PREDICTION AND SPECULATION TECHNIQUES IN ILP

P. Mitrevski, M. Gusev*, A. Misev*
Faculty of Technical Sciences, St. Kliment Ohridski University
I. L. Ribar bb, PO Box 99, 7000 Bitola, Macedonia
pece.mitrevski@uklo.edu.mk
*Faculty of Natural Sciences, St. Cyril and Methodius University
Arhimedova bb, PO Box 162, 1000 Skopje, Macedonia
{marjan, infolab}@pmf.ukim.edu.mk

Abstract: *In this article we review the concepts of branch, value and memory prediction used in conjunction with control and data speculative execution in superscalar processors. Since the amount of available instruction level parallelism within a basic block is relatively small, control speculation techniques increase the number of candidate instructions for execution. Moreover, the integration of value and memory prediction in superscalar processors introduces a new kind of speculative execution. Data speculation techniques allow the processor to execute instructions beyond the limit of true data RAW dependencies. We identify data speculation as a natural extension of control speculation and capture their similarities and differences. We also raise a new question: is the analytical modelling approach really infeasible?*

Keywords: superscalar processor, control and data speculation, branch prediction, value prediction, and memory prediction

1. Introduction

Dependence in program executions as a general term is a relationship between items in two instructions that dictates the execution order. Items in the instructions can be data operands, result or instruction address i.e. order in its execution, as defined in [7]. Two types of dependencies can be determined. The first group form control and resource dependencies as a relationship when the program execution order is affected because of the state of the processor and the type of the instruction. *Control dependence* is a relationship between two instructions according to conditions that dictates their execution order, i.e. when the execution of one instruction controls whether the second instruction should be executed or not. *Resource dependence* is a relationship between two instructions according to conditions that enables resources for their executions, i.e. when the execution of one instruction occupy the resources required for the execution of the second instruction. The second group contains data dependencies. *Data dependence* is the relationship between data items in two different instructions, which dictates their program execution order.
There are different taxonomies such as [8] that determine name, control and data dependencies, and several others, but all of them are subset of the taxonomy in [7].
There are several techniques to eliminate the effect of dependencies or to reduce their consequences. For example, data forwarding and implementation of shelving reduce the effect of RAW dependencies and register renaming eliminates the false or name (WAR and WAW) dependencies. However, the potential of new processors with hundreds of processing units requires more parallelism then the one obtained by these techniques. One alternative is introducing of various speculation techniques based on prediction. The other alternative is introducing more advanced elimination techniques.

2. Taxonomy of prediction techniques

Many superscalar processors speculatively execute control-dependent instructions before resolving the branch outcome. They rely upon *branch prediction* techniques in order to tolerate the effect of control dependencies. Data dependencies can be eliminated at run-time by predicting the outcome values of instructions (*value prediction*) and by executing the true data RAW dependent instructions. The execution becomes speculative when it is not assured that true data RAW dependent instructions were fed with correct input values. Since the correctness of execution must be maintained, speculatively executed instructions retire only if the prediction was proven to be correct – otherwise, they are discarded.

Some authors [1,6,11,13,16] consider value prediction as generalization whenever a result value is expected as outcome of instructions that introduce true data (RAW) dependencies. However it is natural to divide this class to two classes upon the instruction type. Therefore we identify value prediction as prediction of the outcome of arithmetic instruction and memory prediction as prediction of the outcome of memory access instructions.

All these prediction techniques used in conjunction with speculative execution in superscalar processors are our main concern in this paper. There are several taxonomies of speculative execution techniques, such as Lipasti and Shen [11] and others. However we introduce new taxonomy for prediction based on: instruction type and determination of addresses and values, as presented in Fig.1.Branch prediction and control speculation can be realised by predicting the direction (taken/not-taken) or the target (address or instruction). Value prediction is used whenever an arithmetic instruction is going to be skipped although its execution is delayed because of true data (RAW) dependencies. Memory prediction concerns memory data and address dependencies. Whenever memory access instruction is scheduled, there are two steps of execution. The first step considers address generation. It can use true data (RAW) dependent operands and therefore the execution is delayed. One possible way to introduce speculation is to predict the outcome of the address generation step i.e. to predict the address. In speculation we can go even further to predict the data value of the memory load instruction.

