Non-Viable Path Branch Prediction

by Ying Xiong Soner Önder

Michigan Technological University

Computer Science Technical Report
CS-TR-07-02
March 6, 2007

Non-Viable Path Branch Prediction

Ying Xiong   Soner Önder
Department of Computer Science
Michigan Technological University
Houghton, MI 49931-1295

March 6, 2007

Abstract

For modern superscalar processors which implement deeper and wider pipelines, accurate branch prediction is crucial for feeding sufficient number of correct instructions into the superscalar's highly-parallelized execution core. In this paper, we show that what the branch predictor is learning has significant implications for its ability to make effective use of branch correlation and its ability to exploit longer history lengths. Contrary to the commonly employed approach of training each case using both successes and failures of previous branch behavior, a *Non-Viable Path* branch predictor learns those paths that the program will never follow, given prior history; such a predictor is referred to as *Non-Viable Path Predictor (NVPP)*. An NVPP can learn any boolean function, given enough space.

In this paper, we present an analysis of the design space and arrive at a practically feasible implementation of the concept which uses simple traditional one or two-bit tables and traditional table indexing mechanisms. Because of its unique properties, *NVPP* can also exploit very long global histories with only a linear increase in the space requirement and provide rapid training times. Although NVPP exploits large histories, its access time remains relatively constant and it is comparable to small Gshare predictors, permitting a single cycle implementation. The performance of *NVPP* is compared with most recently published highly accurate predictors including *piecewise linear* and the *O-GEometric History Length* predictors. Our results indicate that the *NVPP* accuracy compares well with these predictors while maintaining the advantages of simplicity and the speed of a traditional style predictor design.

1 Introduction

Modern deeply pipelined wide-issue superscalar processors call for accurate branch predictors. During the past two decades, the accuracy of branch predictors have improved significantly. Unfortunately, for deeply pipelined architectures the performance lost due to branch mispredictions is quite significant even with a highly accurate branch predictor.

Historically, dynamic prediction mechanisms have gained wide acceptance since its introduction with the early designs [18]. The introduction of a two-level adaptive scheme by Yeh and Patt in 1991 [21] greatly improved the accuracy of dynamic branch prediction due to its ability to exploit correlation among branch instructions. Following work concentrated on improving the basic two-level schemes and improved the accuracy of branch predictors by reducing the destructive aliasing [22, 19, 10, 3, 14], exploiting different properties [15], utilizing data values [5, 20] or employing hybrid mechanisms [11]. Relevant work in this regard is overwhelming, and is only partially covered in this paper. More recently, radically different approaches such as the perceptron predictor [8], as well as the prophet-critic approach [4] have been proposed. A recent publication, namely, *O-GEometric History Length (O-GEHL)* predictor [16] demonstrated very high accuracies by exploring geometrically increasing history lengths.

We would like to point out that the common approach in prior techniques is to learn from both successes and failures of previous predictions; to summarize the accumulated observations in some manner; and to use these summaries for future predictions. For example, a two-bit Smith predictor [18] learns if a branch is mostly taken or not taken and summarizes the observations in a saturating counter. A traditional correlating branch predictor separates the context that this phenomenon is occuring; for each history a separate counter is reserved either directly (two-level [21]) or indirectly (gshare [12]). As a result, given a history, the corresponding counter will saturate up or down based on the actual outcome of the branch. This approach is quite effective for smaller history lengths, but it cannot use longer histories because of the resulting exponential growth in the counter space as well as the growth in training time, as the training has to be performed sufficient number of times for each counter to capture the bias. From the same perspective, perceptron based predictors also generate summaries. However, since the generated summaries are weighted, they can overcome the difficulty of separating the context to a large extend and can exploit very long histories. On the other hand, the use of such weighted summary information requires adder networks, which may potentially increase the access time of the predictor.

Contrary to existing approaches, a non-viable path predictor learns the exceptional cases, i.e., only those paths that lead up to mispredictions. This is quite effective because such cases make-up a much smaller percentage of the total paths. Simply put, *NVPP* learns whether there is correlated behavior at a given point, since the number of cases where there is no correlation or the cases where the utilization of correlation is not essential for correct prediction is much larger. Clearly, given non-viable paths one can identify the viable ones, and vice versa. In this paper, we adopt the convention of referring to the non-viable paths. Any path we don't have information about is considered viable.

In the rest of the paper, first in Section 2 we discuss the idea of non-viable paths and its relationship to branch correlation. We discuss the effect of limited history length on the accuracy of predictions and present a conceptual level design. We then illustrate through experimental data that the idea is workable. In Section 3 we present a practical implementation of the non-viable path predictor which can exploit very long histories. We illustrate that by using the non-viable paths idea and well established techniques from the branch prediction repertoire we can maintain a design that would permit a single clock cycle implementation. In Section 4 we evaluate the performance of our predictor and compare it to other recently proposed predictor designs. In Section 5 we provide a brief summary of the most relevant work to ours and discuss various similarities among these techniques. Finally, in Section 6 we present our conclusions and future work.

2 Correlation and Non-Viable Paths

In order to see why using non-viable paths makes sense, let us consider a simple example where the outer loop executes a significant number of iterations and the inner loop initializes a given array:

```
for (int j=0; j < somebignumber; j++) {
...
for (int i=0; i<N; i++)
   A[i] = 0;
..  }
```

