Path Prediction for High Issue-Rate Processors

Kishore N. Menezes Sumedh W. Sathaye Thomas M. Conte
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina 27695-7911
(919)-515-5067
e-mail: {knmeneze,swsathay,conte}@eos.ncsu.edu

Abstract

Rapid developments in the exploitation of instruction-level parallelism are prompting deeper-pipelined, wider machines with high issue rates. Speculative execution has been used to provide the required issue bandwidth. Current methods predict a single branch at a time. Performance improvement is possible by predicting multiple branches in a single cycle. This paper presents a technique to predict paths in a single access. The correlation of a path with the branches executed before it, is exploited to provide high prediction accuracy. A novel path prediction automaton is presented. The automaton is easily scalable to predict long paths through arbitrary subgraphs. It also predicts a path through a subgraph in a single access. The automaton requires only $n+1$ bits for predicting the 2^n paths in a subgraph of depth n. The performance of the proposed path predictor is measured. The full path accuracy *(accuracy in predicting all the branches in a path) is higher than or equal to other predictors found in the literature. This performance is achieved at a low hardware cost. The scalability, single access prediction and low hardware cost of the path prediction technique presented in this paper make it suitable for machines requiring high issue bandwidth.*

1 Introduction

Rapid developments in the exploitation of instruction-level parallelism are prompting deeper-pipelined, wider machines with high issue rates. Supporting such high issue rates requires that a large number of instructions be fetched in a single cycle. This goal is limited by the frequency of control instructions in code. To fulfill the requirements of high issue rates instructions may have to be fetched from the targets of multiple taken branches simultaneously. Achieving this implies predicting multiple branches and their targets in a single cycle. Predicting multiple branches directly translates to predicting a path through the code. This paper presents a technique in which, instead of a single branch being predicted, a *path* through the code is considered as a candidate for prediction. The technique presented here will therefore be referred to as *path prediction* and the mechanism as the *path predictor*.

A large body of work on control flow prediction exists in the literature. Some of the first ideas in hardware branch prediction were presented by J. E. Smith [1]. The concept of correlation between branches and its use in improving the accuracy of branch prediction was described by Yeh and Patt [2], [3], and by Pan, So, and Rahmeh [4]. These techniques are designed to predict a single branch at a time. Yeh, Marr and Patt reported the use of the 2-level adaptive branch predictor for predicting multiple branches simultaneously [5]. Conceptually, this scheme for a two-branch prediction performs speculative predictions for both branches *following* the current branch – one on the taken path and one on the fall-thru path. The prediction for one of the two following branches is selected as the prediction for the second branch, using the first branch prediction as the selector. This scheme can be extended to multiple branches along all the paths, but is limited by the need for additional predictor access bandwidth. For predicting paths with more than two branches, the multiple-branch predictor in [5] requires an exponential number of accesses.

Wallace and Bagherzadeh [6] describe a similar technique for multiple branch prediction. Each pattern history is expanded to contain as many two-bit counters as the number of instructions possible in a single cache block. The prediction for a control instruction in any position in the cache block in available by using the appropriate two-bit counter. The first taken prediction is used to fetch the next cache block or multiple cache blocks as required. The scheme is scalable but requires a large amount of information in each pattern history entry.

Techniques for dynamic control-flow prediction for *multiscalar* processors are presented by Pnevmatikatos, Franklin, and Sohi [7] and Jacobson *et al.* [8]. A subgraph is used as the unit of prediction as well as execution. A

conventional two-level predictor [2] is used to predict the next subgraph to be executed. Dutta and Franklin [9] use *tree-like subgraphs* to predict multiple branches in a single access to the predictor. They treat a path through the code as an entity as is done in this paper. In their scheme, prediction of a path within the subgraph is made via the use of a set of counters. A c bit counter is maintained for each path in a *tree-like* subgraph, for a subgraph of depth n. On any access, a path with the highest counter value is predicted *taken*. In the presence of two or more counters with the same maximum value, arbitration is required. The counters saturate at the maximum value possible in the *taken* direction, or at 0 in the *not-taken* direction. If the prediction is correct the counter value used to predict the path is incremented by a fixed value, while all the other counters are decremented by 1. There are similarities between this technique and the path predictor presented here, but there also are a few important distinctions which make the proposed path predictor unique. These are elaborated on in a later section.

Seznec, *et.al.* [10] suggest a technique called the *two-block ahead branch predictor*. In this scheme, information from the current instruction block is used to predict the address of the instruction following the next instruction block rather than the next instruction block.

This paper proposes a technique for predicting paths through program code by means of a single access to the predictor. The address of the first branch in the set of branches that make up the paths within a subgraph is the only information required for a prediction. High bandwidth fetch mechanisms requiring such predictors have been suggested in the literature [11] [12]. The prediction automaton is derived by scaling the two-bit counter [1] to predict multiple branches. These predictions in conjunction with target address information in the *Path Address Cache* can be used to fetch from multiple branch targets. The automaton is easily scalable and requires a minimal number of bits. The paper is organized as follows. Section 2 describes the automaton used to predict paths. Section 3 explains the use of the path prediction automaton in a one-level and two-level path predictor. It also describes the *Path Address Cache* and compares the proposed prediction mechanism with other schemes that have been proposed in the literature. The proposed scheme is evaluated and the experimental results are presented in Section 4. Section 5 concludes the paper.

2 The path prediction automaton

The scheme presented in this paper considers a path as an entity and uses it as the basis of prediction. It uses the correlation of a path with the branches executed before it. Other schemes that predict multiple branches [5] [6], with the exception of Dutta and Franklin, predict each branch as a single entity. Consider the simple subgraph shown in Figure 1(a). The branch in block A determines the control flow to either block B or block C. The prediction of the branch in block A constitutes simple branch prediction. Alternatively, the subgraph can be viewed as two different paths *A-B* and *A-C*. Similarly, any given subgraph of depth n can be split into up to 2^n paths[1]. A larger subgraph with two levels is shown in Figure 1(b). A *depth-2* subgraph can be split into 2^2 paths.

