On the Complexity of Modeling Dynamic Branch Predictors when Computing Worst-Case Execution Times

Claire Burguière and Christine Rochange
IRIT - CNRS
Université de Toulouse
France
Email: {burguier,rochange}@irit.fr

Abstract

Modern processors implement hardware branch predictors to increase their peak performance. However, the behavior of dynamic branch prediction schemes is strongly dependent on the runtime execution paths which makes them very difficult to analyze with usual static analysis techniques. In many processors, dynamic branch prediction can be disabled to favor timing predictability. However this will limit the processor performance which might not be desirable if high execution rates are needed. These last six years, several approaches have been proposed to model dynamic branch predictors within Worst-Case Execution Time (WCET) analysis. In this paper, we focus on solutions that integrate the branch predictor analysis into the WCET computation with the IPET method [8]. We analyze the modeling complexity *of dynamic branch prediction schemes, i.e. the difficulty of describing a branch predictor for WCET estimation. Our purpose is to give an insight into the impact of the branch predictor features on the effort required to build a model for WCET analysis.*

1. Introduction

The design of applications that have to meet strict deadlines makes it essential to estimate their Worst-Case Execution Time (WCET). Academic research promotes techniques based on static code analysis that limit measurements to small parts of code (e.g. basic blocks) against dynamic approaches that measure the execution time of complete paths and may require unacceptable measurement time.

When estimating the Worst-Case Execution Time of a task to be run on a processor that features dynamic branch prediction, the impact of branch mispredictions must be taken into account. This requires analyzing the worst-case misprediction count of each conditional branch instruction.

A trivial and conservative approach would consider that branches are always mispredicted: this would generally lead to a severe overestimation of the WCET which is undesirable in most cases. This is why several schemes have been proposed these last years to analyze dynamic branch prediction more tightly. There are two ways to tackle the problem. *Local approaches* examine each branch individually to determine its worst-case number of mispredictions [2–4, 6]. The estimation is based on the knowledge of the algorithmic structures related to the branches. On the contrary, *global analyses* ignore semantic information and include a description of the prediction algorithm within the model used to derive the WCET. The contribution of this paper concerns global approaches.

In this paper, we extend a model previously proposed by Li *et al.* [7] to describe predictors implemented in off-the-shelves embedded cores. We also introduce a means of comparing the effort needed to build the system of constraints that model the dynamic branch predictor, to read and understand it, and to extend it to new prediction schemes. This is what we call the *modeling complexity* which is *not* the same as the computational complexity that refers to the time required to solve the model. The modeling complexity expresses instead the effort required to *develop* the ILP formulation used to compute the WCET.

The paper is organized as follows. Section 2 gives an overview of branch prediction and Section 3 details the model proposed by Li *et al.* In Section 4, we extend this model to more realistic branch predictors. Section 5 introduces the notion of *modeling complexity* and analyzes the modeling complexity of several branch prediction schemes. In Section 6, we discuss quantitative results. Concluding remarks are given in Section 7.

2. An overview of dynamic branch prediction

2.1 Branch prediction schemes

Branch prediction enhances the pipeline performance by allowing the speculative fetching of instructions along the predicted path after a conditional branch has been encountered and until it is resolved. If the branch was mis-

predicted, the pipeline is flushed and the correct path is fetched and executed.

Dynamic branch prediction analyzes the history of the control flow to anticipate the future branch issues. The history of a branch is generally stored as a 2-bit saturating counter, updated every time the branch is executed, as shown in Figure 1. The outcome of a branch is predicted according to its counter value: *taken* whenever the upper bit is 1, and *not taken* otherwise [9]. A 1-bit counter can also be used to make the prediction. In this case, the last executed direction constitutes the prediction. The counters are maintained in the Branch History Table (BHT)

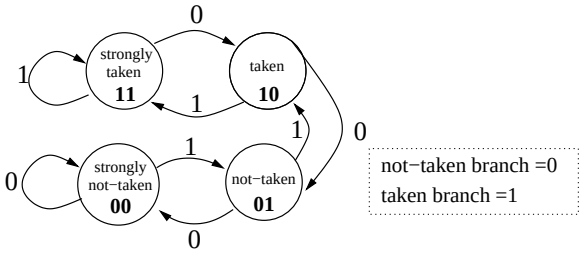

Figure 1. 2-bit prediction counter

Whenever a branch is predicted as *taken*, its target address also has to be anticipated. Previously computed target addresses are stored in the Branch Target Buffer (BTB). The BTB is generally indexed by the branch address (Program Counter, or PC). The BHT can also be indexed by the PC, as in the PC-based bimodal scheme [9], or by an History Register that stores the outcomes of the n last branches [10]. This way, a given branch can be predicted from different counters according to the recent history (see Figure 2).

