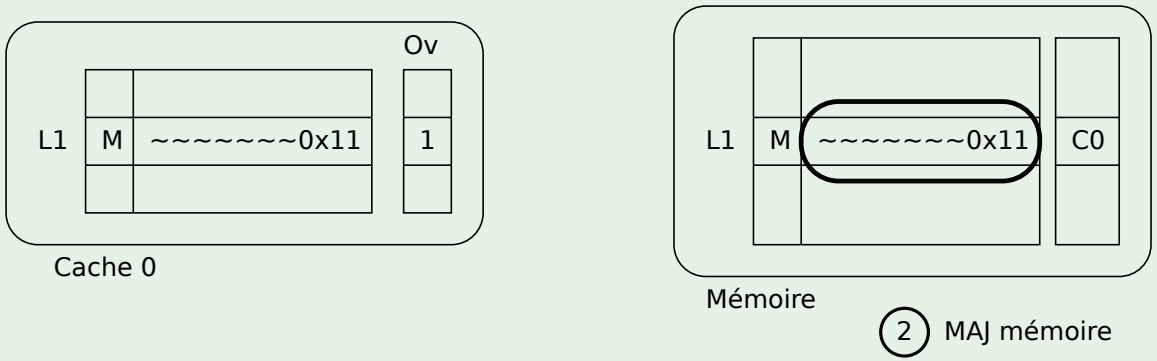


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

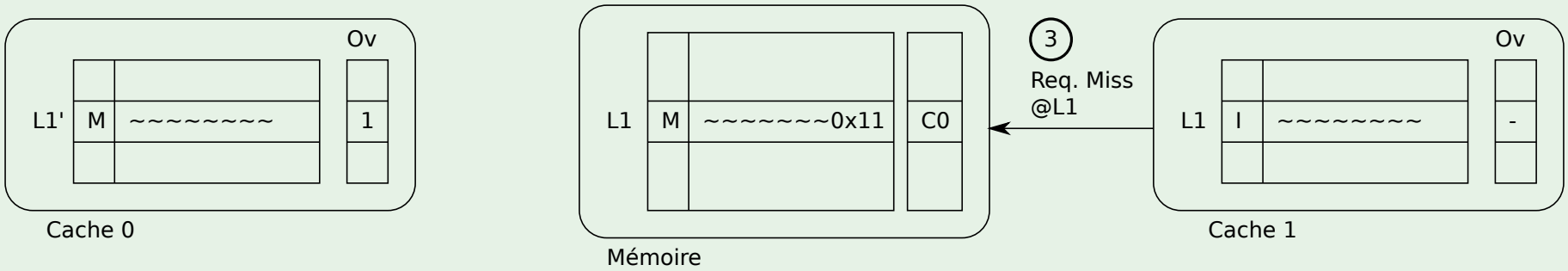


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

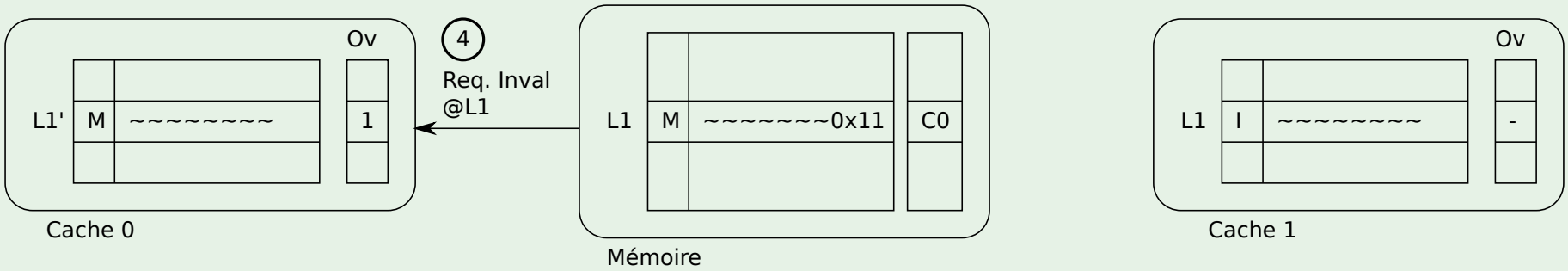


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

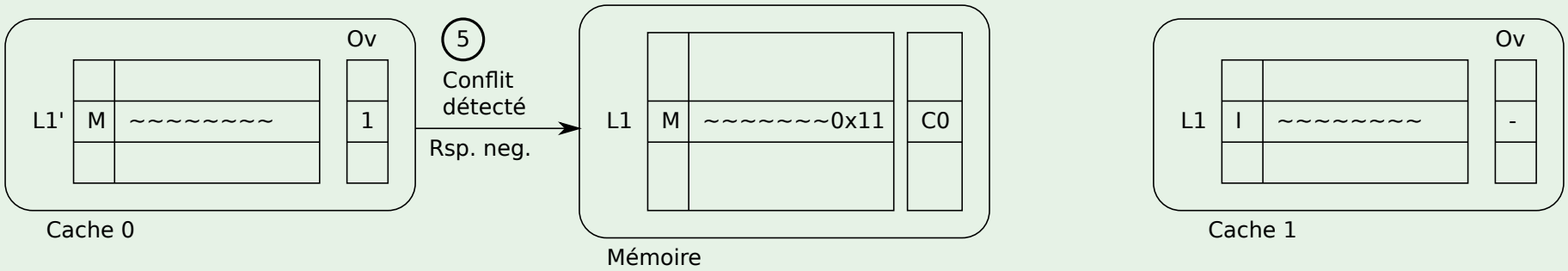


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

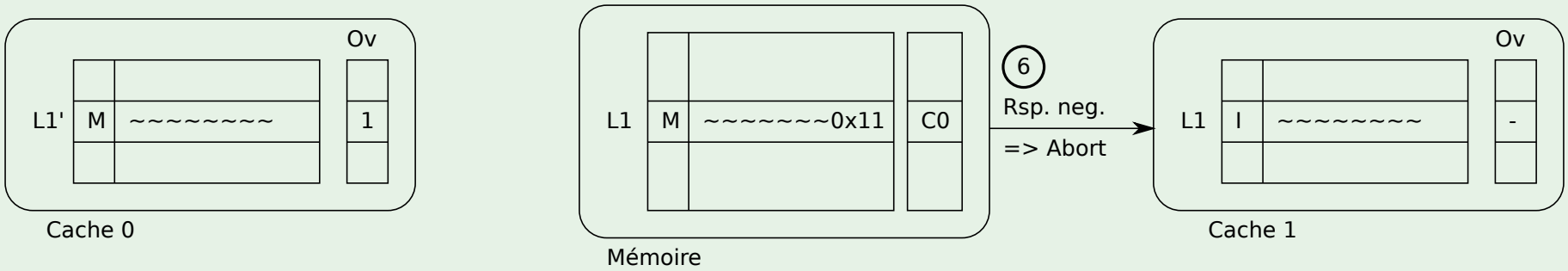


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

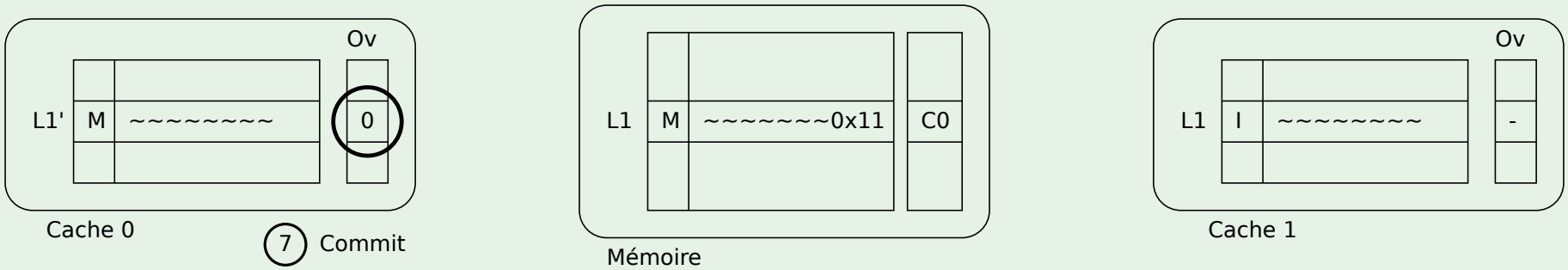


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