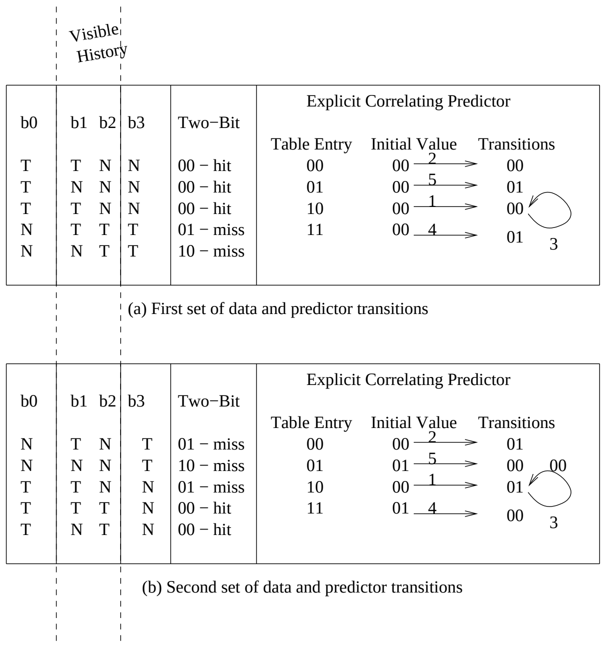
Considering the inner loop back-edge branch, a simple Smith predictor will correctly predict all but the last iteration of this loop. A two-level predictor can predict all instances of the branch if the global history is large enough to hold N branch histories. Unfortunately, in traditional approaches, the hardware budget doubles for every bit of additional global history. If the above inner loop is executing 50 times, in order to successfully predict all iterations, a traditional two-level predictor would need 256 Tera Bytes of space. Even when one does not consider the hardware budget involved, exploiting long histories in this manner causes the same static branch to occupy too many entries in the Pattern History Table (PHT). For a global history length of p, there will be 2^p possible entries corresponding to this branch. These counters should be trained for any combination of global histories leading upto the relevant branch. This increased training time is the lead cause of performance loss that is observed after a certain history length is reached [8].

2.1 Non-Viable Path Prediction

A non-viable path predictor is a simple combination of any local predictor (such as a two-bit saturating counter predictor) and a predictor that memorizes misprediction behavior of the first and correlates it to the observed global history. If the correlator does not know a given history, the local predictor's output is assumed. Observe that, in the above example, after one iteration of the inner loop the correlator learns that a history of N consecutive 1s (one coming from the outer loop back-edge and $N-1$ coming from the inner loop back-edge) results in a misprediction by the local predictor. As a result, it learns only *this history* which represents a *non-viable path* in the example. It can then present a *not-taken* output each time this history is observed. Note that, once this history is learned, the predictor will never misspredict. Also note that the training time of such a predictor is a function of the mispredictions experienced by the local predictor and not a function of the history length. Although we have used a simple loop example to demonstrate the problem, ability to correctly predict using failure histories is not limited to loop-back edges. In fact, when complex control flow is involved where only few histories result in mispredictions, a non-viable path predictor will learn only these histories and never mispredict whereas a traditional predictor has to learn all involved combinations until it can predict equally well.

2.2 Effect of Limited History Lengths

For global history based predictors the use of summarized information such as trained counters presents other problems as well, particularly when the examined history is not sufficient to capture the correlation (i.e., when the correlation is distant). Let us examine the sequence of branch outcomes shown in Figure 1.


Visible History
b0
b1 b2 b3
Two−Bit
Explicit Correlating Predictor
Table Entry
Initial Value
Transitions
(a) First set of data and predictor transitions
(b) Second set of data and predictor transitions

b0	b1	b2	b3	Two−Bit	Table Entry	Initial Value	Transitions
T	T	N	N	00 − hit	00	00 $\xrightarrow{2}$	00
T	N	N	N	00 − hit	01	00 $\xrightarrow{5}$	01
T	T	N	N	00 − hit	10	00 $\xrightarrow{1}$	00
N	T	T	T	01 − miss	11	00 $\xrightarrow{4}$	01
N	N	T	T	10 − miss			3

(a) First set of data and predictor transitions

b0	b1	b2	b3	Two−Bit	Table Entry	Initial Value	Transitions
N	T	N	T	01 − miss	00	00 $\xrightarrow{2}$	01
N	N	N	T	10 − miss	01	01 $\xrightarrow{5}$	00 00
T	T	N	N	01 − miss	10	00 $\xrightarrow{1}$	01
T	T	T	N	00 − hit	11	01 $\xrightarrow{4}$	00
T	N	T	N	00 − hit			3

(b) Second set of data and predictor transitions

Figure 1: Limited History and Explicit Correlation

Considering the first set of data shown in Figure 1(a) branches b2 and b3 appear to be correlated, whereas the actual correlation is between b0 and b3. Since b0 is outside the history window, this fact cannot be observed and learned by a correlating branch predictor. As it can be seen, a two bit saturating counter predictor predicts the first three instances of b3 correctly, but fails to predict the last two. Similarly, a simple correlating branch predictor that allocates a separate counter for each branch experiences difficulty training its counters. In the example, using the outcomes of b1 and b2 as the contents of the global history shift register, table indexes 01 and 11 start changing the bias towards taken. Unfortunately, when the second wave of data shown in 1(b) is processed, the bias of 00 and 10 needs to be changed. If the illustrated data sequences are observed in that succession repeatedly, the correlating predictor can never train its counters and leaves them in a lingering state. Even though the above sequence is a synthetic sequence to illustrate the point, examining counter values in a gshare predictor [12] with small history lengths indicate that the phenomenon is quite frequent.

A non-viable path predictor on the other hand would learn two non-viable paths, namely, a given history that is followed by a *taken* as well as the same history that is followed by a *not taken*, both as non-viable paths. This indicates that the history that is being examined is not sufficient to capture the correlation, or, simply, there is no correlation that could be exploited. In other words, *a non-viable path predictor will know when it does not know the answer*.

In summary, traditional means of exploiting correlation by using counters fails on a number of fundamental issues:

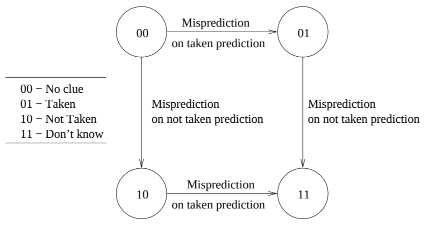
1. It has to see the same positive (negative) example repeatedly before it can train the counters for correct predicton.
2. The counter's information content is weak. It only illustrates the bias of the particular branch instruction given a history of predictions.
3. The predictor does not know when its predictions are flawed. In other words, the predictor cannot tell when the correlation exists and and when it no longer holds. Fundamentally, this is the main reason behind the success of hybrid predictors where a separate confidence predictor learns when a particular predictor's predictions are flawed.
4. The effects of using one counter for another branch is usually destructive. The predictor is prone to destructive aliasing.

