

Modèles du parallélisme et Processeurs modernes

Introduction

Le modèle classique du parallélisme est celui de l'entrelacement. Il correspond à un modèle de la mémoire appelé modèle de la *cohérence séquentielle* (*Sequential Consistency, SC*)

Les processeurs modernes n'implémentent pas le modèle SC, mais des modèles plus "relâchés" ("*relaxed memory models*") tels que "l'ordre de mémorisation total" (*Total Store Order, TSO*)

Modèles de mémoire

Sequential Consistency Tous les accès à la mémoire globale sont immédiatement visibles par tous les processeurs.

Total Store Order (TSO)

- ▶ Les opérations d'écriture passent par une zone tampon (buffer)
- ▶ Chaque processeur lit la valeur la plus récente dans sa zone tampon s'il y en a une
- ▶ Si non, la valeur conservée en mémoire est lue

Modèles de mémoire (suite)

- x86-TSO
 - ▶ Modèle destiné au programmeur
 - ▶ Basé sur TSO
 - ▶ Étendu avec un concept de verrou global
 - ▶ Compatible avec les tests figurant dans la documentation des processeurs
 - ▶ Ne prétend pas correspondre à la structure interne des processeurs

Modèles de mémoire (suite 2)

Des exemples qui peuvent être exécutés totalement dans le modèle TSO (et x86-TSO), mais pas en SC

initialement : $x = y = 0$	
Processeur 1	Processeur 2
$store(p_1, x, 1) (s_1)$ $load(p_1, y, 0) (l_1)$	$store(p_2, y, 1) (s_2)$ $load(p_2, x, 0) (l_2)$

$store(p, m, v)$
 $p : [m] \leftarrow v$

$load(p, m, v)$
 $p : ([m] = v)$

Des entrelacements possibles :

SC		TSO		
s_1	s_1	s_1	s_2	s_1
l_1	s_2	s_2	l_2	l_1
s_2	l_1 bloque	l_1	s_1	s_2
l_2 bloque		l_2	l_1	l_2 bloque

Modèles de mémoire (suite 3)

Des exemples qui peuvent être exécutés totalement dans le modèle TSO (et x86-TSO), mais pas en SC

initialement : $x = y = 0$	
Processeur 1	Processeur 2
$store(p_1, x, 1) (s_1)$	$store(p_2, y, 1) (s_2)$
$load(p_1, x, 1) (l_1)$	$load(p_2, y, 1) (l_3)$
$load(p_1, y, 0) (l_2)$	$load(p_2, x, 0) (l_4)$

Des entrelacements possibles :

SC		TSO		
s_1	s_1	s_1	s_2	s_1
l_1	s_2	s_2	l_3	l_1
l_2	l_1	l_1	s_1	l_2
s_2	l_3	l_2	l_1	s_2
l_3	l_2 bloqué	l_3	l_2	l_3
l_4 bloqué		l_4	l_4	l_4 bloqué

SC

Définition formelle

Relations d'ordre :

- ▶ ordre du programme ($<_p$) (partiel, ordre par processeur)
- ▶ ordre de la mémoire ($<_m$) (global)

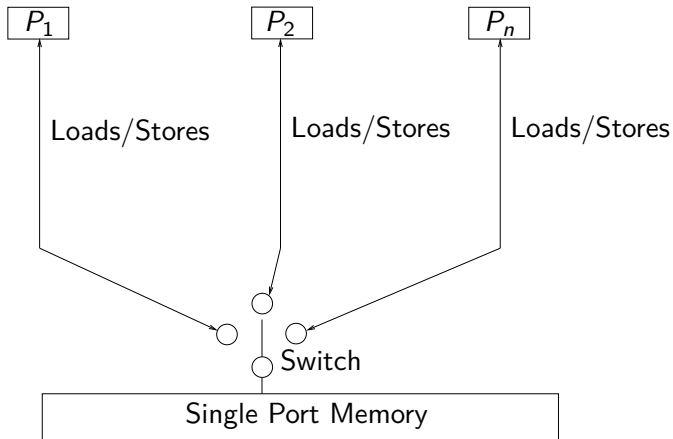
Pour correspondre au modèle SC, une exécution doit satisfaire les conditions suivantes :

- ▶ $\forall op_i, op_j : op_i <_p op_j \Rightarrow op_i <_m op_j$

où op_x représente une opération *store* ou *load*. Le résultat d'un *load* est celui qui est compatible avec l'ordre de la mémoire.

SC (suite)

Définition opérationnelle



SC (suite 2)

Ordres de la mémoire, premier exemple

initialement : $x = y = 0$	
Processeur 1	Processeur 2
$store(p_1, x, 1) (s_1)$	$store(p_2, y, 1) (s_2)$
$load(p_1, y, 0) (l_1)$	$load(p_2, x, 0) (l_2)$

où $s_1 <_p l_1$ et $s_2 <_p l_2$.

Les ordres de la mémoire possibles sont tous les entrelacements respectant les conditions $s_1 <_m l_1$ et $s_2 <_m l_2$.

