# Of Limits and Myths in Branch Prediction

by

**Avinoam Nomik Eden**

A dissertation submitted in partial fulfillment
of the requirement for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2001

ABSTRACT

# Of Limits and Myths in Branch Prediction

**by**

**Avinoam Nomik Eden**

**Chair: Trevor N. Mudge**

The need to flush pipelines when miss-predicting branches occur can throttle the performance of a pipelined super-scalar microprocessor. It is argued that by the year 2010 branch prediction will become the most limiting factor in processor performance [1]. A plethora of research has been done on the subject of branch prediction. While many branch prediction structures have been proposed, their performance is usually demonstrated empirically through simulations that provide little insight into the underlying principle that enables their behavior.

Since the introduction of the two-level dynamic branch prediction scheme, research into branch prediction has followed four different paths. The first attempts to improve prediction by reducing aliasing in the second level table, which was shown to adversely affect prediction accuracy. The second attempts to improve prediction accuracy by combining two or more different components in the branch prediction structure. The third attempts to improve prediction by changing the configuration of a particular predictor. Lastly, the fourth, tries to find new schemes to improve branch prediction. Most papers on research along one path ignored comparisons with other paths on the basis that the different paths are orthogonal.

A set of studies is presented that consolidate the different research paths by showing that the advantage gained by most of them is to reduce aliasing. After showing that reducing aliasing is the prevailing factor in prediction gain regardless of which path of research is followed, we highlight a set of criteria that a predictor should embrace, to have a good prediction. The criteria emerge from the studies we performed and previous work on the subject.

The set of criteria, a predictor should follow to achieve good prediction accuracy, is used to build a new predictor – YAGS. YAGS outperforms the leading branch predictor structures from the different paths of research. It provides the micro-architect with a set of parameters that can be used to meet different restrictions, such as size and latency.

This work highlights misconceptions that resulted from the work done on the topic. It especially stresses the importance of a relevant limit study for understanding a new branch prediction scheme and structure.

## ACKNOWLEDGEMENTS

I would like to thank my parents, which provided a worry free environment for pursuing my higher education, and in the process turned me into an over-educated spoiled brat. Thank you.

Thanks to my advisor, professor Trevor Mudge, for deciding I should pursue my Ph.D. Thank you for the pets on the back and kicks in the butt you provided as necessary, and, of course, for translating my papers into English. Thank you.

Thanks to my boss and my friend Vik Kheterpal for providing a flexible work environment and support that allowed me to pursue my PhD while holding a job.

Special thank you to Professor Kevin Compton and Robin Rennie for providing mental support and concrete advice during my early stages of education. Thank you.

Thank you to Jeff Ringenberg and Adam Freund. It was a pleasure working with you.

Moreover ☺, thanks to Amy Claire Harfeld for editing this dissertation and coping with the bore.

## TABLE OF CONTENTS

# LIST OF TABLES

## LIST OF FIGURES

# Chapter 1 - Introduction

As VLSI technology continues to improve, more resources become available for the branch prediction module. Concurrently, newer high-performance machines are implementing deeper pipelines and greater issue-widths. This, in turn, increases the number of branches predicted and not yet retired, and increases the branch misprediction penalty. Code size is expected to increase [2], and the memory state reached in one cycle to decrease [63][65]. Although more resources are available to computer architects, the decreasing state reached in one cycle dictates the usage of smaller branch predictors, if the prediction is going to happen in one cycle. Thus the need to predict more branches with higher accuracy employing a smaller amount of resources continues to grow.

## 1.1 The Branch Irony

The branch instruction is thing that separates a computer from a calculator. It facilitated the leap from simple sequential calculations performed by a calculator to complex calculations and tasks performed by computers today.

At first, computers executed programs sequentially – one instruction was executed before the next instruction started. By the time the instruction following the branch was fetched, the outcome of the branch instruction was known and it was clear which instruction was to follow. Micro-architectural mechanisms to speed execution led to pipelining and super-scalar cores. With these innovations, more than one instruction is executed concurrently, and possibly, execution is not completed in sequential order.

Pipelining and super-scalar architectures have resulted in several complications. One of the more prominent ones is the control hazard.  This arises when the instruction following a branch is fetched before the branch instruction is fully executed.  When that instruction is executing, it is not clear whether the branch is going to be taken or not.  If the branch is taken, the address of the next instruction (the branch target address) is not yet calculated.  One solution to the control hazard problem is to stop further instruction fetching until the branch is finished executing.  This, however, reduces the advantage gained by pipelined and super-scalar architectures, and therefore is not a desirable solution.  A better solution would be to make an educated guess at the branch direction and target address and to follow the execution accordingly.  If the guess is correct, pipelining and super-scalar architectures would be allowed to fulfill their potential.  If an incorrect guess is made, a recovery mechanism would need to be in place to roll back the machine to the state just after the miss-predicted branch finished executing.  This process is the process of branch prediction.

It is estimated that 1 out of 5 instructions is a branch instruction.  Current microprocessors demand instructions at a high rate, and attempt to fetch 4 and 6 instructions per cycle.  With a pipeline of up to 15 stages deep, the number of instructions that can be executed concurrently is well over the 5 mentioned above.  In order to feed such engines, an accurate branch prediction is needed.  It is argued that by the year 2010 branch prediction will become the most limiting factor in processor performance, surpassing even the limitations imposed by memory systems [1].

The branch irony is that the same mechanism that helped computers evolve past the functions of a mere calculator becomes the limiting factor for future generations of computers.

## 1.2 Solutions to the Branch Problem

A number of ways have been devised to overcome the control flow problem imposed by sequential code. Eliminating false control dependencies allows unnecessary stalls to be eliminated [3][4]. Code transformation by compilers to enlarge basic blocks reduces the occurrence of some branches. Loop unrolling, a form of basic block enlargement, is a popular technique employed by compilers to alleviate the cost of branches. Guarded (predicated) execution also allows basic blocks to be enlarged [5][6][7]. However, methods like guarded execution suffer from the need to change the instruction set architecture (ISA), which poses a problem for backward compatibility.

Another group of techniques relies on branch prediction. The machine speculates on the direction of the branch, and then executes the predicted path. One way of doing prediction is to profile the program and then to include a prediction bit in the branch instruction. This is referred to as static branch prediction. It suffers from the need to change the ISA like guarded executing, which is also done by the compiler. Dynamic branch prediction, on the other hand, records the outcome of the previous branches during the run of the program, and based on this statistic, predicts the outcome of the following branches. It has been shown that dynamic branch prediction achieves better performance than other methods [8].

In order to overcome the control dependency imposed by conditional branches using dynamic branch prediction, the direction of the branch and the target address need to be predicted.  In most cases, the target address can be predicted accurately by utilizing a branch target buffer (BTB) — a cache that records the target address during the previous execution of the branch.  A hit in the BTB ensures a good prediction in the case of a direct branch.  The target address of indirect branches is harder to predict, but indirect branches constitute a small portion of the overall branches.  This dissertation concentrates on predicting the branch direction.  Moreover, this dissertation is limited to dynamic branch prediction.  Those dynamic predictors might employ a static method, but pure static predictors are ignored.

## 1.3 Directions in Dynamic Branch Prediction

The first branch prediction schemes were static ones, where the branch prediction was hard-coded within the processor.  The need for better branch prediction led to dynamic branch predictors, where branch prediction is determined by examining past behavior of the running program.  The introduction of the bimodal structure was one of the first to utilize dynamic branch prediction, and it put the field of dynamic branch prediction on the research map [35][36].  Most processors in the past few years have contained a dynamic branch prediction module.

The introduction of two-level dynamic branch prediction [61][22] was a major step in the advancement of dynamic branch predictors.  From that point, research in the field has taken four different paths[1]. The first path attempted to improve on the two-level branch prediction scheme by incorporating different kinds of branch-related information into the dynamically collected statistics that decide the prediction. This will be referred to here as the ‘scheme path’. Once it was understood that aliasing presented a major hurdle to correct prediction, numerous branch prediction structures that alleviate the aliasing problem were conceived. The second path of research we will refer to here as the ‘aliasing path.’ The third path, the ‘hybrid path,’ is based on the observation that different branches are best predicted by different kind of predictors. Predicting each branch with its respective best predictor, should enhance prediction accuracy. The fourth path is the one least studied. It involves a ‘third-level of adaptivity’. This claims that different branches are better predicted by different configurations of the same branch prediction scheme, or that different phases of the program are better predicted by different configurations of the same branch prediction scheme. We termed this path the ‘third-level’ path, and we note that some papers also claim that the third-level path helps prediction by reducing aliasing.

The different branch prediction research paths have been kept separate in most cases. This is apparent in the lack of comparison between the different structures. For example, the *agree* [9] predictor which is designed to reduce aliasing was never compared to the

[1] Using value prediction to predict branches can be viewed as a fifth path that research took, or it can be conceived of as part of the scheme path. In any case, in preliminary studies not presented here, we learned that most branches predicted well by incorporating value prediction are predicted just as well by other known predictors. We have therefore chosen not to address branch prediction using value prediction in this dissertation.

McFarling hybrid predictor [14]. The hybrid and aliasing paths were considered to be orthogonal, and a hybrid predictor, where each component reduces aliasing could be easily devised. Another example is the *bi-mode* predictor. When it was introduced, there was no mention of the classification method (discussed later) despite the striking structural similarity between the *bi-mode* and classification method. As a result of this line of thinking, there is very little knowledge regarding the interaction between the different paths.

Another problem becomes apparent when, in the rush to publish new branch prediction structures, researchers often failed to understand the reasons why the branch prediction structure worked. Instead, empirical results showing the superiority of the branch predictor have often been presented. These publications often lacked a simple limit study, which would have helped explain the underlying reasons why the branch prediction structure worked well. This omission could lead microarchitects to make poor choices of the branch predictor structure needed for a microprocessor.

## 1.4 Thesis Statement

The need for accurate branch prediction is increasing as processors implement deeper and wider instruction fetching. Understanding why known branch predictor structures work is essential to the decision-making process of the micro-architect. It is also important to find feasible solutions to the branch prediction problem without ignoring constraints imposed by the underlying technology.

This dissertation presents a series of studies aimed at understanding why the different paths taken by dynamic branch prediction work, and what sorts of interaction have existed between those paths. Presented here are results that show an unexpected amount of consolidates among the different paths taken in branch prediction. Those results, combined with a detailed analysis of previous studies and a look at trends in the underlying technology, lead to a set of criteria that produce an ideal model for a two-level dynamic branch prediction structure. Using those criteria, a new dynamic branch prediction structure is constructed that outperforms other known predictors from the other three paths. The microarchitect is presented with different configurations of the new predictor that fit different architectures and constraints.

## 1.5 Contributions of This Dissertation

This dissertation makes several contributions to the field of branch prediction. First, it consolidates the hybrid and aliasing research paths in branch prediction by showing that most of the advantage gained in combining branch predictors is due to the selection mechanism's ability to reduce aliasing. The myth that a branch changes its best predictor during the execution of a program is refuted.

Second, this dissertation shows that a dynamic and a properly profiled static selection mechanism in hybrid predictors work well for the same main reasons. They both reduce aliasing. The prevailing factor in increasing prediction accuracy is aliasing reduction. The advantages and disadvantages of static and dynamic selection mechanisms are highlighted.

This dissertation also consolidates the third-level and aliasing research paths in the branch prediction field. Showing that most of the advantage gained by the third-level branch prediction structures is due to filtering, the third-level path is reduced to the aliasing path. An important observation made here is that the same advantage depicted in the third-level path can be gained by picking the best history size configuration for each benchmark.

The lessons learned in this dissertation combined with a thorough analysis of the advantages and disadvantages of previously proposed branch prediction structures are used to draw a set of criteria that branch prediction structures should follow.

Drawing on this proposed set of criteria, a new branch prediction structures is proposed – YAGS. Utilizing the set of criteria allows YAGS to provide a significant performance improvement over existing structures at modest cost. A comparison between YAGS and previously proposed structures is presented.

A profile version of YAGS is introduced. This version makes better use of resources by allowing the branch bias to be determined statically, but might require some ISA change for certain architectures. Arguably, the best attribute of the profile version of YAGS is the ability to use it as a cascading predictor. A cascading predictor supplies a prediction in one cycle and a more accurate prediction after two cycles.

This thesis stresses the importance of a relevant limit study for research done on branch prediction. Most of the misconceptions/myths revealed in this dissertation resulted directly from the lack of a relevant limit study.

## 1.6 Organization

This dissertation is organized as follows: Chapter 2 elaborates on the four different paths discussed above, and walks the reader through previous work performed in each path, highlighting the pros and cons of each method.  Chapter 3 discusses the experimental methodology and benchmarks used in the studies.

The next 3 chapters are each dedicated to one of the four paths discussed above.  Chapter 4 presents a limit study on the different schemes belonging to the scheme path.  Chapter 5 investigates what makes a hybrid predictor work well and evaluates the benefits of incorporating a hybrid predictor with a structure to reduce aliasing.  Finally, Chapter 6 investigates the possibilities and limitations of the third-level path.

The remainder of the dissertation capitalizes on the conclusion of the previous chapters. First, Chapter 7 introduces a trend in micro-architecture that has generally been ignored within the branch prediction research community.  Utilizing previous work and studies done earlier chapters, Chapter 8 summarizes the criteria necessary for a good dynamic branch predictor.  This chapter goes on to introduce a predictor that capitalizes on these criteria to produce a better prediction compared to previous known predictors.  Chapter 9 provides a summary and possible future work.

# Chapter 2 - Previous Work

## 2.1 Prediction Schemes

4This section walks through previous work done on the scheme path. While the first two subsections discuss one level rather than two level dynamic branch prediction schemes, they provide a foundation for the two-level branch prediction schemes discussed in the rest of this chapter.

### 2.1.1 Bimodal

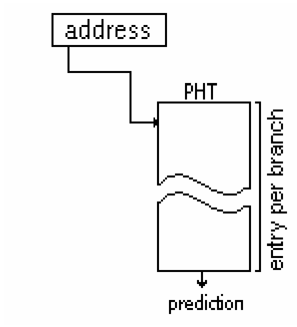


**Figure 2.1 - Diagram for the bimodal Scheme**

A table of two bit saturating counters (2bc) called a pattern history table (PHT), indexed by the branch address, was proposed early in the history of branch prediction research field [35][36]. This was one of the earliest dynamic schemes, and was later referred to as the bimodal scheme (Figure 2.1)[2]. The 2bc became the standard state machine and the bimodal branch predictor is frequently used as a "lower bound" branch prediction benchmark against which to judge other branch prediction structures. In other words, a branch prediction scheme should not, under any circumstances, perform worse than the bimodal scheme. The bimodal attempts to predict the direction of a branch according to the past behavior of that branch during program execution. The 2bc

[2] The Figures in Section 2.1 show a diagram for the branch prediction scheme, although those schemes were introduced by specific branch prediction structures (see sec 4.1 for more).

provides some hysteresis so that one spurious prediction does not alter the next prediction. The branch should behave the same at least two times consecutively in order for the prediction to change. The bimodal can be seen as capturing the dynamic bias of the branch.

### 2.1.2 History Only Branch Predictor

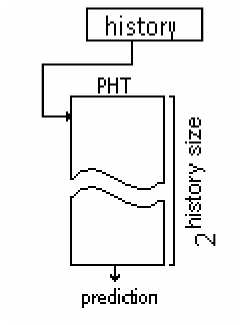


**Figure 2.2 Diagram for the history scheme**

A special case of the global branch prediction scheme is the history branch prediction scheme [44] (Figure 2.2). A table of 2bcs, indexed by a global history register, provides the prediction. History branch prediction schemes assume that a correlation exists between the last *n* branches and the current branch. Since the branch address is not involved in determining the prediction, the assumption is that the correlation works regardless of which branch is involved. In other words, if the *n* branches preceding branch A and branch B behave the same, branches A and B will behave the same as well.

### 2.1.3 Two-level Adaptive Branch Predictors

A major milestone in the branch prediction research field was the introduction of the local two-level adaptive branch predictor [22]. It was shown to achieve up to 97% correct prediction accuracy on the early SPEC89 benchmarks. Later analysis has shown that the SPEC89 benchmarks are not hard to predict, even the bimodal predictor achieves over 90% prediction accuracy on the same set of benchmarks. The authors varied the associativity in the history table, and examined different state machines as the predictors in the PHT. This study found that the 2bc state machine performed the best among the state machines tested. It is important to note that since that study, this assertion has not been challenged and the 2bc been accepted as a standard.

Three different classes of two-level adaptive branch predictors were identified [23], and a terminology based on taxonomy was proposed. For example, the term GAg indicates a global history register with a shared (global) PHT. The size of the PHT is $2^{\text{history register size}}$ in this case. PAg indicates a table of history registers indexed by the program counter, where the PHT is shared. In contrast, PAp indicates a table of history registers, each of which has its own PHT. In practice, the PAp scheme can only be realized for very small history sizes. Separate work showed that PAp is the best predictor and GAg is the worst [24]. Notice, however, that GAg consumed the least amount of resources.

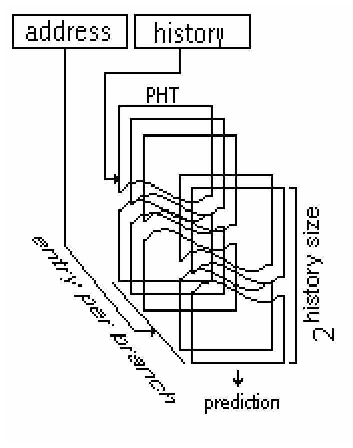


**Figure 2.3 - Diagram of the global scheme**

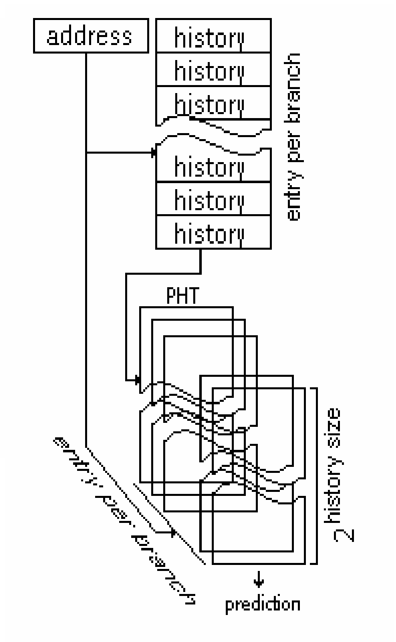


**Figure 2.4 - Diagram for the local scheme**

Today, the common opinion is that the global family of branch prediction can offer a better prediction accuracy than the local branch predictors for an integer workload because the branches in integer workloads tend to be highly correlated [25]. On the other hand, the local family of branch predictors offers better prediction accuracy for scientific workloads.

### 2.1.4 Global Two-level Branch Predictors

The global two-level branch prediction scheme depicted in Figure 2.3 attempts to predict the branch based on the pattern of outcomes of the *n* preceding branches. When the program has a lot of if-then-else statements, the results are usually good. When the global branch prediction scheme was introduced arguments about program behavior, and snippets of high-level languages code were used to justify its merit [44]. In trace driven simulations it was shown that an implementation of the global branch prediction scheme performed better than a bimodal scheme implementation for the same amount of resources.

### 2.1.5 Local Two-level Branch Predictors

The local two-level branch prediction scheme shown in Figure 2.4 attempts to predict a specific branch according to the last *n* preceding outcomes of the predicted branch. A common notion is that local schemes are better than global schemes at predicting branches in scientific code. This is attributable to the presence of a large number of loops in scientific code. Having a per branch history register is beneficial for loop constructs.

### 2.1.6 Path-Based Branch Predictors

The correlated schemes described thus far record the branch outcome in the history register. The information reflecting which branches resulted in those outcomes, is therefore lost. The inclusion of this information might be beneficial for prediction accuracy. If the last *n* branches preceding branch *A* resulted in a certain pattern, it is not necessarily the case that when a different set of branches precedes branch *A* form the same pattern, branch *A* will behave the same.

To rectify this loss of information problem it was suggested that the addresses along the path leading to the branch be factored into the information stored in the history register [26]. Using the branch address path explicitly captures information about the addresses of the branches leading to the one being predicted, and implicitly captures the outcomes of the branches on that path as well. The mechanism proposed is a static mechanism, which is performed by software.

The next development was a dynamic path-based branch prediction mechanism [26]. This structure is similar to the global two-level branch prediction structure. It was observed that when a branch target address falls inside the branch's basic block, the branch outcome is lost in the history register, because the path leading to the branch is identical whether or not the branch was taken. This led to the idea of using the branch target address, instead of the branch address, as the information stored inside the history register.

One weakness of path-based correlation in dynamic branch predictors is that the history register needs to hold a lot of information, typically a word per branch, much more than the one bit per branch of competing schemes. Since the most important information are the least significant bits (LSBs) of the branch address, only a small portion of the address is pushed into the history register.

Path-based prediction schemes resulted in very similar prediction accuracy as did global two-level branch prediction structures of the same size. However, it was noted that the path-based branch prediction scheme, while requiring about the same amount of resources, used less branches for the history than the global branch prediction structure.

### 2.1.7 Other Schemes

As a result of the work mentioned so far researchers observed that capturing more branch-related information improves the prediction potential of the branch prediction scheme. An attempt was made to identify the branch by the branch address, global history, and path-based history [46]. Information related to the above is “exclusive-or” (xor) together and used as the index to the PHT. A slightly better prediction accuracy was accomplished than the *gshare* scheme. No limit study was performed to assess the potential of such a scheme, and the gain in performance is so minute that it could be due to experimental error rather to the inherent capability of the prediction scheme.

### 2.1.8 Summary

A trend was established that the more information that is recorded about a branch to distinguish it from other branches, the better the prediction accuracy that will be achieved for that branch. However, more information entails more hardware dedicated to the branch prediction structure. Therefore, it might be the case that the best branch prediction scheme available does not necessary result in the most cost-effective branch prediction structure.

## 2.2 Aliasing in Global Predictors

### 2.2.1 The Problem

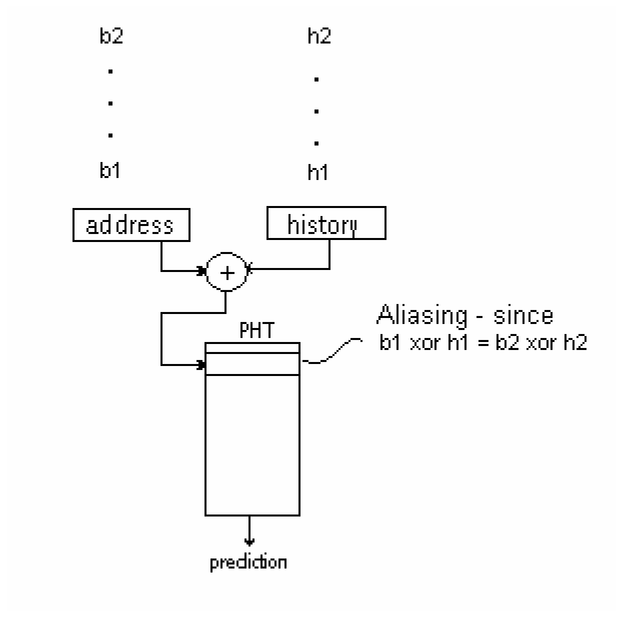


**Figure 2.5 – Aliasing in the gshare predictor**

The main problem that causes prediction degradation in global branch prediction structures is aliasing [15][16] (Figure 2.5). Aliasing occurs when two indices, typically formed from history and address bits, map to the same entry in the PHT. Since the information stored in the PHT entries is either “taken” or “not taken,” two aliased indices whose corresponding information is the same, will not result in mispredictions. We refer to this as neutral aliasing. On the other hand, two aliased indices with contradictory entries might interfere with each other and result in a misprediction. We call this destructive aliasing.

### 2.2.2 Aliasing Reducing Branch Prediction Structures

A lot of work has been done to reduce aliasing in the PHT. In what follows, we describe some of the more notable structures and highlight their strengths and weaknesses.

#### 2.2.2.1 Gshare

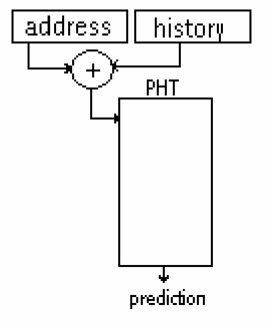


**Figure 2.6 – Diagram for the gshare structure**

The first structure to address the aliasing problem in two-level adaptive branch predictors was *gshare* [14] (Figure 2.6). The observation that the usage of the PHT entries is not uniform when indexed by concatenations of the global history and the branch address, led to idea of using the xor function instead of concatenation to more evenly use the entries in the PHT. Moreover, the usage of the xor function enables more history bits to be incorporated into the prediction and as a result, enables the predictor to increase its correlation. Detailed studies have shown that this yields a slight advantage [19].

### 2.2.2.2 Agree Predictor

The *agree* predictor displayed in Figure 2.7 assigns a biasing bit to each branch in the BTB according to the branch direction just before it is written into the BTB [9]. The PHT information is then changed from “taken” or “not taken” to “agree” or “disagree” with the prediction of the biasing bit. The idea behind the *agree* predictor is that most branches are highly biased to be either taken or not taken and the hope is that the first time a branch is introduced into the BTB it will exhibit its biased behavior. If this is the case, most entries in the PHT will “agree,” so that if aliasing does occur it will more likely be neutral aliasing, which will not result in a misprediction. This observation suggests redundancy in the PHT.

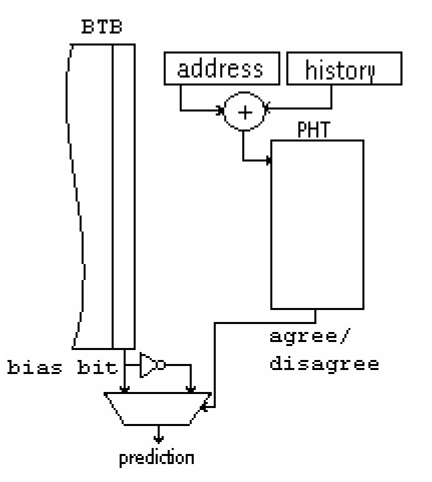


**Figure 2.7 – Diagram for the agree structure**

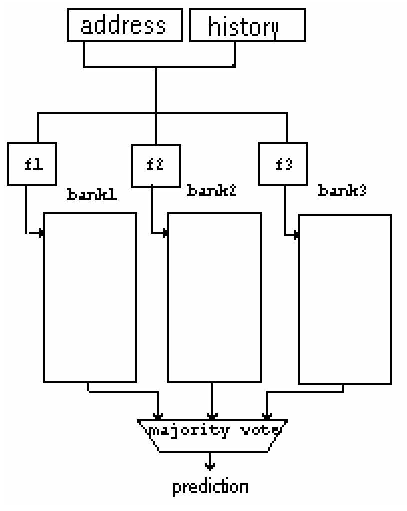


**Figure 2.8 – Diagram for the skew structure**

A patent registered by HP [67] preceded the *agree* predictor in taking advantage of a branch's biased behavior to reduce destructive aliasing by replacing destructive aliasing with neutral aliasing. The *agree* predictor considerably reduces destructive aliasing. However, there is no guarantee that the first time a branch is introduced to the BTB its behavior will correspond to its bias. When such cases occur, the biasing bit will stay the same until the branch is replaced in the BTB by a different branch. Meanwhile, it will pollute the PHT with "disagree" information. Also, there is still aliasing occurring between instances of a branch that do not comply with the bias, and instances where the branch does comply with the bias. When a branch is not cached in the BTB, no prediction is available.

#### 2.2.2.3 Skew Predictor

The *skew* branch predictor seen in Figure 2.8 is based on the observation that most aliasing occurs not because of a small PHT size, but because of a lack of associativity in the PHT. In other words, the major contributor to aliasing is conflict aliasing and not capacity aliasing. The best way to deal with conflict aliasing is to make the PHT set-associative, but this requires tags and is not cost-effective. Instead, the *skew* predictor emulates associativity using a special skewing function [11].

The *skew* branch predictor splits the PHT into three equal banks and hashes each index to 2bc in each bank using a unique hashing function per bank (f1, f2 and f3). The prediction is made according to a majority vote among the three banks. If the prediction is wrong all three banks are updated. If the prediction is correct, however, partial updating will occur, which means that only the banks that made a correct prediction will be updated.

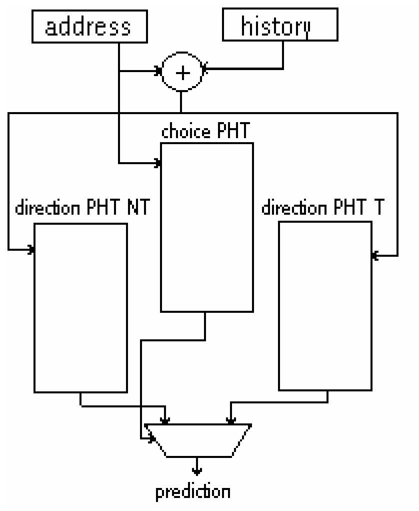


**Figure 2.9 – Diagram for the bi-mode structure**

The skewing function should have inter-bank dispersion. This is necessary in order to make sure that if a branch is aliased in one bank, it will not be aliased in the other two banks. This ensures that the majority vote will produce a un-aliased prediction. The reasoning behind partial updating is that if a bank gives a misprediction when the other two give correct predictions, the bank with the misprediction probably holds information belonging to a different branch. In order to maintain the accuracy of the other branch, this bank is not updated.

The *skew* branch predictor tries to eliminate all instances of aliasing and thus all destructive aliasing. Unlike the other methods, it tries to eliminate destructive aliasing between branch instances that obey the bias and those that do not. However, to achieve this, the *skew* predictor stores each branch outcome in two or three banks. This redundancy of 1/3 to 2/3 of the PHT size creates capacity aliasing by putting more information in the PHT, but eliminates by a greater degree conflict aliasing, resulting in a lower misprediction rate. However, the increase in size slows warm-up on context switches.

#### 2.2.2.4 Bi-Mode Predictor

The *bi-mode* predictor shown in Figure 2.9, similar to the *agree* predictor, replaces destructive aliasing with neutral aliasing [12]. The *bi-mode* PHT gets split into three even parts. One of the parts is the choice PHT, which is just a bimodal predictor (an array of 2bcs) with a slight change in the updating procedure. The other two parts are direction PHTs; one is a “taken” direction PHT and the other is a “not taken” direction PHT. The direction PHTs are indexed by the branch address xored with the global history. When a branch is present, its address points to the choice PHT entry, which in turn chooses between the “taken” and “not taken” direction PHTs. The prediction of the direction PHT chosen by the choice PHT serves as the prediction. Only the direction PHT chosen by the choice PHT is updated. The choice PHT is normally updated too, but not when it gives a prediction that contradicts the branch outcome and the direction PHT chosen gives the correct prediction.

During operation, branches that are biased to be taken will have their predictions in the “taken” direction PHT, and branches that are biased not to be taken, will have their predictions in the “not taken” prediction PHT. So at any given time most of the information stored in the “taken” direction PHT entries is “taken” and any aliasing is more likely not to be destructive. The same phenomenon happens in the “not taken” direction PHT. The choice PHT serves to dynamically choose the branches’ biases.

In contrast to the *agree* predictor, if the bias is incorrectly chosen the first time the branch is introduced to the BTB, it is not bound to stay that way while the branch is in the BTB and pollute the direction PHTs with destructive aliasing. It should be noted, however, that the choice PHT takes a third of all PHT resources just to dynamically determine the bias. It also fails to solve the aliasing problem between instances of a branch that do not agree with the bias and instances that do, because both are stored in the same direction PHT.

