Load Value Prediction using Two-Hop Reference Address Renaming

Toshinori Sato
Toshiba Microelectronics Engineering Laboratory
580-1, Horikawa-Cho, Saiwai-Ku, Kawasaki 210-8520, Japan
toshinori.sato@toshiba.co.jp

Abstract

In this paper, we present an alternative implementation of load value prediction. Load values are proposed to be predicted using store information. A pair of a load and a store instructions referring a same memory location is linked and the stored value is forwarded to the load instruction. For the data forwarding, store-indexed value table (SIVT) *and* load-indexed store table (LIST) *are proposed. By indexing the LIST with a load instruction address, the load instruction address is translated to a store instruction address, and the SIVT indexed by the translated address supplies a data value. In order to link the store and load instructions,* data-indexed store table (DIST) *is proposed. Preliminary results using a cycle-by-cycle simulator shows that the proposed predictor is useful for data dependence speculation.*
Keywords: instruction level parallelism, out-of-order execution, dynamic speculation of data dependence, load value prediction, reference address renaming

1 Introduction

Dependences are obstacles disturbing processor performance. They include control, name, and data dependences. The control dependence attracts many researchers and is attacked by the techniques such as branch prediction and speculative execution. The name dependence is caused by resource shortage and can be eliminated using register renaming. However, the data dependence can not be removed by such techniques, as it is called *true dependence*. Hence, the data dependence is a serious obstacle limiting instruction level parallelism (ILP). There are two types of the data dependences. One is the dependence through registers, and the other is that through memory. In this paper, we focus on the latter one. The dependence through memory is caused by ambiguous memory aliasing. This is because any data addresses can not be determined until the program is executed. Thus, any succeeding load and store instructions can not be executed until a preceding store instruction is resolved. This is the ambiguous memory dependence disturbing the exploitation of the ILP.

Recently, Lipasti el al.[4] have proposed a load value prediction scheme. Due to removal of the data address calculation, the scheme is free from the ambiguous memory dependence problem. However, based on the scheme a load instruction uses the same value which is read from memory at the last time when the same instruction is emerged in the future, and thus the scheme does not work effectively if the value changes frequently. In this paper, we propose an alternative implementation of the load value prediction. Our proposed scheme is so simple that its implementation is practical. Load values are proposed to be predicted using store information. A pair of a load and a store instructions referring a same memory location is linked and the stored value is forwarded to the load instruction. In order to forward the data, *store-indexed value table (SIVT)* and *load-indexed store table (LIST)* are proposed. By indexing the LIST with a load instruction address, the load instruction address is translated to a store instruction address, and the SIVT indexed by the translated address supplies a data value. In order to link the store and load instructions, *data-indexed store table (DIST)* is proposed. Using the predicted value, instructions following the load instruction are executed speculatively.

The organization of the rest of this paper is as follows. A load value prediction mechanism is explained in Section 2. In Section 3, the data speculation is evaluated. Finally, our conclusions are presented in Section 4.

2 Load Value Prediction

In this section, we propose a load value prediction mechanism using two-hop reference address renaming[8]. First, we explain the concept of two-hop reference address renaming. Next, we describe three tables which are utilized to implement the proposed load value prediction scheme. They are *store-indexed value table* (SIVT), *load-indexed store table* (LIST), and *data-indexed store table* (DIST). The SIVT and LIST cooperate with each other and forward a stored data value to a load instruction. The DIST links a pair of a load and a store instructions. And next, the load value prediction mechanism is explained.

2.1 Two-Hop Reference Address Renaming

The load value prediction scheme studied by Lipasti el al.[4] uses only load instruction information. The value prediction accuracy will be improved if the store instruction information is available, since the data value read by a load instruction is stored by a store instruction. It is said that significant amount of load instructions suffering from ambiguous memory aliasing are dependent upon the same store instructions[5]. If the store instruction renames the data address into a tag and the load instruction refers the data using the same tag, it becomes possible to forward the data from the store instruction to the load instruction before the load instruction calculates the data address. However, it is difficult to deliver the tag defined by the store instruction into the load instruction. Therefore, we propose that the store and load instructions define their tags independently. If the tag defined by the load instruction is translated into the one defined by the store instruction, the store and load instruction use the same tag effectively. This scheme is shown in Figure 1. The `tag(l)` defined by the load instruction is translated into the `tag(s)`, which is defined by the store instruction. The load instruction refers the temporal memory with the translated tag. We call this scheme two-hop reference address renaming.