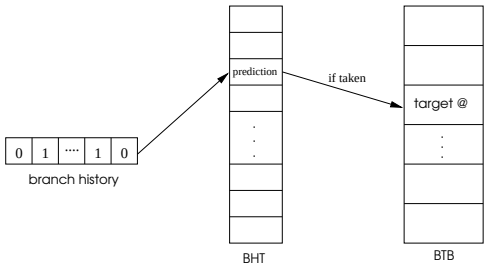

Figure 2. A history-indexed dynamic branch predictor

However the branch prediction tables are indexed, different branches might compete for the same entry. The BTB is tagged (with the branch PC), which allows detecting possible conflicts. But, to save transistors, the BHT is generally not tagged, and several branches can share the same entry. This is particularly true when the BHT is indexed by the History Register. This phenomenon is referred to as *aliasing*. Aliasing can be limited by using complex indexing schemes (e.g. XOR-ing some bits of the branch PC with the global history) but it cannot be completely avoided. In some cases, aliasing can be constructive (i.e. a branch can take advantage from the fact that its 2-bit counter has been updated by another branch) but it is more often destructive.

Table 1. Dynamic branch predictor in embedded cores

	ARM Cortex-A8	PowerPC 750 MIPS 34K	ARM11
History length	10 bits	-	-
BHT index	PC and history	PC	PC
BHT size	4096	512	128

2.2 Dynamic branch prediction in embedded processors

Table 1 shows the characteristics of dynamic branch predictors implemented in some modern embedded processors. The ARM Cortex-A8 [1] features a 4096-entry BHT (2-bit saturating counters) indexed by a 10-bit global branch history concatenated with 4 bits of the program counter. The PowerPC 750 and MIPS 34K cores both include a PC-based branch predictor where the BHT is directly indexed by the program counter. The ARM 11 core includes a unified table to store the branch target addresses and the 2-bit prediction counters, and this table is tagged. When a fetched branch is not present in the table, it is predicted using a static scheme.

3. Background: modeling dynamic branch prediction for WCET analysis

3.1. WCET analysis with the IPET method

The IPET method [8] to compute WCETs (and then to determine the longest execution path) considers the Control Flow Graph of the task under analysis. A set of integer variables stands for the execution counts of the nodes (basic blocks) and edges of the CFG. Estimating the WCET consists in maximizing an expression of the task execution time under a set of linear constraints. The execution time of the task is defined as the sum of the execution counts weighted by the corresponding partial execution times. Structural constraints express the control flow structure of the task, i.e. the relations that must exist between the execution counts. Flow constraints express loop bounds and infeasible paths. The WCET can be obtained through Integer Linear Programming algorithms which are implemented in tools like `lp_solve` [1]. The solution is returned as the set of values of the execution counts that maximize the task execution time. This is illustrated in Figure 3.

