THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 9, Number 3/2008, pp. 000–000

RANDOM DEGREES OF UNBIASED BRANCHES

Lucian N. VINŢAN, Adrian FLOREA, Arpad GELLERT

Computer Science Department, "Lucian Blaga" University of Sibiu, Emil Cioran Street, No. 4, Sibiu 550025, Romania
E-mail: {lucian.vintan, adrian.florea, arpad.gellert}@ulbsibiu.ro

In our previous published research we discovered some very difficult to predict branches, called unbiased branches that have a "random" dynamical behavior. We developed some improved state of the art branch predictors to predict successfully unbiased branches. Even these powerful predictors obtained very modest average prediction accuracies on the unbiased branches while their global average prediction accuracies are high. These unbiased branches still restrict the ceiling of dynamic branch prediction and therefore accurately predicting unbiased branches remains an open problem. Starting from this technical challenge, we tried to understand in more depth what randomness is. Based on a hybrid mathematical and computer science approach we mainly developed some degrees of random associated to a branch in order to understand deeply what an unbiased branch is. These metrics are program's Kolmogorov complexity, compression rate, discrete entropy and HMM prediction's accuracy, that are useful for characterizing strings of symbols and particularly, our unbiased branches' behavior. All these random degree metrics could effectively help the computer architect to better understand branches' predictability, and also if the branch predictor should be improved related to the unbiased branches.

Keywords: processor architecture, branch prediction, unbiased branches, discrete entropy, random degree

1. INTRODUCTION

In two of our previous papers [1, 2] we found a minority of dynamic branches showing a low degree of polarization towards a specific prediction context since they tend to shuffle between taken and not-taken and are very difficult-to-predict. We called them unbiased branches and we have demonstrated that actual programs have a significant fraction of such difficult to predict branches that severely affect prediction accuracy. Taking into account that the actual branch predictors exploit limited prediction contexts, we demonstrated that use of more prediction contexts is required to further improve prediction accuracies.

Unbiased branches are unpredictable because their behavior's nature is still not deeply understood, based on a qualitative and quantitative approach. Rigorously defining and understanding unbiased branches means to know rigorously what randomness is. Without effectively understanding their "random" behavior, we cannot expect to develop accurate predictors.

A pragmatic aim consists in finding some deterministic hidden information that could reduce the unbiased branches' entropy. This is extremely difficult at least from two reasons: first, due to the enormous complexity of the benchmarks' dynamic behavior and, second, due to the fact that the simulated object code obviously has far less semantics comparing with the HLL program. However, we consider that our developed random degrees could indicate the chance for uncovering this new relevant information. A high random degree might indicate a huge complexity and therefore, small chances to discover the right useful information.

Based on a combined mathematical and computer science approach, in this paper we mainly developed some degrees of random associated to a certain branch in order to better understand what an unbiased branch is. According to our previous work, the percentages of unbiased branches are quite significant, depending on the different used contexts and their lengths, giving a new research challenge and a useful niche for branch prediction research. Through this paper we showed that these difficult predictable branches cannot be well-predicted even using efficient state of the art predictors. In this scope, we specially developed two idealized powerful branch predictors: an improved idealized piecewise linear branch predictor and a Hidden Markov Model (HMM) branch predictor. Unbiased branches need some specific efficient predictors that are using some new, more relevant prediction information. Finding a new relevant context to reduce significantly the number of unbiased shuffled branches remains an open problem.

Further, we tried to understand in more depth what randomness, from a mathematical point of view is. We started from the following fundamental question: Could a deterministic program generate some branches having a "random behavior"? Moreover, if a branch outcome is "completely random", how should any algorithm be able to predict it? Obviously, the unbiased branch behavior is influenced by the benchmarks' input data files, too.

As we show in this paper the answer is not simple. Based on our bibliographical research focused on understanding what a random string is, finally we proposed and developed some random degree metrics, like program's Kolmogorov complexity, compression rate, discrete entropy and HMM-based prediction accuracy, that might be useful for characterizing strings of symbols and particularly, our unbiased branches' behavior. All these random degree metrics could really help the computer architect to understand in more depth the nature of a certain branch and also if the branch predictor should be improved in order to accurately predict even the corresponding unbiased branches. Our developed branches' random degrees could effectively help in quantifying program's predictability, too.

2. RELATED WORK

V. Desmet compared in her PhD thesis [3] different branch prediction information, including local/global branch history and path information, from the entropy point of view. An important difference between our approach and Desmet's is that we measured per dynamic branch-context polarization and presented the average percentage of branch contexts having polarization less than 0.95, while Desmet measured per branch entropy and presented the average entropy.