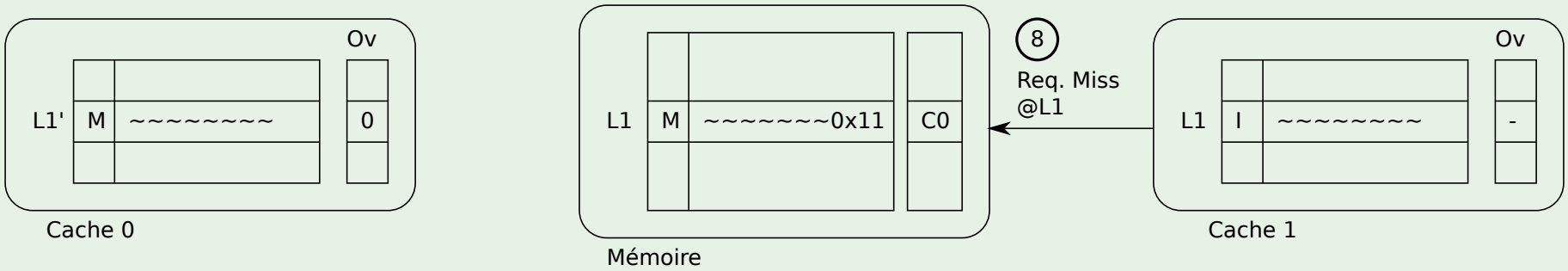


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

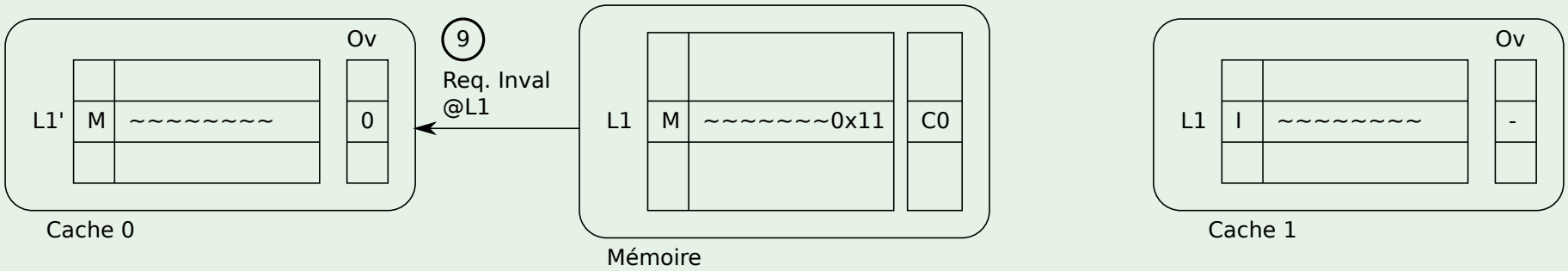


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

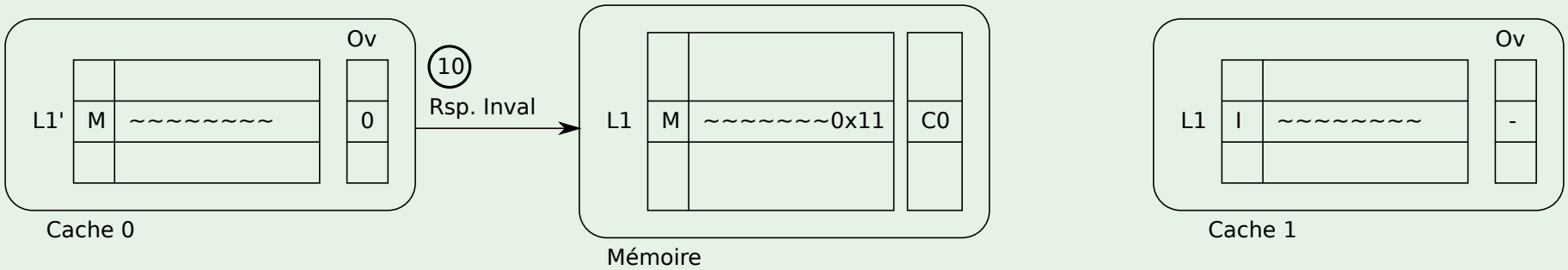


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

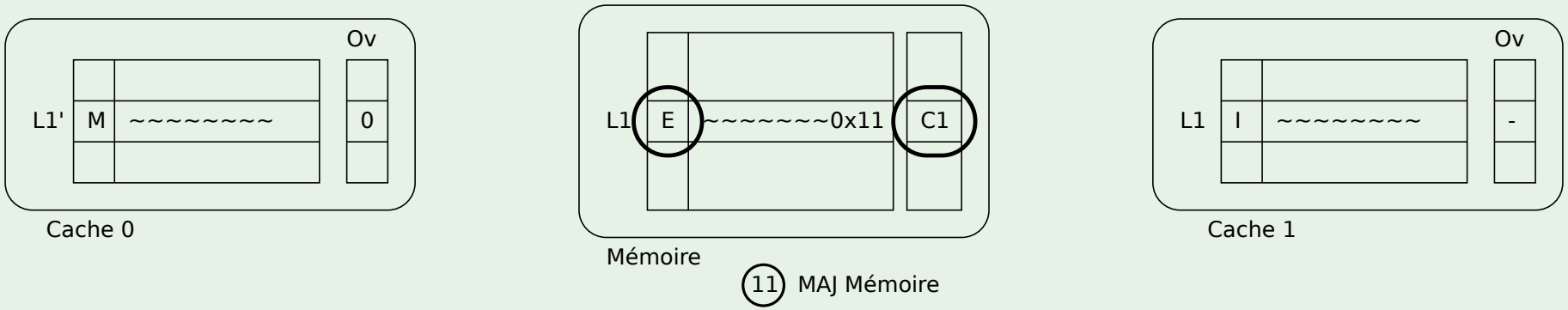


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

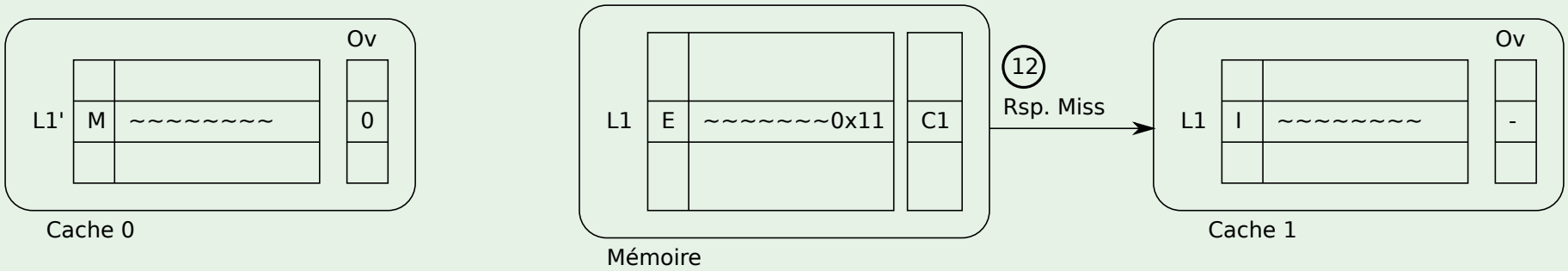


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

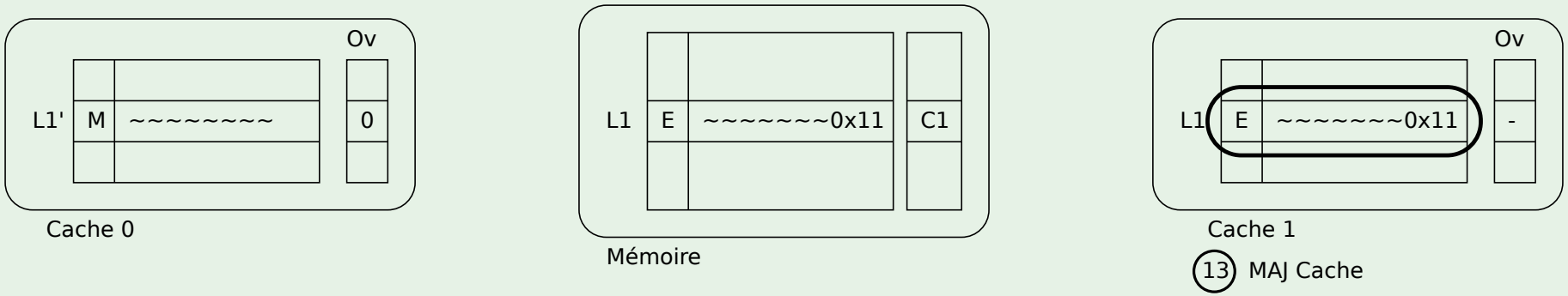


Gestion des débordements en Write-Back

Principe

- Bit de débordement par ligne de cache (*Ov*)