### 2.2.2.5 Filter Mechanisms

Reducing the amount of necessary information stored in the PHT is the main point of *filter* mechanisms [10]. The idea is that highly biased branches can be predicted with high accuracy using just one bit. Easy-to-predict branches are filtered out of the PHT by a combination of a bias bit and a saturating counter for each BTB entry, which can be seen in Figure 2.10. When a branch is introduced to the BTB, the bias bit is set to the direction of the branch when it is resolved and the counter is initialized. When every branch instance is resolved, if the direction of the branch is the same as the bias bit, the counter is incremented. If not, the counter is zeroed and the bias bit is toggled. A branch is predicted using the PHT if the counter is not saturated. If the counter is saturated, it means that the branch is highly biased in the direction indicated by the bias bit, and therefore that the bias bit is used as a prediction. In this case, when the counter is saturated, the PHT is not updated with the branch outcome – the saturated counter filters this information from the PHT. The size of the counter has to be tuned to the size of the PHT. If the PHT size is large, the amount of filtering needed is small, and therefore the size of the counters should be

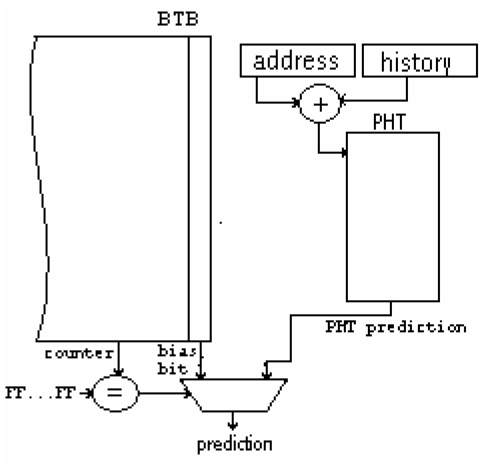


**Figure 2.10 – Diagram for the filter mechanism structure**

large. When a branch is first introduced in the BTB, the counter is initialized. It was found that it is best to initialize the counter to its maximum value so that the filtering will start to work immediately. If the branch is not highly biased, the bias bit will flip fairly quickly and the counter will be zeroed. On the other hand, if the counter is initialized to zero and the branch is highly biased, it will take time for the filtering mechanism to start working and the PHT will be polluted in the meantime.

The *filter* mechanism attempts to eliminate all aliasing instances by considerably reducing the amount of information stored in the PHT. However, this mechanism has difficulty predicting instances of highly biased branches, which do not comply with the bias. Due to filtering, as the PHT size increases the predictor will never reach the full potential of the global scheme that it implements.

### 2.2.3 Summary

The branch prediction structures discussed above use three techniques to reduce aliasing. The first takes advantage of the underlying information stored in the PHT, and converts destructive aliasing to neutral aliasing as a means of improving prediction. The second method to remove aliasing is associativity. Classical associativity (by an inclusion of tags) was determined as not cost-effective. Pseudo associativity, a different way to achieve the same effect, was devised. Third, filtering information selectively allocates greater resources for the more important information.

Those advantages come at a cost. Most of the structures carry some redundancy. For example in the *skew* predictor, the same information is stored in up to three different counters. This redundancy can exacerbate the negative effects of a cold start. In all the above branch prediction structures, certain types of aliasing are neglected or some branches' accuracy is jeopardized.

## 2.3 Hybrid Predictors

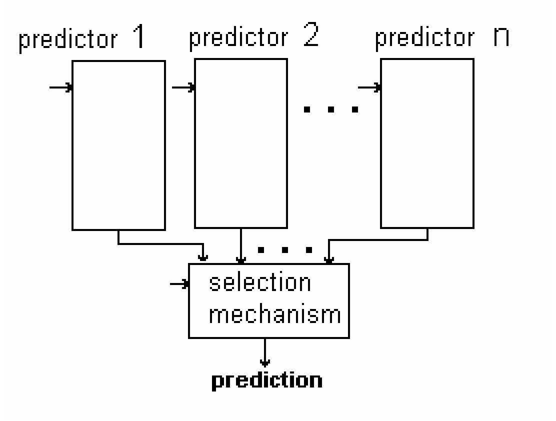


**Figure 2.11 – Diagram of general Hybrid Structure**

The notion that a certain kind of predictor better predicts one class of branches, while a different kind of predictor better predicts a different class of branches led to the idea of combining branch predictors. This class of branch predictors are known as the hybrid branch predictors.

Figure 2.11 depicts a general drawing of a hybrid branch predictor. A selection mechanism is used to choose between two or more branch predictor structures to be used for the prediction of a specific branch instance.

### 2.3.1 Hybrid Branch Predictors

The first hybrid structure suggested combining the *bimodal* and *gshare* structures [14]. The selection mechanism is a table of 2bcs and is very similar to a *bimodal* structure. The selection counter is updated only if the prediction given by the two predictors is different. If only the *bimodal* predictor gives a correct prediction, the counter is decremented. The counter is incremented if only the *gshare* structure gives a correct prediction. If both predictors either give a correct or an incorrect prediction, the selection counter is not updated. Consequently, state 0 and 1 of the 2bc state machine entails a selection of the *bimodal* predictor, while state 2 and 3 result in the use of the *gshare* structure for prediction purposes.  This hybrid structure was shown to outperform all other single schemes known at that time.

It is interesting to observe the percentage of times each predictor was used.  For most benchmarks, the *bimodal* was used significantly more than the *gshare* scheme.  This might indicate that the this hybrid predictor performs well because it filters the easy-to-predict branches out of the *gshare* structure, and not due to the fact that each of its components better predict a different class of branches.  This way, a small amount of resources are used to predict the easy-to-predict branches, leaving the majority of resources to predict the hard to predict branches.

The same study proposed that a hybrid predictor combine *gshare* and PAs structures.  This hybrid predictor is known as the McFarling predictor, named after the author of the paper.  The combination of global and local schemes outperformed the bimodal-*gshare* hybrid predictor only for predictors larger than 16KB.  The simulations were done on the SPEC89 benchmark suite that is notorious for a small branch signature.  A branch classification method was suggested to enable a branch to be predicted by a predictor best suited to predict it [13].  Branch classification was based on the observation that branches that are highly biased can be predicted well with a short history predictor, while the rest of the branches typically require a longer history.  This observation led to a combination of predictors with different history lengths.  The classification predictor outperformed the *gshare* scheme.  The selection mechanism for the classification predictor is done via profiling.

The classification predictor does not clearly belonging to just one of the research paths.  It can be seen instead as a hybrid predictor, and is therefore discussed in this section, but can also be seen as belonging to the ‘third-level’ path. The two components and a selection mechanism associate it with the hybrid path. On the other hand, the two different correlation depths of the same branch prediction scheme associate it with the third-level path.

As an alternative selection mechanism to the *bimodal* structure, the two-level structure was proposed [48]. The hybrid predictor under examination was the McFarling predictor. The assumption was that since the two-level branch predictor could better predict the direction of branches, it would also be better able to select between the different branch prediction structures of the hybrid predictor. It was shown that using a two-level global structure to select between the local and global schemes yielded a very small improvement. However, this was not shown to be cost-effective, and the results were far from the ideal oracle selection mechanism.

A conglomeration of predictors was incorporated into the multi-hybrid predictor [49]. The multi-hybrid consists of the bimodal, two variations of the global predictor, and two variations of the local predictor, a loop predictor, and a static predictor. It was shown to have a slightly better prediction accuracy than the bimodal-global and the McFarling hybrid predictors when tested under context switching. No explanation was given as to why those particular predictors were chosen, or why there was a need for more than one global and local predictor. The comparison was not done against the original McFarling predictor, but rather against a revised version of it, where the selection mechanism was tied to the BTB. As the size of the predictor grew, the selection mechanism size could not grow because it was tied to the BTB. This gave an unfair advantage to the multi-hybrid predictor because its elaborate selection mechanism had to be tied to the BTB.

### 2.3.2 Selection Mechanisms

Selection mechanisms for hybrid branch predictors followed the same line of development as did single scheme branch predictors. The classes consist of a static selection mechanism, a dynamic per branch selection mechanism, and finally, a two-level selection mechanism as discussed above.

Possibly the only contribution of the multi-hybrid predictor is its selection mechanism [49]. Before the multi-hybrid hybrid predictors consisted of only two separate branch predictor components. The multi-hybrid consists of a selection mechanism that can select between an arbitrary numbers of predictors.

This selection mechanism consists of multiple 2bc per entry. The exact number of 2bc is determined by the different components of the multi-hybrid predictor. The predictor to be used is determined by the 2bc with the value of three in it. If multiple 2bcs in the selector entry have the value three, a priority encoder is used to determine which predictor to use. Once the branch is resolved, the 2bc, which corresponds to the predictors giving a correct prediction, is incremented. If one of the predictors, which had the value of 3, was correct, all 2bc that would correspond to all other predictors are decremented.

## 2.4 Third-level of Adaptivity

It was suggested that having the depth of correlation (i.e. the size of the BHR in the global scheme) adapt to the program execution or branch behavior could improve branch prediction. This observation spawned the third-level of adaptivity path.

### 2.4.1 Third-level of Adaptivity Structures

The first structure suggested was the Elastic History Buffer (EHB) [20]. The EHB took branch classification [13] to a finer granularity. Instead of having the option of choosing between two lengths of history register, the EHB gave the option for each branch to use its optimal history length. Moreover, it facilitated filtering of some easy-to-predict branches from the PHT, by using a profiled bias bit instead of the PHT. Filtering the easy-to-predict branches from the PHT reduces aliasing, which in turn increases prediction accuracy. Profiling determines the history length to use for each branch, and requires a modification to the ISA. As in all profiling, there is no guarantee that the data collected during profiling is representative of the actual branch behavior during execution. The EHB structure operates under the assumption that there is an optimal history size per branch without investigating the possibility that a branch could have a different optimal history size in different phases of the program execution.

The Dynamic History Length Fitting (DHLF) dynamically determines the size of the history size used [21]. DHLF divides the dynamic stream of branches into sub-streams termed steps of several thousands instructions. In every other step, the length of the history register is evaluated and might change if the evaluation method finds the change beneficial. The evaluation is done only every other step to omit the effects of cold starts from getting in the way of the evaluation method. The step was set to 16K branch instructions.

In another development, a similar idea was entertained, but instead of using branch outcomes, the variable length path branch predictor used target address in the history register [54]. Profiling was used to determine how much history to use for each static branch.  The predictor was shown to be especially useful with indirect branches, but it was not compared against the EHB [20].

### 2.4.2 Selection Mechanisms

Similar to selection mechanisms present in hybrid predictors, selection mechanisms for third-level of adaptivity can be divided into dynamic and static selection mechanisms. While a hybrid static selection mechanism usually only needs one bit of information in the ISA's branch instructions, the presence of third-level of adaptivity requires log2 bits of the BHR size.

Dynamic selection mechanism was not attempted on per branch granularity as in the hybrid dynamic selection mechanism.  Instead, the dynamic BHR size is on program granularity and is examined and changed every certain number of instructions [21].

# Chapter 3 - Experimental Methodology and Benchmark Description

## 3.1 Experimental Methodology

4Throughout this work, trace driven simulations have been used to evaluate different branch prediction schemes and structures. For simplicity's sake, most simulations predict and resolve a branch, and update the branch predictor before fetching the next branch. Although this approach sacrifices some accuracy because not always the branch outcome can be used to update the history register before the next branch is fetched and predicted, studies have shown that such simulations provide a tight estimation to finer, cycle level, simulations [55][56].

| Scheme | Description | Design Space | Unlimited Size |
|---|---|---|---|
| **Bimodal** | A table of 2bc accessed by the branch address | None | # branches |
| **History** | A table of 2bc accessed by a global history register | History size from 1 to 64 | $2^{\text{history size}}$ |
| **Global** | A table of 2bc accessed by the branch address and a global history register | History size from 1 to 64 | # branches x $2^{\text{history size}}$ |
| **Local** | A table of history registers accessed by the branch address. The result is used with the branch address to access a table of 2bc. | History size from 1 to 64 | Number of branches x ($2^{\text{history size}}$ + # branches) |
| **Path-Branch** | Same as global but the information in the history register is previous branch addresses and not their outcomes | History size from 1 to 64 | # branches x $2^{\text{history size x word size}}$ |
| **Path-Target** | Same as global but the information in the history register is previous branch target addresses and not their outcomes | History size from 1 to 64 | # branches x $2^{\text{history size x word size}}$ |

**Table 3.1 – Branch Prediction Schemes and Their Attributes**

All through this dissertation extensive limit studies of branch prediction schemes and combinations of multiple schemes were performed. Table 3.1 depicts the different branch prediction schemes considered in the course of this work. Table 3.1 gives a short description, the design space, and the size of the predictor if no size restriction is imposed for each branch prediction scheme.

```
typedef struct hybridStruct {
            unsigned long long globalPred;
            unsigned long long localPred;
            unsigned long long pathBranchPred;
            unsigned long long pathTargetPred;
            unsigned long long historyPred;
            unsigned int bimodalPred;
            unsigned int baddr;
            unsigned int btarget;
            unsigned int taken;
}           hybridElement;
```

**Figure 3.1 – Hybrid Trace Structure**

To facilitate such processor and memory intensive simulations, a new trace termed the hybrid trace was created for each benchmark. Each entry in the hybrid trace contains predictions for every type of scheme for multiple correlation depth, ranging from 0 to 63. Figure 3.1 depicts the structure used for each entry in the hybrid trace. For example, the 3$^{rd}$ bit in the global variable represents the prediction a global branch prediction scheme made with a correlation depth of 3.

## 3.2 Benchmarks' Description

This dissertation conducted studies on a set of 16 benchmarks. Eight of these are SPECINT95 and six are SPECFP95. Two more of these benchmarks, the s390 and the PowerPC, were provided by IBM. Table 3.2 lists the characteristics of all benchmarks. The SPEC95 benchmark suite represents the typical workload a computer might expect. The IBM traces are of database applications, and are interesting for their large branch footprints compared to the SPEC95 traces.

| | **SPECFP95** | | | |
|---|---|---|---|---|
| | Regular Set | | | Train Set |
| Benchmark | Static Branches | Dynamic Branches | Indirect Branches | Dynamic Branches |
| applu | 1498 | 31,843,665 | 291 | 17,867,895 |
| apsi | 3006 | 41,370,429 | 581 | 126,828,375 |
| fpppp | 1089 | 14,550,247 | 188 | 4,540,419 |
| hydro2d | 2128 | 133,675,998 | 438 | 238,609,181 |
| mgrid | 1449 | 13,901,572 | 274 | 208,359,079 |
| turb3d | 1626 | 52,785,185 | 305 | 238,609,181 |
| | **SPECINT95** | | | |
| | Regular Set | | | Train Set |
| Benchmark | Static Branches | Dynamic Branches | Indirect Branches | Dynamic Branches |
| gcc | 13,763 | 49,193,611 | 3317 | 52,277,032 |
| compress95 | 495 | 196,295,114 | 49 | 6,145,300 |
| go | 7401 | 147,352,115 | 3278 | 80,274,927 |
| ijpeg | 2760 | 71,798,033 | 478 | 173,576,042 |
| li | 1701 | 233,260,230 | 315 | 41,801,717 |
| m88ksim | 1646 | 160,658,276 | 343 | 20,530,078 |
| perl | 3443 | 191,717,635 | 647 | 2,144,594 |
| vortex | 7581 | 158,719,765 | 765 | 238,609,181 |
| | **IBM** | | | |
| | Regular Set | | | Train Set |
| Benchmark | Static Branches | Dynamic Branches | Indirect Branches | Dynamic Branches |
| s390 | 21,727 | 2,360,458 | 631 | 1,360,459 |
| powerpc | 16,710 | 32,497,139 | N/A | 19,000,001 |

**Table 3.2 - Benchmark Characteristics**

Table 3.3 shows the datasets used as inputs for the different benchmarks. Each SPEC95 benchmark has two datasets. The first is used in most simulations, while the test dataset is used to obtain profiling information when appropriate. The IBM benchmarks were provided as traces and without an accompanying test trace. As a result, whenever profiling information was needed, the first half of the trace was used to obtain profiling and the second half was used to obtain simulations statistics. The reader is therefore advised to place less confidence in results that used those traces in studies that include profiling.

| Benchmark | Description | Training Set | Test Set |
|---|---|---|---|
| gcc | GNU C compiler version 2.5.3 | stmt.i | jump.i |
| go | Computer program playing go | short.in | 2stone9.in |
| compress | Data compression program | prof.in | test.in |
| ijpeg | Image compression program | vigo.ppm | speicmun.ppm |
| xlisp | XLISP interpreter | 7queen.lsp | train.lps |
| vortext | Object-Oriented database | vortex.35M | vortes.in |
| M88ksim | Motorola 88100 simulator | dhry.test.big | dcrand.train.big |
| perl | Train interpreter | primes.pl | scrabbl.pl |
| applu | Solves matrix system with pivoting. | | |
| apsi | Calculates statistics on temperature and pollutants in a grid. | | |
| fppp | Performs multi-electron derivatives. | | |
| hydro2 | Hydrodynamical Navier Stokes equations are used to compute galactic jets. | | |
| mgrid | Calculation of a 3D potential field. | | |
| swims390 | Solves shallow water equations | | |
| S390 | N/A | N/A | Not specified |
| powerPC | N/A | N/A | Not specified |

**Table 3.3 – Benchmark description and datasets**

The studies in this dissertation were conducted using the SPEC95 benchmark suite. As the name implies, those benchmarks were available in 1995. A newer version of the SPEC is available – SPEC2000. However, simulations done on the newer benchmarks revealed no indication of harder to predict branches, nor a larger number of static branches. For the most part, it seems that SPEC2000 is just a revised version of the benchmarks present in the SPEC95 suite. We therefore continued conducting the studies in this dissertation with the SPEC95 as we did before the SPEC2000 became available.

## 3.3 Performance Metrics

The studies in this dissertation are evaluated using the metric of branch prediction accuracy. The main disadvantage of this metric is the inability to directly convert improvement in branch prediction accuracy to improvements in overall system performance. Overall improvement in system performance can be better achieved by using a metric like Cycles Per Instruction (CPI). Previous studies have shown that a strong correlation exists between branch prediction accuracy and overall system performance [57][58][59]. As a result, the disadvantage of using branch prediction accuracy as a guide to system performance is minimal.

There are numerous advantages of using branch prediction accuracy as a metric. Using prediction accuracy detaches the evaluation of the branch predictors' performance from system dependent parameters, such as the misprediction penalty. Moreover, it facilitates concentration on improving the branch prediction mechanism without the interference of other potential system bottlenecks, such as cache misses.

In summary, using branch prediction accuracy enables concentration on global factors in branch prediction that will facilitate a better branch predictor in every system.

## 3.4 Results Presentation

This dissertation presents 20 possible graphs: the SPEC95 benchmarks (16 different benchmarks), the PowerPC benchmark, the S390 benchmark, the arithmetic average of the SPECINT95, and the arithmetic average of the SPECFP95.

# Chapter 4 - The Scheme of Schemes

## 4.1 The Difference Between Schemes and Structures

4Past work on branch prediction has failed to distinguish clearly between branch prediction schemes and branch prediction structures—in fact, those words have been used interchangeably. A branch prediction structure is the mechanism that implements the algorithm, which is the branch prediction scheme. For example, the global branch prediction mechanism described in Section 2.2.4 is implemented by many branch prediction structures (Sections 2.2.2.1 – 2.2.2.5). If a branch prediction structure is not limited in resources, it will reach the branch prediction scheme's peak potential. As a result, with no limits on resources, all branch prediction structures implementing the same branch prediction scheme will achieve the same prediction accuracy.

The distinction between branch prediction structures and schemes is instrumental in choosing an appropriate branch predictor. Regardless of how many resources will be dedicated to the *gshare* structure, the *gshare* structure will never surpass the prediction potential of the global branch prediction scheme. It is important to be aware of the global scheme's limits. If a certain branch prediction structure approaches the prediction limits of the branch prediction scheme, Amdahl's law dictates that work should be directed towards finding new and improved branch prediction schemes, as opposed to finding new branch prediction structures that will approach the limits of the branch prediction scheme.

Throughout this dissertation, this convention of distinguishing between branch prediction structures and branch prediction schemes is followed. Next, the limits of known branch prediction schemes are studied.

## 4.2 Limits on Branch Prediction Scheme

Figure 4.1 displays prediction accuracy as a function of correlation depth for four major branch prediction schemes. The four branch prediction schemes are the global scheme, the local scheme, history scheme and the bimodal scheme. All four schemes are discussed in details in Section 2.1. The graphs are presented for a) the SPECINT95 benchmarks, b) the SPECFP95 and the IBM benchmarks and c) the SPEC95 averages. Notice that the y-axis coordinates are not uniform for all graphs. For purposes of clarity, the grid line is held constant at 1% prediction accuracy for easy comparison. Correlation depth applies to the two-level schemes, but it does not apply to the bimodal scheme, which utilizes only one level. The bimodal plot is therefore constant across correlation depth. Because this is a limit study, there is no limit on resources and therefore the graph does not represent resource allocation.

It is clear that the two-level branch prediction schemes are superior to the bimodal scheme. On the other hand, the history scheme, which doesn't make use of the branch address, surpass the bimodal scheme only for large correlation depths. The global and local schemes surpass the bimodal scheme starting with correlation size of one across nearly all benchmarks. It is therefore imperative to use the two-level scheme to achieve high accuracy of branch prediction.

The ability of the history scheme to approach the prediction of the global scheme raises the question of what causes two-level branch prediction to work.  Traditionally it

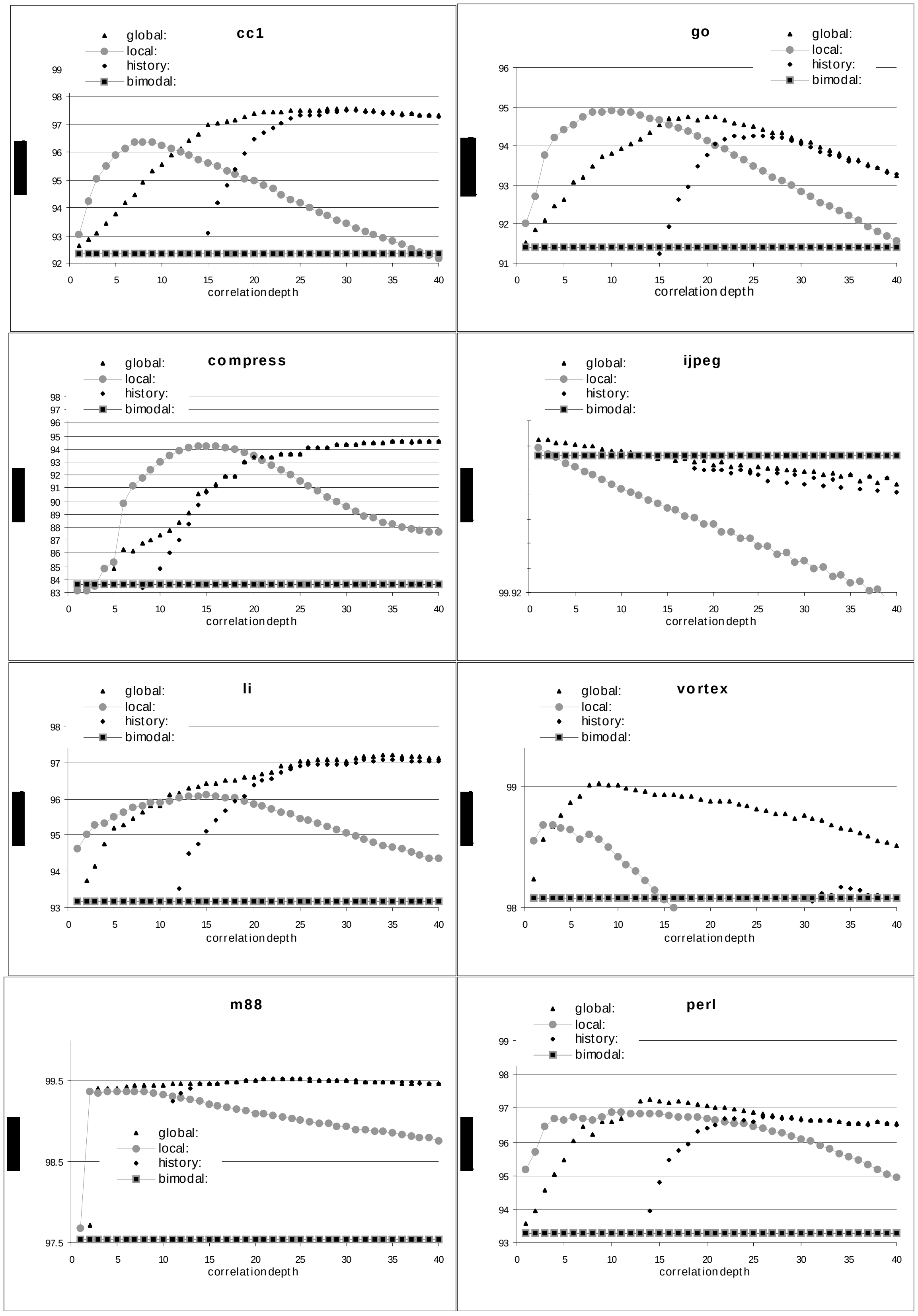


**Figure 4.1 - a) Limits study of common branch prediction schemes for the SPECINT95**

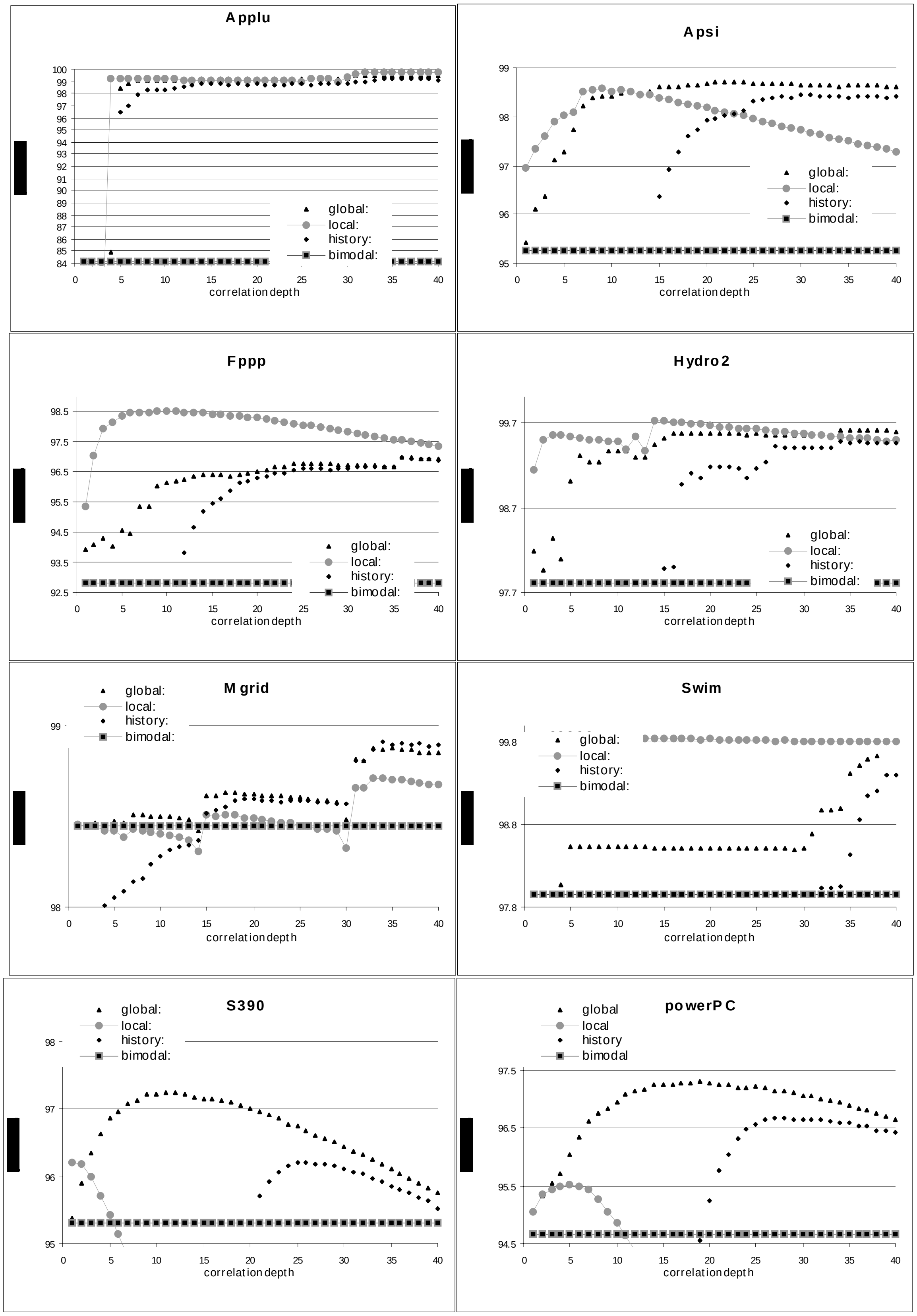


**Figure 4.1 - b) Limits study of common branch prediction schemes for the SPECFP95 and IBM benchmarks**

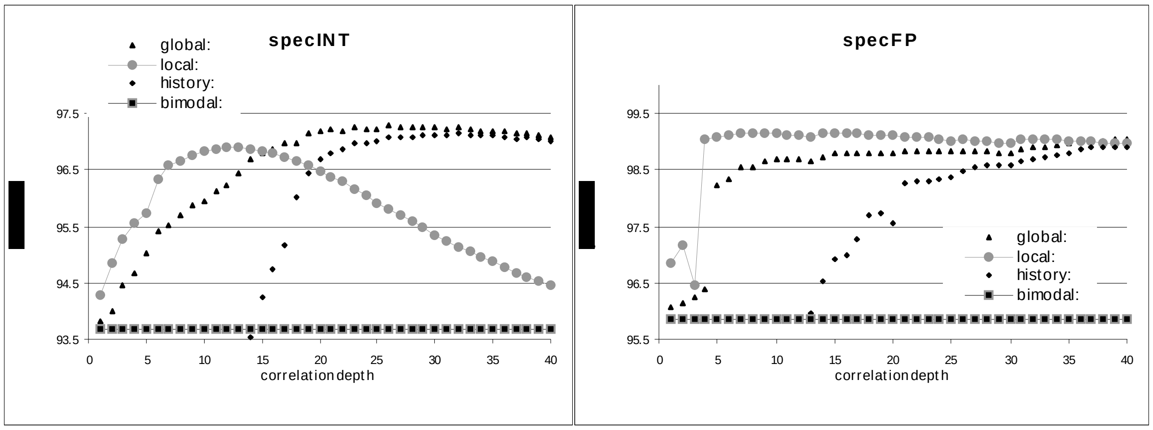


**Figure 4.1 - c) Limits study of common branch prediction schemes for the SPECFP95 and SPECINT95 averages**

has been thought that the two-level branch prediction schemes work well because the prediction of a branch is correlated to either previous branches in the global scheme, or to previous instances of the same branch in the local scheme. This traditional explanation is brought in question in light of ability of the history scheme to outperform the global or local scheme for several benchmarks. Remember that the history scheme applies correlation regardless of which branch is in question. One explanation could be that after a certain correlation is encountered, all branches will tend to have the same behavior, regardless of which branch is predicted and what branches came before it. However, one could easily draw different conclusions from this, and we refrain from fully addressing the topic.

It is believed that the global branch prediction scheme predicts integer programs better than the local branch prediction scheme due to the greater frequency of if-then-else statements that will cause branches to correlate to preceding branches. On the other hand, the local scheme predicts scientific programs better than the global scheme due to the large loop constructs in the program. Loops cause branches to be correlated to previous instances of the same branch. Therefore, the local scheme will outperform the global scheme for scientific code. The averages of the SPEC95 support this conventional wisdom, but it happens only at a correlation depth of 16. Looking at individual benchmarks, on the other hand, this conventional wisdom is not always the case. For example, the go benchmark, which is a prominent integer benchmark for its large branch signature, is better predicted by the local scheme. On the other hand, apsi and mgrid, which are scientific benchmarks, are better predicted by the global scheme.

The limit study presented in Figure 4.1 ignores two major considerations. The first relates to the size of a potential implementation of the scheme. Due to the mount of information stored, the local scheme is more expensive to implement in terms of hardware than the global scheme. For the local scheme correlation needs to be stored for every branch compared to only one correlation register for the global scheme. Of course an actual implementation cannot have a history register for each branch and therefore different branches must share the same history register. This aliasing effect will degrade performance. Second, the limit study ignored the warm-up effect. A static branch prediction scheme takes no time to warm up on a context switch, while the bimodal scheme needs to warm up 2bc per branch. The warm-up effect is aggravated for the global scheme that needs to warm up multiple 2bcs according to the depth of correlation used. In general, the deeper the correlation utilized, the greater the warm up time. The local scheme suffers even further due to the need to warm up the history register per branch. Nevertheless, Figure 4.1 gives an accurate indication of the maximum prediction achievable when implementing a certain branch prediction scheme.

Another notable observation is that every benchmark reaches its peak prediction performance for different correlation depths. Two extreme examples are the ijpeg benchmark that reaches its peak performance for the global scheme at a correlation depth of 2, and the cc1benchmark, which peaks at a at correlation depth of 29. If the microarchitect is able to choose the best correlation depth for each program, an overall better prediction average can be achieved.

