Influence of Compiler Optimizations on Value Prediction

Toshinori Sato[1,2], Akihiko Hamano[3], Kiichi Sugitani[4], and Itsujiro Arita[1]

[1] Department of Artificial Intelligence, Kyushu Institute of Technology
[2] Center for Microelectronic Systems, Kyushu Institute of Technology
680-4 Kawazu, Iizuka, 820-8502 Japan
[3] Heart at Work Co., Ltd., 3-18 Honmachi, Iizuka, 820-0042 Japan
[4] Fujitsu Kyushu Digital Technology Ltd.,
3-22-8 Hakata-ekimae, Hakata-ku, Fukuoka, 812-0011 Japan
{tsato,akihiko,kiichi,arita}@mickey.ai.kyutech.ac.jp
http://www.mickey.ai.kyutech.ac.jp/~tsato/cosmos/

Abstract. The practice of speculation in resolving data dependences based on value prediction has been studied as a means of extracting more instruction level parallelism. There are many studies on value prediction mechanisms with high predictabilities. However, to the best of our knowledge, the influence of compiler optimizations on value prediction has not been investigated. In this paper we evaluate efficiency of value prediction on several binaries which are compiled with different optimization levels. Detailed simulations reveal that value prediction is still effective for highly optimized binaries.
Keywords: instruction level parallelism, data speculation, value prediction, optimization level, high-performance compilers

1 Introduction

In order to improve performance, modern microprocessors rely on exploiting instruction level parallelism (ILP). However, ILP is limited by dependences between instructions. They are classified into three classes — control, name, and data dependences. There are many studies to reduce control and name dependences, but data dependence remains as a major bottleneck limiting ILP. Data speculation based on value prediction is such a technique that addresses the problem by resolving data dependences speculatively. An outcome of an instruction is predicted by value predictors[6, 8]. The instruction and its dependent ones can be executed simultaneously, thereby exploiting ILP aggressively. Early works on value prediction mechanisms mainly consider hardware structures, and only their predictabilities and hardware costs are investigated. However, it is obvious that the characteristics of program binaries affect the value predictability, and thus it is not enough to study only hardware construction. For example, someone might expect that sophisticated optimizations obviate value prediction. From these considerations, in this paper we evaluate the relationship between compiler optimization levels and efficiency of value prediction.

The organization of the rest of this paper is as follows. Section 2 surveys related works. Section 3 describes our evaluation methodology. Section 4 contains simulation results. Finally, Section 5 presents our conclusions.

2 Related Work

Data speculation[6, 8] is a technique which executes instructions speculatively using predicted data values. Data dependences are speculatively resolved and thus ILP is increased. There are many studies proposing value prediction mechanisms, some of which achieve prediction accuracy as high as 80%, such as 2-level[20] and context-based[16] predictors. In order to improve prediction accuracy, several hybrid predictors[9, 11, 20] 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 using program execution profiles[3, 7, 13]. Using the profiles, instructions are classified according to the predictability, and a compiler provides the classified information to processors.

High prediction accuracies require considerable hardware cost. Morancho et al.[9], Rychlik et al.[11], and Calder et al.[4] examine capacity constraints of value predictors. Morancho et al.[9] and Rychlik et al.[11] proposed to reduce the hardware cost by classifying instructions based on their value predictability. Instructions which are easily predicted use simpler predictors such as the last-value and the stride predictors, whose hardware cost is low. High-cost predictors such as the 2-level, the hybrid, and the context-based predictors are used only for hard to predict instructions. Calder et al.[4] filter instructions based on their influence on processor performance. Only instructions on critical paths are held in a value predictor, reducing the number of entries of the predictor. Recently, we examined a technique reducing the hardware cost by exploiting narrow width values[14] and partial resolution[15]. Across SPEC programs, over 50% of integer operands are 16 bit or less[1]. That is, high-order bits of value prediction tables are merely utilized. Therefore, we proposed to keep only low-order bits of data values in the tables to reduce their hardware cost. Tag array size can be saved by employing partial resolution, using fewer tag address bits than necessary to uniquely identify every instruction. Full resolution of value predictors is not necessary since they do not have to be correct all the time. On the other hand, Fu et al.[5] and Tullsen et al.[19] remove value prediction hardware completely with the aid of compiler management of values in registers.