[1] http://lpsolve.sourceforge.net/5.5/

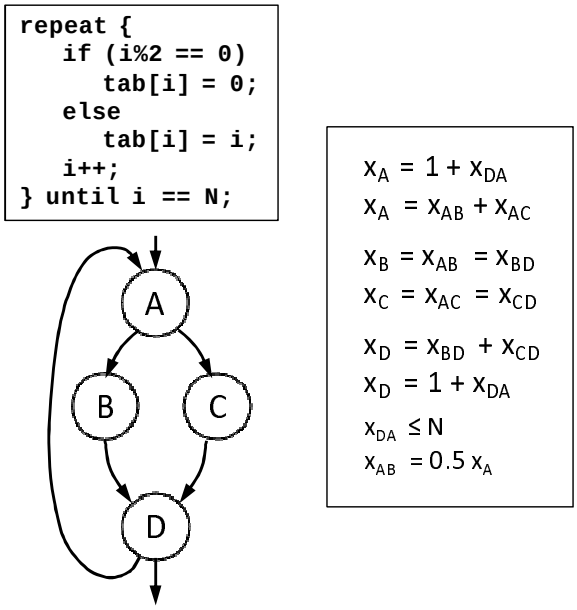

Figure 3. IPET model to compute the WCET

3.2. Integrated analysis of dynamic branch prediction

In this paper, we consider branch prediction modeling techniques that do analyze each branch as part of the determination of the longest (worst-case) execution path. This means that the target is to find out the misprediction count of a branch on the worst-case path, which is possibly different from computing the maximum number of mispredictions for that branch. This can be illustrated by considering a repeated conditional branch that implements an *if ... then ... else ...* statement within a loop. If the branch condition depends on the input data, the branch successive issues can not be predicted. Then, the branch must be thought as possibly always mispredicted and its maximum misprediction count equals its execution count. On the other hand, path analysis might find that one of the paths (e.g. the *else ...* path, reached by the branch when it is *taken*) is far longer than the other one. Then the branch is always taken on the longest path and its misprediction count on this path is at most two (depending on the initial state of the prediction counter). This is the reason why a global analysis of branch prediction might be preferable to achieve tight WCET estimates.

Modeling dynamic branch prediction within the IPET framework consists in adding some linear constraints that link the misprediction counts to the execution counts by expressing the behavior of the branch predictor. This solution has first been proposed by Li *et al.* [7].

3.3. Li's model of branch prediction

In [7], Li *et al.* proposed a model of a history-based 1-bit branch predictor. The direction is predicted from a single bit that reflects the direction followed at the last execution. They consider a scheme where the BHT is indexed by a History Register. However, they only model a "history-alone" indexing while real implementations combine some bits of the Program Counter and history bits to form the index. Typically, the history bits would be concatenated to or XOR-ed with some bits of the PC. A solution to model this has been sketched out in Li's paper but it adds complexity and we will not consider it in this paper. The model also accounts for the interferences due to aliasing.

Table 3 lists the constraints of Li's model. The first part of the system expresses the history-based indexing of the BHT. The second part describes the possible effects of aliasing. The notations are clarified in Table 2.

4. Modeling 2-bit branch predictors

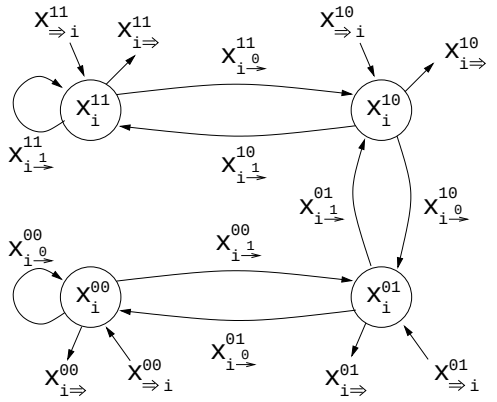

Figure 4. Variables used to model prediction counter of the branch in block i

Behavior of a 2-bit prediction counter When a branch executes, its 2-bit saturating counter is updated as described in Section 2. The behavior of a prediction counter can be expressed using the variables specified in Figure 4. The set of constraints to be added to the basic ILP formulation for WCET estimation is listed in Table 4 (these constraints were commented in earlier papers like [5]).

Putting it all together Extending Li's model to a 2-bit prediction scheme introduces additional variables and constraints. In particular, the formulation of a prediction counter behavior has to be revised to express that several branches might update its value. For example, if a counter is indexed by two different branches b_1 and b_2, a transition from state c to state c' might be fired by b_1 executed after b_2, or b_2 after b_1, or b_1 after b_1, or b_2 after b_2. The resulting model is shown in the two rightmost columns of Table 5.

In a PC-based predictor, the table of 2-bits counters is no more indexed by the branch history but by branch addresses. As a matter of fact, a branch always uses the same counter but this counter might be shared with other branches. The leftmost column of Table 5 shows which constraints are used to describe a PC-based branch predictor. In this model, π does not represent the history but