In general, we can draw the relationship between correlation and prediction accuracy from Figure 4.1. Increasing the size of the history register increases the correlation depth, which in turn, increases prediction accuracy. This relationship has one caveat— it holds true only until a certain correlation depth is reached. For most programs the depth of correlation where this relationship fails is large enough that it is not likely to be realized in hardware in the near future. Therefore, we can accept this relationship as true. The discussed drop in prediction accuracy is due to cold start effect and for long programs, increases correlation depth will entail increased prediction accuracy for even larger correlation depths than depicted in Figure 4.1.

## 4.3 Global Branch Prediction Schemes

When a branch outcome is saved in the BHR, the predictor can tell whether the last few branches were taken or not, but it cannot distinguish which branches they were. If the branch address rather than the branch outcome is to be pushed into the BHR, as discussed in Section 2.1.6, the predictor will retain this lost information (pathBranch in graphs). There is even some loss of information for the pathBranch scheme. If the target address and the fall-through address both falls in the same basic block, the prediction scheme is unaware whether the branch is taken or not. To solve this problem, the pathTarget scheme pushes the branch target address instead of the branch address to the BHR as discussed in Section 2.1.6.

Figure 4.2 compares the prediction accuracy of pathBranch and pathTarget schemes against the global scheme without the imposition of any resource limitations. As discussed above, the pathBranch scheme captures more information about previous branches than the global scheme, and the pathTarget scheme captures more information than both pathBranch scheme and the global scheme. It is not clear, however, that capturing the extra information always helps prediction accuracy. Comparing pathTarget to pathBranch it becomes clear that in most cases the difference between the two schemes is negligible. Previous studies have claimed that pathTarget is better than path branch because it suffers no loss of information when the branch target address is within the basic block of the branch. However, those studies ignored that pathTarget lost information when two or more branches have the same target address. In a similar scenario pathBranch will not lose information.

The vortex benchmark is the exception in the sense that the pathTarget outperforms the pathBranch scheme. The vortex benchmark demonstrates that the global scheme, while under-performing for small correlation depth, outperforms the pathBranch and pathTarget schemes for larger depth of correlation. Moreover, the global scheme is able to reach the highest prediction accuracy across correlation depth.

The limit study presented in Figure 4.2 pushes 32 bit address entries into the BHR. A real implementation of either pathBranch or pathTarget can only push a few of the address' LSB due to hardware restrictions. This is bound to cause loss of information and degradation in the performance of the pathBranch and pathTarget schemes. Assuming however, that the pathBranch and pathTarget information is not lost even when using as few as 3 LSB of the address, a pathBranch or pathTarget scheme can only

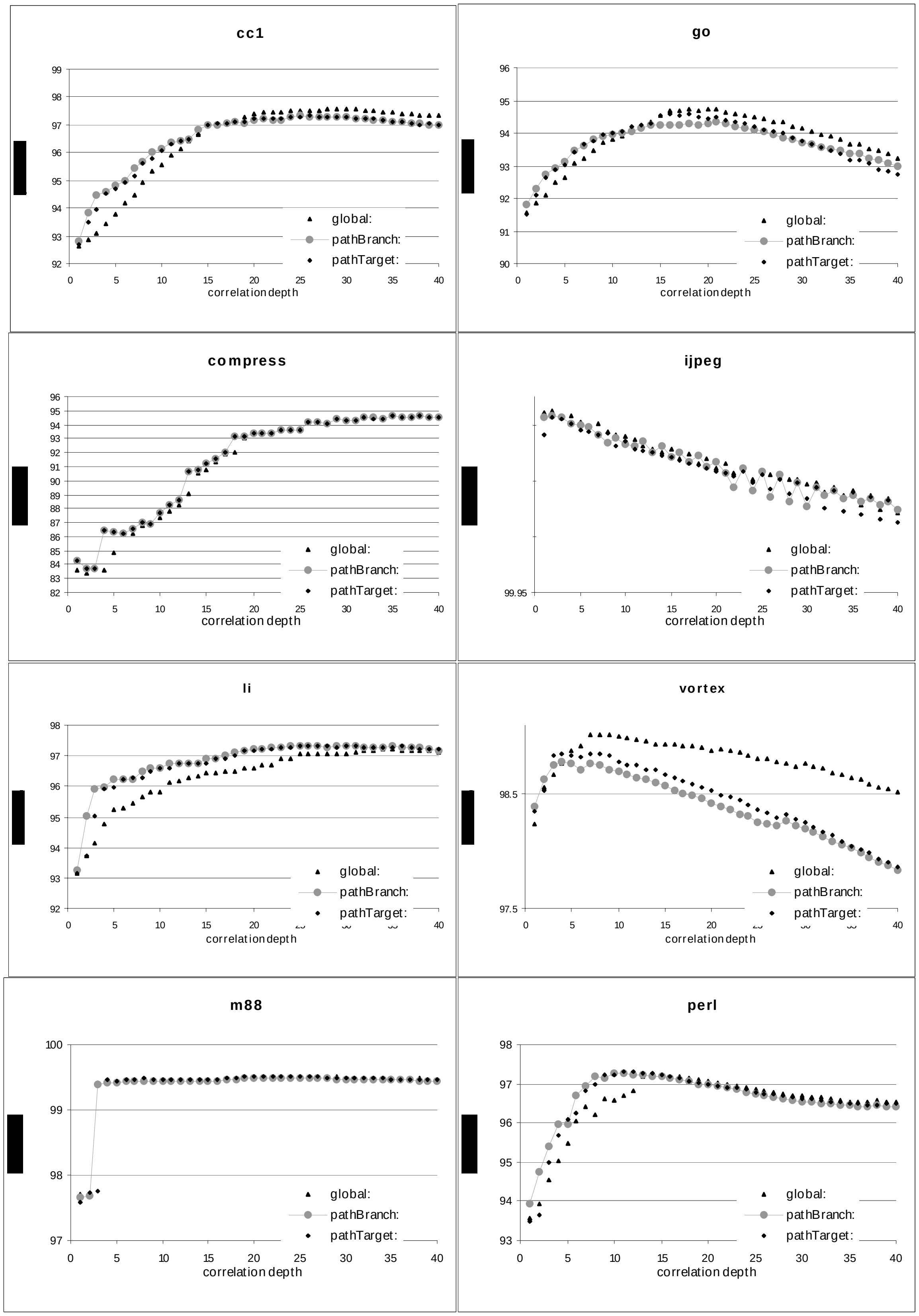


**Figure 4.2 - a) Limits study of global branch prediction schemes for the SPECINT95**

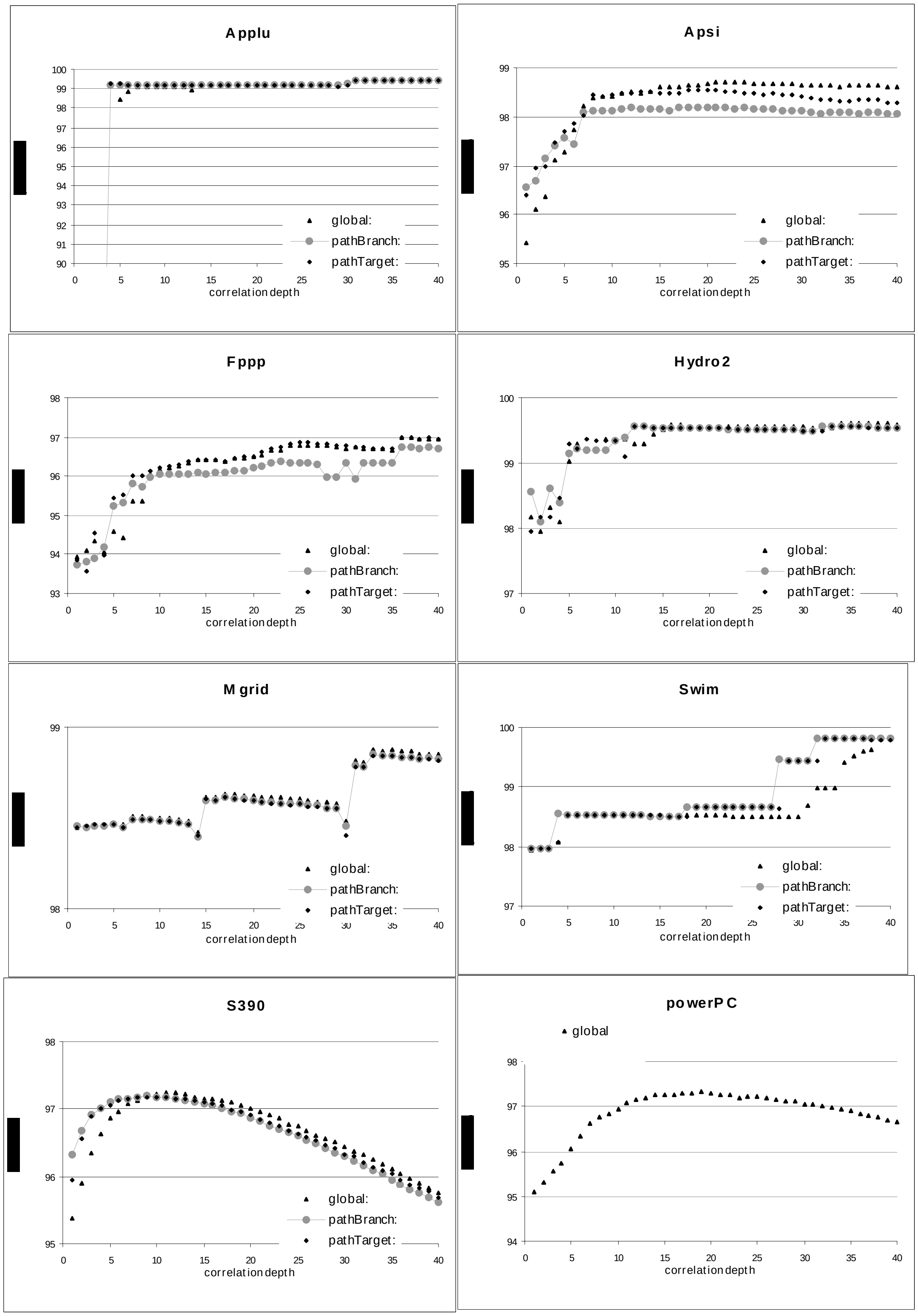


**Figure 4.2 - b) Limits study of global branch prediction schemes for the SPECFP95 and IBM**

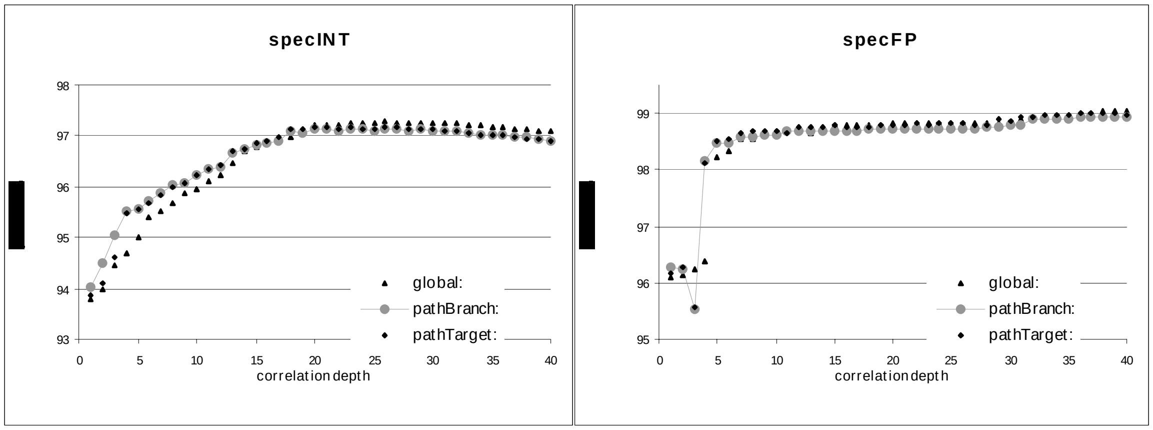


**Figure 4.2 - c) Limits study of global branch prediction schemes for the SPECFP95 and SPECINT95 averages**

use correlation depth of 3 when a 0.5K entry PHT is available. In comparison the global scheme is able to utilize a correlation depth of 9 for the same size PHT. For the same reason, a correlation depth of 4 for the pathBranch and pathTarget is comparable to a correlation depth of 12 in the global scheme, and so on. Figure 4.3 shows the adjusted comparison between the pathBranch and pathTarget schemes to the global scheme for the SPECINT95 average and the SPECFP95 average, assuming there is no loss in prediction for the pathBranch and pathTarget schemes due to the usage of only 3 LSB of the

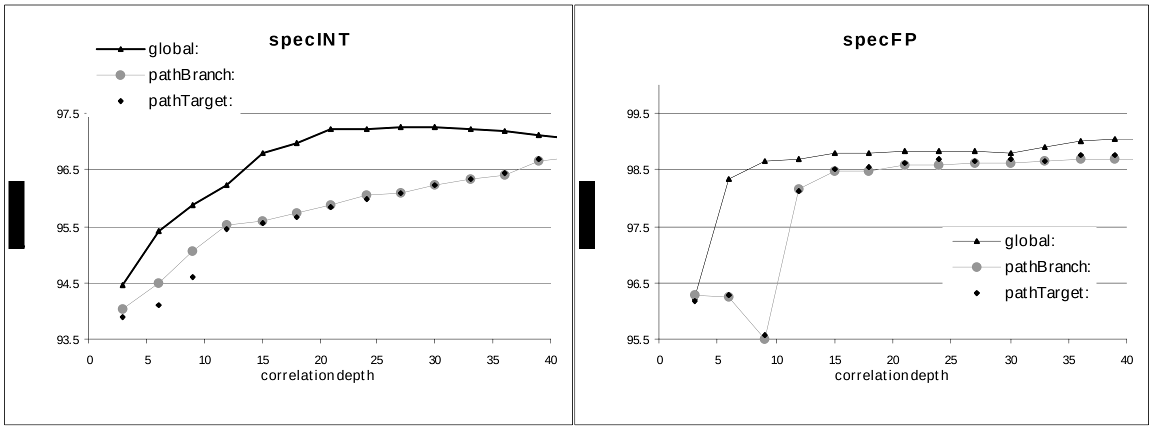


**Figure 4.3 – Limit study of global branch prediction size adjusted for the SPECFP95 and SPECINT95 averages. The study assumes that there is no lost of information when only the 3 LSB of the address are used**

address. From Figure 4.3, it is clear that the pathBranch and pathTarget schemes lose their edge when adjusted in size to the global scheme. In other words, it is more cost effective to increase the correlation depth than to retain the path information when no limit on resources is imposed.

This is not an indication that pathBranch and pathTarget under resource restrictions do not perform better than the global scheme as indicated by previous studies. What is indicated here is merely that the pathBranch and pathTarget, while performing a little better than the global scheme, when adjusted in size as scheme is outperformed by the global scheme. If under size restrictions the pathBranch and pathTarget outperform the global scheme for better indexing method or resource utilization, it makes it a better structure implementation, not a better scheme. Therefore, for the rest of the thesis we will not conduct studies using the pathBranch and pathTarget schemes as the global scheme has emerged as the more cost effective choice.

## 4.4 The Effect of Aliasing

Once the scheme is chosen, the reduction of aliasing is the only known method to improve prediction accuracy. It is therefore beneficial to know how close to fulfilling the full scheme's potential the structures implementing it are. If there is a gap between the scheme's potential and structures that implement the scheme prediction accuracy, it is because of aliasing.

Figure 4.4 shows the *gshare* and the *bi-mode* predictors compared to an aliasing-free version of the global scheme. As expected, as the size of the predictor increases, the adverse effects of aliasing diminish. The critical size where aliasing is no longer a

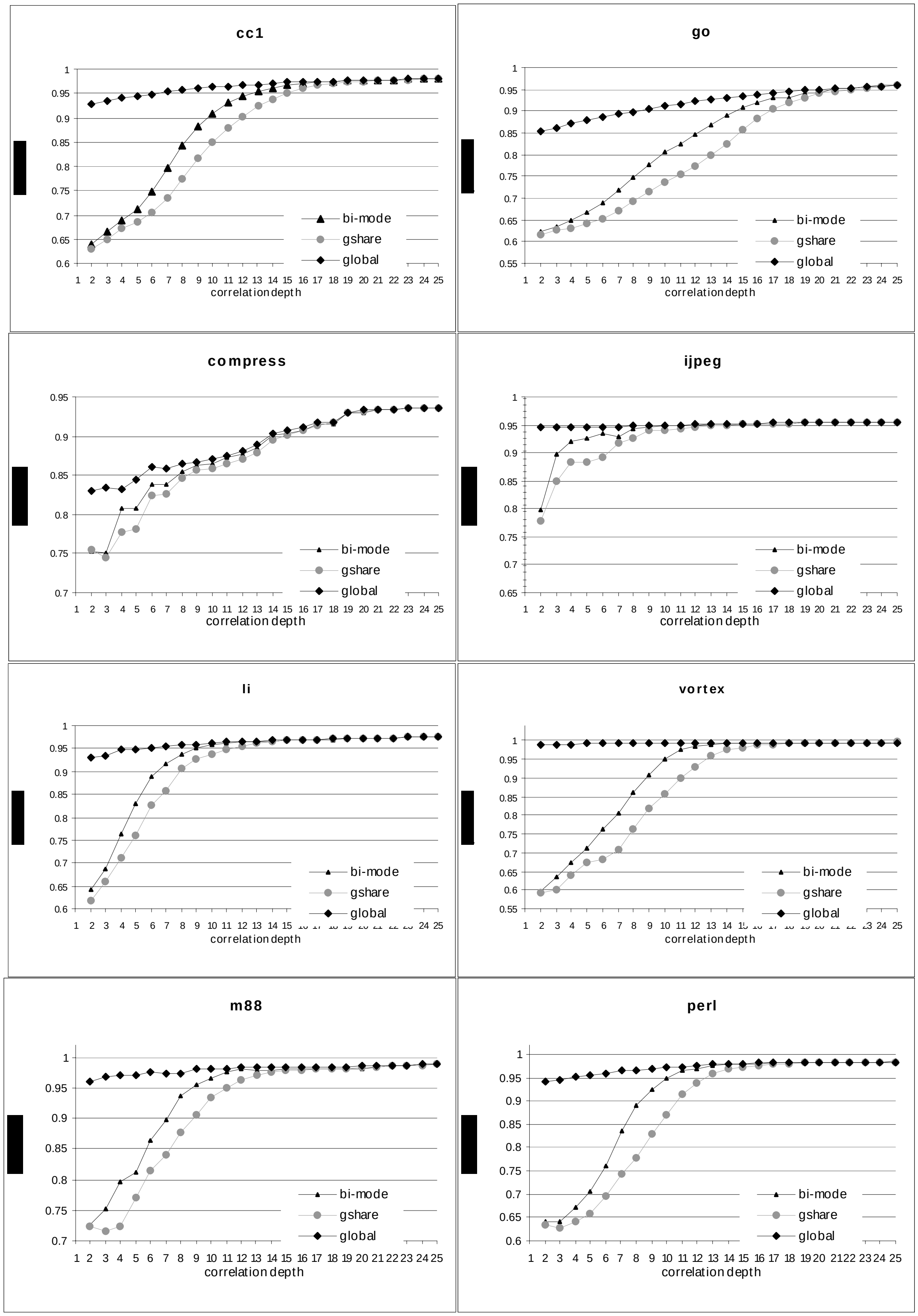


**Figure 4.4 - a) The effect of aliasing for the SPECINT95.**

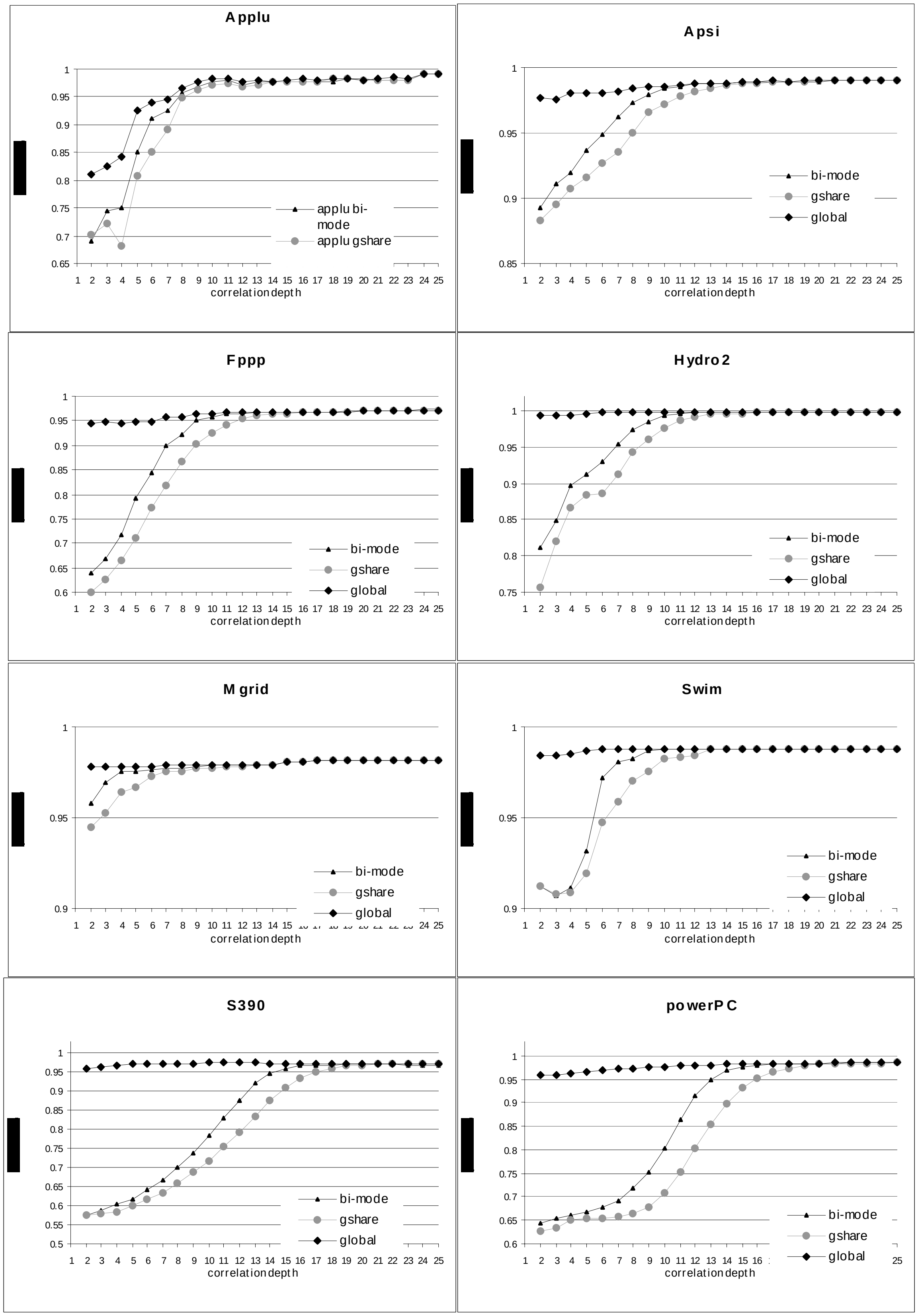


**Figure 4.4 - b) The effect of aliasing for the SPECFP95 and IBM traces.**

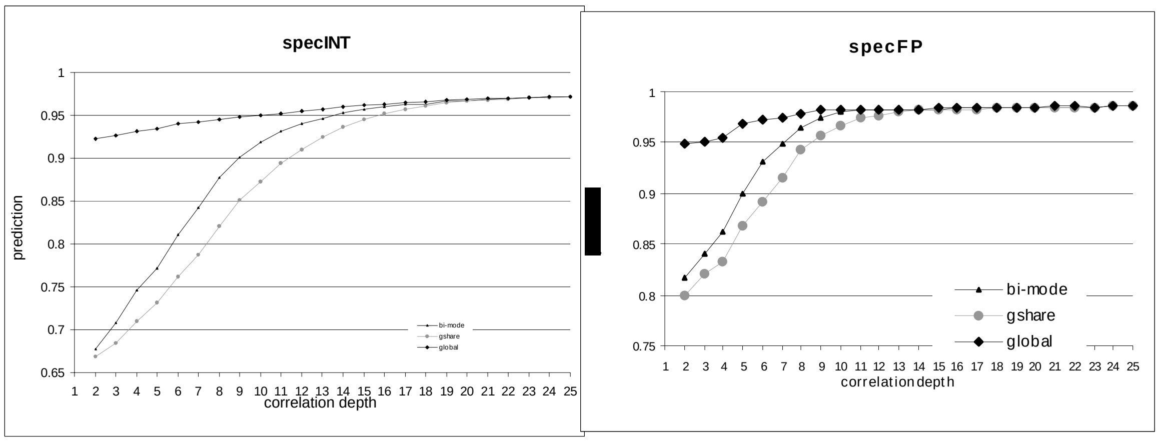


**Figure 4.4 - c) The effect of aliasing for the SPECINT95 and SPECFP95 averages.**

problem varies between different programs, and for some is not even reached for $2^{20}$ entries predictor.

The *gshare* predictor achieves 93.7% prediction accuracy for a realistic size of $2^{14}$ entries, and the *bi-mode* predictor achieves 95.3% prediction accuracy for the same size predictor. The potential of the global scheme for correlation depth of 14 is 96% prediction accuracy. Both implementations of the global scheme are short of achieving the global scheme potential. The *gshare* predictor is short by 2.3% and the *bi*-mode is short by 0.7% prediction accuracy. Traces with a large static branch signature suffer more from the degrading effect of aliasing. The *bi-mode* predictor is short by 4.1%

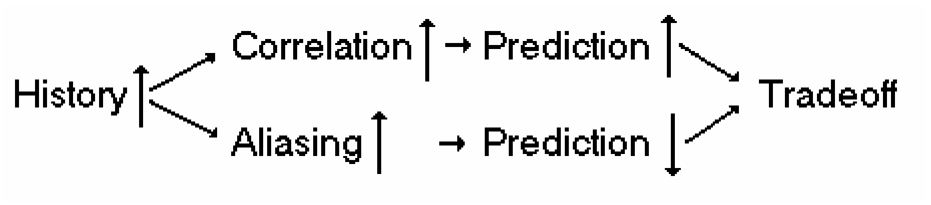


**Figure 4.5 – Correlation vs. Aliasing tradeoff.**

prediction accuracy from the global scheme potential for the go benchmark, and by 2.7% for the s390 benchmark.

Whether aliasing is a problem depends on the size of the predictor, the program that is running, and the branch prediction structure used. It is obvious that for a large enough predictor aliasing ceases to be a problem. However, as Section 7.1 will show, predictor sizes that were assumed to be realistic for future processors are not. Moreover, future processors will be forced to use smaller predictors than current microprocessors. This trend will aggravate the aliasing problem even further.

## 4.5 Correlation vs. Aliasing Tradeoff

The correlation depth in two level branch prediction schemes is determined by the size of the BHR. The BHR is usually combined with the program counter in some way to index the PHT. Deeper branch correlation is beneficial when resources are unlimited, as discussed in Section 4.2. When the PHT is limited in size, on the other hand, the best correlation depth is smaller than the equivalent limitless one. Moreover, the best correlation depth is program dependent as discussed in Section 4.4.

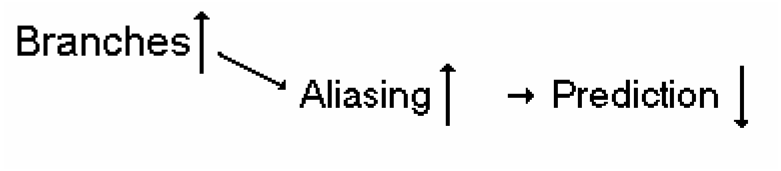


**Figure 4.6– Correlation vs. Aliasing tradeoff tradeoff.**

As the size of the BHR increases, the correlation depth increases. This correlates the branch in question to a greater number of previous branches, which further separates the current instance of the branch from other instances of the branch. This was shown to improve the prediction in Section 4.2. On the other hand, increasing the BHR size increases the density of information stored in the PHT. This increases the amount of aliasing, which impairs prediction. Increasing the size of the history register, therefore, has two competing effects on prediction as illustrated by Figure 4.5.

The greater the number of static branches in a program, the greater the amount of information the PHT has to store, and therefore the greater the aliasing effect (Figure

$$\text{PHT Capacity} = 2^{\text{history}} \times \text{\# of branches}$$

**Figure 4.7 PHT capacity equation**

4.6). This suggests that the more static branches that are in the program, the more destructive the aliasing effect is. Indeed, the benchmarks suffering from aliasing all have a high static branch signature. Such is the case with the s390 with 21,727 branches, the PowerPC with 16,710 branches, and the go benchmark with 7,401 branches. Compress, on the other hand, with its 495 branches, does not suffer considerably from aliasing.