The piecewise linear branch prediction is a generalization of perceptron branch prediction, which uses a single linear function for a given branch, and respectively path-based neural branch prediction, which uses a single global piecewise-linear function to predict all branches [4, 5, 6]. The piecewise linear branch predictors use an aggregated piecewise-linear function for a given branch, exploiting in this way different paths that lead to the same branch in order to predict – otherwise linearly inseparable – branches. Taken as a whole, the linear functions used to predict a branch form a piecewise-linear surface separating paths that lead to predicted taken branches from paths that lead to predicted not taken branches.

Seznec recently developed perhaps the most powerful idealistic/realistic branch predictors. His idealistic hybrid GTL predictor is composed by 3 distinct branch predictors (TAGE, GEHL and Loop) exploiting very deep history correlations and a metapredictor derived from the skewed predictor [7]. The GEHL predictor is the main component being the most accurate. In Section 3 we also used it in predicting our unbiased branches. In [8] it is presented a realistic branch predictor called L-TAGE consisting of a 13-component TAGE predictor and a loop predictor. Both these predictors won the 2nd Championship Branch Prediction obtaining [9] at average 3.314 mispredictions/KI. Even if these predictors are the best known branch predictors they are using the same limited prediction information (branch address, global/local histories and path) that is insufficient for reducing unbiased branches entropy and, therefore, for accurately predicting them.

In [10] the authors are focused on hard-to-predict branches that depend on long-latency cache-missing loads. These dependences involve high-penalty mispredictions becoming serious performance obstacles and causing significant performance degradation in executing instructions from wrong paths. The authors demonstrated that in some memory intensive workloads the address of the data rather than the data value is sufficient to determine the branch outcome. Consequently, a misprediction can be resolved much more efficiently when the data access results in a long-latency cache miss. The authors called such locality "address-branch correlation" and they showed that certain memory intensive benchmarks, especially those with heavy pointer chasing, exhibit this locality. It is developed a dedicated predictor that dynamically captures correlations between producer load addresses and consumer branch outcomes. Address-branch correlation based predictions are generated when a producer address is known. This predictor significantly reduces misprediction rate, execution time and energy consumption.

3. UNDERSTANDING AND PREDICTING UNBIASED BRANCHES

In our previous work [1] we identified the unbiased branches in the SPEC 2000 benchmark suite [11] for different prediction contexts consisting in local history (LH) and global history (GH). The unbiased branch concept is related to an entropic shuffled branch behavior and not to its prediction accuracy through a certain branch predictor. As we will further show unbiased branches are poorly predictable. In our experiments, we concentrated only on benchmarks with a fraction of unbiased branches greater than 1%. Following this methodology, 6 integer benchmarks fulfilled this condition. Therefore, in this work we are simulated only these difficult predictable benchmarks (gzip, bzip, mcf, parser, twolf, gcc). Finally, using a 56 bits LH & GH history context, 6.19% of dynamic branches are unbiased and, therefore, unpredictable.

We showed that the best state of the art branch predictors (Markovian, Prediction by Partial Matching, and neural) [9, 12] are obtaining very low prediction accuracies on unbiased branches, at average about 70%. The same predictors are predicting a "normal" branch with accuracies ranging between 95% and 99%. Our experimental results showed that the unbiased branches cannot be predicted accurately even with the actual most powerful branch predictors. The highest average prediction accuracy on the unbiased branches, of 77.30%, was provided by the idealized piecewise linear branch predictor [6]. This low prediction rate is understandable taking into account that even a neural predictor cannot effectively learn unbiased branches. As a comparison, the same predictor obtained far better average prediction accuracy, of 94.92%, on all branches.

Our second idea in order to reduce unbiased branches' number was to find new relevant information that could reduce their entropy, making them more predictable. Representing the problem in a superior feature space dimension is a general well-known method in solving many Computer Science classification and/or prediction problems. Unfortunately, despite an interesting instructive research in this sense, we practically failed in finding efficient new information [2]. In Figure 1, we present our most recently developed scheme to predict the unbiased branches.

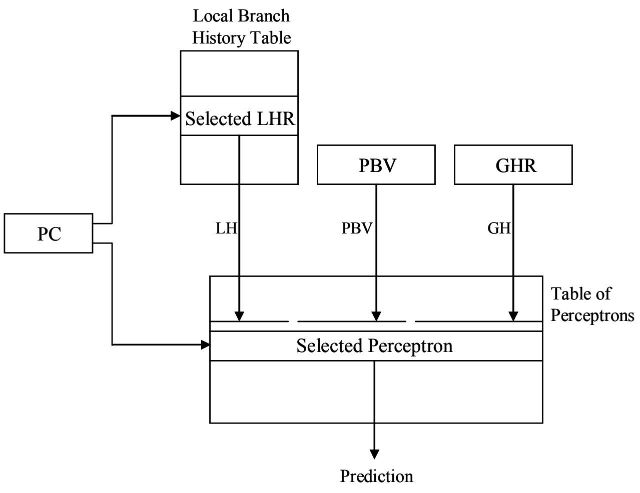

Figure 1. Piecewise linear branch predictor using the previous dynamic branch's condition value