- Débordement : ligne recopiée en mémoire et marquée comme étant toujours modifiée par le cache
- Le répertoire fait suivre les requêtes
- Conflit détecté au niveau du cache si la ligne correspondante a le bit *Ov* à 1
- Faux conflits possibles

Exemple

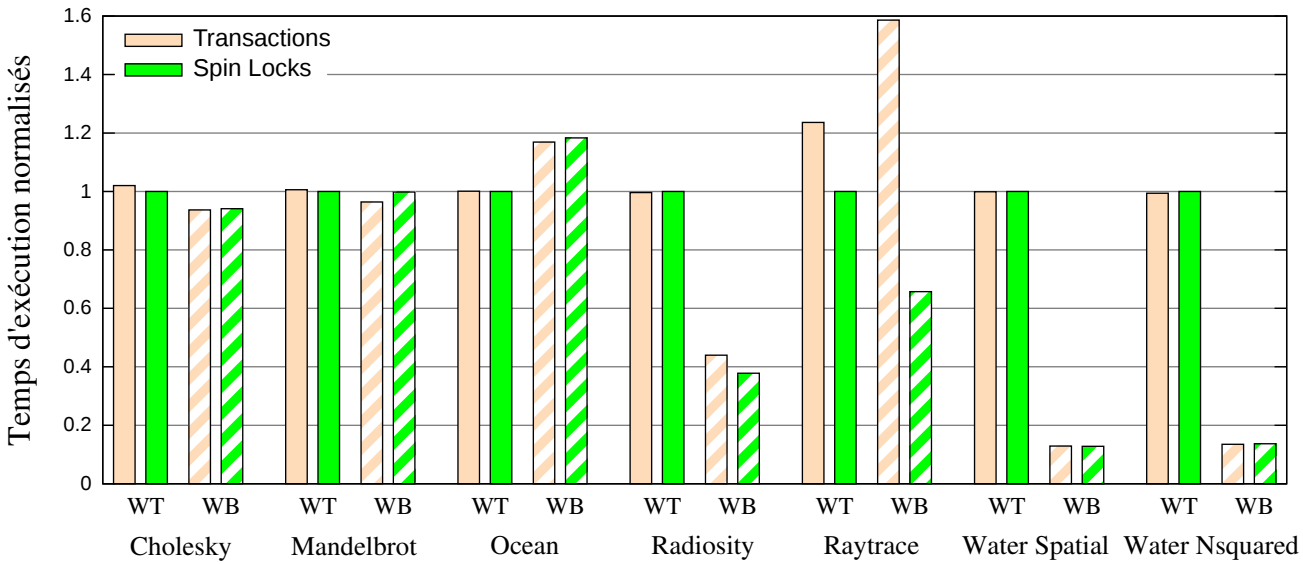


Benchmarks : Applications Splash-2

Analyse des résultats

- Avantage au write-back (5/7)
- Différences entre transactions et verrous faibles : performance plus dépendante du PCC que du choix transactions/verrous
- Sections critiques pas assez importantes ?
- Assez loin des résultats de la littérature