However, to the best of our knowledge, research on the influence of compiler optimizations on value prediction has been neglected. Recently, we have evaluated the relationship between compiler optimization levels and value predictability and found that there are meaningful value predictabilities in highly optimized binaries[18]. However, we have not evaluated the influence of compiler optimizations on processor performance. Therefore, in this paper we evaluate the efficiency of value prediction on variety of binaries compiled with different optimization levels and its contribution to processor performance.

3 Evaluation Methodology

In this section, we describe our evaluation methodology by explaining processor model and binaries used in this study.

3.1 Processor model

We model a realistic 8-way out-of-order execution superscalar processor based on register update unit (RUU)[17] which has 128 entries. Each functional unit can execute any operations. The latency for execution is 1 cycle except in the case of multiplication (4 cycles) and division (12 cycles). A 4-port, non-blocking, 128KB, 32B block, 2-way set-associative L1 data cache is used for data supply. It has a load latency of 1 cycle after the data address is calculated and a miss latency of 6 cycles. It has a backup of an 8MB, 64B block, direct-mapped L2 cache which has a miss latency of 18 cycles for the first word plus 2 cycles for each additional word. No memory operation can execute that follows a store whose data address is unknown. A 128KB, 32B block, 2-way set-associative L1 instruction cache is used for instruction supply and also has the backup of the L2 cache which is shared with the L1 data cache. For control prediction, a 1K-entry 4-way set associative branch target buffer, a 4K-entry gshare-type 2-level adaptive branch predictor, and an 8-entry return address stack are used. The branch predictor is updated at instruction commit stage.

In this paper, we investigate two value prediction mechanisms — last-value predictor[8] and hybrid predictor consisting of the stride and the 2-level predictors[20]. We use direct-mapped tables for these predictors. The former represents simple predictors and the latter does complex ones. The configuration of the hybrid predictor is based on [20]. When a misspeculation occurs, it is necessary to revert processor state to a safe point where the speculation is initiated. We use an instruction reissue mechanism[12] which selectively flushes and reissues misspeculated instructions.

We use two types of simulators for this study. One is a functional simulator for counting value predictability and the other is a timing simulator for evaluating processor performance. We implemented the simulators using the SimpleScalar tool set (ver.3.0a)[2]. The SimpleScalar/PISA instruction set architecture (ISA) is based on MIPS ISA.

3.2 Benchmark binaries

The binaries evaluated in this study are distributed by University of Michigan. They were compiled by GNU GCC (version 2.7.2.3) and MIRV[10]. For each compiler, three optimization levels, `-O0`, `-O1`, and `-O2`, were performed. Seven programs from eight SPEC95 CINT benchmarks are used for this study. The input files are modified so that evaluation time is practical. Each program is executed to completion. The candidate instructions predicted by the value predictors are register-writing ones, and do not include branch and store instructions. Tables 1 and 2 present total number of instructions executed and that

Table 1. Dynamic instruction count (GCC)

program	-O0		-O1		-O2	
099.go	268,225,895	(223,777,492)	146,032,803	(115,9999,78)	134,766,291	(104,239,257)
124.m88ksim	215,027,498	(155,551,958)	124,461,169	(86,167,274)	119,705,428	(81,391,241)
126.gcc	146,724,176	(100,785,825)	105,315,933	(69,649,020)	103,357,196	(67,359,249)
129.compress	77,092,827	(54,636,345)	48,821,478	(32,717,148)	47,719,938	(31,615,607)
130.li	307,778,502	(185,680,534)	208,780,589	(123,022,538)	206,414,110	(121,247,791)
132.ijpeg	18,082,420	(14,009,018)	8,831,646	(6.490,266)	8,606,970	(6,255,132)
134.perl	12,166,065	(7,775,114)	10,641,524	(6,611,315)	10,645,288	(6,607,758)

