2-Bit Branch Predictor Modeling Using Markov Model

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 predicated instructions instead of counting on the predictor can be made.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

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 which translate into more power consumption. As the processor mis-loads the next instruction to be executed, then flushes it to load another instruction instead, thereby wasting valuable processing cycles. So, if those power greedy branches are precisely detected in advance, the processor can stall fetching instruction till the branch target is known, thus, a significant amount of power is saved. Predicate instructions were introduced to overcome misprediction by converting control-dependence into data-dependence via guarded execution. 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. Markov Chain of 2-bit Saturating Counter Branch Predictor

The 2-bit saturating counter is a Finite State Machine (FSM) that is widely used as a branch predictor, Figure 1. We consider modeling this FSM as a markov chain where the probability of success $p$ is the probability of a branch 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.
Fig. 1. 2-bit Saturating Counter Branch Predictor

\[
\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \times \begin{pmatrix} S_{st} & S_{wt} & S_{wnt} & S_{snt} \\ S_{st} & p & 0.0 & 0.0 \\ S_{wt} & 0.0 & p & 0.0 \\ S_{wnt} & 0.0 & 0.0 & q \end{pmatrix} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}
\]

and, \(x + y + w + z = 1\)

3. 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. That is often happens in random data comparisons as what usually exist in searching and sorting algorithms. While the tails of the graph indicates 100% correct prediction as the branch is always taken or always not taken. Also, we simulated the behaviour of the 2-bit predictor in code to capture the correlation between the input data – which is the basic specifier of the probability to take the branch – and the probability of correct prediction. The considered algorithm for this experiment is the iterative mergesort. Then, we use the model to calculate the same probability. The comparison between the mathematically calculated and the simulation counted probabilities is shown in Table 1. Input data distribution affects the probability of correct prediction, as shown for uniform and normal distributed and sorted numbers. Usually, input data would be uniformly distributed.

4. Analysis on RaspberryPi

As a test case we simulate the predictor and test it on the RaspberryPi 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 contains a single branch instruction if the code written using a jump or else using conditional instruction. Both versions of the code are shown in Listing 1, 2. When these pieces of code run on different sets of input data (1M integers) that controls the probability of the branch to be taken (or the conditional instruction to be executed), we record 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 misprediction (predictor uncertainty). It is worth noting that, the best runtime of the branched code is the same value of conditional instruction code and would be even better for larger if-body.
Table 1. Probability of Correct (Successful) Prediction of Mergesort Branches

<table>
<thead>
<tr>
<th>Data Distribution</th>
<th>Outer for loop branch</th>
<th>Left part merge branch</th>
<th>Right part merge branch</th>
<th>Data Comparison branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>$p = 0.1284$</td>
<td>$p = 0.105655$</td>
<td>$p = 0.09817$</td>
<td>$p = 0.52583$</td>
</tr>
<tr>
<td></td>
<td>$P_{\text{success}}$</td>
<td>0.867704</td>
<td>0.866077</td>
<td>0.881864</td>
</tr>
<tr>
<td></td>
<td>$P_{\text{math}}$</td>
<td>0.855814</td>
<td>0.883489</td>
<td>0.892418</td>
</tr>
<tr>
<td>Normal</td>
<td>$p = 0.1284$</td>
<td>$p = 0.105655$</td>
<td>$p = 0.096506$</td>
<td>$p = 0.52486$</td>
</tr>
<tr>
<td></td>
<td>$P_{\text{success}}$</td>
<td>0.867704</td>
<td>0.869048</td>
<td>0.890183</td>
</tr>
<tr>
<td></td>
<td>$P_{\text{math}}$</td>
<td>0.855814</td>
<td>0.883489</td>
<td>0.894591</td>
</tr>
<tr>
<td>Sorted</td>
<td>$p = 0.1284$</td>
<td>$p = 0.0$</td>
<td>$p = 316$</td>
<td>$p = 316$</td>
</tr>
<tr>
<td></td>
<td>$P_{\text{success}}$</td>
<td>0.867704</td>
<td>0.997024</td>
<td>0.660714</td>
</tr>
<tr>
<td></td>
<td>$P_{\text{math}}$</td>
<td>0.855814</td>
<td>1.0</td>
<td>0.503531</td>
</tr>
</tbody>
</table>

Listing 1. Conditional instruction code

```assembly
main:  ldr r3, address_of_return  
       str lr,[r3]  
       ldr r0, address_of_scan_pattern  
       bl scanf  
       ldr rl, address_of_message  
       cmp rl,#0  
       add r1, rl, rl  
       ldr rl, address_of_result  
       str rl,[rl]  
       bl printf  
       ldr rl, address_of_return  
       b lr  

address_of_scan_pattern: word scan_pattern  
address_of_message: word message  
address_of_result: word result
```

Listing 2. Branched code

```assembly
main:  ldr r3, address_of_return  
       str lr,[r3]  
       ldr r0, address_of_scan_pattern  
       ldr rl, address_of_number_read  
       bl scanf  
       ldr rl, address_of_number_read  
       ldr rl,[rl]  
       cmp rl,#10  
       bl b1  
       add r1, rl, rl  

b1:    ldr r0, address_of_result  
       str rl,[r0]  
       ldr r0, address_of_message  
       ldr rl,[rl]  
       bl printf  
       ldr r3, address_of_return  
       ldr lr,[r3]  
       b lr  

address_of_scan_pattern: word scan_pattern  
address_of_number_read: word number_read  
address_of_message: word message  
address_of_result: word result
```

5. Related Work

Analyze the modeling complexity of dynamic branch prediction schemes on the bases of worst-case execution times (WCETs) of their critical tasks is studied in. Finite state machines are used for modeling branch behaviour during runtime in.

6. Conclusion and Future Work

In this paper, we revisit the problem of optimizing time cost in power-limited processors, due to wasted work done by the processor when a branch predictor mistakes. We presented a markov model of the 2-bit branch predictor used in ARMv6 processor. This model enables offline analysis to detect highly mispredicted branches. Hence, these branches are better to be converted into predicated execution using conditional instructions (already exist in the ARM ISA). The model is tested on a mergesort program and another simple code. In the future, we are working to expand the model for more complicated architectures and verify against larger programs.

References