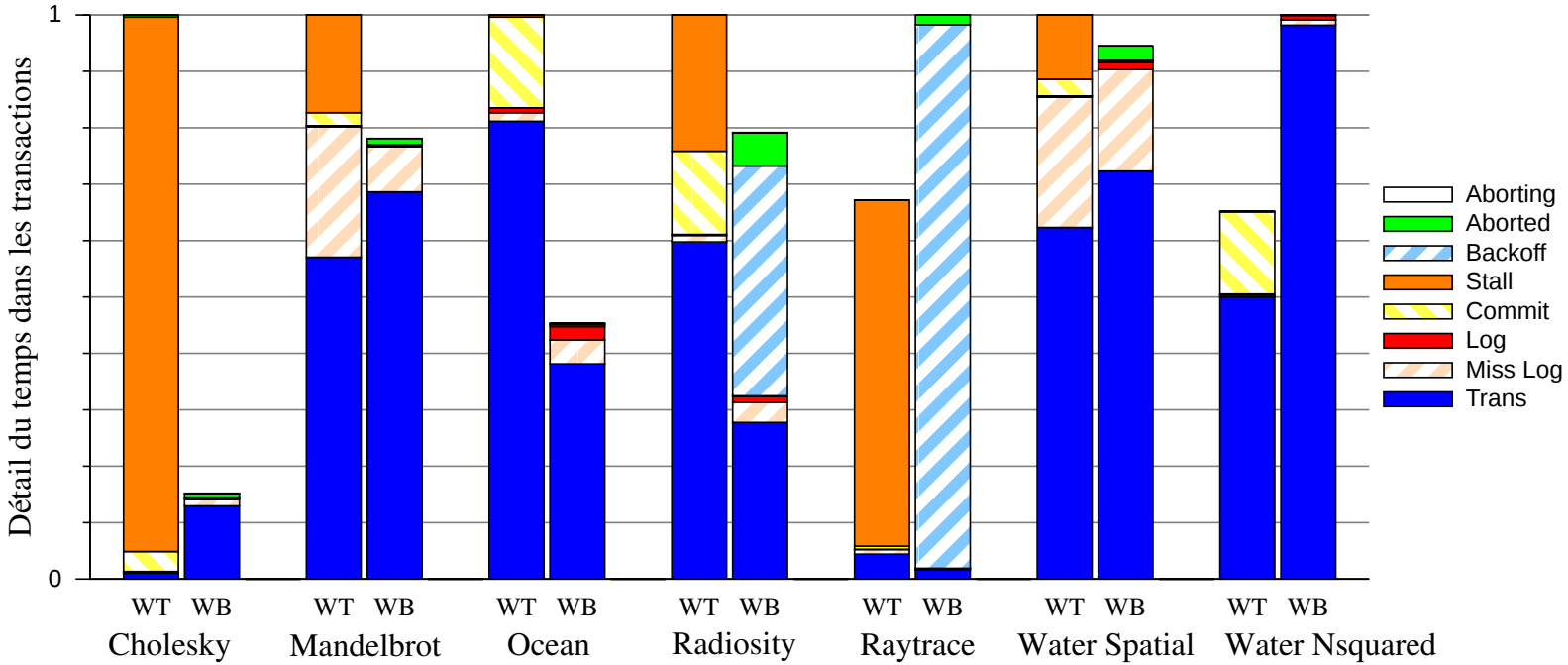
Application	Données en entrée
Cholesky	tk14
Mandelbrot	résol. 300x200 3000 iter. max.
Ocean	partitions contigües, 258
Radiosity	Room
Raytrace	teapot (256 x 256)
Water N-Squared	512 molécules
Water Spatial	512 molécules



Benchmarks : Applications SPLASH-2

Détail du temps dans les transactions

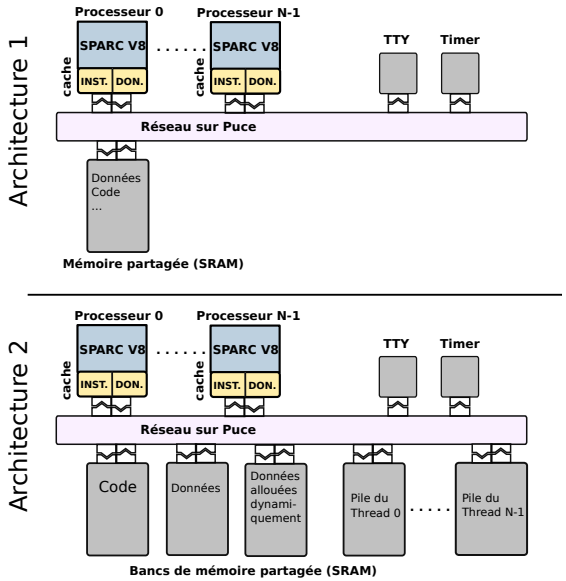
- Normalisé par rapport à la configuration ayant passé le plus de temps dans les transactions
- Meilleure mesure pour comparer les deux systèmes ? (Ocean, Water-Nsquared)
- Tendance toujours en faveur du write-back
- Temps de backoff et de gel très important pour certaines applications

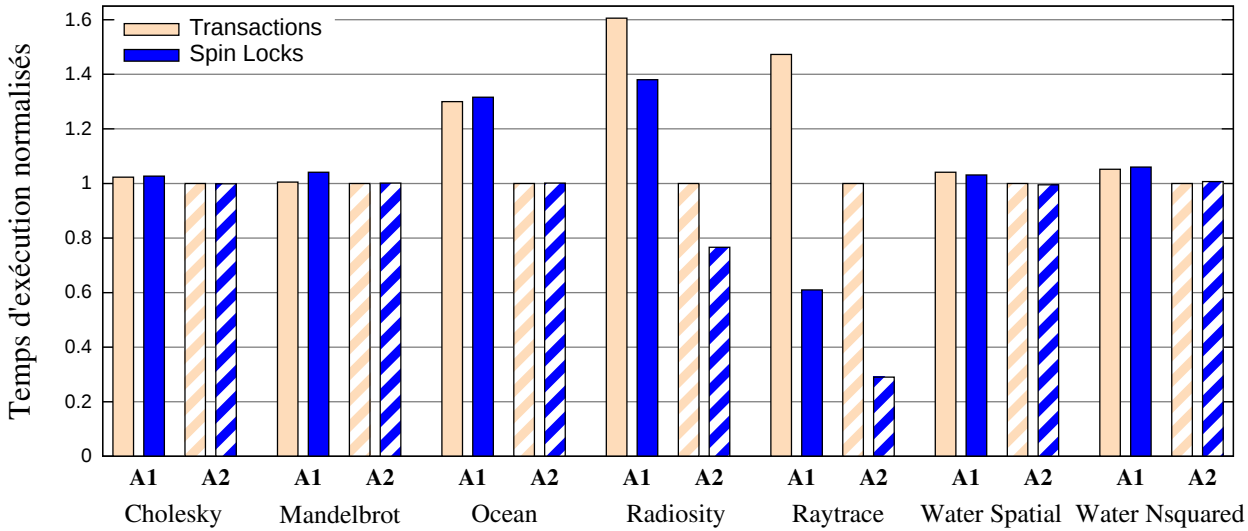


Benchmarks : Applications SPLASH-2

Impact de la répartition mémoire

- Données locales aux threads distribuées
- Ne concerne que le write-back
- La distribution a toujours un effet positif
- Donne un avantage au write-back pour les transactions
- Division du temps pour Raytrace par 2 (verrous) et 1.5 (transactions)





Résumé des différences entre les deux systèmes

Tableau récapitulatif

	Write-Through	Write-Back
Nombre de bancs mémoire	1	quelconque
Taille du log (en mots)	nb. lignes accédées	nb. lignes écrites * (mots/lig. + 1)
Surcoût du cache	2 bits/ligne	3 bits/ligne
Surcoût mémoire	élevé	faible
Nombre d'états des FSMs (implém.)	85 (49)	79 (62)
Lectures concurrentes	Non	Oui
Perfs. Benchmarks (tps tot. - tps trans.)	2 - 2	5 - 5

