

Short Paper

Evaluating Influence of Compiler Optimizations on Data Speculation

TOSHINORI SATO^{*,} KIICHI SUGITANI^{+,} AKIHIKO HAMANO^{**}
AND ITSUJIRO ARITA^{*}

**Department of Artificial Intelligence*

+Center for Microelectronic Systems

Kyushu Institute of Technology

680-4 Kawazu, Iizuka, 820-8502 Japan

++Fujitsu Kyushu Digital Technology Ltd.

Fukuoka, Japan

***HAW International Inc.*

Iizuka, Japan

The practice of using 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 the 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 levels, 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. Many studies have attempted to reduce control and name dependences, but data dependence remains a major bottleneck limiting ILP. Data speculation based on value prediction is a technique that addresses this problem by resolving data dependences speculatively. An outcome of an instruction is predicted by means of value predictors [6, 10]. The instruction and its dependent ones can be executed simultaneously, thereby exploiting ILP aggressively. Early works on value prediction mechanisms mainly considered hardware structures, and only their predictabilities and hardware costs were investigated. However, it is obvious that the characteristics of program binaries affect value predictability; thus,

Received August 27, 2001; accepted April 15, 2002.

Communicated by Jang-Ping Sheu, Makoto Takizawa and Myongsoon Park.

it is not sufficient to study only hardware construction. For example, one might expect that sophisticated optimizations obviate value prediction. If we have both a legacy binary and its source code, we would like to know which is better to use the legacy binary with hardware-based optimization like value prediction or to re-compile the source code using up-to-date compilers. Based on these considerations, in this paper, we evaluate the relationship between compiler optimization levels and the 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, 8-10, 14] is a technique which executes instructions speculatively using predicted data values. Data dependences are speculatively resolved; thus, ILP is increased. Many studies have proposed value prediction mechanisms, some of which achieve prediction accuracy as high as 80%, such as 2-level [24] and context-based [20] predictors. In order to improve prediction accuracy, several hybrid predictors [11, 13, 24] have been proposed. A hybrid predictor is a combination of several value predictors with a selector choosing the most probably accurate one. Another approach to improving value prediction accuracy is using program execution profiles [3, 7, 15]. Using profiles, instructions are classified according to predictability, and a compiler provides the classified information to processors.

Achieving high prediction accuracy requires a considerable hardware cost. Morancho et al. [11], Rychlik et al. [13], and Calder et al. [4] examined capacity constraints of value predictors. Morancho et al. [11] and Rychlik et al. [13] proposed to reduce the hardware cost by classifying instructions based on their value predictability. Instructions which are easily predicted use simpler predictors, such as last-value and 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] proposed to filter instructions based on their criticality. Only instructions on critical paths were held in a value predictor, thus reducing the number of entries of the predictor. Recently, we examined a technique that reduces the hardware cost by exploiting narrow width values [16] and partial resolution [17]. Across SPEC programs, over 50% of the 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. The tag array size can be reduced by employing partial resolution and 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. [23] proposed to remove value prediction hardware completely with the aid of compiler management of values in registers.

However, to the best of our knowledge, the influence of compiler optimizations on value prediction has been neglected by researchers. 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 [22]. This paper

extends our previous study and we evaluate the efficiency of value prediction on a 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 the processor model and binaries used in this study.

3.1 Processor Model

We model an 8-way out-of-order execution superscalar processor based on a register update unit (RUU)[21], that has 128 entries. Each functional unit can execute any operation. 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 consisting 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 a backup consisting of an L2 cache that is shared with an 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 the instruction commit stage.