Table 2. Dynamic instruction count (MIRV)

program	-O0		-O1		-O2	
099.go	179,701,392	(142,959,132)	131,900,827	(98,921,730)	132,506,520	(99,186,262)
124.m88ksim	184,814,315	(130,030,485)	122,977,073	(82,888,272)	123,766,329	(85,244,701)
126.gcc	140,356,442	(96,215,914)	115,904,906	(75,804,673)	115,334,584	(86,961,482)
129.compress	69,108,399	(48,019,527)	49,210,033	(32,525,461)	47,700,633	(31,587,802)
130.li	270,793,655	(161,858,905)	207,709,666	(121,168,612)	207,705,881	(121,167,937)
132.ijpeg	12,045,397	(8,705,739)	9,751,916	(7,033,935)	9,995,242	(7,282,418)
134.perl	12,574,740	(8,233,448)	10,489,957	(6,644,109)	10,510,075	(6,660,903)

of instructions touched to predicted (in bracket) for every optimization level in the cases of GCC and MIRV respectively. Please note that the numbers are equivalent between the last-value and the hybrid predictors, since we count them using the functional simulator. In general, MIRV `-O0` executes considerably less instructions than GCC `-O0`, because MIRV has graph coloring allocation and copy propagation in `-O0` while GCC has no register allocation in `-O0`[10]. For the remaining optimization levels, the two compilers have almost equal abilities.

4 Simulation Results

In this section, we present simulation results. We define predictability as the number of instructions that are (correctly and incorrectly) predicted by a value predictor over that of all register-writing instructions. Prediction coverage is defined as the number of instructions correctly predicted over that of all register-writing instructions. Prediction accuracy is the percentage of instructions correctly predicted over all predicted instructions. Therefore, we have the following equation.

$$(Prediction\ coverage) = (Predictability) * (Prediction\ accuracy)$$

It has been found that the dominant factor in performance improvement is not prediction accuracy but prediction coverage[12, 19]. Therefore, We use the predictability and the prediction coverage as metrics for evaluation. After that, processor performance is evaluated.

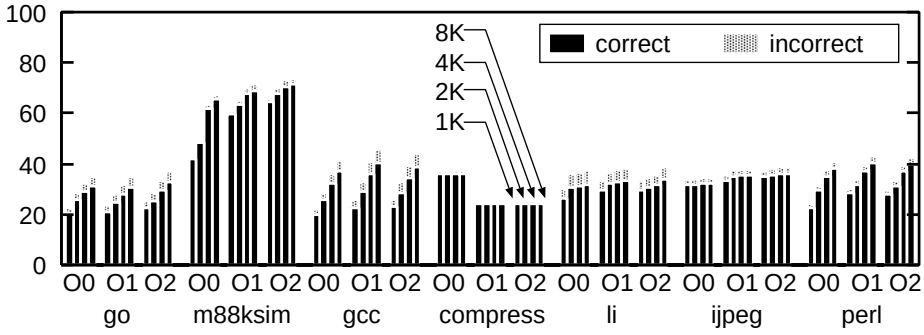

Fig. 1. %Value predictability (GCC/last)

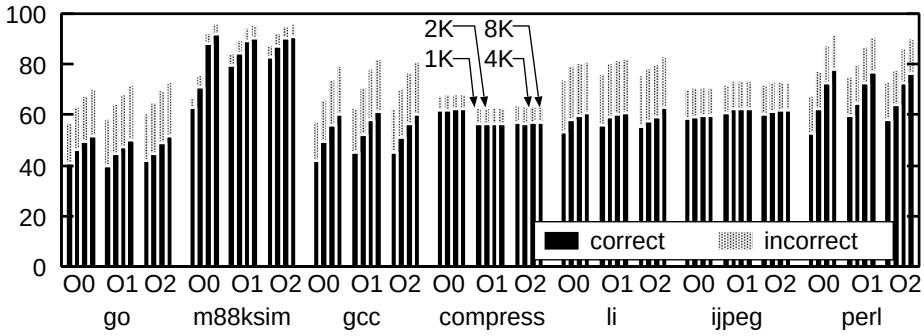

Fig. 2. %Value predictability (GCC/hybrid)

