PERFORMANCE EVALUATION OF BRANCH AND VALUE PREDICTION USING DISCRETE-EVENT SIMULATION OF FLUID STOCHASTIC PETRI NETS

P. Mitrevski[1], M. Gušev[2]

[1]Faculty of Technical Sciences, St.Clement Ohridski University
Ivo Lola Ribar b.b., 7000 Bitola, Macedonia
pece.mitrevski@uklo.edu.mk

[2] Institute of Informatics, Faculty of Natural Sciences and Mathematics,
Sts. Cyril and Methodius University
Arhimedova bb, PO Box 162, 1000 Skopje, Macedonia
marjan@ii.edu.mk

Abstract: Speculative execution is one of the key issues to boost the performance of future generation microprocessors. In this paper, Fluid Stochastic Petri Nets are used to capture the dynamic behavior of an ILP processor, and discrete-event simulation is applied to assess the efficiency of predictions and speculative execution in boosting the performance of ILP processors that fetch, issue, execute and commit a large number of instructions per cycle. The evaluation leads to numerous conclusions regarding the performance impact of branch and value prediction under different scenarios, with varying parameters of both the micro-architecture and the operational environment – branch and value prediction accuracy, machine width, instruction window size and operational profile.

Keywords: Instruction Level Parallelism, speculative execution, Branch Prediction, Value Prediction, Fluid Stochastic Petri Nets, discrete-event simulation

1. Introduction

Most of the recent microprocessor architectures assume *sequential programs* as input and use a *parallel execution* model. Their efficiency is highly dependent on both the hardware mechanisms and the program characteristics, i.e. the *instruction-level parallelism* (ILP) the programs exhibit. Instructions cannot be issued for parallel execution due to three types of constraints: *control dependences*, *false (name) dependences* and *true data dependences* [19]. Several techniques eliminate the effect of dependences or reduce their consequences – *data forwarding* and implementation of *shelving* reduce the effect of true data dependences (*read after write*), while *renaming* eliminates the false or name dependences

(*write after read* and *write after write*). In addition, many ILP processors speculatively execute control-dependent instructions before resolving the branch outcome. They rely upon *branch prediction* in order to tolerate the effect of control dependences. A variety of branch prediction schemes have been explored [5,22,26,39,44,45] – they range between *fixed*, *static displacement-based*, *static with profiling*, and various dynamic schemes, like *Branch History Table with n-bit counters*, *Branch Target Address Cache*, *Branch Target Instruction Cache*, *mixed*, *two-level adaptive*, *hybrid*, etc. Recent research studies have also proposed new concepts to implement high-bandwidth instruction fetch engines based on *multiple branch prediction*. Such concepts include the *trace cache* [3,32] and the more conventional *multiple-block fetching* approach [37,41,46].

Given that a majority of static instructions exhibit very little variations in values that they produce/consume during the course of a program's execution [21,23,25], data dependences can be eliminated at run-time by predicting the outcome values of instructions (*value prediction*) and by executing the true data dependent instructions. In general, the outcome value of an instruction can be assigned to registers, memory locations, condition codes, etc. Several architectures have been proposed for value prediction [10,24,25,33,34,36,42] – *last value predictor*, *stride predictor*, *context predictors* and *hybrid approaches* in order 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. Based on instruction type, value prediction is sometimes identified as prediction of the outcome of arithmetic instructions only, and the prediction of the outcome of memory access instructions as a different class, referred to as *memory prediction* [29].

2. Motivation and related work

Vast majority of the studies on prediction techniques quantify the ability of predictions to boost the ILP from an experimental point of view – either by *measurements* in a real system, or by *trace-driven* or *execution-driven simulation* [1,4,8,9,17]. As processor complexity continues to increase at a rapid rate and micro-architectures continue to become more speculative, it is not clear whether the performance simulators can continue to effectively predict actual machine performance due to several crucial issues: *simulator retargetability*, *simulator validation*, *simulation speed* and *simulation accuracy* [2]. Moreover, there are only a few *analytical models* [12,27,35] that provide some insight by isolating important parameters, but still too simple to capture the behavior of a real system. All these models are *deterministic* – they deal with average parameter values or a parameter takes only one value. There is no randomness and the result is based on known functions of inputs with no dependence on chance.

Opposed to this common trait, a stochastic model of the dynamic behavior of an ILP processor that aims to minimize the impact of control and true-data dependences and employs speculative execution based on branch and value prediction, has been introduced for the first time in [30]. In view of the fact that in a machine with multiple execution units capable to execute large number of instructions in parallel the service and storage requirements of each instruction are small compared to the total volume of the instruction stream, individual instructions are regarded as *atoms of a fluid* and large buffer levels are approximated by continuous fluid levels. As a result, state-space complexity is decreased. The dynamic behavior model is built using Fluid Stochastic Petri Nets (FSPN) [15,16,20,40,43] and the stochastic process underlying the Petri Net is described by a system of first-order hyperbolic partial differential equations with appropriately chosen initial and boundary conditions.

In this paper, we illustrate the use of the recently introduced FSPN model in deriving measures of interest and present performance evaluation results obtained using discrete-event simulation [7,14]. Since the modeling framework is essentially implementation-independent, the performance potential of branch and value prediction under different scenarios, as well as the operational environment influence on the performance of ILP processors with much more aggressive instruction issue, can be efficiently evaluated.

3. The FSPN model

The simplest description of a processor pipeline is that instructions flow and pass through separate pipeline stages connected by buffers. Control dependences stall the inflow of useful instructions (fluid) into the pipeline, whereas true data dependences decrease the aperture of the pipeline and the outflow rate. The buffer levels always vary and affect both the inflow and outflow rates. Branch prediction techniques tend to eliminate stalls in the inflow, while value prediction techniques help keeping outflow rate as high as possible.