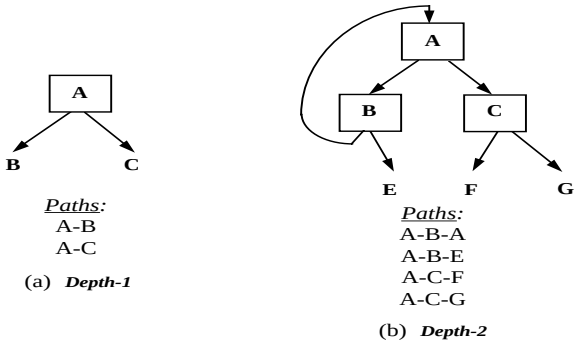


Figure 1. *Depth-n* **subgraphs. (a) A branch is a decision between two paths. Simple branch prediction is a prediction of one of two paths. (b) A** *depth-2* **subgraph gives rise to four paths.**

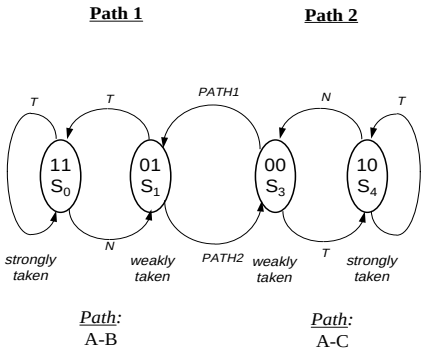


Figure 2. The two-bit counter branch predictor used as a prediction mechanism to predict one of two paths. Each path has a *strongly-taken* **state and a** *weakly-taken* **state.**

[1] The number of paths may be greater than 2^n when a subgraph contains one or more branches with more than two targets.

The prediction for a single branch is a speculative decision between two paths. For the *depth-1* subgraph shown in Figure 1(a) the choice is between path *A-B* and *A-C*. Branch prediction can therefore be considered to be the degenerate case of path prediction. The 2-bit Smith counter can be viewed as a path predictor for such a case, as shown in Figure 2. In Figure 2, states $S_0(11)$ and $S_1(01)$ correspond to path *A-B* being *strongly-taken* and *weakly-taken*, respectively. States $S_2(10)$ and $S_3(00)$ correspond to path *A-C* being *strongly-taken* and *weakly-taken*, respectively. The least significant bit in each state indicates the path to be executed (branch taken or branch not-taken), and the most significant bit indicates saturation. States with the saturation bit as 0 are *weakly-taken* states and those with the saturation bit as 1 are termed *strongly-taken* states. As seen in the figure, one *weakly-taken* state and one *strongly-taken* state constitute the states for a given path.

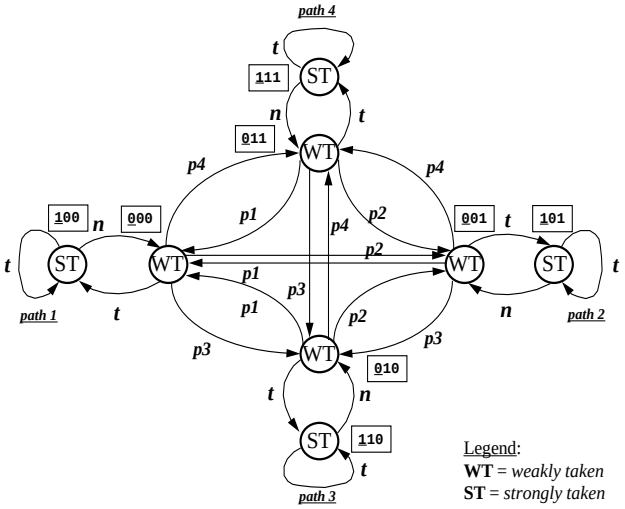


Figure 3. Finite state machine for a *depth-2* 3-bit path predictor. The FSM can predict one of four paths.

The speculative execution of paths in a *depth-n* subgraph, where $n > 1$, requires a decision between more than two paths. The 2-bit Smith counter can be scaled to predict more than two paths by adding one *weakly-taken* state and one *strongly-taken* state for each additional path. Such automata and the performance they yield when used to predict paths are the topic of this paper. For a 2-level subgraph which yields 2^2 possible paths of execution, an automaton with $2^2 \times 2$ states is required. Such an automaton can be achieved with a 3-bit finite-state machine. Such a finite state machine is shown in Figure 3.

Present State		Execution Outcome	Next State
Path x	WT	Path x	Path x (ST)
		Path y	Path y (WT)
	ST	Path x	Path x (ST)
		Path y	Path x (WT)

Table 1. State transition table for path prediction automaton.

To generalize, an n-level subgraph requires n branch predictions and can be split into 2^n paths. Path prediction among 2^n paths can be achieved using an $(n+1)$ bit finite state machine which yields $2^{(n+1)}$ states. Each path in the finite state machine consists of a *weakly-taken* state and a *strongly-taken* state, similar to that for a branch in the 2-bit Smith counter. Table 1 summarizes the transitions made by the path prediction automaton.

3 The Path Predictor

The path predictor can be constructed using any of the configurations for branch prediction suggested in the literature [1], [2], [4]. The following subsections present the one-level and two-level configurations for the path predictor.

3.1 One-Level Path Predictor

The structure of a *one-level path predictor* is shown in Figure 4. It is based on the counter-based scheme suggested in [1]. It consists of a predictor table, each entry of which contains the path prediction automaton instead of a two-bit counter. The address of the first branch in the subgraph is used to index into the predictor table. The n-bit prediction for a path of *depth-n* is available from the selected entry. Each bit in the n-bit prediction represents the prediction for each branch in the path. The automaton is updated using the actual path outcome.

