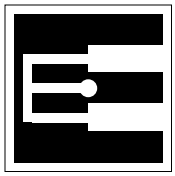

Scholarly Paper in Computer Engineering

Non-Thesis Option for Master of Science Degree
May 1998

This document describes the requirements for writing and submitting a scholarly paper in Computer Engineering.

Intent of Scholarly Paper

The intent of the scholarly paper is to demonstrate expertise in an area of your choosing by summarizing past work in the area and placing the open questions in perspective. The document should be on the order of ten double-spaced pages in length, or equivalently two to three pages of a conference paper or journal article.

The goal is that the submitted scholarly paper resemble the combined Introduction, Background, and Related Work sections of a publishable conference paper or journal article. No new research is required; the paper is intended as a survey of existing research in the chosen field.

Expectations and Requirements

The student is expected to generate the equivalent of a publishable Introduction and Research Overview. This leads to the following requirements:

- The scholarly paper must reference and place in perspective all relevant literature on the topic. This is likely to amount to several dozen papers.
- The paper must motivate and place in perspective at least one of the important open research questions in the area.
- The paper should not describe in detail previous research designs, but should instead motivate those designs and compare and contrast them.
- The length of the paper should be 1000–3000 words.
- The format of the paper should be 11–12 point font, double spaced, with 1-inch margins on all sides. The student must submit hardcopy as well as electronic copy in Framemaker, Microsoft Word, or LaTeX format.
- The paper must be written entirely in the student's own words. See the University's policy on plagiarism for more details.

The student is expected to interact regularly with the advisor of the scholarly paper, and the advisor is expected to verify that the work is original.

Examples

There are a number of particularly good examples of well-written Introduction and Related Work sections. Here are a few; note that they are a bit shorter than the length expected for a scholarly paper.

Branch prediction

C. Lee, I. Chen, and T. Mudge. 1997. “The bi-mode branch predictor.” In *Proc. 30th Annual International Symposium on Microarchitecture (MICRO-30)*, Research Triangle Park NC.

Data prediction

Y. Sazeides and J. Smith. 1997. “The predictability of data values.” In *Proc. 30th Annual International Symposium on Microarchitecture (MICRO-30)*, Research Triangle Park NC.

Hardware-software interaction

T. E. Anderson, H. M. Levy, B. N. Bershad, and E. D. Lazowska. 1991. “The interaction of architecture and operating system design.” In *Proc. Fourth Int’l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-4)*, pages 108–120.

(Here is the Branch Prediction example)

Abstract

Dynamic branch predictors are popular because they can deliver accurate branch prediction without changes to the instruction set architecture or pre-existing binaries. However, to achieve the desired prediction accuracy, existing dynamic branch predictors require considerable amounts of hardware and/or compiler assistance to minimize the interference effects due to aliasing in the prediction tables. We propose a new dynamic predictor, the bi-mode predictor, which divides the prediction tables into two halves and, by dynamically determining the current “mode” of the program, selects the appropriate half of the table for prediction. This approach is shown to preserve the merits of global history based prediction while reducing destructive aliasing and, as a result, improving prediction accuracy. Moreover, it is simple enough that it does not impact a processor’s cycle time. We conclude by conducting a comprehensive study into the mechanism underlying two-level dynamic predictors and investigate the criteria for their optimal designs.

1. Introduction

The ability to minimize stalls or pipeline bubbles that may result from branches is becoming increasingly critical as microprocessor designs implement greater degrees of instruction level parallelism. There are several techniques for reducing branch penalties including guarded execution, basic block enlargement, and static and dynamic branch prediction [12, 7, 14, 4, 19, 11]. Among these, dynamic branch prediction is perhaps the most popular, because it yields good results and can be implemented without changes to the instruction set architecture or pre-existing binaries.