The distribution of the time between two consecutive occurrences of branch instructions in the fluid stream is considered exponential with rate λ, which depends on the instruction fetch bandwidth, as well as the program's average *basic block size*. Branches vary widely in their dynamic behavior, and predictors that work well on one type of branches may not work as well on others. There are *easy-to-predict* and *hard-to-predict branches*, and the expected branch prediction accuracy is higher for the first and lower for the second. The probabilities to classify a branch as either easy- or hard-to-predict depend on program's characteristics.

Another important factor is the number of instructions that consume results of simultaneously initiated producer instructions in each cycle. The distribution of the time between two consecutive occurrences of consuming instructions in the fluid stream is considered exponential with rate μ, which depends on the number of instructions that simultaneously initiate execution at a functional unit, as well as the program's average *dynamic instruction distance*. There are, also, two classes of consuming instructions: (i) instructions that consume *easy-to-predict values* and (ii) instructions that consume *hard-to-predict values*. The expected value prediction accuracy is higher for the first and lower for the second, while the probability to classify a value as either easy- or hard-to-predict depends on the program's characteristics, similarly to the branch classification.

The set of programs executed on a machine represent the *input space*. Programs with different characteristics are executed randomly and independently according to the *operational profile*. The input space is partitioned by grouping programs that exhibit as nearly as possible homogenous behavior into *program classes* [28]. Since the number of partitions is finite, the upper limits of λ and μ, as well as the probabilities to classify a branch/value as either easy- or hard-to-predict are considered to be discrete random variables and have different values for different program classes.

The pipeline is organized in four stages: Fetch, Decode/Issue, Execute and Commit. Fluid places P_{IC}, P_{IB}, $P_{RS/LSQ}$, P_{ROB}, P_{RR}, P_{EX} and P_{REG}, depicted by means of two concentric circles (Fig. 1), represent buffers between pipeline stages: *instruction cache*, *instruction buffer*, *reservation stations and load/store queue*, *reorder buffer, rename registers, instructions that have completed execution* and *architectural registers*. Five of them have limited capacities: Z_{IBmax}, $Z_{RS/LSQmax}$, Z_{RRmax}, Z_{ROBmax} and Z_{EXmax}. The fluid place P_{TIME} has the function of an hourglass: it is constantly filled at rate 1 up to the level 1 and then flushed out, which corresponds to the machine clock cycle. $Z_{TIME}(t)$ denotes the fluid level in P_{TIME} at time t. Fluid arcs are drawn as double arrows to suggest a pipe. Flow rates are piecewise constant, i.e. take different values at the beginning of each cycle and are limited by the fetch/issue width of the machine (W). Rates depend on the vector of fluid levels $\mathbf{Z}(t)$ and change when T_{CLOCK} fires and the fluid in P_{TIME} is flushed out. The flush out arc is drawn as thick single arrow.

Let $Z_{IC_0}, Z_{IB_0}, Z_{RS/LSQ_0}, Z_{RR_0}, Z_{ROB_0}$ and Z_{EX_0} be the fluid levels at the beginning of the clock cycle, i.e. $Z_{IC_0} = Z_{IC}(t_0)$, $Z_{IB_0} = Z_{IB}(t_0)$, $Z_{RS/LSQ_0} = Z_{RS/LSQ}(t_0)$, $Z_{RR_0} = Z_{RR}(t_0)$, $Z_{ROB_0} = Z_{ROB}(t_0)$ and $Z_{EX_0} = Z_{EX}(t_0)$, where $t_0 = \lfloor t \rfloor$ and $Z_{TIME}(t_0) = 0$.

A high-bandwidth instruction fetch mechanism fetches up to W instructions per cycle and places them in the instruction buffer. The fetch rate is given by:

$$r_{FETCH} = \min(Z_{IB_{\max}} - Z_{IB_0} + r_{ISSUE}, Z_{IC_0}, W) \quad (1)$$

In the case of a branch misprediction, the fetch unit is effectively stalled and no useful instructions are added to the buffer. Instruction cache misses are ignored.

Instruction issue tries to send W instructions to the appropriate reservation stations or the load/store queue on every clock cycle. Rename registers are allocated to hold the results of the instructions and reorder buffer entries are allocated to ensure in-order completion. Among the instructions that initiate execution in the same cycle, speculatively executed consuming instructions are forced to retain their reservation stations. As a result, the issue rate is given by:

$$r_{ISSUE} = \min(Z_{RR\max} - Z_{RR0} + r_{COMMIT}, Z_{ROB\max} - Z_{ROB0} + r_{COMMIT}, Z_{RS/LSQ\max} - Z_{RS/LSQ0}, Z_{IB0}, W) \quad (2)$$

Shortages of read ports for the registers, conflicts that occur when special registers are accessed, and serialization imposed by special instructions is ignored.

