Appeared in the *Proceedings of the Third Conference of The Academy of Technical Sciences from Romania*, ISBN: 978-973-713-223-9, Cluj-Napoca, November 2008

FORCING SOME ARCHITECTURAL CEILINGS OF THE ACTUAL PROCESSOR PARADIGM

Lucian N. Vinţan[1,2], Adrian Florea[2], Arpad Gellert[2]

[1] Academy of Technical Sciences from Romania
[2] Computer Science Department, "Lucian Blaga" University of Sibiu, Emil Cioran Street, No. 4, 550025 Sibiu, Romania
{lucian.vintan, adrian.florea, arpad.gellert}@ulbsibiu.ro

Abstract: In our previously published research we discovered some very difficult to predict branches, called unbiased branches that have a "random" dynamic behavior. We developed some state of the art branch predictors to successfully predict them. Even these powerful predictors obtained very modest average prediction accuracies on the unbiased branches whereas their global average prediction accuracies are high. The unbiased branches still restrict the ceiling of dynamic branch prediction and therefore accurately predicting them remains an open problem. Since the overall performance of modern superscalar processors is seriously affected by misprediction recovery, especially these difficult branches represent a source of important performance penalties. Our statistics show that about 28.68% of branches are dependent on critical Load instructions. Moreover, 5.61% of branches are unbiased and depend on critical Loads, too. These dependences involve high-penalty mispredictions becoming serious performance obstacles and causing significant performance degradation. The negative impact of (unbiased) branches over global performance should be seriously attenuated by anticipating the results of long-latency instructions, including critical Loads. On the other hand, hiding instructions' long latencies in a pipelined superscalar processor represents an important challenge itself.

In this work we focused on three main objectives: first, we proposed several metrics for characterizing the unbiased branches from the random degree viewpoint, to effectively help the computer architect to better understand branches' predictability and also if the predictor should be improved related to unbiased branches. Starting from the dynamical behavior of unbiased branches, we tried to understand in more depth what randomness is. Based on a hybrid mathematical and computer science approach we mainly developed some random degrees associated to a branch. These metrics are: program's Kolmogorov complexity, compression rate, discrete entropy and HMM-based prediction accuracy, that are useful for characterizing strings of symbols and particularly, our unbiased branches' behavior. If some difficult branches are not "intrinsic" random according to our metrics, their prediction accuracy might be further improved. Otherwise, the answer is a pessimistic one, generating a powerful limitation in Computer Architecture.

As the second aim of this paper, we developed a superscalar architecture that selectively anticipates the values produced by high-latency instructions. We are focusing on Multiply, Division and Loads with miss in L1 Data Cache, implementing a Dynamic Instruction Reuse (DIR) scheme for the Mul/Div instructions and respectively a Last Value Predictor (LVP) for the critical Load instructions.

As a final objective of our research, we quantify the impact of our developed Selective Instruction Reuse and Value Prediction techniques in a simultaneous multithreaded architecture (SMT) that implies per thread Reuse Buffers (RB) and LVP tables. We measure the IPC and the dynamic power consumption of the proposed SMT architecture by varying the number of threads. Also, we evaluate, for different number of

threads, the IPC speedup and the EDP gain of a SMT architecture enhanced with Selective Instruction Reuse and Value Prediction against a classical SMT architecture.

Keywords: ILP processors, branch prediction, unbiased branches, discrete entropy, random degree, dynamic instruction reuse, load value prediction, speculative execution, SMT architecture, power consumption

1. Introduction

The branch prediction becomes a challenge problem for processors' designers. Without performing branch prediction it won't be possible to aggressively exploit program's instruction level parallelism (ILP). All present branch prediction techniques are limited in their accuracy. An important limitation cause is given by the used prediction contexts (global and local histories respectively path information). In our previous work, we show that, irrespective of the prediction information length and type, used in the state of the art branch predictors, some branches are unbiased and non-deterministically shuffled, and are characterized by low prediction accuracies (at average about 70%) [Vin06, Gel07]. 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 rigorously know what randomness is. Without effectively understanding their "random" behavior we cannot expect to develop accurate predictors. We started from the following fundamental question: *could a deterministic program generate some branches having a "random behavior"*? Unfortunately, the answer is not simple. Based on a combined mathematical and computer science approach, we proposed and developed, as the first aim of this paper, some random degree metrics, like program's Kolmogorov complexity, compression rate, discrete entropy and HMM (Hidden Markov Model [Rab89]) 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 branch random degrees could effectively help in quantifying program's predictability, too.

