First Step to Combining Control and Data Speculation

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

Abstract

Recently there are many studies of data value prediction for increasing instruction level parallelism, and it is found that data speculation affects branch prediction accuracy. Even when data dependences are speculated successfully, processor performance would be degraded if branch prediction accuracy were decreased. On the other hand, branch prediction studies are nearly matured. While it becomes very difficult to increase the accuracy, there is still a vast gap between actual and ideal processor performance. From these considerations, we investigate to combine control and data speculation. In this paper, we evaluate the correlation between control speculation and data speculation and then propose to predict branch outcomes using data value prediction.
Keywords: instruction level parallelism, speculative execution, branch prediction, value prediction, speculative verification

1 Introduction

Research in instruction level parallelism (ILP) architecture over the past decade has achieved considerable success to improve uniprocessor performance. However, the performance limit which is reported in the publication [1] has not been achieved. Obstacles disturbing the processor performance are dependences between instructions. They include control, name, and data dependences. Control dependences are dependences caused by branch instructions. Whenever the processor decodes a branch instruction, the instruction fetching stalls until the outcome of the branch instruction is produced. In order to eliminate the control dependences, branch prediction and (control) speculative execution are used. The outcome of a branch instruction is predicted at the fetch or decode stages and following instructions on the target path are speculatively executed. Name dependences are dependences caused by resource shortage of storage location. If results of two instructions are to be written in the same storage location, a write after write (WAW) hazard occurs and the latter instruction must stall. The name dependences can be eliminated by giving a different name to each storage location. For example, the dependences on registers can be removed by register renaming. Data dependences are dependences caused by a serialization on the program execution. When an instruction (a consumer) needs outcomes produced by other previous instructions (producers) as source operands, a read after write (RAW) hazard occurs and the consumer instruction must stall. The data dependences have to be strictly reserved in order to keep program semantics and are called true dependences.

Currently, we are investigating how to combine control speculation with data speculation. The motivations of this study are as follows. First, recently data speculation is extensively studied. We have found that branch prediction accuracy is affected by data speculation [2]. Even when data dependences can be effectively speculated, processor performance would be degraded if branch prediction accuracy were decreased. Second, studies of branch prediction are nearly matured. While recently studied branch predictors have high prediction accuracy, there is a vast gap between actual and ideal processor performance. Furthermore, it is becoming very difficult to improve the branch prediction accuracy. For example, in order to explore highly accurate branch predictors, the ones synthesized by genetic programming are proposed [3]. If the branch prediction accuracy can not be significantly improved in the future, reducing branch misprediction penalty is one of the factors improving effectiveness of control speculative execution. For example, data speculation is useful for reducing the branch misprediction penalty since branch outcomes are computed earlier [4]. From these considerations, we are beginning to develop a paradigm which exploits both control and data speculation.

The organization of the rest of this paper is as fol-

lows. In Section 2, previously proposed related works are surveyed. In Section 3 experimental model is described and preliminary simulation results are shown in Section 4. In Section 5 we describe our proposed microarchitecture. Finally, our conclusions are presented in Section 6.

2 Related Work

Branch prediction is a technique to eliminate the control dependences caused by branch instructions. The outcome of a branch instruction is predicted at the fetch or decode stages and following instructions on the target path are speculatively executed. The branch history table [5] is one of the simplest hardware-based branch predictors. It is indexed by the instruction address of a branch instruction. Each entry holds a history of a branch outcome which is usually encoded using 2-bit saturation counter. More sophisticated branch predictors are two-level adaptive branch predictors [6]. A two-level adaptive branch predictor consists of two tables. The branch history table of the two-level predictor is not directly referred by the branch instruction address. The branch instruction address is combined with branch history information and indexes the branch history table. The branch history information is held in a shift register called branch history register (BHR). When the latest branch outcome is stored in the BHR, the oldest outcome is removed. The branch history table is indexed by the branch instruction address and the BHR. Namely, the branch history table is indexed by a function of the pattern of branch outcomes. Hence, the branch history table is called pattern history table (PHT). The PHT consists of several two-bit saturated counters. These counters decide the predictions. It is found that the two-level adaptive branch predictors have high prediction accuracy and that the accuracy is approximately 97% [6]. One of the problems of the two-level adaptive branch predictor is aliasing in the tables. While recently studied branch predictors have high prediction accuracy, there is a vast gap between actual and ideal processor performance. Furthermore, it is becoming very difficult to improve the branch prediction accuracy significantly. Under these situations, predictors based on the data compression algorithms [7] and the genetic programming [3] are proposed. These proposals show that the branch prediction studies are almost mature.