Figure 4.7 depicts the equation dictating PHT capacity, where PHT capacity is the possible amount of information that could be stored in the PHT. From the equation, it seems like PHT capacity is dominated by the size of the history since the term is exponential. However, this term of the equation represents an upper bound, and we expect that as the history size increases, the percentage of actual patterns out of the possible $2^{\text{history}}$ will decrease. We therefore expect the term $2^{\text{history}}$ to much slower than the exponential maximum, and not to entirely dominate the number of branches in this equation. To verify this assertion the amount of unique vectors stored in the PHT was recorded for increasing correlation depth.  Figure 4.8 empirically shows that as the

depth

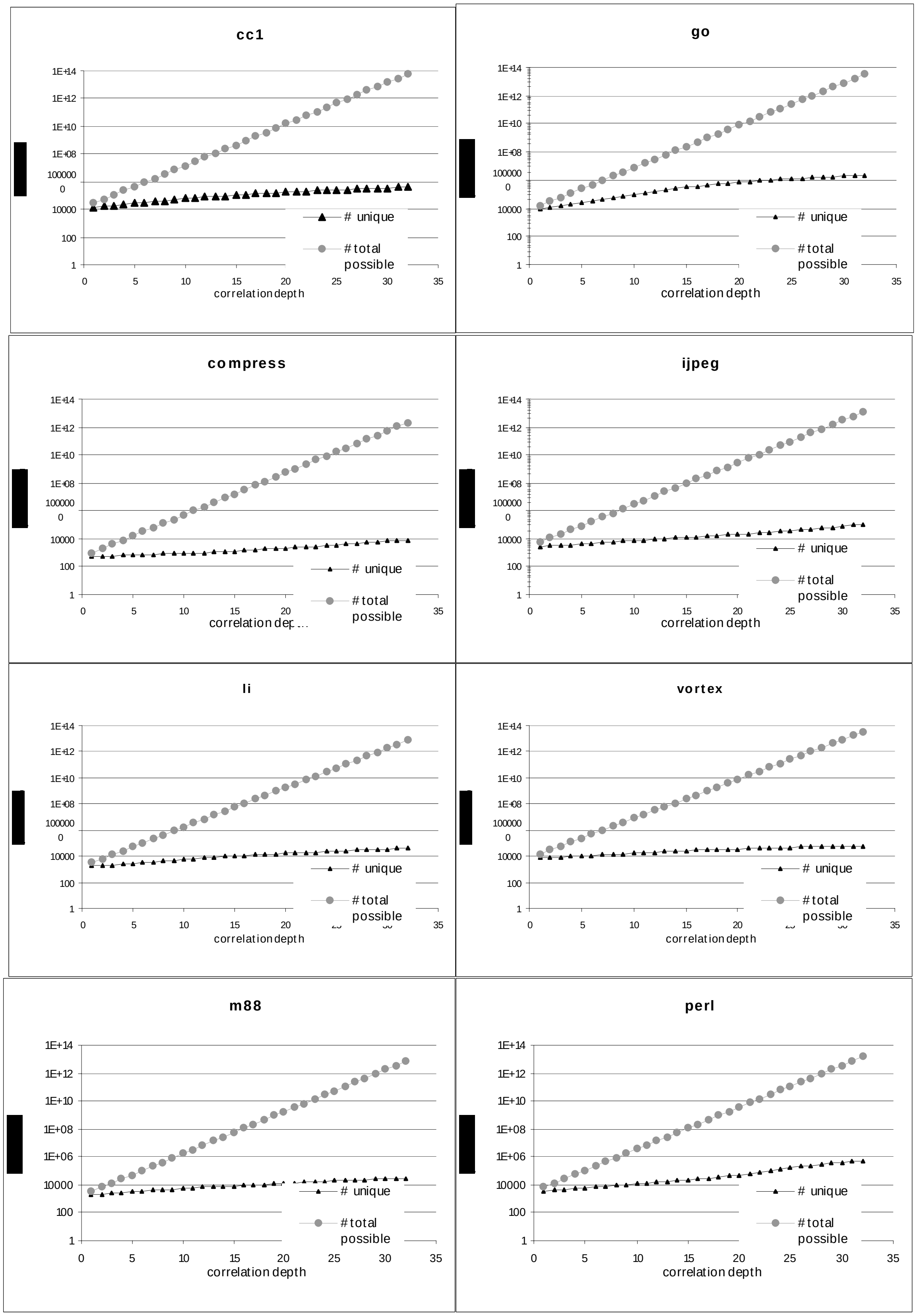


**Figure 4.8 - a) The amount of information to be stored in the PHT compared to the maximum possible information for the SPECINT95.**

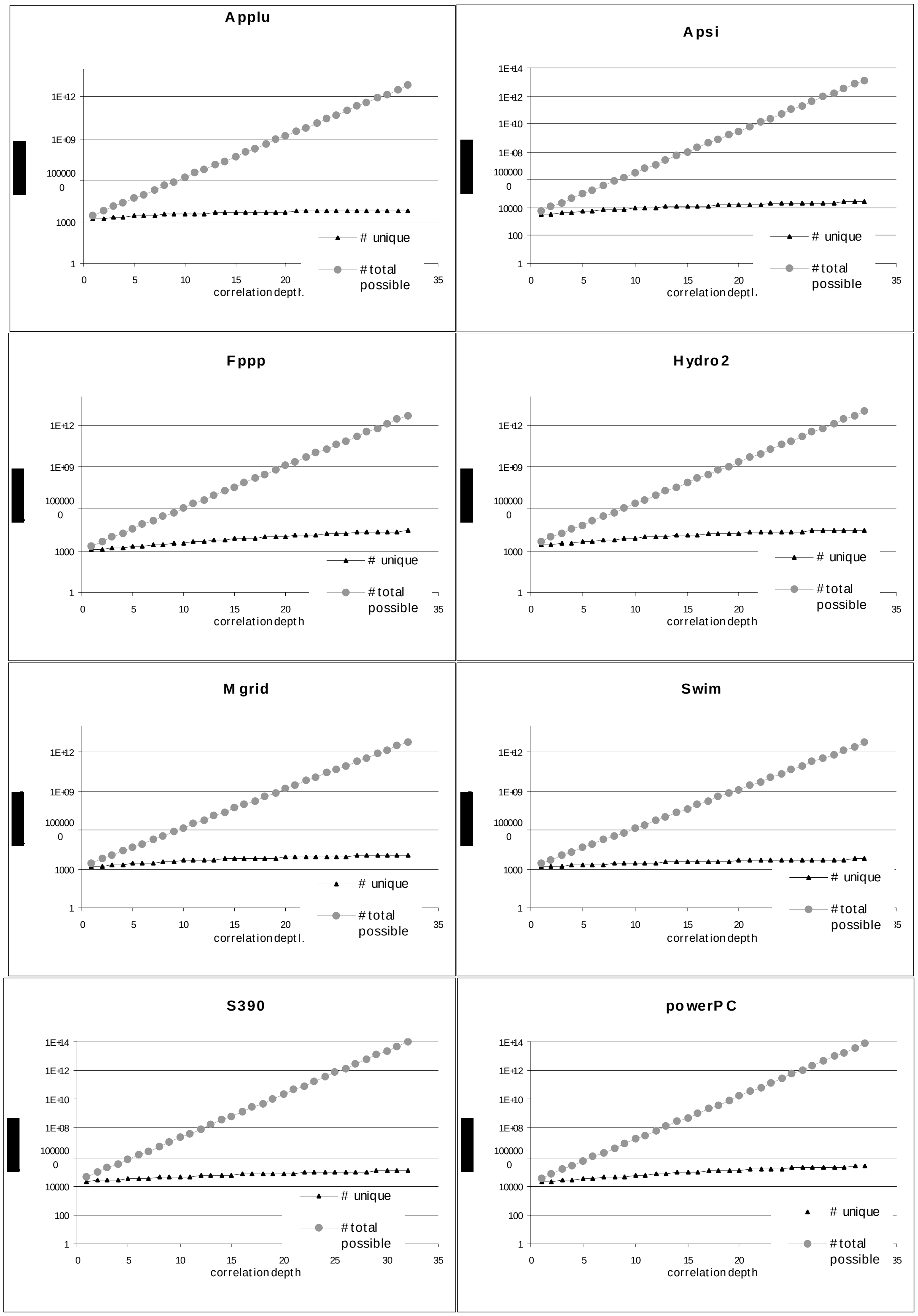


**Figure 4.8 - b) The amount of information to be stored in the PHT compared to the maximum possible information for the SPECFP95 and IBM traces .**

of correlation increases, the amount of information much slower than the exponential maximum possible, as suggested by the upper bound function of Figure 4.7. According to equation 4.7, if the number of static branches in a program is large, the size of the history register should be small, and vise versa. It is known that the number of static branches in a program varies and therefore, in accordance to the PHT capacity equation, the size of the history register should vary as well. The observation that different programs perform better for different history sizes was made before [19]. However, that work failed to give an explanation for this phenomenon. Following is an example, which illustrates how poor the understanding of the history size tradeoff has been thus far. The *bi-mode* predictor, a predictor conceived to reduce aliasing, was compared against *gshare* when it was introduced [12]. This comparison reveals the problematic fact that different prediction structures, due to their underlying structure, utilize different size history register. So, for example, a 1K entries *gshare* utilizing 10 bits BHR is compared to a 0.75K entries *bi-mode* predictor utilizing 8 bits BHR size. As a result, the *bi-mode* predictor, which is supposed to reduce aliasing, has less aliasing than the *gshare* predictor. A better comparison would be to utilize 8 bits BHR in the 1K entry *gshare*.

## 4.6 Decoupling Correlation from PHT size

Traditionally, the depth of the correlation is coupled with the size of the PHT. For example, a *gshare* predictor with a 1K entry PHT employs 10 bits of correlation. If a 2K entries size PHT is available, 11 bits BHR is used. The only attempt to decouple the

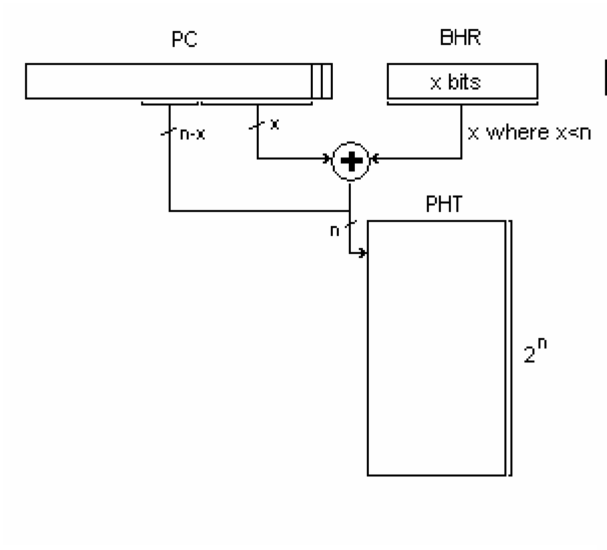


**Figure 4.9 - a) Decoupling correlation from PHT size for sizes BHR size less than log2 of PHT size**

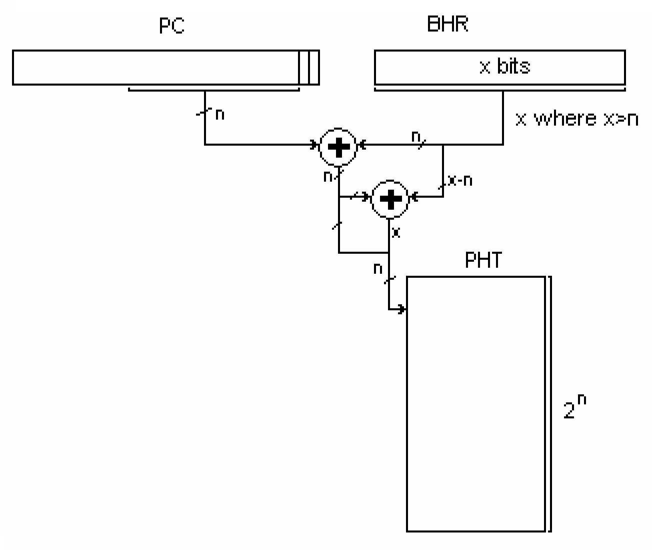


**Figure 4.9 - b) Decoupling correlation from PHT size for sizes BHR size greater than log2 of PHT size**

depth of correlation from the PHT size was done by reducing the number of BHR bits below the maximum depth of correlation that can be used for a given size PHT [19].

It has been shown before that prediction of programs' branches benefits from deeper correlation (Section 4.2). What might interfere with better prediction when using a deeper correlation is aliasing. Section 4.1 explains why this is the case and concludes that benchmarks with small number of static branches would benefit from deeper correlation without being as adversely effected by aliasing as might benchmarks with large numbers of static branches.

Because different programs can benefit from different size of correlation, It is beneficial to decouple the depth of correlation from the size of the PHT, and provide each program its optimal depth of correlation. Decoupling correlation from the size of the PHT entails a BHR of an optimal size regardless of whether it is smaller or larger than the log2 of the PHT size. If $2^{BHRsize}$ is smaller than the PHT, the rest of the bits indexing the PHT will come from the PC only as shown in Figure 4.9.a on the other hand, if $2^{\text{BHR size}}$ is greater than the PHT, the BHR will be folded as needed to form an index of size log 2 of the PHT, as seen in Figure 4.9.b.  This simple idea has never been considered

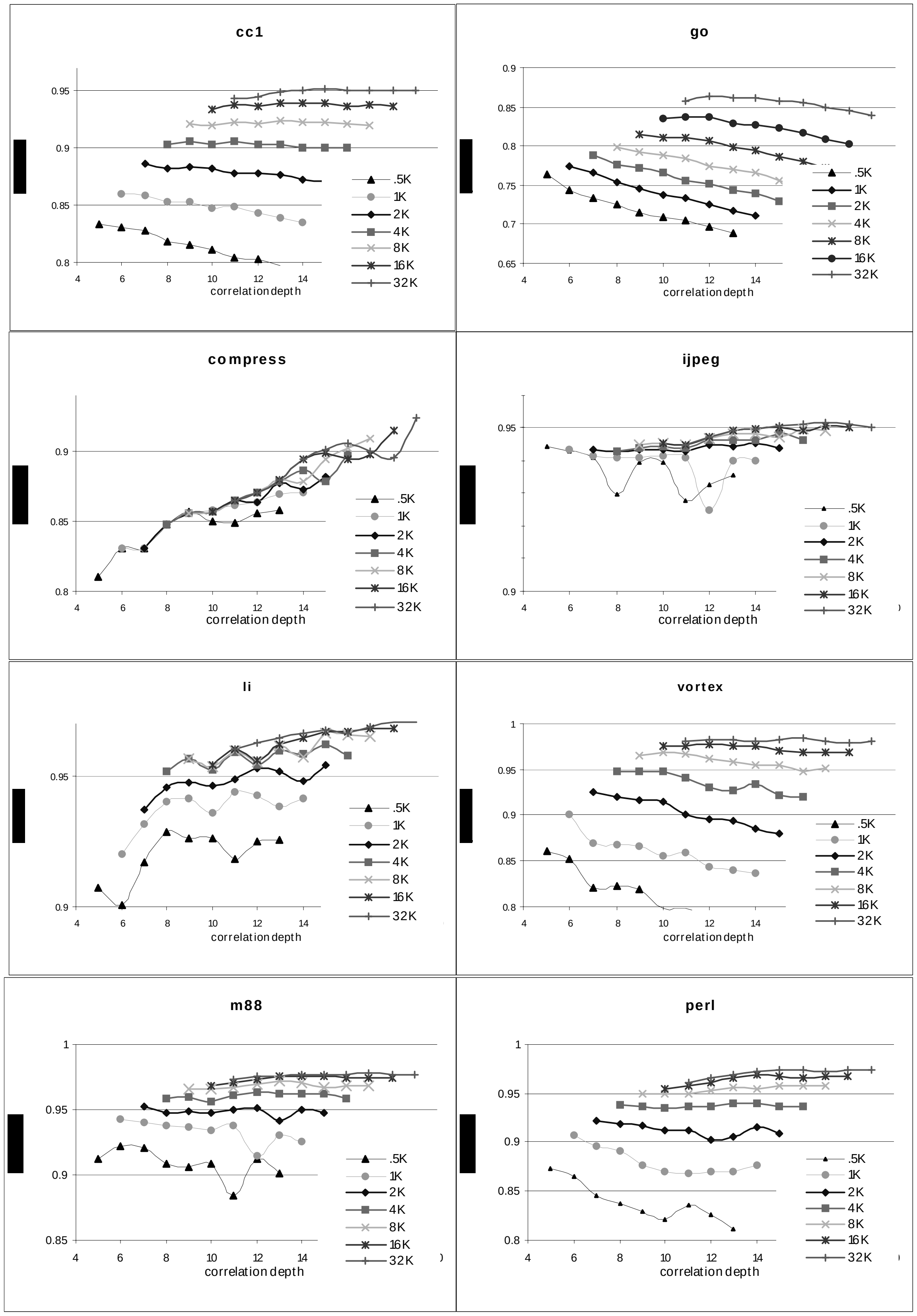


**Figure 4.10 a) Decoupling correlation from PHT size for the SPECINT95 benchmarks**

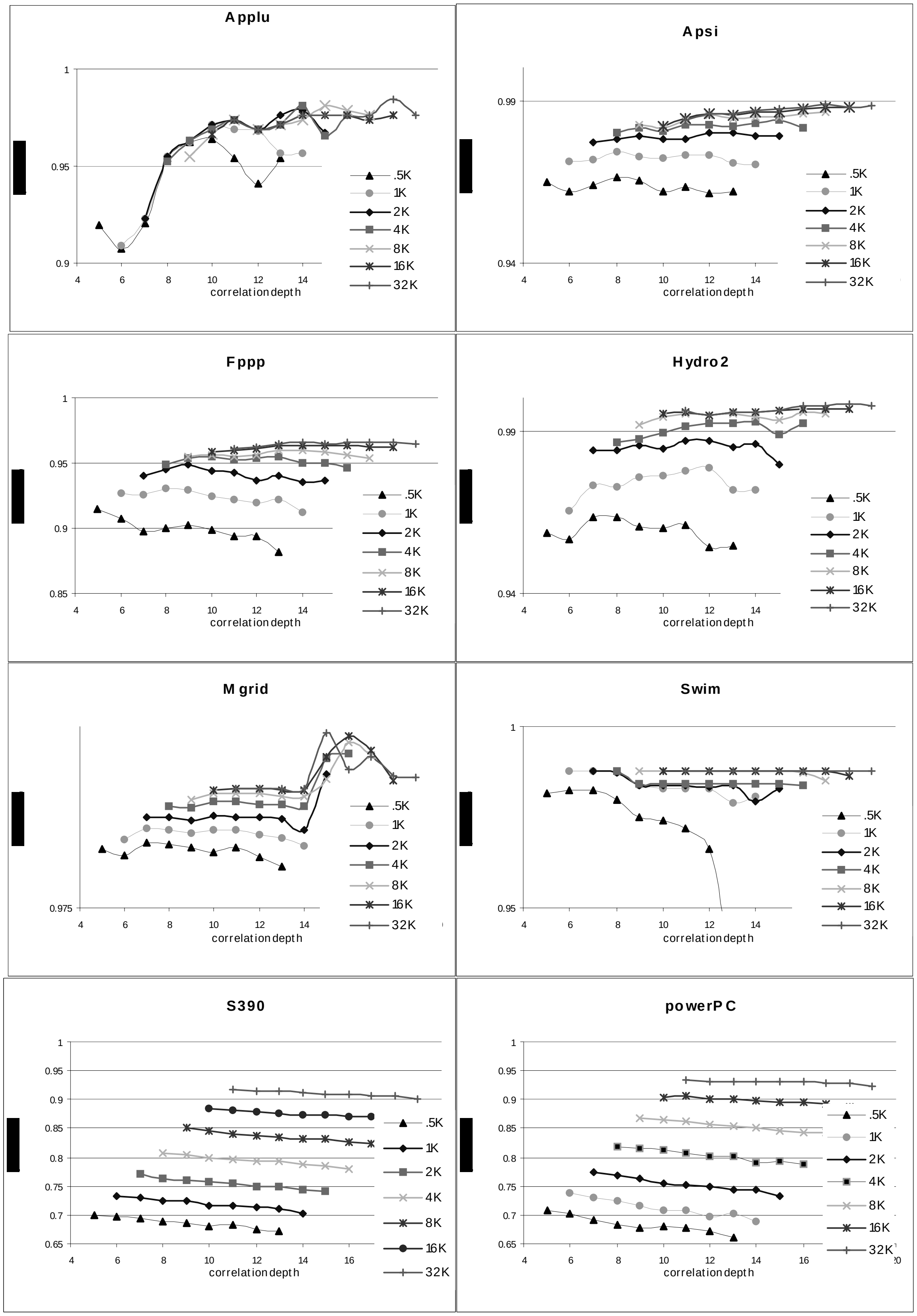


**Figure 4.10 b) Decoupling correlation from PHT size for the SPECFP95 ande IBM benchmarks**

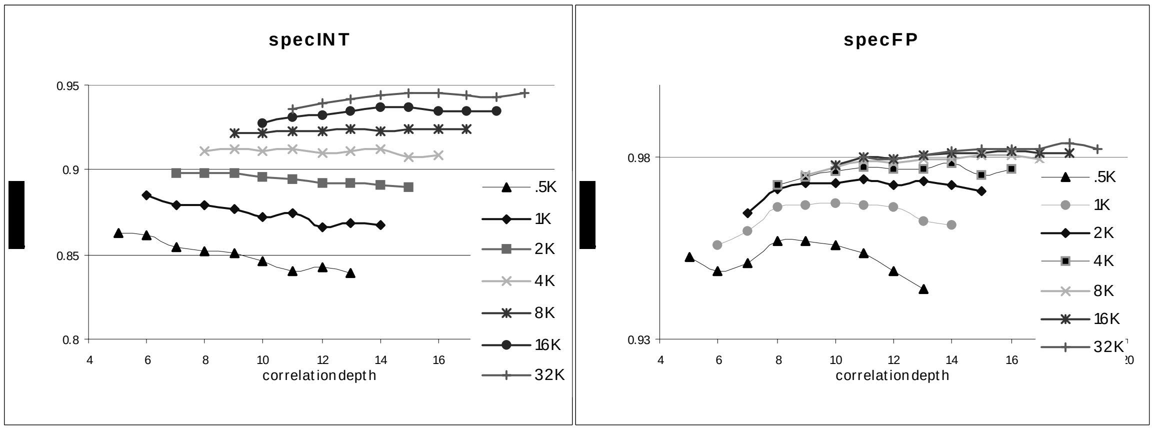


**Figure 4.10 c) Decoupling correlation from PHT size for the SPECINT95 and SPECFP95 benchmarks**

because in most studies the size of the PHT and the depth of correlation were considered the same.

Figure 4.10 shows prediction accuracy when varying the BHR depth for different PHT sizes. The different plots represent different sizes of PHT starting from 0.5K entries and going up to 32K entries by factors of 2. Each plot has 9 points where the middle point represents the “classical fit” for *gshare*. For example, the “classical fit” for *gshare* with PHT 1024 is 10 bits BHR. The other points on each plot represent BHRs of sizes –1,-2,-3 and –4 from the “classical fit” as shown in Figure 4.9.a, and +1, +2, +3,and +4 BHR size as depicted in Figure 4.9.b. The plots are ordered by size indicating that an easy way of achieving better prediction accuracy is dedicating more resources to the branch predictor. In benchmarks with a large number of static branches like go, s390, and PowerPC, it is not a good tradeoff to add more correlation than the “classical fit.” In fact, the smallest BHR always achieve the best prediction. It is likely that a smaller BHR than what is recorded in Figure 4.9 will achieve even better prediction accuracy for those benchmarks. While it is true that more correlation adversely effects prediction in benchmarks with a high number of static branches, notice that as the size of the PHT increases, the negative effects seem to fade. It is possible that even for those benchmarks, if the PHT of sizes greater than 32K entries are considered, more correlation will become a good choice.

The perl benchmark is a good example of a benchmark that for small PHT, a large BHR is not a good choice, while for larger PHT sizes it is a good choice. It is expected that for most benchmarks a large enough PHT will have the same effect.

If a certain BHR size needs to be chosen, as traditional implementation of branch predictor are required, the size of the BHR needs to be chosen according to the size of the PHT. The SPECINT95 average and SPECFP95 average follow the same path described above. For a 0.5K entries less correlation prevails, while at a PHT of size 32K, more correlation turns out to be a good choice.

# Chapter 5 - Myths of Hybrid Predictors

## 5.1 Issues in Hybrid Predictors

4Visual comparison between the *bi-mode* [12] structure and the McFarling hybrid predictor [14] reveals a striking similarity, even though the two branch predictors were conceived for different purposes. The *bi-mode* was created to reduce aliasing, and the McFarling predictor to combine the advantages of the local and global branch prediction schemes.

Within the hybrid path, some studies promote a static selection mechanism while others studies prefer to use a dynamic selection mechanism. The advantage of using a static selection mechanism over a dynamic selection mechanism is reduction of information stored in the predictor in two ways. First, the selection mechanism does not take hardware resources, but conveyed to the branch predictor via the ISA. Second, since each branch only uses one of the hybrid components, it is unnecessary to update both components for each branch. Using a dynamic selection mechanism, on the other hand, is useful because some branches might change their best predictor throughout the execution of the program.

This study was initiated in order to better understand how to best combine branch predictors that reduce aliasing and hybrid branch predictors. However, during the investigation it was revealed that those two paths are one and the same, as this chapter will demonstrate. First, this chapter will discuss the simulation methodologies used specifically for this chapter.

## 5.2 Simulation Methodology

In all simulations performed for this chapter, the depth of correlation, or the size of the history register/s, follows directly from the size of the PHT. For example, if the global component in the hybrid predictor had 1K entries in its PHT, the history register size would be 10 bits. Throughout this chapter, the McFarling local-*gshare* hybrid predictor is used because in preliminary simulations it was found to be the best true hybrid behavior.

In the limited size simulations, a two-way set associative BTB with 4K entries was used. This is large enough to prevent it from being a performance bottleneck and enabled concentration on the tradeoffs in the PHTs.

The McFarling hybrid predictor simulated had two components— a *gshare* structure implementing the global branch prediction scheme, and a PAs structure implementing the local branch prediction scheme. In cases where a dynamic selection mechanism was employed, the bimodal structure was used.

Unless stated otherwise, profiling was done on the same data sets that were used for running the simulations. This enabled us to obtain an upper limit on the prediction accuracy. As is shown in one of this chapter's studies, it is anticipated that using a different data set (the more realistic situation) for profiling would degrade the performance of the hybrid predictor with a static selection mechanism.

## 5.3 Selection Mechanism

First, the relative merits of using static versus dynamic selection mechanism to choose between the different components of a hybrid predictor are examined. As noted earlier, a static selection mechanism requires less information to be stored in the predictor structure because each branch utilizes only one component. This reduces contention, which reduces aliasing and helps prediction accuracy. Moreover, hardware resources that would have been used for the selection mechanism are now available for increasing the size of the predictor's components. The main problem with static selection is the additional bits needed in the ISA. Although some ISAs have this bit in place, others will require that the ISA be altered. Dynamic selection mechanisms are thought to have an edge over static ones because it has been suggested that the best component for predicting a branch can change during the execution of a program.

### 5.3.1 The Merit of Dynamic Selection Mechanism

It is unclear whether there is an inherent benefit in choosing the component used by a specific branch dynamically. If the best component to predict a branch dynamically changes during the program run, of course it would be better to dynamically select the component used by a branch. However, if there is no inherent benefit in choosing the component used by a branch dynamically, it is better to choose it statically and avoid the extra cost of using both components for each branch, and the cost of the selection mechanism.

Figure 5.1 shows the prediction accuracy for a global-local hybrid predictor with unlimited resources. The three plots represent three types of selection mechanisms: per-branch oracle, per-instance oracle, and an implementation of a real selection mechanism - the *bimodal*. The per-branch oracle records prediction accuracy for both components. When the program terminates, it chooses the best component as the predictor for each branch. The per-instance oracle gets a prediction from both components, and if either of them is correct, it records a correct prediction. It is interesting to note that the per-instance oracle is an overestimation and that even for a randomly generated prediction, it probability dictates a 75% correct prediction.

Determining whether the best component to predict a branch changes during program execution is difficult. One approach to assist in this determination is to slice the dynamic stream of a specific branch into *n* subsets of branch instances, and then to choose the best component for each set [52]. The problem with doing this is that a small *n* leads to an optimistic outcome, while a large *n* might erase the benefit of having a dynamic selection mechanism. Using either a large *n* or a small *n* can lead to erroneous conclusions. Clearly, it does not matter whether the best component for each branch changes throughout the program run if a known selection mechanism cannot identify the best component dynamically.

In our experiments, we used an unbounded hybrid predictor with an unbounded *bimodal* selection mechanism. This eliminated the adverse effects of aliasing and allowed a check on whether the *bimodal* selection mechanism could capture the changing best predictor throughout the program execution. Figure 5.1 shows that there is no inherent gain in using a dynamic selection mechanism. In other words, if there is a gain to be made in changing the component used for each branch during the program execution, the *bimodal* selection mechanism does not capture it. This is clearly demonstrated in the graphs of Figure 5.1 where it can be seen that the *bimodal* selection mechanism always under-performs the per-branch oracle. Moreover, it appears that the bimodal selection mechanism makes mistakes in selecting the proper components, which degrades the overall performance.  This phenomenon is accentuated in programs with a large number of branches like the S390 and PowerPC.  They display a significant gap

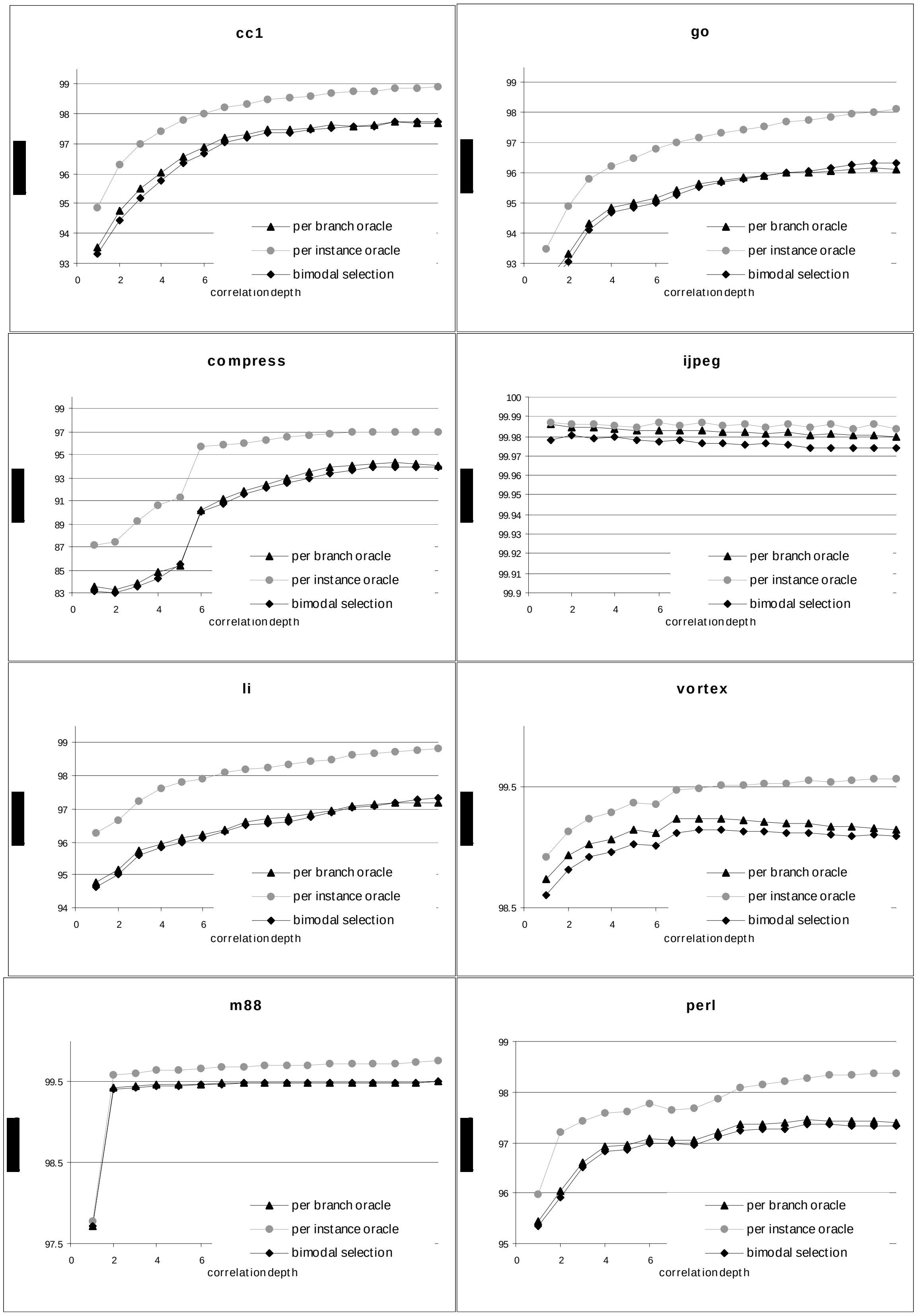


**Figure 5.1 – a) Testing the potential of static vs. dynamic selection mechanisms in a unlimited resource environment for the SPECINT95**

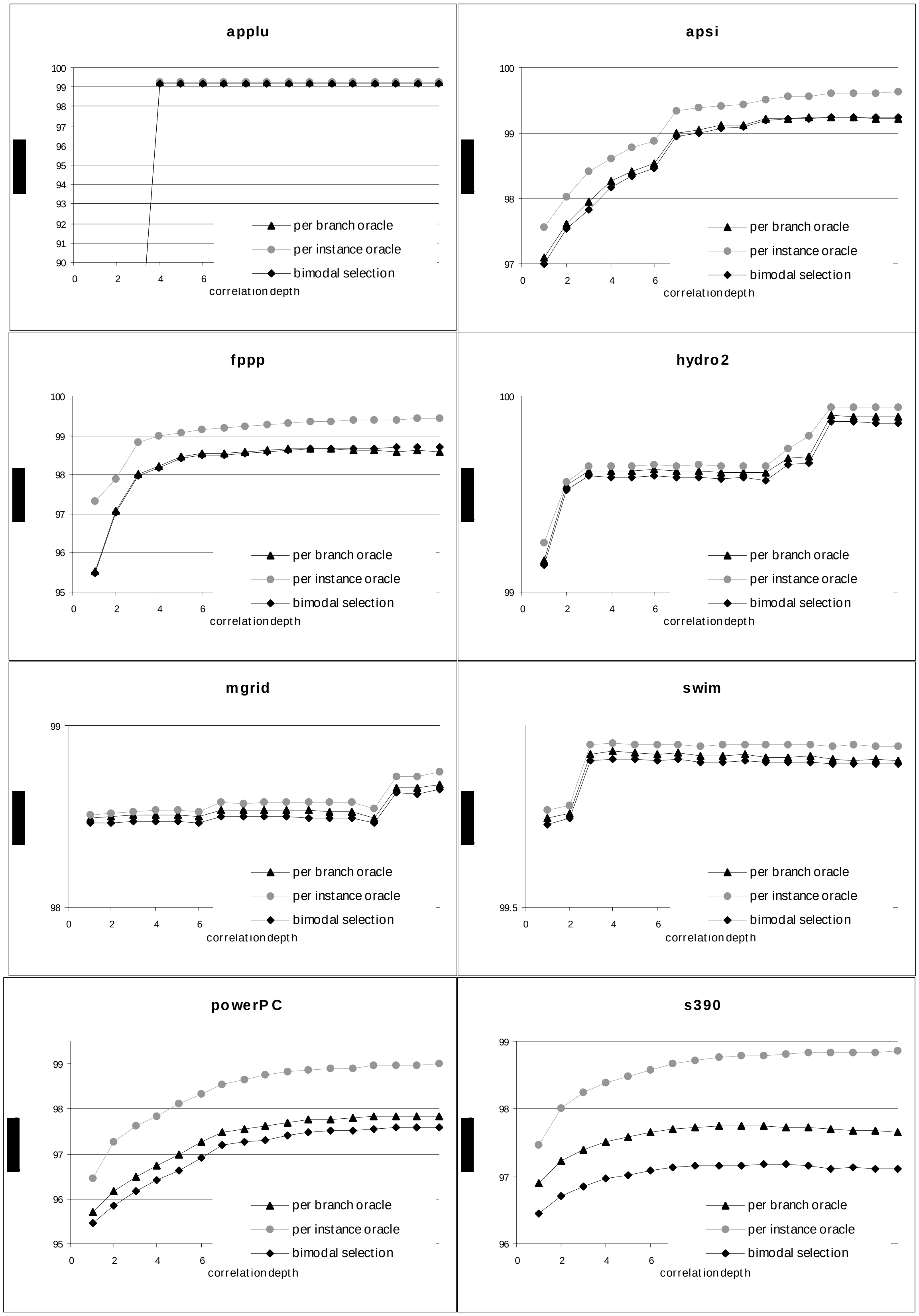


**Figure 5.1 – b) Testing the potential of static vs. dynamic selection mechanisms in a unlimited resource environment for the SPECFP95 and IBM benchmarks**

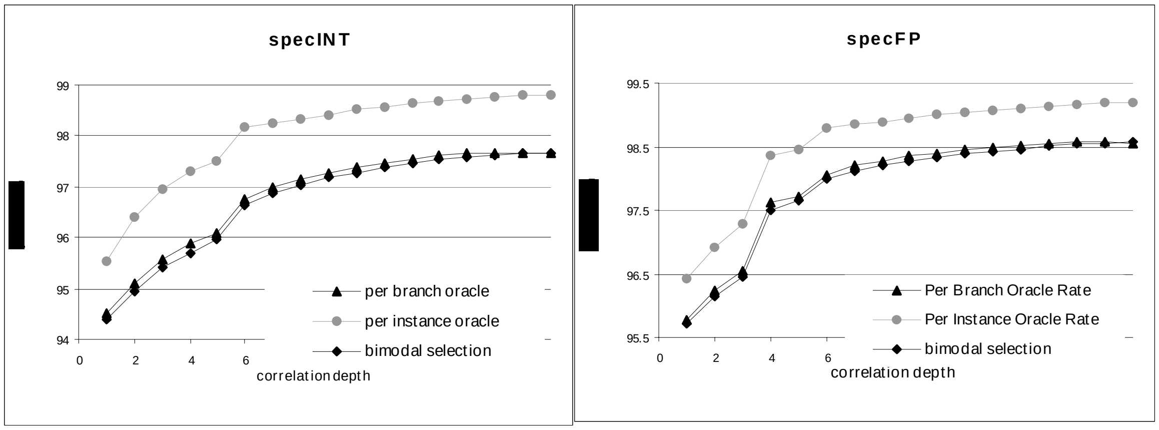


**Figure 5.1 – c) Testing the potential of static vs. dynamic selection mechanisms in a unlimited resource environment for the SPECINT95 and SPECFP95 averages**

between the prediction of the oracle static selection mechanism and the prediction when using the bimodal selection mechanism.