In this paper, we investigate two value prediction mechanisms – last-value predictor [10] and a hybrid predictor consisting of the stride and the 2-level predictors [24]. 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 [24]. 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 [19], 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 the University of Michigan. They were compiled by GNU GCC (version 2.7.2.3) and MIRV [12]. For each compiler, three optimization levels, -O0, -O1, and -O2, were performed. Due to the limit of space, the detail of each optimization is not described in this paper. Please refer [12]. Seven programs from eight SPEC95 CINT benchmarks were used for this study. The input files

were modified so that the evaluation time would be practical. Each program is executed to completion. The candidate instructions predicted by the value predictors were register-writing ones, and did not include branch and store instructions. Tables 1 and 2 present the total number of instructions executed and that of instructions touched to predicted (in brackets) for every optimization level in the cases of GCC and MIRV respectively. Please note that the numbers are equivalent for the last-value and the hybrid predictors since we counted them using a functional simulator. In general, MIRV -O0 executed considerably fewer instructions than GCC -O0 because MIRV has graph coloring allocation and copy propagation in -O0 while GCC has no register allocation in -O0 [12]. For the remaining optimization levels, the two compilers produced almost equal results.

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.m8ksim	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.jpeg	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.m8ksim	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.jpeg	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)

4. SIMULATION RESULTS

In this section, we present simulation results [18, 22]. We define predictability as the number of instructions that are (correctly and incorrectly) predicted by a value predictor over the total number of register-writing instructions. Prediction coverage is defined as the number of instructions correctly predicted over the total number of register-writing instructions. Prediction accuracy is the percentage of instructions correctly predicted out of all the predicted instructions. Therefore, we have the following equation;

$$(\text{Prediction coverage}) = (\text{Predictability}) * (\text{Prediction accuracy}).$$

It has been found that the dominant factor in performance improvement is not prediction accuracy but prediction coverage [19, 23]. Therefore, we used predictability and prediction coverage as metrics for evaluation. After that, processor performance was evaluated.

4.1 Predictability

Fig. 1(i) shows simulation results obtained when the last-value predictor was 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 instructions 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 obtained when the value predictor had 1024-, 2048-, 4096-, and 8192-entry tables, respectively. We can find the following. First, in the case of 129.compress, both the predictability and the prediction coverage were considerably reduced when the optimization level changed 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 were slightly improved as the optimization level became higher for the remaining programs. This denies the first investigation. Third, the efficiency of the value prediction changed little when the optimization level changed. Therefore, we can confirm that the conservative last-value predictor is effective for highly optimized binaries.

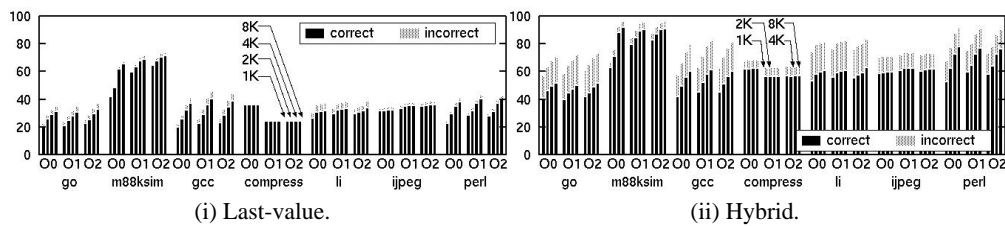


Fig. 1. % Value predictability (GCC).

Fig. 1(ii) shows simulation results obtained when the hybrid predictor was utilized on GCC binaries. The characteristics similar to those of the last-value predictor case was observed. However, two programs should be mentioned. In the case of 129.compress, the differences in predictability and the prediction coverage between -O0 and -O1 were than considerably smaller than those for the last-value predictor case. In the case of 134.perl, both predictability and prediction coverage worsened slightly as the optimization level became higher. However, these observations do not lead to different conclusions from those drawn for the last-value predictor case. In general, both predictability and prediction coverage rarely changed when different optimization levels were considered. Thus, the aggressive hybrid predictor is also effective for highly optimized binaries.