Since the overall performance of modern superscalar processors is seriously affected by misprediction recovery, these difficult branches represent a source of important performance penalties. As we pointed out in [Gel06], 28.68% of branches are dependent on critical Load instructions (Loads with miss in the L2 data cache that reach the head of the Reorder Buffer), and 5.61% are unbiased and dependent on a previously committed critical Load instruction. Such unbiased (or at least hard-to-predict) branches occur in pointer chasing applications based on linked list traversal:

(e.g., *while (node)* *// Branch B*
node = node→next *// Load L*).

In hereinbefore example, since Branch B depends on Load L, a branch misprediction cannot be solved until Load L returns the value. If Load L has a high L2 cache miss rate, the branch misprediction penalties of Branch B will have significant impact on the overall performance. For example, the average misprediction penalty of such a branch, measured as the latency between fetching the branch instruction and resolving the misprediction, is about 540 cycles, considering a L2 cache miss penalty of 300 cycles [Gao08]. Thus, the forementioned dependences involve high-penalty mispredictions becoming serious performance obstacles and causing significant performance degradation in executing

instructions from wrong paths. Therefore, the negative impact of mispredicting branches, particularly of mispredicting unbiased branches over the global performance should be seriously attenuated by anticipating the results of long-latency instructions. On the other hand, hiding instructions' long latencies in a pipelined superscalar processor represents an important challenge itself.

As the second aim of this work we developed a superscalar architecture that selectively anticipates the values produced by high-latency instructions. We will focus on Multiply, Division and Loads with miss in the L1 data cache. These instructions would be solved by a Dynamic Instruction Reuse scheme. However, an additional Reuse Buffer for Load Value (Data) Reuse is not necessary, because a similar reuse mechanism is already provided by the existing L1 and L2 data caches. Therefore, the Load instructions with miss in the L1 data cache (selective approach) would be solved through value prediction.

As a final objective of our research, we quantify the impact of our developed Selective Instruction Reuse and Value Prediction techniques in a simultaneous multithreaded architecture that implies per thread Reuse Buffers and LVP tables. We measure the IPC and the dynamic power consumption of the proposed SMT architecture by varying the number of threads. Also, we evaluate, for different number of threads, the IPC speedup and the EDP gain of a SMT architecture enhanced with Selective Instruction Reuse and Value Prediction against a classical SMT architecture.

The organization of the rest of this paper is as follows: Section 2 contains our last developments in understanding and predicting unbiased branches. During Section 3 we make a qualitative and quantitative analysis of four distinct metrics to characterize the random degree for a certain dynamic branch. Section 4 describes the two techniques that we implemented for anticipating the results of long-latency instructions. In Section 5 we quantify the impact of DIR and VP techniques in SMT Architectures. The last Section debates and concludes on the most important obtained results and proposes some further work.

2. Understanding and Predicting Unbiased Branches

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. Thus, we specially developed two idealized powerful branch predictors: an improved idealized piecewise linear branch predictor [Jim05] and a HMM-based branch predictor. Unbiased branches need some specific efficient predictors that are using some new, more relevant prediction information. Finding a new relevant context to significantly reduce the number of unbiased shuffled branches remains an open problem.

In our experiments we concentrated only on SPEC 2000 benchmarks [SPEC] with a fraction of unbiased branches greater than 1%. Following this methodology, 6 integer benchmarks fulfilled this condition. As a consequence, in Sections 2 and 3 we have simulated only these difficult predictable benchmarks (*gzip, bzip, mcf, parser, twolf, gcc*). Figure 1 presents the prediction accuracies obtained with the idealized piecewise linear branch predictor (PW) on all branches respectively on the unbiased branches, using the previous global dynamic branch's condition value (PBV) as an additional prediction information [Vin08]. The first two bars represent the prediction accuracies on all branches respectively on unbiased branches, obtained with the idealized piecewise linear branch

predictor. The rest of the bars were obtained using PBV (32 bits) as additional prediction information, varying the number of weights (from 8590 up to 30713).

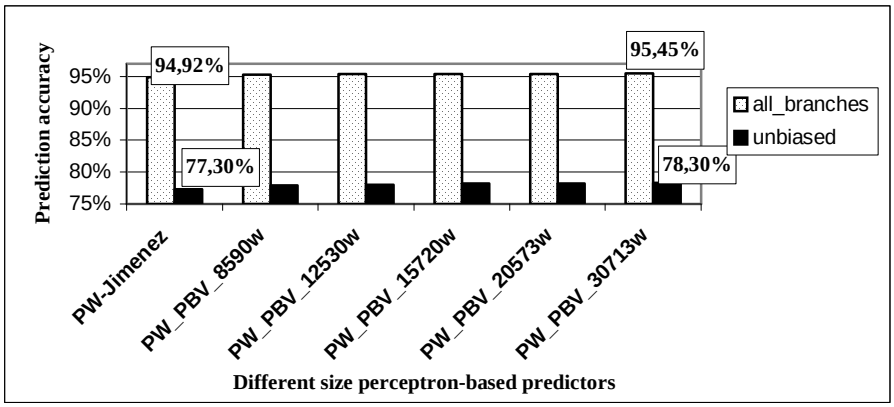

Figure 1. The prediction accuracies obtained with *piecewise linear branch predictor* on unbiased branches versus all branches, using the global PBV as additional prediction information

