Dynamic Selection of Branch History Length for Branch Predictions

Xiangying Chen†, Sang-Jeong Lee‡ and Pen-Chung Yew†

† Dept. of Computer Science and Engineering, Univ. of Minnesota, Minneapolis, MN 55455
‡ Division of Information Technology Engineering, Soonchunhyang Univ., Chungnam, Korea
chenxy@msi.umn.edu, sjlee@sch.ac.kr, yew@cs.umn.edu

Abstract

We propose a scheme based on the concept of Fourier analysis to dynamically adjust the length of the global Branch History Register (BHR) used in several popular branch prediction schemes such as gshare [5] and BiMode [12]. The scheme allows us to identify the most dominating branch history patterns in a time period, and use them to determine the appropriate branch history length (BHL) for the next time period. The simulation results show that, using the proposed scheme to adjust BHL dynamically, it can better adapt to those programs which have more erratic branch history patterns and to those environments in which context switches occur frequently.

1. Introduction

An efficient and effective branch prediction scheme has become increasingly critical in multiple-issue processors such as superscalars and trace processors [18]. Among a large number of branch prediction schemes proposed so far, two-level branch predictors using a global branch history to index into a Pattern History Table (PHT) similar to that proposed by Yeh and Patt [2,4], have shown to be very effective. Many schemes have since been proposed to improve their prediction accuracy [5,9-12,15-17].

Most of the proposed improvements have been focused on reducing the interference among
different branch patterns which are mapped to the same entry in the PHT [5,9,10,12]. Such interference could be positive, negative or neutral. It has been shown that negative interference is more dominant than positive interference [9]. Hence, reducing interference is crucial to the effectiveness of such schemes. Increasing the branch history length (thus, increasing the size of PHT) and incorporating branch addresses (e.g. exclusive-or'ed with the branch addresses as in the gshare scheme [5]) are very effective in reducing the negative interference. Some of the more recent schemes such as Agree [9] and BiMode [12] further reduce the negative interference by separating the branch outcomes from their branch behavior with an extra level of table lookup, or providing associativity in PHT to reduce conflicts among similar branch patterns [10]. Heil et. al[17] incorporates data value prediction to reduce the branch misprediction rate for some difficult to predict branches.

However, except [11] [15], almost all of the schemes assume a single fixed branch history length (BHL) for all of the programs. Studies have shown that different programs will need different branch history lengths to achieve their best branch prediction results [15]. In [15], they propose a simple strategy to change the BHL dynamically by counting the number of mispredictions in each time period, and use the information to determine a new BHL for the next time period. Even with a much simplified strategy, they found that such adaptive schemes can obtain good performance, especially in a context switching environment in which PHT is flushed frequently.

In this paper, we propose a new dynamic scheme which can adjust BHL dynamically during the course of a program execution. The scheme is based on the concept of the Fourier analysis. Since our scheme is used primarily for selecting a suitable BHL based on the branch history patterns, it can be used in conjunction with most other branch prediction schemes using only a fixed BHL. In this paper, we demonstrate our scheme using gshare [5] and Bimode [12], and show that further
improvement can be obtained using the proposed dynamic selection scheme.

For the rest of the paper, we will first explain the motivation using the Fourier analysis by illustrative examples in Section 2. Approximation to the Fourier transform analysis and some possible implementations using this idea are discussed in Section 3. In Section 4, we will show some simulation results, followed by some discussions. In Section 5, we will have our conclusions.

2. Motivation

To illustrate the basic ideas behind using Fourier analysis on branch history patterns to select a suitable BHL, we use two example programs in this section: a 2D computational fluid dynamic program CFD2D, and a simple synthetic program.