There are many studies predicting register values [8–16]. These predictors are classified into two types in terms of the predicted instruction class. One is that predicting execution results of all types of instructions which write values into registers [8–11]. The other is that predicting only load values [12–16]. Last value predictor proposed by Lipasti [9, 13] introduces the value prediction concept, and is based on value locality. An instruction uses the same value which is executed at the last time, when the same instruction emerges in the future. Stride predictor proposed by Gabbay et al. [8] keeps not only the last outcome of an instruction but also a stride which is the difference between last two outcomes of the instruction. The predicted value is the sum of the last outcome and the stride. In order to improve prediction accuracy, several hybrid predictors [10, 11] are proposed. The hybrid predictor is a combination of several value predictors with a selector choosing the probably most accurate one. An another approach to improve value prediction accuracy is utilization of program execution profiles [8]. Using the profiles, instructions are classified according to the predictability, and compiler provides the classified information to processors. The prediction accuracy of load values can be improved by utilizing store instruction information. The schemes proposed by Moshovos et al. [14], Sato [15], and Tyson et al. [16] are very similar with each other, and thus we call them renaming-based value predictors in this paper. The renaming-based predictors speculatively streamline a stored value to a load instruction. The load value predictor used in this paper is based on one of the renaming-based predictors [15].

Recently, data speculation is combined with control speculation in order to execute multiple threads speculatively [17–20]. This paradigm of thread level parallelism is one of the promising candidates utilizing both control and data speculation. Instead, we are interested in combining control and data speculation in the paradigm of instruction level parallelism. In other words, the goal of this study is improving branch prediction accuracy and reducing branch misprediction penalty using data speculation.

3 Experimental Model

In this section, we describe the evaluation methodology by explaining a processor model and benchmark programs.

3.1 Processor model

The baseline model is an out-of-order execution superscalar processor based on register update unit (RUU) [21]. Following the discussion explained in [22], we decide the configuration of the baseline processor summarized in Table 1 as the representative of a practical processor in the coming generation. This decision

Table 1. 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

is made because our interest focuses on realistic processors. In order to evaluate the effect of speculations, an execution-driven simulation is more desirable than a trace-based simulation, since trace-based simulators can not simulate misspeculated instructions. We evaluate the effect by using SimpleScalar tool set [23]. The SimpleScalar architecture is based on MIPS architecture, and a fully execution-driven and cycle-by-cycle simulator is executed on a SPARCstation.

3.2 Data address prediction model

In order to predict data addresses, we utilize reference prediction table (RPT) [24]. The RPT, which has a similar structure with the instruction cache, is proposed by Chen et al. for hardware prefetching and keeps track of previous memory references. An entry of the RPT is indexed by an instruction address and holds a previous data address, a stride value, and a state information. The stride is the difference between last two data addresses generated by an instruction. The state information encodes the past history and indicates if next prediction is valid. The predicted address is the sum of the previous address and the stride. If the state information decides that the predicted address is valid, the load instruction speculates data dependences using the predicted address. The detailed explanations are presented in [2].

For the load address predictor, it is assumed that the RPT is direct-mapped and has 4096 entries.

3.3 Value prediction model

Our renaming-based load value predictor consists of three tables which are data-indexed store table (DIST), store-indexed value table (SIVT), and load-indexed store table (LIST). The DIST is indexed by a date address and keeps histories of memory references generated by store instructions. When it is referred by a load instruction, it links the load and a store instructions which refer the same memory location. The LIST is indexed by an instruction address and keeps the links. When the load instruction is fetched, the LIST is referred and supplies a tag which represents the link. The SIVT is indexed by the tag and keeps data values each of which is written by the store instruction specified by the tag. Thus, the load instruction can obtain the data value before calculating the data address. The detailed explanations are presented in [15].

For the load value predictor, it is assumed that the SIVT, LIST, and DIST are direct-mapped and have 4096 entries respectively.

3.4 Workload

The SPEC95 CINT benchmark suite is used for this study. The `test` input files which are provided by SPEC are used. Table 2 lists the benchmarks and the input sets. We use the object files provided by University of Wisconsin Madison [23]. Each program is executed to completion or for the first 100 million instructions. We count only committed instructions.

4 Preliminary Evaluation

We present preliminary evaluations to take a first step forward exploring the paradigm.

