CROSS-FERTILISATION BETWEEN COMPUTER ARCHITECTURE AND OTHER COMPUTER SCIENCE FIELDS

Lucian N. VINTAN, Adrian FLOREA

"L. Blaga" University of Sibiu, Computer Science Department
Str. E. Cioran, No. 4, Sibiu-2400, ROMANIA
E-mail: vintan@jupiter.sibiu.ro, aflorea@vectra.sibiu.ro

Abstract: The main aim of this paper is to prove that using new scientific tools in Computer Architecture research could be quite efficient and useful. We exemplified this idea during the paper, by analysing some innovative proposals in branch prediction research field. It's necessary that computer architects to be more opened at other Computer Science areas and research methods, in order to enlarge their view and to improve their results. Cross-fertilisation is seen as essential if microprocessors are to continue their exponential growth. In conclusion we consider that using some more mature theoretical research tools, belonging to different science fields, could be an excellent alternative approach in the present-day Computer Architecture research method.

Keywords: *Computer Architecture, Instruction Level Parallelism, Branch Prediction, Neural Nets, Genetic Programming, Simulation.*

1. INTRODUCTION

During the last 25 years, computer architecture researches have been followed an evolutionary path, based mainly on some quite old concepts and ideas. The performance growth over these years was really amazing but it's difficult to believe that it will be possible to follow the same path of development in the future. Thus, there are necessary some innovative ideas in this domain, revolutionary and evolutionary at the same time. Especially in Instruction Level Parallel Processors (ILPP) research field, the key seems to be an integrated approach based on a cross-fertilization between hardware and software scheduling and respectively between technology and architecture. Also, the performance centerpiece criteria needs to be accomplished by other very important interdependent quality criteria like availability, maintainability and scalability of services (Hennessy, 1999).

According to our opinion, the most promising present-day approach in ILPP domain consists in value prediction, dynamic instruction reuse (DIR) and multi-threading. The value prediction super-speculative technique is based on the value locality concept, a third facet of locality that is frequently present in programs. Value locality describes the likelihood of the recurrence of a previously - seen value within a storage location (Lepak and Lipasti, 2000) Researchers show that some micro-architectural enhancements that enable value prediction can effectively exploit value locality to collapse true dependencies, reduce memory latency and bandwidth requirements. On the other hand, empirical statistics suggest that many instructions and groups of instructions, having the same inputs, are executed dynamically (Sodani, 2000) due to some program characteristics (loops and recursion etc.). Such instructions do not have to be executed repeatedly because their results are obtained from a

reuse buffer where they were saved previously. In the case of reusing groups of true data dependent instructions, DIR can outperform the programs intrinsic sequentially ceiling (is Amdahl's low contradicted here?). Other feasible recent approaches in processor architecture research field are multiscalar processors, trace processors, data-scalar processors, multithreaded and simultaneous multithreading processors, multiprocessors on a chip etc. However, this innovative research way seems to be insufficient in our opinion.

At this moment the main (unique?) way we can understand Computer Architecture is by doing laborious simulations. But, the question is, this means a real profound understanding? From our point of view certainly not, because sometime is very difficult or even impossible an in depth understanding of the hardware- software interface in a qualitative manner, based only on benchmarking. Simulation is, of course, absolutely necessary but, unfortunately, it isn't sufficient for an in depth understanding. Sometimes, the obtained quantitative results represent only effects and not the real causes of some processing phenomena.

Treated as separate entities, hardware and software seem to be really "sciences". As an example, automata theory, circuit complexity, digital design etc. could be considered, at this moment, being mature theoretic "hardware" fields. On the other hand, pure "software" domains like computer algorithms and complexity theory, formal languages or compiling theory could be also considered "sciences". But could we consider Computer Architecture, therefore a hardware-software area, being really a science? The answer is obviously no, because in this field, situated at the hardware-software interface, the main approach is based on benchmarking and simulation rather than mathematical models. That means a lot of empirism, heuristics, "strange" optimisation techniques, principles and rules (ex. "90/10 rule") but not a logic formalised mature theory. Why this ? Perhaps because the "hardware-software" interface represents a too complex synergy concept and - up to this moment - we are unable to formalise and understand it **in a real qualitative manner.**

In our opinion, taking into account all these briefly presented aspects, Computer Architecture's pure evolutionary era must be finished. Perhaps, in the nearest future, there are necessary some new revolutionary architectural concepts, too. The alternative seems to be a more integrated approach that mainly means a synergism between the following aspects:

- technology respectively architecture, concepts, algorithms and methods
- hardware, software and applications - not treated as separate entities (Patt and Patel, 2001; Hennessy, 1999) (as an example, for some object oriented programs, where more indirect branches may be executed, establishing an optimal hardware branch predictor scheme still represents an open problem).
- different (Computer) Science areas and techniques working together for a certain Computer Architecture challenge

The main aim of this paper is to prove that this last new proposed synergism could be quite efficient and useful. We'll exemplify this new idea during the next sections, by analysing some innovative proposals in branch prediction research field (neural branch predictors, markovian predictors, genetic predictors etc.)