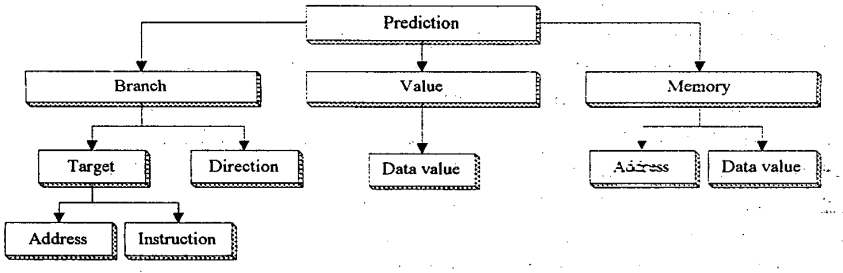

Figure 1. Taxonomy of prediction techniques.

3. Branch prediction and control speculation techniques

The branch performance problem can be divided into two sub problems [11]. The first is a prediction of the *branch direction* that requires *binary* decision (taken vs. not-taken). The second sub problem is providing the instructions from the *branch target* to be available for execution with minimal delay. The latter requires a *multi-valued* decision, since the target can be anywhere in the program's address space.

Many branch prediction schemes have been explored in the literature [9,12,18,19]. They can range between no prediction, fixed, static displacement based, static with profiling, and various dynamic schemes, like BHT with n-bit counters, BTAC, BTIC, mixed etc. Yeh and Patt [19] describe and characterize possible variations of their adaptive branch prediction model according to the manner in which the first-level branch history register or second-level pattern history table is kept:

- G (g) – outcomes of different branches update the same *global* branch history register (table),
- S (s) – a set of branch instructions share the same branch history register (table),
- P (p) – branch history is kept for each distinct static branch instruction individually (each static branch instruction has its own pattern history table),

This yields nine variations of the adaptive (A) model: GAg, GAs, GAp, SAg, SAs, SAp, PAg, PAs and PAp. With respect to the cost-effectiveness of different variations, PAs turns to be the most effective among low-cost schemes, and GAs among high-cost schemes with the best average prediction accuracy of 97.2 percent. A good survey of branch prediction strategies and measurements of their accuracy, their comparison can be found in [15].

Are there limits to branch prediction? That is the question Mudge *et al.* try to answer in [14]. They apply two different approaches to measure inherent limits of branch prediction. For simple programs, they analyse asymptotic predictability exactly, but in dealing with more complicated programs, they use well-understood universal compression/prediction algorithms, such as *Prediction by Partial Matching* (*PPM*), as a measure of inherent predictability. The bases of a 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), which, in turn, exhibit strong similarities with two-level adaptive branch predictors. Since the complexity of PPM is roughly twice that of a two-level predictor using same history, it may only suggest some improvements to existing predictors.

Being on the right course, in a recent work Reinman *et al.* present an architecture that decouples the branch predictor from the instruction fetch unit [16]. They introduce a *Fetch Target Queue* (*FTQ*) that is inserted between the branch predictor and the instruction cache, allowing them to operate relatively autonomously. If the branch predictor has a multi-level hierarchy or requires multiple cycles to access, the instruction cache can continue fetching from the FTQ while the predictor is stalled or in mid-access, or the branch predictor, on the other hand, can work ahead of the instruction cache when the cache is stalled due to a miss or a full instruction buffer.