Exemple : $s_1 <_m l_1 <_m s_2 <_m l_2$

TSO

Définition formelle

Pour correspondre au modèle TSO, une exécution doit satisfaire les conditions suivantes :

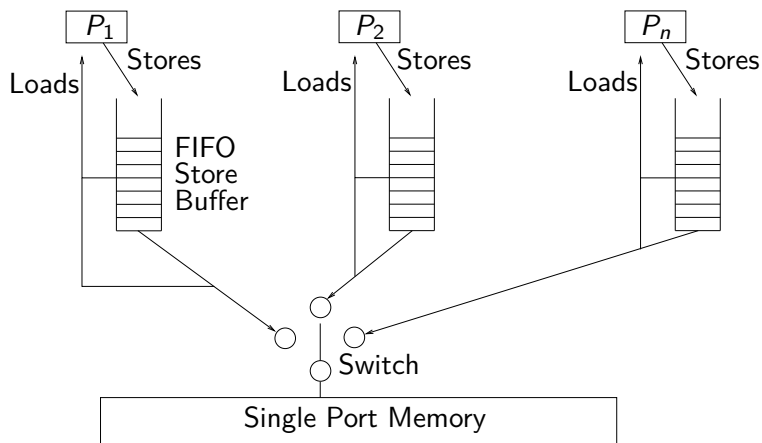
1. $\forall l_a, l_b : l_a <_p l_b \Rightarrow l_a <_m l_b$
2. $\forall s_a, s_b : s_a <_p s_b \Rightarrow s_a <_m s_b$
3. $\forall l, s : l <_p s \Rightarrow l <_m s$
4. $val(l_a) = val(\max_{<_m} \{s_a \mid s_a <_m l_a \vee s_a <_p l_a\})$.

Notez que :

- ▶ les *stores* peuvent être retardés,
- ▶ mais les *loads* ont accès à la dernière valeur mémorisée localement.

TSO (suite)

Définition opérationnelle



Le transfert d'un *store* de la zone tampon vers la mémoire principale est appelé un *commit*.

TSO (suite 2)

Ordres de la mémoire, premier exemple

initialement : $x = y = 0$	
Processeur 1	Processeur 2
$store(p_1, x, 1) (s_1)$	$store(p_2, y, 1) (s_2)$
$load(p_1, y, 0) (l_1)$	$load(p_2, x, 0) (l_2)$

où $s_1 <_p l_1$ et $s_2 <_p l_2$.

Les ordres de la mémoire possibles sont tous les entrelacements de s_1, l_1, s_2 et l_2 .

Exemple : $l_1 <_m s_2 <_m l_2 <_m s_1$ ou $l_1 <_m l_2 <_m s_1 <_m s_2$

TSO (suite 3)

Ordres de la mémoire, second exemple

initialement : $x = y = 0$	
Processeur 1	Processeur 2
$store(p_1, x, 1) (s_1)$	$store(p_2, y, 1) (s_2)$
$load(p_1, x, 1) (l_1)$	$load(p_2, y, 1) (l_3)$
$load(p_1, y, 0) (l_2)$	$load(p_2, x, 0) (l_4)$

où $s_1 <_p l_1$, $l_1 <_p l_2$, $s_2 <_p l_3$ et $l_3 <_p l_4$.

Les ordres de la mémoire possibles sont tous les entrelacements de s_1, l_1, l_2, s_2, l_3 et l_4 satisfaisant $l_1 <_m l_2$ et $l_3 <_m l_4$

Exemple : $l_1 <_m l_2 <_m l_3 <_m l_4 <_m s_1 <_m s_2$

TSO (suite 4)

Ordres de la mémoire, second exemple - Détails

Séquence d'opérations	Ordre mémoire
$store(p_1, x, 1)$ (s_1)	-
$load(p_1, x, 1)$ (l_1)	l_1
$load(p_1, y, 0)$ (l_2)	$l_1 <_m l_2$
$store(p_2, y, 1)$ (s_2)	$l_1 <_m l_2$
$load(p_2, y, 1)$ (l_3)	$l_1 <_m l_2 <_m l_3$
$load(p_2, x, 0)$ (l_4)	$l_1 <_m l_2 <_m l_3 <_m l_4$
— ($commit(s_1)$)	$l_1 <_m l_2 <_m l_3 <_m l_4 <_m s_1$
— ($commit(s_2)$)	$l_1 <_m l_2 <_m l_3 <_m l_4 <_m s_1 <_m s_2$

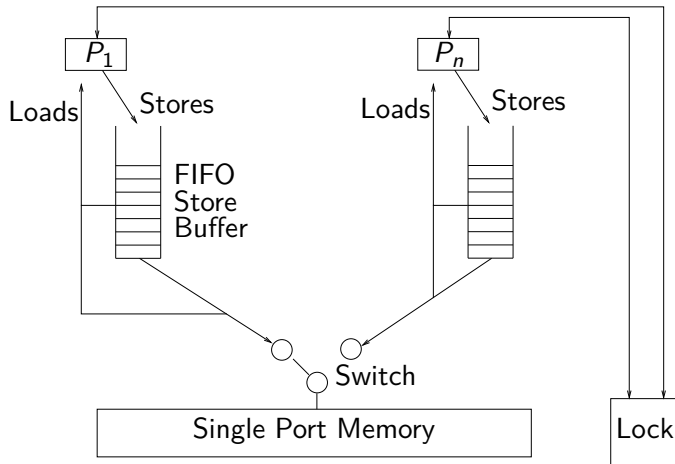
x86-TSO

Définition formelle

- ▶ Les contraintes d'ordre sur les *loads* et *stores* sont les mêmes qu'en TSO.
- ▶ Opérations étendues :
 - mfence(p)* bloque le processeur p jusqu'à ce que son buffer soit vide
 - lock(p)* si le verrou n'est pas déjà pris par un autre processeur, p prend le verrou et obtient un accès exclusif à la mémoire globale : les autres processeurs ne peuvent pas exécuter des opérations *commit* ou *load*
 - unlock(p)* p vide le buffer vers la mémoire globale et libère le verrou

x86-TSO

Vue opérationnelle



Autre modèles de mémoire

- ▶ Partial Store Order (PSO)
- ▶ Relaxed Memory Order (RMO)