2.1. CFD2D

CFD2D is a well-developed 2-D Computational Fluid Dynamics (CFD) solver. Figure 1(a) through Figure 1(c) show some partial branch history profiles of three different computation-intensive subroutines in CFD2D. In Figure 1, the vertical axis indicates if a branch has been taken or not taken, numerically represented by 1 and 0, respectively. Each point on the horizontal axis corresponds to one branch instruction executed (all other non-branch instructions are excluded). Hence, the horizontal axis represents the dynamic execution sequence of branch instructions, which more or less corresponds to time. A snapshot of branch execution profile plotted in Figure 1(a) comes from subroutine VOLGRD, which is used to calculate geometry parameters of each finite element. It contains a lot of loop iterations. As can be observed, the dominant pattern of branch taken or not taken is very uniform, and there is no other obvious pattern exists in this snapshot. The branch execution profile in Figure 1(b) comes from subroutine FLUXX. It is used
to calculate the flow flux. In this subroutine, the branch directions are closely related to the random flow flux movement. It can be seen that at least one dominant pattern exists, and it is associated with some other sub-patterns which could be examined further by the Fourier analysis. Also shown in Figure 1(c) is part of the branch execution profile taken from subroutine SLVUST which simulates the time-dependent boundary layer profiles. There are several more complicated patterns exist in that profile. We use Fourier spectrum analysis to analyze the branch execution profile of the above three subroutines, and its results are shown in Figure 2(a) through Figure 2(c).

In that analysis, $2^{12}$ sample points (i.e. dynamic branch instructions executed) are used in each case, and the values of final power spectra are normalized to 1. In the figures, the horizontal axis represents simply the inverse of the length (or the period) of each repeated branch history pattern, i.e. the frequency of each repeated branch history pattern. The vertical axis represents the amplitude of power spectrum, which can be interpreted as the probability that a branch history pattern with a particular frequency would occur. A dominant or principal peak can be seen in Figure 2(a). Notice that it has a relatively large period, i.e. a very low frequency. It indicates that its branch history patterns of taken (or not taken) would remain unchanged for a very long period of time before it changes to another state. Figure 2(a) also shows several different harmonic and ultraharmonic resonances with considerable high amplitudes. It indicates that several branch history patterns from different loops may have occurred simultaneously, and are coupled together in its
A FFT-Based Adaptive Branch Prediction Scheme

Figure 1: Global branch histories from three branch-intensive subroutines of CFD2D branch history profile. Figure 2(b) shows the power spectrum of a typical random process, in which amplitude of the spectrum decreases rapidly along the frequency axis. A dominant peak is clearly shown and has a particular frequency, which is higher than the one in Figure 2(a). Apparently, the alternation between branch taken and not taken happens more frequently than in the previous case. In another word, the period in which branches are taken (or not taken) is shorter than that in the previous case. Figure 2(c) depicts a power spectrum in which it has a dominant peak together with several isolated harmonic resonances. It indicates that there is only one strong pattern dominating the branch history profile. The amplitudes of those harmonic resonances form a unique pattern, called *beats*. The sinusoidal-shape *beats* seems to indicate certain regularity of the branch history patterns.
2.2 A Synthetic Program

Let us consider a hypothetical program which contains three branch history patterns governed by three different frequencies. The graphical representations of those branch history patterns are given in Figure 3. To demonstrate the effect of the BHL on the performance of a branch predictor, a global branch predictor [5] is used on these three branch history patterns with different BHR lengths (i.e. different BHL's). The resulting misprediction rates are plotted in

Figure 2: Fourier spectra from three global branch histories described in Figure 1
Figure 4. The horizontal axis in the Figure 4 shows the length of corresponding BHR (i.e. BHL) and the vertical axis shows the misprediction rates using a logarithmic scale.

From the study of this synthetic program, we can see that:

1. When the dominating frequency $f$ is 0.3333, the misprediction rate reaches the value which is very close to its minimum value at the point where the BHL equals 3. This BHL corresponds to the period of the dominating frequency (which is the inverse of the frequency, $1/f$). As can be observed in Figure 4, by increasing the BHL, the misprediction rate will decrease somewhat at the points where the BHL equals to 6, 9 and 12, which correspond to the one-second, the one-third and the one-fourth subharmonic, respectively. Similar phenomena can be observed for the other two frequencies at 0.1666 and 0.1111.