### 5.3.2 Dynamic vs. Static Selection Mechanism

Figure 5.1 depicts the inability of the dynamic selection mechanism to dynamically adapt to the changing behavior of branches, even if such transient behavior exists. There does not appear to be any advantage to employing dynamic selection mechanisms instead of static ones. It is thus expected that in a limited resource setting, a static selection mechanism would outperform a dynamic selection mechanism for the reasons mentioned above. Figure 5.2, however, shows the exact opposite. In a limited resources setting, the hybrid predictor with a dynamic selection mechanism outperforms a hybrid with a perfect static selection mechanism.

Holding the heel of this observation a question is born: What is it about the dynamic selection mechanism that boosts the performance of a hybrid predictor with a dynamic selection mechanism when working in a size-restricted structure? Alternatively,

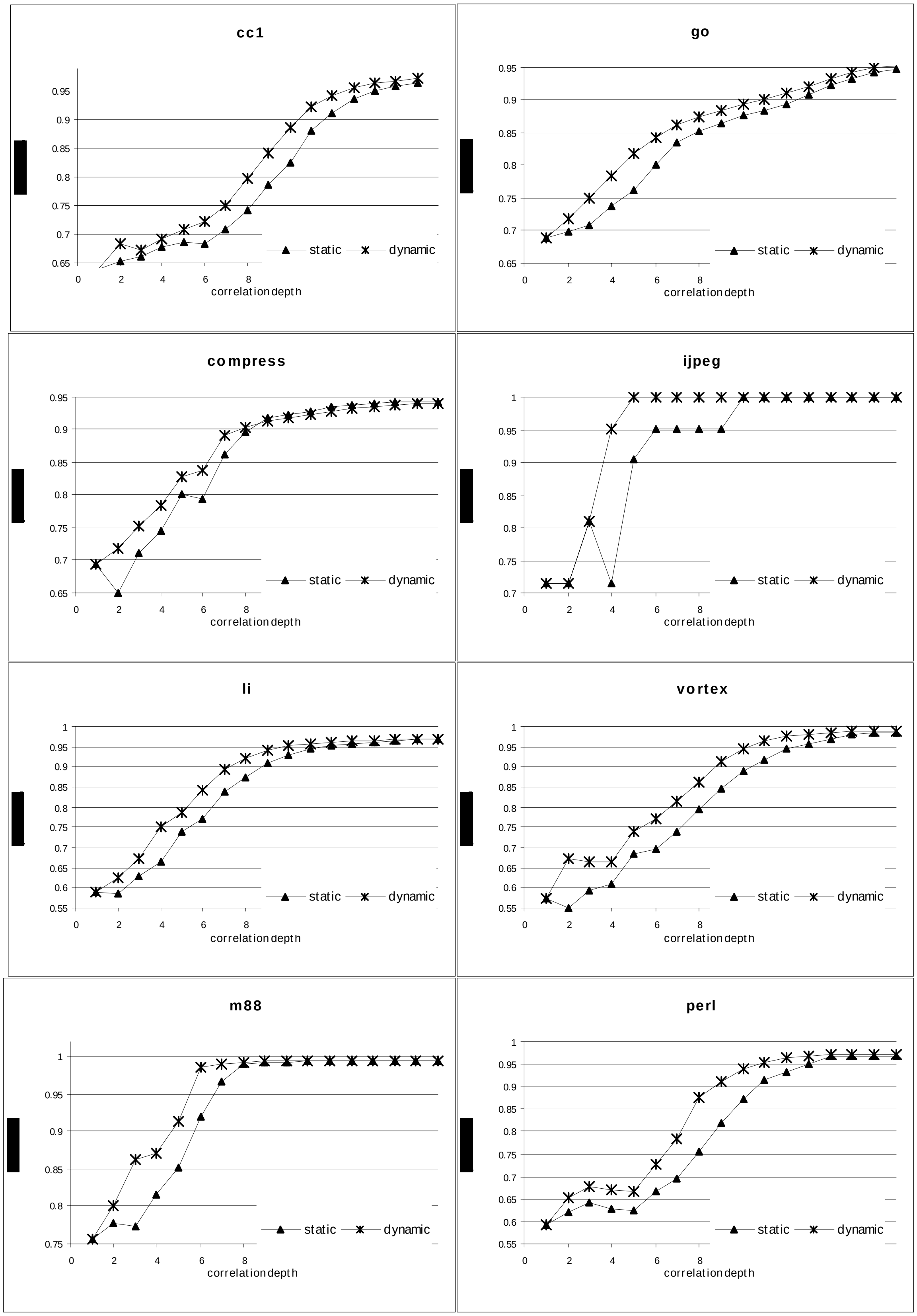


**Figure 5.2 – a) Dynamic vs. perfect static selection mechanism in hybrid predictors for the SPECINT95 benchmarks**

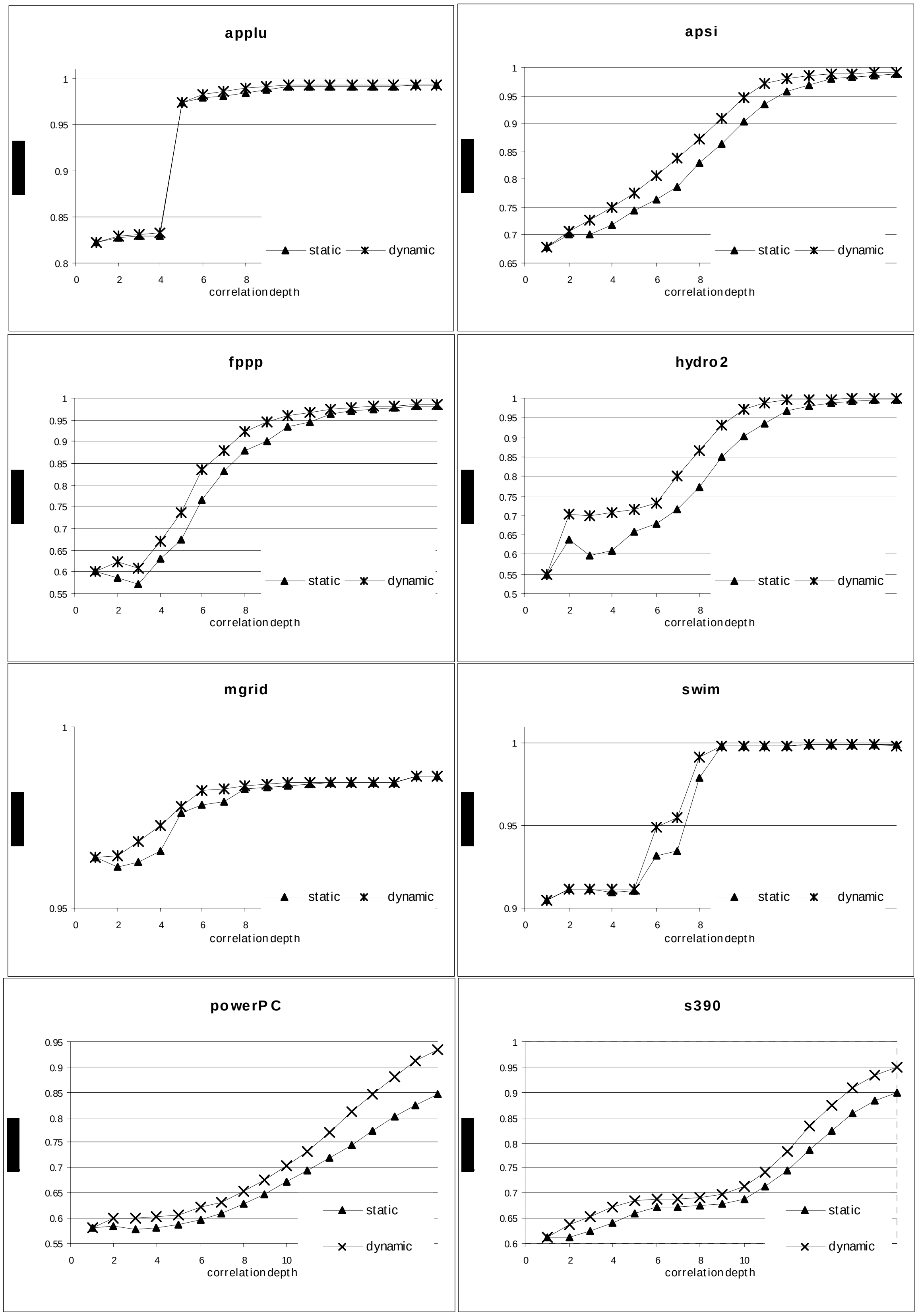


**Figure 5.2 – b) Dynamic vs. perfect static selection mechanism in hybrid predictors for the SPECFP95 and IBM benchmarks**

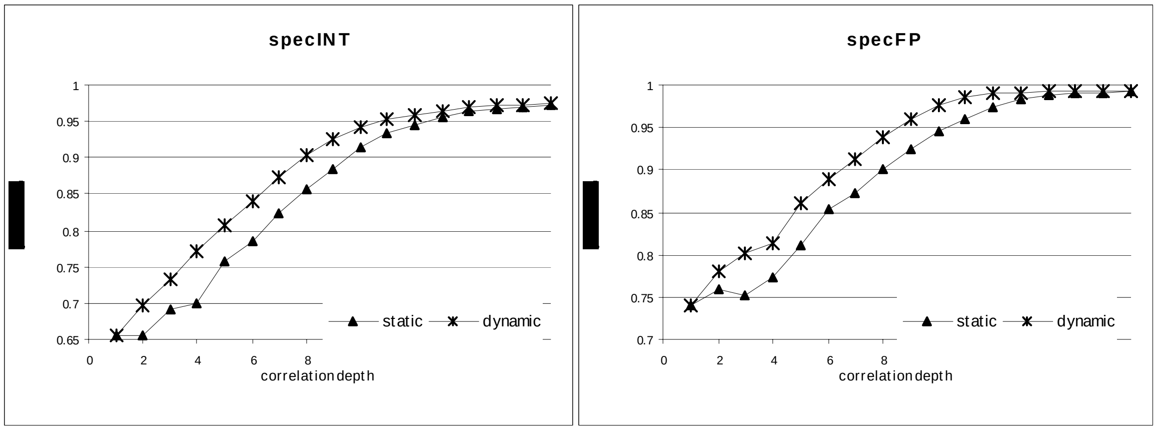


**Figure 5.2 – c) Dynamic vs. perfect static selection mechanism in hybrid predictors for the SPECINT95 and SPECFP95 averages**

what is it about the static selection mechanism that in a limited-resource setting degrades the performance of a hybrid predictor?

### 5.3.3 The Omniscient Dynamic Selection Mechanism

One possible hypothesis to explain this question is that a dynamic selection mechanism reduces aliasing. For example, consider the case where two branches *A* and *B* are both better predicted by the global component of the hybrid predictor. In an unlimited resource setting, a dynamic selection mechanism will choose the global component to predict them. In a resource limited setting, branch *A* will suffer from aliasing, which considerably degrades the prediction of its global component. As a result, the dynamic selection mechanism chooses the local component to predict branch *A*'s outcomes. Although both branches *A* and *B* are inherently predicted more accurately by a global component, branch *A* will be better predicted by the local component in a limited resources environment. We next examine how much aliasing reduction helps a hybrid predictor.

Figure 5.3 shows the extent to which reducing aliasing helps boost the performance of hybrid prediction. It compares a resource bound local-global hybrid predictor (hybrid), with a resource bound local-global hybrid (aliasing hybrid), whose selection mechanism does not take into consideration the effects of aliasing. To simulate this effect, a run of the local-global hybrid predictor was made with no limits on resources. The selection pattern for the entire run was logged and later served as the selection mechanism in the limited hybrid version. The selection mechanism in this case is that for the true hybrid behavior with no regards to aliasing, since it was recorded in an aliasing-free setting. The conclusion from Figure 5.3 is that a large portion of the benefits brought by hybrid predictors with dynamic selection mechanisms comes from reducing aliasing. Moreover, comparing the hybrid predictor to an unlimited version of the global scheme (UL global), shows that the local-global hybrid predictor never fulfils its promise of improving prediction beyond that of a single scheme, even for generous resource allocation. Notice that the difference between UL Hybrid and UL global is the potential difference between the hybrid predictor (global-local) and the global scheme. This difference pales in comparison to the difference between UL global and hybrid that represents the remaining aliasing after the *bimodal* selection mechanism was able to reduce some of them (the difference between hybrid and aliasing-hybrid).

In summary, a large portion of the benefits brought by hybrid predictors with dynamic selection mechanism comes from reducing aliasing. Moreover, the benefit of combining predictors to increase the potential prediction is questioned in this section, even though the studies were conducted in this section ignoring the size overhead of implementing the PAs structure (i.e. the table of history registers).

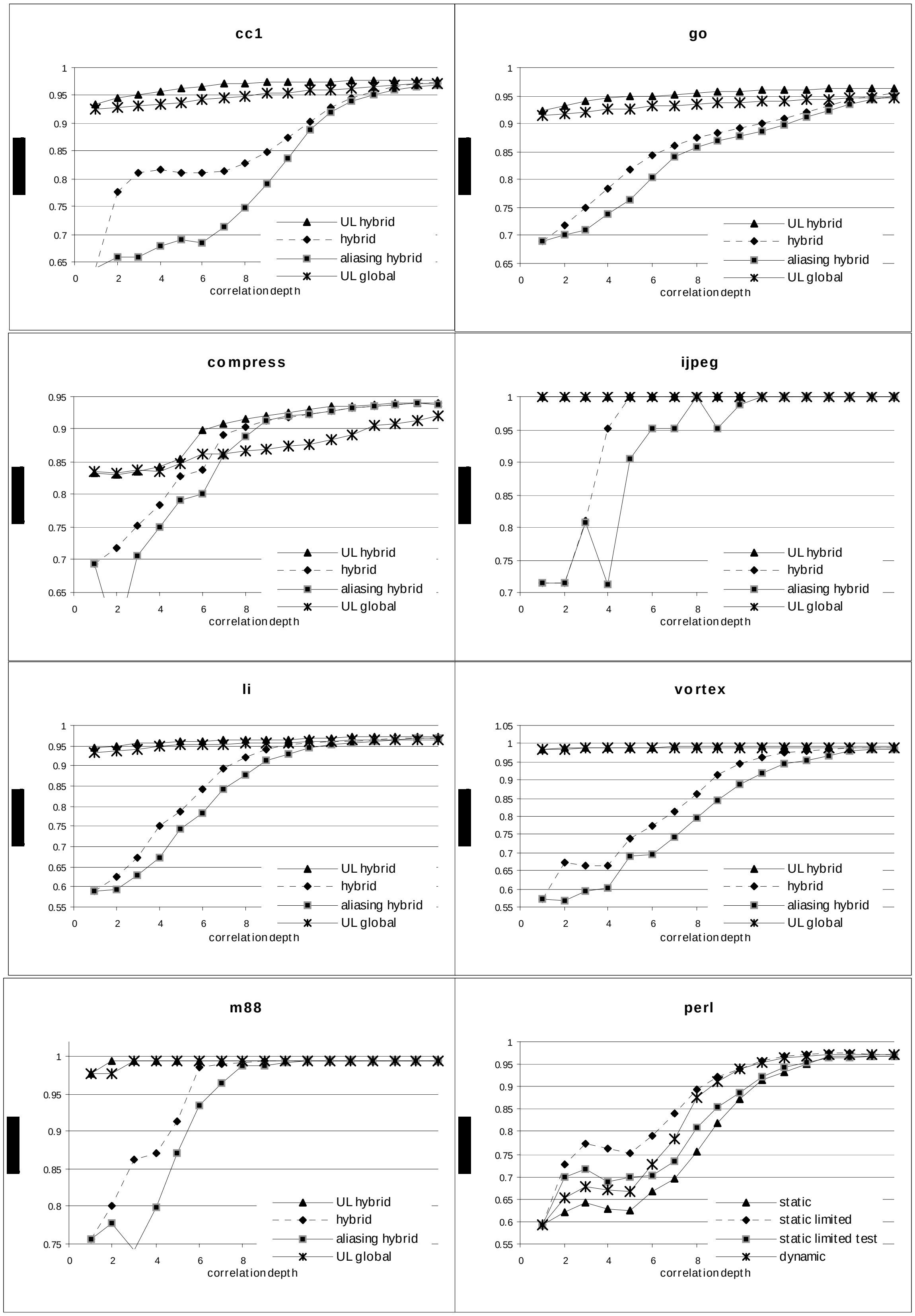


**Figure 5.3 – a) The role of Hybrid predictors in reducing aliasing for the SPECINT95 benchmarks**

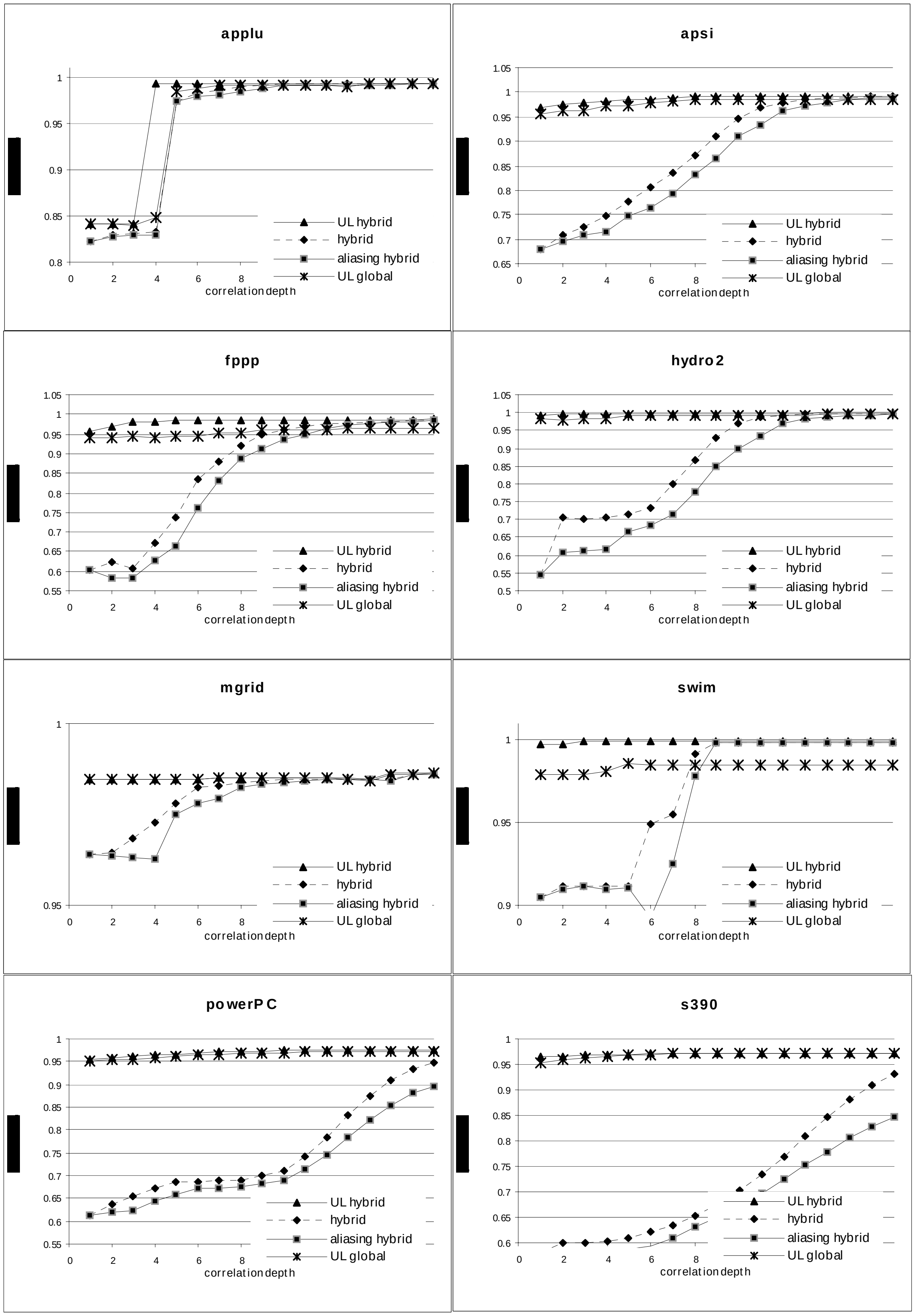


**Figure 5.3 – b) The role of Hybrid predictors in reducing aliasing for the SPECFP95 and IBM benchmarks**

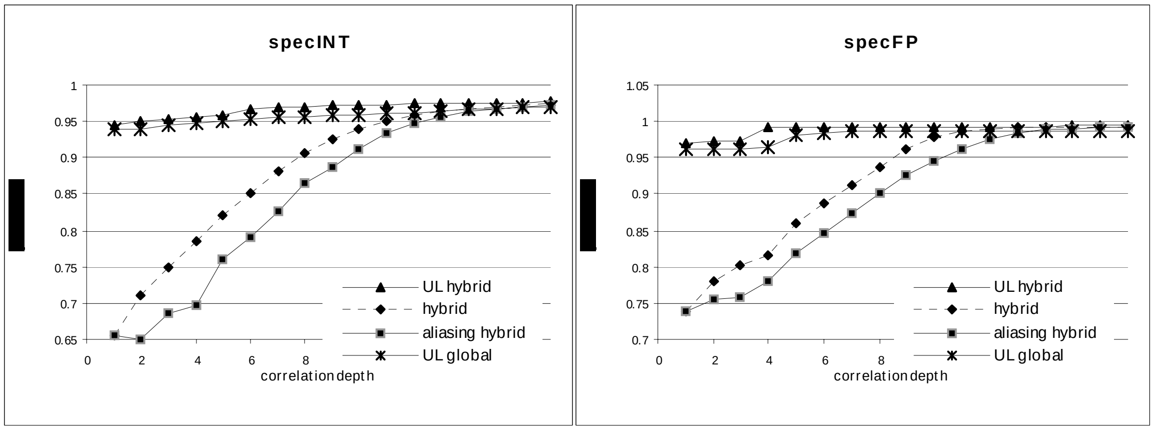


**Figure 5.3 – c) The role of Hybrid predictors in reducing aliasing for the SPECINT95 and SPECFP95 averages**

### 5.3.4 Static Aliasing Aware vs. Dynamic Selection Mechanism

At this point we have shown that both static and dynamic selection mechanisms reduce aliasing in hybrid branch predictors. The former does so by reducing contention in the structure and by eliminating the hardware cost in the selection mechanism. The later does so by dynamically distributing the branch stream across the two components, which alleviates contention in the PHT. The dynamic selection mechanism performs much better than an ideal static selection mechanism. In the ideal static selection mechanism, profiling was done with no limitation on resources. This led to branches that are better predicted by the global scheme to be mapped to the *gshare* component, and branches that are better predicted by the local scheme to be mapped to the PAs component. Notice that the ideal static selection mechanism does not take aliasing into consideration.

One way of considering aliasing is to use the actual table size when profiling. Figure 5.4 shows the importance of taking into consideration the size of the predictor structure when profiling. When taking size into consideration during profiling, the branches get distributed not just by their true hybrid behavior, but also by taking aliasing into consideration. Figure 5.4 shows that while a dynamic selection mechanism is better than a static selection mechanism with perfect profiling, employing profiling that takes the size of the structure into consideration (static limited) results in even better performance than dynamic selection. The fact that the difference between the prediction percentages diminishes with size indicates that the difference is due mostly to better aliasing reduction. Using this profiling method combines the advantage of static and dynamic selection mechanisms as explained previously.

The advantages of using a static selection mechanism with aliasing-bound profiling are as follows: the branches are distributed among the components according to contention in the structure; the selection hardware is eliminated; and only one component is used per branch, which further reduces contention.

### 5.3.5 Shortcomings of Static Selection

The question arises whether such good prediction can be achieved when profiling from a test data set. As Figure 5.4 shows, when using a different data set to profile the program, the static selection mechanism (static limited test) suffers degradation in performance. For small predictors, the static selection mechanism still performs better than the dynamic selection mechanism, but the dynamic selection mechanism eventually surpasses it.

This problem can be accentuated when code that is compiled and profiled for a certain size of predictor is used to run on a different implementation of the same ISA. Since profiling is done on a different size of predictor than the one, which the code is run

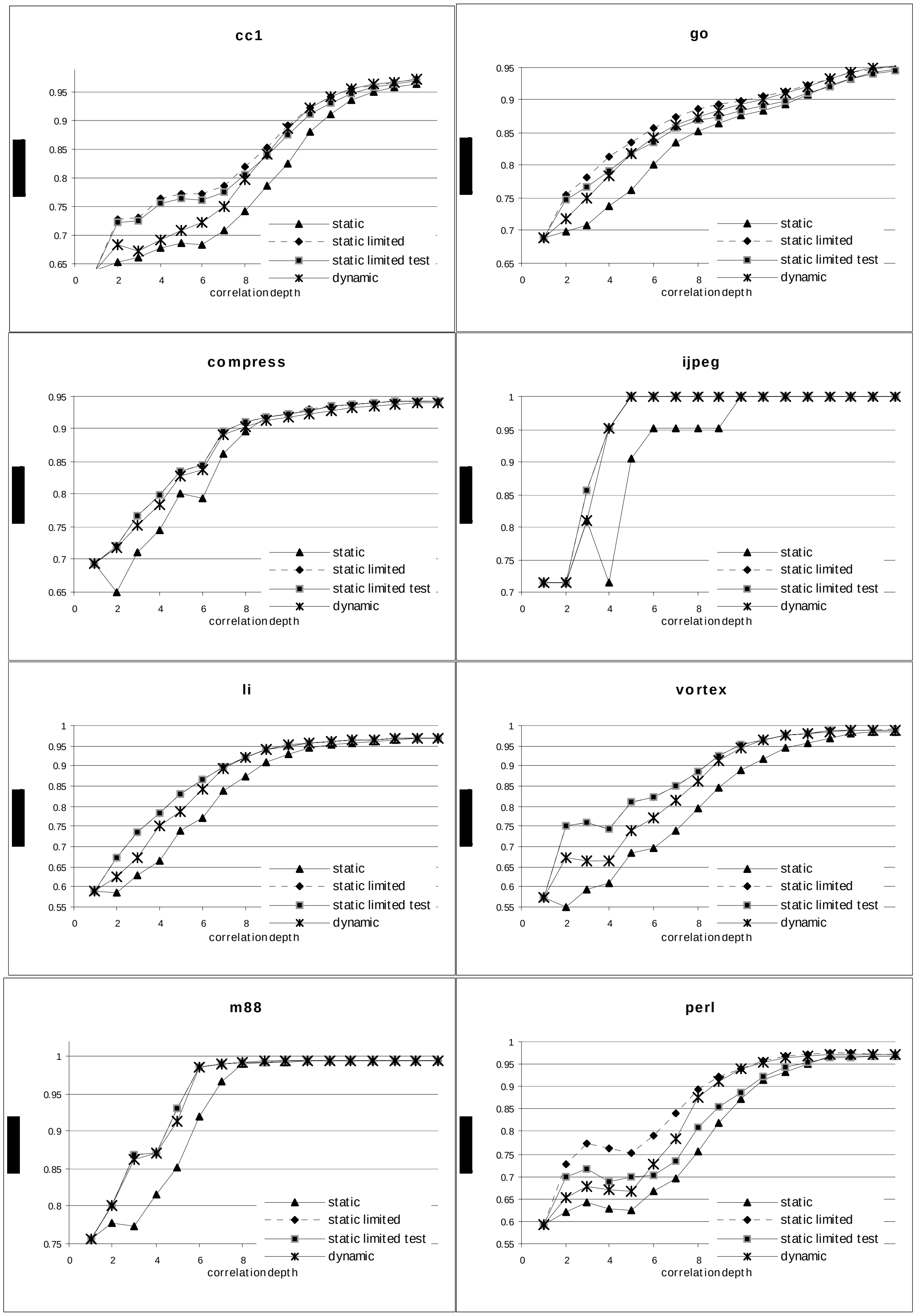


**Figure 5.4 – a) Dynamic vs. aliasing aware static selection mechanism in hybrid predictors for the SPECINT95 benchmarks**

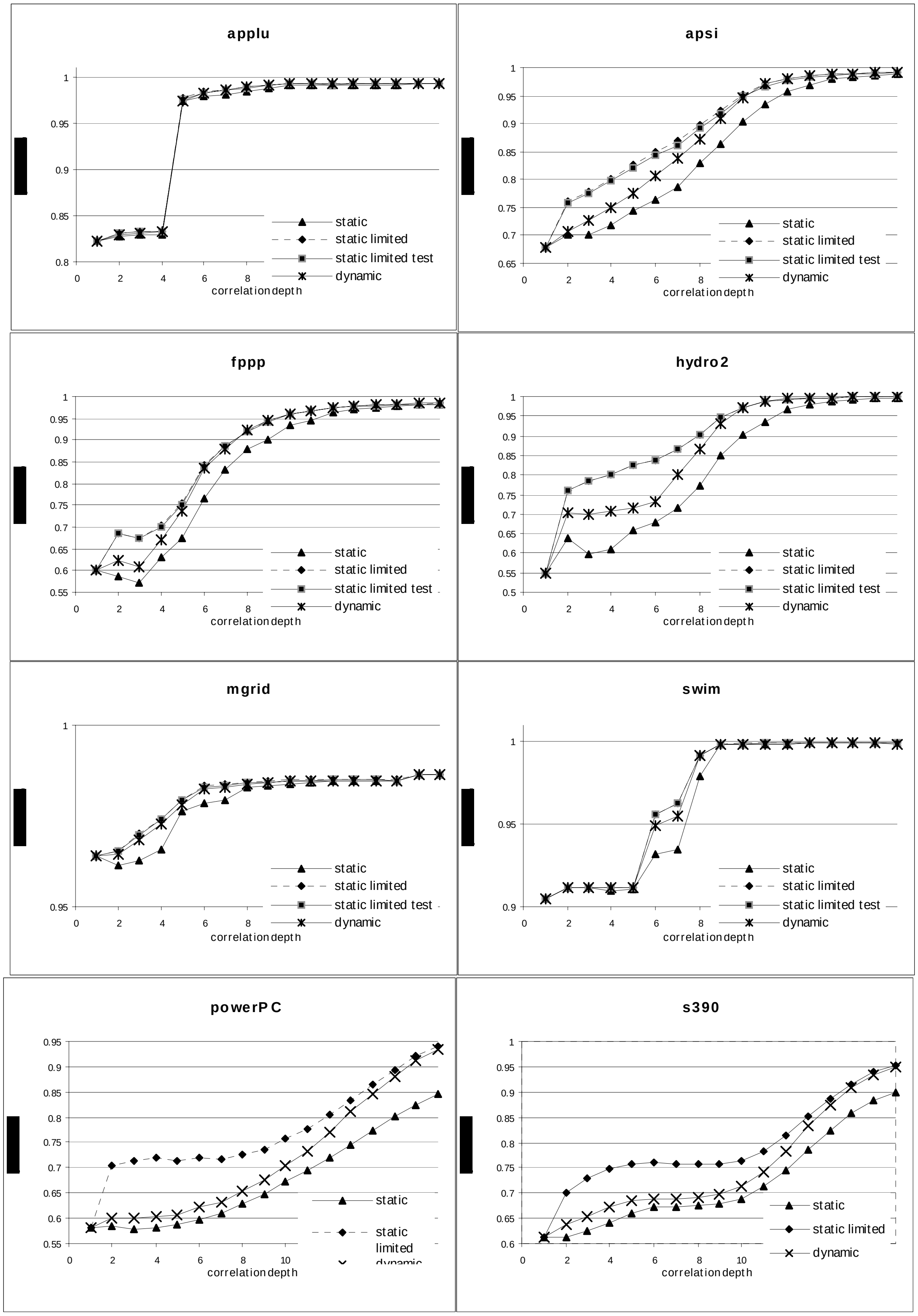


**Figure 5.4 – b) Dynamic vs. aliasing aware static selection mechanism in hybrid predictors for the SPECFP95 and IBM benchmarks**

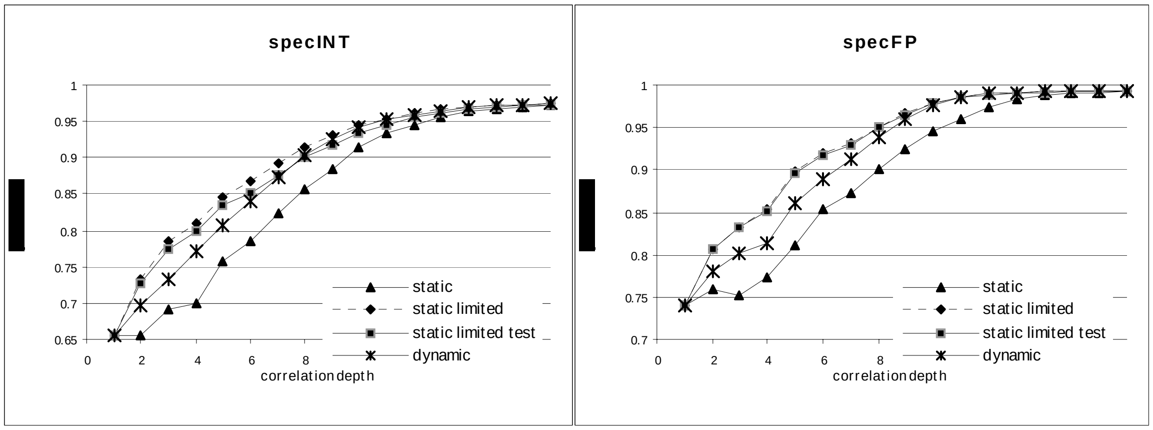


**Figure 5.4 – c) Dynamic vs. aliasing aware static selection mechanism in hybrid predictors for the SPECINT95 and SPECFP95 averages**

on, the aliasing reduction will fall below optimal. This phenomenon might be aggravated when the predictor on which the code is run on and the predictor on which profiling was done implement different structure/s. All that is under the assumption that the ISA was designed to convey the selection information to the processor. Otherwise, changing the ISA is not inconsequential.

