AN INTERACTIVE GRAPHICAL TRACE-DRIVEN SIMULATOR FOR TEACHING BRANCH PREDICTION IN COMPUTER ARCHITECTURE

Ciprian Radu[1], Horia Calborean[1], Adrian Crapciu[1], Arpad Gellert[1], Adrian Florea[1]

[1] "Lucian Blaga" University of Sibiu, Computer Science Department, Emil Cioran Street, No. 4, 550025 Sibiu, Romania,

{radu_ciprianro, horia.calborean, adrian.crapciu}@yahoo.com, adrian.florea@ulbsibiu.ro

Abstract

In modern superscalar microarchitectures that speculatively execute a great quantity of code, without performing branch prediction, it won't be possible to aggressively exploit instruction level parallelism from programs. Both the architectural and technological complexity of current processors emphasizes the negative impact on performance due to every branch misprediction. Due to this importance, branch prediction becomes a core topic in Computer Architecture curricula. The fast development of computer science and information technology domains, and of computer architecture especially, have determined that many software tools used not long ago in research, to be enhanced with an interactive graphical interface and to be taught in Introductory Computer Organization respectively Computer Architecture courses. The lack of simulators dedicated to branch prediction used for didactical purposes despite of plenty used in research goals, represents the starting point of this paper. The main aim of this work consists in identifying the difficult-to-predict branches, to quantify them at benchmarks level and to find the relevant information in order to reduce their numbers. Finally, we evaluate the impact of these branches on three commonly used prediction contexts (local, global and path) and their corresponding predictors, ranging from classical two-level predictors to present-day predictors (neural – *Simple Perceptron* and *Fast Path-based Perceptron*). The developed ABPS simulator provides a wide variety of configuration options. Beside statistics related to the number of difficult-to-predict branches, the simulator generates graphical results illustrating the influence of different simulation parameters (number of entries in prediction table, history length, etc.) on prediction accuracy, resource usage, etc., for every implemented predictor.

Keywords: Simulation, Advanced Microarchitectures, Branch Predictors, Benchmarks.

Presenting Author's biography

ADRIAN FLOREA obtained his MSE (1998) and his PhD in Computer Science from the 'Politehnica' University of Bucharest, Romania (2005). He is a lecturer of Computer Science and Engineering at the 'Lucian Blaga' University of Sibiu, Romania. Adrian is an active researcher in the fields of High Performance Processor Design and Dynamic Branch Prediction. He published over 19 scientifically papers about Computer Architecture in some prestigious journals (ISI-INSPEC) and international conferences. He got "*Tudor Tanasescu*" Romanian Academy Award 2005, for the book entitled "Microarchitectures simulation and optimization" (in Romanian). His Web-page can be found at http://webspace.ulbsibiu.ro/adrian.florea/html/ .

1 Introduction

Since the simulation community – usually very broad – is not necessarily familiar with the branch prediction problem, we present a short introduction into this domain. Computers made important progress and microprocessors (CPUs) are the main responsible for this. Thus, it is a stringent necessity to improve the current computer architecture performance both from a quantitative and qualitative viewpoint. The technological trends refer to increasing integration degree of transistors on chip, increasing the processors frequency, reducing the main memory access time, reducing the hardware implementation cost at same power consumption or same memory size, etc. The architectural tendencies, pursuit exploiting and increasing instruction level parallelism both through static and dynamic techniques, in order to overcome the control-flow and data-flow bottleneck. Both the architectural complexity of current processors (deep pipeline structures – 20 at INTEL Pentium4 and wide width instructions issue) and technological complexity (higher processing frequency – greater than 3.3 GHz at same processor) emphasizing the negative impact on performance due to every branch misprediction [11]. Branch instructions activate at control-flow level generating performance loss by unknowing in the instruction fetch stage the direction and the target of branch. Thus, the modern architectures should incorporate very efficacious prediction schemes.