We improved the idealized piecewise linear branch predictor by using the previous dynamic branch's condition value (PBV) as an additional prediction information. The lower part of the branch's address (PC) selects a perceptron from the table of perceptrons and a local history register from the local branch history table. Thus, local and global branch histories together with the PBV value are used as inputs for the selected perceptron in order to generate a prediction. The three indexes used within the weight selection mechanism are obtained through a hash function that uses three prime numbers, as follows [5]:

$$index_{GH}^{i} = [(PC \cdot 511387) \oplus (PC_{i-1} \cdot 660509) \oplus (i \cdot 1289381)] \bmod NWeights$$
$$index_{LH}^{j} = [(PC \cdot 511387) \oplus (j \cdot 1289381)] \bmod NWeights$$
$$index_{PBV}^{k} = [(PC \cdot 511387) \oplus (k \cdot 1289381)] \bmod NWeights$$

where $i = \overline{1, GHlength}$, $j = \overline{1, LHlength}$, $k = \overline{LHlength + 1, LHlength + PBVlength}$, and *NWeights* is the total number of weights (parameter varied in our simulations between 8590 and 30713). PC_{i-1} represents the previous $(i-1)^{th}$ branch's PC, belonging to the current branch's path. A certain prediction is generated using ($GHlength + LHlength + PBVlength$) number of selected weights. These weights were selected from a table containing *NWeights* weights. The first two relations were used according to Jimenez's simulator proposals [5] while we introduced the third one, according to the new introduced PBV information.

Figure 2 presents the prediction accuracies obtained with the idealized piecewise linear branch predictor using the global PBV as additional prediction information. The first two bars represent the prediction accuracies obtained with the idealized piecewise linear branch predictor (PW). The rest of the bars were obtained using PBV as additional prediction information, varying the number of weights (from 8590 to 30713).

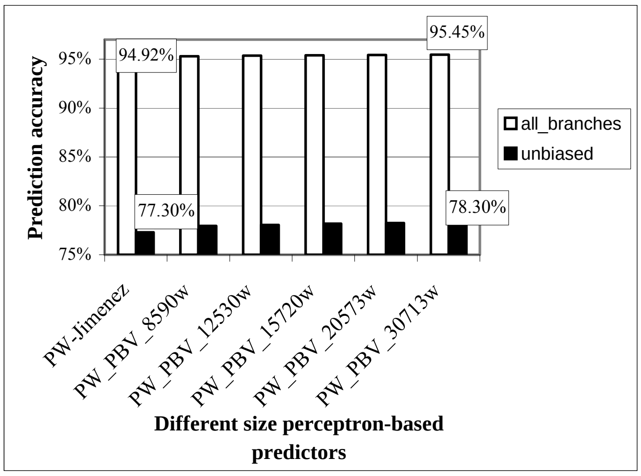

Figure 2. The prediction accuracies obtained with the *piecewise linear branch predictor* using the global PBV

The PBV value determines the improvement of unbiased branch prediction accuracy, overcoming with at least 1% the best state of the art predictor's performance. Even if the improvement seems less significant, it is very clear how this small percentage improves the global prediction accuracy (with more than 0.53%).

Therefore, the unbiased branches behavior is practically unpredictable. Why this? Are these special branches unpredictable due to some relevant information misses or are they "intrinsic random"? However, they were obtained by compiling some deterministic programs; therefore, they were not randomly generated. But...what is random? During the next paragraph, we will try to understand random strings of symbols from a mathematical point of view in order to propose some concrete metrics characterizing them. These metrics could help us to better understand and analyze the unbiased branches behavior and their predictability.

4. WHAT IS A RANDOM SEQUENCE?

The questions are: *is it possible to give an intrinsic or ontological definition of a random string of symbols*? *Could generate a deterministic program a "random" sequence*? Mathematicians show that for strings it is only possible to develop a notion of randomness degree, the difference between random and nonrandom being therefore quite fuzzy.

There is a strong logic connection between the concept of randomness and computability theory. It is natural to consider that any string of symbols generated by an algorithm is not random. For an in-depth rigorous definition of randomness, it is necessary to use the fertile Turing Machine (TM) concept. Any binary input sequence in a TM belongs to the so-called Finite Binary set (FB) and it codifies the input data. For $\forall x \in FB$, if the TM reaches its final state, it generates the corresponding output sequence TM(x). The set of all TMs is a countable (infinite) set; therefore, TM set can be put in a one to one correspondence with the natural number set N.

