Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 62 (2015) 650 – 653

Procedia Computer Science

The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

2-Bit Branch Predictor Modeling Using Markov Model

Reem Elkhouly[a], Ahmed El-Mahdy[a,b], Amr Elmasry[a,c]

[a] *Dept. of Computer Science and Engineering, Egypt-Japan University of Science and Technology (E-JUST), Alexandria, Egypt*
[b] *On leave from Dept. of Computer Engineering and Systems, Alexandria University, Alexandria, Egypt*
[c] *Dept. of Computer Engineering and Systems, Alexandria University, Alexandria, Egypt*

Abstract

Power consumption is a very important issue when it comes to embedded devices, therefore every processing cycle should be optimally utilized and considered. In speculated execution, highly mispredicted branches are considered a critical threat for both time and power saving. In this paper, we show that, for a specific branch, misprediction rate of a 2-bit branch predictor can be precisely calculated using Markov model. Further, this can be done offline for more power saving. Thus, a decision of replacing the branch with conditional (predicated) instructions instead of counting on the predictor can be made.

Keywords: Branch Predictor; Modeling; Embedded Systems.

1. Introduction

Low power processors such as the ones used in embedded devices are becoming ubiquitous. These limited hardware platforms have to be carefully considered on the software design phase. Relying on speculative execution may not always be an optimal course of action in terms of time cost that translates into more power consumption[1]. As the processor mis-loads the next instruction to be executed, then flushes it to load another instruction instead[2], thereby wasting valuable processing cycles. So, if those power greedy branches are precisely detected in advance, the processor can stall fetching instructions till the branch target is known. In accordance, a significant amount of power is saved. Predicated instructions were introduced to overcome misprediction by converting control-dependence into data-dependence via guarded execution[3]. However, predication comes at the extra cost of executing ‘nullified’ instructions, potentially degrading performance and costing more processing cycles. Moreover, branches interact in terms of allowing for different execution schedules, for which finding the optimal schedule is generally a hard combinatorial search problem. Therefore, these type of instructions should be carefully considered. In this paper, we introduce a Markov model for the 2-bit branch predictor, usually used for embedded processors such as (ARMv6k), that can estimate the misprediction rate offline. Consequently, this information could be used in the software compiling to convert some branches to otherwise costly predicated instruction and enhancing the overall performance and time cost, hence saving power consumption.

2. The Model

The 2-bit saturating counter is a Finite State Machine (FSM) that is widely used as a branch predictor; see Figure 1. We consider modeling this FSM as a Markov chain where the probability of success (p) is the probability of a branch

doi:10.1016/j.procs.2016.05.115

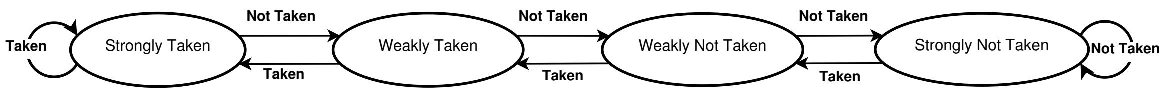

Fig. 1. 2-bit Saturating Counter Branch Predictor

being taken and the probability of failure ($q = 1 - p$) is the probability of a branch being not taken. Then, the model can be expressed with the following equations:

$$\pi \times \begin{array}{c|cccc} & S_{st} & S_{wt} & S_{wnt} & S_{snt} \\ \hline S_{st} & p & q & 0.0 & 0.0 \\ S_{wt} & p & 0.0 & q & 0.0 \\ S_{wnt} & 0.0 & p & 0.0 & q \\ S_{snt} & 0.0 & 0.0 & p & q \end{array} = \pi$$

$$\text{,where } \pi = \begin{bmatrix} x\ y\ z\ w \end{bmatrix} \text{ and } x + y + w + z = 1$$

$$x = \frac{p^3}{p^3 + p^2q + pq^2 + q^3} \qquad y = \frac{p^2q}{p^3 + p^2q + pq^2 + q^3}$$

$$z = \frac{pq^2}{p^3 + p^2q + pq^2 + q^3} \qquad w = \frac{q^3}{p^3 + p^2q + pq^2 + q^3}$$

$$P(correctprediction) = P(success) = p(x + y) + q(z + w) = \frac{p^3 + q^3}{p^3 + p^2q + pq^2 + q^3}$$

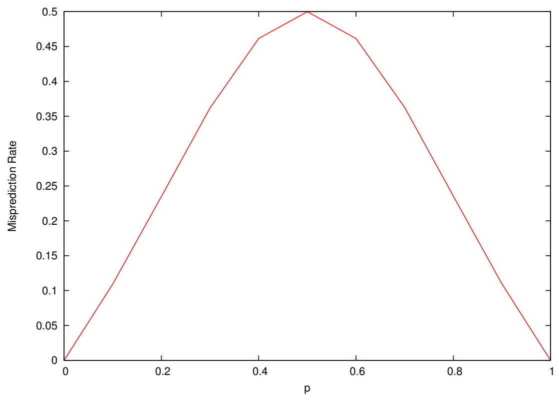

Fig. 2. Misprediction rate calculated by the model

When we apply the model to the range of branch probability to be taken or not taken starting from zero to one, we get the result shown in Figure 2. The peak of the graph at $p = 0.5$ represents the maximum uncertainty. That is when the branch is 50% mispredicted. While the tails of the graph indicate 100% correct prediction as the branch is always taken or always not taken.

3. Experiments

We simulated the behaviour of the 2-bit predictor to capture the correlation between the input data – which is the basic specifier of the probability to take the branch (p) – and the probability of correct prediction ($P_{exper.}$, the experimental value). Then, we used the model to calculate the same probability analytically ($P_{math.}$, the calculated value). We ran an iterative mergesort algorithm; see Algorithm 1, on inputs that are integers from different distributions: uniform, normal, and sorted data. The comparison between the mathematically calculated and the simulation counted probabilities is shown in Table 3. We also simulated the predictor and tested it on the RasberryPi ARM1176JZF-S (ARMv6k) 700 MHz. In this processor, the branch misprediction penalty is 6 cycles. To show how expensive the misprediction is on the ARM processor, we test a simple code that would contain a single branch instruction if the code version was written using branched code (contains jumps) as opposed to the other version which was written using conditional (predicated) instructions. Both versions of the code are shown in Listing 1, 2. We executed these pieces of code with different sets of input data (1M integers), which consequently control the probability of the branch to be taken (or the conditional instruction to be executed),and we recorded the results shown in Figure 3. Similar to what is found in Figure 2, the peak of runtime is at 50% taken branch due to maximum mispredic-

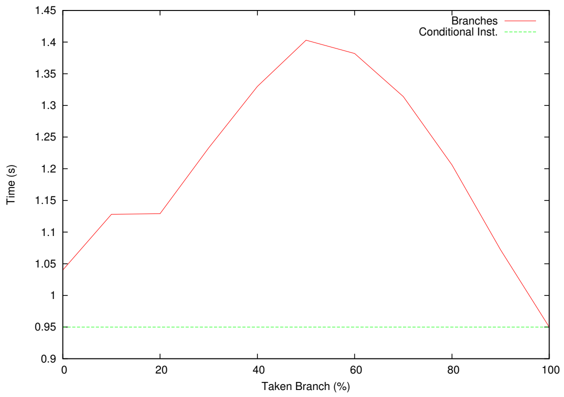

Fig. 3. ARM Performance Measurements