First, we present how data speculation affects branch prediction accuracy. Table 3 shows branch misprediction rate when load instructions are speculated using load address prediction [2]. The first

Table 3. (%)Branch misprediction rate/load address prediction

program	base			squash			reissue		
	addr	dir	jr	addr	dir	jr	addr	dir	jr
099.go	21.2	19.5	2.69	21.5	19.7	7.43	21.2	19.5	2.68
124.m88ksim	4.22	3.49	9.92	5.35	3.49	24.6	4.19	3.46	9.92
126.gcc	17.7	12.6	20.7	13.0	8.89	24.2	12.7	8.87	20.2
129.compress	3.39	2.98	3.53	3.51	3.00	4.96	3.41	3.00	3.54
130.li	6.39	4.40	13.7	7.94	3.86	28.3	6.22	4.26	13.5
132.ijpeg	1.60	1.20	0.26	1.62	1.20	0.37	1.60	1.20	0.26
134.perl	8.36	3.08	35.1	9.25	3.21	40.9	8.30	3.02	35.1
147.vortex	7.52	5.28	2.52	8.30	5.29	8.77	7.54	5.30	2.52

Table 2. Benchmark program and input file

program	input file
099.go	null.in
124.m88ksim	ctl.in
126.gcc	cccp.i
129.compress	test.in
130.li	test.lsp
132.ijpeg	specmun.ppm
134.perl	primes.in
147.vortex	vortex.in

column shows program name selected from SPEC95 benchmark suite. The remaining three groups of three columns indicate the results for the baseline, squash, and reissue models, respectively. The baseline model does not utilize data speculation. The squash and reissue models speculate load instructions, while recovery mechanisms for misspeculation are different. The squash model flushes all instructions following a mispredicted load instruction and fetches them again from instruction memory. The reissue model invalidates only instructions dependent upon a misspeculated load instruction and reissues them in instruction window. For each group, each column presents the accuracy of target address prediction (addr), that of branch direction prediction (dir), and that of the indirect jump address prediction (jr), respectively.

For the squash model branch prediction accuracy is degraded in general, especially for the indirect jump address prediction. One of the reasons why the branch prediction accuracy becomes worse is that it is harmful to update registers by useless speculative load instructions. For the reissue model, branch prediction accuracy increases in half of the evaluated programs. This is because branch instructions are resolved earlier. On the other hand, the number of programs whose prediction accuracy is decreased is only two. The reason why branch prediction accuracy does not decrease is the quick recovery from mispredictions of data dependence speculation. If the period between a misprediction and its recovery is long, many useless branch instructions are executed and thus the branch predictor is updated using wrong information. Instruction reissue shortens the period, and therefore the branch prediction accuracy is not decreased.

Table 4 shows branch misprediction rate when load instructions are speculated using load value prediction [15]. Layout and subject matter of Table 4 are the same as for Table 3. The same tendency with the former results is easily observed. The branch prediction accuracy for the squash model is significantly diminished and that for the reissued model is almost the same with that for the baseline model. However, in this evaluation, improvement of the branch prediction accuracy is hardly observed.

From these evaluations, it is found that one of the key factors improving the branch prediction accuracy is to resolve branch instructions as early as possible. This is possible by predicting source operands of branch instructions using data value prediction. Next, we present what happens when branch instructions are speculated using source value prediction.

Table 5 shows simulation results when ideal source value predictor is used. That is, source operands for branch instructions are always correctly predicted. The left side is for the baseline model, and the right side is for the evaluated model. For each side, the first three columns present branch misprediction rate. The forth column indicates average cycle time for resolving branch instructions, and the last column shows committed instructions per cycle (IPC). It is easily observed that the difference of prediction accuracy is very

Table 4. (%)Branch misprediction rate/load value prediction

program	base			squash			reissue		
	addr	dir	jr	addr	dir	jr	addr	dir	jr
099.go	21.2	19.5	2.69	25.5	19.2	71.7	21.2	19.5	2.65
124.m88ksim	4.22	3.49	9.92	4.98	3.49	19.8	4.23	3.49	9.92
126.gcc	17.7	12.6	20.7	19.0	12.5	34.8	17.7	12.6	20.7
129.compress	3.39	2.98	3.53	3.81	3.01	9.10	3.40	3.00	3.53
130.li	6.39	4.40	13.7	10.5	3.57	47.9	5.86	3.84	13.9
132.ijpeg	1.60	1.20	0.26	1.62	1.20	0.42	1.60	1.20	0.26
134.perl	8.36	3.08	35.1	11.4	3.20	57.1	8.37	3.09	35.1
147.vortex	7.52	5.28	2.52	8.20	5.23	8.69	7.56	5.32	2.52