Table 2. Notations

$\mathcal{X}$: set of blocks in the program
$\mathcal{B}$: set of blocks that end with a branch
$\mathcal{C} = \{00, 01, 10, 11\}$: prediction counter states
$\mathcal{D} = \{0, 1\}$: possible branch directions
x_b : execution count of block $b \in \mathcal{X}$
$x_{b\overset{d}{\rightarrow}}$: execution count of the edge that leaves block $b \in \mathcal{B}$ when the branch direction is d
$m_{b\overset{d}{\rightarrow}}$: misprediction count of the branch at the end of block $b \in \mathcal{B}$ when the direction is d
Π : set of all the possible h-bit branch histories
$\mathcal{P}_b$: set of the predecessors of block $b \in \mathcal{B}$ that end with a branch
$\mathcal{S}_b$: set of the successors of block $b \in \mathcal{B}$ that end with a branch
x_b^π : execution count of block $b \in \mathcal{B}$ when the branch history is π
m_b^π : misprediction count of the branch at the end of block $b \in \mathcal{B}$ when the history was π
$\pi.d$: new value of branch history π after a branch has been executed with direction d
$p \overset{d}{\rightarrow} s$: true if block $p \in \mathcal{P}_s$ ends with a branch that has direction d to block s
Π_b : set of all the possible h-bit branch histories at block $b \in \mathcal{B}$
$\mathcal{B}^\pi$: set of blocks that end with a branch and can execute with history π
$c_{p,s}^\pi$: conflict count with history π at block $s \in \mathcal{B}$ when the last occurrence of history π was at block $p \in \mathcal{B}$
$c^\pi_{p\overset{d}{\rightarrow},s}$: conflict count with history π at block $s \in \mathcal{B}$ when the last occurrence of history π was at block $p \in \mathcal{B}$ and the executed direction of the branch in p was d
$\mathcal{P}_b^\pi$: set of the predecessors of block $b \in \mathcal{B}$ that end with a branch and can execute with history π
$\mathcal{S}_b^\pi$: set of the successors of block $b \in \mathcal{B}$ that end with a branch and can execute with history π
$c_{start,b}^\pi = 1$ when block b is the first executed with history π
$c_{b,end}^\pi = 1$ when block b is the last executed with history π
$\mathcal{A}$: set of all the PC-indexes that point to an entry of the BHT used by at least one branch
$x_{p,s}^{\pi,c}$: conflict count with BHT-index π at block $s \in \mathcal{B}$ when the last occurrence was at block $p \in \mathcal{B}$ The counter accessed by π is in state c before the execution of p
$x^{\pi,c}_{p\overset{d}{\rightarrow},s}$: conflict count with BHT-index π at block $s \in \mathcal{B}$ when the last occurrence was at block $p \in \mathcal{B}$. The counter accessed by π is in state c before the execution of p and the executed direction of the branch in p was d

Table 3. Constraints that model a history-based 1-bit branch predictor

$\forall b \in \mathcal{B}$	$x_b = \sum_{\pi\in\Pi} x_b^\pi$ $\forall d \in \mathcal{D}, m_{b\overset{d}{\rightarrow}} = \sum_{\pi\in\Pi} m^\pi_{b\overset{d}{\rightarrow}}$ $\forall \pi \in \Pi_b, \forall d \in \mathcal{D}, m^\pi_{b\overset{d}{\rightarrow}} \le x^\pi_{b\overset{d}{\rightarrow}}$	History
	$\forall \pi \in \Pi_b,$ $x_b^\pi \le \sum_{p\in\mathcal{P}_b}\sum_{\pi'\in\Pi} x_p^{\pi'}$ where $\pi = \pi'.d$ and $p \overset{d}{\rightarrow} b$ $x_b^\pi \le \sum_{s\in\mathcal{S}_b} x_s^{\pi'}$ where $\pi' = \pi.d$ and $b \overset{d}{\rightarrow} s$	History
$\forall b \in \mathcal{B}$	$\forall \pi \in \Pi_b, x_b^\pi = \sum_{s\in\mathcal{S}_b^\pi} c_{b,s}^\pi$ $\forall \pi \in \Pi_b, x_b^\pi = \sum_{p\in\mathcal{P}_b^\pi} c_{p,b}^\pi$	Conflicts
	$\forall \pi \in \Pi_b, \forall s \in \mathcal{S}_b^\pi,\ c_{b,s}^\pi = c^\pi_{b\overset{0}{\rightarrow},s} + c^\pi_{b\overset{1}{\rightarrow},s}$ $\forall d \in \mathcal{D},\ \sum_{\pi\in\Pi_b}\sum_{s\in\mathcal{S}_b^\pi} c^\pi_{b\overset{d}{\rightarrow},s} \le x_{b\overset{d}{\rightarrow}}$	Conflicts
	$\forall \pi \in \Pi_b,$ $m_b^\pi = m^\pi_{b\overset{0}{\rightarrow}} + m^\pi_{b\overset{1}{\rightarrow}}$ $m^\pi_{b\overset{0}{\rightarrow}} \le \sum_{s\in\mathcal{S}_b^\pi} c^\pi_{b\overset{0}{\rightarrow},s}$ $m^\pi_{b\overset{0}{\rightarrow}} \le \sum_{p\in\mathcal{P}_b^\pi} c^\pi_{p\overset{1}{\rightarrow},b}$ $m^\pi_{b\overset{1}{\rightarrow}} \le \sum_{s\in\mathcal{S}_b^\pi} c^\pi_{b\overset{1}{\rightarrow},s}$ $m^\pi_{b\overset{1}{\rightarrow}} \le \sum_{p\in\mathcal{P}_b^\pi} c^\pi_{p\overset{0}{\rightarrow},b}$	Conflicts
$\forall \pi \in \Pi$	$\sum_{b\in\mathcal{B}^\pi} c^\pi_{start,b} \le 1 \qquad \sum_{b\in\mathcal{B}^\pi} c^\pi_{b,end} \le 1$	Conflicts