4.1 Predictability

Figure 1 shows simulation results when the last-value predictor is utilized on GCC binaries. The horizontal and vertical axes denote the optimization levels and the percentage respectively. Each bar, divided into two parts, indicates the predictability and the prediction coverage. The lower part (black) indicates the percentage of the instruction whose data value is correctly predicted. The upper part (gray) indicates the percentage that is mispredicted. That is, the lower part is the prediction coverage and the sum of the two parts is the predictability. For each group, bars from left to right indicate the results when the value predictor has 1024-, 2048-, 4096-, and 8192-entry tables, respectively. We can find the followings. First, in the case of `129.compress`, both the predictability and the prediction coverage are considerably reduced when the optimization level changes from `-O0` to `-O1`. This might mean that relatively high optimizations obviate the value prediction. Second, different from the previous observation, both the predictability and the prediction coverage are slightly improved as the optimization level becomes higher in the cases of the remaining programs. This

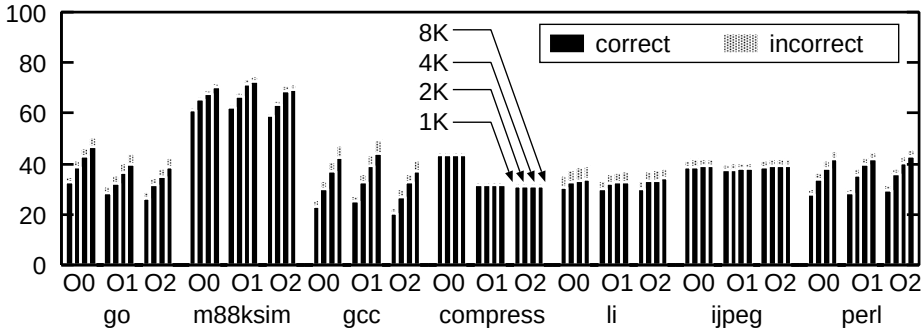

Fig. 3. %Value predictability (MIRV/last-value)

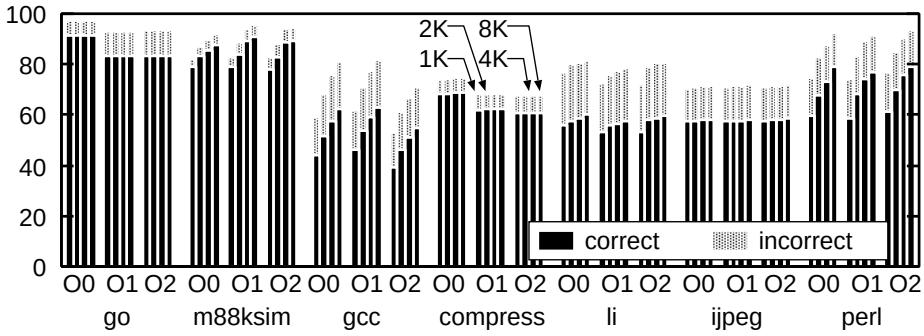

Fig. 4. %Value predictability (MIRV/hybrid)

denies the first investigation. Third, the efficiency of the value prediction changes little when the optimization level changes. Therefore, we can confirm that the conservative last-value predictor is effective for highly optimized binaries.

Figure 2 shows simulation results, when the hybrid predictor is utilized on GCC binaries. The characteristics similar to the last-value predictor case is observed. However, two programs should be mentioned. In the case of `129.compress`, the difference in the predictability and the prediction coverage between `-O0` and `-O1` are considerably smaller than the last-value predictor case. In the case of `134.perl`, both the predictability and the prediction coverage are slightly reduced as the optimization level becomes higher. However, these observations does not require the different conclusions from the last-value predictor case. In general, both the predictability and the prediction coverage rarely changes when different optimization levels are considered. And thus, the aggressive hybrid predictor is also effective for highly optimized binaries.