2. A NEURAL BRANCH PREDICTOR

As it's presented in (Steven and Morales, 2000; Vintan and Steven, 2000)), an interesting alternative approach in branch prediction research is to look to other computer science application areas for novel solutions to this important problem. So, we proposed the application of neural networks to dynamic branch prediction. In this sense it's developed a Multi- Layer Perception (MLP) with a single intermediate layer neural network accomplished with Backpropagation learning algorithm. We compared this neural branch predictor with a Two Level Adaptive Branch Predictor named GAp (Yeh and Patt, 2000) according to Yeh & Patt's taxonomy. The proposed GAp predictor uses an *k* bit shift register (HRg) to record the outcome (Taken/Not Taken) of the last *k* branches executed. The per-address Prediction History Table (PHT) is accessed by concatenating the PC address with HRg. In this experiment, each PHT entry consists in a tag field, the branches target address and only one prediction bit. The PHT was considered of unlimited size.

In order to allow a direct comparison with this conventional GAp predictor the neural branch predictor receives on its entries the PC concatenated with HRg, too. The MLP then produces a true output for predicting taken and respectively a false output for predicting not taken. There are considered here two distinct training processes: static and dynamic. Static training was achieved by generating a training vector set, consisting of (PC, HRg) pairs, for each benchmark. First, the benchmarks instruction trace was pre-processed to determine how frequently individual branches were taken for each HRg specific pattern. Some of the branch outcomes (Taken/Not Taken) are highly biased with branches being taken

(or not taken) most of the running time. In contrast, other branches are very hard to be predicted on the basis of the given information. Only highly biased (PC, HRg) pairs were included in the training set. The statically trained predictor was then used in a final trace-driven simulation. On the other hand, the dynamically training process is done during the trace processing, thus without a statically pre-training. Figure 1 presents the improvement introduced by the statically training process. As it can be observed, the statically learning algorithm involves an average prediction growth of about 0.8 percents.

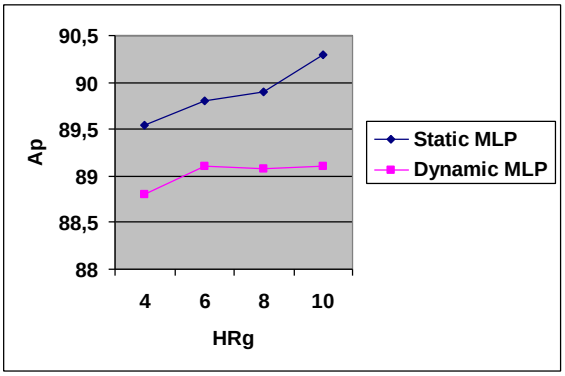

Figure 1. A Statically Trained vs. a Dynamically Trained neural predictor

Another used "static" training method consists in learning all the traces in a random order, until the recognition error became smaller than a certain limit. Figure 2 compares the proposed neural branch predictor with an equivalent GAp predictor considering different values of correlation history length (bits of HRg register). As it can be observed, the neural branch predictor is consistently more accurate than the GAp predictor by a margin of around 2-3%.

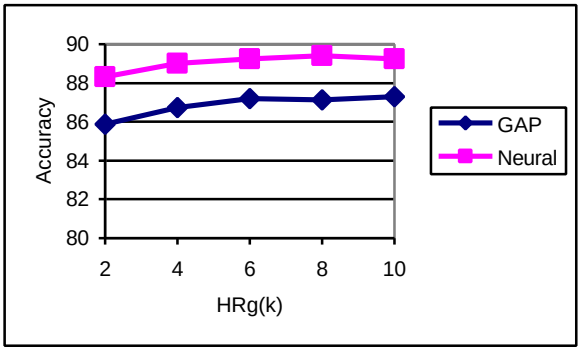

Figure 2. An “unlimited” GAp Predictor vs. a Neural Branch Predictor

As a partial conclusion, clearly, neural networks are a useful vehicle for future branch prediction research, even as an upper metric of branch predictability, useful in quantitative evaluations of (classical) branch prediction schemes.