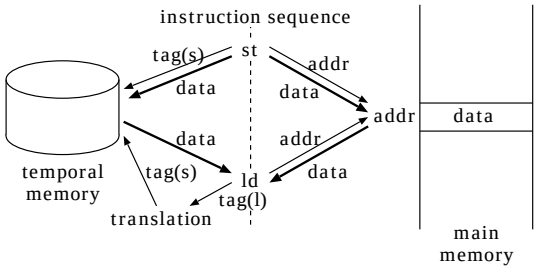

Figure 1: Two-Hop Reference Address Renaming

If an instruction address is used as the tag, a data address is renamed as follows. When a store instruction is executed, the data value is stored also in a temporal memory with the store instruction address used as a tag. A load instruction translates the load instruction address into the store

instruction address and accesses the temporal memory with the translated address. And thus, the load instruction can obtain the data value from the temporal memory before the data address is calculated.

2.2 Store-Indexed Value Table

The SIVT is indexed by a store instruction address, and provides the data value written by the store instruction. Each entry of the SIVT holds a data value and is indexed by a store instruction address. When the SIVT is accessed with a store instruction address, it supplies a data value which the store instruction wrote at the last time. Thus, the SIVT renames a data address to a store instruction address.

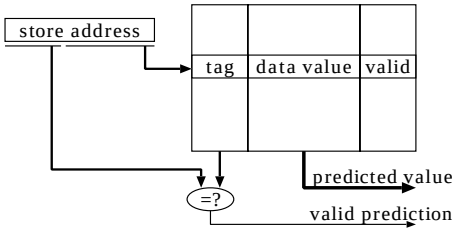

Figure 2: Store-Indexed Value Table

The SIVT has a similar structure with cache memory. Figure 2 shows the direct-mapped SIVT. Each entry consists of a tag address field (`tag`), a data value field (`data_value`), and a valid bit (`valid`).

2.3 Load-Indexed Store Table

The LIST translates a load instruction address into the address of the store instruction which refers the same memory location accessed by the load instruction. Each entry of the LIST keeps a store instruction address and is indexed by the load instruction address. When a load instruction refers the LIST, it obtains a store instruction address. This store instruction wrote the data which the load instruction will read. Hence, the LIST renames a store instruction address to a load instruction address.

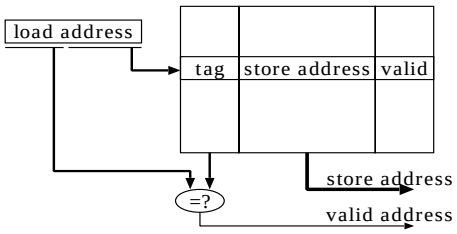

Figure 3: Load-Indexed Store Table

The LIST has a similar structure with cache memory. Figure 3 shows the direct-mapped LIST. Each entry consists of a tag address field (`tag`), a store instruction address field (`store_address`), and a valid bit (`valid`).

2.4 Data-Indexed Store Table

The DIST corresponds a load instruction with a store instruction whose reference address is same to the one which the load instruction refers. It is indexed by a data address and supplies the store instruction address. When a load instruction refers the DIST with the data address, it obtains an address of a store instruction which wrote the data at the last time.

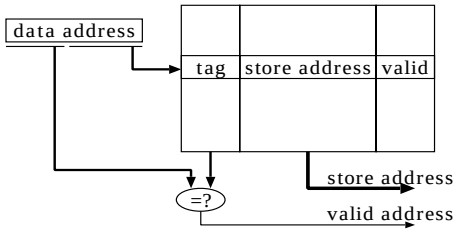

Figure 4: Data-Indexed Store Table

The DIST also has a similar structure with cache memory. Figure 4 shows the direct-mapped DIST. Each entry consists of a tag address field (`tag`), a store instruction address field (`store_address`), and a valid bit (`valid`).

2.5 Load Value Prediction Mechanism

We explain a load value prediction mechanism using the simple pipeline described in Figure 5. A data value has been predicted when an instruction is being issued into instruction window. When the instruction is fetched, the LIST is accessed with the instruction address and provides a store instruction address. The store instruction address is used to refer the SIVT at the decode stage. The SIVT provides a data value stored by the store instruction. When the instruction accessing the LIST is a load instruction, the data value is used as the predicted one and its following instructions are executed speculatively using the predicted value. Note that the LIST and SIVT are accessed at different stages, and hence there are not any impact of serialized lookups of the LIST and SIVT on processor cycle time.