One of the first approaches in hardware branch prediction consists in Branch Target Buffer (BTB) structures [6]. BTB is a small, associative memory, integrated on chip that retains the addresses of recently executed branches, their targets and optionally other information (e.g. target opcode). Due to some intrinsic limitations, BTB's accuracies are limited on some benchmarks having unpropitious characteristics (e.g. correlated branches). Among first processors that implemented BTB structures are Intel Pentium [18], AMD K5 and Alpha 21064.

In order to improve BTB's efficiency, Yeh and Patt (1992) generalized it through a new approach called Two Level Adaptive Branch Prediction. (TLABP) According to [16], the TLABP uses two distinct levels of branch history information to make predictions. The first level consists in the History Register (HR) that contains the last k branches encountered (taken / not taken) or the last k occurrences of the same branch instruction. The second level consists in the branch behavior of the last occurrences of the specific pattern of these branches. A Pattern History Table (PHT) that contains essentially the branch prediction automaton (usually 2 - bit saturating counters) implements it. HR shifts left with a binary position when updated according to the actual branch behavior (taken=1/ not taken=0). There is a corresponding entry in the PHT for each of the 2^k HR's patterns. Intel Pentium Pro, Pentium II [18] and AMD K7 [3] are examples of processors that incorporate a two-level predictor.

In order to obtain a greater prediction accuracy the nowadays processors use hybrid prediction structures, combining two (or more) tables, one correlated with local history of the predicted branch (PAg predictor) and other correlated with global history of the predicted branch (GAg predictor). The selection between two predictions is made using a confidence table that records the dynamic behavior of each predictor. The processor Alpha 21264 embeds a hybrid predictor having a local predictor with 1024 entries (keeping a local history of 10 bits) and a global predictor with 4096 entries reaching to almost 95% of prediction accuracy [4].

The most accurate single-component branch predictors in the literature are neural branch predictors [4, 5]. Their main advantages consist in the possibility of using longer correlation information at linear cost. The *Perceptron* predictor – the simplest neural branch predictor – keeps a table of *weights vectors* (small integers that are learned through the *perceptron* learning rule) [4]. As in global two-level adaptive branch prediction, a shift register records a global history of outcomes of conditional branches, recording *true* for *taken*, or *false* for *not taken*. To predict a branch outcome, a weights vector is selected by indexing the table with the branch address modulo the number of weights vectors. The dot product of the selected vector and the global history register is computed, where *true* in the history represents 1 and *false* represents -1. If the dot product is at least 0, then the branch is predicted taken, otherwise it is predicted not taken. Once the perceptron's output has been computed, the training algorithm starts: it increments the *i*-th correlation weight when the branch outcome agrees with the *i*-th bit from global branch history shift register and decrements the weight otherwise. Unfortunately, the high latency of the perceptron predictor and impossibility to predict the linearly inseparable branches makes it impractical yet for hardware implementation. In order to reduce the prediction latency, the *Fast Path-based Perceptron* [5] chooses its weights for generating a prediction according to the current branch's path, rather than according to the branch's PC and history register. The prediction latency is hidden, due to the speculative calculation of the perceptron's output. Intel includes the perceptron predictor in one of its IA-64 simulators for researching future microarchitectures [4].

Vintan et al. proved that a branch in a certain dynamic context is difficult-to-predict if it is unbiased and the outcomes are shuffled [14]. In other words, a dynamic branch instruction is unpredictable with a given prediction information if it is unbiased in the considered dynamic context and the behavior in that certain context cannot be modeled through Markov stochastic processes of any order. Based on Vintan's methodology, in this paper we identify unbiased branches by repeating various and different length prediction contexts. We show how the frequency of unbiased branches decreases, when the length of context information extends. Also, we determine the impact of unbiased branches over prediction accuracy and processing performance.

2 Simulation Methodology. Benchmarks

After more than two decades, the researches from computer science domain got the conclusion that simulators have become an integral part of the computer architecture research and design process [17]. Their most important advantages, comparing with real processors, are low implementation cost, development time, flexibility and extensibility allowing the architects to quickly evaluate the performance of a wide range of architectures and to quantify the efficacy of every enhancement. Besides its importance proved in computer architecture research field, in the latest time, simulators have been extensively employed as a valuable pedagogical tool as they enable students to visualize how microarchitecture components work and interact with each other. For example, at last Workshop on Computer Architecture Education held in conjunction with the 33rd International Symposium on Computer Architecture (the best conference in computer architecture domain in the world), two papers aim at fundamental topics of computer architecture curricula: processor – cache interface in a RISC architecture (MIPS) [8] and power and performance analysis in superscalar out-of-order architecture [10].