3. OTHER NEURAL APPROACHES IN COMPUTER ARHITECTURE

Also an interesting neural nets approach in branch prediction was proposed by Calder (Calder *et al*, 1995). The authors discussed the application of neural nets to the problem of static branch prediction at compile time. The prediction methodology is therefore entirely based on information about program’s structure that can be readily determined by a compiler. As an example, a branch successor path that leads out of a loop or function is less likely to be followed than a path that remains within the loop or function. In our opinion, the main contribution of this work consists in determining a static feature set of information related to the important static elements of the corpus of programs. So, in this set, there are included some branch features and also some features of the taken/not taken successor of the branch. All these static features that are correlated with branch outcomes are coded as binary inputs in the static neural predictor. Therefore a neural network is used to map static features associated with each branch to the probability that the branch will be taken. The main idea of this static neural branch predictor is that the behaviour of a corpus of programs (training set) can be used to infer the behaviour of new programs. Static branch prediction is important especially for compilers because it contributes to implement optimisations like trace-scheduling and other profile-directed optimisations (Calder *et al*, 1995). Using this approach, the authors reported a misprediction

rate of only 20%, remarkably low for static branch prediction. Another interesting neural approach applied in Computer Architecture was presented in (Khalid, 1995). This paper presents an original neural algorithm (KORA) that uses a back propagation neural net in order to guide the block replacement decisions made by the optimal algorithm (Belady). The key consists in identifying and discarding the dead blocks in cache memories. More precisely, the neural net is used here to identify two distinct types of cache blocks: *transient* blocks (new arrivals to the cache) respectively *shadow* blocks (recently replaced from the cache) stored in a so named *shadow directory*. As it's known, there is a significant statistical probability that the shadow blocks are going to be referenced soon. Therefore, the neural net identify these shadow blocks and provide them with a preferential treatment. If the transient blocks do not become active during a certain time period, they are removed from the cache. The corresponding trace driven simulation results indicate that this neural replacement algorithm achieves approximately 8% performance improvement over a LRU scheme, measured for the SPEC benchmarks. This idea seems to open new research directions when applied to the page replacement algorithms in virtual memory systems and disk caches.

Related to the problem of new original replacement algorithms, we found at least another innovative paper. This is Stiliadis & Varma paper, that introduced a new heuristics named "Selective Victim Cache" in order to minimise the number of replacements between the main cache and the victim cache. In our opinion there is a similarity between the Selective Victim Cache and respectively the shadow directory proposed by Pomerene and improved by Khalid through neural nets algorithms.

4. A MARKOVIAN BRANCH PREDICTOR

During this section it was analysed a markovian branch prediction scheme, first introduced by Trevor Mudge (Mudge et al,1996). This new approach was investigated based on a trace driven simulation methodology in our previous work. The prediction is based here on the PPM (Prediction by Partial Matching) algorithm that represents an universal compression/prediction algorithm. PPM has been theoretically proven optimal in data compression and prefetching and also in some speech recognition problems. The bases of the PPM algorithm of order m are a set of (m+1) Markov predictors. A Markov predictor of order j predicts the next bit based upon the j immediately preceding bits pattern (a simple Markov chain). More precisely, the prediction process counts every time when that pattern on j bits was found, if a '1 'or respectively by a '0' followed it. The prediction is according to the most frequent bit that follows the searched and founded pattern. PPM uses the m immediately preceding bits to search a pattern in the highest order Markov model, in this case m. If the search succeeds, which means the pattern appears in the input sequence seen so far, PPM predicts the next bit using this m-th order Markov predictor. However, if the pattern is not found, PPM uses the (m-1) immediately preceding bits to search the next lower order (m-1)-th order Markov predictor. Whenever a search misses, PPM reduces the pattern by one bit and uses it to search in the next lower order Markov predictor. This process continues until a match is found and the corresponding prediction can be made.