Analyzing Figure 1 it can be observed how 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 contributes to the global prediction accuracy (value that overcome with more than 0.53% the best state of the art predictor's performance).

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 make a qualitative and quantitative analysis of four distinct metrics to characterize the random degree for a certain dynamic branch. These metrics could help us to better understand the unbiased branches behavior and their potential predictability.

3. Random degree metrics for characterizing unbiased branches' behavior

Like in Section 2, we used the same 6 difficult predictable SPEC 2000 benchmarks simulating one billion dynamic instructions for each one, skipping the first 300 million instructions. It was considered a 16-bit global history context for each branch. We selected from each benchmark very frequently processed (hundreds of thousands instances per a certain context) strongly unbiased branch contexts having low polarization indexes ($P(S) \in [0.501, 0.565]$) and respectively strongly biased contexts with high polarization indexes ($P(S) \in [0.979, 0.997]$). The polarization index was defined in [Vin06]. Each context has associated a binary string representing its behavior. This binary string represents the input sequence for the HMM predictor developed by us in paragraph 3.1. During paragraph 3.2 we calculated the random degrees associated to the same binary strings based on their entropy. In paragraph 3.3 we calculated the compression rates corresponding to the same branches' behaviors, as with random degrees.

3.1. *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. We modified the number of hidden states of the HMM in order to

maximize the prediction accuracy. Anyway, our proposed metric is quantitatively very relevant.

Figure 2 comparatively presents, for unbiased and biased branches, the average prediction accuracies obtained by our optimal 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 we expected, the HMM predictor obtains an excellent average prediction accuracy on biased branches showing its significant prediction power. 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.

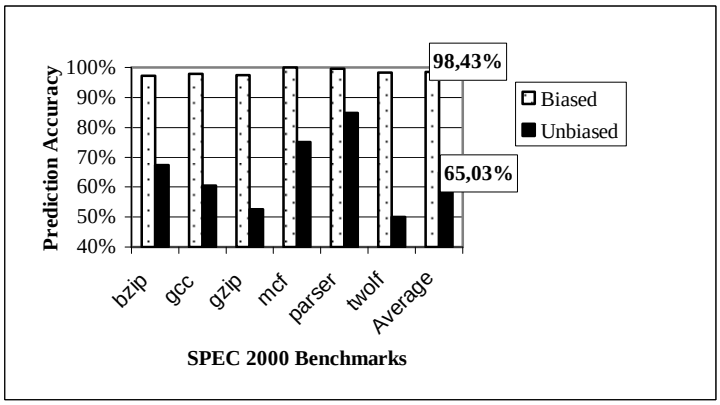

Figure 2. Prediction accuracies using the best evaluated HMM (R=1, N=2)

3.2. *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* [Vin08]. Thus, $RD(S) = D(S) \cdot E(S)$. Figure 3 shows statistical results concerning the random degree of (biased and respectively unbiased) binary sequences obtained through the previously exposed methodology.

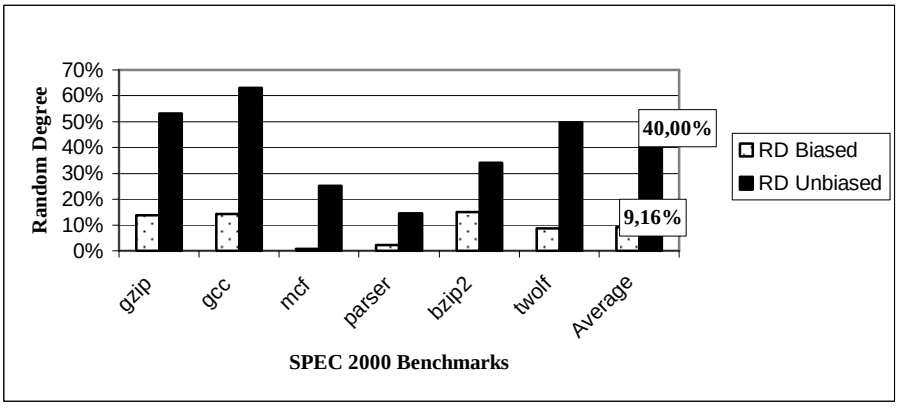

Figure 3. The random degree of biased respectively unbiased branches

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.