In this work we implement the ABPS (Advanced Branch Prediction Simulator), an interactive graphical trace-driven simulator for teaching branch prediction. The ABPS simulator is currently used in undergraduate and graduate courses / laboratories in (Advanced) Computer Architecture at "Lucian Blaga" University of Sibiu. The simulator code is open source and can be found at http://webspace.ulbsibiu.ro/adrian.florea/html/simulatoare/simulatoare.htm (or http://sourceforge.net/projects/abps).

Related to the first part of our investigation – identifying the difficult-to-predict branches and quantifying them on testing programs, we used the traces obtained based on the eight C Stanford integer benchmarks, designed by Professor John Hennessy (Stanford University), to be computationally intensive and representative of non - numeric code while at the same time being compact. All these benchmarks were compiled by the *HSA gnu C* compiler, which targets the HSA (Hatfield Superscalar Architecture) instruction set. A dedicated HSA simulator [13] that generates the corresponding traces simulated the resulted HSA object code. The average instruction number is about 273.000 and the average percentage of total instructions that are branches is about 18%, with about 76% of them being taken. Some of these benchmarks are well known as very difficult to be predicted. For example, as Mudge et al. proved very clearly [7], 75% accuracy could be an ultimate limit on the *"quick-sort"* benchmark. Following our aims, we developed an original dedicated trace-driven simulator that uses the above-mentioned traces.

For the second part, in which we investigate the present-day branch prediction schemes we extend the space of exploration, performing trace-driven simulation on a collection of 17 programs (having 1 million dynamic branch instructions each) from different versions of SPEC benchmarks [12]. We simulate all of the SPEC CPU2000 integer benchmarks, and all of the SPEC CPU95 integer benchmarks that are not duplicated in SPEC CPU2000. The benchmarks are compiled with the *CompaQ GEM* compiler with the optimization flags *-fast -O4 -arch ev6* [2]. All these benchmarks cover a lot of applications ranging from compression to word processing, from compilers and architectures to games enhanced with artificial intelligence, etc. We choose to simulate different version of benchmarks (Stanford and SPEC) in order to discover how these different testing programs influence the neural branch predictors' micro-architectural features.

From a pedagogical point of view, the proposed tool benefits the learning process because it helps students to observe the influence of each parameter on the simulation model. The simulator provides a wider variety of configuration options. Thus, it can be determined how prediction accuracy varies with input parameters (number of entries in prediction tables, history length, number of bits for weights representation, etc). The ABPS simulator assures three of the features specific to almost high-performance standard simulators: free availability for use, extensibility and portability. Full inheritance and polymorphism is used, allowing for ease of extension in the future, adding new functionalities.

High prediction accuracy is vital especially in the case of multiple instruction issue processors. It is important for students to understand how to investigate architectures that are optimized in terms of cost, performance (prediction accuracy, execution rate), and power consumption. Projects designed around ABPS simulator are used to teach students concepts related to the unbiased branch, state of the art branch predictors, branch prediction constraints and limits of instruction level parallelism. This version of simulator uses only an analytical model to determine the impact of unbiased branch and branch misprediction on global processing performance [15]. In his model, related to a superscalar processor, Vintan ignores stalls like cache misses and bus conflicts focalizing only about the penalty introduced by branch misprediction. In their assignments, students are asked to explore architecture configurations extending them for optimizing the power, performance, or both within a given chip area budget (based on other simulation tools – CACTI, WATTCH [9, 1]).