Figure 3 shows simulation results, when the last-value predictor is utilized to MIRV binaries. The different characteristics from GCC binaries can be observed.

First, we can find that in general both the predictability and the prediction coverage are greater in MIRV than in GCC. This can be observed throughout all programs and all optimization levels. Tables 1 and 2 say that there are not significant difference on dynamic instruction counts between two compilers. Therefore, MIRV can benefit from value prediction more than GCC. Second, it is generally found that both the predictability and the prediction coverage decrease slightly as the optimization level becomes higher. This difference from the GCC case can be found especially in the case of `099.go`. However, the decrease is not as large as that can be found in GCC `129.compress`. Thus, we can conclude that the conservative last-value predictor is still effective for MIRV binaries.

Figure 4 shows simulation results, when the hybrid predictor is utilized to MIRV binaries. The characteristics similar to the last-value predictor case is observed, while `099.go` shows considerably different behavior. Both the predictability and the prediction coverage, which are greater than in the GCC case, are reduced slightly when higher optimization levels are applied. In addition, for `099.go` both the predictability and the prediction coverage are considerably reduced when the optimization level changes from `-O0` to `-O1`. However, we can conclude that the aggressive hybrid predictor is also effective for highly optimized binaries.

4.2 Processor performance

Next, the influence on processor performance is evaluated. We use execution cycle time as a metric for the evaluation. Hence, the smaller the number is the higher the performance is. Every execution cycle time is normalized by the time in the case of `-O0` binary without data prediction. We use 4096-entry direct-mapped tables for this evaluation.

Figure 5 shows the contribution of the last-value predictor on GCC binaries. The horizontal axe indicates program names and optimization levels, and the vertical axe indicates relative processor performances. For each group of two bars, the left bar (gray) is for the case without data prediction and the right one (black) is for the case with prediction. First, when we compare the cases without data prediction, it is easily observed that processor performance is considerably improved as the optimization level is changed from `-O0` to `-O1`. In addition, the difference between performance of `-O0` binaries with data prediction and that of `-O1` binaries without data prediction is still significant. This implies that it is difficult for data prediction to contribute on binaries with conservative compiler optimizations such as legacy binaries. In other words, re-compilation is better than hardware-based optimizations for these binaries. Second, when we compare `-O1` binaries with prediction with `-O2` binaries without prediction. It is easily found that the former ones mark better performance than the latter ones. That is, it is possible to improve highly optimized binaries using data prediction.

Figure 6 shows the contribution of the hybrid predictor on GCC binaries. While the absolute contribution of data prediction increases, the influence of compiler optimizations on processor performance is similar with the case of the

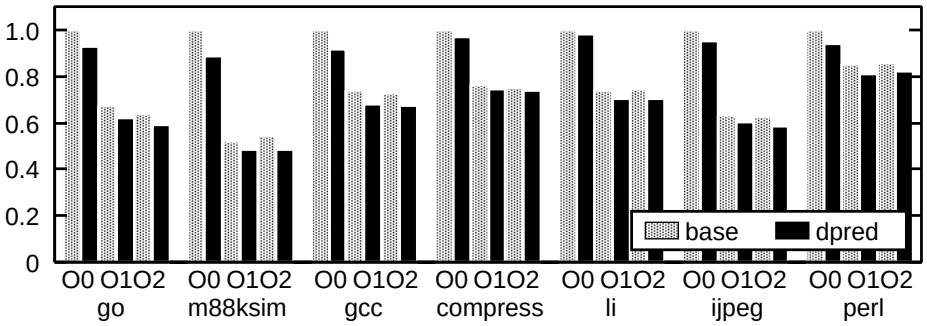

Fig. 5. Execution cycle time (GCC/last-value)