In the following sections, we will iteratively develop a practical non-viable path predictor (*NVPP*). Doing so, we will employ well established techniques commonly used in the design of branch predictors and demonstrate that the above problems can naturally be eliminated when one uses the concept of non-viable paths to design the predictor.

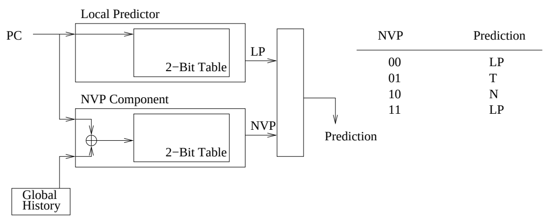
2.3 Conceptual Level Design

As indicated before, contrary to a counter based correlating predictor, a non-viable path predictor learns only the exceptional cases. To illustrate the point, we will take a table of two-bit entries indexed by a global history register similar to a counter based correlation predictor. We will however interpret the bits of the entry differently as shown in Figure 2.3 (a).

The interpretation of the table entries are as follows. *No clue* means the predictor has not seen this history before. As long as there is no misprediction at *this history*, leading upto *this branch*, the entry will maintain *no clue* status. If a misprediction is observed at *this* history leading upto *this branch*, the incorrect (correct) prediction is recorded in here (corresponding to the taken and not taken cases). In other words, the predictor learns that the history value indexing the table entry is a *non-viable* prefix for this branch and the predictor should never (always) predict the value indicated in the table when the entry is


00
01
10
11
Misprediction on taken prediction
Misprediction on not taken prediction
Misprediction on not taken prediction
Misprediction on taken prediction
00 – No clue
01 – Taken
10 – Not Taken
11 – Don't know

(a) NVPP State Transitions



NVP	Prediction
00	LP
01	T
10	N
11	LP

(b) Basic NVP Predictor

Figure 2: Non Viable Path Prediction

either taken or not-taken. The *don't know* value indicates that the predictor have seen this branch both taken and not taken at this particular history. It means that the amount of history that is used is not sufficient to distinguish the non-viable (viable) paths. In this design, the local predictor can be any local predictor. For simplicity, we will consider it to be a simple two-bit Smith predictor. It is interesting to note that the NVP predictor shown acts very much like a confidence predictor.

With the above design, our example sequence yields interesting results. Consider the example sequences shown in Figure 1(a) and (b) again. The first three probes for branch b3 yields success as the two-bit predictor correctly predicts them. For these probes, the NVP component will not be updated, as no mispredictions have been recorded. The remaining probes b1=T b2=T and b1=N b2=T will both cause the local predictor to provide the prediction and will result in mispredictions. The corresponding counters in NVP component will switch to predict taken. If the set of data in (a) is seen again without a visit to (b) all branches will be correctly predicted the second time. In this case, b2 is indeed correlated with b3 and the predictor has learned this correlation with only one misprediction. If however, the second set of data (b) is observed, table entries for 01, 10, and 11 will all switch to *don't know* status. In this case, the predictor will provide the result of the local prediction. The table entries will stay in *don't know* status until they are reset by another means. Even though this appears to be less than desirable, knowing that the provided history is not long enough to distinguish paths is extremely valuable. In the next section, we show how this information can be used to exploit very large global histories.

Note that we have addressed all the concerns about the counter based correlation predictor, even though we haven't built a predictor that can handle the example case successfully:

1. NVPP needs one example to learn the possibility of a correlation.
2. The infomation content of the counter is high.
3. The confidence in the result is built in. When it provides the prediction (i.e., 01,10 cases), it is confident. When it does not know, it knows that it does not know.
4. The entry values 00 and 11 are benign. In case of aliasing of these entries, the predictor won't perform any worse than the local predictor.

As previously stated, the basic predictor could be any one which uses only local information to make the prediction. In fact, the local predictor can be a constant valued predictor such as *predict taken* or compiler encoded, such as *forward not taken*, *backward taken* and NVPP still performs quite well. In these cases however, more misprediction information need to be learned by the NVP component.

In our implementation, we chose to use the simple two-bit saturating counter predictor [18] as our base predictor. Using a two-bit saturating counter predictor we observed that the accuracy of both the NVP component and the two-bit predictor become extremely high. In fact, the accuracy of the NVP component is high across all benchmarks. When NVP provides a prediction, it is almost always correct (over 99%). The reason behind the increase in the accuracy of the two-bit predictor is the filtering effect provided by the NVP predictor. The predicted branches by NVP are those which are hard to predict but can be predicted well by using proper correlation information. Obviously, we do not update the two-bit predictor when the prediction is provided by the NVP predictor. This way, those instances of branches which are prone to be mispredicted by the base predictor are effectively screened out, whether these branches require correlation to correctly predict, or, they are mispredicted because of ill effects of aliasing. As a result, the base predictor will not be polluted by them.

2.4 Observation and Analysis

By using a straightforward implementation of the NVP predictor at various history lengths, from 8 to 128 bits, we analyzed the behavior of the NVP module and the base predictor. We classified all the branch instances into three categories: those predicted by the base predictor when the NVP module produced a *no clue* (i.e., 00), those predicted by the base predictor when the NVP module produced a *don't know* (i.e., 11), and those predicted by the NVP module when the NVP returns a 01 or 10. In each subcategory, branch instances were further subcategorized into two parts: those are correctly predicted and those are mispredicted. The results are shown in Figure 3. In the figure, '00_hits','00_misses','nvp_hits','nvp_misses','11_hits' and '11_misses' represent the fraction of branch instances that are correctly and incorrectly predicted when the NVP output is 00, 01 or 10, and 11 respectively. The experiment is based on the average over all benchmarks that we use in this paper, as will be illustrated in more detail in later sections. As it can be easily seen the accuracy of both the NVP and the base component increase as the history length is increased. This is because more and more of difficult to predict branches are correctly predicted by the NVP module which filters them out from the base predictor, in turn increasing its accuracy. Branch instances that are falling into the case when the NVP module outputs *don't know* are intrinsically hard to predict ones. As expected, the two-bit predictor's prediction for these branches is not accurate.