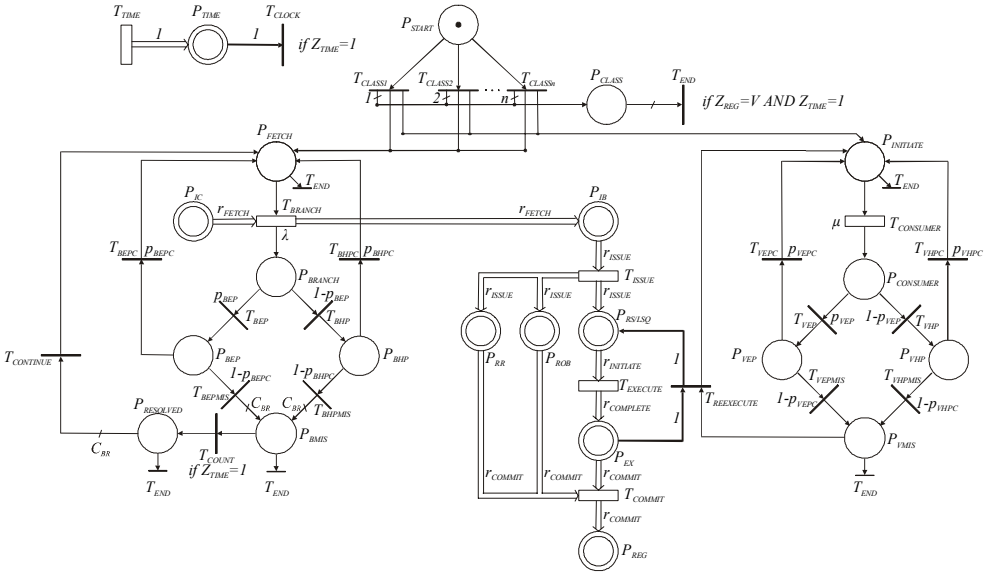


Figure 1: A Fluid Stochastic Petri Net model

Up to W instructions are *in execution* at the same time. With the assumptions that functional units are always available and out-of-order execution is allowed, the instructions *initiate* and *complete* execution with rate:

$$r_{INITIATE} = r_{COMPLETE} = \min(Z_{RS/LSQ_0}, W) \quad (3)$$

During the execute stage, the instructions first check to see if their source operands are available (predicted or computed). For simplicity, the execution latency

of each instruction is a single cycle. Instructions execute and forward their own results back to subsequent instructions that might be waiting for them (no result forwarding delay). Every reference to memory is present in the first-level cache. With the last assumption, the effect of the memory hierarchy is eliminated.

The instructions that have completed execution are ready to move to the last stage. Up to W instructions may commit per cycle. The results in the rename registers are written into the register file and the rename registers and reorder buffer entries freed. Since stalls in the instruction commit stage are very infrequent [19], shortages of write ports for the registers and head-of-line blockings are ignored. Hence:

$$r_{COMMIT} = \min(Z_{EX_0}, W) \tag{4}$$

In order to capture the relative occurrence frequencies of different program classes, a set of weighted immediate transitions is introduced in the Petri Net. Each program class is assigned an immediate transition T_{CLASS_i} with weight w_{CLASS_i}. The operational profile is a set of weights. The probability of firing the immediate transition T_{CLASS_i} represents the probability of occurrence of a class i program, given by:

$$\widehat{w}_{T_{CLASS_i}} = \frac{w_{T_{CLASS_i}}}{\sum_{k=1}^{n} w_{T_{CLASS_k}}} \tag{5}$$

A token in P_{START} denotes that a new execution is about to begin. The process of firing one of the immediate transitions randomly chooses a program from one of the classes. The firing of transition T_{CLASS_i} puts i tokens in place P_{CLASS}, which identify the class. At the same time instant, tokens occur in places P_{FETCH} and $P_{INITIATE}$, while the fluid place P_{IC} is filled with fluid with volume V_i equivalent to the total number of useful instructions (*program volume*).

Firing of exponential transition T_{BRANCH} corresponds to a branch instruction occurrence. The parameter λ changes at the beginning of each clock cycle and formally depends on both the number of tokens in P_{CLASS} and the fetch rate:

$$\lambda = \mathrm{f}(\#P_{CLASS})\frac{r_{FETCH}}{W} = \mathrm{f}(i)\frac{r_{FETCH}}{W} = \lambda_i \frac{r_{FETCH}}{W} \tag{6}$$

where λ_i is its upper limit for a given program class i at maximum fetch rate ($r_{FETCH}=W$). The branch is classified as easy-to-predict with probability $p_{BEP}=p_{BEPi}$, or hard-to-predict with probability $1-p_{BEPi}$. In either case, it is correctly predicted with probability p_{BEPC} (p_{BHPC}), or mispredicted with probability $1-p_{BEPC}$ ($1-p_{BHPC}$). These probabilities are included in the FSPN model as weights

assigned to immediate transitions T_{BEP}, T_{BHP}, T_{BEPC}, T_{BHPC}, T_{BEPMIS} and T_{BHPMIS}, respectively. This approach is known as *synthetic branch prediction.* Branch mispredictions stall the fluid inflow for as many cycles as necessary to resolve the branch (C_{BR} tokens in place P_{BMIS}). Usually, a branch is not resolved until its execution stage (C_{BR}=3). With several consecutive firings of $T_{CLOCK,}$ these tokens are consumed one at a time and moved to $P_{RESOLVED}$. As soon as the branch is resolved, transition $T_{CONTINUE}$ fires, a token appears in place P_{FETCH} and the inflow resumes.

Similar to this, firing of exponential transition $T_{CONSUMER}$ corresponds to the occurrence of a consuming instruction among the instructions that initiated execution. The parameter μ changes at the beginning of each clock cycle and formally depends on both the number of tokens in P_{CLASS} and the initiation rate:

$$\mu = \mathrm{g}(\#P_{CLASS})\frac{r_{INITIATE}}{W} = \mathrm{g}(i)\frac{r_{INITIATE}}{W} = \mu_i \frac{r_{INITIATE}}{W} \tag{7}$$