3 The Functional Kernel of the Simulator

The realized simulator must remove the bottlenecks that limit the processor performance and search for possible changes (architectural or optimization techniques) for improving it. Providing a highly parameterized model for every microarchitectural instance, the performance obtained by simulation will represent a quick feedback mechanism related to proposed changes. The simulator execution consists in the following sequential steps:

1) *Configuring the microarchitecture with the input parameters including the benchmarks*;
2) *Initialization phase* (prediction tables, local/global history registers, etc.);
3) *Starting the trace processing and computing the simulation metrics.*

3.1 Identifying unbiased branches

The majority of branches demonstrate a bias to either the taken or the not-taken path which means branches are highly polarized towards a specific prediction context (a local prediction context, a global prediction context or a path-based prediction context) and such polarized branches are relatively easy-to-predict. However, a minority of branches (6% to 24%, depending on the used history length [14]) shows a low degree of polarization since they tend to shuffle between taken and not-taken and are therefore unbiased and difficult-to-predict. The *Detector* kernel of ABPS finds the unbiased branches (those that have their polarization index – the percentage of “not taken” or “taken” branch instances corresponding to a certain context – lower than a *polarization degree*, set prior the simulation) and quantifies their number. Repeating the unbiased branches detection methodology for a length-ordered set of contexts it could be observed how the number of unbiased branches decreases.

3.2 Branch prediction simulators

The prediction process supposes accessing the tables for every instruction from traces and establishing the prediction function of associated prediction automaton or perceptron computed output. After branch’s resolution, it starts the updating algorithm (every good prediction increases the automatons state or perceptron weights, otherwise decreases the same parameters). The automatons are implemented as saturating counters and, in the neural predictors’ case, the threshold keeps from overtraining, permitting the perceptron to adapt quickly at every changing behavior.

4 The User Interface

To run the ABPS simulator, the host computer must have installed the *jre-1_5* (or higher) or *jdk-1_5* (or higher), for future development. ABPS is written in JAVA, thus it is platform independent. For properly use of ABPS simulator it should be accomplished some system requirements. Thus, it is recommended to have a processor with at least 1 GHz frequency. Otherwise, due to JVM (Java Virtual Machine), the simulation time, especially on SPEC2000 benchmarks, risks to become prohibit. The RAM memory recommended is 256Mbytes. Since we can represent on the same chart up to 17 benchmarks, to have a good view, it is required a 1024x768 minimum screen resolution.

The ABPS simulator is organized around a main window that contains two panels. The left one is used to **configure** (initialize all requested parameters) and **control** the simulation. The right panel is based on two tabs – one that shows every **simulations’ results** in text format, and another, that permits to generate graphical charts, illustrating the influence of different simulation parameters on metrics like unbiased branches percentage, prediction accuracy, processing rate. The left panel is divided in two parts: the upper part contains the available benchmarks. The lower part of left panel contains two tabs ***Detector / Predictor***, each having its own configuring parameters. The inputs for *Detector* are: the *global history length* – GH, the *local history length* – LH, a flag that shows if path information correlation is used, and the *polarization degree* of each context instance. Figure 1 shows the simulation results when Detector tab was selected. The *Predictor* tab contains each predictor implemented (*GAg, PAg, PAp* and *Perceptron*). Each predictor can predict all branches or only unbiased branches. If the second choice is made, the simulator applies first the *Detector* phase, hidden for user. After determining the unbiased branches percentage it can be computed the performance loss comparative with an equivalent multiple instruction issue processor having an ideal branch predictor.

If the user selected from Configuration panel the *Predictors* / *Perceptron* tab (*Simple* or *Fast Path-based*) and in the Results panel only simple execution (not charts generating), the simulation results consist in four important metrics. The **prediction accuracy** is the number of correct predictions divided to total number of dynamic branches. We compute also a **confidence** metric that represents the total cases when the prediction was correct and the perceptron didn’t need to be trained (the magnitude of perceptron output was greater than threshold) divided to total number of correct predictions. While the first two have impact on processor’s performance, the next two metrics have direct influence on transistors’ budget and integration area (the **number of perceptrons used** in prediction process and respectively the **saturation degree** of perceptrons). The saturation degree represents the percentage of cases when the weights of perceptrons can’t be increased / decreased because they are saturated. If the last two metrics are quite low means that the perceptrons are underused. The **prediction accuracy** and the usage degree of prediction table are computed also in the case of two-level predictors. The