The strength of dynamic branch prediction is that it can track branch behavior closely at run-time, providing a degree of adaptivity that other approaches are lacking. This adaptivity is especially critical when behavior of branches can be affected by the input data of different program runs. With the introduction of two-level schemes [19], the prediction accuracy of dynamic branch predictors has been pushed above 90%. As a result, two-level dynamic branch predictors have been incorporated in several recent high-performance microprocessors. Perhaps the best known examples are the Pentium Pro [5] and Alpha 21264 [6].

Among two-level predictors, those using global history schemes have been shown to yield the best performance for integer benchmarks [21]. However, to achieve high levels of accuracy, current dynamic branch predictors require con-

siderable amounts of hardware because their most significant weakness, the destructive aliasing problem, is most easily solved by increasing the size of the predictors [13]. This paper propose a new technique, the bi-mode branch predictor, that is economical and simple enough to avoid critical timing paths. Furthermore, we demonstrate that for the IBS and SPEC CINT95 benchmarks the bi-mode predictor performs on average better than the best global history based predictors for the same cost. Finally, we present an analysis of aliasing in dynamic branch predictors that explains the source of the improved performance.

The report is organized into five sections. In section 2, we summarize the aliasing problem, and then introduce our solution for de-aliasing. Section 3 describes our simulation methodology and presents the simulation results. In section 4 we investigate the behaviors of existing two-level dynamic branch predictors and explain why our proposed scheme performs better. Finally, in the conclusion we propose future directions for this work.

2. Aliasing and de-aliasing

Branch outcomes are not usually the result of random activities; most of time they are correlated with past behavior and the behavior of neighboring branches. By keeping track of the history of branch outcomes it is possible to anticipate with a high degree of certainty which direction future branches will take.

However, current dynamic branch predictors still exhibit performance limits. These are due in part to the restricted availability of information upon which to base predictions, but more importantly due to shortcomings of design, especially the way that branch outcome history is exploited. In current designs, dynamic predictors spend large amounts of hardware to memorize this branch outcome history. Each static (per-address) branch often has a biased behavior so that it is either usually taken or usually not-taken. This can be exploited by the conventional two-bit counter scheme to predict future outcomes of a particular static branch. However, two-bit counter schemes are limited because branches may behave differently from their biases under some special conditions. These conditions are not difficult to recognize, but recognition requires memory space. Therefore, to achieve very high prediction accuracy, both per-address bias and the special conditions need to be identified and memorized by dynamic predictors.

Global history—the outcomes of neighboring branches—is a common way to identify special branch conditions. Previous studies have shown that the global history indexed schemes have good performance by storing the outcomes of global history patterns in two-bit counters, e.g., the GAg and GAs schemes of Yeh and Patt [20]. Another way to identify special branch conditions is to use per-address history—the past outcomes of a branch itself, such as PAg and PAs schemes [19]. The per-address history scheme is also shown to be effective, especially for loop-intensive floating-point programs. However, as we noted earlier, [21] show that for integer programs global history schemes tend to perform better than per-address history schemes, because global schemes can make better predictions for if-then-else branches due to their ability to track correlation with neighboring branches.

Nevertheless, the global history scheme is still limited by destructive aliasing that occurs when two branches have the same global history pattern, but opposite biases [17, 22]. This is not due to the limited availability of information, but to the indexing method which does not discriminate between branches with the same global history patterns, as will be shown in this paper.

One proposal for de-aliasing to overcome the problem, gshare, randomizes the index by xor-ing the global history with branch address [9]. However, it provides limited improvement [13]. Recently, there have been several new proposals to reduce the aliasing [2, 10, 16]. It is necessary to examine and compare for these proposals in details, and therefore we report our comparison results in [8].

References

[1] Chang, P., Hao, E., Yeh, T., and Patt, Y., “Branch Classification: a New Mechanism for Improving Branch Predictor Performance,” *IEEE Micro-27*, Nov. 1994.

[2] Chang, P., Evers, M., and Patt, Y., “Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference,” *International Conference on Parallel Architecture and Compilation Techniques*, Oct. 1995.

[3] Eustace, A. and Srivastava, A., “ATOM: A flexible interface for building high performance program analysis tools,” *Proceedings of the Winter 1995 USENIX Technical Conference on UNIX and Advanced Computing Systems*, 303-314, Jan. 1995.