2. In each case discussed above, when the BHL reaches the half of the period of its dominating frequency, the misprediction rate drops dramatically. For example, in Figure 4, when BHL reaches 2 (actually 1.5), 3, and 5 (actually 4.5) for $f = 0.3333$, 0.1666, and 0.1111, respectively, the sharp drops occur.

Hence, using a BHL larger than the period of dominating history patterns has very little effect in improving the misprediction rate because it contains no extra useful history information. On the contrary, it might have detrimental effect on the overall misprediction rate. Let us use gshare as an example.

In the gshare scheme, for each particular branch instruction, it will XOR its address with the value of BHR and use the result as an index to access the PHT. A larger BHL means that, for each particular branch instruction, it will utilize more entries in the PHT than when it uses a smaller BHL. This most likely will cause more interference with other branch instructions in PHT. From our discussions above, most of the interference is negative. Hence, if no extra useful history information can be obtained using a BHL larger than the period of its dominant branch history...
patterns, all it may do is to incur more negative interference with other branch instructions, which could be detrimental to the overall misprediction rate.

In this paper, we try to take advantage of this observation by dynamically selecting and adjusting appropriate BHL to improve its misprediction rate. This scheme can be used in conjunction with any other branch prediction schemes which use global branch history, such as gshare and BiMode. Ideally, we would like to use a real-time Fourier analyzer to identify the most dominant global branch history patterns and select the appropriate BHL at runtime. However, real-time Fourier analyzer is very expensive and time consuming. There is also an overhead associated with the changing of BHL dynamically because PHT will need a training period every time BHL is changed. During this training period, the misprediction rate will increase substantially. Hence, even though we would like to change BHL as often as possible to dynamically adapt to the change of dominant branch history patterns, the associated overhead in training period keep us from adjusting BHL too frequently.

To facilitate our study, in the next section, we try to propose a scheme which can approximate Fourier analysis with a reasonable hardware cost and computation overhead. The approximation scheme works reasonably well in our study. However, an optimal design of such an approximation scheme is beyond the scope of our paper.
Figure 3: Three global branch histories governed by three single frequencies
3. An Approximation Scheme for the FFT Analysis

Fourier transform analysis is a well-known and extensively studied subject. Basically, it is simply an efficient numerical analysis method to study the periodic waveforms of a given signal. With a sampling interval $\Delta$, the maximum computed frequency is

$$f_{\text{max}} = \frac{1}{2\Delta} \quad (1)$$

For our study, it is obvious $\Delta=1$ (since each integer point on the horizontal axis in Fig. 1 and Fig. 3 corresponds to an executed branch instruction), and $f_{\text{max}} = 0.5$.

Despite its desirable quality and quantity of information, the real Fourier transform analysis is not recommended for our purpose because (i) all mathematical operations are of floating-point
type, which is very time consuming; (ii) It is very expensive to implement FFT in hardware; and (iii) The resolution we need here is very low, and we don't want to change BHL too frequently. The extra cost of generating exact power spectrum is thus unwarranted. The key idea here is to identify the dominant branch history pattern, and to determine its length (or its period) for BHL. What we need is an easy-to-compute and easy-to-implement approximation of the Fourier analysis with an acceptable accuracy.

3.1 An Approximate Algorithm

The scheme used in our study to approximate Fourier analysis is as follows:

1. We count the number of occurrences of consecutive 1's (taken) or 0's (no taken) in the branch history profile at run time. The length of consecutive 1's or 0's approximates the half period of the branch history patterns (or called the waveforms in FFT) with its corresponding frequency. The number of occurrences of such branch history patterns directly corresponds to the probability that such bit patterns occur in the branch profile. Let count(i) represent the number of occurrences of bit patterns with i consecutive bits of 1's or 0's. The set of such counts is called the raw power spectrum. Its mathematical expression is

   \[ \text{Raw}_\text{PS}(i) = \text{count}(i), \quad i = 1, 2, \ldots, 20 \]