Now each string in FB is a binary representation of a positive integer through an encoding function c: FB →N. A partial function f: N→N is Turing computable if there is a TM such that, for every *n* in the domain of *f*, there is an input $x \in FB$ with n = c(x) for which the machine eventually stops and such that the output TM(x) satisfies f(n) = c(TM(x)). From the countability of the collection of all TMs, it follows immediately that the set of partial Turing computable functions is a countable subset of the much larger uncountable set of all partial functions from N to N. In this sense, very few functions are Turing computable even if the number of these computable functions is infinite. This means that the Turing non-computable function set is uncountable. It is well-known the Church-Turing thesis saying that for any algorithm (finite steps procedure) there is an equivalent

TM [13].

Returning to the random binary string's definition problem, as we already stated, it is obvious that such a string must not be generated through an algorithm (i.e., Turing computable function). Thus, any random binary string is one generated by a non-computable Turing function. Taking into account that non-computable Turing function set is an uncountable set it involves that random binary strings set is uncountable, too. Unfortunately, this rigorous definition of a random sequence is useless because it cannot effectively generate any concrete random string. Rigorously defining and effectively generating random sequences seems to be an open problem for the actual mathematics and also for the general studies related to cognitive and noetic behavior.

R. Solomonoff, A. Kolmogorov, and G. Chaitin proposed the practical idea of randomness as incompressibility independently in the sixties of previous century. The main intuition is that a string is random if it cannot be "described" more efficiently than by giving the whole string itself. Thus, a string is random if it is algorithmically incompressible or irreducible. According to this approach, a string is random if no computer program of size substantially smaller than the string itself can generate or describe it. This is the notion of program size algorithmic or Kolmogorov's complexity. Obviously, its concrete value depends on the particular formal language that implements the generator algorithm. Because of this practical approach of randomness we will propose the compression rate as a random degree of a string of symbols.

Considering a sequence S of symbols belonging to the set $X=\{X_1X_2 \ldots X_k\}$, another practical approach for characterizing the randomness of S might be based on its entropy:

$$\mathrm{E}(S) = -\sum_{i=1}^{k} P(Xi)\log_2 P(Xi) \geq 0 \tag{1}$$

Obviously, its maximum ($\log_2 k$) is obtained for symbols of equal probabilities in S. Therefore, we propose a random degree (RD) for a branch's output sequence given by the formula:

$$\mathrm{RD}(S) = \mathrm{D}(S)\cdot\mathrm{E}(S) \in [0, \log_2 k] \tag{2}$$

In expression (2), D(S) represents the shuffle degree and it is defined in the following formula:

$$D(S_i) = \begin{cases} 0, & n_t = 0 \\ \dfrac{n_t}{2\cdot\min(NT, T)}, & n_t > 0 \end{cases} \tag{3}$$

where:

- n_t = number of branch outcome transitions in a certain context S_i;
- $2\cdot\min(NT, T)$ = maximum number of possible transitions;
- k = number of distinct contexts, $k \leq 2^p$, where p is the length of the binary context;
- if $D(S_i) \to 1, (\forall)i = 1, 2, ..., k$, then the behavior of the branch in context S_i is "contradictory";
- if $D(S_i) \to 0, (\forall)i = 1, 2, ..., k$, then the behavior of the branch in context S_i is constant.

A high value for RD might involve a high random degree. Of course, our proposed RD(S) is not theoretically perfect. As an example, the sequence 01010101010101... maximizes both D and E but despite of this fact it is very deterministic and, therefore, very predictable.

As we previously emphasized, new relevant information could reduce the string's entropy and thus its random degree. Unfortunately, this information might be very difficult or even impossible to be found. Consequently, we think it would be interesting trying to predict a sequence using HMMs like those developed in [14, 15]. A HMM is a doubly embedded stochastic process with a hidden stochastic process that can only be observed through another set of stochastic processes that generate the sequence of observable symbols. HMM predictors are very powerful adaptive stochastic models. Our hypothesis is that HMMs could compensate relevant information miss-knowledge through its underlying stochastic process that is not observable. Therefore, we will propose HMM prediction accuracy as another practical metric for calculating the random degree associated with a sequence of symbols. Of course, all these random degree metrics will be applied to our unbiased branches behaviors in order to estimate how much random they are.

5. RANDOM DEGREE METRICS FOR CHARACTERIZING UNBIASED BRANCHES BEHAVIOR

As far as concern the unbiased branches problem, after bibliographical research [13, 14, 16, and 17], it outlines the following practical ideas for characterizing them from the random degree viewpoint:

The prediction accuracy of a symbols sequence provided by a HMM predictor could define the random degree of that sequence. Particularly, it is interesting to see whether this idealized powerful predictor would successfully predict the unbiased branches. An affirmative answer would mean that the relevant prediction information exists but is hard to identify it, differing from one branch to another. Otherwise, if the answer is negative, the intrinsic random degree (determinist chaos) of these branches would be very significant.