Charts tab offers the possibility to illustrate graphical simulation results. From the two list boxes, the user can select which metrics are to be measured and which input parameter varies on all selected benchmarks. An interesting chart shows the Issue Rate (IR) relative speedup obtained by growing the context length. We used the formula [IR(L)–IR(16)] / IR(16), for computing IR relative speedup, where L is the context's length, L∈{20, 24, 28, 32}). The last group of columns represents the average. The chart can be saved in *png* format. Figure 2 illustrates how prediction accuracy varies with global history length when *fast path-based perceptron* predictor is used, on 5 critical SPEC2000 benchmarks.

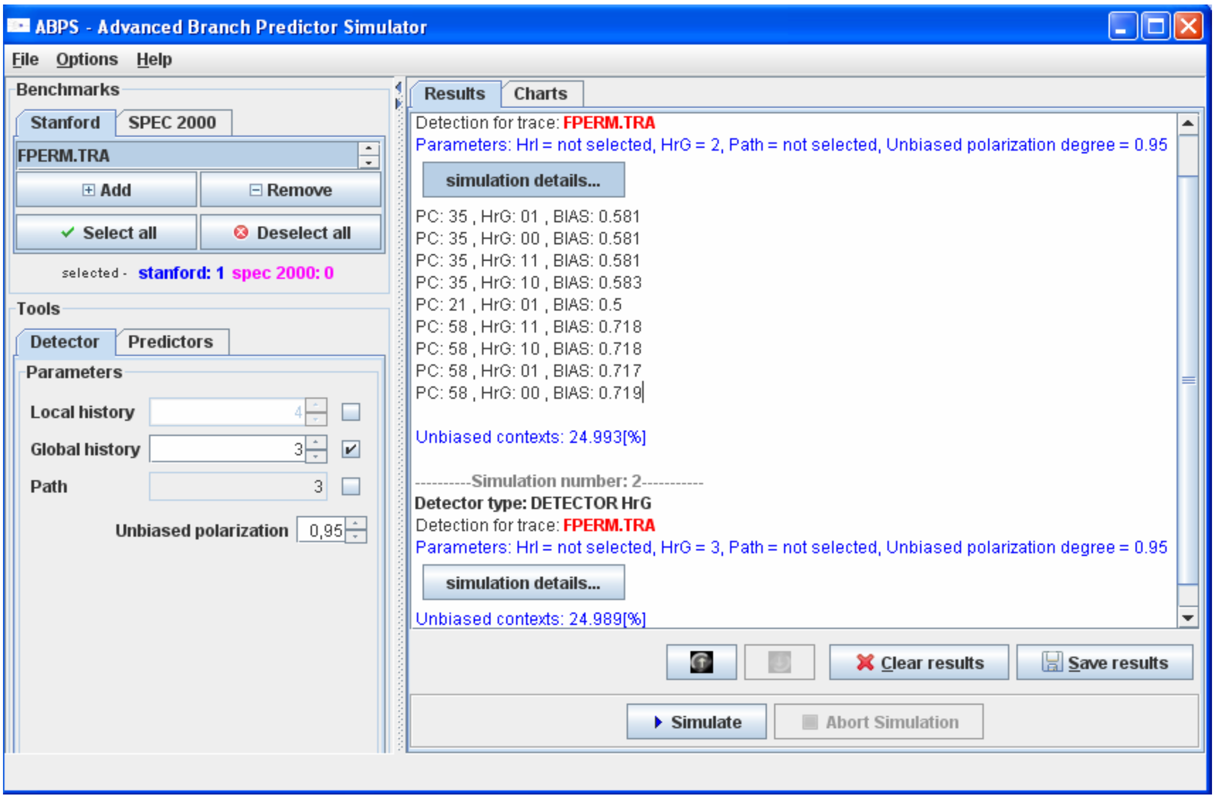

Figure 1. ABPS simulator – unbiased branches detection

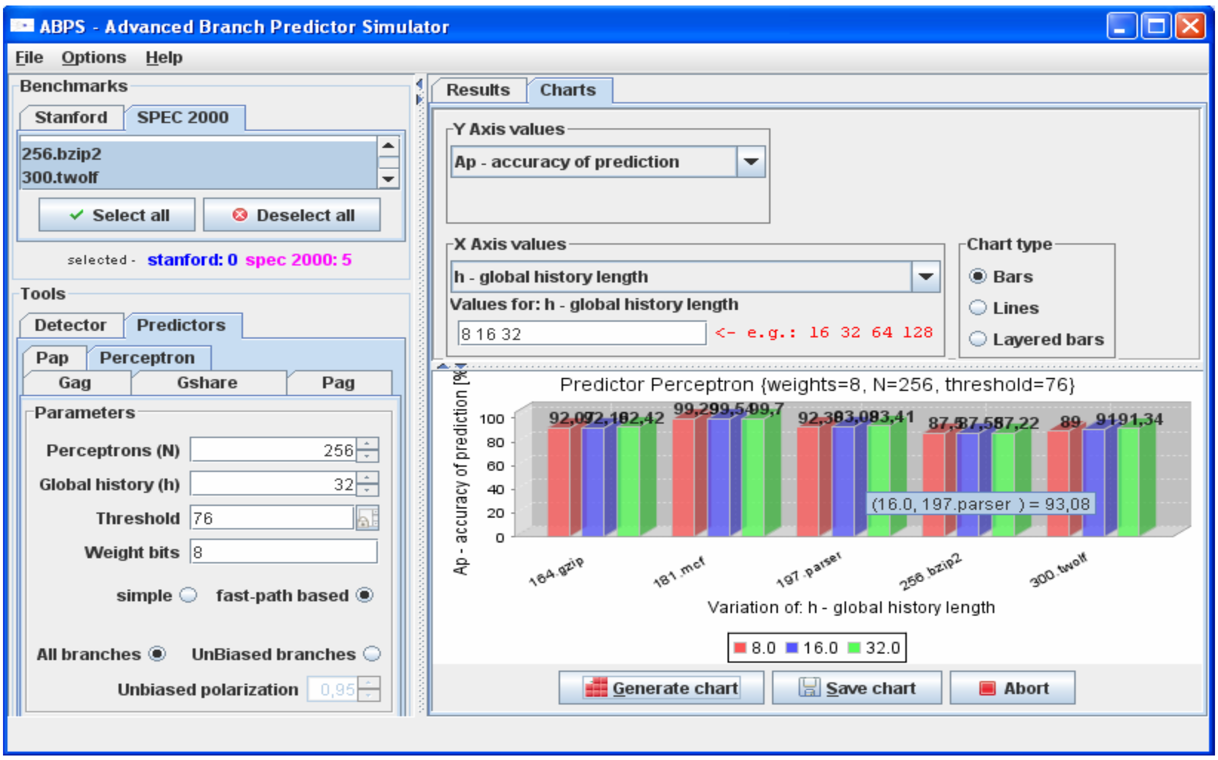

Figure 2. ABPS simulator – variation of prediction accuracy with global history length

5 Acknowledgments

This work was supported in part by the Romanian National University Research Council grants CNCSIS 71/2004. Also our gratitude to Dr. C. Egan from the University of Hertfordshire, UK, for providing HSA Stanford traces and for his encouragement related to our Instruction Level Parallelism research.

6 Conclusions and Further Work

The classical approach in teaching branch prediction is based largely on oral communication of professors that spent a lot of time in computer architecture research or, using paper and pencil to follow the sequences of accesses in predictions tables, computing different rules (e.g. perceptron) for prediction and updating. Our approach represents a formative necessity since computer architectures are mainly approached in a descriptive manner. Through our approach, students have the opportunities to be creative / innovative in computer architecture or in other fundamental research / didactical domains of computer science even in countries not very developed from economical point of view. Based on highly parameterized developed simulation tools, students can understand more in depth the theoretical concepts related to branch prediction constraints, limits of instruction level parallelism. It could be observed the different benchmarks' influence on every proposed architectural innovation. With ABPS simulator we identify unbiased branches. Repeating the detection methodology for a length-ordered set of contexts it could be observed how the number of unbiased branches decreases, from tested benchmarks. Another facility consists on running a plenty of branch predictors, from classical two-level up to neural state-of-the art, varying the most important parameters and illustrating simulations' results. Also important, ABPS permits the migration of some mature actual scientific problems to students' understanding level.