3.3. *Random degree based on compression rate and program's Kolmogorov complexity*

The compression rate of a symbols sequence (or the space savings due to its compression), provided by the well-known lossless compression algorithms such *Huffman* and *Gzip*, could represent another effective metric for characterizing the random degree of that sequence.

Further we transformed in extended ASCII files the binary sequences generated by unbiased respectively biased branches behavior obtained through the methodology exposed in Section 3. We grouped 8-bit sequences and generate the corresponding ASCII codes. We compressed these files using the *Gzip* utility [Gzip] and respectively an own developed application that implements *Huffman* encoding based on [Cor01]. We based our statistics on two commonly used metrics in data compression:

$$Compression\,Rate = \frac{Uncompressed\ Size}{Compressed\ Size} \cdot 100\% \quad (1)$$

$$Space\,Savings = \left(1 - \frac{Compressed\ Size}{Uncompressed\ Size}\right) \cdot 100\% \quad (2)$$

In Figure 4, we illustrate the space savings obtained by compression of biased respectively unbiased branches with the previously described algorithms (*Gzip* and *Huffman*). The main conclusion refers to the space savings obtained through unbiased branches compression (19.15% with the *Gzip* utility) that are significantly lower than those obtained through biased branches compression (90.37% with *Gzip*).

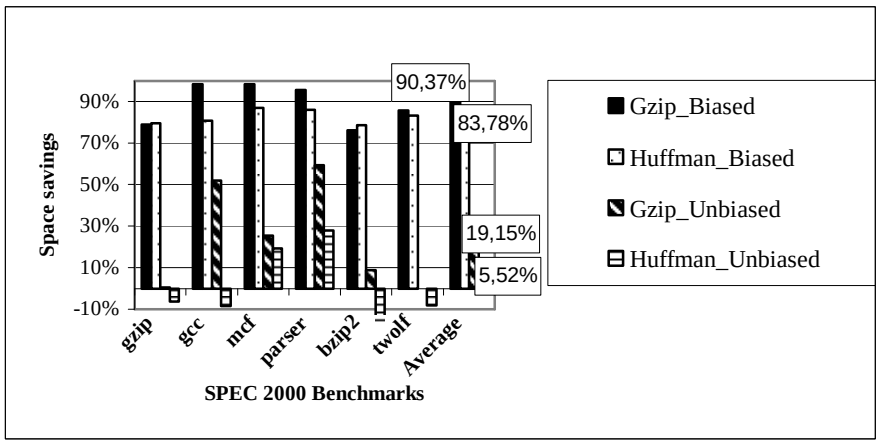

Figure 4. Space savings using the *Gzip* and *Huffman* algorithms

The Kolmogorov complexity of code sequence that effectively generates unbiased branches could be a useful metric for describing the random degree, too. 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 corresponding generator code sequence. 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. Such examples occur in the link lists or trees cases where the address of an element is tested and followed by a recurrent call of the same function to test the next element in the tree.

4. Anticipating Long-Latency Instructions Results

Further we developed a superscalar architecture that selectively anticipates the values produced by high-latency instructions. This is particularly useful in order to reduce the negative impact of unbiased branches in superscalar processors, too. The reusability degree of MUL and DIV instructions, measured with an unlimited Reuse Buffer, was 28.9% on the integer (INT) benchmarks and 61.9% on the floating-point (FP) benchmarks. These instructions would be solved by a DIR scheme. We also detected and reused the results of trivial operations like V*0, V*1, 0/V, V/1 and V/V. The reusability degree of Load values was 77.4% on the integer benchmarks and 76.4% on the floating-point benchmarks. However, an additional Reuse Buffer for Load Value (Data) Reuse is not necessary, because a similar reuse mechanism is already provided by the existing L1 and L2 data caches. Therefore, the Load instructions with miss in the L1 data cache (selective approach) would be solved through a simple Last Value Predictor. The hardware structures used for selective instruction reuse and load value prediction and their functionality mechanism are detailed in [Gel08].

Figure 5 presents the relative IPC speedup and the relative energy-delay product (EDP) improvement for the integer respectively floating-point SPEC 2000 benchmarks. The EDP represents the processor's total power, divided by the squared IPC. We determined the energy-delay product for the architecture without RB and LVPT respectively for the architecture with a RB of 1024 entries and LVPTs of different sizes. The *EDP Gain* represents the relative energy-delay product improvement for each LVPT size. As it can be observed, the optimal LVPT size is 1024.

