**Date:** November 20, 2011 11:16 AM
**Topic:** cis501 hw4: paper review

Memory Dependence Prediction using Store Sets by George Chrysos and Joel Emer

Q1: When predicting memory dependencies, what is the cost of "over predicting" (falsely predicting a dependence)? What is the cost of "under predicting" (failing to predict an actual dependence)?

And: For “over predicting”, the issue scheduler will unnecessarily delay a load instruction due to the false dependency on a former store instruction.

For “under predicting”, a memory dependence violation will occur due to the undetected memory dependency, therefore
the load instruction and the following instructions have to be squashed.

Q2: How were the simulation results for the "perfect" memory dependence predictor generated? Why can't the hardware just use the same approach, thus achieving perfect memory dependence prediction?

And: The result is generated by assuming no memory-order violation penalty applies and no false dependency exists, or say the predictor can know the future exactly.

The hardware cannot know the memory address alias until the effective address is resolved.

Q3: Briefly describe the store sets approach to memory dependence prediction (just a paragraph).

And: A pair of load and store instructions which cause memory-order violation will be inserted into SSIT with one assigned SSID and each entry in SSIT,by using a store inum, will point to a corresponding store set in LFST which is updated when a store is issued or a new violation occurs, but if the load has already conflicted with another store, the load will depend on the new store and the new store will depend on the older store, in addition, at most two entries are allowed in a store set.
While executing instructions, a load instruction comes, if it is valid in SSIT, a non-empty store set pointed by SSID will cause issue scheduler to impose a dependence on the store, otherwise no action is needed; a store instruction comes, if it is valid in SSIT, it will update corresponding store set.

Q4: The stores sets approach works well for most benchmarks, but the authors found it doesn't work well on the benchmark "applu". What was the root cause of the poor performance on applu? What do they suggest the compiler might do to help? Although they leave it to "future work", how might the hardware's dependence predictor be modified to do better in such cases?

And: the root cause is the memory-order dependency crosses iterations in a loop, resulting in latter iteration has to wait for former iteration to finish. But this memory-order dependency only happens in certain loads and stores during the whole loop, and the false dependencies result in performance degradation.

They suggest fully unroll the loop in order to create different PCs for loads and stores in different iterations.

We may add iteration information into SSID by XOR the iteration index with existing SSID, so that in each iteration, the loads will have its exclusive store set.