Fig. 2(i) shows simulation results obtained when the last-value predictor was applied to MIRV binaries. Characteristics different from those of GCC binaries can be observed. First, we can find that, in general, both predictability and prediction coverage improved in MIRV more than in GCC. This can be observed throughout all the programs and all optimization levels. Tables 1 and 2 show that there were not significant differences in the dynamic instruction counts between the two compilers. Therefore, MIRV can benefit

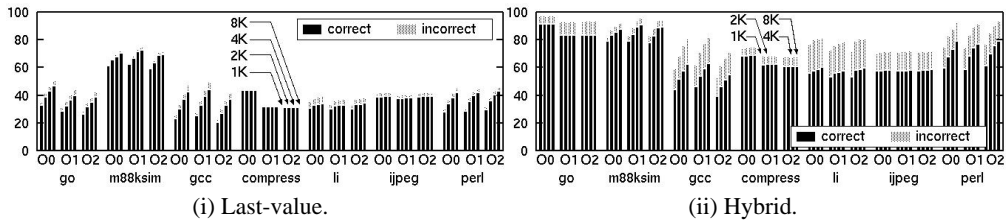


Fig. 2. % Value predictability (MIRV).

from value prediction more than GCC. Second, it is generally found that both predictability and prediction coverage worsened slightly as the optimization level became higher. This difference from the GCC case can be found especially in the case of 099.go. However, this decrease is not as large as that in GCC 129.compress. Thus, we can conclude that the conservative last-value predictor is still effective for MIRV binaries.

Fig. 2(ii) shows simulation results obtained when the hybrid predictor was applied to MIRV binaries. The characteristics similar to those of the last-value predictor case are observed, while 099.go shows considerably different behavior. Both predictability and prediction coverage, which were better than in the GCC case, worsened slightly when higher optimization levels were applied. In addition, for 099.go, both predictability and prediction coverage were considerably worse when the optimization level changed 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 was evaluated. We used the execution cycle time as a metric for evaluation. Hence, the smaller the number was the better the performance was. Every execution cycle time was normalized by the time in the case of -O0 binary without data prediction. We used 4096-entry direct-mapped tables for this evaluation.

Fig. 3(i) shows the contribution of the last-value predictor on GCC binaries. The horizontal axis indicates the program names and optimization levels, and the vertical axis indicates the relative processor performance. 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 improved considerably as the optimization level is changed from -O0 to -O1. In addition, the difference between the performance of -O0 binaries with data prediction and that of -O1 binaries without data prediction was 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 achieved better performance than the latter ones. Therefore, it is possible to improve highly optimized binaries using data prediction.

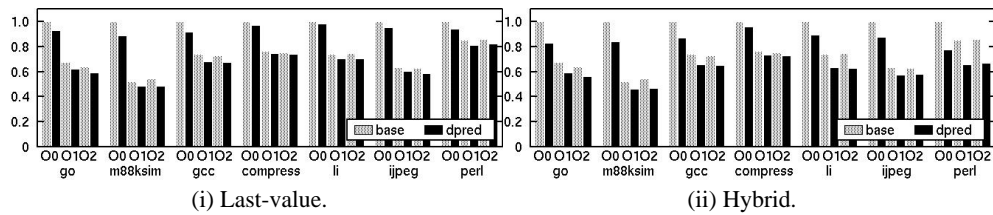


Fig. 3. Execution cycle time (GCC).

Fig. 3(ii) shows the contribution of the hybrid predictor to GCC binaries. While the absolute contribution of data prediction increased, the influence of compiler optimizations on processor performance was similar to that of the last-value predictor. The only exception is 134.perl, where the performance of -O0 binary with prediction is better than that of -O1 binary without prediction.

Fig. 4 show the execution time results for the MIRV compiler. First, when we compare the cases without data prediction, the difference in performance between -O0 and -O1 is insignificant. This is because the MIRV compiler performs higher optimization with the -O0 option than GCC does, as shown in Table 2. Nevertheless, the difference between the 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.