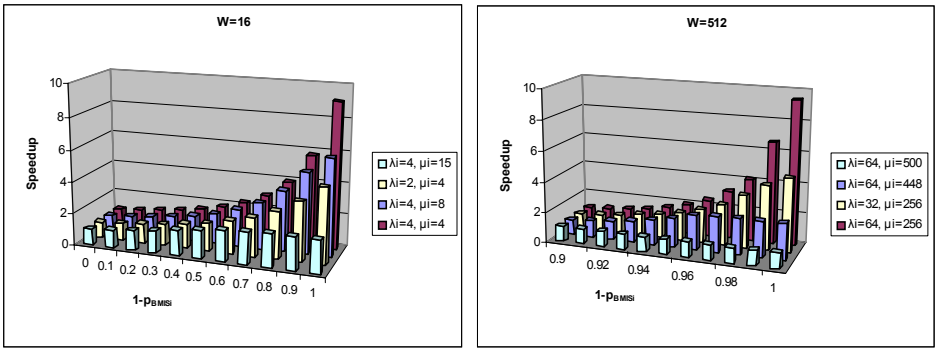
where μ_i is its upper limit for a given program class i when maximum possible number of instructions simultaneously initiate execution ($r_{INITIATE}=W$). The consumed value is classified as easy-to-predict with probability $p_{VEP}=p_{VEPi}$, or hard-to-predict with probability $1\text{-}p_{VEPi.}$ In either case, it is correctly predicted with probability p_{VEPC} (p_{VHPC}), or mispredicted with probability $1\text{-}p_{VEPC}$ ($1\text{-}p_{VHPC}$). These probabilities are included in the FSPN model as weights assigned to immediate transitions T_{VEP}, T_{VHP}, T_{VEPC}, T_{VHPC}, T_{VEPMIS} and T_{VHPMIS}, respectively. Whenever a misprediction occurs (token in place P_{VMIS}), the consuming instruction has to be rescheduled for execution. The firing of immediate transition $T_{RE\text{-}EXECUTE}$ causes transportation of fluid in zero time. Fluid jumps have deterministic height of 1 (one instruction) and take place when the fluid levels in P_{RS} and P_{EX} satisfy the condition $Z_{RS}(t) \le Z_{RS_{\max}} - 1$ and $Z_{EX}(t) \ge 1$. Jumps that would go beyond the boundaries cannot be carried out. The arcs connecting fluid places and immediate transitions are drawn as thick single arrows. The fluid flow terminates at the end of the cycle when all the fluid places except P_{REG} are empty and T_{END} fires.

4. Performance evaluation results

The results have been obtained using discrete-event simulation, specifically implemented for this model and not for a general FSPN. The types of events that need to be scheduled in the event queue are either transition firings or the hitting of a threshold dependent on fluid levels. We have used a Unif[0,1] pseudo-random number generator to generate samples from the respective cumulative distribution functions and determine transition firing times via inversion of the *cdf* ("Golden Rule for Sampling").

Initially we analyze the efficiency of branch prediction by varying branch prediction accuracy. Value prediction is not involved at all. The speedup is computed by dividing the IPC achieved with certain branch prediction accuracy over the IPC achieved without branch prediction ($1-p_{BMIS_i}=0$). For the moment, the input space is not partitioned and program volume is set to V_i=10^6.

Looking at Fig. 2, it is observed that branch prediction curves have an exponential shape. This is in agreement with the results obtained using trace-driven simulation in [13], and the results from the analytical perfect value prediction model reported in [35]. Building branch predictors that improve the accuracy just a little bit may be reflected in a significant performance increase. The impact of a given increment in accuracy is more noticeable when it experiences a slight improvement beyond the 90%. Another conclusion drawn from these figures is that one can benefit most from branch prediction in programs with relatively short basic blocks (high λ_i/W) and which do not suffer excessively from true data dependences (low μ_i/W). When the ratio μ_i/W is high, true data dependences overshadow control dependences. As a result, the amount of ILP that is expected without value prediction in a machine with extremely aggressive instruction issue is far below the maximum possible value, even with perfect branch prediction. Value prediction has to be involved to go beyond the limits imposed by true data dependences.



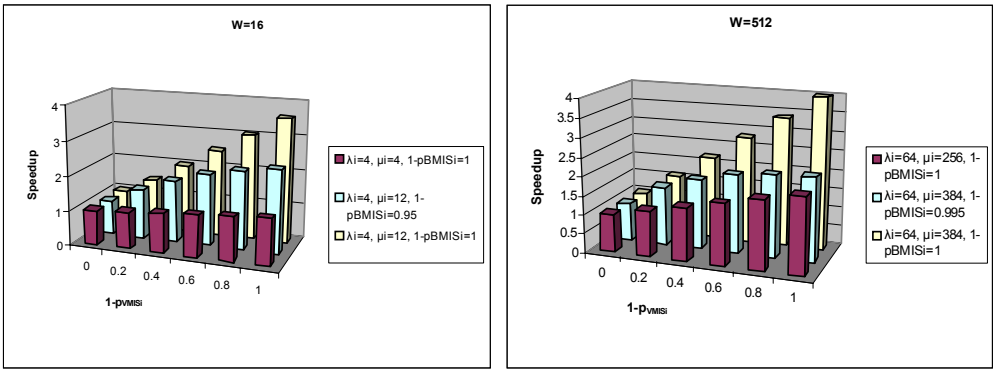
(a) (b)

Figure 2: Speedup achieved by branch prediction with varying accuracy

Next, we analyze the efficiency of value prediction by varying value prediction accuracy (Fig. 3). The speedup is computed by dividing the IPC achieved with certain value prediction accuracy over the IPC achieved without value prediction ($1-p_{VMIS_i}=0$). With perfect branch prediction, it seems clear that the value prediction curves have a linear behavior. The same trends have been revealed in [13]

using trace-driven simulation. Therefore, it is worthwhile to build a predictor that significantly improves the accuracy. Only a small improvement on the value predictor accuracy has a little impact on ILP processor performance, regardless of the accuracy range. Another conclusion drawn from these figures is that the effect of value prediction is more noticeable when a significant number of instructions consume results of simultaneously initiated producer-instructions during execution (high μ_i / W), i.e. when true data dependences have a much higher influence on the program's total execution time.