Table 4. Constraints that model 2-bit branch prediction

$\forall b \in \mathcal{B}$	$x_b^{00} = x^{00}_{b\overset{0}{\rightarrow}} + x^{01}_{b\overset{0}{\rightarrow}} + x^{00}_{\Rightarrow b}$ $x_b^{01} = x^{10}_{b\overset{0}{\rightarrow}} + x^{00}_{b\overset{1}{\rightarrow}} + x^{01}_{\Rightarrow b}$ $x_b^{10} = x^{11}_{b\overset{0}{\rightarrow}} + x^{01}_{b\overset{1}{\rightarrow}} + x^{10}_{\Rightarrow b}$ $x_b^{11} = x^{11}_{b\overset{1}{\rightarrow}} + x^{10}_{b\overset{1}{\rightarrow}} + x^{11}_{\Rightarrow b}$
	$\forall c \in \mathcal{C}, x_b^c = x^c_{b\overset{0}{\rightarrow}} + x^c_{b\overset{1}{\rightarrow}} + x^c_{b\Rightarrow}$
	$x_b = \sum_{c\in\mathcal{C}} x_b^c$ $\forall d \in \mathcal{D}, x_{b\overset{d}{\rightarrow}} = \sum_{c\in\mathcal{C}} x^c_{b\overset{d}{\rightarrow}} + \sum_{c\in\mathcal{C}} x^c_{b\overset{d}{\Rightarrow}}$
	$\sum_{c\in\mathcal{C}} x^c_{\Rightarrow b} = 1 \qquad \sum_{c\in\mathcal{C}} x^c_{b\Rightarrow} = 1$ $\forall c \in \mathcal{C}, x^c_{b\Rightarrow} = \sum_{d\in\mathcal{D}} x^c_{b\overset{d}{\Rightarrow}}$
	$m_{b\overset{0}{\rightarrow}} = x^{10}_{b\overset{0}{\rightarrow}} + x^{11}_{b\overset{0}{\rightarrow}}$ $m_{b\overset{1}{\rightarrow}} = x^{00}_{b\overset{1}{\rightarrow}} + x^{01}_{b\overset{1}{\rightarrow}}$

