CSE 420 Fall 2015
Project #4 Report

Problem 1)

A]

- Not all instructions could be executed in O3 way. Memory access instructions sharing the same memory address must be executed in order to keep the original program semantic. For this reason, O3 processors use memory dependence predictors. These are specialized units in charge of reducing, as much as possible, the number of loads and stores executed in-order. Good predictors aid to release all the ILP potential in O3 processors.
- The store sets predictor represents multiple potential dependences efficiently by grouping together all possible stores a load may dependent upon.
- It informs the IQ which memory instructions it predicts as ready to issue (in terms of memory ordering). In the Alpha models, memory operations have been atomic operations where the address calculation and memory access are bundled as one instruction. Because the effective addresses are not calculated separately, memory dependence prediction is necessary in order to give some idea of the order in which memory operations can execute.
- The memory dependence prediction to identify the stores upon which a load depends, and communicate that information to the instruction scheduler. The set of stores upon which each load has depended are known as the load's "store set". The processor can discover and use a load's store set to accurately predict the earliest time the load can safely execute.
- Store sets accurately predict memory dependencies in the context of large instruction window, superscalar machines, and allow for near-optimal performance compared to an instruction scheduler with perfect knowledge of memory dependencies.

B] Important Functions :-

1) **void StoreSet::insertStore (Addr store_PC, InstSeqNum store_seq_num, ThreadID tid)**

Function:

- Inserts a store into the store set predictor.
- Updates the LFST if the store has a valid SSID.

2) **void StoreSet::violation (Addr store_PC , Addr load_PC)**

Function: Records a memory ordering violation between the younger load and the older store.

3) **InstSeqNum StoreSet::checkInst (Addr PC)**

Function: Checks if the instruction with the given PC is dependent upon any store.

Returns: The sequence number of the store instruction this PC is dependent upon. Returns 0 if none.

Problem 2)

Please find the working code and stats files for all types of predictor in the zip file. The trace files are also included in zip file(vload_eliminated.txt & vload_NotEliminated.txt).

Part 1:

i. code changes are as follows –

store_set.hh :

We introduced one extra public member as ‘last_store’ of type integer. This variable will hold the value of sequence number of last store instruction before current load instruction.

store_set.cc :

1) In the function ‘insertStore’ , we assigned the value of store sequence number to the variable last_store.

2) In the ‘checkInst’ function, we added the 1st statement as ‘return last_store’ .

ii. Sim_ticks value given in Table 1 at the end.

Part 2:

i. Code changes as following –

Store_set.cc :

In the ‘checkInst’ function, we added the 1st statement as ‘return 0’ .

iii. Sim_ticks value given in Table 1 at the end.

Part 3 :

i. Code changes are as following –

SConscript file :

We introduced a new debug flag as ‘vload’ to note down the sequence numbers of both the violated loads and corresponding store instructions.

Mem_dep_unit_impl.hh :

We added the parameters store and load sequence numbers in the function call to function ‘violation’ (line 533).

store_set.hh :

1) We introduced one extra public member as ‘last_store’ of type integer. This variable will hold the value of sequence number of last store instruction before current load instruction.

2) We introduced 2 extra arguments in function declaration of function violation as ‘store_seqnum’ and ‘load_seqnum’ of datatype ‘InstSeqNum’.

store_set.cc :

1) In function ‘violation’ we added a Dprintf statement-

“ DPRINTF (vload , "vload , creating a new one : for load %#x , store %#x , store SequenceNum: %i , Load SequenceNum: %i\n" , load_PC , store_PC , store_seqnum , load_seqnum); ”.

This will print both PC and sequence numbers of load and store instructions.

2) In the function ‘insertStore’ , we assigned the value of store sequence number to the variable last_store.

3) In the ‘checkInst’ function –

We wrote an IF statement which compares the last store sequence number with all the violated store sequence numbers from the trace file.

If the condition in IF statement is true then we return ‘last_store’ variable.

ii. Sim_ticks value given in Table 1 at the end.
iii. Values given in Table 2 at the end.

Table 1: sim Ticks of all types of predictors :

Parameter	Original predictor	No speculation	Naive speculation	100% accurate
Sim_Ticks	**47591000**	**60490500**	**56279500**	**47591000**

Table 2: Statistics of Gem5 predictor and implemented 100% accuracy predictor :

Parameter	Gem5 predictor	100% accurate predictor
system.cpu.iew.lsq.thread0. memOrderViolation	**54**	**51**
system.cpu.iew. memOrderViolationEvents	**54**	**51**
system.cpu.memDep0. insertedLoads	**45267**	**45266**