If we consider the search pattern for every Markov predictor as a HRg (Global History Register) pattern, it is proved (Mudge et al, 1996) that the complete PPM predictor can be viewed as a set of Two Level Predictors, having not one size of HRg but a set that spans m down to 0. It's obviously that this PPM branch predictor generalises the Two Level Adaptive Predictors. To implement this Markov predictor in hardware would require almost double of the hardware resources, straining the limits of practicality. Trying to avoid this circuit complexity, we investigated a simplified PPM branch predictor consisted by only one Markov predictor of order m, quite feasible to be implemented in hardware. Surprisingly, in both cases the obtained prediction accuracy are quite similar, showing clearly that a simplified PPM predictor outperforms a classical scheme (GAg in our experiments) and also it would be feasible to be implemented in hardware, the results being in a perfect concordance with those published by other researchers (Mudge et al,1996). Dynamic branch prediction in high-performance processors is a specific instance of a general Time Series Prediction problem that occurs in many areas of science. In contrast, most current branch prediction researches are focused on Two-Level Adaptive Branch Prediction techniques, a very specific solution to the branch prediction problem. An alternative approach is to look to other application areas and fields for novel solutions to the problem. In our opinion the main merits of this markovian approach is that it investigates the branch prediction problem as a mature scientifically problem, involving also corresponding research tools.

5. GENETIC BRANCH PREDICTORS

In this section it is discussed an interesting approach in order to search for new branch predictors using a genetic programming method. Emer and Gloy (Emer

and Gloy, 1997) first developed this interesting idea. The authors proposed a language named BP that is an algebraic-style branch predictor language, based on the observation that every predictor can be divided in several elementary blocks such – memories, functions and numbers etc. The actual Two Level Adaptive predictors can be divided into two distinct classes: static and dynamic. In the case of a static predictor the prediction is always the same logical function. On the other hand, dynamic predictors learn to make better predictions using information that is only available after the prediction is made. Dynamic predictors thus use feedback to learn from past behaviour and hence make better predictions in the future.

In order for such a feedback control system to learn, it needs some sort of memory. To provide this memory was defined a primitive. This primitive noted *P [w, d] (I; U),* is basically a memory that is w bits wide and d entries deep. This memory has two types of operations: predict and update. The predict phase consist in accessing memory at address index I, and the value read is used as the prediction P. Some time later, when the branch is resolved, an update value U is delivered to the predictor and written into the same location indexed by I. The static parameters (w,d) allow describing a class of predictors of various sizes. The dynamic parameters (I,U) are partitioned between the input arguments, listed first, and the update arguments, listed after the semicolons (;). Using of these predictors can be thought of as inputting a series of index (I), generating a series of predictions (P) and, when the prediction resolves, updating the predictor's state (U). By using specific values or expression as the input to the predictor, it can generated a variety of predictors. The classical 1 bit predictor can be represented in the BP language as:

Onebit[d](PC;T) = P[1,d](PC;T)

where,

PC = current Program Counter (index value)
T = branch resolution (update value)
(0-> not taken, 1 -> taken)

Also it can be defined the predictor “Onebit” with parameter d, input PC and update value T. This predictor can be parameterised by its depth noted d. For example if it is chosen d = 2048 (2K) it can be written:

Onebit[2048](PC;T) = P[1,2048](PC;T)

In the same formal manner it can be developed more complex structures. So, below is presented using the BP syntax, an array of n-bit saturating counters each of which counts up or down based on their update value.

Counter[n,d](I; T) = P[n,d](I; if T then P+1 else P-1)

The adding and subtraction operations are in this case saturating operations. If we combined this counter with function MSB – which return the most significant bit of a value - it can described well known predictors like:

Twobit(d)[PC; T) = MSB(Counter[2,d](PC; T))

and respectively

Twobit(d)[PC; T)=MSB(P[2,d]((I; if T then P+1 else P-1)).

This designing approach is based on automatic search for predictors using some Genetic Programming concepts. Genetic programming is derived from Genetic Algorithms that represent a method for efficiently searching extremely large problem spaces. A genetic algorithm encodes potential solutions to a given problem as fixed - length bits. In the BP representation each bit represents a language atom (function, memory, terminal etc.). To generate the initial population is necessary to create the individuals. Each individual is created using a random algorithm. Individuals are represented by a tree structure, which is easily translated into a corresponding expression in the BP language and viceversa. First, it's necessary to evaluate the fitness – prediction accuracy in this case - of each individual through branch predictor simulation. The next step consists in creating new individuals derived from old ones by applying genetic operators that recombine the components of the old individuals in different ways. In this manner will be developed a new generation of genetic branch predictors. The individuals that serve as inputs to the genetic operations are chosen with a probability based on their fitness value. Individuals with a higher fitness value have a higher probability of being chosen, so that they may appear many more times than individuals of lower fitness value. This means that the next generation will contain many individuals having one or more components from successful individuals, which makes it likely for having a better average fitness rate than the previous generation. By repeating this process many times, are produced successive generations. According to the authors, somewhere between 15 and 30 generations the experiments converge to a few distinct predictors. From the prediction accuracy point of view they are comparable with de most advanced human designed branch predictors. Unfortunately, they are logically much more complex and probably not directly implementable (Emer and Gloy, 1997). Also the computing process is very time consuming, therefore there are necessary to be used powerful computer systems. However, in our opinion, the paper opens a new era in branch prediction research and, more generally, in

finding new innovative efficient digital structures, useful in different Computer Engineering fields.

