International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Special Issue of ICRTET-2023, pp 248-252, June-2023

International Conference of Recent Trends in Engineering & Technology (ICRTET-2023)

Memory Dependence Predictors

[1]Simran Satish Kulkarni, [2]Soumil Krishnanand Heble, [3]Dr. Eric Rotenberg

[1,2,3]Department of Electrical & Computer Engineering North Carolina State University, Raleigh, USA

***Abstract* - This paper details the implementation of a novel Memory Dependence Predictor based on the relative distance between loads and their dependent stores in the dynamic program order. In presence of unknown store addresses blind speculation never stalls loads which are ready. This results in load violations since the loads may depend on those stores with unknown addresses. To prevent the violation penalty we employ the use of memory dependence predictors that train a predictor table to detect load-store dependency. In this project we will be implementing two kinds of memory dependence predictors namely, Sticky Bit and Store Vectors.**

I. INTRODUCTION

In presence of unknown store addresses blind speculation never stalls loads which are ready. This results in load violations since the loads may depend on those stores with unknown addresses. To prevent the violation penalty we employ the use of memory dependence predictors that train a predictor table to detect load-store dependency. In this project we will be implementing two kinds of memory dependence predictors namely, Sticky Bit and Store Vectors. All predictors usually have the following four stages or operations to be done,

- Initialization, After Reset
- Lookup/Prediction
- Scheduling
- Update

Key Outcomes:

- Implement a Sticky Bit predictor that just remembers whether a load vioated with any of its dependent stores.
- Find potential factors that limit the sticky bit predictor performance to be lower than oracle memory disambiguation.
- Implement the Store Vector Memory Dependence Predictor and a modified Store Vector Memory Dependence Predic- tor that uses the global branch history along with load PC to access the prediction table.
- Compare the performances of no speculation, blind speculation, oracle memory disambiguation, sticky bit predictor and the store vector predictor.

II. STICKY BIT MEMORY DEPENDENCE PREDICTOR

It is the most basic algorithm used to predict memory dependencies. It consists of a 1 bit predictor table indexed by the Load PCs. If the bit is set for a particular PC, then that indicates that the load is seen to be dependent on stores and therefore is made to stall and wait for all stores before it, to have known/valid addresses. Basically it's a 1 bit saturating counter that is never decremented.

Structures and Circuitry Required: The sticky bit im- plementation requires a prediction table that has sufficient read ports so that enough Load instructions can be processed at the dispatch stage. To throttle the Loads in presence of older stores with unknown addresses the existing mechanism of leveraged to use with the table prediction to stall the Loads. Also, a write port will be required at retire for training the predictor.

Initial State upon Reset: All entries of the prediction table are cleared to 0, which predicts that loads do not cause violations when allowed to speculatively exe- cute. Since the table storage requirements are small, they can be implemented with Registers or SRAM. We also employ a cyclic clear strategy which flash clears all entries of the table and resets them, every few mil- lion instructions. This is done to prevent incorrect train- ing of the table and also to prevent false dependencies.

Lookup (@ Dispatch): In the dispatch stage, loads access the prediction table to get a prediction and are dispatched into the load queue along with the prediction. A value of 1 means the load is seen to conflict with older store's and needs to stall and 0 - vice-versa.

Scheduling (@ LD/ST Unit): When loads issue, the load store unit consumes the prediction to throttle the issued load's progress if it is predicted to stall and there exist stores with unknown addresses older than the issued load. A stalled load is periodically tried to unstall by the load store unit until it is completes execution.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Special Issue of ICRTET-2023, pp 248-252, June-2023

International Conference of Recent Trends in Engineering & Technology (ICRTET-2023)

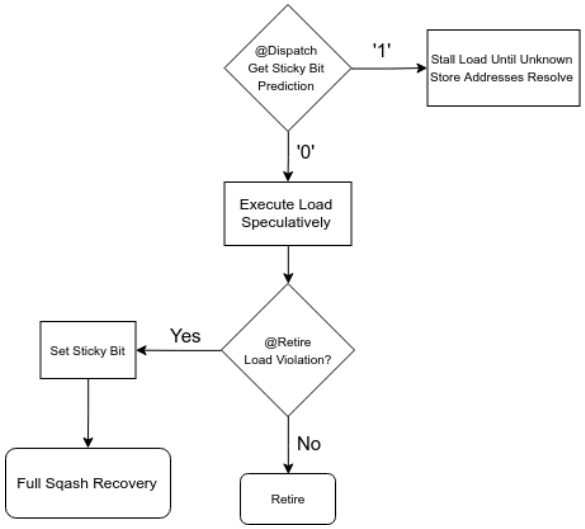

Figure 1: Sticky Bit Predictor Flowchart

Update (@ Retire): For loads that are predicted to not stall and proceed further in presence of unknown store addresses, there is a possibility of a load violation occurring. In this case the store after finishing its execution search younger loads and mark them for violation if they executed early and have the wrong value. When this load reaches the head of the Active List, it triggers a complete squash and the predictor table entry corresponding to the load's PC is set to 1.