[4] Fisher, J.A., and Freudenberger, S.M. “Predicting Conditional Branch Directions From Previous Runs of a Program, Proc.” *5th Annual Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 1992.

[5] Gwennap, L. “Intel’s P6 Uses Decoupled Superscalar Design,” *Microprocessor Report*, Vol. 9, No.2, Feb. 16, 1995.

[6] Gwennap, L. “Digital 21264 Sets New Standard,” *Microprocessor Report*, Vol. 10, No. 14, Oct. 28, 1996.

[7] Hwu, W-W., Mahlke, S.A., Chen, W-Y., Chang, P-P., Warter, N.j., Bringmann, R.A., Ouellette, R.G., Hank, R.E., Kiyohara, T., Haab, G.E., Holm, J.G., and Lavery, D.M., "The superblock: An effective technique for VLIW and superscalar compilation," *Journal of Supercomputing,* 7(9-50), 1993.

[8] Lee, C-C. and Mudge, T., "Comparing techniques that reduce interference for the global history two-level dynamic branch predictor," *Tech. Report CSE-TR- -97,* Univ. of Michigan, Ann Arbor, Sep. 1997.

[9] McFarling, S. "Combining Branch Predictors," *WRL Technical Note TN-36,* Jun. 1993.

[10] Michaud, P., Seznec, A., and Uhlig, R. Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. *Proc. of the 24th Ann. Int. Symp. on Computer Architecture,* May 1997, pp. 292-303.

[11] Pan, S.T., So, K., and Rahmeh, J.T. Improving the Accuracy of Dynamic Branch Predication Using Branch Correlation. *Proc. 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems,* Oct. 1992, pp. 76-84.

[12] Pnevmatikatos, D.N. and Sohi, G.S. Guarded Execution and Branch Prediction in Dynamic ILP Processors. *Proc. of the 21st Ann. Int. Symp. on Computer Architecture,* Apr. 1994, pp. 120-129.

[13] Sechrest, S, Lee, C-C, and Mudge, T, "Correlation and Aliasing in Dynamic Branch Predictors," *Proceedings of the 23rd International Symposium on Computer Architecture,* May 1996.

[14] Smith, J.E. "A Study of Branch Prediction Strategies," *Proceedings of the 8th International Symposium on Computer Architecture,* 135-148, May 1981.

[15] SPEC CPU'95, Technical Manual, Aug. 1995.

[16] Sprangle, E., Chappell R., Alsup, M., and Patt, Y. The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference. *Proc. of the 24th Ann. Int. Symp. on Computer Architecture,* May 1997, pp. 284-291.

[17] Talcott, A.R., Nemirovsky, M., and Wood, R.C., "The Influence of Branch Prediction Table Interference on Branch Prediction Scheme Performance," *Proceedings of the 3rd International Conference on Parallel Architectures and Compilation Techniques,* Jun. 1995.

[18] Uhlig, R., Nagle, D, Mudge, T., Sechrest, S., and Emer, J. "Instruction Fetching: Coping with Code Bloat," *Proceedings of the 22th International Symposium on Computer Architecture,* Italy, Jun. 1995.

[19] Yeh, T-Y. and Patt, Y. "Two-Level Adaptive Training Branch Prediction," *Proceedings of the 24th International Symposium on Microarchitecture,* 51-61, Nov. 1991.

[20] Yeh, T-Y. and Patt, Y. "Alternative Implementations of two-level adaptive branch predictions," *Proceedings of the 19th International Symposium on Computer Architecture,* 124-134, May 1992.

[21] Yeh, T-Y. and Patt, Y. "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," *Proceedings of the 20th International Symposium on Computer Architecture,* May 1993.

[22] Young, C., Gloy, N., and Smith, M. "A Comparative Analysis of Schemes for Correlated Branch Prediction," *Proceedings of the 22th International Symposium on Computer Architecture,* Italy, Jun. 1995.