Conclusion

- Aucun système toujours meilleur que l'autre, choix possible
- Mais le protocole Write-Back présente un avantage global relativement net

Plan

1 Introduction

2 Problématique et cadre général du travail

3 Comparaison de 2 systèmes HTM basés sur des PCC différents

4 **Étude de différentes politiques de résolution des conflits**

5 Conclusion et perspectives

Motivation

Contexte

- Importance de la politique de résolution
- Focalisation sur une architecture avec un PCC de type write-back

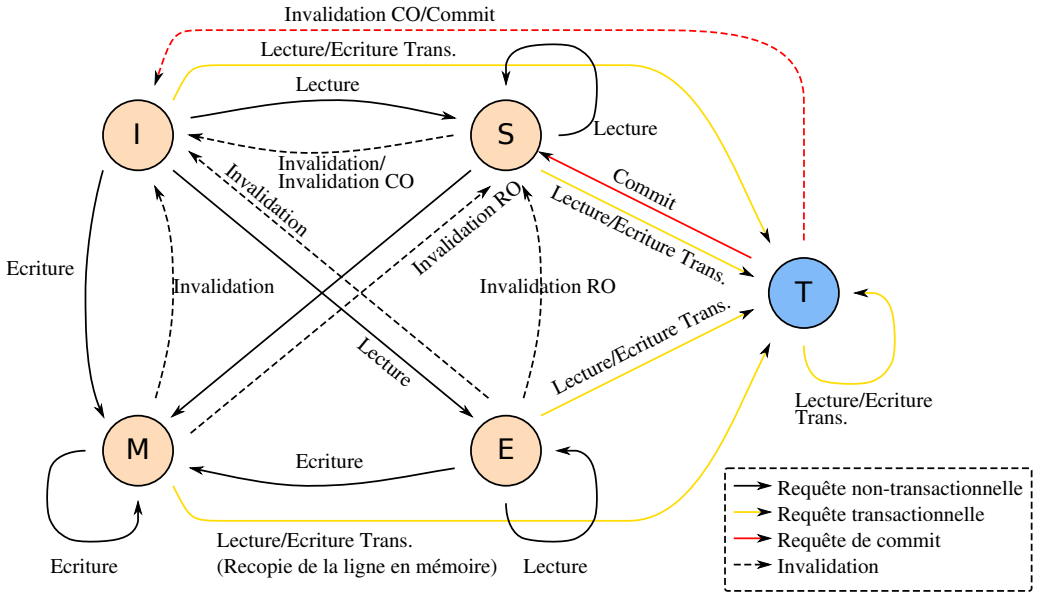
Travail effectué

- Comparaisons de plusieurs systèmes : 3 de type *Eager*/*Eager* (EE), et un de type *Lazy*/*Lazy* (LL)
- Impact du backoff en EE et LL
- Performances globales
- Quelles garanties d'un point de vue avancement du système ?
- Benchmarks STAMP : transactions (très) longues, pas d'implémentation à base de verrous

Principe du système de type LL

Détection et résolution des conflits

- Ajout d'un nouvel état T au protocole MESI
- Une ligne dans l'état T peut être partagée entre plusieurs processeurs, même en écriture
- Détection des conflits au commit, par envoi d'une requête d'invalidation particulière
- Résolution des conflits par abort des processeurs recevant une invalidation de ce type



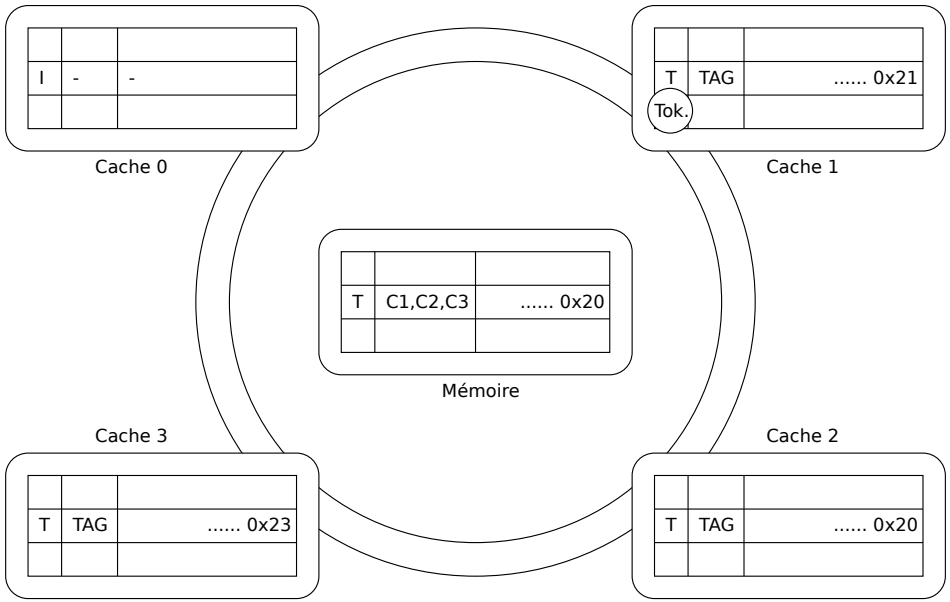
Gestion de version

- Valeurs spéculées mises dans un tampon
- Taille du tampon infinie
- Au commit, propagation de toutes les valeurs du tampon en mémoire

Principe du système de type LL

Détection et résolution des conflits

- Ajout d'un nouvel état T au protocole MESI
- Une ligne dans l'état T peut être partagée entre plusieurs processeurs, même en écriture
- Détection des conflits au commit, par envoi d'une requête d'invalidation particulière
- Résolution des conflits par abort des processeurs recevant une invalidation de ce type



Gestion de version

- Valeurs spéculées mises dans un tampon
- Taille du tampon infinie
- Au commit, propagation de toutes les valeurs du tampon en mémoire

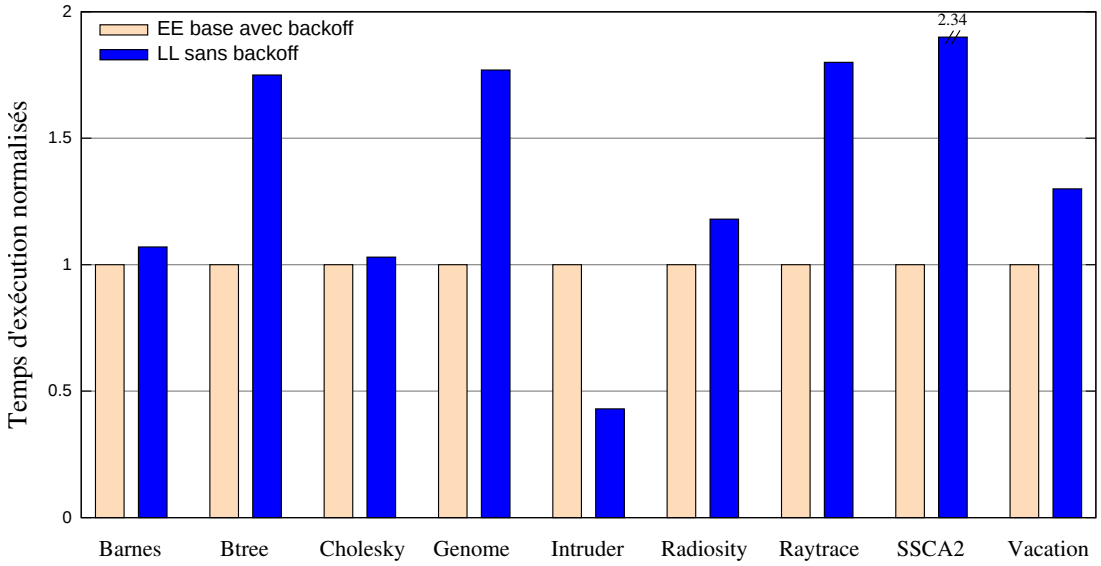
Comparaison des performances des politiques EE et LL