3.2 Two-Level Path Predictor

Correlation-based schemes have been suggested for branch prediction. Such schemes have been shown to provide higher branch prediction accuracy [2], [4]. The *two-level path predictor* uses the Two-level Adaptive Branch Predictor configuration as described in [2]. Any of the different possible configurations may be used for path prediction. The GAg path prediction configuration is shown in Figure 5. The two structures used are the *Global History Register* (GHR) and a *Pattern History Table* (PHT). The

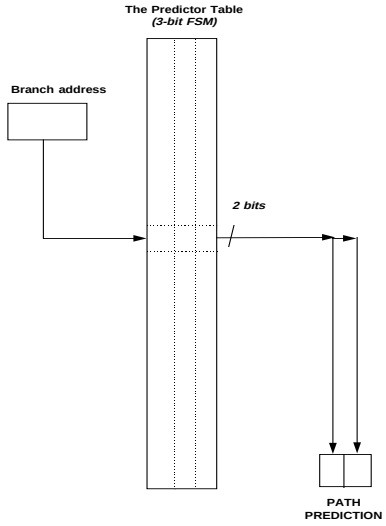


Figure 4. The *one-level path predictor* for predicting *depth-2* paths. The branch address is used to index into the prediction table. The two least significant bits in the selected entry are the predictions for the two branches in the path.

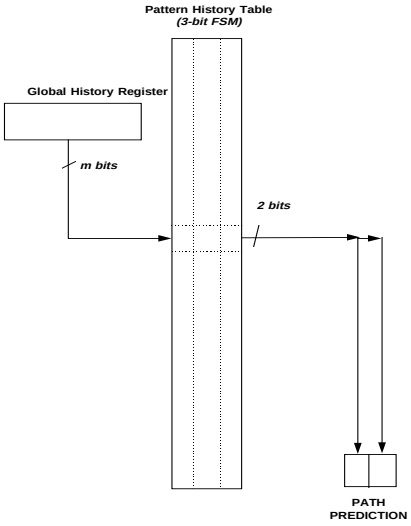


Figure 5. The *two-level path predictor* for predicting *depth-2* paths. The GHR is used to index into the pattern history table. The two least significant bits in the selected entry are the predictions for the two branches in the path. The most significant bit is the saturation bit.

two-level path predictor differs from the traditional *two-level adaptive branch predictor* in the following ways:

- The automaton in each entry of the Pattern History Table is a $n+1$ bit path prediction finite state machine rather than a 2-bit counter used in the GAg scheme [3].

- Path outcomes, in contrast to branch outcomes in a branch predictor, are shifted into the Global History Register. The GHR is shifted left by n bits and the path outcome forms the rightmost bits for a *depth-n* predictor. In case of speculative updates, the path prediction rather than the actual outcome are shifted into the GHR.

Figure 5 shows a two-level path predictor for predicting one of four paths. When a prediction is requested from the path predictor, a PHT is selected based on the address and indexed into using the GHR. The pattern entry in the PHT consists of three bits for predicting one of four paths. The least significant bit is the prediction for the first branch in the path. It represents the first half of the path being predicted. The second bit is the prediction for the second branch and determines the second half of the selected path. The most significant bit is the bit that indicates saturation and is neglected during prediction.

3.3 Path Address Cache

For a prediction to be useful it is necessary that the fetch address of the target or the fall-thru path of the branch being predicted be available. These addresses are stored in a *Path Address Cache*(PAC). The PAC is indexed using the address of the first branch in the path to be predicted. Each entry in the PAC stores the addresses for the *taken* and *fall-thru* paths for each branch in the path. Based on the prediction for the path, the required addresses are extracted from the PAC entry selected. These addresses are then used to initiate the fetch from the instruction cache. The target addresses in the PAC are updated with the actual execution outcomes of the branches in the path.

For a *depth-2* path prediction, six addresses need to be stored in each PAC entry, the *taken* and *fall-thru* addresses for the first and second branch in the subgraph. In general, for a *depth-n* path prediction, $(2^1 + \cdots + 2^n)$ addresses need to be stored in each PAC entry. Alternatively, only $((2^1 + \cdots + 2^n) - 1)$ addresses may be stored since the *fall-thru* address for the first branch in the subgraph may be computed by incrementing the address of the branch. In addition to the addresses of the targets, the PAC may store other information which may be useful to the instruction fetch mechanism, such as the length of the basic block at each target.

The *path predictor* indicates a correct prediction only when the address extracted from the *Path Address Cache* using the prediction from the predictor is correct. When any branch in the path is mispredicted, recovery action is initiated. Misprediction recovery can be achieved in the following ways:

- If the k^{th} branch in the path is mispredicted, the *path predictor* is queried for a new *depth-n* path prediction for the target of the mispredicted branch.
- Alternatively, the mispredicted branches in the path may be repaired sequentially without re-accessing the *path predictor*. This misprediction repair mechanism is assumed for the results in this paper.

3.4 Comparison with other schemes

This section compares the *two-level path predictor* to other predictors that have been proposed for predicting multiple branches or paths. The comparison is based on: scalability, hardware cost, and time for prediction.

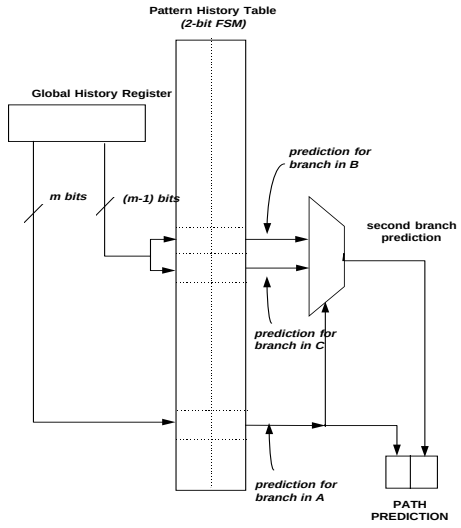


Figure 6. The *multiple branch predictor* **[5]. The m bits in the GHR are used to index into the pattern history table. The pattern in this entry yields the prediction for the first branch in the path. In parallel, the least significant $m-1$ bits of the GHR are used with a 0 and 1 appended as the rightmost bit to index into two other entries. The prediction for the first branch is used to select one of the two as the prediction for the second branch.**

The *multiple branch predictor* [5] uses the *two-level adaptive branch predictor* to predict multiple branches. Figure 6 shows the configuration of the *multiple branch predictor*. Though a single PHT is shown in the figure, multiple PHTs are possible. The branch address is used to select a PHT. The selected PHT is indexed using the pattern in the GHR. Since the branches are predicted individually, and the intermediate addresses in a *depth-n* path are not available, this scheme is limited to using a *global history register* rather than *per address history registers*. Also, every successive branch prediction after the first branch in the path is dependent on the predictions for the branches before it. This could potentially increase the time required to predict a path. For any branch following the first branch, predictions are obtained for branches on the *taken* path as well as *fall-thru* path simultaneously. An exponential number of accesses to the predictor are required. For example, for predicting the four branches in a *depth-4* path, $2^0 + \cdots + 2^3$ predictor accesses would be required in a single cycle. In general, for a *depth-n* subgraph, the number of simultaneous accesses to the predictor would be $\sum_{i=0}^{n-1} 2^i$. The *multiple branch predictor* is therefore not easily scalable for predicting long paths.