Table 5. Speculation of branch instructions

program	base					speculative				
	addr	dir	jr	cycle	IPC	addr	dir	jr	cycle	IPC
099.go	21.2	19.5	2.69	8.76	1.52	21.2	19.5	2.77	3.79	1.99
124.m88ksim	4.22	3.49	9.92	5.93	2.85	4.21	3.48	9.94	3.42	3.14
126.gcc	17.7	12.6	20.7	7.29	1.49	17.7	12.7	20.7	4.16	1.71
129.compress	3.39	2.98	3.53	6.68	2.62	3.37	2.96	3.53	4.73	2.68
130.li	6.39	4.40	13.7	6.62	2.24	6.32	4.11	15.3	4.18	2.54
132.ijpeg	1.60	1.20	0.26	5.32	3.05	1.60	1.20	0.26	3.83	3.09
134.perl	8.36	3.08	35.1	6.33	2.09	8.28	3.00	35.2	4.03	2.40
147.vortex	7.52	5.28	2.52	6.72	1.91	7.49	5.26	2.52	4.50	1.98

small. This is disappointing and because the branch predictor used is updates speculatively at the decode stage using prediction. And thus, fast resolution of branch instructions does not contribute to branch prediction accuracy. Therefore, it is necessary to consider alternative branch predictors in order to utilize the fast resolution for improving the prediction accuracy. We are now investigating a mechanism which predicts a branch outcome using its predicted operands. It is an interesting characteristic that this branch predictor may correctly predict the outcome even if the predicted operands are incorrect, since what is essential to decide the outcome is not the individual operand values but the difference between them. Thus, this branch predictor will be more accurate than the source value predictor.

Referring back to Table 5, it is found that the branch resolution cycle is significantly reduced in the evaluated model. This means that branch misprediction penalty is also reduced. This is confirmed since the IPC is increased, especially for the programs whose branch misprediction rates are high. From these observations, we are also investigating to speculatively verify branch prediction by executing branch instructions actually using predicted source operands in order to reduce branch misprediction penalties. This is meta-speculation. When a prediction is incorrect, the misprediction penalty can be reduced by the correct speculative verification. A problem is that incorrect verification penalty might be huge. Thus, we have a plan to evaluate the incorrect verification penalty.

5 Branch Prediction using Value Prediction

In this section, we explain how to predict branch outcomes using data speculation technique. We also propose to verify branch predictions using data speculation technique.

5.1 Value-prediction-based branch predictor

Figure 1 depicts a processor utilizing a branch predictor we propose. We call the branch predictor a value-prediction-based branch predictor. A data value

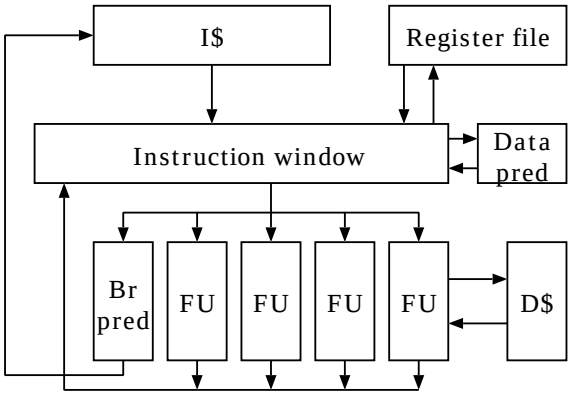

Figure 1. Processor diagram

predictor is indispensable for the processor to predict branch outcomes. The value-prediction-based branch predictor is placed below instruction window. It stands in a line with functional units. That is, the value-prediction-based branch predictor executes branch instructions actually.

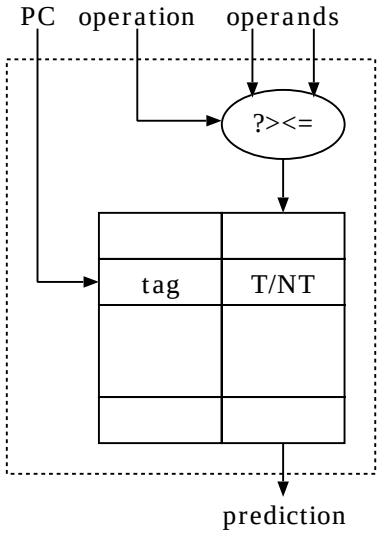

Figure 2. Branch predictor

Figure 2 shows the value-prediction-based branch predictor. It mainly consists of two components. One is a branch execution unit and the other is a branch outcome table. The branch execution unit is equivalent with ordinary branch unit. The branch outcome table is indexed by instruction addresses and keeps branch outcomes produced by the branch execution unit. When a branch instruction is fetched, the branch outcome table is referred in parallel and provides its predicted outcome.