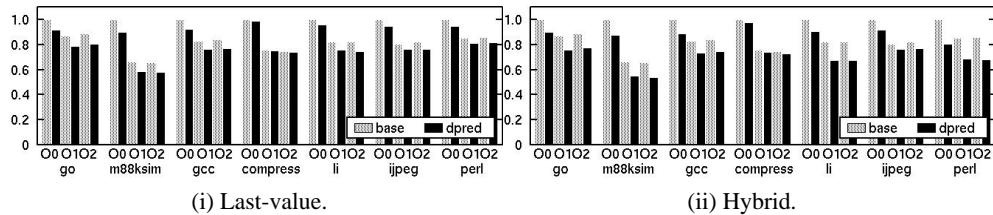


Fig. 4. Execution cycle time (MIRV).

5. CONCLUSIONS

This paper has evaluated the influence of compiler optimizations on value prediction using GCC and MIRV binaries that were compiled with different optimization levels. Value prediction is a new technique that resolves data dependences speculatively in order to extract more ILP from programs with low levels of 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 the contribution of value prediction to processor performance has not been investigated. Therefore, in this paper we have evaluated the value predictability of several binaries. From detailed simulation results, we have found the fol-

lowings. For both GCC and MIRV, the value prediction is still effective for binaries compiled with every optimization level. However, it is better to re-compile legacy binaries than to optimize them with the help of dedicated hardware such as value predictors. Lastly, it is still possible to improve highly optimized binaries using data prediction.

ACKNOWLEDGMENTS

This work was supported in part by Grant-in-Aid for Encouragement of Young Scientists (No. 12780273) and the Grant-in-Aid for Scientific Research (No. 13558030) both from the Japan Society for the Promotion of Science. Toshinori Sato was supported in part by a grant from Fukuoka Industry, Science & Technology Foundation (No. H12-1).

REFERENCES

1. D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proceedings of 5th International Symposium on High Performance Computer Architecture*, 1999, pp. 13-22.
2. D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," *ACM SIGARCH Computer Architecture News*, Vol. 25, 1997, pp. 13-25.
3. B. Calder, P. Feller, and A. Eustace, "Value profiling," in *Proceedings of 30th International Symposium on Microarchitecture*, 1997, pp. 259-269.
4. B. Calder, G. Reinman, and D. M. Tullsen, "Selective value prediction," in *Proceedings of 26th International Symposium on Computer Architecture*, 1999, pp. 64-73.
5. C. -Y. Fu, M. D. Jennings, S. Y. Larin, and T. M. Conte, "Software-only value speculation scheduling," Technical Report, Dept. of Electrical and Computer Engineering, North Carolina State University, 1998.
6. F. Gabbay, "Speculative execution based on value prediction," Technical Report #1080, Dept. of Electrical Engineering, Technion, 1996.
7. F. Gabbay and A. Mendelson, "Can program profiling support value prediction?," in *Proceedings of 30th International Symposium on Microarchitecture*, 1997, pp. 270-280.
8. F. Gabbay and A. Mendelson, "The effect of instruction fetch bandwidth on value prediction," in *Proceedings of 25th International Symposium on Computer Architecture*, 1998, pp. 272-281.
9. J. Gonzalez and A. Gonzalez, "The potential of data value speculation to boost ILP," in *Proceedings of 12th International Conference on Supercomputing*, 1998, pp. 21-28.
10. M. H. Lipasti and J. P. Shen, "Exceeding the dataflow limit via value prediction," in *Proceedings of 29th International Symposium on Microarchitecture*, 1996, pp. 226-237.
11. E. Morancho, J. M. Llaberia, and A. Olive, "Split last-address predictor," in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, 1998, pp. 230-237.
12. M. Postiff, D. Greene, C. Lefurgy, D. Helder, and T. Mudge, "The MIRV SimpleS-