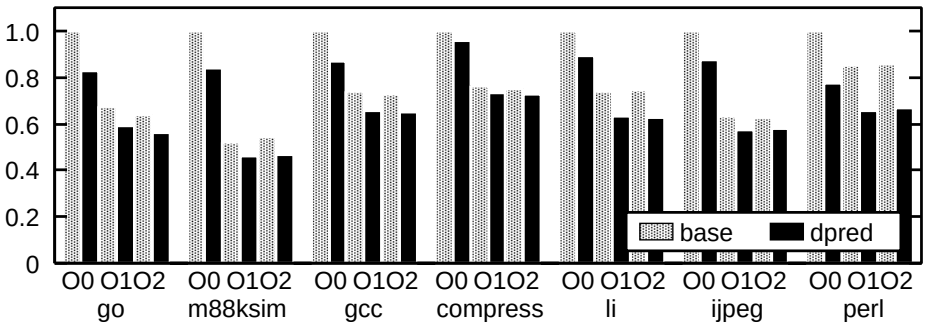

Fig. 6. Execution cycle time (GCC/hybrid)

last-value predictor. Only one exception is `134.perl`. Performance of `-O0` binary with prediction is better than that of `-O1` binary without prediction.

Figures 7 and 8 are for MIRV compiler. First, when we compare the cases without data prediction, the difference on performance between `-O0` and `-O1` is insignificant. This is because MIRV compiler performs higher optimization with `-O0` option than GCC as shown in Table 2. Nevertheless, the difference between performance of `-O0` binaries with data prediction and that of `-O1` binaries without prediction is still considerable. This confirms that it is difficult for data prediction to improve processor performance when legacy binaries are executed. It is also observed that improving highly optimized binaries using data prediction is possible for MIRV binaries.

5 Conclusion

This paper evaluated the influence of compiler optimizations on value prediction using GCC and MIRV binaries which were compiled with different optimization levels. The value prediction is a new technique, which resolves data

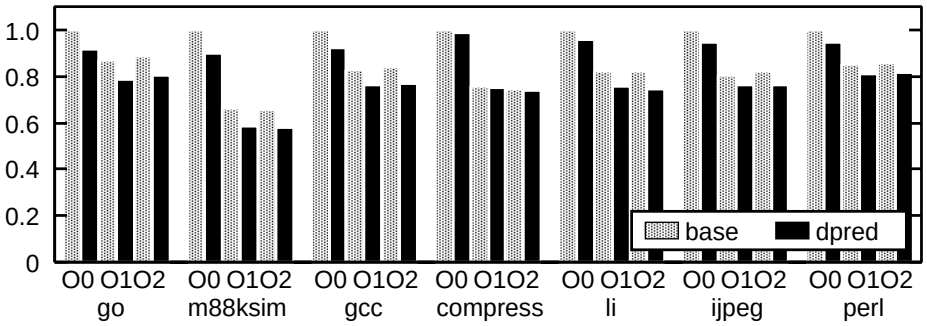

Fig. 7. Execution cycle time (MIRV/last value)

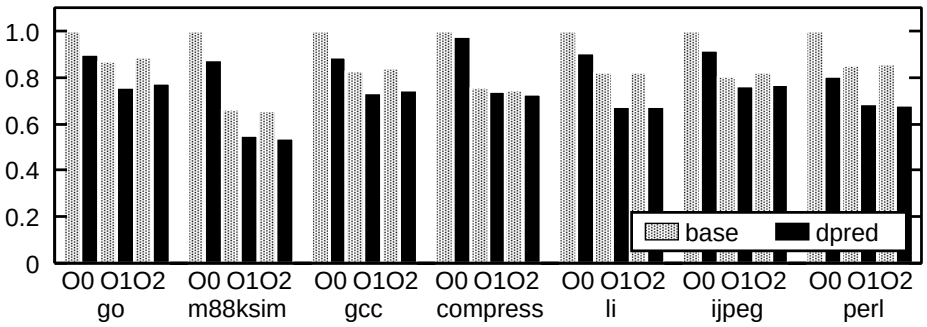

Fig. 8. Execution cycle time (MIRV/hybrid)

dependences speculatively in order to extract more ILP from programs with small parallelism. Previous works on value prediction proposed several tradeoff points between value predictability and hardware complexity. However, to the best of our knowledge, the relationship between compiler optimization levels and contribution of value prediction to processor performance has not been investigated. Therefore, in this paper we have evaluated value predictability of several binaries. From detailed simulation results, we have found that for both GCC and MIRV the value prediction is still effective for binaries compiled with every optimization level.