Choix des versions comparées

- Temps EE avec backoff
- Temps LL sans backoff : pas de garantie en plus et ralentissements

Résultats

- Variations importantes (0.43 à 2.34)
- EE plus rapide sur 8 des 9 applications : semble être un meilleur choix



Configuration :

Nombre de processeurs	$p = 16$
Nombre de bancs mémoire	$p + 3$
Modèle du processeur	SPARC-V8 avec FPU
Taille du cache (D)	16*Ko*
Taille des lignes du cache (D)	8 mots (32 octets)
Taille du cache (I)	16*Ko*
Taille des lignes du cache (I)	8 mots
Associativité du cache	Correspondance directe
Taille du tampon d'écritures	8 mots
Topologie du NoC	Maillage 2D

Résolution des conflits dans un système HTM

Politique de résolution des conflits

- Actions à entreprendre pour résoudre un conflit
- Buts :
  - Diminuer le nombre de conflits futurs
  - Éviter les pathologies
  - Assurer des performances
- Lié à la notion de Gestionnaire de contention (*Contention Manager*)
  - Rôle : Limiter la contention après un abort en retardant la ré-exécution de la transaction abortée
  - Par extention, on peut aussi attribuer la politique de résolution des conflits au gestionnaire de contention
- Gestionnaire implémenté en matériel (plus efficace) ou en logiciel (plus flexible)

Politique de résolution des conflits dans un système HTM

Degrés de liberté lors d'un conflit dans un système EE

- Choix de la transaction affectée
- Retenter la requête ou la transaction (possiblement après n requêtes)
- Temps d'attente entre 2 tentatives de requêtes/transactions

Choix de la transaction affectée

- Fonction du type des requêtes (ex : privilégier les écritures)
- Fonction de la transaction "acquéreuse" T_A (*vs.* la transaction "victime" T_V)
- Fonction de l'historique des conflits
- Fonction d'un score de priorité
  - Numéro de processeur
  - Temps d'exécution de la transaction (estampille)
  - Nombre d'accès en mémoire (lectures et/ou écritures)
  - Nombre d'adresses différentes accédées (R/W) ou de lignes en débordement
  - Nombre de conflits gagnés et/ou perdus
  - Cumulatif ou non après un abort

Politique de résolution des conflits dans un système HTM

Exemples de gestionnaires de contention

- *Passive* : T_A aborte
- *Polite* : T_A retente sa requête N fois (attente croît de manière exponentielle) puis aborte T_V
- *Karma* : score = nombre cumulé d'accès à une donnée ; si $s(T_A) > s(T_V)$, T_V aborte ; sinon T_A retente sa requête $(s(T_V) - s(T_A))$ fois puis aborte TV
- *Eruption* : Karma + ajout du score de la transaction aborté à la transaction gagnante
- *Greedy*
- *Greedy-FT*
- *Timestamp*
- *Published-Timestamp*
- *Kindergarten*
- *Polka*
- ...

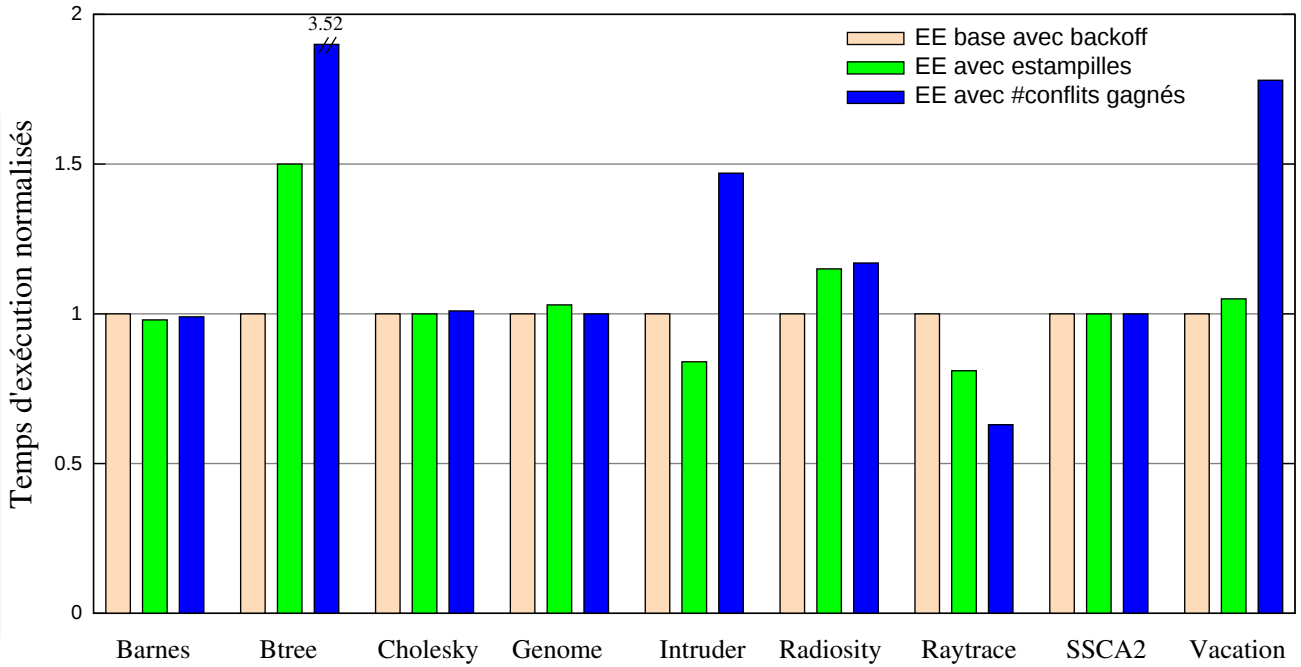
Définition et performances d'autres politiques EE

Différentes politiques

- Résolution des conflits basique avec priorité aux écritures (*EE-base*)
- Résolution des conflits à base d'estampilles (*EE-est*)
- Résolution des conflits basée sur le nombre de conflits gagnées (*EE-ncg*)