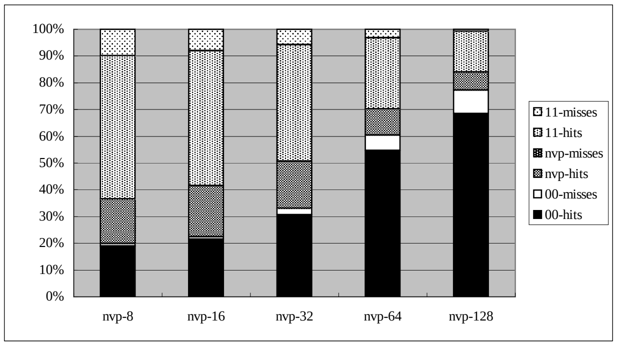


Figure 3: NVP Coverage Experimental Results

So far, we have not shown how having more information content on the NVP predictor entries would be helpful. After all, if a program is executed for a sufficient number of branch instructions, it is possible that eventually each NVP entry will become saturated (i.e., *don't know* case) because of aliasing. The NVP predictor would stop providing predictions and the accuracy of the overall prediction will go down to that of the base predictor. Although periodic clearing of the entries can prevent this problem, the intent is not to destroy this useful information, but to use it to our advantage. Going back to our motivating example, we see that it can be predicted well if we had used a longer history. But a naive approach would exponentially increase the space requirement as well as the training time. In the next section, we show how the additional information content help us exploit longer history lengths with a linear increase in space requirement.

3 A Practical Implementation

3.1 Exploiting Longer and Variable Length Branch Histories

In order to exploit a longer history with only a linear increase in space requirement, we use *segmented histories* [13]. The segmented history technique divides the branch history register to segments and uses the fragments of the history as input to individual predictors as shown in Figure 4.

There are several good reasons behind this choice: (a) By using segmented histories, a predictor can record long non-viable path information with a linear increase in the space requirement. (b) By using the *don't know* information recorded by the *NVPP*, we can exploit variable length histories. (c) With longer histories, the access time of the predictor remains relatively constant. This is because all the tables in the predictor are accessed in parallel and the access time will typically be dominated by the access time of the tables.

Of course, the technique will also bring in the problem of aliasing in the tables; the space provided by two segments of 16 bits ($2 * 2^{16}$ entries) cannot record all the combinations of 32 bits of history amounting to 2^{32} entries. However, the *don't know* case helps us distinguish the aliasing in the table entries as well since the destructive aliasing rapidly turns table entries into *don't know* status.

The operation of the predictor is as follows. On a query, the predictor selects the output of the first predictor table that is not a *don't know*, examining tables from the least significant position towards the most significant position. As a result, one can also exploit variable length histories by using a simple policy. A misprediction is first recorded on the least significant table using only a history length of one segment. If the length of one segment is sufficient to capture the correlation, then the entry will provide correct predictions as long as there is no aliasing. If the history length is not sufficient, or, there is destructive aliasing, the corresponding entry will become *don't know*. A second misprediction will record it in the next table, since the entry in the first is in *don't know* status, in effect utilizing twice the amount of history. This process effectively uses only the necessary amount of history to distinguish and exploit the correlation. When the last segment entry reports *don't know*, all entries are in *don't know* status. In

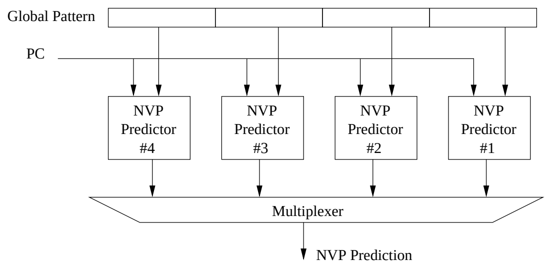


Figure 4: History Segmented NVPP

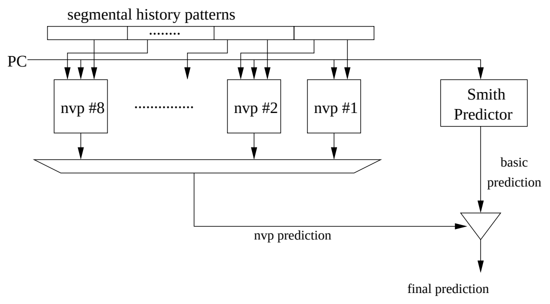


Figure 5: NVPP with History Distribution

this case, we assume that the event is because of aliasing and reset all segment entries to *no clue* and start retraining.

Returning back to our motivating example, if we provide two NVP predictors and use two bit history segments, after sequences (a) and (b) are observed once the first NVP predictor will switch to *don't know* status. The second NVP predictor however will start providing perfect predictions as long as the corresponding entry can stay in the second NVP predictor.

Note that the distant correlation among branches is sparse. Most correlation occurs within the vicinity of the branch which is typically captured by the first predictor. Our experiments indicate that the bulk of the NVP predictions are provided by the first predictor, whereas the additional predictors increasingly improve the performance by handling difficult to predict branches quite well using long histories as long as these entries can remain in the tables. Unfortunately, there are many distinct histories which share the same bit pattern as a substring. This results in the loss of misprediction information from higher-level tables. In order to alleviate this problem, we exclusive or the history segments. In other words, the least significant segment that is input to predictor #1 is also input the second predictor and exclusive or'ed with the second segment. The process is repeated by exclusive or'ing the second segment with the third before it is applied to the third predictor and so on. This approach significantly improves the survival rates of correlation information when parts of a complete history is destroyed due to aliasing some fragments still remain and continue to provide highly accurate predictions. We call this approach *history distribution*. With these changes, the global organization of the predictor becomes the one shown in Figure 5.