As we already stated, we believe that an acceptable engineering metric for defining the random degree RD(S) of a branch's output sequence S, could be the product between discrete entropy E(S) and the shuffle degree D(S). Thus *RD(S) = D(S)·E(S)*.

The compression rate of a symbols sequence (or the space savings due to its compression), provided by well-known lossless compression algorithms such *Huffman* and *Gzip*, could represent another effective metric for characterizing the random degree of that sequence. In our opinion, the compression rate and obviously, the space savings of sequences generated by unbiased branches behavior should be lower than that obtained for sequences generated by biased branches.

The Kolmogorov complexity of code sequence that effectively generates unbiased branches could be a useful metric for describing the random degree. Thus, the unbiased branches complexity should be higher than the other conditional branches complexity. Nevertheless, Kolmogorov complexity has a static nature while it tries to characterize the dynamic behavior of a certain branch. On the other hand, this metric is the single one that emphasizes the semantic complexity of the generator code sequence.

In the next paragraphs, we will detail and make a qualitative and quantitative analysis of all these metrics. Like in paragraph 3 we used the same six difficult predictable SPEC 2000 benchmarks simulating one billion dynamic instructions for each one. It was considered a 16-bit global history (GH) context for each branch. We selected from each benchmark strongly unbiased contexts having low polarization indexes ($P(S) \in [0.501, 0.565]$) and respectively strongly biased contexts with high polarization indexes ($P(S) \in [0.979, 0.997]$), very frequently processed (hundreds of thousands instances per a certain context). The polarization index was defined in [1]. Each context has associated a binary string representing its behavior (Taken/Not Taken). This binary string represents the input sequence for the HMM predictor used by us in paragraph 5.1. During the paragraph 5.2 we calculated the random degrees associated to the same binary strings. In paragraph 5.3 we calculated the compression rates corresponding to the same branches' behaviors.

5.1. RANDOM DEGREE BASED ON HMMS

During this paragraph we considered a per branch local history of 64 bits. Using a longer history significantly complicated our developed HMM predictors and grew up the computing time. Anyway, our proposed metric is quantitatively very relevant. Good presentations of HMM of order one could be found in [14]. A HMM algorithm of superior order ($R \geq 1$) was developed by us in [15]. We evaluated prediction accuracies on strongly unbiased branches using a HMM predictor of order one ($R=1$) and two ($R=2$) for different numbers of possible hidden states (N). For the majority of the benchmarks considering two hidden states generate the best accuracies. The average prediction accuracy obtained using the quasi-optimal HMM ($R=1$, $N=2$) is far greater on biased contexts than on unbiased contexts. Figure 3 comparatively presents, for unbiased and biased branches, the average prediction accuracies obtained by the optimal HMM ($R=1$, $N=2$).

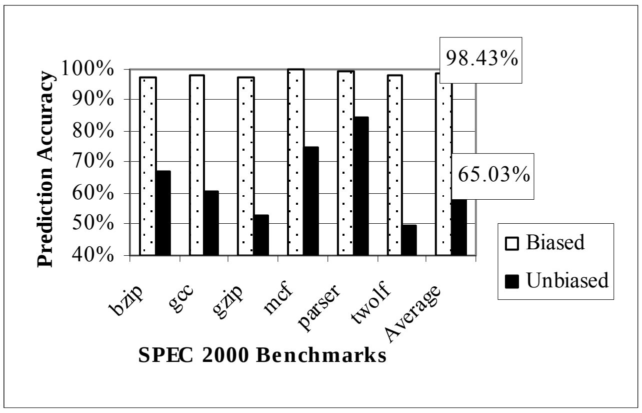

Figure 3. Prediction accuracies using the best evaluated HMM (R=1, N=2)

There is a significant difference between the average prediction accuracy on biased branches (98.43%) and respectively on unbiased branches (65.03%). As far as we know, we are the first researchers investigating HMMs as an ultimate branch prediction limit. Unfortunately, even these powerful predictors cannot accurately predict unbiased branches. This fact suggests that unbiased branches are "intrinsic random" in some way, being generated by very complex program structures as we will further show.

5.2. RANDOM DEGREE BASED ON DISCRETE ENTROPY

In this paragraph we considered as the random degree of a binary sequence RD(S), the product between discrete entropy E(S) and shuffle degree D(S) associated to S. Thus $RD(S) = D(S) \cdot E(S)$.