Branch prediction has a very important influence on the benefits of value prediction. One can see that the performance increase is less significant when branch prediction is realistic. Because mispredicted branches limit the number of useful instructions that enter the instruction window, the processor is able to provide almost the same number of instructions to leave the instruction window, even with lower value prediction accuracy. As a result, graphs tend to flatten out. Correct value predictions can only be exploited when the fetch rate is quite high, i.e. when mispredicted branches are infrequent. Branch misprediction becomes a more significant performance limitation with wider processors (Fig. 3b).

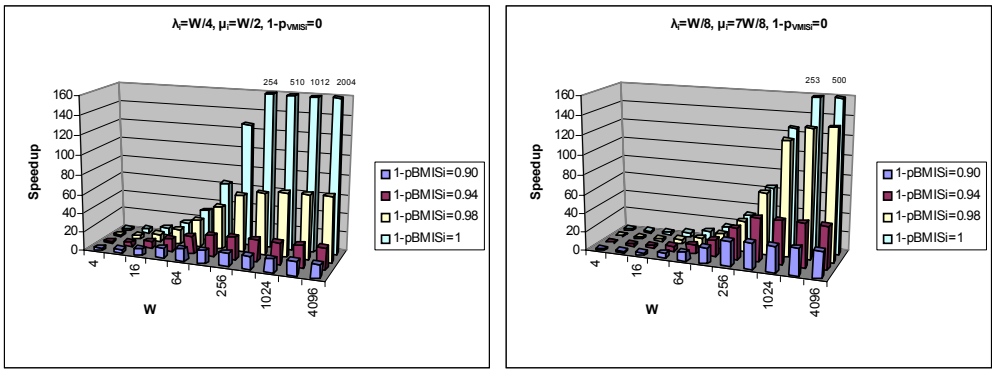


(a) (b)

Figure 3: Speedup in a machine with perfect branch prediction achieved by value prediction with varying accuracy

In addition, we investigate branch and value prediction efficiency with varying machine width (Fig. 4). The speedup in this case is computed by dividing the IPC achieved in a machine over the IPC achieved in a scalar counterpart (W=1, μ_i=0). The speedup due to branch prediction is obviously higher in wider machines. With perfect branch prediction, the speedup unconditionally increases with the machine width. For a given width, the speedup is higher when there are a smaller number of consuming instructions (low μ_i / W). With realistic branch

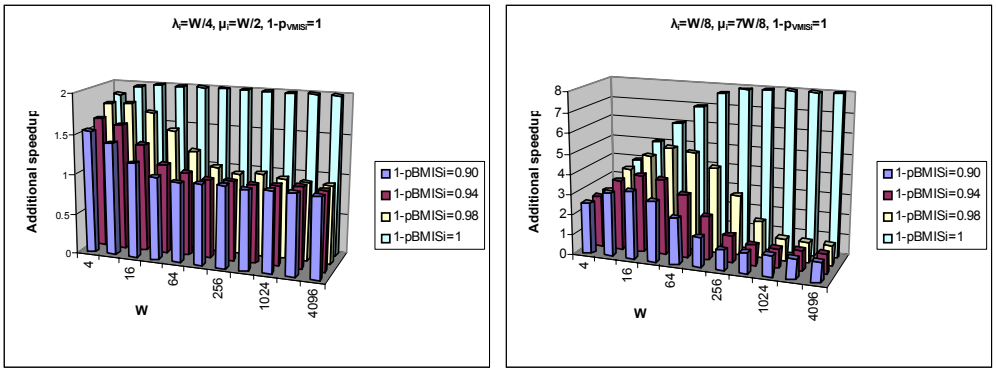
prediction, there is a threshold effect on the machine width: below the threshold the speedup increases with the machine width, whereas above the threshold the speedup is close to a limit – machine width is by far larger than the average number of instructions provided by the fetch unit. The threshold decreases with increasing the number of mispredicted branches. This is in agreement with the results reported in [27] where a threshold effect on the instruction fetch rate was exposed.



(a) (b)

Figure 4: Speedup achieved by branch prediction with varying machine width

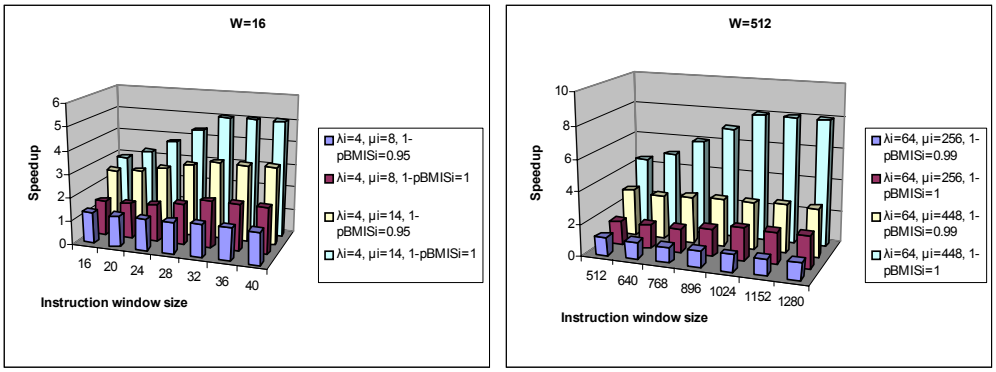
The maximum additional speedup that value prediction can provide is computed by dividing the IPC achieved with perfect value prediction over the IPC achieved without value prediction (Fig. 5). With perfect branch prediction, some true data dependences can always be eliminated, regardless of the machine width. Actually, the maximum additional speedup is predetermined by the ratio $W/(W-\mu_i)$. However, with realistic branch prediction, the additional speedup diminishes when the machine width is above a threshold value. It happens earlier when there are a smaller number of consuming instructions and/or a larger number of mispredicted branches. In either case, the number of independent instructions examined for simultaneous execution is sufficiently higher than the number of fetched instructions that enter the instruction window. Once more, branch prediction becomes more important with wider processors.