4. Value prediction and data speculation techniques

Another major hurdle to ILP processing is the presence of *data dependencies*, which prevent the execution of instructions in parallel. If an instruction is data dependent on a preceding instruction, it can be executed only after the preceding instruction's result becomes available. For a long time in the past, these constraints imposed by true data dependencies were regarded as an *absolute limit* on parallel execution of serial programs. However, Lipasti *et al.* show that it is possible to overcome the hurdles imposed by data dependencies with the use of *data value prediction* [10], [11]. They conclude that although a 32-bit register can contain any one of over four billion values, the task of data value prediction becomes much easier when each static instruction is considered individually. This technique is built on the concept of *value locality* [10], as the likelihood of a previously seen value recurring repeatedly within a storage location inside a computer system. Their research summarizes value locality of memory load instructions and instructions that write to registers. The former is called load value locality and the other is register value locality. With a history depth of 1, most of integer benchmark programs exhibit load (register) value locality in 50% (40-50%) range, while extending the history depth to 16, it can improve to more than 80% (60-70%) range. This means that a majority of static instructions exhibit very little variations in values that they write during the course of a program's execution.

Several architectures have been proposed for value prediction [1] including: last value prediction, stride prediction, context predictors and hybrid approaches. Lipasti and Shen [10,11] describe and evaluate a *Value Prediction Unit* (*VPU*) for predicting the results of instructions at dispatch by exploiting the affinity between instruction addresses and the values they produce. There are similarities between the BTAC (with associated prediction bits) and the VPU. The latter consists of

a direct mapped *Value Prediction Table* (*VPT*) for generating value predictions (history depth can be greater than one) and a *Classification Table* (*CT*) for deciding which predictions are likely to be correct (a table of 1- or 2-bit saturating counters which are incremented whenever the predicted value is correct and decremented otherwise). One might think the work of Lipasti *et al.* is a *generalization* of the well-known control speculation techniques. Dynamic branch prediction can be considered a restricted application of data value prediction i.e. prediction of a single condition bit based on its past behaviour. Even though, it is not possible to directly apply branch prediction techniques to data value prediction. Branch prediction deals with 1-out-of-2 predictions, while value prediction deals with 1-out-of-2^W predictions (W is the word size of the computer).

The previous scheme actually keeps track of the last value of an instruction. A good heuristic would be to record the recent results (values) produced by several consecutive instances of an instruction, and predict the result of the instruction's next instance based on past results [17], that forms a context predictor. The main part of such a predictor is the *Value History Table* (*VHT*). Usually, each entry has a *tag* to identify the instruction currently mapped to that entry.

Wang and Franklin investigate the potential of monitoring the *strides* by which the results produced by different instances of an instruction change, as well as the potential of pattern-based *two-level prediction schemes* (extensively exploited for branch prediction) [17]. In general, by storing a maximum of 2^n ($n << W$) most recent unique values (in the VHT of the first level), and by doing a binary encoding of these outcomes, behaviour patterns can be captured using a two-level predictor that performs 1-out-of-2^n predictions. The VHT incorporates a *Value History Pattern* field that stores the last p outcomes of an instruction as a $n \cdot p$-bit pattern. For each $n \cdot p$-bit pattern, a condensed history of the previous outcomes of the pattern is recorded in the *Pattern History Table* (*PHT*) of the second level, by means of 2^n independent up/down counter values.

The experimental results show that it is better to use *hybrid predictors* to get good accuracy over a set of programs, due to the different data value locality characteristics that can be exploited only by different schemes. Wang and Franklin investigate two hybrid predictors. The first one is a combination of *last outcome-based* prediction and *stride-based* prediction, while the second one is a combination of *two-level* prediction and *stride-based* prediction.