III. STORE VECTOR MEMORY DEPENDENCE PREDICTOR

As opposed to Sticky Bit where a store merely sets the bit to indicate dependency, Store Vectors use distance between a loads most recent store and the violating store to specify precisely which stores cause a violation. This is beneficial because it helps avoid false dependencies and stalls only for true dependencies. There are 2 essential structures for implementing Store Vectors: Store Vector Table and Load Scheduling Matrix.

Structures and Circuitry Required: The store vector predictor works with the load and store queues to perform memory dependence prediction and easing scheduling of the loads. The sticky bit implementation requires a prediction table that has sufficient read ports so that enough Load instructions can be processed at the dispatch stage. Also, each table entry is a distance vector that needs to be aligned to the current state of the store queue. Which suggests the requirement of a rotate logic and bitwise AND hardware.

Initial State upon Reset: This can be seen as a sticky bit predictor where every table entry allots atmost one sticky bit per store queue entry. So, similar implementation can be used as the sticky bit and the same flash clear algorithm is used.

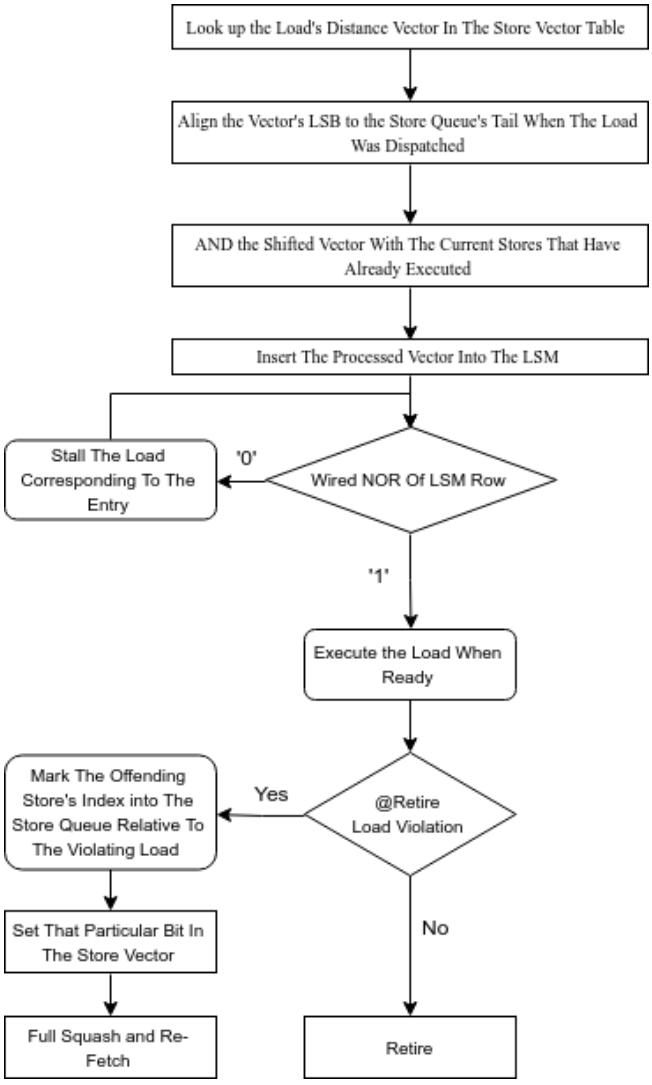

Figure 2: Store Vector Flowchart

Lookup (@ Dispatch): In the dispatch stage loads ac- cess the prediction table (SVT - Store Vector Table) to get the load's corresponding store vector. Where in the LSB of the store vector is the prediction for the most recent store as seen by the load. This vector needs to be rotated to match the most recent store in the store queue when the load was inserted. After a rotate, a bitwise AND of the current status of executed loads is performed to avoid deadlock. After this the store vec- tor is inserted in to the LSM (Load Scheduling Ma- trix) at a row that corresponds to an entry in the load queue.