In order to use the outcome as a prediction, the branch execution unit in the value-prediction-based branch predictor generates it using future operand values. The future values are predicted by the data value predictor. The value predictor must predict two consecutive values generated by an instruction. Figure 3 presents a value predictor based on the stride predictor. The current value is predicted as the sum of the previous value and the stride. The future value is predicted as the sum of the current value and the stride. This extension for providing two consecutive values is applicable to other value predictors. For example, the two-level predictor can predicts the future value if the predicted current value is used to index its table.

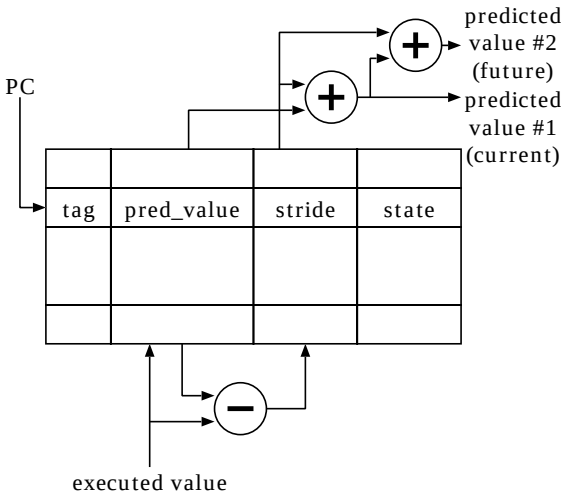

Figure 3. Value predictor

The branch execution unit in the value-prediction-based branch predictor executes a branch instruction using the future values predicted by the value predictor, when it executes the same branch instruction normally. The future branch outcome is held in the branch outcome table and will be provided when the same branch instruction emerges in the future.

From the above explanation, it can be seen that it becomes possible to predict branch outcomes using data value predictors.

5.2 Speculative verification

Figure 4 shows a processor which can detect mispredicted branches speculatively. Compared with Figure 1, it has a present branch predictor and the value-predictor-based branch predictor is replaced by

a branch prediction verifier. The branch prediction verifier performs the same function with the value-predictor-based branch predictor. Thus, it holds future branch outcomes. When a branch instruction emerges and its outcome is predicted by the present branch predictor, the verifier decides if the prediction is correct using the outcome in the branch outcome table. Since this verification is done before the branch instruction tests the condition actually, it is speculative. Therefore, the processor suffers miss-penalty when the speculative verification is incorrect.

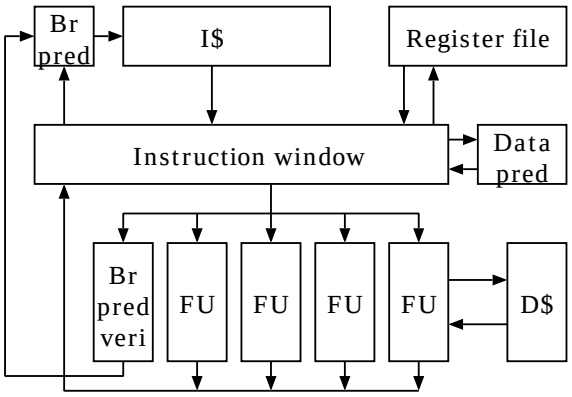

Figure 4. Speculative verification

6 Conclusions and Future Work

In this paper, in order to enhance control speculation using data speculation, we have proposed two schemes. One is predicting branch outcomes using data value prediction, and the other is speculative verification of branch prediction. We believe that it is a key factor for exploiting more instruction level parallelism to combine control and data speculation and that our proposals are first step to the goal.

Future studies dealing with the combination include implementing the proposed mechanisms and evaluating their usefulness. We also have a plan to investigate a hybrid predictor which consists of present branch predictors and the value-prediction-based branch predictor. And lastly, we must compare the efficiency of data speculation on thread-level and instruction-level parallelism architectures.