2. In our study, the maximum half period to be monitored is up to 20. Hence it corresponds to the maximum period of 40.

3. We modify the amplitude of the raw power spectrum count(i) using its corresponding length i, as follows:

   \[ \text{Modified}_\text{PS}(i) = \text{count}(i) \times i, \quad i = 1, 2, \ldots, 20 \]
The reason for such a modification is to equalize the value of amplitude for different i's. It is because for a given interval of 400 branch instructions, the bit pattern with a period of 10 can have 40 of them at most in that interval, while the bit patterns with a period of 20 can only have 20 of them at most in the same interval. Hence, even though in both cases, they all should have the same probability of 100%, the amplitude for bit patterns (which corresponds to the probability of their occurring) with the period of 20, i.e. $\text{Raw}_PS(20)$, is less than $\text{Raw}_PS(10)$ because of its longer period. By multiplying $\text{Raw}_PS(i)$ with its length $i$, both $\text{Modified}_PS(10)$ and $\text{Modified}_PS(20)$ will have the same amplitude of 400.

(3) The modified power spectrum can be further improved by including some empirical coefficients, $\text{coef}(i)$, in the final power spectrum,

$$\text{Final}_PS(i) = \text{count}(i) \times i \times \text{coef}(i), \quad i = 1, 2, \ldots, 20$$

(4)

The coefficients are used to compensate for the fact that, using our approximation scheme, the bit patterns with a longer period has a relatively higher probability to be disrupted and being grouped into bit patterns with shorter periods. Consequently, the occurrences of the bit patterns with smaller periods will be more frequent. We can used $\text{coef}(i)$ to compensate for such a tendency. By combining the length $i$ and the $\text{coef}(i)$, we can consider their product, $i \times \text{coef}(i)$, as a weight $w(i)$ for $\text{Raw}_PS(i)$, i.e.

$$w(i) = i \times \text{coef}(i), \quad i = 1, 2, \ldots, 20$$

(5)

These weights can act like a filter as in signal processing to filter out unnecessary noises. It is quite difficult to obtain these weights analytically. Through simulations, we found the best weights in our study are those shown in Figure 5. Also, to avoid the multiply operations needed in applying the weights, we approximate each weight to its nearest $2^n$. We can thus use shift
operations instead of multiplications to apply the weights. We found there is no observable
differences using such an approximation.

3.2 A Possible Hardware Implementation

Figure 6 shows a possible hardware implementation for the scheme proposed in section 3.1. There are two counters for the global branch history profile. One counts the length of consecutive 1's in the profile, while the other counts the length of consecutive 0's. The counters need to be reset when the bit string changes from 1 to 0, or from 0 to 1. In this study, we use a total of twenty counters (marked by length 1 counter,..., length n counter in Figure 10) to count the occurrences of its corresponding bit string length of i. For example, length 3 counter counts the occurrences of 000 and 111 bit strings. In addition to those counters, there are twenty coefficient shifters that perform the shift operations and transform the Raw_PS into Final_PS. Finally, the maximum amplitude selector picks the highest amplitude among all Final_PS(i) to determine the next BHL.

As a simple example, consider a 60-bit global branch history profile:

110011001011011000111100000111111111000000111110011100110

The results obtained after each of the three steps described in Section 3.1 are shown in Figure 7 from top to bottom. Figure 7 (top) shows Raw_PS(i) after different bit patterns have been counted. Note that the first two bits 11 and the last bit 0 in the above profile are not included in the calculation because they are cut off from the previous bit stream and from the following bit stream, respectively. Figure 7(middle) shows Modified_PS(i). They are obtained by multiplying Raw_PS(i) with its corresponding length i. Figure 7(bottom) shows Final_PS(i). The weights used is shown as a dotted curve.
A FFT-Based Adaptive Branch Prediction Scheme

Figure 5: Distributions of weight coefficients for both multiplication and shift operations

Figure 6: Hardware design of the approximate Fourier analysis
Figure 7: The results from three procedures of the approximate algorithm

There are two details worth mentioning. Firstly, both amplitude values corresponding to the half wavelength of 6 and 9 in Modified_PS (Figure 7(middle)) are set to zero in Final_PS (Figure 7(bottom)) because of their bad repeatability (they only appears once in the profile). Secondly, if there are two equally strong bit patterns, e.g. both peaks with half wavelength of 2 and 3 in Figure 7(middle), we should choose one with the longer wavelength, i.e. 3. It is because a longer BHL obviously covers a shorter BHL. From Figure 7(bottom), we should set the BHL to 10 bits, which is twice the half wavelength of 5.