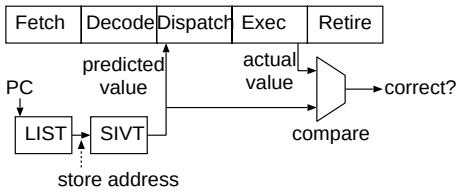

Figure 5: Pipeline Diagram

A confidence mechanism[2] decides if the speculation is initiated or not. The confidence mechanism consists of two-bit counters, each of which corresponds with an entry of the LIST. The 2-bit counter is incremented by 1 when a value prediction succeeds, and decremented by 1 when it fails. When a load instruction predicts the load data value, it refers the confidence mechanism and decides if the prediction is useful or not. When the counter indicates more than 2, the prediction is considered as useful. The 2-bit counter is same as that used in the branch target buffer. Figure 6 shows the state transition of the counter.

The predicted value is kept in instruction window. When the load instruction reads an actual data value from memory, the predicted value must be compared with the actual one. If they match, the prediction succeeds and the speculation is useful. Otherwise, the prediction fails and the processor state must be corrected. Only instructions dependent upon the misspeculated load instruction are selectively invalidated and reissued[9]. This process is easily implemented using the register update unit (RUU)[10].

The SIVT, DIST, and LIST are updated as follows. The SIVT is updated when a store instruction obtains a data value which will be stored in memory even if the data address has not been calculated. Till this time, the instruction knows the store instruction address and the data value. Therefore, it is possible to keep the value in the SIVT indexed by the store

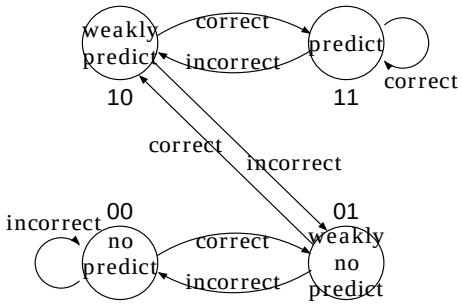

Figure 6: 2-bit Counter for Confidence

instruction address. The DIST is updated when the store instruction has calculated the data address. Till this time, the instruction knows the store instruction address and the data address. Hence, the DIST indexed by the data address can hold the store instruction address. The updating process of the LIST is shown in Figure 7. When a load instruction finishes, the DIST is referred. Till this time, the data address is calculated and it is possible to access the DIST. The DIST provides a store instruction address. Therefore, it is possible to keep the provided store instruction address in the LIST indexed by the load instruction address.

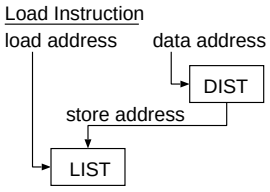

Figure 7: Updating Scheme for LIST

From the above explanation, the load data value prediction is performed. The DIST links a load and a store instructions which refer the same memory location, and the LIST keeps the link. When the load instruction is executed, the LIST supplies the store instruction address which is used to access the SIVT. The load instruction can obtain the data value provided by the SIVT before it calculates the data address. In summary, in order to predict the load value without calculating the data address, the two-hop reference address renaming from the data address into the load instruction address is performed.

2.6 Related Work

Speculative memory cloaking[6] and memory renaming[11] are similar with the two-hop reference address renaming described in this paper, however all these works are independently studied in parallel. One of the superior points of our work over the other two is the simplicity of the implementation scheme provided in this paper. The scheme does not require any complicated operations such as associative search, and thus is easy to be implemented without severe impact on processor cycle time. This is enabled by distributing three operations into the individual tables.

3 Evaluation

This section presents experimental results. First, we describe the evaluation methodology by explaining a processor model and benchmark programs. And next, we evaluate the usefulness of the load value prediction.

3.1 Methodology

We evaluated the effect of the proposed mechanism by using SimpleScalar tool set[1]. The SimpleScalar architecture is based on MIPS architecture, and a fully execution-driven and cycle-by-cycle simulator is executed on a SPARCstation. The baseline model is an out-of-order execution processor based on the RUU[10]. Following the discussion explained in [3], we decide the configuration of the baseline processor summarized in Table 1. The SIVT, LIST, and DIST are assumed to be direct-mapped and to have 4096 entries respectively.