6. CONCLUSIONS

According to our opinion, the present-day research paradigm in Computer Architecture field is too specialised and therefore, too limited. The corresponding research tools are quite old. It's necessary that computer architects to be more opened at other Computer Science areas and research methods, in order to enlarge their view and to improve their results. During this paper we presented the necessity of a more integrated approach in Computer Architecture field where – as we previously claimed – the present day paradigm must be enlarged and improved. More precisely, we consider that using some more mature theoretical research tools, belonging to different Computer Science fields, could be an excellent alternative approach in the present - day Computer Architecture research methodology. This new research idea is concretely exemplified by considering as an example, the branch prediction challenge, through some very original papers, unfortunately not very well-known. So, firstly it's presented the neural dynamic branch predictor that uses a backpropagation neural network in order to make predictions. Subsequently, it's presented another interesting paper that also uses a neural net related to the block replacement decision in a cache memory. After this, it's analysed a markovian branch predictor first proposed by T. Mudge (Mudge et al,1996). This branch predictor generalised the well-known Two Level Adaptive Branch Predictors. Finally, it's briefly analysed, a very original paper that proposes an innovative technique for developing branch prediction schemes using some genetic programming concepts.

From our point of view, an important common merits of all these researches consist in integrating a pure certain Computer Architecture problem with other interesting theoretical Computer Science areas and methods. Therefore, all these presented papers are exemplifying our initial idea: progresses in Computer Architecture research are possible in the nearest future, using a synergism between different mature science fields and techniques. It's essential trying to explore the potential benefits that could be realised through a cross-fertilisation of ideas between Computer Architecture and other Computer Science domains. We think that other similar original approaches, like those presented here, are very useful in this new Computer Architecture era.

ACKNOWLEDGMENTS

This work was supported in part by the Romanian National Agency of Science, Research and Technology grant ANSTI No. 6229 (B18) / 2000**,** respectively by the Romanian National Council of Academic Research grants CNCSIS No. 8 / 2000. Our gratitude to Professor Gordon B. Steven and Dr. Colin Egan from the University of Hertfordshire, UK, for their very useful and highly appreciated support, suggestions and encouragement related to our Instruction Level Parallel Processors research.

REFERENCES

Calder B., Grunwald, D., Lindsay, D. – *Corpus based Static Branch Prediction*, ACM Sigpon Notices, vol. **30**, No. 6, June, 1995.

Emer J., Gloy N. – *A Language for Describing Predictors and it's Application to Automatic Synthesis*, ISCA97, Denver, USA, 1997.

Hennessy J. – *The Future of Systems Research*, Computer, August 1999.

Khalid H. – *A New Cache Replacement Scheme based on Backpropagation Neural Networks*, Computer Architecture News, May, 1995.

Lepak K., Lipasti M. – *On the Value Locality of Store Instructions*, ISCA00, Vancouver, Canada, 2000.

Mudge T. Chen, I., Coffey, J. – *Limits to Branch Prediction*, Technical Report, University of Michigan, January 1996.

Patt Y., Patel S. – *Introduction to Computing Systems: From Bits and Gates to C and Beyond, McGraw-Hill*, 2001.

Sodani A. – *Dynamic Instruction Reuse*, PhD Thesis, Univerdity of Wisconsin – Madison, USA, 2000.

Steven G., Morales R. A. – *Using Neural Networks to Perform Dynamic Branch Prediction in a High Performance Superscalar Architecture*, Technical Report, University of Hertfordshire, UK, January 2000.

Stiliadis D., Varma A. – *Selective Victim Caching: A Method to improve the Performance of Direct Mapped Caches*, Technical Report, University of California, 1994.

Vintan L., Steven G. – *Investigating a New Dynamic Neural Branch Predictor*, Transactions on Automatic Control and Computer Science, vol **45**, no. 4, Timisoara, Romania, 2000.

Yeh T. Patt Y. – *Alternative implementations of Two Level Adoptive Branch Prediction*, 19th Annual Int'l Symposium on Computer Architecture, 1992.