Résultats

- Résolution basée sur le nombre de conflits gagnés moins performante que les deux autres politiques
- Résolution à base d'estampilles un petit peu plus lente que la résolution de base



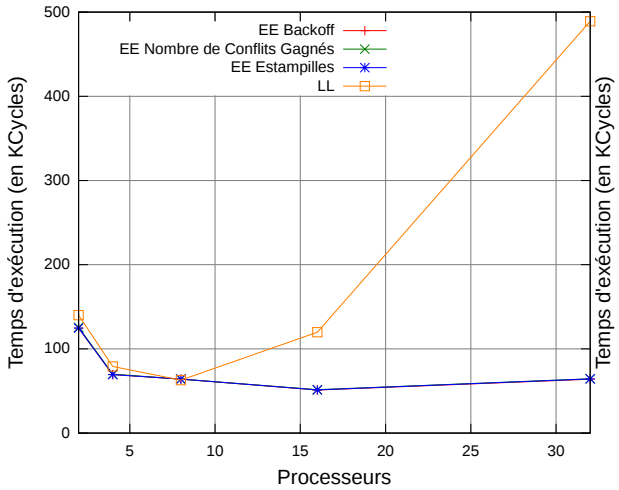
Influence des paramètres architecturaux

Configuration

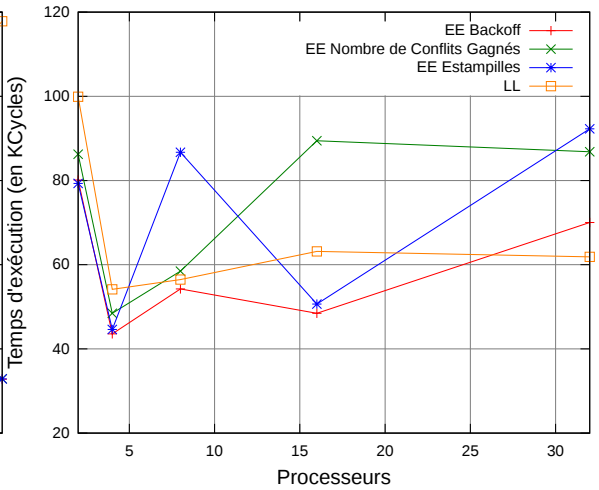
- Paramètres : nombre de processeurs, latence du réseau, taille des caches
- Applications : SSCA2 (peu de conflits), Vacation (nombre élevé de conflits)

Valeurs des paramètres :

Nombre de processeurs	2,4,8,16,32
Latence du NoC (cycles)	4,7,10,13,17
Taille des caches (Ko)	2,4,8,16,32



SSCA2



Vacation

Résultats

- LL prohibitif avec un grand nombre de processeurs et sans conflits
- Avec conflits, tendances moins claires

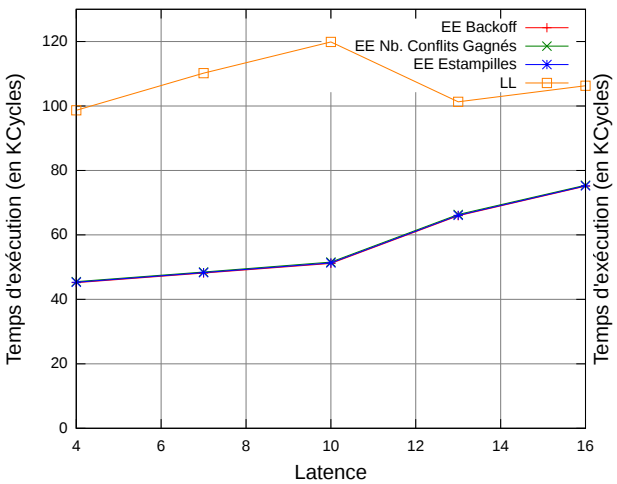
Influence des paramètres architecturaux

Configuration

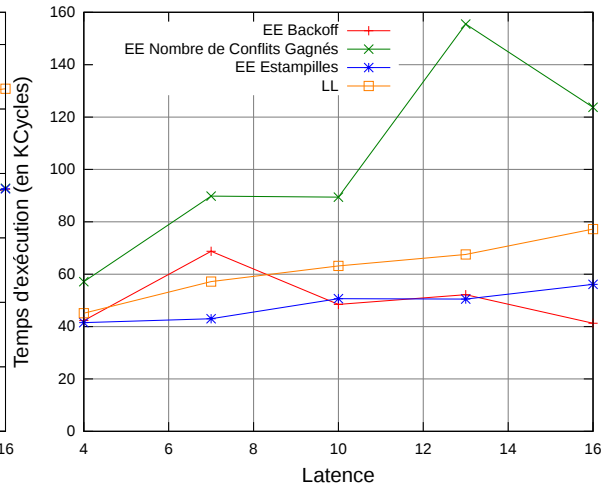
- Paramètres : nombre de processeurs, latence du réseau, taille des caches
- Applications : SSCA2 (peu de conflits), Vacation (nombre élevé de conflits)

Valeurs des paramètres :

Nombre de processeurs	2,4,8,16,32
Latence du NoC (cycles)	4,7,10,13,17
Taille des caches (Ko)	2,4,8,16,32



SSCA2



Vacation

Résultats

- LL prohibitif avec un grand nombre de processeurs et sans conflits
- Avec conflits, tendances moins claires
- Impact de la latence : linéaire sans conflit (sauf LL), plus arbitraire avec conflits

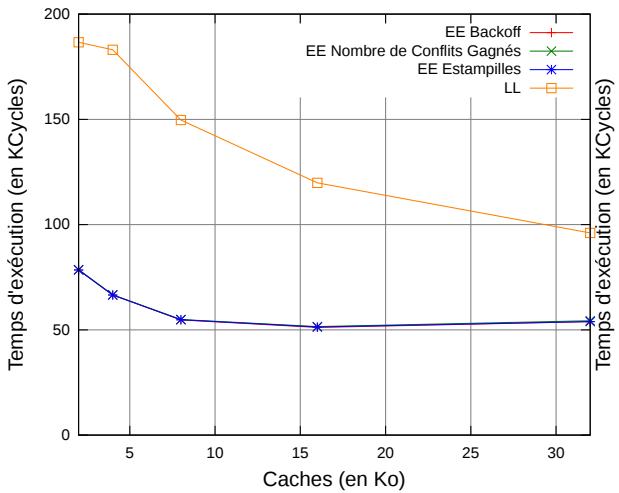
Influence des paramètres architecturaux

Configuration

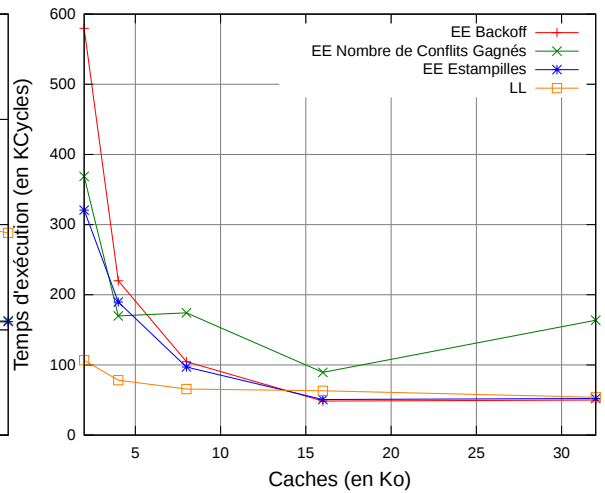
- Paramètres : nombre de processeurs, latence du réseau, taille des caches
- Applications : SSCA2 (peu de conflits), Vacation (nombre élevé de conflits)