The SPEC95 CINT benchmark suite is used for this study. The reason why we evaluate only integer programs is as follows. Since data operated in floating-point programs tend to have regular structures due to array references, it is possible to detect memory dependences using sophisticated compilers. Hence, we are interested in integer programs whose data reference is irregular. The `test` input files which are provided by SPEC are used. We used the object files provided by University of Wisconsin Madison[1]. Each program was executed to completion or for the first 100 million instructions.

3.2 Experimental Results

The left side of Table 2 indicates the accuracy of the load value prediction. We define the prediction accuracy as the cases when the prediction succeeds over the cases when the prediction is initiated. The load instructions which are not predicted are not counted. Table 2 shows that the prediction accuracy is relatively high. It is 79.0% on average with a maximum of 93.1%.

Table 2: Value Prediction Accuracy and Hit Rate

program	(%)accuracy	(%)hit rate	
		LIST	SIVT
099.go	49.39	84.6	96.8
124.m88ksim	91.04	63.3	96.7
126.gcc	77.84	61.0	97.6
129.compress	89.03	60.8	99.9
130.li	71.47	86.0	95.6
132.ijpeg	90.33	77.3	99.7
134.perl	69.82	81.5	96.3
147.vortex	93.10	70.5	95.9
average	79.00	73.1	97.3

The right side of Table 2 indicates the hit rates of the LIST and SIVT. We define the hit rate of the LIST as the cases when the LIST can provide store instruction addresses over the number of all load instructions, and define the hit rate of the SIVT as the cases when the data values are obtained over the cases when the LIST can provide store instruction addresses. As can be seen in Table 2, the average hit rates of the LIST and SIVT are 73.1% and 97.3%, respectively. Straightforwardly, the value prediction accuracy may be regarded as the LIST hit rate by the SIVT hit rate: i.e. about 70%. However, the prediction accuracy is slightly higher than the expectation. This is due to the confidence mechanism. Since the speculation is initiated only when a prediction is decided to be useful by the confidence mechanism, the prediction accuracy is improved.

Figure 8 shows the rate that data value prediction is performed. Each bar is divided into three parts. The bottom part (black) indicates the percentage of the load instruction whose data value is correctly predicted. The middle part (white) indicates the percentage that is mispredicted. And the top part (gray) indicates the percentage that is not predicted by the proposed mechanism. From Figure 8, it is found that the harmful predictions are removed by the confidence mechanism effectively. While the prediction accuracy is high, the percentage of load instructions correctly predicted is relatively small and is 49.7% on average. This is due to the much lower hit rate of the LIST than we expected. The hit rate might be improved by optimizing the LIST structure.

Table 1: Baseline Processor Configuration

Fetch Width	8 instructions
Branch Predictor	512 entry 2way set-associative BTB, gshare scheme, 12-bit BHR, 4096 entry PHT, speculatively updated in ID stage, 8 entry return address stack, 3 cycle miss penalty
Insn. Windows	64 entry instruction queue, 8 entry load/store queue
Issue Width	8 instructions
Commit Width	8 instructions
Functional Units	5 iALU's, 1 iMUL/DIV, 4 Ld/St, 2 fALU's, 2 fMULs, 2 fDIV/SQRT's
Latency(total/issue)	iALU 1/1, iMUL 3/1, iDIV 35/35, Ld/St 2/1, fADD 2/1, fMUL 3/1, fDIV/SQRT 6/6
Register Files	32 32-bit fixed point registers, 32 32-bit floating point registers
Insn. Cache	64K 4way set-associative, 32 byte blocks, 4-port, 6 cycle miss penalty
Data Cache	64K 4way set-associative, 32 byte blocks, 4-port, write-back, non-blocking load, hit under miss, 6 cycle miss penalty
L2 Cache	unified, 256K 4way set-associative, 64 byte blocks, 32 cycle miss penalty

We expect that larger capacity and higher associativity increase the LIST hit rate due to the reduction of capacity and conflict misses like cache memories.