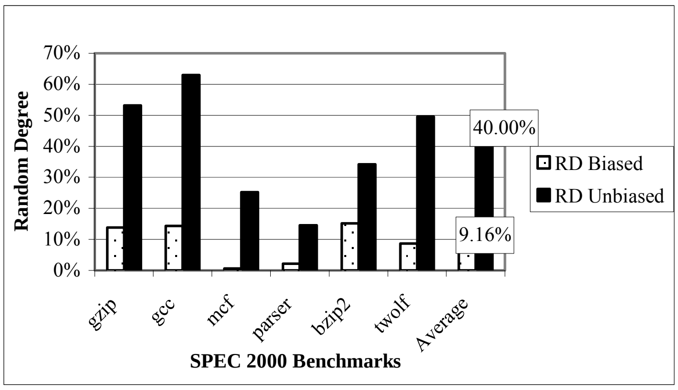

Figure 4. The random degree of biased respectively unbiased branches

Figure 4 shows statistical results concerning the random degree of binary sequences obtained through the previously exposed methodology. Since our initial supposition was that *biased* branches sequences should have a lower random degree, the simulation results confirm that the RD(S) metric represents a good measure for random degree of binary sequences. A random degree around 40% shows that respective unbiased branch is difficult or, practically, even impossible to be accurately predicted.

5.3. RANDOM DEGREE BASED ON COMPRESSION RATE

Further we transformed in extended ASCII files the binary sequences generated by unbiased respectively biased branches behavior obtained through the methodology exposed in paragraph 5. We grouped 8-bit sequences and generate the corresponding ASCII codes. We compressed these files using *Gzip* utility [18] and respectively an own developed application that implements *Huffman* encoding [17].

Huffman proposes an entropic encoding greedy algorithm, effective and very useful in lossless compression, commonly used as final stage of compression. The basic idea is to map an alphabet to a representation for that alphabet, composed of variable length strings, so that symbols with a higher occurrence probability have a smaller representation than those that occur less often.

The kernel of Gzip utility is *DEFLATE* algorithm [19], that represents a combination between *LZ77* algorithm [20] (dictionary encoding technique) and Huffman algorithm (statistical encoding technique). The compression is performed in two successive stages: i) the identification and replacement of duplicate strings with pointers (LZ77) and ii) replacing the previously obtained symbols with new, weighted symbols based on frequency of use (Huffman).

We based our statistics on a commonly used metric in data compression:

$$Space\,Savings = \left(1 - \frac{Compressed\ Size}{Uncompressed\ Size}\right) \cdot 100\% \tag{4}$$

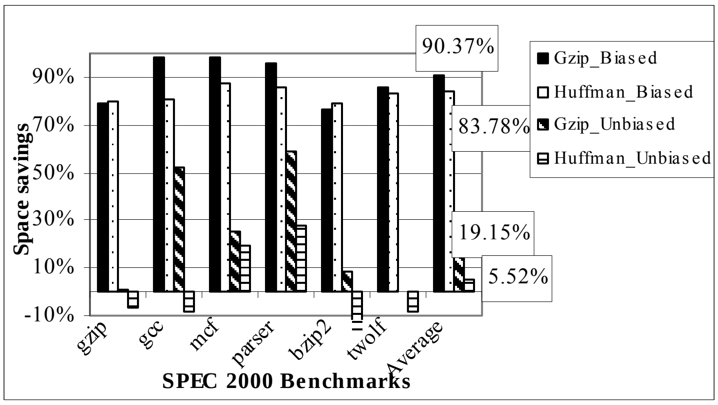

Figure 5. Space savings using *Gzip* and *Huffman* algorithms

In Figure 5, we illustrate the space savings obtained by compression of biased and unbiased branches with previously described algorithms (*Gzip* and *Huffman*). The space savings obtained through unbiased branches compression (19.15% with *Gzip* utility) are significantly lower than those obtained through biased branches compression (90.37% with *Gzip*). The ascendancy of *Gzip* algorithm toward *Huffman* algorithm is understandable taking into account that Huffman encoding represents the final stage of *Gzip* compression. However, it can be observed that the space saving on *twolf* benchmark becomes negative (-0.29%) even if the *Gzip* compression algorithm is used. The *LZ77* algorithm's influence is almost inexistent leading to the conclusion that it is impossible to find repetitive patterns. Actually, we obtained similar results in [2], where we have shown that using some Markovian predictors, the unbiased branches prediction accuracy is very low.

5.4. RANDOM DEGREE BASED ON KOLMOGOROV COMPLEXITY

First, we focused on the most important unbiased branch (having PC=58) from *Perm* benchmark [21] that exhibits an unpredictable behavior even if its context length is very long (53 bits of global history). Actually, the percentage of unbiased branches (1.53%) from whole *Perm* program is exclusively due to the branch from PC=58.