In this paper, we studied segment sizes up to 21 bits and number of predictors up to 8, ranging from 4KB of NVP space to 1MB. As a result, we have collected data up to 152 bits of history (8 tables, 19 bit segment size). Utilization of such large histories by predictors such as gshare is simply not possible. On the other hand, **the access time of NVP will be comparable to the access time of a gshare predictor where the global history length of the gshare predictor is equal to the segment size**. Despite this positive picture, one cannot carry the idea of segmentation too far. In other words, there is a minimum segment size below which destructive aliasing becomes quite significant. Experimentation shows that a segment size of at least 12 bits or more is needed. Above this treshold, *NVPP* provides excellent predictor performance, yielding not only very accurate but also a very fast predictor.

The design space of history segmentation is large and it is beyond the scope of this paper. In this paper, we assume that all the segments are of equal length and change the number of segments, given the segment size. It would be interesting to explore the design space where the segments are of different lengths. We leave this part of the design space as future work.

3.2 Handling Cross Interference

So far, we have ignored a significant problem, not only for NVPP but in general a problem for any global history based technique; namely the cross interference between different histories leading to different branch instructions. Simply put, throughout the execution, programs generate history patterns which are identical, but they lead to different branches. Using the information collected for one particular branch history path leading to a particular branch instruction for another branch instruction is detrimental. This is particularly true for the non-viable path predictor. Our solution builds on prior art as well as reasoning using simple set theory.

The set theory comes into the picture because given a particular history there are a number of branches which have observed that the base predictor had a misprediction. Alternatively, one could envision all the histories a particular branch instruction have experienced a misprediction. We find that the former is substantially smaller than the latter. Therefore, we choose to split the individual NVP predictor into two tables as in YAGS [22, 3] so that the cases *01* and *10* are recorded in separate tables. One can

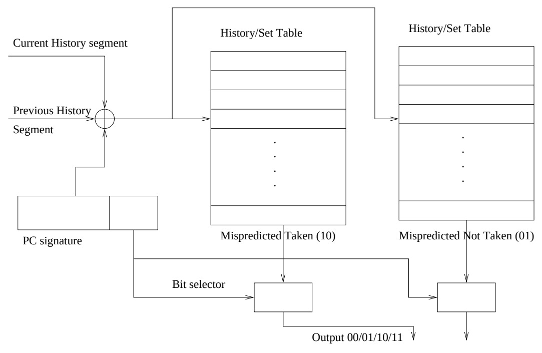


Figure 6: Structure of individual NVP predictor

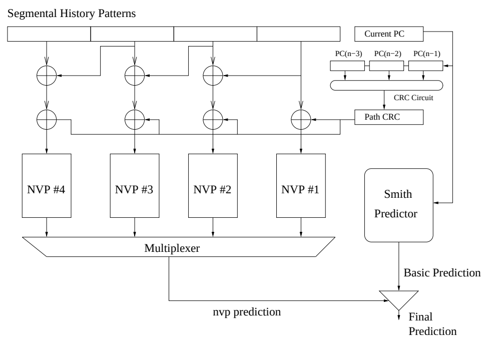


Figure 7: NVPP with History Distribution and Path CRC Information

implement the set concept in one of two ways. The *PC signature* can be concatenated to the history, which in-effect provides a distinct position for each branch at a particular history, or, one can design a predictor where each entry is a bit vector indexed by a *PC signature* indicating set membership. In other words, if a given PC signature s we have observed a misprediction at the history segment h, we set the corresponding entry by calculating :

```
table[h] = table[h] | (1 << s)
```

In other words, instead of merely storing a yes/no information in each entry, we make each entry a bit set such that it contains a 2^i bit pattern. Taking 4 bits of PC signature requires 2 bytes of information on each table since for a 4-bit PC signature, there are sixteen possible patterns: $0X0000, 0X0001, \ldots, 0XFFFF$. For each reference in the table, the corresponding PC signature is used to set or test the corresponding bit.

Implementing the set concept has proved to be invaluable in the success of the predictor. It significantly improves the accuracy of the NVP component as well as increasing the coverage. The coverage is increased because destructive aliasing in NVP tends to convert entries to *don't know*, which in turn reduces the covered cases. Ideally, the PC signature uniquely identifies this branch instruction. However, reducing the space requirement forces us to use only a small number of bits from the PC signature. We therefore choose to exclusive or the remainder of the PC signature with the history index to distribute the entries. This technique works extremely well. With these changes, an individual NVP component is illustrated in Figure 6.

A simple way to compute the PC signature is to use the least significant bits of the PC for this purpose. However, the sparseness of the field demands much bigger hardware budgets than needed. Furthermore, the program can reach a branch PC that has the same computed signature with the same history pattern, but the two branches can be completely unrelated. We therefore incorporate the path information computed from branch instruction PCs by using two techniques for both of which we present results. The first technique continuously computes the exclusive-or of the last 3 branch PCs and use it as the current PC signature. We then select the low order bits of the signature to index the set bit and high order bits as part of the hashing with the global history. Alternatively, we compute CRC7 of the least significant 7 bits of last 3 branch instruction PCs. This implementation is shown in Figure 7. Note that the current PC is not used in the computation of the CRC that is used in the current access but used to compute the CRC that will be used in the next access. As a result, CRC computation is not on the critical path of the predictor and the access time of the predictor is increased by only one gate delay compared to the case without the CRC. In a single cycle implementation of the NVPP, the whole cycle is available to compute the crc to be used in the next access by using the current PC and two prior PCs.

4 Experimental Evaluation

We have simulated the two final designs of the *NVPP* for a collection of Spec2000 benchmarks which includes 8 integer and 7 floating point benchmarks. Benchmarks written in C++ or Fortran 90 are excluded as currently we do not have cross-compilers to compile them to MIPS code. Benchmarks 253,254,255,168,188 and 200 are excluded because they can't be simulated correctly in our test bed. We will incorporate these benchmarks in the final version of the paper. Benchmarks are all compiled with gcc 3.2.2 with -O3 optimization flags. All benchmarks were run with data set *ref* after skipping the first half billion instructions and collecting statistics for 1 billion instructions. A random selection of few benchmarks (164.gzip and 176.gcc) which were executed to completion indicate some minor change in the performance of the predictors with respect to the data collected with 1 billion instructions, but the relative performance of the five predictors studied remains constant.