The *subgraph predictor* [9] shown in Figure 7 uses the concept of paths as done in this paper. The predictor presented in [9] uses *per address branch history registers*. The *subgraph predictor* requires one counter for each path. Assuming each counter is c bits for a *depth-n* path prediction, each PHT entry holds a $n \times c$ bit pattern. The scalability of the predictor may be limited by the number of bits required in each PHT entry. The number of bits required also adds to the hardware cost of building such a predictor. When a prediction access is made a PHT entry is selected and the maximum counter value is selected as the prediction. This may marginally increase the time to predict. In the event that two counters hold the same maximum value, arbitration is required. The authors suggest predicting the last path outcome as the new prediction. Additional bits in each may be required to accomplish this, consequently adding to the hardware cost.

The *path predictor* treats a path as a single entity for prediction. Though the *path predictor* presented in this paper uses a *global history register*, unlike the *multiple branch predictor* it can also be implemented using *per address path history registers* since the address required for a path prediction is that of the first branch in the path only. The *path predictor* uses only $n + 1$ bits for each entry in the PHT for predicting a *depth-n* path. The predictor is therefore easily scalable for predicting longer paths. The prediction is available immediately since the bits in the PHT themselves constitute the path prediction.

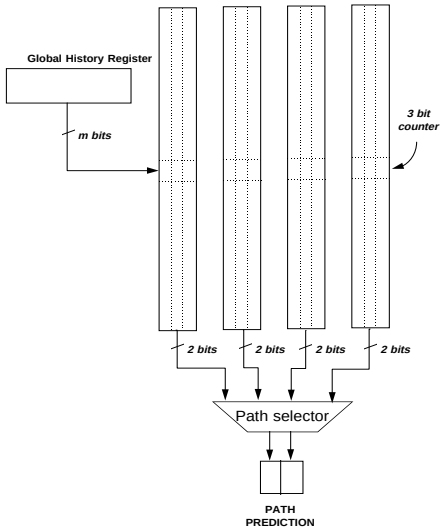


Figure 7. The *subgraph predictor* using a global history register. The subgraph predictor presented in [9] uses per subgraph or path history registers. The GHR is used to index into the pattern history table. The selected entry has 4 counters for a *depth-2* path. The path belonging to the maximum counter value is predicted. Arbitration may be required if two or more counters have the same maximum value.

4 Experimental Results

Experiments were conducted for the *depth-2* and *depth-4* two-level path predictors. The performance of the *path predictor* is compared to the *multiple branch predictor* for *depth-2* and *depth-4* (even though the *multiple branch predictor* requires $2^0 + \cdots + 2^3$ predictions in a single cycle for *depth-4*, which may not be practical). It is also compared to a version of the *subgraph predictor* that uses a single, global history register as shown in Figure 7. Note that the original subgraph predictor presented by Dutta and Franklin [9] uses multiple path history registers. A single path history register has been used here in order to be able to compare it to the *path predictor* which for the purposes of this paper uses only a single history register. An adaptation of the *path predictor* to use multiple path history registers is possible but is not addressed in this paper. All the predictors used in the experiments use a 1024-entry *Path Address Cache*. The number of addresses held in each entry varies according to the *depth* of the predictions required. The behavior of the *Path Address Cache*, and consequently its contribution to the prediction of a path, remains the same when simulating the *path predictor*, *multiple branch predictor* and the *subgraph predictor*.

All of the integer programs from the SPEC95 suite were used as benchmarks. The benchmarks were run with the training inputs provided with the SPEC95 suite. All benchmarks were run to completion.

The first metric used for comparison is *prediction accuracy*. The prediction accuracy is divided into several subsets. The k-length accuracy ($k < n$) is the percent number of paths in which the first k branches in a *depth-n* path prediction are predicted correctly, and for which the $(k+1)^{th}$ branch is mispredicted. The n-length accuracy (also referred to as *full path accuracy*), is the percentage of path predictions where all the branches in the path through the subgraph were correctly predicted. For $k = 0$, the k-length accuracy indicates the percent paths in which the first branch in the subgraph was mispredicted.

The second metric used in this paper is the *Effective Path Accuracy* (EPA). The effective path accuracy is an indicator of the cumulative instruction fetch possible given a perfect fetch mechanism. This number provides a single figure of merit for the comparison of the schemes simulated. For a *depth-n* path prediction it is computed from the k-length accuracies as follows:

$$\left.\begin{array}{r}\text{Effective Path}\\ \text{Accuracy}\end{array}\right\} = \sum_{k=1}^{n} \frac{k}{n} \times (\text{k}-\text{length accuracy}) \quad (1)$$

For *depth-4* path prediction, Equation 1 becomes

$$\begin{aligned}\text{EPA for Depth}-4 &= (1.0 \times (4-\text{length accuracy}))\\ &+ (0.75 \times (3-\text{length accuracy}))\\ &+ (0.5 \times (2-\text{length accuracy}))\\ &+ (0.25 \times (1-\text{length accuracy}))\end{aligned}$$

It should be noted that even when the k^{th} branch in a path is mispredicted, branches $(k+1)$ to n could be predicted correctly. Such correctly predicted branches do not contribute to the ability to fetch instructions. However, the branch being correctly predicted implies that the target address is available. This target address can be used to quicken the misprediction recovery process. The scalar *branch prediction accuracy* reflects the fraction of times such indirect correct predictions occur. The scalar *branch prediction accuracy* is used as another metric to show that the path predictor can also provide competitive scalar branch prediction.