3.3 Length of Interval to Adjust BHL

To sample data and to change BHL, we need to determine the length of the interval between we check and adjust BHL’s. The appropriate length of interval is determined by several factors.
First of all, to match more closely to the period of the dominant branch history patterns, we would like to check and change BHL as often as possible. However, in each new interval in which the BHL is changed, the misprediction rate is usually pretty high because it takes time to update the branch prediction information in PHT. The duration of this training period depends on the size of the PHT and BHL. The larger the PHT size and the larger the difference between two BHL’s the longer the training time requires. Also, from our measurements, the set of branch instructions executed in each interval tend to fluctuate less significantly when the interval is longer. In one obvious extreme, if the interval is the entire program execution time, it will include all of the branch instructions and the set remains unchanged. Hence, if we use the branch history profile in one interval to predict the branch behavior of the next interval, the more similar the set of branch instructions are, the more accurate the prediction will be. Hence, we would like to set the interval as long as possible. To compromise these contradicting objectives, our experiments show that the interval of 500K branch instructions is appropriate for most of the programs. It is beyond the scope of this paper to find an optimal length, or to find a scheme to dynamically adjust the interval length for optimal results. However, these are interesting issues worth looking into further.

Also, to avoid the increase in misprediction rate due to the change of BHL, we use a 2-bit saturating counter to determine whether we should change BHL. The counter is incremented when the total misprediction number in the current interval is greater than that in the previous interval, and is decremented if it is smaller. BHL will be changed only when the saturating counter is incremented twice in a roll.

4. Experiments

4.1 Simulation Methodology

We use SimpleScalar/Alpha 3.0 tool set [13] to simulate our scheme. We add both our adaptive
scheme and the BiMode scheme [12] (in addition to the default gshare scheme used in the tool set), and use SPECint95 for our simulation (To shorten our simulation time, compress, perl and vortex are terminated after 500M instructions). Some of the program characteristics relevant to our study are listed in Table 1.

Table 1. SPECint 95 benchmark programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Input</th>
<th># of Instructions</th>
<th># of Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>bigtest.in</td>
<td>500 M</td>
<td>98 M</td>
</tr>
<tr>
<td>gcc</td>
<td>cccp.i</td>
<td>264 M</td>
<td>54 M</td>
</tr>
<tr>
<td>go</td>
<td>50 9 2stone9.in</td>
<td>548 M</td>
<td>80 M</td>
</tr>
<tr>
<td>jpeg</td>
<td>specmun.ppm</td>
<td>553 M</td>
<td>51 M</td>
</tr>
<tr>
<td>li</td>
<td>train.lsp</td>
<td>183 M</td>
<td>41 M</td>
</tr>
<tr>
<td>m88ksim</td>
<td>dhry.big</td>
<td>492 M</td>
<td>113 M</td>
</tr>
<tr>
<td>perl</td>
<td>primes.pl</td>
<td>500 M</td>
<td>95 M</td>
</tr>
<tr>
<td>vortex</td>
<td>vortex.in</td>
<td>500 M</td>
<td>79 M</td>
</tr>
</tbody>
</table>

We compare the branch misprediction rate for the gshare scheme, the BiMode scheme and with and without our adaptive scheme. We vary the BHL and its corresponding PHT size. An index length of L means that the PHT will have $2^L$ entries. BiMode needs an extra level of table lookup. It has a "choice table" during the first table lookup and two direction tables in the second table lookup. That means the total table size of BiMode is three times as large as that of gshare if index length is the same.

