Operational Semantics for TSO Model
TSO Machine Model

CPU

Write

Buffer

Read

Write-Back

Main Memory

CPU

Buffer
Basic Set Up

Thread subsystem remains same.

Each thread $i \in \text{Tid}$ is assigned to processor $i$.

Each processor has a local write buffer.

Introducing write buffer in memory subsystem.

Memory state $M, B$ where

- $M : \text{Loc} \mapsto \text{Val}$
- $B : \text{Tid} \mapsto (\text{Loc} \times \text{Val})$

Initial state: $M_0, B_0$ where

- $M_0 = \lambda x.0$ i.e. all memory locations are initialized to zero
- $B_0 = \lambda i.\epsilon$ i.e. all write buffers are empty
Write-to-Buffer A processor writes a value $v$ for a location $X$ at it’s write buffer. The location, value pair is enqueued at the write buffer.

$$\ell = W(X, v)$$

$$M, B \xrightarrow{i: \ell} M, B[i \mapsto \langle X, v \rangle].B(i)$$

Buffer-to-Memory The oldest value at a write buffer can silently be dequeued to memory.

$$B(i) = b.\langle X, v \rangle$$

$$M, B \xrightarrow{i: \epsilon} M[X \mapsto v], B[i \mapsto b]$$
Read-from-Buffer A processor can read value \( v \) from its write buffer for a location \( X \) if the write buffer has \( v \) as the newest write to \( X \) in it.

\[
\ell = R(X, v) \\
\exists b, b'. B(i) = b.\langle X, v \rangle.b' \land \not\exists \langle X, v' \rangle \in b \\
M, B \xrightarrow{i: \ell} M, B
\]

Read-from-Memory A processor can read value \( v \) for location \( X \) from memory if the local write buffer has no buffered writes to \( X \) and if the memory contain value \( v \) at location \( X \).

\[
\ell = R(X, v) \quad \not\exists \langle X, v \rangle \in B(i) \quad M(X) = v \\
M, B \xrightarrow{i: \ell} M, B
\]
Update A processor atomically reads from and write to the memory when its write buffer is empty.

\[
\ell = U(X, v, v') \quad B(i) = \epsilon \quad M(X) = v
\]

\[
M, B \xrightarrow{i: \ell} M[X \mapsto v'], B
\]

Fence A processor can perform a fence operation when its write buffer is empty.

\[
\ell = F \quad B(i) = \epsilon
\]

\[
M, B \xrightarrow{i: \ell} M, B
\]
**Linking Thread, Buffer & Memory in TSO**

**Thread-silent-step**

\[ P, S \xrightarrow{i: \epsilon} P', S' \]

\[ P, S, M, B \Rightarrow P', S', M, B \]

**Memory-silent-step**

\[ M, B \xrightarrow{i: \epsilon} M', B' \]

\[ P, S, M, B \Rightarrow P, S, M', B' \]

**Non-silent step**

\[ P, S \xrightarrow{i: \ell} P', S' \]

\[ M, B \xrightarrow{i: \ell} M', B' \]

\[ P, S, M, B \Rightarrow P', S', M', B' \]

**Consistent Execution**

Given a program \( P \), an execution configuration \( P', S', M', B' \) is consistent under TSO iff \( P, S_0, M_0, B_0 \xrightarrow{\text{TSO}^*} P', S', M', B' \) holds.

**Allowed Outcome**

Memory state \( M \) denotes an outcome of program \( P \) under TSO where \( P, S_0, M_0, B_0 \xrightarrow{\text{TSO}^*} \text{skip} \parallel \cdots \parallel \text{skip}, S', M, B_0 \) holds.
Pros

- Step-by-step construction of execution
- Incremental in nature
- Provides an algorithm for tool development

Cons

- Expose internals of machine architecture
- Complex rules for more complicated architectures e.g. ARM