Table 5. Constraints that model a PC- or history-based branch predictor

	$\forall b \in \mathcal{B}$	$x_b = \sum_{\pi \in \Pi} x_b^\pi$ $\forall d \in \mathcal{D}, m_{b \xrightarrow{d}} = \sum_{\pi \in \Pi} m^\pi_{b \xrightarrow{d}}$ $\forall \pi \in \Pi_b, \forall d \in \mathcal{D}, m^\pi_{b \xrightarrow{d}} \leq x^\pi_{b \xrightarrow{d}}$
		$\forall \pi \in \Pi_b,$ $x_b^\pi \leq \sum_{p \in \mathcal{P}_b} \sum_{\pi' \in \Pi} x_p^{\pi'}$ where $\pi = \pi'.d$ and $p \xrightarrow{d} b$ $x_b^\pi \leq \sum_{s \in \mathcal{S}_b} x_s^{\pi'}$ where $\pi' = \pi.d$ and $b \xrightarrow{d} s$
$\forall b \in \mathcal{B}$	$\forall b \in \mathcal{B}$	$\forall d \in \mathcal{D}, \sum_{\pi \in \Pi_b} \sum_{s \in \mathcal{S}_b^\pi} \sum_{c \in \mathcal{C}} x^{\pi,c}_{b \xrightarrow{d},s} \leq x_{b \xrightarrow{d}}$
	$\forall b \in \mathcal{B}$ $\forall \pi \in \Pi_b$	$x_b^\pi = \sum_{s \in \mathcal{S}_b^\pi} \sum_{c \in \mathcal{C}} x^{\pi,c}_{b,s}$ $x_b^\pi = \sum_{p \in \mathcal{P}_b^\pi} \sum_{c \in \mathcal{C}} x^{\pi,c}_{p,b}$
		$\forall s \in \mathcal{S}_b^\pi, \forall c \in \mathcal{C}, \; x^{\pi,c}_{b,s} = x^{\pi,c}_{b \xrightarrow{0},s} + x^{\pi,c}_{b \xrightarrow{1},s}$
		$x_b^{\pi,00} = \sum_{p \in \mathcal{P}_b^\pi} (x^{\pi,00}_{p \xrightarrow{0},b} + x^{\pi,01}_{p \xrightarrow{0},b}) + x^{\pi,00}_{start,b}$ $x_b^{\pi,00} = \sum_{s \in \mathcal{S}_b^\pi} \sum_{d \in \mathcal{D}} x^{\pi,00}_{b \xrightarrow{d},s} + \sum_{d \in \mathcal{D}} x^{\pi,00}_{b \xrightarrow{d},end}$ $x_b^{\pi,01} = \sum_{p \in \mathcal{P}_b^\pi} (x^{\pi,00}_{p \xrightarrow{1},b} + x^{\pi,10}_{p \xrightarrow{0},b}) + x^{\pi,01}_{start,b}$ $x_b^{\pi,01} = \sum_{s \in \mathcal{S}_b^\pi} \sum_{d \in \mathcal{D}} x^{\pi,01}_{b \xrightarrow{d},s} + \sum_{d \in \mathcal{D}} x^{\pi,01}_{b \xrightarrow{d},end}$ $x_b^{\pi,10} = \sum_{p \in \mathcal{P}_b^\pi} (x^{\pi,01}_{p \xrightarrow{1},b} + x^{\pi,11}_{p \xrightarrow{0},b}) + x^{\pi,10}_{start,b}$ $x_b^{\pi,10} = \sum_{s \in \mathcal{S}_b^\pi} \sum_{d \in \mathcal{D}} x^{\pi,10}_{b \xrightarrow{d},s} + \sum_{d \in \mathcal{D}} x^{\pi,10}_{b \xrightarrow{d},end}$ $x_b^{\pi,11} = \sum_{p \in \mathcal{P}_b^\pi} (x^{\pi,10}_{p \xrightarrow{1},b} + x^{\pi,11}_{p \xrightarrow{1},b}) + x^{\pi,11}_{start,b}$ $x_b^{\pi,11} = \sum_{s \in \mathcal{S}_b^\pi} \sum_{d \in \mathcal{D}} x^{\pi,11}_{b \xrightarrow{d},s} + \sum_{d \in \mathcal{D}} x^{\pi,11}_{b \xrightarrow{d},end}$
		$m^\pi_{b \xrightarrow{0}} = \sum_{s \in \mathcal{S}_b^\pi} x^{\pi,11}_{b \xrightarrow{0},s} + x^{\pi,10}_{b \xrightarrow{0},s}$ $m^\pi_{b \xrightarrow{1}} = \sum_{s \in \mathcal{S}_b^\pi} x^{\pi,00}_{b \xrightarrow{1},s} + x^{\pi,01}_{b \xrightarrow{1},s}$
$\forall \pi \in \mathcal{A}$	$\forall \pi \in \Pi$	$\sum_{b \in \mathcal{B}^\pi} \sum_{c \in \mathcal{C}} x^{\pi,c}_{start,b} \leq 1$ $\sum_{b \in \mathcal{B}^\pi} \sum_{c \in \mathcal{C}} x^{\pi,c}_{b,end} \leq 1$
PC-BASED	**HISTORY-BASED**	

instead the index of the BHT and it is calculated using branch instruction address ($|\Pi_b| = 1$).