Valeurs des paramètres :

Nombre de processeurs	2,4,8,16,32
Latence du NoC (cycles)	4,7,10,13,17
Taille des caches (Ko)	2,4,8,16,32



SSCA2



Vacation

Résultats

- LL prohibitif avec un grand nombre de processeurs et sans conflits
- Avec conflits, tendances moins claires
- Impact de la latence : linéaire sans conflit (sauf LL), plus arbitraire avec conflits
- Excepté LL, taille des caches non cruciale sans conflits
- Avec conflits, des petits caches plombent les politiques EE

Résistance aux pathologies

Pathologies dans les systèmes TM

- Existence de plusieurs types de pathologies
- Critère utilisé ici : absence d'étreinte active
- Garanties au niveau du commit de transactions :
  - Commit d'une transaction dans le futur (CTQ)
  - Commit d'une transaction en cours sans abort (CTC)
  - Commit d'une transaction donnée (CTD)
- Garantie au niveau du système :
  - Le programme se termine (FP)
- Nombre de transactions (NT) dans l'application : borné ou non-borné

Système	Garanties	NT borné	NT non borné
LL	CTC	FP	-
EE-base	CTQ	FP	-
EE-est	??	??	-
EE-ncg	CTQ	FP	-

Résistance aux pathologies

Pathologies dans les systèmes TM

- Existence de plusieurs types de pathologies
- Critère utilisé ici : absence d'étreinte active
- Garanties au niveau du commit de transactions :
  - Commit d'une transaction dans le futur (CTQ)
  - Commit d'une transaction en cours sans abort (CTC)
  - Commit d'une transaction donnée (CTD)
- Garantie au niveau du système :
  - Le programme se termine (FP)
- Nombre de transactions (NT) dans l'application : borné ou non-borné

Système	Garanties	NT borné	NT non borné
LL	CTC	FP	-
EE-base	CTQ	FP	-
EE-est	??	??	-
EE-ncg	CTQ	FP	-

Problème

- Pas de garantie de terminaison si le nombre de transactions est non-borné

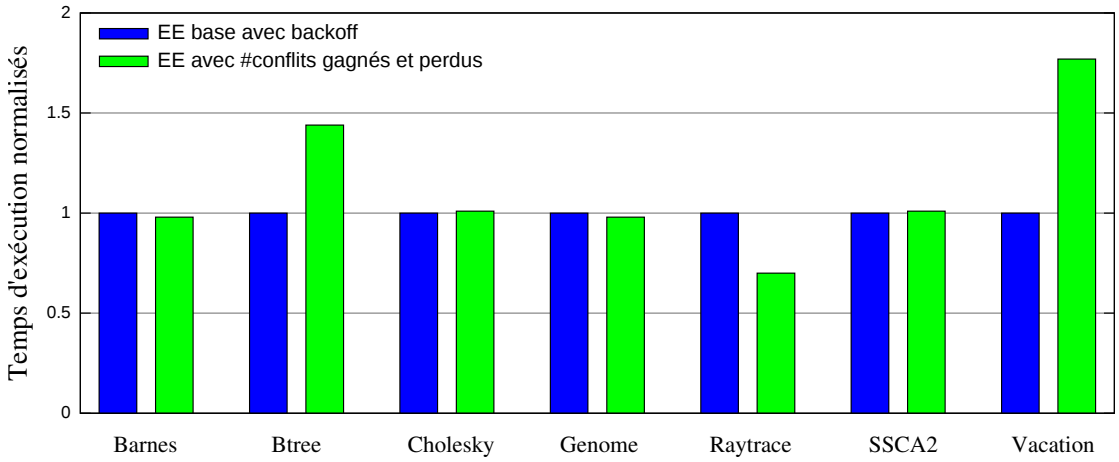
Nouvelle politique de résolution des conflits

Objectif

- Favoriser les transactions qui perdent pour qu'elles se terminent un jour
- Mais tout en favorisant encore plus les transactions qui gagnent
- Utilisation du principe des scores, mais basés cette fois sur le nombre de conflits gagnés et perdus

Définition/Exemple

- *Experience* : basée sur un score uniquement
  - T_1 en conflit avec T_2, $s(T_1) > s(T_2)$
  - T_1 gagne le conflit, T_2 aborte ; $s(T_1) \leftarrow s(T_1) + s(T_2) + 2$ et $s(T_2) \leftarrow s(T_2) + 1$



Résultats

- Comparable à EE-base
- "Garantie" qu'une transaction donnée arrive au commit
- Absence de famine
- Mais pas toujours possible...

Nouvelle politique de résolution des conflits

Objectif

- Favoriser les transactions qui perdent pour qu'elles se terminent un jour
- Mais tout en favorisant encore plus les transactions qui gagnent
- Utilisation du principe des scores, mais basés cette fois sur le nombre de conflits gagnés et perdus

Définition/Exemple

- *Experience* : basée sur un score uniquement
  - T_1 en conflit avec T_2, $s(T_1) > s(T_2)$
  - T_1 gagne le conflit, T_2 aborte ; $s(T_1) \leftarrow s(T_1) + s(T_2) + 2$ et $s(T_2) \leftarrow s(T_2) + 1$

Système	Garanties	NT borné	NT non borné
LL	CTC	FP	-
EE-base	CTQ	FP	-
EE-est	??	??	-
EE-ncg	CTQ	FP	-
EE-ncgp	CTD	FP	FP

Résultats

- Comparable à EE-base
- "Garantie" qu'une transaction donnée arrive au commit
- Absence de famine
- Mais pas toujours possible...

Plan

1 Introduction

2 Problématique et cadre général du travail

3 Comparaison de 2 systèmes HTM basés sur des PCC différents

4 Étude de différentes politiques de résolution des conflits

5 **Conclusion et perspectives**

Conclusion

Résultats

- Conviction que le modèle transactionnel sera utilisé dans le futur
- Protocole de cohérence de type write-back mieux adapté comme base
- La politique de résolution des conflits *EE-base* (avec priorité aux écritures et backoff) donne les meilleurs résultats parmi celles étudiées
- Proposition d'une politique de résolution qui permet d'éviter la famine dans un système HTM (quand applicable)

Problèmes ouverts

- Résistance aux pathologies
- Garanties de propriétés de vivacité
- Passage à l'échelle

Mémoires Transactionnelles : étude du protocole de cohérence de cache et de la politique de résolution des conflits

Quentin Meunier

Laboratoire d'Informatique de Paris 6
Équipe Alsoc
4 Place Jussieu, 75252 Paris, France

15 Novembre 2012