This paper is one result of our ongoing research in high-performance, low-power, and complexity-effective microprocessor *COSMOS* at Kyushu Institute of Technology. More information is available at our home page.

Acknowledgments

This work is supported in part by the grants from Japan Society for the Promotion of Science (no.12780273) and from Fukuoka Industry, Science & Technology Foundation (no.H12-1).

References

1. Brooks D., Martonosi M.: Dynamically exploiting narrow width operands to improve processor power and performance. 5th Int'l Symp. on High Performance Computer Architecture (1999)
2. Burger D., Austin T.M.: The SimpleScalar tool set, version 2.0. ACM SIGARCH Computer Architecture News, 25(3) (1997)
3. Calder B., Feller P., Eustace A.: Value profiling. 30th Int'l Symp. on Microarchitecture (1997)
4. Calder B., Reinman G., Tullsen D.M.: Selective value prediction. 26th Int'l Sym. on Computer Architecture (1999)
5. Fu C-y., Jennings M.D., Larin S.Y., Conte T.M.: Software-only value speculation scheduling. Technical Report, Dept. of Electrical and Computer Engineering, North Carolina State University (1998)
6. Gabbay F.: Speculative execution based on value prediction. Technical Report #1080, Dept. of Electrical Engineering, Technion (1996)
7. Gabbay F., Mendelson A.: Can program profiling support value prediction? 30th Int'l Symp. on Microarchitecture (1997)
8. Lipasti M.H., Shen J.P.: Exceeding the dataflow limit via value prediction. 29th Int'l Symp. on Microarchitecture (1996)
9. Morancho E., Llaberia J.M., Olive A.: Split last-address predictor. Int'l Conf. on Parallel Architectures and Compilation Techniques (1998)
10. Postiff M., Greene D., Lefurgy C., Helder D., Mudge T.: The MIRV SimpleScalar/PISA compiler. Technical Report CSE-TR-421-00, Dept. of Computer Science, University of Michigan (2000)
11. Rychlik B., Faistl J.W., Krug B.P., Kurland A.Y., Sung J.J., Velev M.N., Shen J.P.: Efficient and accurate value prediction using dynamic classification. Technical Report CMuART-98-01, Dept. of Electrical and Computer Engineering, Carnegie Mellon University (1998)
12. Sato T.: Analyzing overhead of reissued instructions on data speculative processors. Workshop on Performance Analysis and its Impact on Design held in conjunction with 25th Int. Symp. on Computer Architecture (1998)
13. Sato T.: Profile-based selection of load value and address predictors. 2nd Int'l Symp. on High Performance Computing (1999)
14. Sato T., Arita I.: Table size reduction for data value predictors by exploiting narrow width values. 14th Int'l Conf. on Supercomputing (2000)
15. Sato T., Arita I.: Partial resolution in data value predictors. 29th Int'l Conf. on Parallel Processing (2000)
16. Sazeides Y., Smith J.E.: Implementations of context based value predictors. Technical Report TR-ECE-97-8, Dept. of Electrical and Computer Engineering, University of Wisconsin-Madison (1997)
17. Sohi G.S.: Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. IEEE Trans. Comput., 39(3) (1990)
18. Sugitani K., Hamano A., Sato T., Arita I.: Evaluating effect of optimization level on value predictability. 4th Int'l Conf. on Algorithms and Architectures for Parallel Processing (2000)
19. Tullsen D.M., Seng J.S.: Strageless value prediction using prior register values. 26th Inte'l Symp. on Computer Architecture (1999)
20. Wang K., Franklin M.: Highly accurate data value prediction using hybrid predictors. 30th Int'l Symp. on Microarchitecture (1997)