Table 2 shows the percent k-length accuracies for $k = 0$ to 4 for a *depth-4* path predictor and the corresponding *multiple branch predictor* and *subgraph predictor* which predict 4 branches in a single access. All three predictors were simulated using a *global history register* size of 8, and 4 *pattern history tables*. The *path predictor* consis-

tently outperforms the other predictors in predicting complete paths ($k = 4$). The greatest performance improvement of the *path predictor* over the *multiple branch predictor* is 7.75% for *134.perl*. The *full path accuracy* for *130.li* and *147.vortex* are more than 5% greater, and more than 3% greater for *126.gcc* for the *path predictor*. The performance of the *path predictor* exceeds that of the *subgraph predictor* by more than 6% for *099.go* and *130.li*. For *129.compress*, *134.perl* and *147.vortex* the full-path accuracy is more than 4% greater than that for the *subgraph predictor*.

The *effective path accuracy* for the predictors using a *global history register* size of 8 and 4 PHTs is shown in Figure 8(a). The *path predictor* surpasses the *multiple branch predictor* in EPA by more than 2% for *130.li* and *147.vortex*. The performance for *134.perl* is about 5% greater. The EPA for the *path predictor* is more than 4% greater than that for the *subgraph predictor* for *099.go*. It is also at least greater than 2% for most of the other benchmarks. The *multiple branch predictor* and *subgraph predictor* do not fare well in predicting full paths. However, they do succeed in predicting the first few branches in the subgraph correctly.

Table 3 shows the k-length accuracies for a *global history register* size of 12. The performance of all three predictors increases with an increase in the size of the global history register. The *multiple branch predictor* performance approaches that of the path predictor. The difference in the full path accuracy for *134.perl* reduces from about 7.75% for a GHR size of 8 to about 2% for a GHR size of 12. However, the difference in performance of the *path predictor* and the *subgraph predictor* widens at a GHR size of 12. The full-path accuracy for the *subgraph predictor* lags that of the path predictor by 9.2% for *126.gcc* and 7.94% for *099.go*. The EPA for the predictors at a GHR size of 12 is plotted in Figure 8(b). The EPA performance of the *path predictor* exceeds that of the *multiple branch predictor* by about 1% for some benchmarks and about 2% for *130.li*. It exceeds the EPA for the *subgraph predictor* by over 6% for *126.gcc*, 5% for *099.go* and by at least 2% for most of the other benchmarks.

Results of the experiments to measure the full path accuracy of the path predictor, the multiple branch predictor, and the subgraph predictor are shown in Figures 9 and 10. For each of the benchmarks two plots are presented. Four and eight PHTs were used for the measurements presented in the first and the second plot respectively. A *depth-4* path predictor, multiple branch predictor capable of predicting 4 branches, and a subgraph predictor capable of predicting 4–deep subgraphs were used. The full path prediction accuracy achieved by both predictors for the *depth-2* case was comparable, and hence is not presented in this paper. For each benchmark, the full path prediction accuracy of the path predictor is indicated as "path Full-path". For the multiple branch predictor, the full path prediction accuracy is indicated by "multi Full-path". For the subgraph predictor, the full path prediction accuracy is indicated by "subgr Full-path". The three are shown as the first set of bars for each of the various sizes of GHR ranging from 8 bits to 14 bits.

It is apparent from these plots that the path predictor achieves superior accuracy in all the cases. Benchmarks such as *130.li*, *147.vortex*, and *134.perl* show remarkable improvement over the multiple branch predictor: observe that the difference between *path Full-path* and *multi Full-path* for these benchmarks is in the range of 3%–7.5% for both 4 and 8 PHTs and for smaller GHR sizes (8– and 9–bits). The difference diminishes to about 1% for larger GHRs (14–bits). For *126.gcc* and *099.go*, which are notoriously hard to predict, as well as for *129.compress* and *132.ijpeg* the difference is about 2–3.5% at smaller GHR sizes. When compared to the subgraph predictor, the difference between *path Full-path* and *subgr Full-path* is between 3% (for *147.vortex* for most of the GHR sizes), and 9% (for *126.gcc* across the GHR sizes). The full path prediction accuracy for *124.m88ksim* is near-perfect across all GHR sizes, for all three predictors.

Figures 9 and 10 also show the branch prediction accuracy achieved in the same experimental setup (indicated as "path Branch pred", "multi Branch pred", and "subgr Branch pred"). It can be seen that for *134.perl, 147.vortex,* and *130.li,* for smaller GHR sizes (8–10 bits), *path Branch pred* is about 2–3% greater than *multi Branch pred*. For larger GHRs, it is slightly better than or matches *multi Branch pred*. Comparing *path Branch pred* with *subgr Branch pred*, the path predictor branch prediction accuracy is consistently better than the subgraph predictor. Again, the branch prediction accuracy for *124.m88ksim* is near-perfect across all GHR sizes, for all three predictors.

5 Conclusions and Future Work

This paper proposes a novel method for predicting paths through program code. A path through a program directly translates to the directions taken by multiple branches. Previous work has concentrated on predicting branches using correlation between the branch to be predicted and the branches executed before it in the code. This paper treats a path as an entity by itself. The path predictor can be configured in a similar manner to branch predictors used before. However, it differs in the prediction automaton used.

The prediction accuracy of the path predictor is measured. The capability of the path predictor to predict *full paths* (all the branches in a path) is better than or equal to that for other methods to predict multiple branches. The path predictor also outperforms or equals other multiple-branch predictors in branch prediction accuracy. The prediction automaton used in the path predictor requires two

states for each path. It operates in a manner similar to the two-bit counter used to predict branches. The prediction automaton is easily scalable such that it can be used to predict a large number of paths through arbitrary subgraphs. It also provides the path prediction in a single access. The hardware cost of the prediction automaton is minimal since it requires only $n + 1$ bits to predict the 2^n paths through a subgraph of depth n. The path predictor provides high performance, scalability and faster prediction at a lower hardware cost.

Many different applications for a predictor that predicts paths or multiple branches have been suggested in the literature. The path prediction automaton can also be used in a one-level path predictor. The performance of instruction fetch mechanisms using one-level and two-level path prediction is currently under investigation.