We developed a particular fast path-based perceptron (FPBP) predictor [4] with a global history length of 53 bits and 100 entries. FPBP predicted the branch 58, in its unbiased contexts, with 65.91% accuracy. The number of FPBP mispredictions was 286. The complete PPM predictor exploits the recursive character of the Perm benchmark. The prediction accuracy (PA) obtained by our developed PPM using a global context length of 500 bits and a search pattern of 30 bits on the branch 58, was 94.30%. As far as this solution is hardware unfeasible we tried a simplified PPM, but the result was dissatisfactory (PA=79.85%). The global prediction accuracy provided by complete PPM is 98.41%, lower than that generated by the FPBP predictor (99.04%). Actually, from 869 PPM mispredictions, the branch 58 generates 287. Thus, we can conclude that both PPM and FPBP predictors do not succeed to accurately predict an unbiased branch. The high prediction accuracy (94.30%) on the branch 58 provided by PPM is actually centered on the whole behavior of the branch and not only on its unbiased context.

According to Kolmogorov, the length of the shortest program for a universal Turing machine that correctly reproduces the observed data is a measure of complexity [16]. Thus, analyzing the behavior of the branch 58 from Kolmogorov's complexity perspective (we noted it $K(58)$), it can be observed that the minimal length of machine-code that generates this unbiased branch is equal with the *Permute* routine length (measured in instructions). This happens because in order to reach the branch 58 the Permute routine should completely execute at least once (due to recursive call). Thus, $K(58) = 42$ HSA instructions or $K(58) = 8$ C instructions.

Among the other conditional branches only one (PC=35) proved to be unbiased for shorter global history length (≤32 bits). However, increasing the global history length to 53 bits the branch 35 became fully biased, and, therefore predictable. Analyzing Kolmogorov's complexity of the branch 35 we calculated $K(35) = 12$ HSA instructions or $K(35) = 3$ C instructions. It involves that $K(35) \leq K(58)$. This happens because the test of the branch 35 does not require the complete execution of *Permute* routine. Therefore, the code sequence's complexity that generates the unbiased branch (58) induces a determinist chaos, frequently occurred in many science domains. In addition, based on analysis of many integer recursive benchmarks we have reasons to believe that recurrence combined with some certain conditional branches will generate branches with unbiased behavior and thus with high Kolmogorov complexity.

6. CONCLUSIONS AND FURTHER WORK

At this moment, there is not a universal accepted paradigm for effectively defining random strings of symbols. Not surprisingly, understanding randomness is closely related with fertile mathematical concepts like computability and algorithms, information theory and complexity, actual infinites theory, etc. The problem is therefore open and of great interest in many fields of science. Particularly we showed that unbiased branches' behavior could be understandable in more depth using this methodological frame. Using both theoretical approaches and simulation methodologies, we mainly developed some degrees of random associated to a certain branch in order to understand better the unbiased branches behavior.

We used some powerful state of the art branch predictors to predict unbiased branches. Particularly, we developed an improved state of the art piecewise linear branch predictor to successfully predict unbiased branches. However, even this idealized powerful predictor obtained modest average prediction accuracy on the unbiased branches (78.3%) while its global average prediction accuracy is of 95.45%. Other very powerful general predictors, like our developed HMMs, predict unbiased branches with an even lower average accuracy. Therefore, predicting unbiased branches still represents a hard challenge. Finding new relevant prediction information to reduce significantly the number of unbiased shuffled branches remains an open problem. Computer Architects cannot therefore continue to expect a prediction accuracy improvement with present-day prediction techniques and alternative approaches are necessary.

In order to understand better the unbiased branches' behavior we developed four metrics that are defining the random degree of a string of symbols. Our experiments proved that all these four developed metrics are converging at the same point. The unbiased branches are "almost random" due to some complex corpus of programs. They are characterized by a deterministic chaotic behavior. For example, the "RD" metric is 1 for a "completely random" branch, but it is about 0.40 for unbiased branches and 0.09 for biased branches (Figure 4). The "Space savings" must be 0 for a "completely random" branch; in our experiments it was about 0.83 for biased branches and 0.05 for unbiased branches using the Huffman compression algorithm (Figure 5). The HMM predictor also obtains an excellent average prediction accuracy on biased branches (98.43%) showing its significant prediction power while the average prediction accuracy on unbiased branches is limited to 65.03% (Figure 3). Moreover, the Kolmogorov complexity of an unbiased branch is higher than the Kolmogorov complexity of any conditional branch belonging to the same program. All these random degree metrics could practically help the computer architect to better understand branches' predictability and if the branch predictor should be improved related to the unbiased branches. They are showing how much intrinsic randomness contains a string of symbols and, particularly, our discovered unbiased branches. If some difficult branches are not intrinsic random according to our metrics, their prediction accuracy could be further improved by the researcher. Otherwise, if these branches are proving to be intrinsic random, the answer is a pessimistic one, generating a powerful limitation in Computer Architecture.