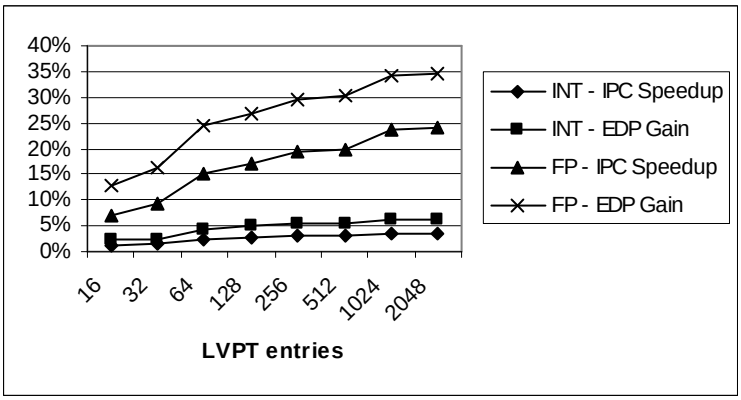

Figure 5. Relative IPC speedup and relative energy-delay product gain with a Reuse Buffer of 1024 entries, the Trivial Operation Detector, and the Load Value Predictor

Predicting critical Load instructions through an additional Last Value Predictor, improves the IPC with 3.5% on the integer benchmarks respectively with 23.6% on the floating-point benchmarks. This significant speedup lowers the energy consumption with 6.2% on the integer benchmarks respectively with 34.5% on the floating-point benchmarks. This difference occurs because the number of critical Loads is more than twice higher in the floating-point benchmarks. Consequently, selectively applying some well-known techniques on long-latency instructions provides serious performance gain and significantly reduces energy consumption within the superscalar architecture.

5. Selective Instruction Reuse and Value Prediction in SMT Architectures

As a final objective of this research, we quantified the impact of our developed techniques for anticipating long-latency instructions results in a simultaneous multithreaded

architecture that implies per thread RB and LVP tables. We developed a Reuse Buffer and a Trivial Operation Detector for MUL and DIV instructions respectively a Last Value Predictor for critical Load instructions, and we integrated all these structures into the M-Sim simulator [Sha05]. M-Sim supports single threaded execution (superscalar mode) as well as the multithreaded mode in which multiple threads of control are executed simultaneously, according to the Simultaneous Multithreaded (SMT) model [Egg97]. In the SMT mode, some processor structures (i.e. issue queue, physical register files, functional units, caches) are shared among the threads, and others (rename tables, ROBs, Load/Store Queues, branch predictors) are private to each thread (see Figure 6).

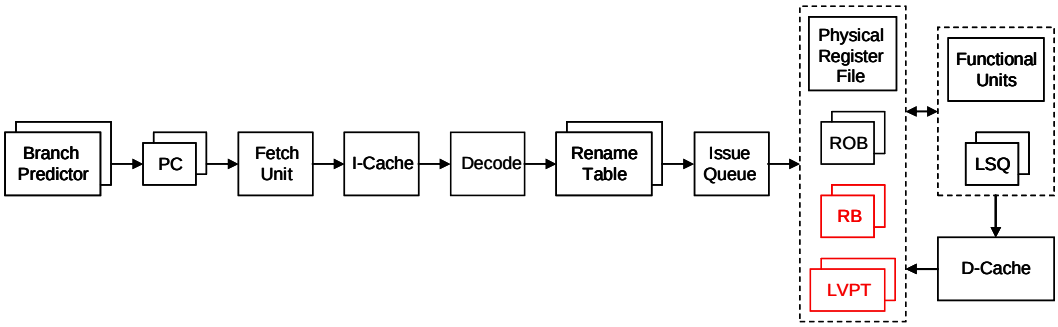

Figure 6. SMT architecture enhanced with selective instruction reuse and value prediction

Threads maintain separate program counters (PC), but share the fetch unit and I-Cache. Threads share the available bandwidth in the front end, including fetch, decode and renaming. The M-Sim implements the well known ICOUNT fetch policy, by default, fetching from up to two threads per cycle. The M-Sim has implemented separate branch predictors per thread, which provide the best performance for multithreaded processors. The Reorder Buffers (ROB) as well as our Reuse Buffers (RB) and Load Value Prediction Tables (LVPT) are private. Each thread maintains its own rename table because it has its own set of architectural registers. After renaming, instructions from all threads are dispatched into the shared Issue Queue. In the Issue Queue, instructions from all threads participate in instruction wakeup and compete for the issue bandwidth in selection. Instructions that are selected for issue continue to register file access. There are separate integer and floating-point physical register files, both being shared among threads. After register file access is complete, instructions begin execution on the functional units, which are also shared. Loads and Stores access the shared data cache. In order to maintain the correct ordering of memory accesses, the Load/Store Queue (LSQ) is used. The M-Sim uses separate LSQs per thread, so that an unresolved address from one thread does not prevent Loads in other threads from issuing. After execution, instructions write back to the register files. Commitment is done in order for each thread.