References

[1] J. E. Smith, "A study of branch prediction strategies," in *Proc. 8th Ann. Int'l. Symp. Computer Architecture*, pp. 135–148, June 1981.

[2] T. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proc. 24th Ann. Int'l Symp. on Microarchitecture*, (Albuquerque, NM), pp. 51–61, Nov. 1991.

[3] T. Yeh and Y. N. Patt, "Alternative implementations of two-level adaptive branch prediction," in *Proc. 19th Ann. Int'l Symp. on Computer Architecture*, (Gold Coast, Australia), May 1992.

[4] S.-T. Pan, K. So, and J. T. Rahmeh, "Improving the accuracy of dynamic branch prediction using branch correlation," in *Proc. Fifth Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems*, (Boston, Massachusetts), pp. 248–259, Oct. 1992.

[5] T. Yeh, D. T. Marr, and Y. N. Patt, "Increasing the instruction fetch rate via multiple branch prediction and a branch address cache," in *Proc. 7th ACM Int'l Conference on Supercomputing*, pp. 67–76, July 1993.

[6] S. Wallace and N. Bagherzadeh, "Multiple branch and block prediction," in *Proc. 3rd IEEE Symp. on High Performance Computer Architecture* [13].

[7] D. N. Pnevmatikatos, M. Franklin, and G. S. Sohi, "Control flow prediction for dynamic ILP processors," in *Proc. 26th Ann. Int'l Symp. on Microarchitecture*, (Austin, TX), pp. 153–163, Dec. 1993.

[8] Q. Jacobson, S. Bennett, N. Sharma, and J. E. Smith, "Control flow speculation in Multiscalar processors," in *Proc. 3rd IEEE Symp. on High Performance Computer Architecture* [13].

[9] S. Dutta and M. Franklin, "Control flow prediction with tree-like subgraphs for superscalar processors," in *Proc. 28th Ann. Int'l Symp. on Microarchitecture*, (Ann Arbor, MI), pp. 258–263, Dec. 1995.

[10] A. Seznec, S. Jourdan, P. Sainrat, and P. Michaud, "Multiple-block ahead branch predictors," in *Proc. Seventh Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems*, (Boston, MA), pp. 116–127, Oct. 1996.

[11] T. M. Conte, K. N. Menezes, P. M. Mills, and B. A. Patel, "Optimization of instruction fetch mechanisms for high issue rates," in *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, (Santa Margherita Ligure, Italy), pp. 333–344, May 1995.

[12] E. Rotenberg, S. Bennet, and J. Smith, "Trace Cache: a low latency approach to high bandwidth instruction fetching," in *Proc. 29th Ann. Int'l Symp. on Microarchitecture*, (Paris, France), Dec. 1996.

[13] *Proc. 3rd IEEE Symp. on High Performance Computer Architecture*, (San Antonio, TX), Jan. 1997.

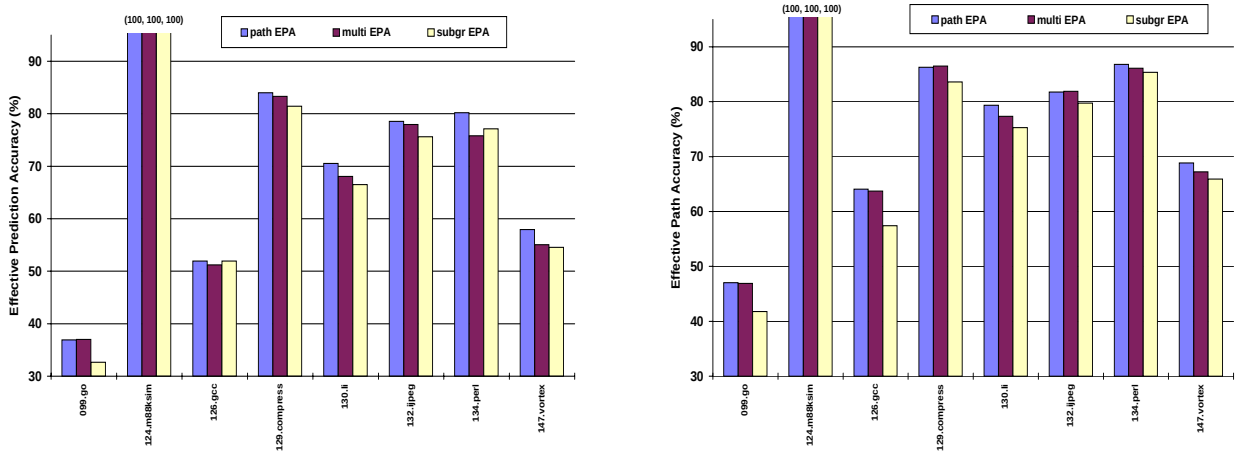
(Tables 2 and 3, and Figures 8, 9 and 10 are on the following pages.)

	$k=4$			$k=3$			$k=2$			$k=1$			$k=0$		
Benchmark	$Subgr$	$Multi$	$Path$	$Subgr$	$Multi$	$Path$	$Subgr$	$Multi$	$Path$	$Subgr$	$Multi$	$Path$	$Subgr$	$Multi$	$Path$
099.go	16.48	20.84	22.64	6.866	7.004	5.675	11.67	11.67	10.28	20.85	20.3	19.56	42.03	40.19	41.85
124.m88ksim	100	100	100	9E-06	2E-05	6E-06	2E-05	2E-05	2E-05	6E-05	5E-05	4E-05	8E-05	6E-05	7E-05
126.gcc	35.95	35.59	38.7	7.92	7.746	6.013	11.17	11.18	9.47	17.77	16.92	15.94	31.08	28.58	29.87
129.compress	71.29	74.44	75.97	6.358	5.484	4.784	6.967	5.906	5.587	7.533	7.268	6.597	7.85	6.906	7.067
130.li	53.4	54.3	59.64	7.662	7.485	5.678	9.187	10.25	8.019	11.01	12.01	10.59	16.39	15.95	16.06
132.ijpeg	65.93	68.17	68.58	5.094	5.259	4.839	7.256	7.353	8.485	9.064	8.632	8.485	11.62	10.58	11.21
134.perl	70.51	66.89	74.64	3.464	4.93	2.793	4.997	6.372	3.98	6.065	8.095	5.836	11.37	13.71	12.75
147.vortex	37.88	37.33	42.45	9.648	10.15	8.12	11.12	11.58	10.72	15.43	17.33	16.14	19.81	23.6	22.57