During the development of this work, we have gained additional knowledge about branch behavior and predictability. A next step of our research is to apply this knowledge to design a novel prediction algorithm, dedicated to unbiased branches, that provides better results. Alternative solutions in order to reduce efficiently unbiased branches effects could be predication or dynamic execution of both branch paths. Also, as a further work it would be useful to quantify the unbiased branch ceiling in a multithreaded or even in a multicore architecture.

ACKNOWLEDGMENTS

We like to thank to Professor Solomon Marcus, member of the Romanian Academy, for providing some useful references and for competently explaining some concepts related to randomness' approaches. In addition, we express our gratitude to Professor Mateo Valero from UPC Barcelona, to Dr. Oliverio Santana from ULPGC, Spain, to Professor G. Stefan from UP Bucharest and to Professor Theo Ungerer from Augsburg University, Germany, for competently reviewing this work and for their very useful suggestions.

REFERENCES

1. Vințan L, Gellert A, Florea A, Oancea M, Egan C, *Understanding Prediction Limits through Unbiased Branches*, Lecture Notes in Computer Science, Springer-Verlag Berlin / Heidelberg, vol. **4186**, pp. 480-487, 2006.
2. Gellert A, Florea A, Vințan M, Egan C, Vințan L., *Unbiased Branches: An Open Problem*, Lecture Notes in Computer Science, Advances in Computer Systems Architecture, Springer-Verlag Berlin / Heidelberg, vol. **4697**, pp. 16-27, 2007.
3. Desmet V., *On the Systematic Design of Cost-Effective Branch Prediction*, PhD Thesis, Ghent University, Belgium, June 2006.
4. Jiménez D., *Fast Path-Based Neural Branch Prediction*, Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO-36), San Diego, CA, pp. 243-252, December 3-5, 2003.
5. Jiménez D., *Idealized Piecewise Linear Branch Prediction*, Championship Branch Prediction (CBP-1), 2004.
6. Jiménez D., *Idealized Piecewise Linear Branch Prediction*, Journal of Instruction-Level Parallelism, vol.**7**, pp.1-11, April, 2005.
7. Seznec A., *The Idealistic GTL Predictor*, Journal of Instruction Level Parallelism, Vol. **9**, pp.1-11, May, 2007.
8. Seznec A., *The L-TAGE Branch Predictor*, Journal of Instruction Level Parallelism, Vol. **9**, pp.1-13, May, 2007.
9. The 2nd Journal of Instruction-Level Parallelism Championship Branch Prediction Competition (CBP-2), Florida, USA, 2006.
10. Gao H, Ma Y, Dimitrov M, Zhou H., *Address-Branch Correlation: A Novel Locality for Long-Latency Hard-to-Predict Branches*, 14th International Symposium on High Performance Computer Architecture, Salt Lake City, Utah, pp. 74-85, February, 2008.
11. SPEC2000, *The SPEC benchmark programs*, http://www.spec.org.
12. The 1st Journal of Instruction-Level Parallelism Championship Branch Prediction Competition (CBP-1), 2004.
13. Volchan S B., *What Is a Random Sequence?,* The American Mathematical Monthly, The Mathematical Association of America, vol. **109**, pp.46-63, January 2002.
14. Rabiner L R., *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, Vol. **77**, No. 2, pp.257-286, February 1989.
15. Gellert A., Vințan L.. *Person Movement Prediction Using Hidden Markov Models,* Studies in Informatics and Control, National Institute for Research and Development in Informatics, Bucharest, Vol.**15**, No. 1, pp.17-30, March 2006.
16. Gammerman A., Vovk V., *Kolmogorov Complexity: Sources, Theories and Applications*, The Computer Journal, Vol.**42**, No. 4, pp. 252-255, 1999.
17. Cormen T H, Leiserson C E, Rivest R L, and Stein C., *Introduction to Algorithms, Second Edition*, MIT Press and McGraw-Hill, Section 16.3, pp.385–392, 2001.
18. http://www.gzip.org/
19. Deutsch P., *DEFLATE Compressed Data Format Specification version 1.3,* Aladdin Enterprises, Network Working Group, RFC 1951, pp.1-15, 1996.
20. Ziv J., Lempel A., *A Universal Algorithm for Sequential Data Compression*, IEEE Transactions on Information Theory, Vol. **IT-23**, No. 3, pp. 337-343, 1977.
21. Steven G, Christianson B, Collins R, Potter R, Steven F., *A Superscalar Architecture to Exploit Instruction Level Parallelism*, Microprocessors and Microsystems, Elsevier, Vol. **20**, Nr. 7, pp. 391-400 (10), March 1997.

Received July 26, 2008