(a) (b)

Figure 5: Additional speedup achieved by perfect value prediction with varying machine width

The rate at which consuming instructions occur depends on the initiation rate. Therefore, we also investigate the value prediction efficiency with varying instruction window size (varying capacity $Z_{RS/LSQ_{MAX}}$ of the fluid place $P_{RS/LSQ}$) (Fig. 6). The speedup is computed in the same way as in the previous instance. It increases with the instruction window size in $[W, 2W]$, but the increase is more moderate when there are a smaller number of consuming instructions (low μ_i / W) and/or branch prediction is not perfect. As the instruction window grows larger, performance without value prediction saturates (also reported in [31] using trace-driven simulation), as does the performance with perfect value prediction. The upper limit value emerges from the fact that in each cycle up to W new instructions may enter the fluid place $Z_{RS/LSQ_{MAX}}$ and up to W consuming instructions may be forced to retain their reservation stations. Similar trends have been revealed in [35] using execution-driven simulation for issue widths of 4, 8 and 16 instructions. One should also note that the speedup for $W>>1$ and realistic branch prediction is almost constant with increasing instruction window size. Two scenarios arise in this case: (i) the number of consuming instructions is large – the speedup is constant but still noticeable as there are not enough independent instructions in the window without value prediction, and (ii) the number of consuming instructions is small – there is no speedup as there are enough independent instructions in the window even without value prediction, regardless of the window size. Again, the main reasons for this behavior are the small number of consuming instructions and the large number of mispredicted branches.



(a) (b)

Figure 6: Speedup achieved by perfect value prediction with varying instruction window size

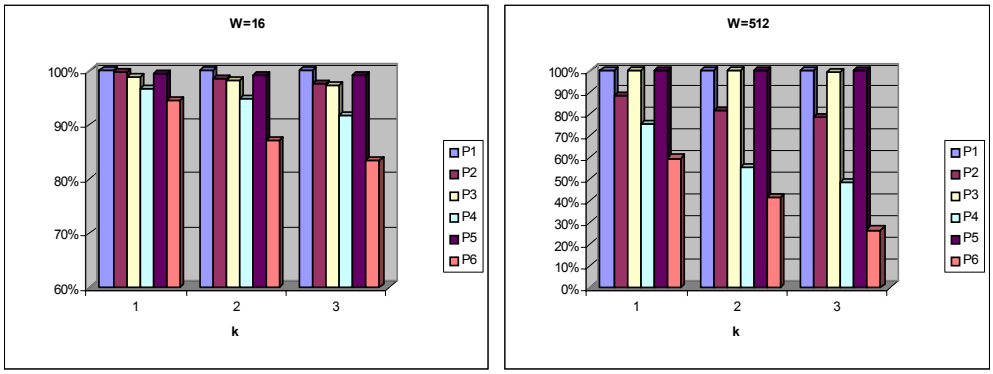
In order to investigate the operational environment influence, we partition the input space into several program classes (Table 1), each of them with at least one different aspect: branch rate, consuming instruction rate, probability to classify a branch as easy-to-predict or probability to classify a value as easy-to-predict. Branches and values are predicted with probabilities p_{BEPC} =0.98, p_{BHPC} =0.75, p_{VEPC} =0.75 and p_{VHPC} =0.5. Programs with different characteristics are executed randomly and independently, according to the operational profile $\mathbf{P}_n$ $(1 \le n \le 6)$. The i-th element of the vector $\mathbf{P}_n$ denotes the average number of occurrences of a class i program in ten consecutive executions.

$$\mathbf{P}_1 = [10 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$\mathbf{P}_2 = [10-k \quad k \quad 0 \quad 0 \quad 0 \quad 0]$$
$$\mathbf{P}_3 = [10-k \quad 0 \quad k \quad 0 \quad 0 \quad 0]$$
$$\mathbf{P}_4 = [10-k \quad 0 \quad 0 \quad k \quad 0 \quad 0]$$
$$\mathbf{P}_5 = [10-k \quad 0 \quad 0 \quad 0 \quad k \quad 0]$$
$$\mathbf{P}_6 = [10-k \quad 0 \quad 0 \quad 0 \quad 0 \quad k]$$

i	1	2	3	4	5	6
V_i	10^6	10^6	10^6	10^6	10^6	10^6
λ_i	W/8	W/4	W/8	W/8	W/8	W/4
μ_i	W/2	W/2	7W/8	W/2	W/2	7W/8
p_{BEPi}	1	1	1	0.67	1	0.67
p_{VEPi}	1	1	1	1	0.67	0.67

Table 1: Program classes

In general, the overall performance decrease is more pronounced when programs with poor characteristics are more frequent (k=3) (Fig. 7). Individually, the largest performance decrease emerges from the execution of programs with larger number of branches, especially hard-to-predict ones. The sporadic execution of programs with larger number of consuming instructions, or larger number of hard-to-predict values, has a lower influence on performance. Such a decrease almost diminishes with increasing machine width, given that the branch prediction is not perfect. When some programs include all the aforementioned poor characteristics (profile $\mathbf{P}_6$), the performance decrease is more than obvious. Even if such programs are executed ten times less frequently (k=1) than "class one" programs, their excessively long execution time has an adverse impact on the overall performance. We conclude that the set of programs executed on a machine have a considerable influence on the *perceived IPC*. Since the term *program* may be interchangeably used with the term *instruction stream*, these observations give good reason for the analysis of the time varying behavior of programs in order to find simulation points in applications to achieve results representative of the program as a whole [38]. From a user perspective, a machine with more sophisticated prediction mechanisms will not always lead to a higher *perceived performance* as compared to a machine with more modest prediction mechanisms but more favorable operational profile.