In summary, there appears to be no reason why a hybrid predictor should utilize a static selection mechanism over a dynamic one. While both dynamic and static selection mechanisms reduce aliasing, the static selection mechanism has some shortcomings that are hard to make up for.

### 5.3.6 In Depth Analysis

We have shown that the selection mechanism in hybrid predictors enables hybrid predictors to outperform some generic two-level predictors by reducing aliasing. We next present a serious of studies that shed light on the behavior of the selection mechanism throughout the program execution. We choose to only show the results for the gcc benchmark. Results for all other benchmark are similar.

Figure 5.5 depicts the number of switches each static branch goes through for a perfect selection mechanism and a real implementation of the selection mechanism. The perfect selection mechanism is an oracle that after the branch is executed, if the selected component miss predicted the branch and the other component predicted the branch correctly, the oracle registers a switch. The real selection mechanism is implemented as the bimodal selection mechanism. Both the perfect and the real selection mechanism are simulated with no resource limit and with resource limit of 6 KB entry hybrid predictor. Only about 1% of the total static branches show on the x-axis. The static branches are sorted according to the number of switches, and this 1% of static branches represents the majority of overall switches.

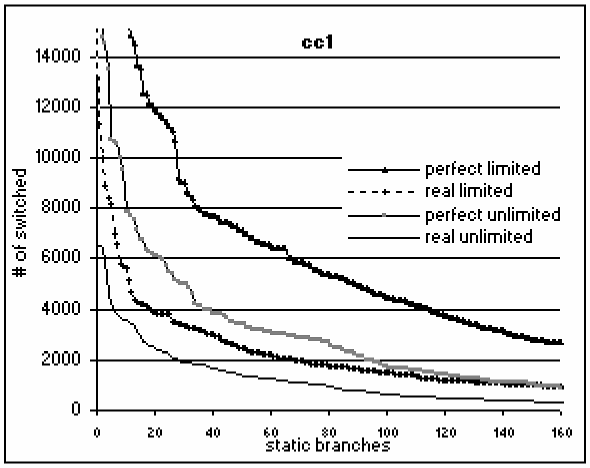


**Figure 5.5 - per branch switches for the McFarling hybrid predictor**

The perfect selection mechanism performs many more switches under resource limit as can be seen by comparing the perfect-limited and perfect-unlimited plots. The difference between those two plots represent the amount of work the selection mechanism will do to reduce aliasing. Similarly, the work the selection mechanism does to reduce aliasing when resources are limited can be viewed when comparing the real-limited and real-unlimited plots.

Comparing the real and the perfect selection mechanism, either for limited or unlimited resources, reveal the potential of the hybrid predictors that is not realizable by the bimodal selection mechanism.

Figure 5.6 displays the accumulated number of switches as the program progresses for a limited and unlimited bimodal selection mechanism. Data was collected for increments of 100,000 dynamic instructions. Each selection mechanism is simulated for history size of 3,6 and 9 for both the local and global components. There is no obvious correlation between the accumulated number of switches and the correlation depth. Correlation depth of 3 has the highest number of switches followed by correlation depth of 6 and 9. However, this varies considerably between different benchmarks. On the other hand, the selection mechanism restricted in size always has a larger number of switches than the unlimited selection mechanism with the same size of correlation. This, once again, suggest that a large number of switches is attributed to aliasing.

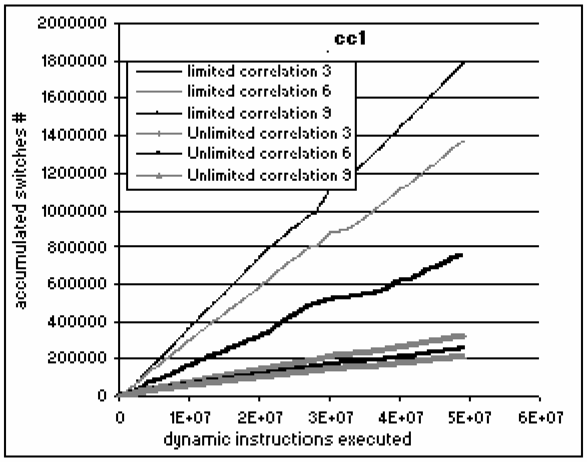


**Figure 5.6 – Accumulated # of switches as program progress**

The linearity of the plots in Figure 5.6 suggests that switches occur throughout the execution of the program evenly, and do not just occur in a warm-up phase only. This coincides with a different experiment we run where we employed a dynamic selection mechanism in the beginning of the program and froze it after an initial warm-up. If most switches occur only in the warm-up phase, such a mechanism will be able to take the advantages of both static and dynamic selection mechanism. The initial dynamic phase will conduct a kind of profiling on the current dataset. The second phase, the static one, will be able to only update one the components for each branch and as a consequence reduce the amount of information stored in the PHTs. This mechanism failed to outperform a conventional dynamic selection mechanism. As figure 5.6 depicts, this static-dynamic selection mechanism failed because the dynamic switches occur evenly throughout the execution of the program.

Figure 5.7 shows the number of switches performed by the selection mechanism as a function of correlation depth for a limited resources and unlimited resources McFarling predictor. Figure 5.6 shows the same date but only for correlation depth of 3,6 and 9, and a trend could not be established. In Figure 5.7, however, the trend is clear. The number of switches decreases as correlation depth increases. For the unlimited selection mechanism this decrease in switches is moderate and the reason is that as the depth of correlation increases, the prediction accuracy in both the global and local increases. As the prediction accuracy increases, the need

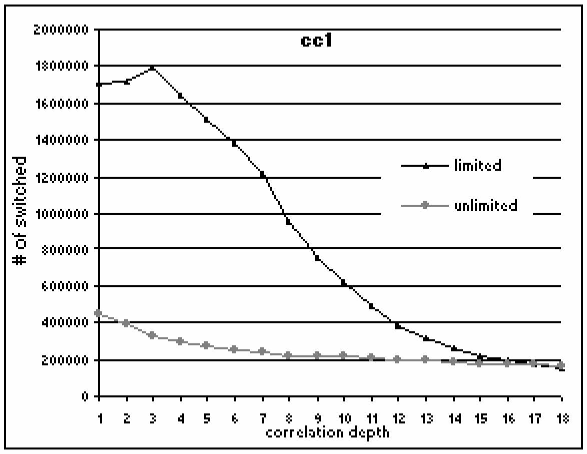


**Figure 5.7 – number of switches as a function of correlation for limited and unlimited predictors**

for switches decreases.

In the limited resources simulation the decrease in switches as correlation increases is much more pronounce than in the unlimited simulations. On top of the increase in prediction accuracy as correlation depth increases, as in the unlimited simulation, the size of the predictor increases as well in the limited resource simulation. The increase in the size of the predictor reduces the amount of aliasing, and as a consequence the number of switches due to aliasing decreases. When aliasing cease to be a problem, the limited and unlimited resources predictors' switches converges. The moderate decrease of switches in the unlimited predictor represents the elimination of switches due to increase in prediction accuracy as correlation depth increases. The difference between this moderate decrease and the rapid decrease in switches of the limited predictor is due to elimination of aliasing.

## 5.4 The Notion of Hybrid Predictors

The next issue to address is whether there is an inherent gain in the local-global hybrid predictor over a single scheme, or whether the gain realized by the hybrid predictor is limited to reducing aliasing. Figure 5.8 shows the improvement of the program's prediction for each branch (x-axis) when using the local predictor versus the global predictor with no limits on resources. Positive percentages indicate that the branch is better predicted by the local scheme, while negative percentages indicate the branch is better predicted by the global scheme. The branches are sorted on the x-axis according to the percentage improvement.  Figure 5.8 shows that the number of branches that contribute to the true hybrid behavior of the local-global hybrid predictor is small.  These small number of branches will be referred to hereinafter as the hybrid branches.  For most branches, the improvement obtained by using the global component instead of the local component or vice versa is insignificant.

Only a few of the hybrid branches are responsible for the improvement of a local-global hybrid predictor over a single scheme predictor.  If the predictor component for the other branches, which make up the majority, changes dynamically to reduce aliasing, it remains to make sure that the hybrid branches are predicted by the component that does it best. This will allow the predictor to take advantage of both alias reduction and true hybrid behavior.  When employing a static selection mechanism, this can be done at profile time. In the case of a dynamic selection mechanism, it seems that an explicit way of indicating the appropriate component for the hybrid branches is needed.  However, a study conducted but not shown here indicated that the dynamic selection mechanism already performs the task of mapping the hybrid branches into their respective best components. Attempting to lock the hybrid branches into their respective best components, while letting the rest of the branches' components to be chosen dynamically, resulted in degraded performance.

Despite the potential embedded in hybrid predictors and the ability of the selection mechanism to identify the hybrid branches, this potential is not fulfilled.  Performance degradation due to aliasing dominates the hybrid potential.

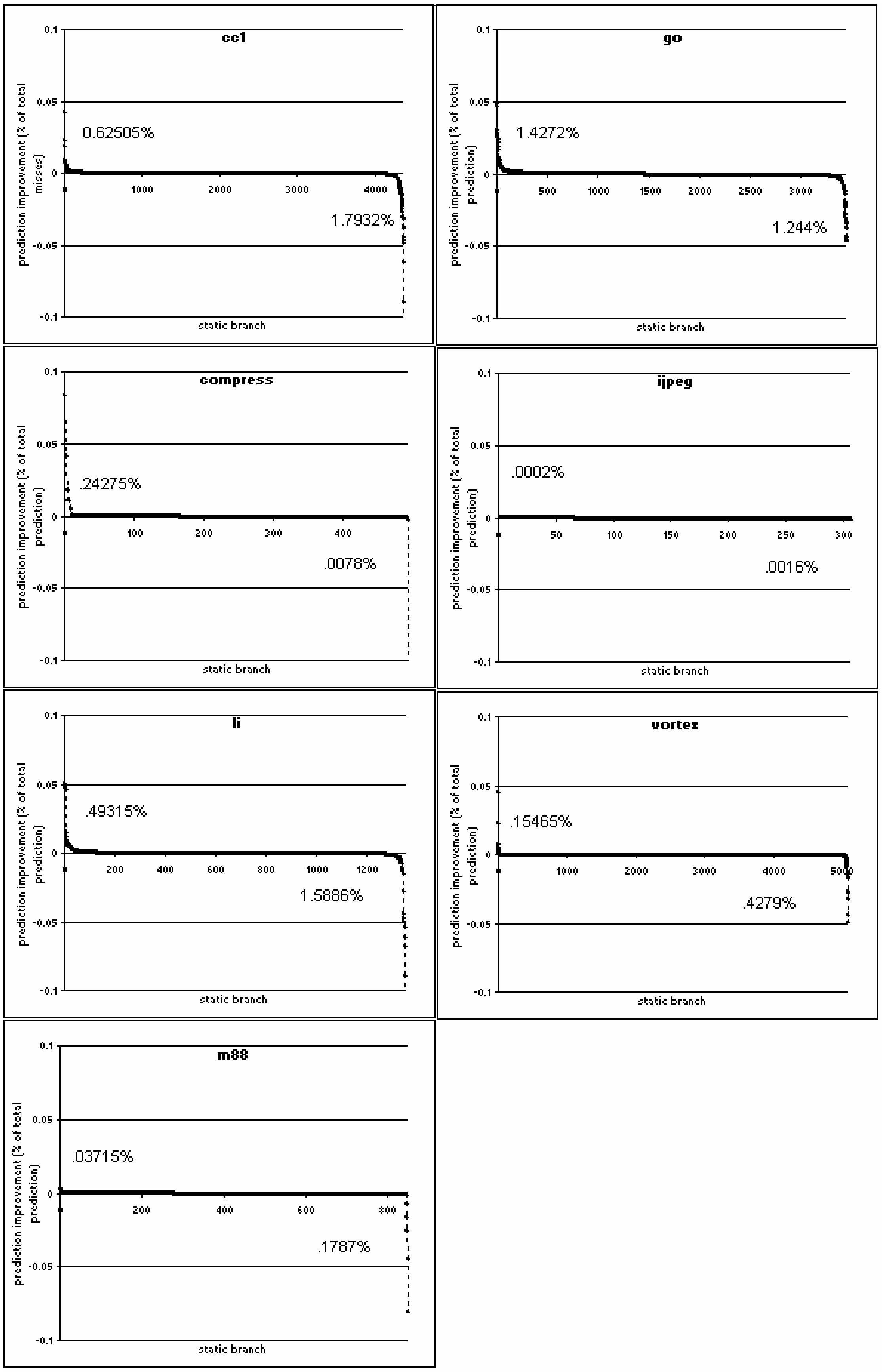


**Figure 5.8 – a) per branch potential of a local-global hybrid predictor for the SPECINT95 benchmarks**

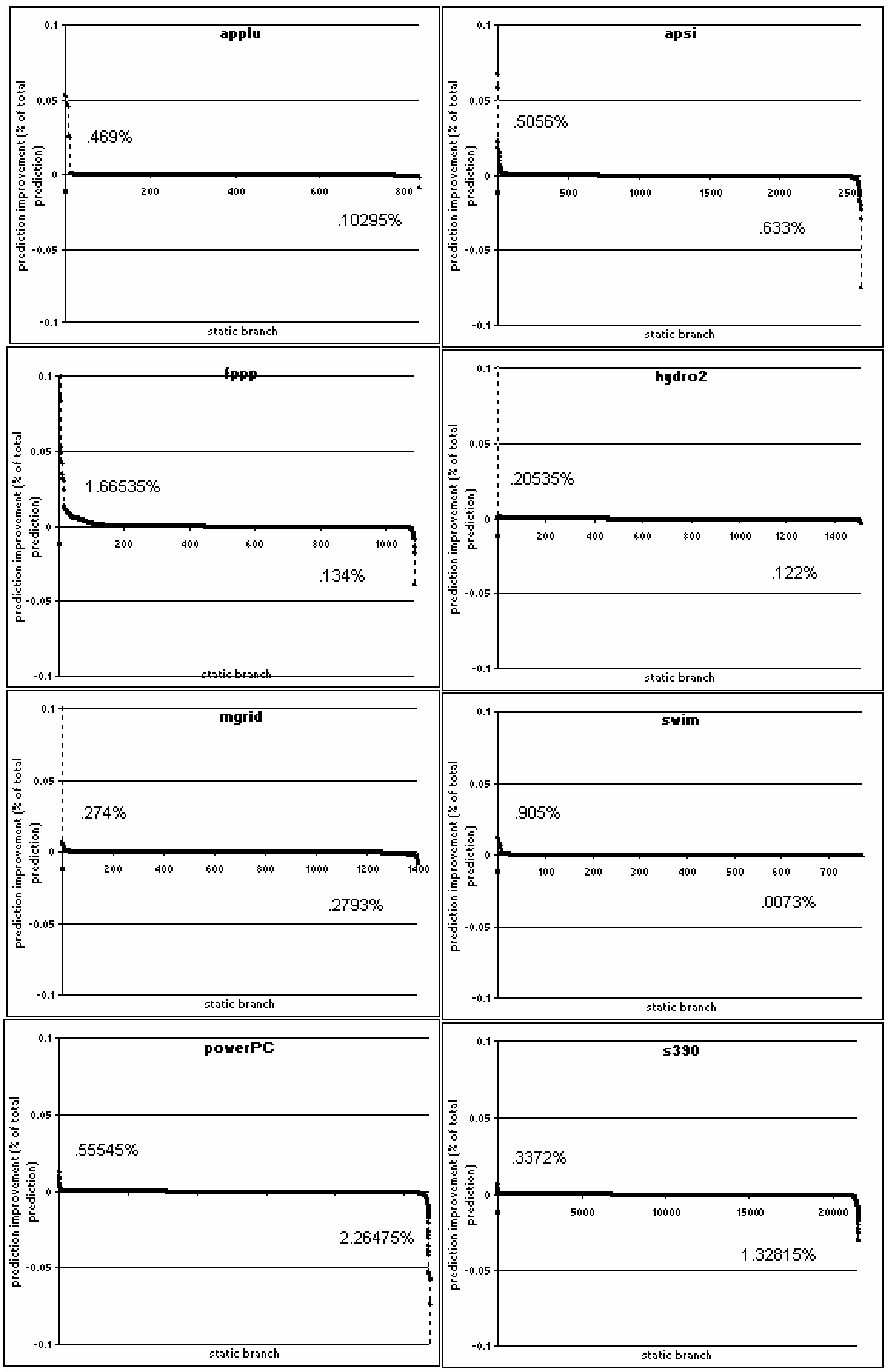


**Figure 5.8 – b) per branch potential of a local-global hybrid predictor for the SPECFP95 benchmarks**

## 5.5 Updating Policies and Aliasing

As was mentioned before, a static selection mechanism has serious shortcomings. One mechanism to overcome these shortcomings might be to bring the advantage of the static selection mechanism into hybrid predictors with a dynamic selection mechanism. An attempt is made to accomplish this using a modified updating mechanism. When a branch is resolved, the branch predictor is updated with the branch outcome. In a hybrid structure, this entails updating both the global history register and the respective local history register, and the PHT for both of the hybrid components. In order to reduce contention in the PHT, the updating mechanism should update only the PHT for the component currently selected. Figure 5.9 depicts the prediction percentage as a function of the $\log_2$ of the PHT size and the correlation depth for three updating policies. The updating policies are: 1) both, where both PHTs are updated; 2) lgt - stands for Local-Global hybrid with a “this” updating mechanism. With the “this” updating mechanism only the current PHT pointed by the selection mechanism is being updated; and 3) our proposed new updating policy, lgnt - stands for Local-Global hybrid with the “this & next” updating mechanism. This updating mechanism will be described later. Figure 5.9 shows that for small size predictors, it is beneficial to update only the current PHT. This update policy reduces the amount of information stored in the PHT and therefore reduces contention, which in turn helps the prediction accuracy. As the size of the predictor increases, updating both components helps the prediction accuracy. This suggests that updating both components produces helpful information for prediction. This observation leads to the question of whether this helpful information can be captured without

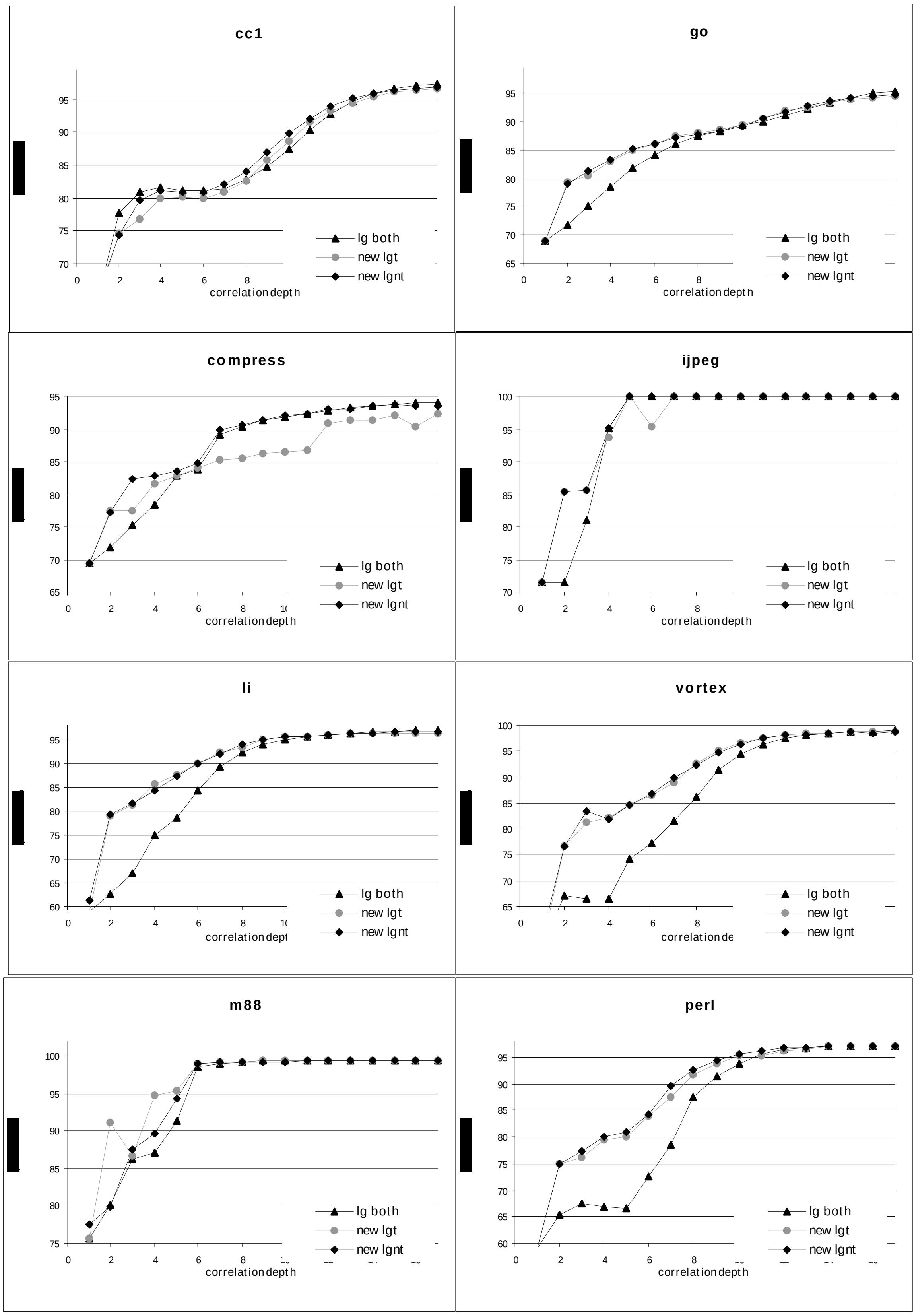


**Figure 5.9 – a) Updating policies in hybrid branch predictors for SPECINT95**

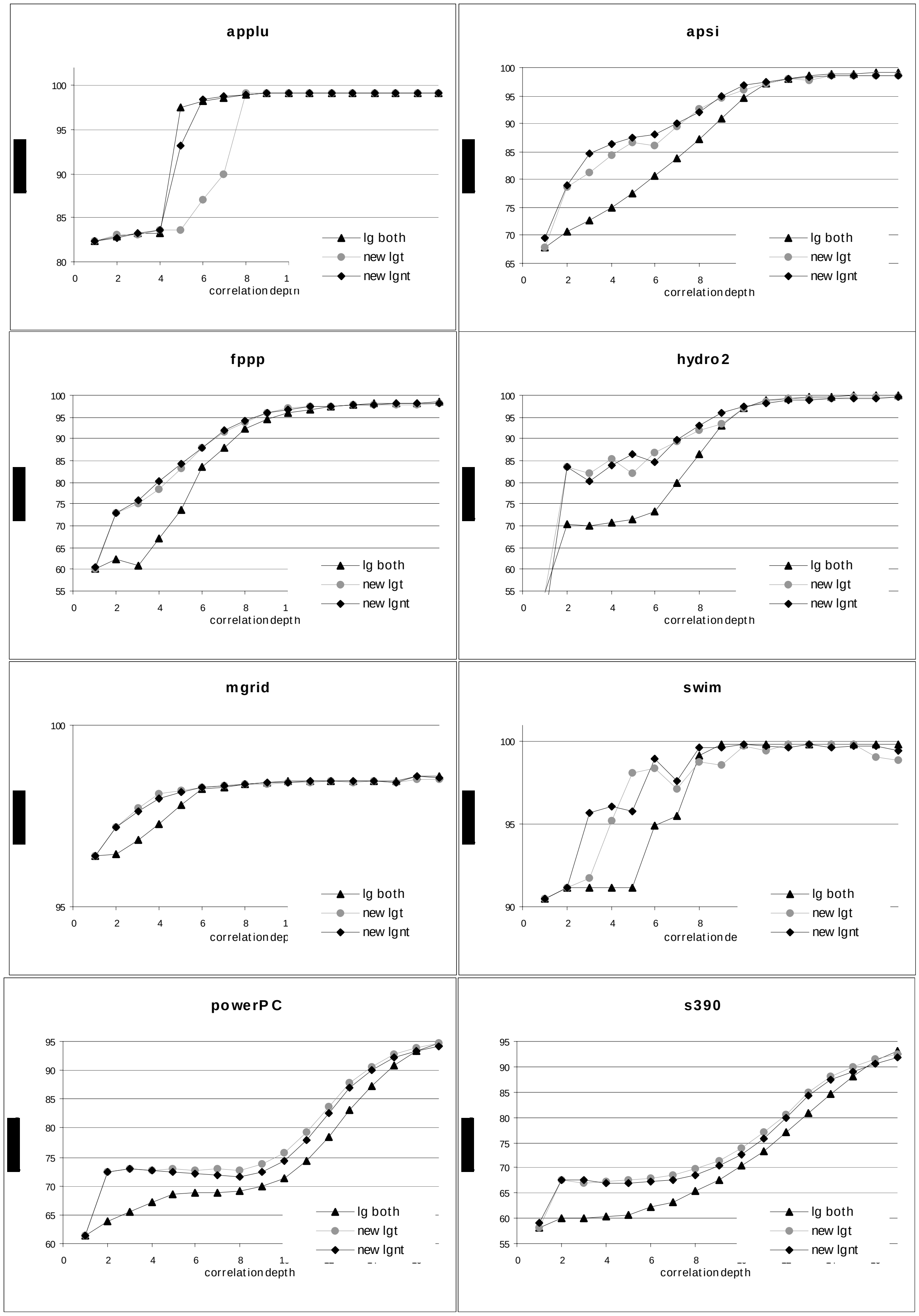


**Figure 5.9– b) Updating policies in hybrid branch predictors for the SPECFP95 and IBM benchmarks**

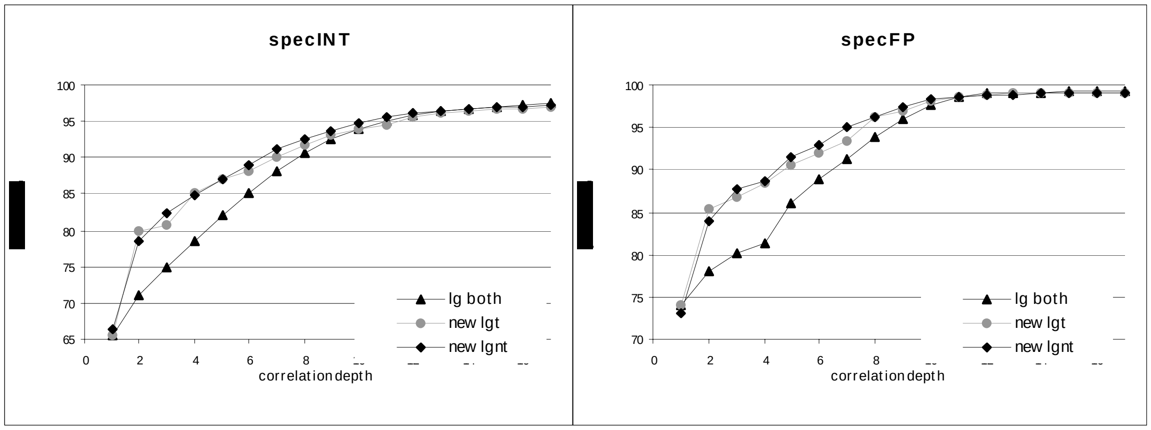


**Figure 5.9– c) Updating policies in hybrid branch predictors for SPECINT95 and SPECFP95 averages**

recording double the information for each branch instance. Notice that for programs with large numbers of branches like the S390 and the PowerPC traces, the tradeoff between aliasing and incorporating the additional useful information favors adding the useful information only for structures of infeasible size.