- calar/PISA compiler,” Technical Report CSE-TR-421-00, Dept. of Computer Science, University of Michigan, 2000.
13. B. Rychlik, J. W. Faistl, B. P. Krug, A. Y. Kurland, J. J. Sung, M. N. Velev, and J. P. Shen, “Efficient and accurate value prediction using dynamic classification,” Technical Report CMuART-98-01, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, 1998.
 14. B. Rychlik, J. Faistl, B. Krug, and J. P. Shen, “Efficacy and performance impact of value prediction,” in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, 1998, pp. 148-152.
 15. T. Sato, “A simulation study of combining load value and address predictors,” *International Journal of High Speed Computing*, Vol. 10, 1999, pp. 301-325.
 16. T. Sato and I. Arita, “Table size reduction for data value predictors by exploiting narrow width values,” in *Proceedings of 14th International Conference on Supercomputing*, 2000, pp. 196-205.
 17. T. Sato and I. Arita, “Partial resolution in data value predictors,” in *Proceedings of 29th International Conference on Parallel Processing*, 2000, pp. 69-76.
 18. T. Sato, A. Hamano, K. Sugitani, and I. Arita, “Influence of compiler optimizations on value prediction,” in *Proceedings of 9th International Conference on High Performance Computing and Networking*, 2001, pp. 312-321.
 19. T. Sato, “Evaluating the impact of reissued instructions on data speculative processor performance,” *Microprocessors and Microsystems*, Vol. 25, 2002, pp. 469-482.
 20. Y. Sazeides and J. E. Smith, “Implementations of context based value predictors,” Technical Report TR-ECE-97-8, Dept. of Electrical and Computer Engineering, University of Wisconsin-Madison, 1997.
 21. G. S. Sohi, “Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers,” *IEEE Transactions on Computers*, Vol. 39, 1990, pp. 349-359.
 22. K. Sugitani, A. Hamano, T. Sato, and I. Arita, “Evaluating effect of optimization level on value predictability,” in *Proceedings 4th International Conference on Algorithms and Architectures for Parallel Processing*, 2000, pp. 695-698.
 23. D. M. Tullsen and J. S. Seng, “Storageless value prediction using prior register values,” in *Proceedings of 26th International Symposium on Computer Architecture*, 1999, pp. 270-279.
 24. K. Wang and M. Franklin, “Highly accurate data value prediction using hybrid predictors,” in *Proceedings of 30th International Symposium on Microarchitecture*, 1997, pp. 281-290.

Toshinori Sato is an associate professor in the Department of Artificial Intelligence at Kyushu Institute of Technology in Iizuka, Japan. He holds BE, ME, and PhD degrees in electronic engineering from Kyoto University. From 1991 to 1999 he was with Toshiba Corporation, where he worked on the design of several embedded processors such as EmotionEngine for PlayStation2. His research interests include dynamically-adaptable, energy-efficient, dependable, and complexity-effective microarchitectures. He is a member of ACM, IEEE, IEICE, and IPSJ.

Kiichi Sugitani is with Fujitsu Kyushu Digital Technology Ltd. in Fukuoka, Japan. He received BE degree from Kyushu Institute of Technology in 2001. He worked on the effect of compiler optimization on value predictability for the KIT COSMOS processor, and implemented a low power instruction cache for embedded processors. His research interests include processor architectures and parallel processing.

Akihiko Hamano is with HAW International Inc. in Iizuka, Japan. He received BE degree from Kyushu Institute of Technology in 2001. He worked on the effect of compiler optimization on value predictability for the KIT COSMOS processor, and designed a platform for computer architecture education. His research interests include processor architectures and parallel processing.

Itsujiro Arita is a professor in the Department of Artificial Intelligence at Kyushu Institute of Technology in Iizuka, Japan. He received his PhD degree from Kyushu University in Fukuoka, Japan. He was with Kyushu University from 1965 to 1984. His research interests include computer architecture, parallel processing, operating systems, and computer networks. He is a member of IEICE, IPSJ, and JSSST.