(a) (b)

Figure 7: Relative performance decrease due to sporadic execution of programs with poor characteristics

5. Conclusion

Effective control and data speculation techniques are essential to explore the full performance of modern ILP processors as they move towards wider issue and deeper pipelines. Using discrete-event simulation of Fluid Stochastic Petri Nets, we have evaluated the performance impact of branch and value prediction by varying several parameters of both the micro-architecture and the operational environment – branch and value prediction accuracy, machine width, instruction window size and operational profile. The main conclusions that can be drawn from this study are the following:

- The benefits of branch and value prediction are higher when control and true data dependences have a much higher influence on the total execution time of a program;
- Value prediction is an effective approach that might enable higher levels of parallelism without increasing machine width and/or instruction window size;
- There is a correlation between the value prediction efficiency and the branch prediction efficiency;
- Branch misprediction becomes a significant performance limitation with wider processors;
- The characteristics of the operational environment have strong influence on ILP processor's performance.

We believe that our implementation-independent stochastic modeling framework reveals considerable potential for further research in this area, needed to better understand speculation techniques and their performance potential under different scenarios.

6. References

1. Bechem, C., Combs, J., Utamaphethai, N., Black, B., Blanton, S., Shen, J.P., "An Integrated Functional Performance Simulator", IEEE Micro, pp. 26-34, May-June 1999
2. Black, B., Huang, A., Lipasti, M., Shen, J.P., "Can Trace-Driven Simulators Accurately Predict Superscalar Performance?", Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors, San Antonio, USA, 1996
3. Black, B., Rychlik, B., Shen, J.P., "The Block-based Trace Cache", Proceedings of the 26th International Symposium on Computer Architecture, pp. 196-207, Atlanta, USA, 1999
4. Burger, D., Austin, T.M., "The SimpleScalar Tool Set, Version 2.0", University of Wisconsin – Madison, Computer Sciences Department, Technical Report #1342, 1997
5. Chang, P.Y., Hao, E., Patt, Y., "Alternative Implementations of Hybrid Branch Predictors", Proc. of the 28th Annual International Symposium on Microarchitecture, pp. 252-263, Ann Arbor, USA, 1995
6. Chang, P.Y., Hao, E., Yeh, T.Y., Patt, Y., "Branch Classificaion: a New Mechanism for Improving Branch Predictor Performance", Proc. of the 27th Annual International Symposium on Microarchitecture, pp. 22-31, San Jose, USA, 1994
7. Ciardo, G., Nicol, D., Trivedi, K., "Discrete-Event Simulation of Fluid Stochastic Petri Nets", Proc. of the 7th international Workshop on Petri Nets and performance Models (PNPM'97), pp. 217-225, Saint Malo, France, 1997
8. Diep, T.A., Shen, J.P., Phillip, M., "EXPLORER: A Retargetable and Visualisation-Based Trace-Driven Simulator for Superscalar Processors", Proc. of the 26th International Symposium on Microarchitecture, pp. 225-235, Austin, USA, 1993
9. Diep, T.A., Shen, J.P., "VMW: Visualisation Based Microarchitecture Workbench", IEEE Computer, pp. 57-64, December 1995
10. Gabbay, F., "Speculative Execution based on Value Prediction", EE Department Technical Report #1080, Technion – Israel Institute of Technology, Haifa, Israel, 1996

11. Gabbay, F., Mendelson, A., "The Effect of Instruction Fetch Bandwidth on Value Prediction", Proceedings of the 25th International Symposium on Computer Architecture, pp. 272-281, Barcelona, Spain, 1998
12. Gabbay, F., Mendelson, A., "Using Value Prediction to Increase the Power of Speculative Execution Hardware", ACM Transactions on Computer Systems, Vol.16, No.3, pp. 234-270, 1998
13. Gonzalez,J., Gonzalez,A., "The Potential of Data Value Speculation to Boost ILP", Proc. of the 12th ACM International Conference on Supercomputing, Melbourne, Australia, 1998
14. Gribaudo, M., Sereno, M., "Simulation of Fluid Stochastic Petri Nets", Proc. of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 231-239, San Francisco, USA, 2000
15. Gribaudo, M., Sereno, M., Bobbio, A., "Fluid Stochastic Petri Nets: An extended Formalism to Include non-Markovian Models", Proc. of the 8th International Workshop on Petri Nets and Performance Models, Zaragoza, Spain, 1999
16. Gribaudo, M., Sereno, M., Horvath, A., Bobbio, A., "Fluid Stochastic Petri Nets Augmented with Flush-out Arcs: Modeling and Analysis", Discrete Event Dynamic Systems, 11(1/2), pp.97-117, Kluwer Academic Publishers, 2001
17. Gušev, M., Popovski, G., Mišev, A., "Simulation of Superscalar Processor", Proc. of the 20th International Conference on Information Technology Interfaces, Pula, Croatia, 1998
18. Haungs, M., Salee, P., Farrens, M., "Branch Transition Rate: A New Metric for Improved Branch Classification Analysis", Proceedings of the 6th International Symposium on High-Performance Computer Architecture (HPCA 6), pp. 241-250, Toulouse, France, 2000
19. Hennessy, J.L., Patterson, D.A., "Computer Architecture: A Quantitative Approach", Second Edition, Morgan Kaufmann Publishers, San Francisco, USA, 1996
20. Horton, G., Kulkarni, V., Nicol, D., Trivedi, K., "Fluid Stochastic Petri Nets: Theory, Applications, and Solution", European Journal of Operations Research, 105(1), pp. 184-201, 1998
21. Huang, J., Choi, Y., Lilja, D., "Improving Value Prediction by Exploiting Both Operand and Output Value Locality", Technical Report ARCTiC-99-06, University of Minnesota, Minneapolis, USA, 1999
22. Lee, J.K.F., Smith, A.J., "Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer, pp. 6-22, January 1984
23. Lipasti, M., Shen, J.P., "The Performance Potential of Value and Dependence Prediction", Proceedings of EUROPAR-97, Passau, Germany, 1997