4.2. Average Misprediction Rates

In Figures 8 and 9, we show the average misprediction rates of all eight SPECint95 programs for gshare and BiMode schemes with and without our selection scheme, respectively. In the figures, DS indicates our proposed scheme which can select history pattern length dynamically using the approximate Fourier analysis method described in Section 3. The results show our dynamic
selection scheme indeed can further improve both gshare and BiMode schemes. The greater improvement is generally seen for the smaller PHT sizes.

Figure 10 shows the misprediction rates for go, gcc and average of all the SPECint95 benchmark programs. We use 12-bit index length (4K total table entries) for gshare schemes and 11-bit index length (6K total table entries) for BiMode schemes. Gshare-best indicates the best misprediction rate among all possible BHL’s for gshare. The BHL is unchanged throughout the entire program execution [15]. Average misprediction rates are 9.8%, 8.4% and 7.9% for gshare, gshare-DS and gshare-best, respectively. BiMode and BiMode-DS shows 7.5% and 7.4%, respectively. Some of the most difficult programs in SPECint95 for most branch prediction schemes are programs such as go and gcc. These programs usually have very erratic branch history patterns. Our dynamic selection scheme is especially effective for those programs. In the case of go, the misprediction rates of gshare is 26.7% but the gshare-DS is 19.5%, which is even lower than BiMode.

Figure 11 shows that the misprediction rates using approximation of the shift operations are almost the same as those using multiplication operations with a less then 0.01% difference.

![Average Misprediction Rates for Gshare Schemes](image.png)

Figure 8: Average misprediction rates for gshare with and without dynamic selection of branch history pattern length.
A FFT-Based Adaptive Branch Prediction Scheme

Figure 9: Average misprediction rates for BiMode with and without dynamic selection of branch history pattern length.

Figure 10: Average misprediction rates for Gshare and BiMode shemes.

Figure 11: Average misprediction rates for gshare-DS Approximations
### 4.3 Context Switching Effect

We also look at the effect of the context switches in an environment in which interrupts occur frequently. Context switches may remove the branch history stored in PHT’s. Consequently, it needs a training period after each context switch. The performance of a history-based predictor could be reduced significantly if context switches happen frequently. However, our dynamic selection scheme can adjust BHL dynamically to accommodate the context switch effects.

The effect of context switches is simulated by periodically flushing the PHT as in [15]. All PHT entries are reinitialized to *saturated taken* each time when context switch occurs. Figure 12 shows the average misprediction rates correspond to different context switch *distances* (i.e., the number of branch instructions executed before a context switch takes place). Both predictors have the same index length of 12 bits. The proposed scheme performs much better than the original gshare and BiMode schemes. BiMode show worse performance than gshare in short context switch distance because it has more tables and larger table sizes, which require a longer training period.

![Average Misprediction Rates with Context Switch](image)

**Figure 12:** Average misprediction rates with context switch.

In Figure 13, we show the principle of branch prediction rates for the go. Each point in the figure corresponds to the average of branch prediction rate of 5000 consecutive branch instructions. The top of Figure 13 shows the profile of regular gshare scheme which is very
erratic. Using our dynamic selection scheme, the profile in the bottom of Figure 13 becomes significantly less erratic because our scheme allow BHL to be adapted to the branch behavior.

Figure 13: The hit rates (calculated for every 5000 branches) of gshare and gshare-DS for the program go. Hit rate = 1 – misprediction rate.

5. Conclusions

In this paper, we study the feasibility of using the concept of Fourier analysis to select appropriate branch history lengths during runtime to better adapt to the change of dominant branch history patterns. Our simulation results show that such schemes can improve several popular branch prediction schemes which use global branch history, such as gshare [5] and BiMode [12]. Our study also shows that such schemes are particularly effective when context switches (perhaps due to interrupts) occur frequently. We also look at schemes which can approximate Fourier analysis and reduce its computation overhead and hardware cost.

Overall, using the concept of Fourier analysis, it seems to be able to provide the theoretical foundation needed to guide the selection of branch history length dynamically to reduce misprediction rates. Further research into this area seems warranted.
A FFT-Based Adaptive Branch Prediction Scheme

Reference