Table 2. *Depth-4* k**-length accuracies for the** *Subgraph predictor* **(**$Subgr$**), the** *multiple branch predictor* **(**$Multi$**), and the** *path predictor* **(**$Path$**) using a global history register size of 8 and 4 PHTs.**

	$k=4$			$k=3$			$k=2$			$k=1$			$k=0$		
Benchmark	$Subgr$	$Multi$	$Path$	$Subgr$	$Multi$	$Path$	$Subgr$	$Multi$	$Path$	$Subgr$	$Multi$	$Path$	$Subgr$	$Multi$	$Path$
099.go	24.58	31.03	32.52	8.38	7.526	6.511	12.36	11.51	10.58	18.96	17.95	17.38	32.67	31.98	33.01
124.m88ksim	100	100	100	8E-06	1E-05	6E-06	2E-05	2E-05	2E-05	5E-05	4E-05	4E-05	7E-05	6E-05	7E-05
126.gcc	43.2	50.38	52.4	7.529	7.198	5.871	9.867	9.356	8.261	14.5	13.07	12.62	22.12	20	20.85
129.compress	74.16	78.86	78.89	6.143	4.816	4.554	6.344	5.301	5.171	6.767	5.45	5.537	6.58	5.573	5.844
130.li	63.77	67.13	68.49	6.842	5.752	6.932	8.07	7.327	6.932	9.341	8.963	8.786	11.17	10.82	10.83
132.ijpeg	70.5	73.14	73.34	5.061	5.034	4.683	6.908	6.45	6.239	7.962	7.134	7.194	9.275	8.245	8.542
134.perl	80.58	81.24	83.05	2.383	2.676	1.778	3.895	3.726	3.003	4.262	4.045	3.779	8.347	8.309	8.39
147.vortex	50.61	52.47	54.78	9.483	8.784	8.293	10.06	9.74	9.38	12.61	13.18	12.73	14.14	15.82	14.82

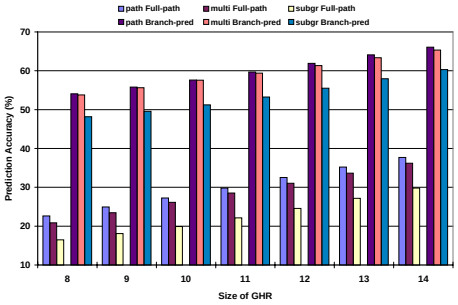
Table 3. *Depth-4* k**-length accuracies for the** *Subgraph predictor* **(**$Subgr$**), the** *multiple branch predictor* **(**$Multi$**), and the** *path predictor* **(**$Path$**) using a global history register size of 12 and 4 PHTs.**



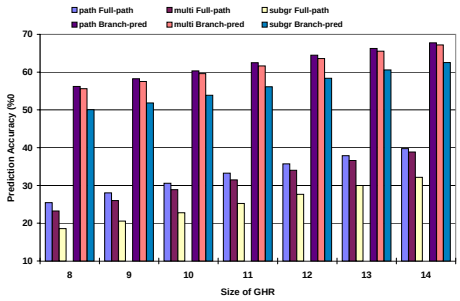
(a) GHR size = 8 **(b) GHR size = 12**

Figure 8. Effective path accuracy (EPA) for *depth-4* **path prediction using 4 PHTs and global history register size (a) 8 (b) 12.**

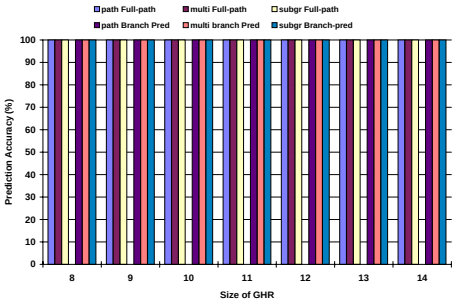
099.go (Number of PHTs = 4)



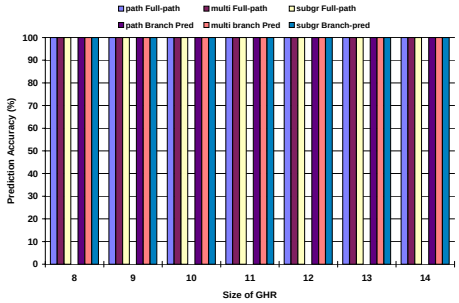
099.go (Number of PHTs = 8)



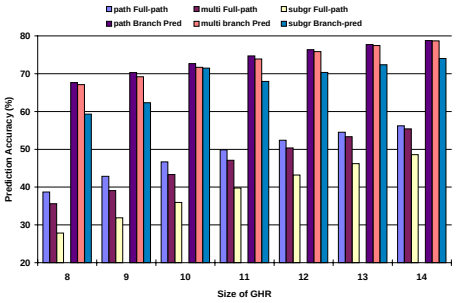
124.m88ksim (Number of PHTs = 4)



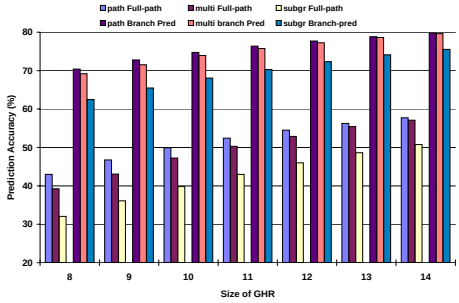
124.m88ksim (Number of PHTs = 8)



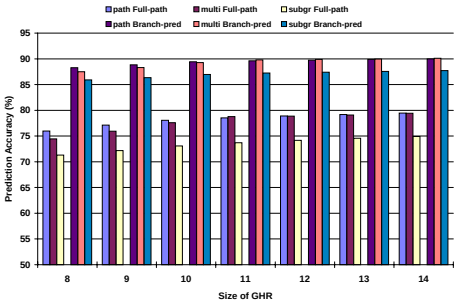
126.gcc (Number of PHTs = 4)