Acknowledgment

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

References

[1] M.S.Lam, R.P.Wilson, "Limites of Control Flow on Parallelism", *Proceedings of 19th International Symposium on Computer Architecture* (ISCA), 1992.

[2] T.Sato, "Speculative Resolution of Ambiguous Memory Aliasing", *Proceedings of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems* (IWIA), IEEE-CS Press, 1997.

[3] J.Emer, N.Gloy, "A Language for Describing Predictors and its Application to Automatic Synthesis", *Proceedings of 24th International Symposium on Computer Architecture* (ISCA), 1997.

[4] J.Gonzalez, A.Gonzalez, "The Potential of Data Value Speculation to Boost ILP", *Proceedings of 12th International Conference on Supercomputing* (ICS), 1998.

[5] J.K.F.Lee, A.J.Smith, "Branch Prediction Strategies and Branch Target Buffer Design", *IEEE Computer*, vol.17, no.1, 1984.

[6] T-Y.Yeh, Y.N.Patt, "Alternative Implementation of Two-Level Adaptive Branch Prediction", *Proceedings of 19th International Symposium on Computer Archtecture* (ISCA), 1992.

[7] I-C.K.Chen, J.T.Coffey, T.N.Mudge, "Analysis of Branch Prediction via Data Compression", *Proceedings of 7th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), 1996.

[8] F.Gabbay, A.Mendelson, "Can Program Profiling Support Value Prediction?", *Proceedings of 30th International Symposium on Microarchitecture* (MICRO), 1997.

[9] M.H.Lipasti, J.P.Shen, "Exceeding the Dataflow Limit via Value Prediction", *Proceedings of 29th International Symposium on Microarchitecture* (MICRO), 1996.

[10] Y.Sazeides, J.E.Smith, "The Predictability of Data Value", *Proceedings of 30th International Symposium on Microarchitecture* (MICRO), 1997.

[11] K.Wang, M.Franklin, "Highly Accurate Data Value Prediction using Hybrid Predictors", *Proceedings of 30th International Symposium on Microarchitecture* (MICRO), 1997.

[12] J.Gonzalez, A.Gonzalez, "Speculative Execution via Address Prediction and Data Prefetching", *Proceedings of 11th International Conference on Supercomputing* (ICS), 1997.

[13] M.H.Lipasti, C.B.Wilkerson, J.P.Shen, "Value Locality and Load Value Prediction", *Proceedings of 7th International Conference on Architectural Support for Programming Languages and Operation Systems* (ASPLOS), 1996.

[14] A.I.Moshovos, G.S.Sohi, "Streamlining Interoperation Memory Communication via Data Dependence Prediction", *Proceedings of 30th International Symposium on Microarchitecture* (MICRO), 1997.

[15] T.Sato, "Load Value Prediction using Reference Address Renaming", *Proceedings of 10th Joint Symposium on Parallel Processing* (JSPP), 1998 (In Japanese).

[16] G.Tyson, T.M.Austin, "Improving the Accuracy and Performance of Memory Communication Through Renaming", *Proceedings of 30th International Symposium on Microarchitecture* (MICRO), 1997.

[17] L.Codrescu, D.S.Wills, "Profiling for Input Predictable Threads", *Proceedings of International Conference on Computer Design* (ICCD), 1998.

[18] L.Hammond, M.Willey, K.Olukotun, "Data Speculation Support for a Chip Multiprocessor", *Proceedings of 8th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), 1998.

[19] P.Marcuello, A.Gonzalez, "Speculative Multithreaded Processors", *Proceedings of 12th International Conference on Supercomputing* (ICS), 1998.

[20] J.E.Smith, S.Vajapeyam, "Trace Processors: Moving to Forth-Generation Microarchitectures", *IEEE Computer*, vol.30, no.9, 1997.

[21] G.S.Sohi, "Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers", *IEEE Transactions on Computers*, vol.39, no.3, 1990.

[22] S.Jourdan, P.Sainrat, D.Litaize, "Exploring Configuration of Functional Units in an Out-of-Order Superscalar Processor", *Proceedings of 22nd International Symposium on Computer Architecture* (ISCA), 1995.

[23] D.Burger, T.M.Austin, "The SimpleScalar Tool Set, Version 2.0", *ACM SIGARCH Computer Architecture News*, vol.25, no.3, 1997.

[24] T-F.Chen, J-L.Baer, "Effective Hardware-Based Data Prefetching for High-Performance Processors", *IEEE Transactions on Computers*, vol.44, no.5, 1995.