5.1. Simulation Methodology

For the superscalar architecture we evaluated from SPEC 2000 suite seven integer benchmarks (*bzip*, *gcc*, *gzip*, *mcf*, *parser*, *twolf*, *vpr*) and six floating-point benchmarks (*applu*, *equake*, *galgel*, *lucas*, *mesa*, *mgrid*). In SMT mode, the M-Sim runs multiple benchmarks as different threads in parallel. Therefore, we combined benchmarks into groups of 2, 3 or 6 depending on the simulated SMT architecture. Thus, we used {*bzip*, *gcc*}, {*gzip*, *parser*}, {*twolf*, *vpr*}, {*applu*, *equake*}, {*galgel*, *lucas*}, {*mesa*, *mgrid*} for our SMT with 2 threads, {*bzip*, *gcc*, *gzip*}, {*parser*, *twolf*, *vpr*}, {*applu*, *equake*, *galgel*}, {*lucas*, *mesa*, *mgrid*} for the SMT with 3 threads, and {*bzip*, *gcc*, *gzip*, *parser*, *twolf*, *vpr*},

{*applu, equake, galgel, lucas, mesa, mgrid*} for the 6-threaded SMT. Table 1 presents some important parameters of the simulated architecture:

Execution Latencies	**Execution unit**	**Number of units**	**Operation latency**
	intALU	4	1
	intMULT / intDIV	1	3 / 20
	fpALU	4	2
	fpMULT / fpDIV	1	4 / 12
Superscalarity	*Fetch* / *Decode* / *Issue* / *Commit* width = 4		
Branch predictor	bimodal predictor with 2048 entries		
Caches and Memory	**Memory unit**		**Access Latency**
	4-way associative L1 data cache, 32 KB		1 cycle
	8-way associative unified L2 data cache, 512 KB		6 cycles
	Memory		100 cycles
Resources	**Register File:** 32 INT / 32 FP		
	Reorder Buffer (ROB): 128 entries		
	Load/Store Queue (LSQ): 48 entries		

Table 1. Parameters of the simulated architecture

M-Sim facilitates the power estimation as supplied by the Wattch framework [Bro00]. The dynamic power consumption measurements are generated using an 80 nm CMOS technology:

$$P_d = C \cdot V_{dd}^2 \cdot a \cdot f \quad (3)$$

where C is the capacitance, generated using *Cacti* [Shi01], V_{dd} is the supply voltage, and f is the clock frequency. V_{dd} and f depend on the assumed process technology. The activity factor a indicates how often clock ticks lead to switching activity on average. For the energy measurements, we used the Energy-Delay Product, a widely used metric [Bro00]:

$$EDP = \frac{Total\ Power}{IPC^2} \quad (4)$$

5.2. Experimental Results

We measured the IPC and the dynamic power consumption of the proposed SMT architecture by varying the number of threads. Figures 7 and 8 present the IPC obtained by evaluating our developed superscalar and SMT architectures with respectively without Reuse Buffer and Load Value Predictor. According to our previous results, we optimally sized the RB respectively the LVPT to 1024 entries.

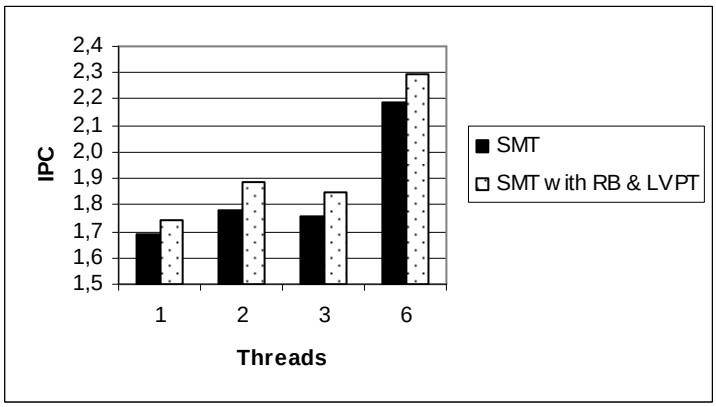

Figure 7. IPC obtained with respectively without RB & LVPT on the *integer* benchmarks