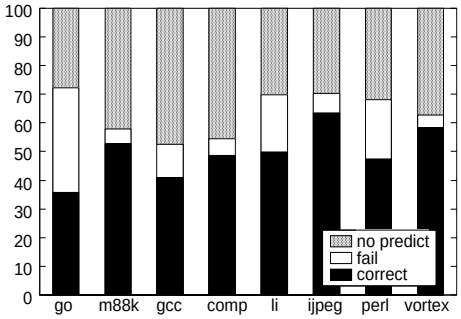

Figure 8: (%)Performing Rate of Value Prediction

Figure 9 shows the performance improvement rate when the proposed mechanism is used. We define the performance improvement rate as the increased IPC over the IPC of the baseline model. The processor performance is improved by 3.4% on average with a maximum of 9.4%. This confirms that the proposed load value prediction mechanism is useful for improving the processor performance.

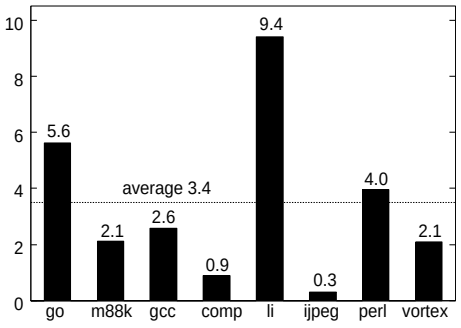

Figure 9: (%)Processor Performance Improvement

4 Concluding Remarks

We have investigated the speculative execution of data dependences. Using the predicted load values, the data dependences are speculatively resolved. We have presented an alternative implementation of load value prediction mechanism. Load values are proposed to be predicted using store information. A pair of a load and a store instructions referring a same memory location is linked and the stored value is forwarded to the load instruction. A data address is renamed to a store instruction address by the SIVT, and the renamed store instruction address is renamed again to a load instruction address by the LIST. We call this scheme two-hop reference address renaming. In order to link the store and load instructions, we have proposed the DIST. Our proposed mechanism consisting of the SIVT, LIST, and DIST is so simple that its implementation is practical.

We have evaluated the value prediction mechanism using a cycle-by-cycle simulator. The value prediction accuracy is relatively high and is 79.0% on average with a maximum of 93.1%. The processor performance is improved by 3.4% on average with a maximum of 9.4%. These results confirm that the proposed load value prediction mechanism is useful for data dependence speculation.

Future study dealing with the load value prediction will investigate an extension of the LIST. When the SIVT misses, the load data value can not be predicted even if the LIST hits. If the LIST has the ability to predict the data address[7], the load instruction predicts the data address and is speculatively executed even when the SIVT misses.

Acknowledgment

The author is grateful to Dr. Mitsuo Saito and Dr. Shigeru Tanaka for their continuous encouragements.

References

[1] D.Burger, T.M.Austin, "The SimpleScalar tool set, version 2.0", ACM SIGARCH Computer Architecture News, vol.25, no.3, 1997.

[2] E.Jacobsen, E.Rotenberg, J.E.Smith, "Assigning confidence to conditional branch predictions", Proc. of MICRO-29, 1996.

[3] S.Jourdan, P.Sainrat, D.Litaize, "Exploring configuration of functional units in an out-of-order superscalar processor", Proc. of ISCA-22, 1995.

[4] M.H.Lipasti, C.B.Wilkerson, J.P.Shen, "Value locality and load value prediction", Proc. of ASPLOS-VII, 1996.

[5] A.Moshovos, S.Breach, T.Vijakumar, G.Sohi, "Dynamic speculation and synchronization of data dependences", Proc. of ISCA-24, 1997.

[6] A.I.Moshovos, G.S.Sohi, "Streamlining inter-operation memory communication via data dependence prediction", Proc. of MICRO-30, 1997.

[7] T.Sato, "Data dependence speculation combining memory disambiguation with address prediction", IPS Japan SIG Notes, 97-ARC-125, August 1997.

[8] T.Sato, "Load value prediction using reference address renaming", Proc. of JSPP'98, June 1998 (In Japanese).

[9] T.Sato, "Analyzing overhead of reissued instructions on data speculative processors", Digest of PAID held in conjunction with ISCA-25, June 1998.

[10] G.S.Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers", IEEE Trans. Comput., vol.39, no.3, 1990.

[11] G.Tyson, T.M.Austin, "Improving the accuracy and performance of memory communication through renaming", Proc. of MICRO-30, 1997.