24. Lipasti,M., Shen,J.P., "Exceeding the dataflow limit via value prediction", Proc. of the 29th Annual International Symposium on Microarchitecture, pp. 226-237, Paris, France, 1996
25. Lipasti,M., Wilkerson,C., Shen,J.P., "Value Locality and Load Value Prediction", Proc. of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 138-147, Cambridge, USA, 1996
26. McFarling, S., "Combining Branch Predictors", Technical Report TN-36, Digital Equipment Corporation, Western Research Lab, 1993
27. Michaud, P., Seznec, A., Jourdan, S., "Exploring Instruction-Fetch Bandwidth Requirement in Wide-Issue Superscalar Processors", Proc. of the International Conference on Parallel Architectures and Compilation Techniques, Newport Beach, USA, 1999
28. Mitrevski, P., Goševa-Popstojanova, K., Grnarov, A., "Reliability and Performability Modeling of N-Version Fault-Tolerant Software in Real-Time Environment using Markov Regenerative Stochastic Petri Nets", Proc. of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2000) , Vol. VII, pp. 463-468, Orlando, USA, 2000
29. Mitrevski, P., Gušev, M., Mišev, A., "Prediction and Speculation Techniques in ILP", Proc. of the 22nd International Conference on Information Technology Interfaces (ITI 2000), pp. 67-72, Pula, Croatia, 2000
30. Mitrevski, P., Gušev, M., "Modeling the Dynamic Behavior of an ILP Processor", Proc. of the 23rd International Conference on Information Technology Interfaces (ITI 2001), pp. 69-74, Pula, Croatia, 2001
31. Neefs, H., De Bosschere, K., Van Kampenhout, J., "Exploitable levels of ILP in future processors", Journal of Systems Architecture, 45, pp. 687-708, Elsevier, 1999
32. Rotenberg, E., Bennett, S., Smith, J., "Trace Cache: a Low Latency Approach to High Bandwidt Instruction Fetching", Proc. of the 29th Annual International Symposium on Microarchitecture, pp. 24-35, Paris, France, 1996
33. Rychlik, B., Faistl, J., Krug, B., Kurland, A., Sung, J., Velev, M., Shen, J.P., "Efficient and Accurate Value Prediciton Using Dynamic Classification", Technical Report CMμART-1998-01, Carnegie Mellon University, Pittsburgh, USA, 1998
34. Rychlik, B., Faistl, J., Krug, B., Shen, J.P., "Efficacy and Performance Impact of Value Prediciton", Proc. of the International Confrernce on Parallel Architectures and Compilation Techniques, Paris, France, 1998
35. Sazeides, Y., "An Analysis of Value Predictability and its Application to a Superscalar Processor", PhD Thesis, University of Wisconsin-Madison, 1999

36. Sazeides, Y., Smith, J.E., "Implementations of Context Based Value Predictors", Technical Report ECE-97-8, University of Wisconsin – Madison, USA, 1997
37. Seznec, A., Jourdan, S., Sainrat, P., Michaud, P., "Multiple-Block Ahead Branch Predictors" Proc. of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 116-127, Cambridge, USA, 1996
38. Sherwood, T., Perelman, E., Calder, B., "Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications", Proc. of the International Conference on Parallel Architectures and Compilation Techniques, Barcelona, Spain, 2001
39. Smith, J., "A Study of Branch Prediciton Strategies", Proc. of the 8th Annual International Symposium on Computer Architecture, pp. 135-148, 1981
40. Trivedi, K., Kulkarni, V., "FSPNs: Fluid Stochastic Petri Nets", Lecture Notes in Computer Science, Vol. 691, M.Ajmone Marsan (ed.), Proc. of the 14th International Conference on Applications and Theory of Petri Nets, pp. 24-31, Heidelberg, Germany, 1993
41. Wallace, S., Bagherzadeh, N., "Multiple Branch and Block Prediction", Proc. of the 3rd International Symposium on High Performance Computer Architecture, San Antonio, USA, 1997
42. Wang, K., Franklin, M., "Highly Accurate Data Value Prediction using Hybrid Predictors", Proc. of the 30th Annual International Symposium on Microarchitecture, pp. 281-290, Research Triangle Pk, USA, 1997
43. Wolter, K., Horton, G., German, R., "Non-Markovian Fluid Stochastic Petri Nets", Technical Report 1996-13, TU Berlin, Germany, 1996
44. Yeh, T.Y, Patt, Y.N., "Two-Level Adaptive Branch Prediction", Proc. of the 24th Annual International Symposium on Microarchitecture, pp. 51-61, Albuquerque, USA, 1991
45. Yeh, T.Y, Patt, Y.N., "Alternative Implementations of Two-Level Adaptive Branch Prediction", Proc. of the 19th Annual International Symposium on Computer Architecture, pp. 124-134, Gold Coast, Australia, 1992
46. Yeh, T.Y., Marr, D., Patt, Y., "Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache", Proc. of the International Conference on Supercomputing, pp. 67-76, Tokyo, Japan, 1993