The new updating policy (lgnt) described next was developed to resolve the problem defined above, and attempted to capture the useful information of the "both" update policy, while alleviating contention in the PHT. The *lgnt* policy updates only the PHT being used currently unless the selection mechanism is in a transition mode where it updates both PHTs. A transition mode is defined when a branch selection points to one component in the hybrid predictor, but the branch resolution will shift it to point to the other component. As demonstrated in Figure 5.9, the updating policy achieves good prediction for small predictors compared to the other two policies, and does not lose its effectiveness for larger predictors. For traces with a large branch signature like the S390 and the PowerPC, *lgtn* stills falls short of the *lgt*, but it cushions the worst case compared to the “both” updating policy. In summary, the *lgnt* updating policy serves as good middle ground between the other two updating policies.

## 5.6 Combining Aliasing and Hybrid Paths

This dissertation shows that the hybrid and aliasing paths are one and the same. Prior to the studies conducted here, it was believed that because those two paths were orthogonal that their advantages would be easily combined. To double check on this premise, we next try to combine the hybrid path with the *bi-mode* predictor. Figure 5.10 depicts the performance of a McFarling predictor, where both the local and global components are implemented as a *bi-mode* structure. If the hybrid and the aliasing paths were orthogonal, such a predictor would have had the potential to take advantage of both paths. Specifically, if the selection mechanism is static and uses an unlimited structure when profiling, each branch will be mapped to the component which best predicts it. Within each component, the *bi-mode* structure should perform the task of reducing aliasing. The performance of such predictor is shown in Graph 5.7 under “static-unlim-bimode”. The static-unlim indicates that selection is done statically with profiling performed on a structure of unlimited size. This structure is compared against several other predictors and consistently does worse than most. In fact the static-unlim-bimode consistently outperforms the same predictor without the *bi-mode* structures in each component, referred to as static-unlim-normal. This indicates that the *bi-mode* structure in each component of the McFarling predictor, helps performance by reducing aliasing. More importantly, however, it indicates that using no limit on the structure size during profiling degrades performance considerably. The inability of this combined predictor to

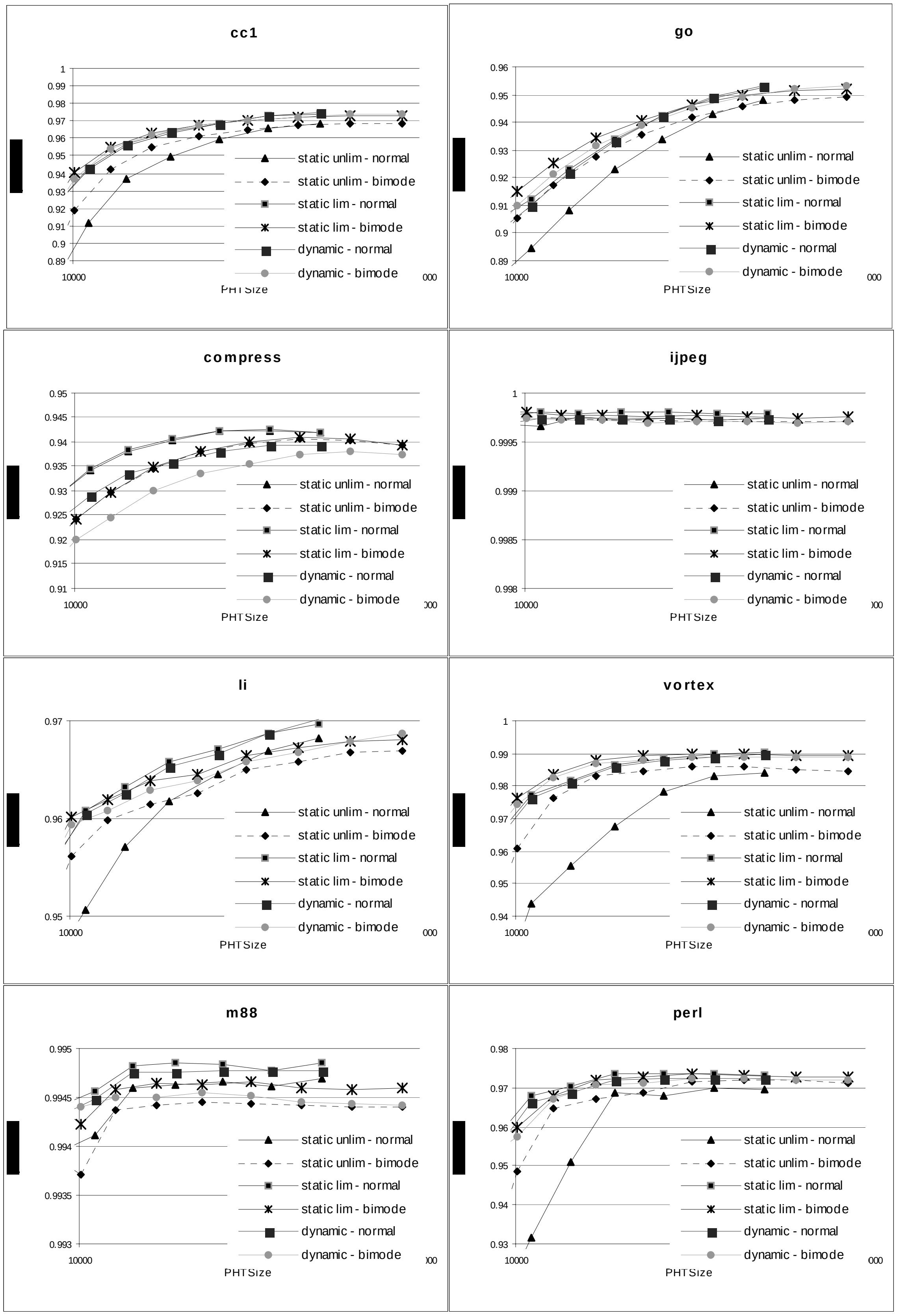


**Figure 5.10 – a) Combining McFarling and bi-mode predictors for SPECINT95**

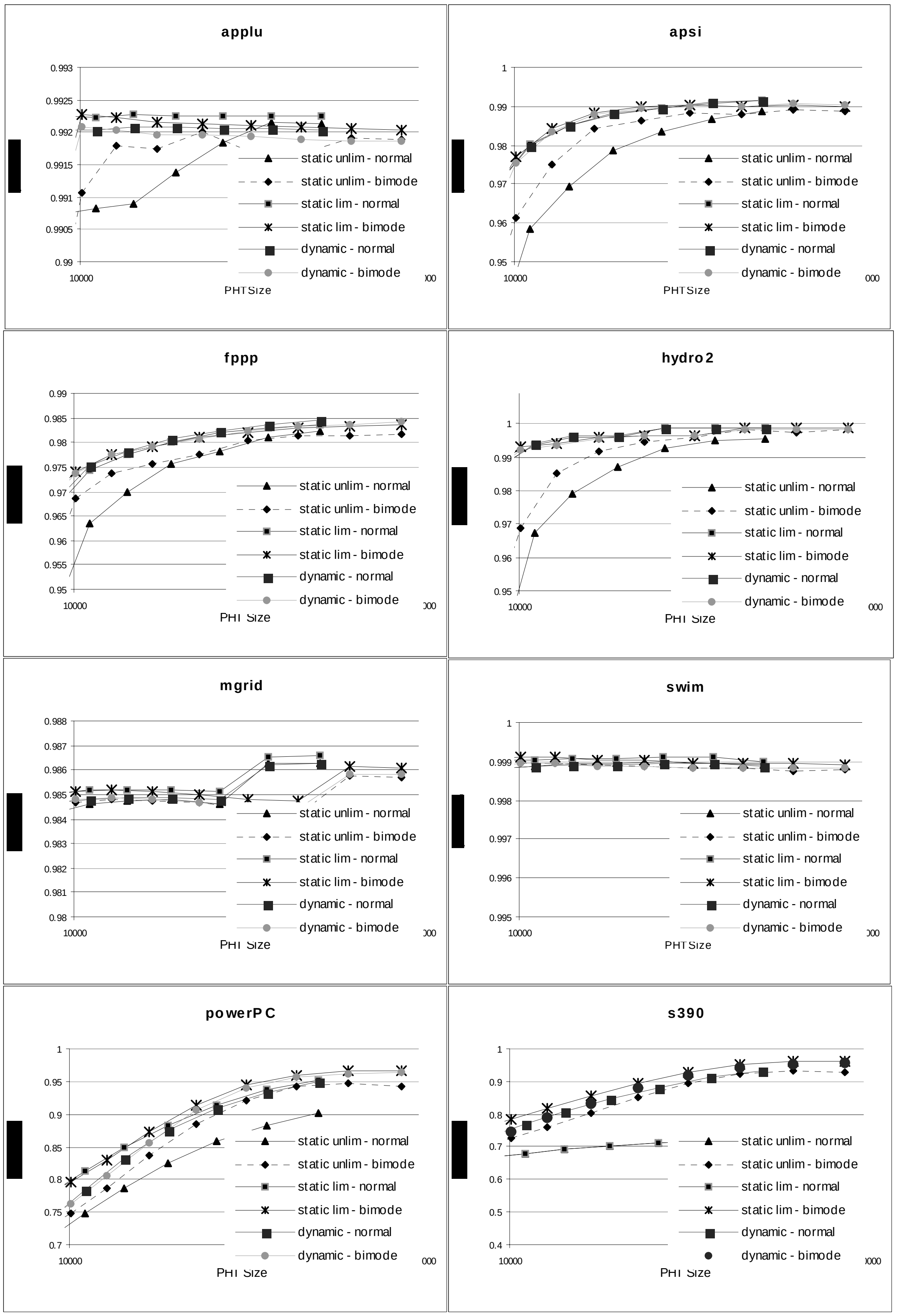


**Figure 5.10 – b) Combining McFarling and bi-mode predictors for SPECFP95 and IBM benchmarks**

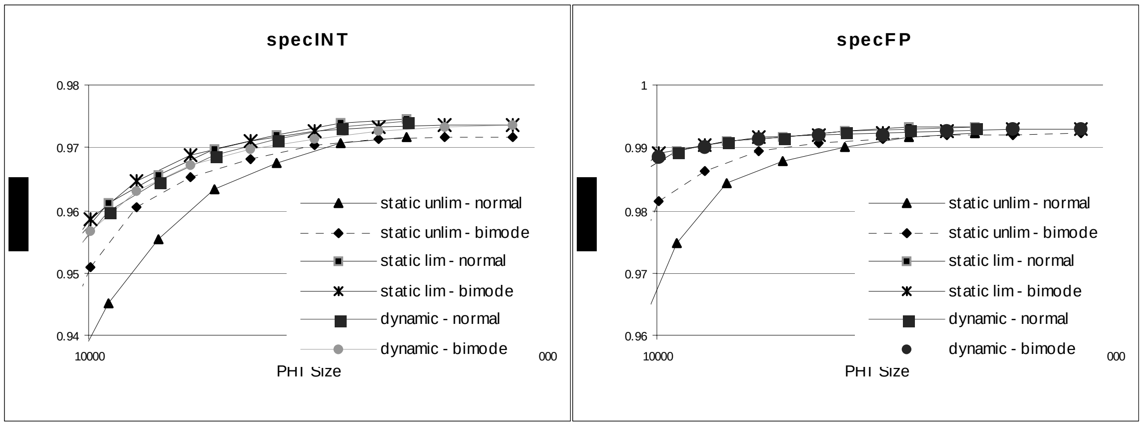


**Figure 5.10 – c) Combining McFarling and bi-mode predictors for SPECINT95 and SPECFP95 averages**

approach the prediction accuracy of the classic McFarling predictor (dynamic-normal) demonstrates that contrary to common belief, combining the advantages of the hybrid and aliasing paths of research is not trivial.

## 5.7 McFarling vs Bi-Mode Predictor

Finally, after discovering that the main strength of hybrid predictors is reducing aliasing, this study makes a direct comparison between one of the most used aliasing reduction implementations, the *bi-mode* predictor, and the McFarling hybrid predictor. For the study conducted here, the McFarling predictor was implemented with a 2K entry BTB. The size of the local history registers was accumulated into the overall predictor size. However, the BTB tags were not considered when calculating the predictor size. This was done under the assumption that the BTB tags were already in place for predicting the branch target address, and therefore could be used for predicting the direction with at no extra cost.

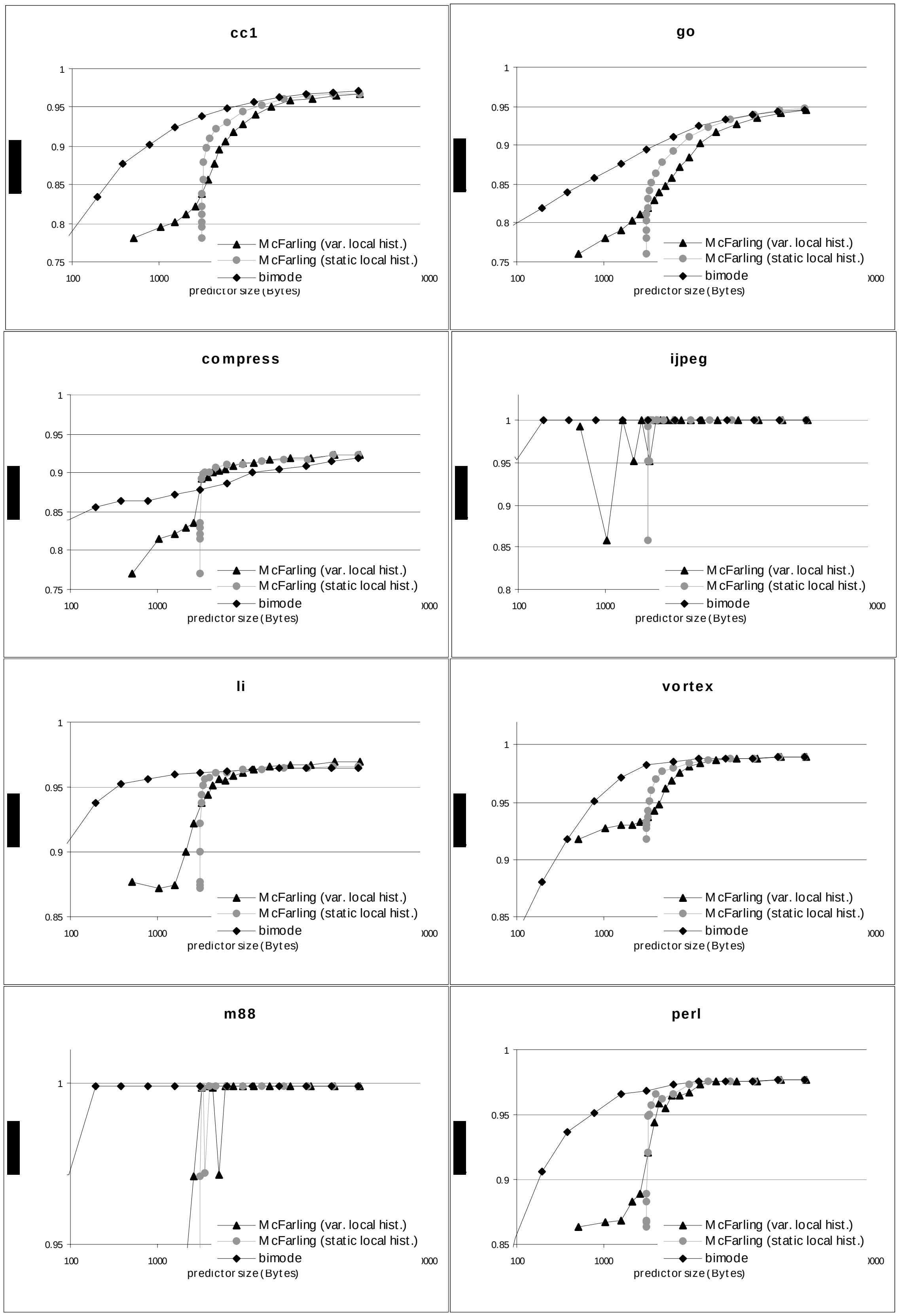


**Figure 5.11 – a) McFarling vs. bi-mode for SPECINT95**

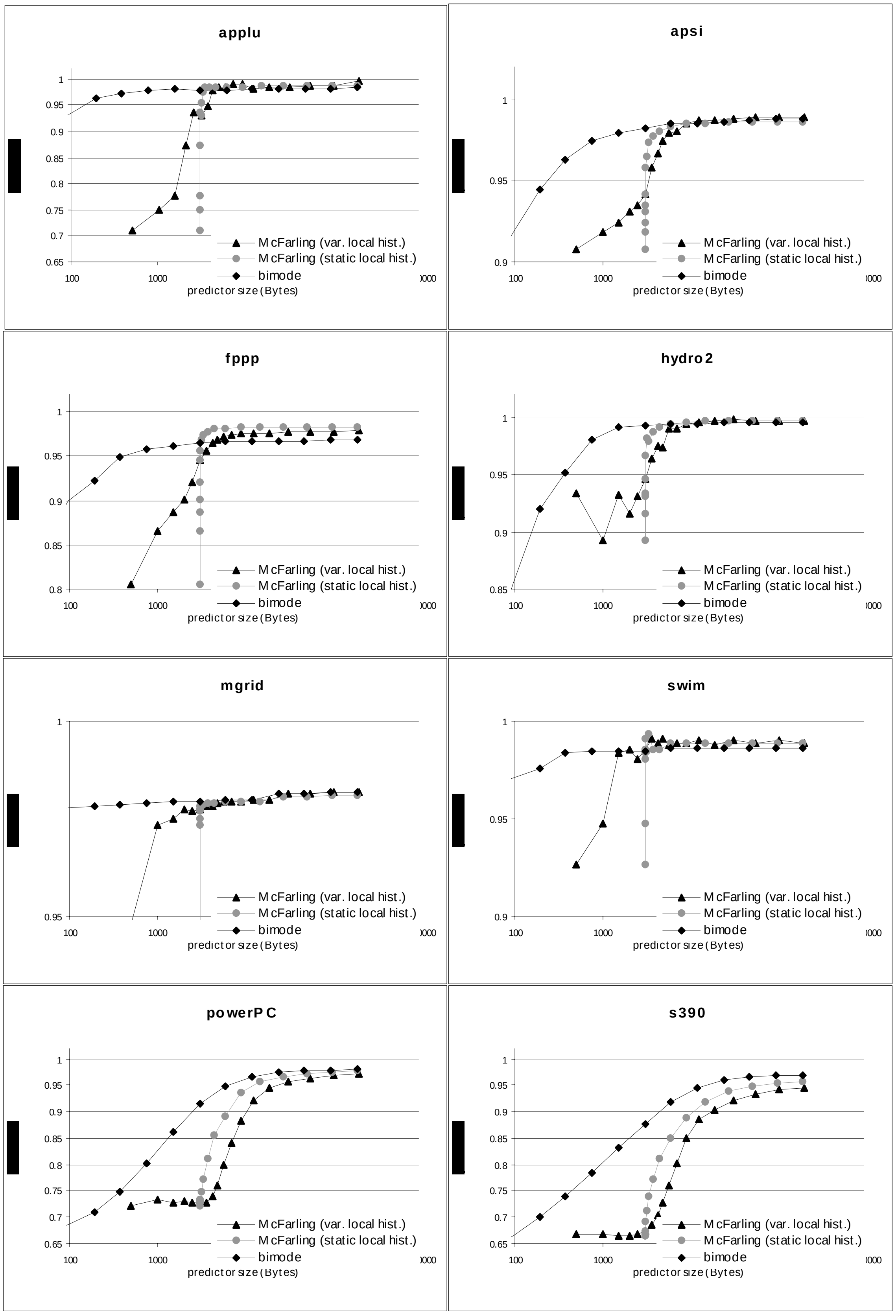


**Figure 5.11 – b) McFarling vs. bi-mode for SPECFP95 and IBM benchmarks**

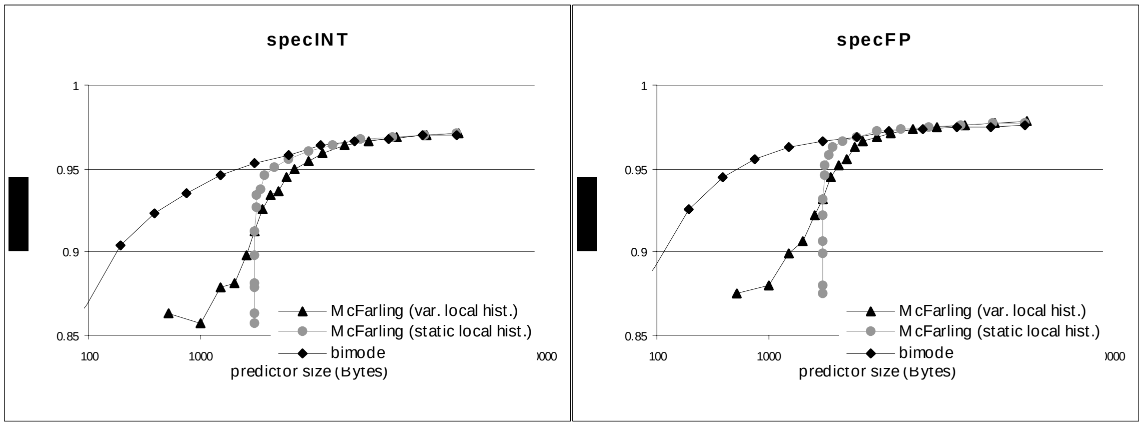


**Figure 5.11 – c) McFarling vs. bi-mode for SPECINT95 and SPECFP95 averages**

Figure 5.11 makes a direct comparison between the McFarling predictor and the *bi-mode* predictor. Two versions of the McFarling predictor are shown. The first version has increasing local correlation depth corresponding to the correlation increase of the global history register. The second version of the McFarling predictor utilizes the best local correlation depth for each benchmark.

The *bi-mode* predictor outperforms the McFarling predictor for small predictor size across all benchmarks. The size overhead of the local registers, which are part of implementing the local scheme, cannot be offset for small predictors, and therefore the *bi-mode* outperforms the McFarling predictor for small predictors. As the size of the predictors increases, the McFarling prediction accuracy caches up with the *bi-mode* predictor. On average for the SPECINT95 average, the McFarling predictor outperforms the *bi-mode* predictor only for sizes larger than 26KB. This is much larger than can be implemented in future processors, as will be discussed in Chapter 7. For benchmarks with large static branch signatures, such as the s390 and PowerPC, the *bi-mode* predictor outperforms the McFarling predictor even for predictors as large as 180KB.

## 5.8 Summary

This chapter eliminates the notion of the hybrid path as an independent research path in branch prediction research. It shows that most of the gains achieved in hybrid predictors are attributable to the ability of the selection mechanism to reduce aliasing, and not to true hybrid behavior. It follows that hybrid predictors should be compared against aliasing-reducing structures and vise versa, because they both achieve their goals by attacking the same problem. True hybrid behavior can be attributed to a limited number of branches, but both dynamic and properly profiled static selection mechanisms map those branches into their respective best components. It is shown that both dynamic and static selection mechanism achieve the same goals, namely, reducing aliasing, in different ways. This chapter also shows that the advantages of dynamic selection mechanisms can be applied to static selection mechanisms by a profiling method, and that the advantages of static selection mechanisms to the dynamic ones by means of a new updating policy. This chapter concludes by comparing the *bi-mode* predictor with the McFarling predictor. This comparison between a well-known aliasing-reducing structure and an equally well-known hybrid predictor shows that the *bi-mode* is a considerably better predictor.

# Chapter 6 - Filtering Characteristic of the Third-Level of Adaptivity

4Compared to the hybrid research path, the third-level path is understudied. Those few studies that have addressed it show empirical results that prove the respective third-level of adaptivity structure outperforms *gshare*. It is not clear from those studies whether applying a third-level of adaptivity improves prediction because of better usage of resources and aliasing reduction, or whether the improvements show that the third-level of adaptivity is a better branch prediction scheme. In this chapter we attempt to answer this question.

The literature describes four third-level-of-adpativity structures. The first, the Elastic History Buffer (EHB), statically determines, via profiling, the correlation size used for each static branch [20]. The second, branch classification, decides statically between two predetermined correlation sizes for each branch [13]. The third structure, Dynamic History Length Fitting (DHLF), dynamically adjusts the correlation size for all the branches [21]. Lastly, a variable length path branch predictor was considered [54]. Similar to the EHB, profiling is used to determine the depth of correlation for each branch. The interested reader should refer to Section 1.5 for a detailed description of the above methods.

Next branch classification and the DHLF are considered. Limit studies were performed for each to determine whether they represent a better branch prediction scheme or just a better branch prediction structure. We didn’t consider the fourth structure, as we consider it to be somewhat orthogonal to other structures, and the improvements discussed for this structure were for indirect branches only. Different branch prediction structures can therefore incorporate a similar structure that predicts only the indirect branches and enhance the predictors performance. The EHB was not considered because it spans two different research paths. Not only does the EHB allow each static branch to use its best correlation, but it also filters easy-to-predict branches out of the PHT. This places the EHB in both the third-level path and the aliasing path[3]. No breakdown of the accuracy improvement was given in the EHB study to determine what percentage of the prediction improvements is due to filtering and what percentage of it is due to the third-level of adaptivity scheme employed.

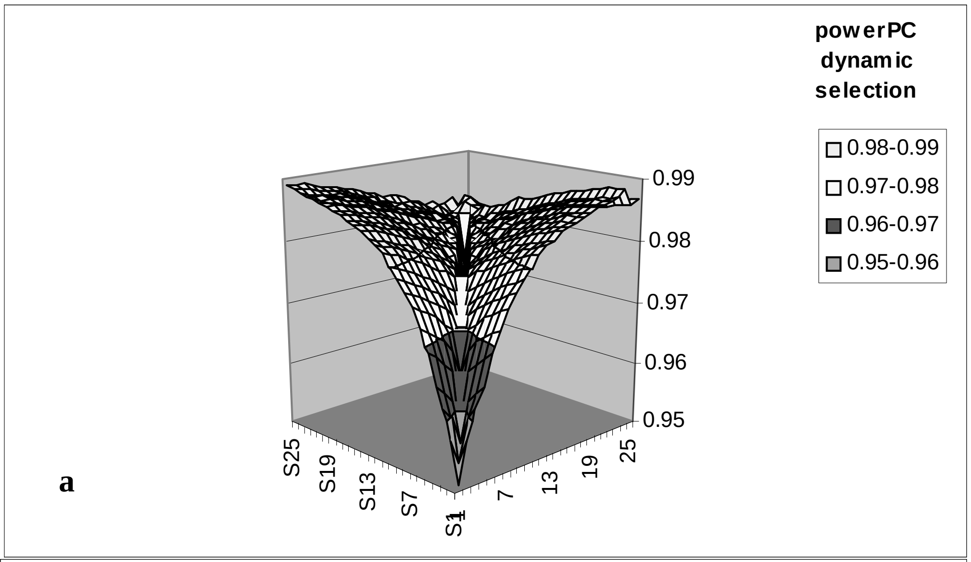
To the best of our knowledge, all third-level adaptivity structures implement the global two-level branch prediction scheme. All simulations in this chapter, therefore, consider only the global two-level branch prediction scheme.

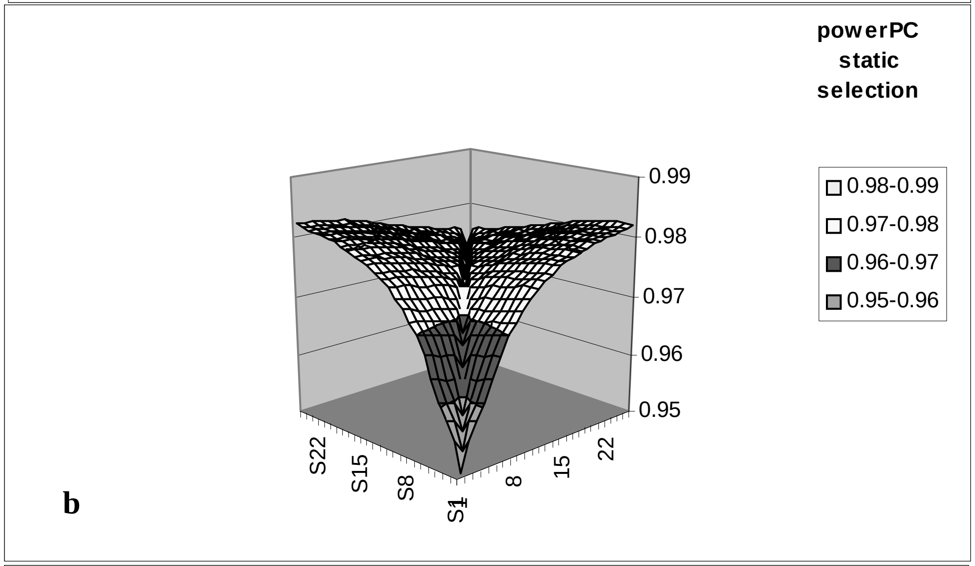
## 6.1 Branch Classification

As was mentioned before, branch classification can be viewed as belonging to the hybrid path or to the third-level path. Using the conclusions drawn in chapter 4, one can get a good idea of what makes branch classification work when it has static or dynamic selection mechanism.

---

[3] Interestingly enough, since the EHB presented itself as a third-level path, it was not regarded as employing filtering as well. When the filter mechanism was introduced, there was no mention of the EHB, or a proper comparison between the filter mechanism and the EHB, which appeared first.





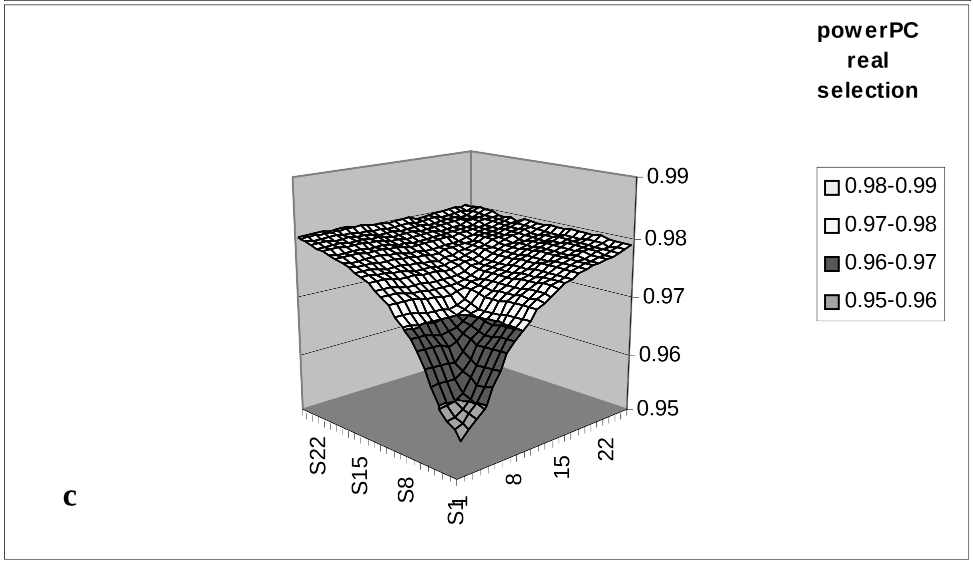


**Figure 6.1 – Branch classification limit study with correlation depth ranging from 1 to 25 for the PowerPC benchmark**

We next however, check the validity of the third-level adaptivity motif in branch classification. Figure 6.1 depicts a limit study of different possible configurations of branch classification ranging from a correlation depth of 1 to a correlation depth of 25 for a) a dynamic oracle b) a static oracle, and c) a real selection mechanism for the PowerPC benchmark. The real selection mechanism is implemented as the bimodal selection mechanism, similar to the one used in the hybrid studies. Each point represents the prediction accuracy for a classification method with one component having correlation depth of x on the x-axis, and the other component having a correlation depth of y on the y-axis. The components are not restricted in size and each vector has its own unique 2bc state machine for the *bimodal* selection mechanism.

Figure 6.1 shows that the dynamic oracle selection mechanism performs better than the static oracle. This is to be expected since the dynamic oracle represents an upper limit on prediction that is not likely to be realized. Once again, similar to Section 5.3.1, we encounter the problem of how to cancel the noise produced by two different lists as oppose to capturing the real advantage of using two different sizes of correlation. This problem will be discussed later.

Both the dynamic oracle and the static oracle selection mechanisms depict a jump in prediction accuracy when using different correlation depth for each component compared to using the same correlation depth for both components. Using the same correlation depth for both components is represented by the middle diagonal. This accuracy jump might lead to the conclusion that third-level adaptivity has merit as a scheme and not only as a structure. It was also found that the prediction accuracy of two widely spaced correlation depths, such as 1 and 25 is higher than that of adjacent correlation depths, such as 24 and 25. We know that deeper correlation results in better prediction. Therefore, having combination of low and high correlation depths surpass a configuration where both components are of high correlation suggests the merit of third-level adaptivity as a scheme.