5. Modeling Complexity

5.1. Definition

The term of *complexity* usually refers to the difficulty in solving mathematical problems. The complexity of a problem is often measured by the computation time, the number of steps or of arithmetic operations, or the memory space, required to solve it. In this paper, we do not consider the *computational* complexity of dynamic branch predictor models for WCET analysis. Instead, we focus on their *modeling* complexity. Our goal is to get an insight into the effort needed to:

- *build the set of constraints* that have to be added to the IPET formulation of WCET estimation to model branch prediction.
- *read and understand the model*: several models may be merged to describe a more complex mechanism. For example, constraints used to express the behavior of a global history vector can be associated to constraints that model 2-bit prediction counters to analyze the worst-case performance of a complex branch predictor. Successfully merging two models (ILP constraints) requires each of them being readable and understandable enough. In turn, the final model, which includes the two original ILP formulations plus some additional linking variables and constraints, should be as much readable as possible.
- *extend the model*: for example, a model for a 3-bit branch prediction counter might be derived from the 2-bit model. This is possible if the model exhibits scalability.

Estimating the modeling complexity might be useful either to determine which branch prediction schemes are easier to analyze. It can also help in pre-evaluating the size of the ILP problem to be solved.

We propose to express the modeling complexity of the ILP formulation of a branch predictor behavior by the means of a triplet (C, V, A) where C is the number of constraints, V is the number of variables involved in these constraints, and A is the system arity (the arity of a constraint is the number of variables it uses, and the arity of the system is the maximum arity among its constraints).

This triplet reveals the accessibility and the readability of the model. In the following, we estimate the modeling complexity of several branch predictors.

5.2. Modeling complexity dynamic branch predictors

To quantify the modeling complexity of branch predictors, we have computed the number of variables, the number of constraints and the system arity from Tables 3 and 5.

For history-based schemes, the complexity depends on: (i) the number of branches in the program; (ii) the number of possible history values for each branch; (iii) the number of conflicting branches for each possible history value; and (iv) the BHT size. The complexity of the 1-bit and 2-bit history-based predictors is expressed in Tables 6 and 7.

For PC-based schemes, the complexity depends on: (i) the number of branches in the program; (ii) the number of BHT entries indexed by at least one branch PC; and (iii) the number of conflicting branches for each BHT entry ($\mathcal{S}_b^{@}$). The complexity of the PC-based 2-bit predictor is given in Table 8.

Table 6. Modeling complexity of a history-based 1-bit predictor

$C = \sum_{b \in \mathcal{B}} (5 + 14 \times \lvert\Pi_b\rvert) + 2 \times \lvert\Pi\rvert$
$V = \sum_{b \in \mathcal{B}} (5 \times \lvert\Pi_b\rvert + 3 \times \sum_{\pi \in \Pi_b} \lvert\mathcal{S}_b^\pi\rvert$
$A = \max_{b \in \mathcal{B}}(1 + \lvert\Pi_b\rvert, 2 + \sum_{\pi \in \Pi_b} \lvert\mathcal{S}_b^\pi\rvert)$

Table 7. Modeling complexity of a history-based 2-bit predictor

$C = \sum_{b \in \mathcal{B}} (5 + 4 \times \lvert\Pi_b\rvert + \sum_{\pi \in \Pi_b} (12 + 4 \times \lvert\mathcal{S}_b^\pi\rvert)) + 2\lvert\Pi\rvert$
$V = \sum_{b \in \mathcal{B}} (5 \times \lvert\Pi_b\rvert + \sum_{\pi \in \Pi_b} (20 + 12 \times \lvert\mathcal{S}_b^\pi\rvert))$
$A = \max_{b \in \mathcal{B}}(\sum_{\pi \in \Pi_b} 4 \times \lvert\mathcal{S}_b^\pi\rvert) + 1$

Table 8. Modeling complexity of a PC-based 2-bit predictor

$C = 2 \times \lvert\mathcal{A}\rvert + \sum_{b \in B} (14 + 4 \times \lvert\mathcal{S}_b^{@}\rvert)$
$V = \sum_{b \in B} (25 + 12 \times \lvert\mathcal{S}_b^{@}\rvert)$
$A = \max_{b \in \mathcal{B}}(4 \times \lvert\mathcal{S}_b^{@}\rvert) + 2$

6. Modeling complexity and program size

In this section, our intent is to derive some quantitative results from the formulas established in the previous section. The purpose is to show how fast the complexity grows with the program size. To derive values for the parameters of the modeling complexity equations, we have made some measurements on the SNU-RT benchmarks [2].

We consider a 128-entry Branch History Table (like in the ARM11 core). We have measured the number of conflicting branches for each used entry of a PC-indexed BHT. Numerous entries are used by only one branch, the other ones are indexed by 2 to 6 branches. Then, to analyze the PC-based predictor, we consider a mean number of two branches sharing the same entry. We also have measured the number of possible 7-bit history values for every branch. It ranges from 4 to 34 on a mean for the different benchmarks, and the mean value is 18. Finally, we assume a perfect distribution of the branches on the BHT entries. Table 9 gives the results obtained under these assumptions.