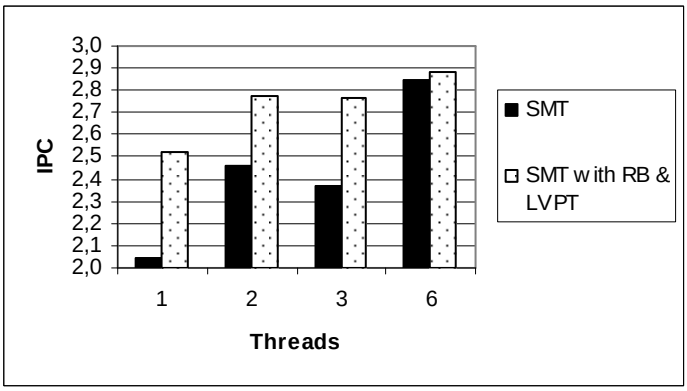

Figure 8. IPC obtained with respectively without RB & LVPT on the *floating-point* benchmarks

Figures 7 and 8 show that the RB and LVPT structures improve the IPC on all evaluated configurations. However, the highest improvement was obtained with one thread, and as the number of threads grows, the IPC improvement becomes lower (see Figure 8). With fewer threads, the ten shared functional units (see Table 1) are underused and therefore the selective instruction reuse and value prediction techniques have an important improvement potential. With a higher number of threads, the same ten functional units are highly used, thus both the instruction reuse and value prediction mechanisms becoming less important. Therefore, especially on floating-point benchmarks, with six threads we obtained the best IPC but the lowest relative IPC speedup (see Figures 7 and 8).

Finally, we evaluated, for different number of threads, the IPC speedup and the EDP gain of a SMT architecture enhanced with Selective Instruction Reuse and Value Prediction against a classical SMT architecture. In Figure 9 the first and third bars represent the EDP gains obtained with our superscalar (one thread) and SMT architecture (2, 3 and 6 threads) on the floating-point respectively integer benchmarks, whereas the second and fourth bars presents the IPC speedups achieved with the same architectures. As Figure 9 depicts, the RB and LVPT structures achieved IPC speedups and EDP gains on all the simulated configurations. The best improvements on the integer benchmarks have been obtained with 2 threads: an IPC speedup of 5.95% and an EDP gain of 10.44%. Although, on the floating-point benchmarks, we obtained the highest improvements with the enhanced (LVPT + RB) superscalar architecture, the SMT with 3 threads also provides an important IPC speedup of 16.51% and an EDP gain of 25.94%. Analyzing Figures 7 and 8 we can observe the advantage of SMT architectures against the superscalar architecture irrespective these are enhanced or not with selective instruction reuse and value prediction mechanism.

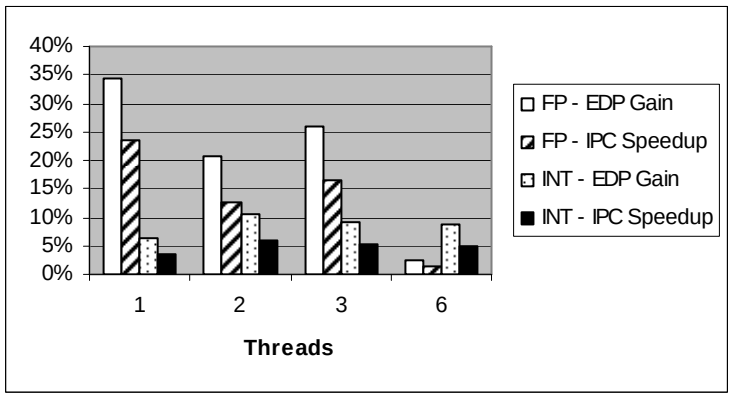

Figure 9. Relative IPC speedup and EDP gain (enhanced SMT vs. classical SMT) by varying the number of threads

6. Conclusions and Further Work

In this work we made the following contributions: first, we have gained additional knowledge about branch behavior and predictability. We mainly developed some random degrees associated to a certain branch based on HMM predictability, discrete entropy, compression rate and respectively program's Kolmogorov complexity. 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 a string of symbols and, particularly, our discovered unbiased branches contain. 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.

The second contribution is starting from the lack of efficiency of state of the art branch predictors to accurately predict unbiased branches. We developed an improved 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. Computer Architects cannot therefore continue to expect a prediction accuracy improvement with present-day prediction techniques and alternative approaches are necessary. Taking into account that an important fraction of branches are in the same time unbiased and dependent by critical Loads, we developed a superscalar architecture that selectively anticipates the values produced by high-latency instructions inclusively in order to reduce the negative impact of unbiased branches in ILP processors. The experimental results, performed on the SPEC 2000 benchmarks, show a significant IPC speedup and reduced energy consumption for the proposed architecture.