5. Memory prediction and data speculation techniques

Data speculation techniques [11] break down in two categories: those that speculate on storage location and those that speculate on actual data value. Furthermore, techniques that speculate on the location come in two fundamentally different flavours: those that speculate on specific attribute of the storage location (speculative disambiguation) and those that speculate on the address (most prefetching techniques). The techniques that speculate on data values can be divided into 2 subclasses: data value speculation and data dependence speculation. In data value speculation an attempt is made to predict the data value that an instruction is going to produce [11]. In data dependence speculation, no explicit attempt is made to predict the data values. Instead, a prediction is made on whether the input data value of an instruction has been generated and stored in the corresponding named location (memory or register) [13]. Although the techniques introduced by Mochovos and Sohi [13] look like register renaming its' realization through memory aliasing is much more complicated by using store queues and different structres like MPT (Memory Predict Table) etc. We empasize that actually we have to differ where the memory prediction and data speculation can be used. One possible implementation is in the instruction window where an instruction is waiting for data value from another memory access instruction in the same window. Another implementation is possible while the memory instruction is executing after the address generation step. This means that the instruction has to wait in buffer waiting other stores to be finished. Although the address can be predicted or already caluclated, still the memory instruction

delays further execution because of memory dissambiguation and other memory dependences. In this case there are a lot of shortcut techniques for elimination or predicting the data value.
Gonzalez and Gonzalez notice that load/store instructions are very good candidates for speculative execution since their effective address is highly predictable. They propose a novel technique called *Memory Address Prediction* (*MAP*) that implements speculative execution of load/store instructions in an out-of-order processor [3]. Later on, the processor is allowed not only to predict the current effective address of either a load or a store instruction, but also the next effective addresses of loads and prefetch the data from memory into a *Memory Prefetch Table* (*MPT*). This technique allows load instructions to obtain values at decode stage [4]. A side-effect benefit of data value speculation is the reduction of branch misprediction penalty, since branch outcomes are computed earlier. By exploiting this phenomenon, they introduce even a new branch predictor that predicts the outcomes of branches by predicting the value of their inputs and performing an early computation of their results according to the predicted values [6].

6. The problem of quantifying the performance of speculative execution techniques

Vast majority of the studies on branch, value and memory prediction techniques have something in common: quantifying the ability of predictions to boost the ILP from an experimental point of view – either by measurements taken in a real system, or by simulation [7].
Only Gabbay and Mendelson analyse the dataflow graph presentation of a program and evaluate the execution time of the critical path for both finite and infinite instruction window size when value prediction is employed [2]. This yields closed form expressions for the *average execution time* of the critical path and the *average boost in the ILP* of the critical path. Their work is a sole effort to provide an analytical model that can be used to better understand the characteristics of value and memory prediction and to obtain a rough estimation of the potential of using them.
Analytical modelling is an attractive alternative to measurement or simulation method. A lot of papers are concerned with the distribution of instructions in superscalar processors. Instructions can efficiently be identified as belonging to several different classes and each of them can be assigned some probability or rate as a measure of the *relative frequency of occurrence* in the total instruction count for dynamically committed instructions. In the same manner, predictions that are to be made can be assigned probabilities as a measure of the *average prediction accuracy*. Latencies of different classes of instructions, misprediction rates and penalties (in clock cycles) are known, too. In addition, the beginning of each cycle can be observed as a regeneration point, because the past history is completely summarized in the current machine state. This pieces of evidence can be used in modelling the operational environment (sample data set), as well as the dynamic behaviour, using probabilistic analysis, elements of the theory of stochastic processes and the renewal theory or, preferably, various classes of Timed Petri Nets, which are well known formalism for the description and analysis of variety of systems where concurrency and parallelism are inherent. Since the complex behaviour of the system can be concisely represented, the Petri Net model of a system is usually easier to understand than a pure probabilistic analysis model. However, this approach may also have an intrinsic problem: the reachability graph underlying the Petri Net increases very rapidly as the model size increases. Therefore, the abstraction of the real-world system into a set of parameters should be done carefully. Otherwise, trying to avoid trace-driven simulation, one may end up with simulation of the stochastic behaviour of the Petri Net. The goal of such an analysis would be quantifying the impact of speculative execution techniques on system level performance.