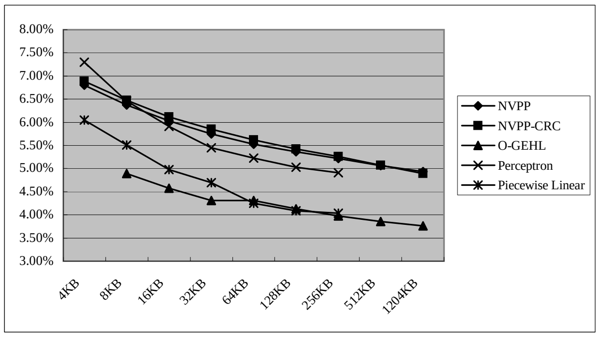
The prediction performance of the *NVPP* predictor has been explored by varying the total predictor space from 4KB to 1MB. For a given area size, the number of predictor tables (2,4 and 8 tables) and the number of set elements (1, 2, 4, 8, and 16 elements) have been varied and a large enough segment size have been used to utilize the given space. In addition to *NVPP*, we have implemented and evaluated three recently proposed predictors, namely, *perceptron*, *piecewise linear*, and *O-GEHL* predictors in the same environment. All of these predictors attain very high branch prediction accuracy by exploiting very long branch histories. *Piecewise linear* and *perceptron* predictors have not been evaluated beyond 256KBs since we did not have the optimal settings beyond 256KB for these predictors. *O-GEHL* predictor have been evaluated using the optimal settings used in the branch prediction contest and the tables have been enlarged to utilize the additional space.

Figure 8 illustrates the performance of the predictors at various hardware budgets. As it can be seen from these graphs, both *Piecewise linear* predictor and the *O-GEHL* predictor perform quite well with integer benchmarks with *NVPP* experiencing little over 1 percent more mispredictions consistently over the budget range. On the other hand, with the floating point benchmarks *Piecewise linear* experience little over 3 percent mispredictions compared to both *NVPP* and *O-GEHL*. When the entire set of programs which were studied is taken into account we observe that *NVPP* experiences around 1 percent more mispredictions than *O-GEHL* across the budget range while *Piecewise linear* experiences more mispredictions compared to both of these predictors.

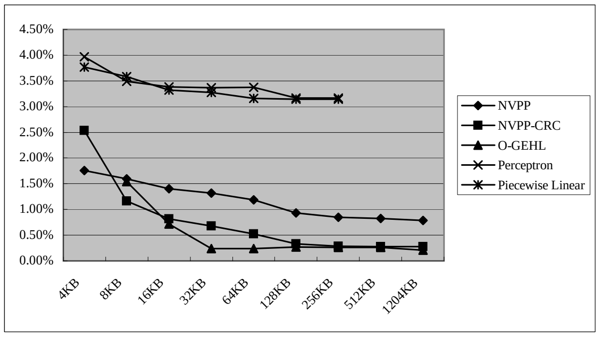
Few branch prediction papers in the literature studied the misprediction behavior of floating point benchmarks, and mainly presented results for integer benchmarks. This perspective is based on the commonly held belief that floating point benchmarks have negligible amount of branch mispredictions. Our studies indicate otherwise. These benchmarks have significant amount of branch mispredictions resulting from a large number of nested loops where the inner-most loop executes a non-negligible amount of times, as well as mispredictions resulting from complex control flow, just like the integer benchmarks.

The success of *NVPP* with floating point benchmarks is possibly related to its ability to exploit long histories resulting both from nested loops as well as complex control flow.

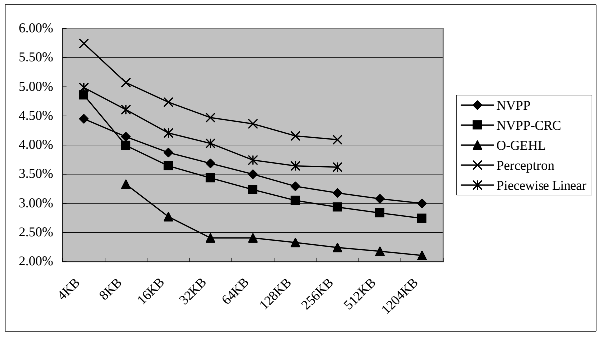
We illustrate the individual benchmarks at a hardware budget of 256KB by selecting their best performing configuration in Figure 9. As it can be seen, some benchmarks where *NVPP* indicates inferior performance such as *256.bzip2* and *164.gzip* share common properties. In



