6.823 Computer System Architecture

Store Sets

http://csg.csail.mit.edu/6.823/

We've seen in Lecture 12 how load/store buffers are essential in allowing out-of-order execution of loads and stores. Out-of-order processors can speculatively issue loads and stores out of program order, as long as their data dependences through memory are properly checked before they commit.

Unfortunately, naively speculating that all loads and stores can be executed out of order will hurt performance: when a load depends on an earlier store, which is relatively common, executing the load before the store with result in mis-speculation. *Memory dependence prediction* tackles this problem by predicting which loads and stores are dependent, even before their addresses are known, so that speculation happens only on likely-independent memory operations.

Stores sets [1] is a way to provide accurate memory dependence prediction. For a given load, its store set is defined as the set of stores for which the load has ever depended upon. A load is required to stall until all stores in its store set have issued.

The store sets design has two components: a predictor for memory dependences, and changes to the pipeline to efficiently enforce in-order execution of dependent stores and loads.

Store sets use a two-level predictor, shown in Figure 1. This structure is in the decode stage, so instructions check it in program order. First, the lower bits of the load/store PC are used to index into a **Store Set ID Table (SSIT)**. The SSIT tracks store sets using a common **store set identifier (SSID)** for loads and stores in the same set. The SSID is then used to query into the **Last Fetched Store Table (LFST)**, which maintains the instruction number (inum) of the last store in the store set that was fetched.

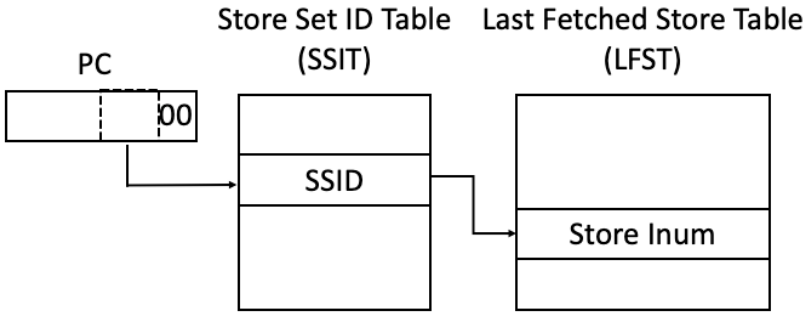

Figure 1. Store sets implementation. Lower bits of the PC are used to index into the Store Set ID Table to acquire the store set identifier (SSID). The SSID is used to index into the Last Fetched Store Table (LFST).

At the start of a program, all entries in the SSIT are invalid and loads and stores will be issued assuming no memory dependences. When the processor detects a memory order violation between a pair of load and store instructions, it populates the SSIT via the following rules:

- If neither the load nor the store has been assigned an SSID, one is allocated and assigned to both instructions.
- If the load(store) has a valid SSID, the store(load) is assigned that SSID.
- If both instructions have a valid SSID, the instruction that has the higher SSID will be assigned the SSID of the other instruction.

We now describe how loads and stores use their store sets to enforce memory ordering. When a load enters the decode stage, it looks up the SSIT. If a load has a valid SSID, it has a valid store set. It will use the SSID to query the LFST for the inum of the latest fetched store in the store set and inform the ROB to only issue the load after that store has issued. If the entry in the LFST is invalid, the load can be issued immediately, since it is not dependent on any recently fetched store.

A store also looks up in the SSIT when it enters the decode stage to read its SSID. If the store has a valid SSID, it will query the LFST. If the LFST entry is invalid, the store records its own inum so that a load with the same store set will wait until the store issues. If there is a valid entry for a store in the LFST, the store will be made dependent upon that preceding store in the ROB, and replaces the entry with its own inum since it is now the latest store in the store set. After the store issues, it accesses its LFST entry and invalidates it if it still refers to itself, resolving any dependency it has with a pending load with the same SSID.

Note that this implementation of store sets requires the load to only check whether the latest store in the store set has issued instead of checking for all stores. This has the benefit of simplifying the dependence logic between the load and its store set. The tradeoff is that stores in the same store set have to execute in order.

References

[1] George Z. Chrysos and Joel S. Emer. 1998. Memory dependence prediction using store sets. In *Proceedings of the 25th annual international symposium on Computer architecture* (*ISCA '98*). IEEE Computer Society, USA, 142–153.