Scheduling (@ LD/ST Unit): The LSM is an ingenious structure used to ease scheduling of loads. It is basically a matrix whose rows equal the load queue size and

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Special Issue of ICRTET-2023, pp 248-252, June-2023

International Conference of Recent Trends in Engineering & Technology (ICRTET-2023)

columns equal to the store queue size. After processing the load's store vector its inserted into the LSM for scheduling. As stores execute they flash clear the corresponding columns of the LSM. The wired NOR of a particular row tells whether any predicted dependent stores are present in the store queue which have not yet executed. Once this wired NOR goes 1, the load becomes a potential candidate for execution. A priority encoder can be used to select the oldest ready load.

Update (@ Retire): For loads that are allowed to execute (by the virtue of the wired NOR value being 1) can still be speculative because an associative search of the store queue is not done to reduce complexity. So, there is a possibility of a load violation occurring. In this case the store after finishing their execution search younger loads and mark them for violation if they executed early and have the wrong value. Meanwhile, the store also deposits its store queue index to aid in predictor training. When this load reaches the head of the Active List, it triggers a complete squash and the offending store's deposited index along with the load's load queue index is used to set the corresponding store vector bit to 1. Some shift logic may be required and an extra RW port at retire.

Modified Store Vector Memory Dependence Predictor

The implementation and micro architecture design of this variant is the same as that of Store Vectors above, except the indexing mechanism for the Store Vector Table. Instead of using the PC of a load, we create an index which is a combination (XOR) of the load PC and a few recent bits of the the global branch history. The idea behind augmenting the index with GBHR (Global Branch History Register) is that branches may skew the dependence vectors in terms of store positions. Thus by using the GBHR we can predict dependencies more accurately with respect to the branches that precede it.

Simulator Configurations

The --mdpred option was added to the parser to configure the memory dependence predictor easily. Following are the available options,

--mdpred=<type>,<entries>,<clrcycle>,<SVentr>

<type>: 0-No Predictor (Uses Oracle/Spec Disambig Options), 1-Sticky Bit Predictor, 2-Store Vector Predictor, 3-Modified Store Vector Predictor

<entries>: Number of Entries in the Table (0 - For Full Address Space)

<clrcycle>: Number of Retired Instructions After Which Table is Cleared (0 - For No Clear)

<SV entr>: Number of Entries in the Store Vector (Option Only Used if <type> = 2 or 3 and Truncated if Length > Store Queue Length) (0 - For The Same Length as Store Queue)

The 456.hmmer test.74.0.22.gz and 473.astar test.76.0.22.gz benchmarks were used for analysis.

Parameter	Size
Fetch Queue	64
Branch Checkpoints	32
Active List	256
Issue Queue	64
Issue Queue Partitions	4
Fetch/Decode/Retire Width	4
Issue Width	8
Load/Store Queue	32
Memory	2 GB

Table 1: Default Superscalar Parameters

Predictor Type	Entries	Clear Frequency	Store Vector Length	BHR Bits
Sticky Bit	2048	1M Instructions	-	-
Store Vector			32	-
Modified Store Vector			32	4

Table 2: Memory Dependence Predictor Configuration

Figure #	Changed Parameter	Value
3 - Left	-	-
3 - Right	-	-
4 - Left	Load/Store Size	128
4 - Right	Load/Store Size	128
5 - Left	Load/Store Size	128
	Table Entries	512 to 8192 in Powers of 2
	Clear Frequency	Don't Clear, 1M and 10M Instructions
5 - Right	Load/Store Size	512 to 8192 in Powers of 2
	Table Entries	128
	Clear Frequency	Don't Clear, 1M and 10M Instructions

Table 3: Table of Graphs

IV. RESULTS AND ANALYSIS

Primary Results

This graph shows the comparison of the IPC of various memory dependence prediction algorithms for the hmmer benchmark with perfect as well as real branch prediction. We can see that as we switch from perfect to real branch predictor we can see that the huge performance differences we see with perfect branch prediction reduces drastically which is an indication that memory disambiguation is not a likely bottleneck if the branch predictor performs poorly. The same

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Special Issue of ICRTET-2023, pp 248-252, June-2023

International Conference of Recent Trends in Engineering & Technology (ICRTET-2023)

cannot be said for the astar benchmark as the performance penalty from perfect to real branch prediction is not much.

In hmmer benchmark, the sticky bit predictor comes really close to oracle performance where as the store vector predictors perform similar to speculative disambiguation, the reason for this behaviour is unknown.