(a) Spec00 Int Arithmetic Mean



(b) Spec00 FP Arithmetic Mean



(c) Spec00 Overall Arithmetic Mean

Figure 8: Misprediction Rates

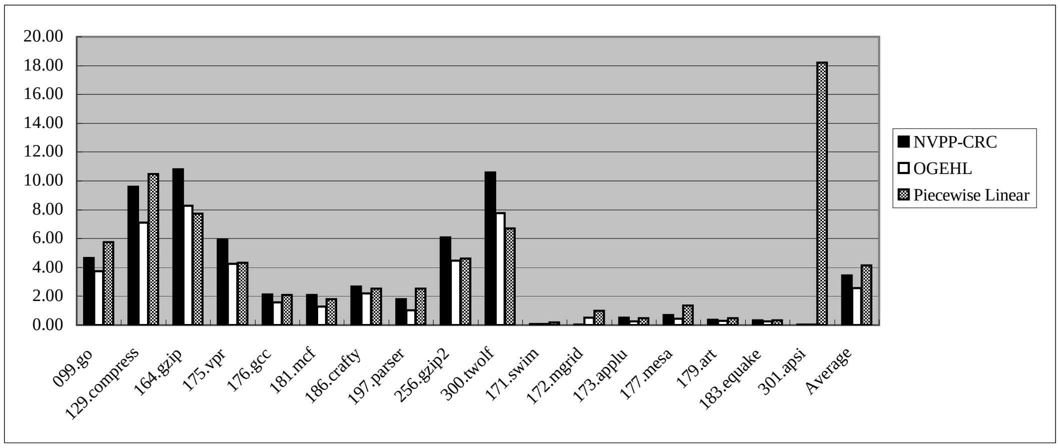


Figure 9: Mispredictions at 256K Hardware Budget Individual Benchmarks

general, these benchmarks have a small branch footprint, but the individual branches are executed many times, mostly in a data dependent fashion. The current implementation of *NVPP* treats all mispredictions as originating from the use of inadequate history, and attempts to use more and more history by shifting the predictions to higher-order tables. This is not particularly good, since the predictor goes through the cycles of turning all tables to *don't know*, resetting them and starting over. In such cases, every prediction by NVP component is a misprediction, and it is likely that the local predictor would have done better. This is an area where we are working to incorporate ability to NVPP so that it can detect why NVP provided prediction results in a misprediction and act accordingly.

Our results compare well with previously published data on the performance of perceptron based predictors with the integer benchmarks. However, in general, it appears that the perceptron-based predictors do not perform as well with the floating point benchmarks and since the previous perceptron based branch predictor papers we have found in the literature did not incorporate results of floating point benchmarks, we cannot cross-check our results with the previously published work in case of floating point benchmarks. Among the floating point benchmarks, the performance of *Piecewise linear* predictor on 301.apsi is notably poor. We have investigated the reasons and found out that in the execution of 301.apsi, the function sqrt() is called extensively. Two branches in this function account for 72.71% of total dynamic branch instances for the sampled duration (the first billion instructions after half a billion instructions are skipped). The code involved and the two branch instructions are shown below.

```
for (i = 1; i <= 51; i++)
  {
    t = s + 1;
    x *= 4;
    r /= 2;
    if (t <= x)
       {
       s = t + t + 2, x -= t;
       q += r;
       }
    else
       s *= 2;
  }
```

This for loop accounts for 36.71% of total branch instances and the prediction accuracy of this branch by the piece-wise-linear predictor is 98.08%. The if statement inside the loop accounts for 36.00% of total branch instances and its accuracy is only 51.50%. The corresponding values for the NVPP are 99.60% and 88.10%, respectively. Although the if statement in the above example is data value dependent and would not typically permit better prediction through correlation, it shows an interesting behavior with very long histories. According to our findings, each 51 groups of history repeats every 112 patterns as shown below.

```
( 1 )110011011100110100001010001000000101011100100100110
( 2 )010000001100100100101100000111011110101000100000110 1
     ...............................................
(112)000110100000001111000110000100111101111111010101000 1

(113)110011011100110100001010001000000101011100100100110
(114)010000001100100100101100000111011110101000100000110 1
     ...............................................
(224)000110100000001111000110000100111101111111010101000 1
```

Obviously, with its explicit history recording, NVPP is able to capture this long correlation, but the perceptron based predictors are unable to capture it in their weighted summaries. O-GEHL also successfully captures the behavior with its geometrically increasing history sizes.

We have also studied the sensitivity of the performance of *NVPP* with respect to number of tables. Figure 10

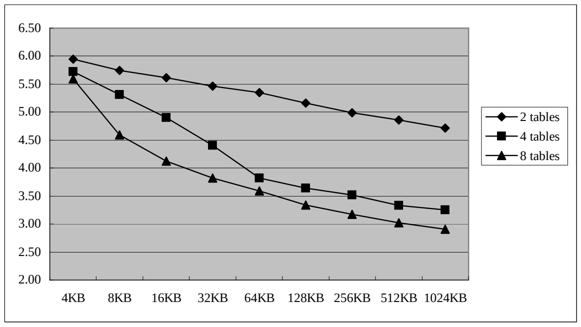


Figure 10: Sensitivity to number of tables

shows that the segmentation technique works well as a table count of 8 yields the lowest number of mispredictions at a given hardware budget.

5 Related work

A large number of dynamic branch prediction techniques have been developed since the introduction of the two bit counter based Smith predictor [18] in 1981. In 1991, Yeh and Patt introduced the Two-Level Adaptive Branch Predictor [21, 22]. McFarling proposed the Gshare predictor [12] in 1993. These predictors shared the common property of exploiting correlation among branches.

Although in principle one can view NVPP as a hybrid predictor [2, 1], NVPP embodies neither an explicit confidence predictor nor a selector. Instead, the global predictor serves both functions. An overriding predictor [7] design attempts the combine the accuracy of a slow but accurate predictor with the speed of a simpler predictor by employing them in sequence in a pipelined design where the decisions of the simpler fast predictor can be overridden by the more accurate slow predictor. This is a general concept that is orthogonal to NVPP. Since NVPP is a highly accurate predictor that can permit a single clock cycle implementation it can be used together with a slower but better predictor (such as O-GEHL) in an overriding pipeline design to further improve performance.

Branch confidence estimation [6] is a speculation control technique which evaluates the confidence level of current speculation status so that actions can be taken on low-confidence speculations to avoid misspeculations. Selective Branch Inversion [9] takes the idea of branch confidence estimation and selectively inverts the predictions when the confidence of a prediction is considered to be low. As opposed to most confidence predictors, NVPP confidence is binary valued, i.e., either there is a known failure history at a given path and the global predictor predicts the other path, or the global component does not know the answer and assumes the local predictor's output. Contrary to SBI approach, local predictor's predictions are not inverted.

In order to explore very long history lengths while keeping the hardware budget at reasonable size, NVPP uses segmented histories. Segmented histories have been employed by other predictors in the past, notably, the PPM-like tag-based predictor [13] as well as O-GEHL indexing mechanism. However, these mechanisms typically employ increasing segment lengths and in case of PPM-like tag based predictor additional tags for *gluing* different segments together. In case of NVPP, each individual bank explores a specific segment/part of the global history.