7. Conclusion

We propose a new taxonomy of prediction techniques based on instruction type. Effective control and data speculation techniques are essential to explore the full performance of modern superscalar processors as they move towards wider issue and deeper super-pipelines. Various branch prediction

schemes which significantly increase the performance by eliminating the instruction fetch stalls in the pipelines have been around for a long time, whereas value prediction, as a natural extension, is still gaining popularity. Yet, speedup obtained by data speculation is significant for a limited window, and huge for an infinite window. Researchers still have to emphasise designs (both control and data speculative) that are faster, less complex, more testable and easy to implement. We also believe that a variety of elements of the probability theory, statistics, stochastic processes and renewal theory can still be identified in this area and used as a starting point for an analytical modelling approach.

8. References

1. Calder,B., Reinman,G., Tullsen,D., (1999), *Selective Value Prediction*, 26th Annual Int. Symp. Computer Architecture, Atlanta, USA
2. Gabbay,F., Mendelson,A., (1998), *Using Value Prediction to Increase the Power of Speculative Execution Hardware*, ACM Trans. Computer Systems, Vol.16, No.3, pp. 234-270
3. Gonzalez,J., Gonzalez,A., (1997), *Memory Address Prediction for Data Speculation*, Proc. of EUROPAR 97, Passau, Germany
4. Gonzalez,J., Gonzalez,A., (1997), *Speculative Execution via Address Prediction and Data Prefetching*, Proc. of the ACM Int. Conf. Supercomputing, pp.196-203, Vienna, Austria
5. Gonzalez,J., Gonzalez,A., (1999), *Control-Flow Speculation through Value Prediction for Superscalar Processors*, Proc. of the Int. Conf. Parallel Arch. and Compilation Tech. (PACT)
6. Gusev,M, Misev.A, Popovski.G., (1999), *Memory Address Dependencies*, ITI-99, pp.191-196, Pula, Croatia.
7. Hennessy,J.L., Patterson,D.A., (1996), *Computer Architecture: A Quantitative Approach*, Second Edition, Morgan Kaufmann Pub., San Francisco, California
8. Lee J.K.F., Smith A.J. (1984), *Branch Prediction Strategies and Branch Target Buffer Design*, IEEE Computer, January 1984, pp. 6-22
9. Lipasti,M., Wilkerson,C., Shen,J.P., (1996), *Value Locality and Load Value Prediction*, 7th Int. Conf. Architectural Support for Programming Languages and Operating Systems, pp. 138-147
10. Lipasti,M., Shen,J.P., (1996), *Exceeding the dataflow limit via value prediction*, 29th Annual ACM/IEEE Int. Symp. Microarchitecture, pp. 226-237.
11. McFarling,S., (1993), *Combining Branch Predictors*, TN-36, DEC, Western Research Lab
12. Moshovos,A., Breach,S., Vijaykumar,T., Sohi,G., (1997), *Dynamic Speculation and Synchronization of Data Dependencies*, 24th ISCA, Denver, USA
13. Mudge,T., Chen,I.K., Coffey,J.T., (1996), *Limits to Branch Prediction*, Technical Report CSE-TR-282-96, University of Michigan
14. Popovski.G., Gusev,M, Misev.A, (1999), *Quantifyng the Performance of Branch Prediction Scheme with the Superscalar Simulator*, ITI-99, pp.219-224, Pula, Croatia.
15. Reinman,G., Calder,B., Austin,T., (2000), *Building a Scalable Branch Predictor and an Instruction Prefetch Engine by Decoupling Branch Prediction from Instruction Fetch*, Technical Report CS00-645, University of California, San Diego
16. Wang,K., Franklin,M., (1997), *Highly Accurate Data Value Prediction using Hybrid Predictors*, 30th Annual Int. Symp. Microarchitecture
17. Yeh,T.Y, Patt,Y.N., (1992), *Alternative Implementations of Two-Level Adaptive Branch Prediction*, 19th Annual Int. Symp. Computer Architecture, pp. 124-134, Gold Coast, Australia
18. Yeh,T.Y, Patt,Y.N., (1993), *A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History*, 20th ISCA, pp 257-266, San Diego, California