126.gcc (Number of PHTs = 8)



129.compress (Number of PHTs = 4)



129.compress (Number of PHTs = 8)

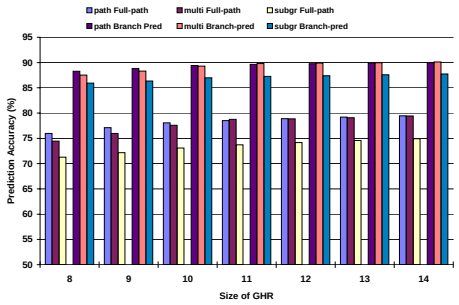
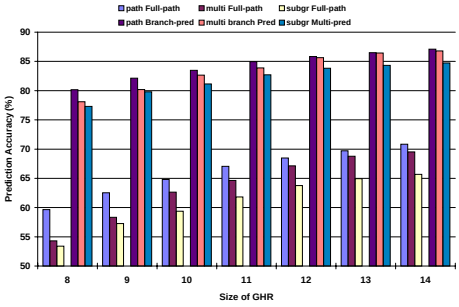
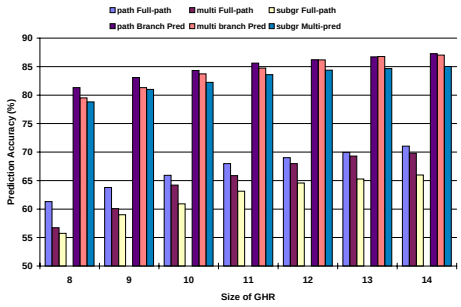


Figure 9. Comparing the performance of the path predictor, the multiple branch predictor, and the subgraph predictor: two plots for each benchmark show the *full-path accuracy* and the *branch accuracy* for 4 and 8 PHTs. Key: **path Full-path** = full path accuracy for the path predictor, **multi Full-path** = full path accuracy for the multiple branch predictor, **subgr Full-path** = full path accuracy for the subgraph predictor, **path Branch-pred** = branch prediction accuracy for the path predictor, **multi Branch-pred** = branch prediction accuracy for the multiple branch predictor, and **subgr Branch-pred** = branch prediction accuracy for the subgraph predictor.

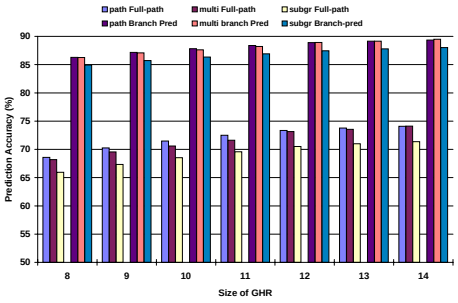
130.li (Number of PHTs = 4)



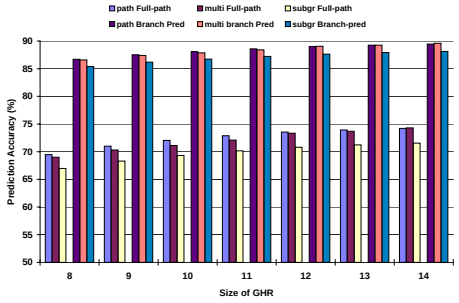
130.li (Number of PHTs = 8)



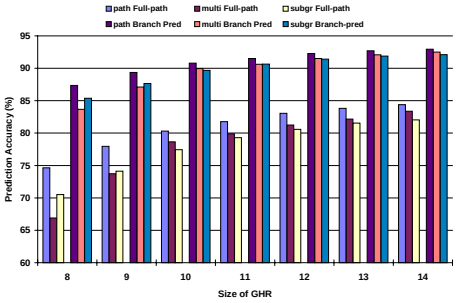
132.ijpeg (Number of PHTs = 4)



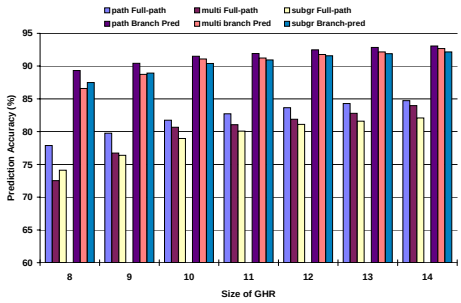
132.ijpeg (Number of PHTs = 8)



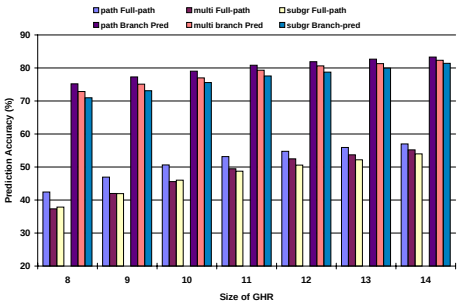
134.perl (Number of PHTs = 4)



134.perl (Number of PHTs = 8)



147.vortex (Number of PHTs = 4)



147.vortex (Number of PHTs = 8)

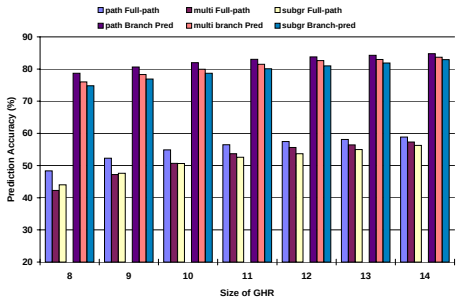


Figure 10. Comparing the performance of the path predictor, the multiple branch predictor, and the subgraph predictor: two plots for each benchmark show the *full-path accuracy* and the *branch accuracy* for 4 and 8 PHTs. Key: **path Full-path** = full path accuracy for the path predictor, **multi Full-path** = full path accuracy for the multiple branch predictor, **subgr Full-path** = full path accuracy for the subgraph predictor, **path Branch-pred** = branch prediction accuracy for the path predictor, **multi Branch-pred** = branch prediction accuracy for the multiple branch predictor, and **subgr Branch-pred** = branch prediction accuracy for the subgraph predictor.