For further work we are concerned to the necessity of an efficient hardware branch predictor from power consumption and performance criterions, within a given chip area budget. Very high prediction accuracy is necessary, because taking into account the multiple-instruction-issue processors characteristics as pipeline depth or issue rates, even a prediction miss rate of a few percent involves a substantial performance loss. Also, we intend to extend the ABPS simulator with functional network characteristics, allowing a distributed simulation process in a client-server manner, useful due to the time consuming simulations.

7 References

[1] Brooks D., Tiwari V., Martonosi M., *Wattch: a framework for architectural-level power analysis and optimizations*. In Annual International Symposium on Computer Architecture, pages 83–94, 2000.

[2] Cohn R., Lowney P. G., *Design and Analysis of Profile-Based Optimization in Compaq's Compilation Tools for Alpha*, Journal of Instruction-Level Parallelism nr 3, 2000, pg. 1—25.

[3] Diefendorff K., *K7 challenges Intel. Microprocessor Report*, 12(14), October, 1998.

[4] Jiménez D., Lin C., *Neural Methods for Dynamic Branch Prediction*, ACM Transactions on Computer Systems, Vol. 20, New York, USA, November 2002.

[5] Jiménez D., *Fast Path-Based Neural Branch Prediction*, Proceedings of the 36th Annual International Symposium on Microarchitecture, December 2003.

[6] Lee J. K. F., Smith A. J., *Branch prediction strategies and branch target buffer design*, IEEE Computer Magazine, pp. 6-22, January 1984.

[7] Mudge T.N., et al., *Limits of Branch prediction,* Technical Report, Electrical Engineering and Computer Science Department, The University of Michigan, Ann Arbor, Michigan, USA, 1996.

[8] Petit S., Tomás N., Sahuquillo J., Pont A., *An Execution-Driven Simulation Tool for Teaching Cache Memories in Introductory Computer Organization Courses*, Workshop on Computer Architecture Education, Boston, 2006.

[9] Shivakumar P., Jouppi N. P., *CACTI 3.0: An Integrated Cache Timing, Power, and Area Model*, WRL Technical Report 2001/2.

[10] Smullen C.W., Taha T.M., *PSATSim: An Interactive Graphical Superscalar Architecture Simulator for Power and Performance Analysis*, Workshop on Computer Architecture Education, Boston, 2006.

[11] Sprangle E., Carmean D. – *Increasing processor performance by implementing deeper pipelines*, 29th International Symposium on Computer Architecture, Anchorage, Alaska, May 25 - 29, 2002.

[12] SPEC2000, *The SPEC benchmark programs*, http://www.spec.org.

[13] Steven G. B. et al - *A Superscalar Architecture to Exploit Instruction Level Parallelism*, Proceedings of the Euromicro Conference, Prague, 1996.

[14] Vintan L., Gellert A., Florea A., Oancea M., Egan C., *Understanding Prediction Limits through Unbiased Branches*, Lecture Notes in Computer Science 4186 LNCS, pp. 480-487, 2006.

[15] Vintan L. Prediction Techniques in Advanced Computing Architectures (in English), MatrixRom Publishing House, Bucharest, 2007 (292 pg.)

[16] Yeh T., Patt Y., *Alternative Implementations of Two-Level Adaptive Branch Prediction*, 19th Annual International Symposium on Computer Architecture, 1992.

[17] Yi J.J., Lilja D.J., *Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies and Recommendations*, IEEE Transactions on Computers, Vol. 55, No. 3, March 2006, pp. 268-280.

[18] *Pentium 4, Part I: the history*, http://www.digit-life.com/articles/pentium4/index.html