As expected, we observe the model of the PC-based 2-bit predictor exhibits the smallest complexity while the model of the history-based 2-bit predictor is the most complex one. The difference grows with the number of branches.

As far as we know, it is not possible to establish a direct relation between the computational complexity and the size of the model. This is why we did not give any quantitative results concerning the computational complexity of the three models. Nevertheless, the difference between the modeling complexity of the models of PC-based and history-based predictors is so important that we can expect that the PC-based scheme is also the far easiest to solve.

7. Conclusion

Until recently, only very simple processors were used within embedded system designs, especially for applications featuring hard real time constraints. However, since the performance requirements are continuously growing, it becomes unavoidable to use high performance processors. Unfortunately, high performance is generally obtained by using complex hardware schemes that exhibit a very dynamic behavior. And a dynamic behavior does generally not fit with static analysis.

In this paper, we focused on the analysis of hardware branch predictors within the estimation of Worst-Case Execution Times of applications that must meet hard deadlines. We are aware that today engineers would probably disable the dynamic branch predictor to favor timing predictability. However, we feel that like caches and pipelined out-of-order execution that were banned a few years ago and are now used in some industrial critical systems, branch predictors will have to be taken into account in the next few years.

[2] http://archi.snu.ac.kr/realtime/benchmark/

Table 9. Modeling complexity of branch predictor as a function of the number of branches

# branches	bimodal			global 1-bit			global 2-bits		
	C	V	A	C	V	A	C	V	A
10	310	730	18	2,826	1,660	129	4,199	7,538	175
40	1,240	2,920	18	10,536	15,750	129	28,176	66,600	478
160	4,960	11,680	18	41,376	208,800	424	306,336	849,600	1,693

Some work has been done quite recently to develop models to analyze worst-case branch misprediction counts as part of WCET estimation [7]. Our purpose in this paper was to extend this model to other prediction schemes and to propose a framework to understand the complexity of modeling these schemes. Having an idea of this complexity can help in determining which kind of scheme might fit the allocated capacity in terms of modeling effort.

We proposed to express the modeling complexity as the combination of the number of ILP constraints, the number of variables and the maximum constraint arity. They convey how much it is difficult to write and read the ILP formulation.

We provided quantitative results that highlight that the modeling complexity of history-based branch predictors is far much higher than that of a PC-based predictor. This shows that complex branch prediction schemes (like in the ARM Cortex-A8) might not fit the analyzability requirements of critical real-time systems.

References

[1] M. Baron. Cortex-A8: High speed, low power. *Microprocessor Report*, november 2005.

[2] I. Bate and R. Reutemann. Worst-case execution time analysis for dynamic branch predictors. In *16^{th} Euromicro Conference on Real Time Systems*, pages 215–222, June 2004.

[3] I. Bate and R. Reutemann. Efficient integration of bimodal prediction and pipelin analysis. In *11^{th} international conference on embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 39–44, August 2005.

[4] C. Burguière and C. Rochange. A Contribution to Branch Prediction Modeling in WCET Analysis. In *Design, Automation and Test in Europe (DATE'05)*, March 2005.

[5] C. Burguière and C. Rochange. History-based Schemes and Implicit Path Enumeration. In *6^{th} International Workshop On Worst-Case Execution Time (WCET) Analysis*, July 2006.

[6] C. Burguière, C. Rochange, and P. Sainrat. A Case for Static Branch Prediction Modeling in Real-Time Systems. In *11^{th} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, August 2005.

[7] X. Li, T. Mitra, and A. Roychoudhury. Modeling control speculation for real-time analysis. In *Real-Time Systems*, volume 29, pages 27–58, January 2005.

[8] Y.-T. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *ACM SIGPLAN Notices*, volume 30, pages 88–98, 1995.

[9] J. Smith. A study of branch prediction strategies. In *8^{th} annual ACM/IEEE international symposium on Computer Architecture*, 1997.

[10] T.-Y. Yeh and Y. Patt. Alternative implementation of the two-level adaptative branch prediction. In *19^{th} international symposium on Computer Architecture*, 1992.

Declaration

All necessary clearances for the publication of this paper have been obtained. If accepted, the author will present the paper at the conference and will prepare the final manuscript in time for inclusion in the conference proceedings.