Neural-based branch predictors has been proposed by Jimenez. Perceptron predictor [8] achieves high accuracies by exploring very long global histories while requiring modest hardware sizes. Piecewise linear predictor addresses a short-coming of perceptron based predictors by enhancing the functions that perceptrons can learn. O-GEHL predictor [17] proposed by Seznec gains high accuracy by implementing dynamic history length fitting based on geometric histories and the use of a perceptron style adder tree. This adder tree is on the critical path and as a result may not permit efficient single cycle implementations of these predictors.

Path based branch prediction mechanisms have been introduced initially by Nair [15] and similar mechanisms have been incorporated into various predictors including the most recent O-GEHL predictor. NVPP is no exception since it also greatly benefits from the inclusion of the path information in the indexing functions. We believe there are a number of properties of NVPP which sets it apart from other predictors, namely, its unique encoding mechanism which permits the global predictor to serve as a confidence predictor as well as a selector, and the recording of only the wrong path information. Since misprediction is a rare event in the execution, the information that needs to be stored is considerably smaller than others.

6 Conclustion and Future Work

In this paper, we have proposed the concept of *Non-Viable Paths* and described a possible application utilizing this concept – the *NVPP* predictor, a new global two-level branch prediction mechanism which could be used along with other dynamic branch predictors to improve their performance.

The main strength of NVPP is its access time. NVPP access time is comparable to a Gshare predictor which has the same size as one of the tables of NVPP and hence it may permit a single clock cycle implementation. On the other hand, both perceptron based and O-GEHL predictors incorporate adder networks on the critical path of the branch prediction and their implementation will likely require pipelining of accesses. Such pipelining is not trivial and potentially may impact their accuracy. Although the integer performance of NVPP is lower than that of these cutting edge branch predictors, a fair comparison between these predictors should involve estimated timing information, which we leave as future work.

We believe the concept will prove to be quite useful in predicting multiple branches per cycle, which we also leave as future work.

References

[1] Po-Ying Chang, Eric Hao, and Yale N. Patt. Alternative implementations of hybrid branch predictors. In *MICRO 28: Proceedings of the 28th annual international symposium on Microarchitecture*, pages 252–257, Los Alamitos, CA, USA, 1995. IEEE Computer Society Press.

[2] Po-Yung Chang, Eric Hao, Tse-Yu Yeh, and Yale Patt. Branch classification: a new mechanism for improving branch predictor performance. In *MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture*, pages 22–31, New York, NY, USA, 1994. ACM Press.

[3] A. N. Eden and T. Mudge. The yags branch prediction scheme. In *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, pages 69–77. IEEE Computer Society Press, 1998.

[4] Ayose Falcon, Jared Stark, Alex Ramirez, Konrad Lai, and Mateo Valero. Prophet/critic hybrid branch prediction. In *ISCA '04: Proceedings of the 31st annual international symposium on Computer architecture*, page 250, Washington, DC, USA, 2004. IEEE Computer Society.

[5] Timothy H. Heil, Zak Smith, and J. E. Smith. Improving branch predictors by correlating on data values. In *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, pages 28–37. IEEE Computer Society, 1999.

[6] Erik Jacobsen, Eric Rotenberg, and J. E. Smith. Assigning confidence to conditional branch predictions. In *MICRO 29: Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture*, pages 142–152. IEEE Computer Society, 1996.

[7] Daniel A. Jiménez. Fast path-based neural branch prediction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, page 243. IEEE Computer Society, 2003.

[8] Daniel A. Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the Seventh International Symposium on High Performance Computer Architecture*, pages 197–206, Nuevo Leone, Mexico, January 2001.

[9] Artur Klauser, Srilatha Manne, and Dirk Grunwald. Selective branch inversion: Confidence estimation for branch predictors. *Int. J. Parallel Program.*, 29(1):81–110, 2001.

[10] Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge. The bi-mode branch predictor. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 4–13. IEEE Computer Society, 1997.

[11] Gabriel H. Loh and Dana S. Henry. Predicting conditional branches with fusion-based hybrid predictors. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, page 165. IEEE Computer Society, 2002.

[12] Scott McFarling. Combining branch predictors. Technical Report WRL-TN-36, Digital Western Research Laboratory, 1993.

[13] Pierre Michaud. A ppm-like tag-based branch predictor. In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[14] Pierre Michaud, Andre; Seznec, and Richard Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture*, pages 292–303, New York, NY, USA, 1997. ACM Press.

[15] Ravi Nair. Dynamic path-based branch correlation. In *MICRO 28: Proceedings of the 28th annual international symposium on Microarchitecture*, pages 15–23, Los Alamitos, CA, USA, 1995. IEEE Computer Society Press.

[16] Andre Seznec. The o-gehl branch predictor. In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[17] Andre Seznec. Analysis of the o-geometric history length branch predictor. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 394–405, Washington, DC, USA, 2005. IEEE Computer Society.

[18] James E. Smith. A study of branch prediction strategies. In *Proceedings of the 8th annual symposium on Computer Architecture*, pages 135–148. IEEE Computer Society Press, 1981.

[19] Eric Sprangle, Robert S. Chappell, Mitch Alsup, and Yale N. Patt. The agree predictor: a mechanism for reducing negative branch history interference. In *Proceedings of the 24th annual international symposium on Computer architecture*, pages 284–291. ACM Press, 1997.

[20] Renju Thomas, Manoj Franklin, Chris Wilkerson, and Jared Stark. Improving branch prediction by dynamic dataflow-based identification of correlated branches from a large global history. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 314–323, New York, NY, USA, 2003. ACM Press.

[21] Tse-Yu Yeh and Yale N. Patt. Two-level adaptive training branch prediction. In *Proceedings of the 24th annual international symposium on Microarchitecture*, pages 51–61. ACM Press, 1991.

[22] Tse-Yu Yeh and Yale N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Proceedings of the 19th annual international symposium on Computer architecture*, pages 124–134. ACM Press, 1992.