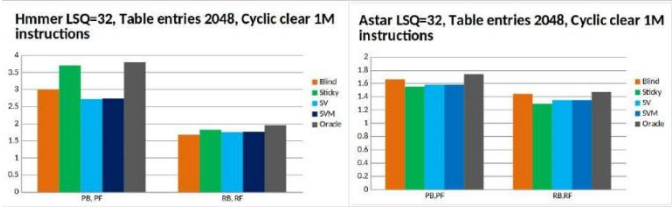

Figure 3: Baseline Benchmark Comparison

Sticky Bit Predictor Results

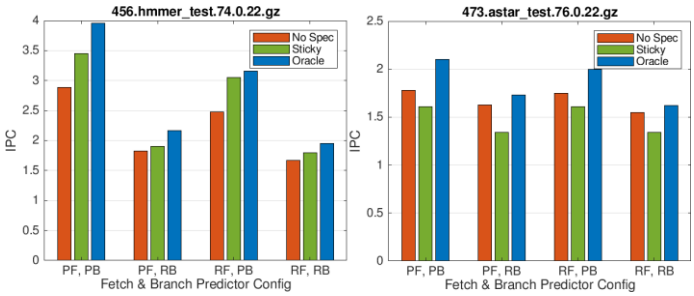

Figure 4: Sticky Bit Performance Comparison

We simulated sticky bit predictor with hmmer and astar benchmarks and we see that regardless of the branch predictor and fetch configuration, in hmmer, sticky performs really well and approaches oracle performance in real branch predictor case. However in astar it performs poorly compared to speculative disambiguation and the reason for this is outlined in the related work report.

Sticky Bit Predic-tor Sensitivity Analysis

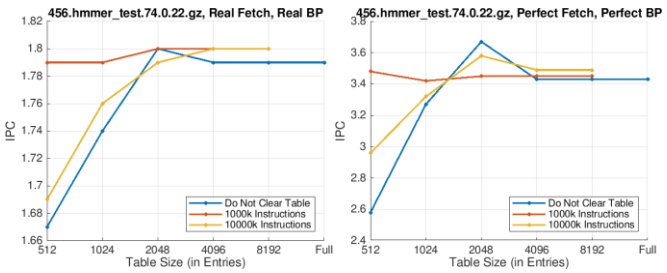

Figure 5: Sticky it Sensitivity Analysis

We performed sensitivity analysis on the sticky bit memory dependence predictor design to select the most performing configuration. From the graphs we can see that the performance gives diminishing returns after a table size of 2048 entries. Also, a table crear after every 1M instructions provides the best performance.

Simulator Accomplishments

All three of the proposed memory dependence predictors (sticky bit, store vectors and modified store vectors) have been implemented and been tested successfully.

Major Microarchitecture Feature and/or Major Simulator Config	Perfect Branch Prediction			Real Branch Prediction		
	astar (#inst.)	hmmer (#inst.)	bzip (#inst.)	astar (#inst.)	hmmer (#inst.)	bzip (#inst.)
Sticky Bit Predictor	100M	100M	100M	100M	100M	100M
Store Vector Memory Dependence Predictor	100M	100M	100M	100M	100M	100M
Modified Store Vector Memory Dependence Predictor	100M	100M	100M	100M	100M	100M

V. FUTURE WORK

We are not seeing much performance increase with the store vec- tor memory dependence predictor as expected from the literature survey. The immediate future work will be to analyze the reason for the mediocre performance. We cannot analyze the impact of aliasing and false dependency with the store vector predictor as easily as we did with the memory dependence predictor. So finding an efficient way to insert some performance counters for the store vector predictor is the next logical step. Also, the store vector size is limited to 64 entries due to variable size limitations. Larger sizes can be handled by splitting the vector across multiple variables.

REFERENCES

[1] Speculation Techniques for Improving Load Related Instruction Scheduling, Adi Yoaz, Mattan Erez, Ronny Ronen, and Stephan Jourdan.

[2] Memory Dependence Prediction using Store Sets, George Z. Chrysos and Joel S. Emer.

[3] Memory Dependence Prediction, by Andreas Ioannis Moshovos.

[4] Store Vectors for Scalable Memory Dependence Prediction and Scheduling, Samantika Subramaniam Gabriel H. Loh, Georgia Institute of Technology.

[5] R. E. Kessler. The Alpha 21264 Microprocessor. IEEE Micro Magazine, 19(2):24.36, March.April 1999.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Special Issue of ICRTET-2023, pp 248-252, June-2023

International Conference of Recent Trends in Engineering & Technology (ICRTET-2023)

Citation of this Article:

Simran Satish Kulkarni, Soumil Krishnanand Heble, Dr. Eric Rotenberg, "Memory Dependence Predictors" in proceeding of International Conference of Recent Trends in Engineering & Technology ICRTET - 2023, Organized by SCOE, Sudumbare, Pune, India, Published in IRJIET, Volume 7, Special issue of ICRTET-2023, pp 248-252, June 2023.

****