As the third contribution, we quantify the impact of our developed selective instruction reuse and value prediction techniques in a simultaneous multithreaded architecture. The new architecture enhanced with RB and LVPT structures achieved IPC speedups and EDP gains on all the simulated configurations. The best improvements on the integer benchmarks have been obtained with 2 threads: an IPC speedup of 5.95% and an EDP gain of 10.44%. Although, on the floating-point benchmarks, we obtained the highest improvements with the enhanced (LVP + Reuse) superscalar architecture, the SMT with 3 threads also provides an important IPC speedup of 16.51% and an EDP gain of 25.94%. With fewer threads, the shared functional units are underused and therefore the selective instruction reuse and value prediction techniques have an important improvement potential. With a higher number of threads, the same functional units are highly used, thus both the instruction reuse and value prediction mechanisms becoming less important. Therefore, especially on floating-point benchmarks, with six threads we obtained the best IPC but the lowest relative IPC speedup. Also, we can observe the advantage of SMT architectures against the superscalar architecture irrespective these are enhanced or not with selective instruction reuse and value prediction mechanism.

A next step of our research is to apply the knowledge gained about unbiased branches to design a novel dedicated prediction algorithm, that provides better results than previous proposals. Also, as a further work it would be useful to quantify the unbiased branch ceiling in a multicore architecture. Understanding, implementing and quantifying

instruction reuse and value prediction benefits in a multicore architecture might be another very important challenge for the future.

Acknowledgments

This work was partially supported by the Romanian National Council of Academic Research (CNCSIS) through the research grants TD-248/2007 and A-39/2007. 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.

References

[Bro00] Brooks D., Tiwari V., Martonosi M., *Wattch: A Framework for Architectural - Level Power Analysis and Optimizations*, International Symposia on Computer Architecture, Vancouver, 2000.
[Cor01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein - *Introduction to Algorithms*, Section 16.3, pp.385–392, Second Edition, MIT Press and McGraw-Hill, 2001.
[Egg97] Eggers S. Emer J., Levy H., Lo J., Stamm R., Tullsen D., *Simultaneous Multithreading: A Platform for Next-Generation Processors*, IEEE Micro, Vol 17, Issue 5, September 1997.
[Gao08] Gao H., Ma Y., Dimitrov M., Zhou H. – *Address-Branch Correlation: A Novel Locality for Long-Latency Hard-to-Predict Branches*, The 14th International Symposium on High Performance Computer Architecture, Salt Lake City, Utah, February 2008.
[Gel06] Gellert A., Florea A. – *Finding and Solving Difficult Predictable Branches*, Science and Supercomputing in Europe, Barcelona, Spain, 2006.
[Gel07] Gellert A., Florea A., Vinţan M., Egan C., Vinţan L. – *Unbiased Branches: An Open Problem*, Lecture Notes in Computer Science, Advances in Computer Systems Architecture, vol. 4697, pp. 16-27, Springer-Verlag Berlin / Heidelberg, 2007.
[Gel08] Gellert A., Florea A., Vinţan L. – *Exploiting Selective Instruction Reuse and Value Prediction in a Superscalar Architecture*, Submitted to Journal of Systems Architecture, ISSN: 1383-7621, Elsevier, 2008.
[Gzip] http://www.gzip.org/
[Jim05] Jiménez D. - *Idealized Piecewise Linear Branch Prediction*, Journal of Instruction-Level Parallelism, April 2005.
[Rab89] Rabiner L. R. - *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, Vol. 77, No. 2, February 1989.
[Sha05] Sharkey J., Ponomarev D., Ghose K., *M-SIM: A Flexible, Multithreaded Architectural Simulation Environment*, Technical Report CS-TR-05-DP01, Department of Computer Science, State University of New York at Binghamton, October 2005.
[Shi01] Shivakumar P., Jouppi N. P., *Cacti 3.0: An Integrated Timing, Power, and Area Model*, WRL Research Report, Aug 2001, USA.
[SPEC] SPEC2000, *The SPEC benchmark programs*, http://www.spec.org.
[Vin06] Vinţan L., Gellert A., Florea A., Oancea M., Egan C. – *Understanding Prediction Limits through Unbiased Branches*, Lecture Notes in Computer Science, Advances in Computer Systems Architecture, vol. 4186, pp. 480-487, Springer-Verlag Berlin, 2006.
[Vin08] Vinţan L., Florea A., Gellert A. – *Random Degrees of Unbiased Branches*, Submitted to Proceedings of the Romanian Academy, Series A: Mathematics, Physics, Technical Sciences, Information Science, ISSN 1454-9069, Bucharest, 2008.