The results presented in the real selection mechanism graph in figure 6.1, however, result in exactly the opposite conclusion. In the real selection mechanism, there is no degradation in performance when both components use the same correlation depth. The real selection mechanism performs as well whether the two components use the same or different depths of correlation. This suggests that there is no merit attributable to the third-level path as a branch prediction scheme. Further proving this point, the relation that deeper correlation results in better prediction holds for the real selection mechanism. As a result, the best prediction accuracy is obtained for the deepest correlation size – the best prediction is achieved when both components are of depth 25. There is no advantage of using two different correlation sizes for the two components in the branch classification scheme, when a real selection mechanism is employed. When there is no size restriction on the predictor, there is no advantage to using two different components at all.

What is it about the dynamic oracle and static oracle limit studies that suggested to a different assumption? The introduction of two lists enhances the prediction accuracy even if the two lists are random and have nothing to do with prediction as discussed in Section 5.3.1. The reason it seems that two very different correlation sizes can enhance prediction is because those two components are so different from one another. It is clear that this phenomenon is more pronounced for the dynamic selection oracle than for the static selection oracle.  Again, this supports the assertion that the improved accuracy comes from random lists rather from inherent potential in the third-level branch prediction scheme.

We were unable to find a way to separate any potential advantage in introducing two different lists of prediction from the actual advantage held by the third-level scheme.  It is possible that such separation cannot be done and that the only potential merit of the third-level path lies in a novel real selection mechanism that would expose it.  If no such selection mechanism exists, any merit that it might have exhibited will be moot.

It is clear, however, that known selection mechanisms cannot take advantage of this elusive inherent advantage of using classification with no limits on resources.  However, it can be inferred from results obtained in Chapter 4 that when a size restriction is involved, branch classification will work due to the ability of the selection mechanism, either static or dynamic, to reduce aliasing.

Throughout this dissertation, we demonstrate how the lack of a limit study can result in misleading conclusions for researchers.  This Section shows, however, how a limit study might itself lead to erroneous conclusions.  We advocate using limits studies cautiously.

## 6.2 Dynamic History Length Fitting

The motivation behind the Dynamic History-Length Fitting (DHLF) is the empirical observation that different programs achieve maximum prediction accuracy by

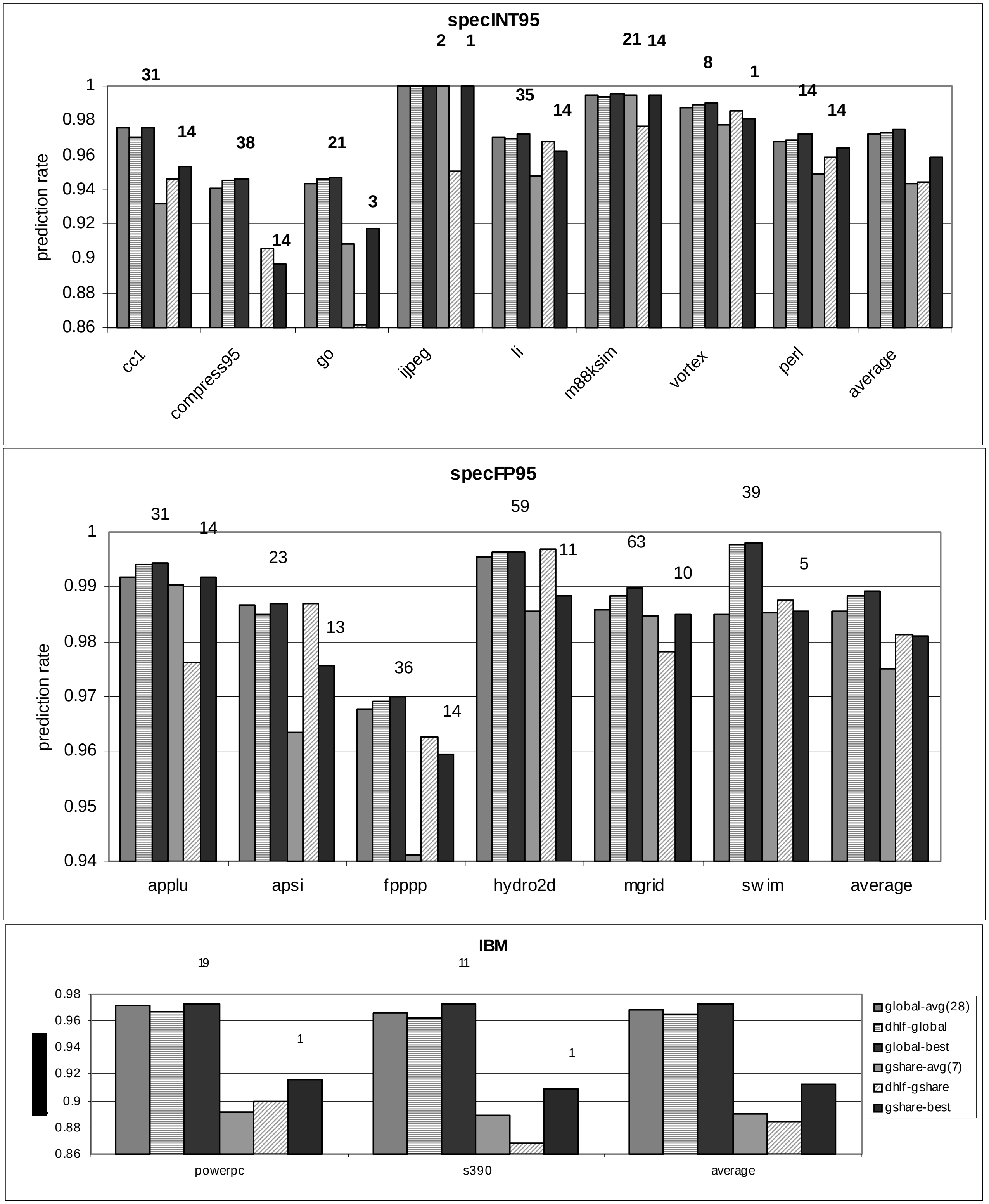


**Figure 6.2 – DHLF comparison in as a scheme and structure**

employing different sizes of history register [21]. This observation was made before in [19] without the proper explanation. In Section 4.5 we provided an explanation as to why predictors have different optimal history sizes for different programs. The example given that li and go have different optimal history register holds true, as Figure 6.2 depicts. The *gshare* predictor achieves its best prediction accuracy for the li benchmark when the history size is set to 14, in comparison to history size of 3 for the go benchmark for an 8K entries *gshare*.

To take advantage of this observation, the DHLF dynamically adjusts the size of the history register during program execution to optimize the register for each program. This is achieved by dividing the dynamic stream of branches into sub-streams termed steps. In every other step the length of the history register is evaluated and might change if the evaluation method finds the change beneficial. The evaluation is done only every other step to omit the effects of cold starts from getting in the way of the evaluation. The step is set to 16K branch instructions.

While the DHLF adheres to the empirical results discussed above, an examination of a limit study indicates that both the li and the go benchmarks could benefit from longer history sizes. The go benchmark can benefit from a history size of up to 21, and the li benchmark can use up to 35. This phenomenon is consistent with the relationship depicted in Figure 4.5 and described in Section 4.6. In other words, more branches and deeper correlation result in more information, which results in more aliasing. More aliasing, in turn, tends to reduce prediction. On the other hand, deeper correlation tends to improve prediction. As a result of those two conflicting relationships, benchmarks with fewer branches can usually benefit more from deeper correlation depth.

Figure 6.2 depicts a limit study and 8K entries *gshare* that give some insight into the inherent advantage of using the DHLF scheme and structure. The first three bars compare the best global scheme across benchmarks (global-avg), the DHLF (dhlf-global), and the best global history size for each benchmark (global-best), in that order. All of the first three graphs depict a limit study, and the structures therein were not subject to size restrictions. Bars four to six, on the other hand, depict the same information, but for resource limit of 8K entries *gshare*. This is the same size used in the original DHLF paper [21]. Results seem to vary from one benchmark to another, however a few observations are possible. First, on average, the DHLF performs worse than the global-avg when no resource limit is imposed and better when resource limit is imposed. Notice that in the global-avg we use an oracle to determine the best history size for each benchmark. This kind of oracle is not realistic. The comparison between the global-avg and global-dhlf, though, highlights the role of the DHLF in reducing aliasing. When no resource is imposed the DHLF has no advantage over the overall best history size. On the other hand, when resources are limited, and aliasing is a factor, the DHLF gains an advantage. While this is true across the averages, different benchmarks show different behavior. For example, for the mgrid benchmark, the DHLF performs better than the global-avg, even when there are no resource restrictions. Other phenomena might explain the fluctuation in behavior. The DHLF starts with a short history size and grows accordingly. It can therefore better bear the negative effect of a cold start. On the other hand, the dynamic size of the history register can cause a degradation in performance if the history size keeps thrashing between different history sizes.

The second observation derived from Figure 6.2 is that when comparing the DHLF to the best history size for each benchmark (global-best), the DHLF has no advantage. Of course, global-best or its restricted counterpart *gshare* best, is not realizable, because it uses an oracle to determine the best history size for each benchmark. However, profiling a dataset can achieve performance very close to an oracle. Not only that, but using the number of static branches in the program to decide on the depth of correlation used when running the program, can results in a very good approximation of the oracle [59]. Using this method doesn't require profiling but it does require some mechanism in the ISA to convey to the processor the BHR size decision made by the compiler.
The conclusions drawn here are consistent with results depicted in the original DHLF paper [21]. Graphing the size of the BHR as a function of time, it was shown that in the beginning of the program there was fluctuation in the size of the BHR, but that after this initial fluctuation, the BHR stabilized on a specific size for each benchmark. This suggests that there is no inherent advantage in changing the size of the BHR dynamically, but rather that the DHLF takes advantage of the fact that each benchmark as an ideal BHR size where the best prediction is achieved.

## 6.3 Paths Comparison

After showing that with known selection mechanisms the third-level path has no merit, and assuming that the performance gain by third-level structures demonstrated in previous studies is attributable to reducing aliasing, a comparison between the different paths is straight forward. It is out of the scope of this dissertation to conduct a thorough

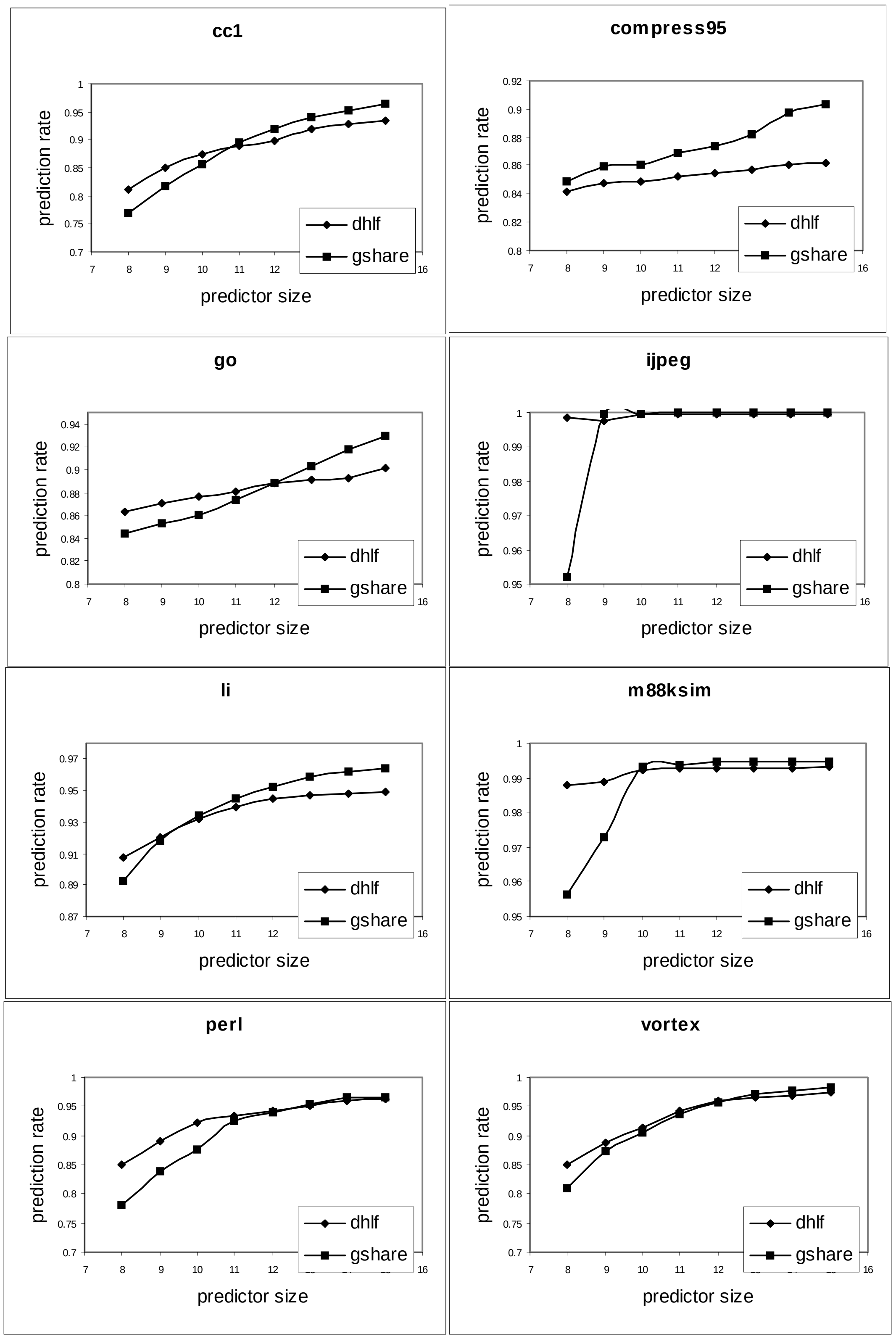


**Figure 6.3 a) comparing DHLF to gshare and bi-mode predictors for the SPECINT95 benchmarks**

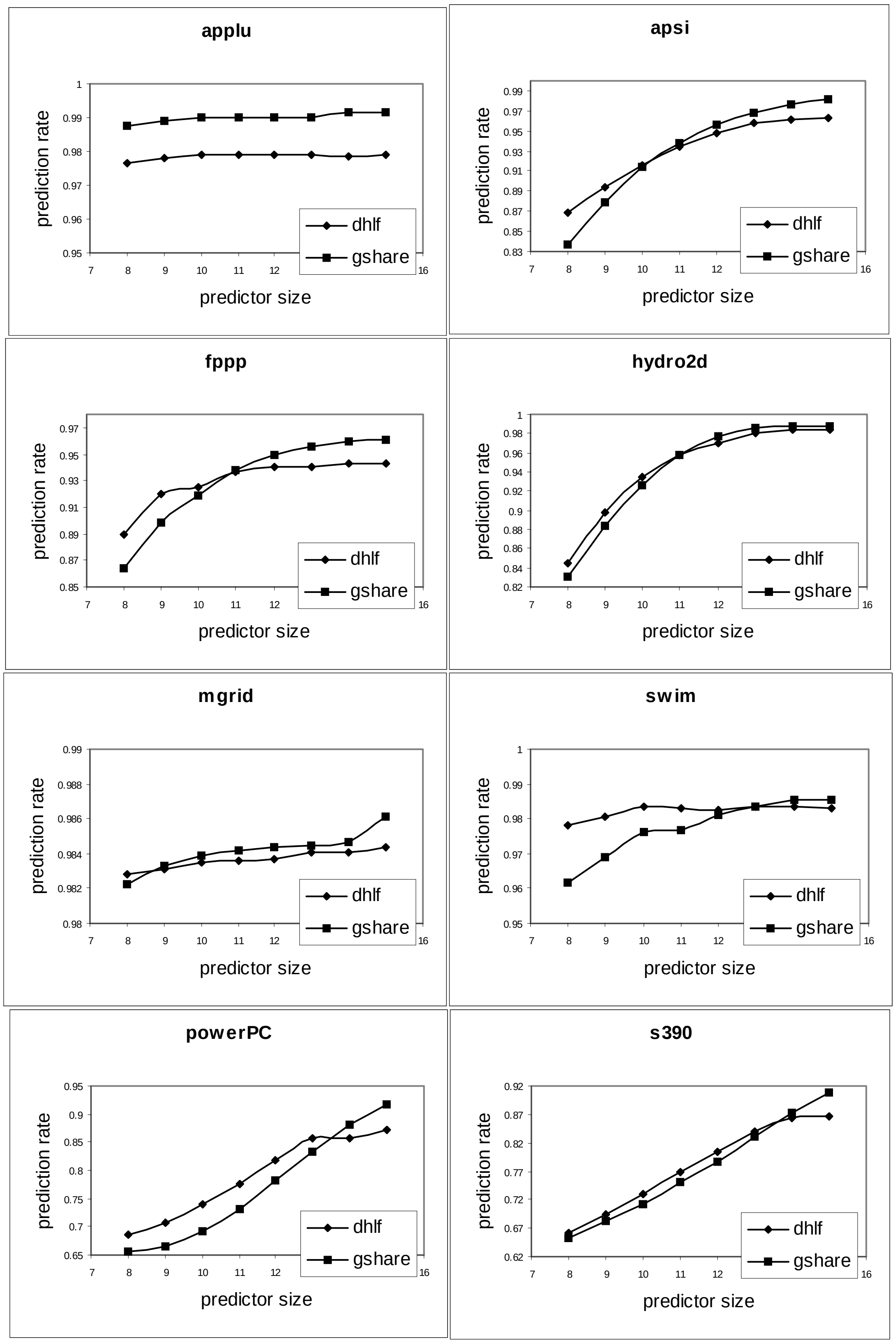
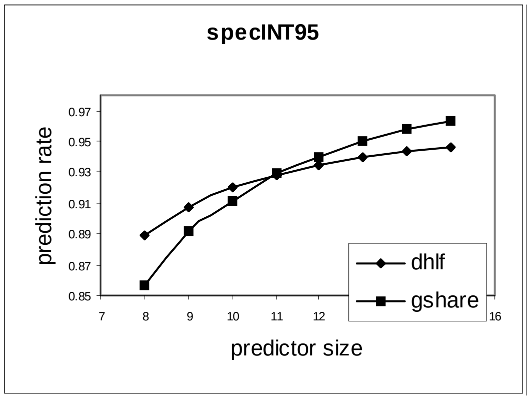


**Figure 6.3 b) comparing DHLF to gshare and bi-mode predictors for the SPECFP95 and IBM benchmarks**



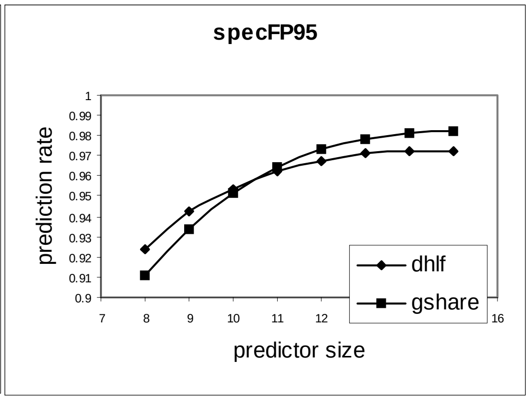


**Figure 6.3 c) comparing DHLF to gshare and bi-mode predictor for the SPECINT95 and SPECFP95 averages**

comparison between all known branch prediction structures that have been studied as part of different research paths. Instead, and because of the similarity of the classification structure to hybrid branch predictors, we have chosen the DHLF branch prediction structure as a point of comparison for other branch prediction structures.

Figure 6.3 compares the DHLF to *gshare* and the *bi-mode* branch prediction structures for varying size resources. The advantage of the DHLF for small predictor sizes is clear. This advantage is an indication that when resources are limited and aliasing degrades prediction, the DHLF improves prediction by reducing aliasing. However, this advantage is lost for predictors above 0.5KB for most benchmarks. For benchmarks with a large number of static branches the advantage continues even for a 2KB predictor over *gshare*. Contrary to what might be expected from two different branch prediction structures that implement the same branch prediction scheme, the DHLF and *gshare* do not converge as resources increase. Instead the *gshare* prediction surpasses the prediction achieved by the DHLF. The lack of convergence might be due to the DHLF filtering large correlation sizes out of the PHT in order to alleviate aliasing, but as resources increase, this filtering becomes unnecessary. A similar phenomenon is present in the *filter* branch prediction structure. It might be the case that for different sizes of PHT, the step size needs to vary in order to alleviate the degradation in performance. Alternatively, it is possible that changing the BHR causes thrashing in the PHT, which cause this loss in performance.

## 6.4 Summary

The conclusions presented in Chapter 5 led us to question the merit of third-level adaptivity as an orthogonal branch prediction research path to the aliasing path. This chapter confirms that assertion. Third-level adaptivity does not represent a better branch prediction scheme, but rather a better branch prediction structure under certain conditions. A better use of resources, specifically through filtering, leads to more accurate prediction for small branch prediction structures. Since some branches can be predicted with a small depth of correlation just as well as with a large depth of correlation, the small depth of correlation is chosen for filtering purpose. Consequently, it is important to compare third-level structures to aliasing structures and hybrid structures. When comparing the *bi-mode* predictor to the *DHLF* we observed that the *bi-mode* predictor achieves a better prediction accuracy.

A more efficient method of capturing the gain introduced by the DHLF was proposed. The method sets the depth of the history register for each benchmark, as opposed to having it constant as it is implemented in current processors, or dynamically changing it as proposed by third-level structures.  The drawback of this method is the need to implement a new ISA instruction to set the length of the history register.

It is important to emphasize that it was not proven that the third-level adaptivity's lack the potential to be a better branch prediction scheme, but rather the inability of current structures to take advantage of any merit that may exist.  We specifically showed that structures known to us do not improve prediction by exploiting the third-level adaptivity, but rather by drawing upon the underlying concept of reducing aliasing.

# Chapter 7 - The Do's and Don'ts of Branch Predictor Structures

3In this chapter we identify good criteria for building branch predictors. The criteria are deduced from three different sources. First, previous branch prediction structures in the literature are considered. This section serves only to summarize the highlights of the studies discussed previously in Section 2.2.2, and to address any different conclusions reached in light of studies done in this dissertation. The second source is the group of studies conducted for this thesis and presented in previous chapters. Those studies reveal an array of misconceptions regarding branch prediction practices, and should be considered in future predictors. Lastly, two micro-architectural trends introduced in previous studies that profoundly impact future branch predictors are considered. We choose to start from the later.

## 7.1 Micro-architectural Trends

### 7.1.1 Wire Delay

Several recent studies have shown that in the near future, wire delay will need to be considered in the design of future processors [63][64][65]. It was noted that for the most part wire quality does not degrade and the number of reachable transistors in a fixed cycle will stay constant. The conclusion of these early studies was that, in fact, there is no wire problem. However, this conclusion ignored the exponentially increasing number of transistors inside a chip.

It was observed that technology has reached a point where the distance a signal can travel in one cycle becomes smaller than the width of a chip [63]. The distance a signal can travel in one cycle compared to the width of a chip has been decreasing rapidly for a long time. However, such a fact was of little consequence because this distance was always larger than the width of one chip. This is changing in current technologies and will continue to deteriorate in the future. This has several immediate implications. First, global communication between on-chip modules will take longer than one cycle, and the number of transistors reachable in one cycle will stop increasing. The ever-increasing disparity between wire and gate delays will cause microarchitects, who have never before needed to concern themselves with wire latency, to attend to this matter [63].

Building on these observations, a scaling experiment was done on two different architecture types [64]: an architecture that aims for fast clock cycle, such as Compaq's Alpha; and an architecture type that aims for large IPC, such as HP's PA-RISC. It was shown that due to the wire technology's inability to scale, microarchitects will soon face the unattractive tradeoff between slowing down the clock cycle and smaller IPC. As a result, both of those architecture types will only be able to sustain performance improvements of 12.5% annually, a far cry from the annual rate we got accustomed to of 50-60%. The reason is that as feature size shrinks, and wires become slower compared to gates, the amount of state reached in a cycle decreases.

This new observation has been, for the most part, ignored in the branch prediction research community. Elaborate structures with sizes of up to 64KB have been proposed [49]. The assumption that in the future more transistors will be available to the branch prediction module has given the illusion that aliasing will cease to degrade prediction accuracy in future chip generations. It was shown, however, that in 35nm technology, expected by the year 2012, it might be that only PHT of sizes between 512 entries and 4K entries will be accessible in one cycle [64]. For these modest sizes, even benchmarks from SPECFP95, which are traditionally easy to predict and do not suffer much from aliasing, suffer significantly in performance. For example, a 512 entry PHT achieves less than 96% prediction accuracy compared to 99% prediction accuracy for an aliasing free scheme for the hydro2 benchmark (see Figure 4.4). Notice that the aliasing free ideal accuracy is almost achieved by a PHT of 2K entries. The hydro2 is usually not included in branch prediction studies since it is easy to predict and usually does not suffer from performance degradation due to resource constraints. We conclude that even benchmarks that are currently not considered to run slower due to poor branch prediction, will suffer performance degradation in the future due to aliasing.

Corporations rarely reveal complete details of the branch prediction structure used in commercial chips. It is even more rare to find a window into the decision-making process foregone an actual implementation of a branch predictor. It is, therefore, hard to evaluate whether the disparity between wire and gate delays has shown itself to be a problem in present chip designs. One example of an exception to corporate secrecy is the G4 PowerPC microprocessor. A 2K entry *gshare* branch predictor was evidently considered but eventually replaced by a 2K entry bimodal predictor. The reason behind the switch was to remove the XOR gate in front of the predictor, because it was in the critical path [58]. Notice that a 2K entry predictor is a very small predictor but was still in the critical path. As we have seen, it has been suggested that the growing disparity between gate and wire delays must be taken into consideration by microarchitects in their design of new chips. One study to take this advice in the branch prediction field discussed the impact of delay on the design of branch predictors [65]. This study showed that trading prediction gains, which come with increasing the predictor size, with increasing delay is never a good idea. In other words, the pipeline should never be halted because a branch instruction is awaiting a decision from the branch predictor even if the prediction will be more accurate than a prediction produced in one cycle. This is somewhat intuitive, since halting the pipeline just to get a prediction defeats the purpose of having a predictor to begin with. After highlighting this impractical tradeoff, alternatives to improve prediction without increasing the size of the predictor were investigated [65]. Observing that 57% of dynamic branches have more than one cycle to be predicted, a cascading look-ahead predictor was suggested. The cascading predictor uses a small predictor for dynamic branches that need to be predicted in one cycle, while using a larger predictor to predict branches that have more than one cycle to be predicted. The cascading branch predictor was able to alleviate the degradation of IPC compared to a *gshare* predictor, but it was not able to compensate for it completely. The same study learned an already used approach of an overriding predictor. The overriding predictor allows the larger structure to override the prediction made by the smaller structure for a small misprediction penalty. The overriding approach showed to outperform the cascading predictor [65].

### 7.1.2 Software Development

As computers become faster and are able to process more information in less time, software developers take advantage of the newly acquired processing power to develop ever-more demanding software. This is the main reason why most computers become obsolete after a few years. The size of programs has been shown to increase constantly and consequently, the I-cache performance degrades [2]. It is easy to believe that as code bloats, the number of static branches in the program increase.

The adverse effects of a larger number of static branches in a program should be obvious by now and is summarized in Figure 4.6. It is peculiar, therefore, that studies which consider future structures by assuming hardware real estate that will only be available in the future, are conducted with current and past software[4].

The difficulty of predicting what software will be available in the future is obvious. However one remedy for this difficulty is to choose a set of benchmarks that is especially heavy with branches as a way of anticipating the increased number of static branches will be common in future software[5]. With that in mind, the reader is encouraged to consider benchmarks that are heavy with static branches like go, gcc, PowerPC, and S390, to better represent future software than other benchmarks.

## 7.2 Observing Past Work

The background presented in Section 2.2 highlights the advantages and disadvantages of current branch prediction structures that aim to reduce aliasing. Those methods are able to reduce aliasing as a result of following good practices and avoiding the following bad practices.

[4] See chapter 3 for why this dissertation is using the SPEC95 and not the SPEC2000 benchmark suite.

[5] While not the first ones to note this, we suspect that the low number of branches present in some of the SPEC benchmarks does not represent current software, let alone future software that will become available.

### 7.2.1 Good Practices

A few good practices for building a structure to reduce aliasing can be extracted from the observation of past work that has been done on the subject. The first and foremost of these practices is reducing negative aliasing. The *agree*, and *bi-mode* predictors do so by splitting the PHT into two. One PHT serves the branches that are mostly taken, and the other one serves those that are mostly not taken. By splitting the branch streams into branches that are biased to be taken and branches that are biased not to be taken, negative aliasing is significantly reduced, and prediction is improved. The classical solution to reduce aliasing is the introduction of tags into the PHT. This was not found to be cost effective. The only structure that is close to utilizing associativity is the *skew* predictor. The *skew* predictor achieves pseudo-associativity by means of redundancy. Finally by filtering easy-to-predict branches out of the PHT, the *filter* mechanism was able to reduce aliasing in the PHT, while retaining good prediction for the filtered branches.

### 7.2.2 Bad Practices

In the past, work done on reducing aliasing in branch prediction has resulted in several pitfalls. The first of these is that all branch prediction structures that reduce aliasing have demonstrated redundancy associated with the structure. For example, in the *bi-mode* structure, the redundancy is in the form of the choice PHT. The *skew* predictor has the same information stored in two or three different places, and so on. Branch prediction structures that reduce negative aliasing neglect to address the other kind of aliasing—that is, aliasing between instances that do agree and do not agree with the branch's bias. This negligence is present in the *agree* and the *bi-mode* predictors. When implementing the scheme in a branch prediction structure, it is essential to retain all the information that helps the branch prediction scheme achieve its peak performance. The *filter* mechanism filters easy-to-predict branches out of the PHT, but at the same time loses the special instances of those easy-to-predict branches that do not comply with the bias, such as loop exist branch instance. Consequently, even with a large amount of hardware dedicated to the filtering mechanism, it will never reach the peak performance of the global scheme it supposed to implement.

## 7.3 Studies Done in This Dissertation

### 7.3.1 The Omniscient Dynamic Selection Mechanism

Through Chapter 5's discussion of hybrid predictors, the wonders of the dynamic selection mechanism were revealed when the need to balance between two unrelated information arises. The dynamic selection mechanism was proven to be instrumental in reducing aliasing. Moreover, its ability to reduce aliasing by mapping branches that are predicted better by component A into component B, the dynamic selection mechanism was able to map the 'hybrid branches' to their respective best component.

It is tempting to conclude that the dynamic selection mechanism in the *bi*-mode predictor explains why it performs better than the *agree* predictor that utilize a static selection mechanism. However, this will be shown to be a false conclusion. The difference in performance results from the *agree* predictor attaching the bias to the BTB. As the size of the *agree* predictor grows, the number of entries dedicated to the bias cannot grow since they are attached to the BTB. In other words, the *agree* predictor throttles the resources dedicated to the bias, while the *bi-mode* predictor does not.[6]

The value of the dynamic selection mechanism may be seen when either a bias bit is not present in the ISA and a dynamic selection mechanism therefore becomes imperative, or when there is large disparity of branch behavior between different datasets. Such disparity is more prevalent when choosing between two different schemes, as in the McFarling predictor, than when selecting the bias of a branch, as in the *bi-mode* predictor.

### 7.3.2 Compile Time Information

Throughout the history of the branch prediction field, it seems as if a dynamic approach to data collection and decision making during run time has always prevailed over the static approach, where the decision making is done during compile time. Dynamic branch predictors are more accurate than static branch predictors. The *bi-mode* predictor outperforms the agree predictor because it was thought to select the bias of the branch dynamically. And dynamic selection mechanisms perform better than static selection mechanisms because the best predictor might change throughout the program execution.[7]

[6] The bias/selection throttling effect has been ignored in multiple studies. It degraded the performance of the *bi-mode* and YAGS predictors when compared against a branch predictor that utilized value prediction [66]. We speculate here that it likely caused a similar degradation to the McFarling predictor when compared to the multi-hybrid predictor [49].

[7] For the later two examples, we indicate what was thought before this dissertation showed it not to be the true.

In the rush to dynamically determine more and more information, it has been forgotten that some information that is very difficult to determine dynamically during run time can be very easily obtained statically during compile time.  In such cases, compile time optimization can be very useful.  The most obvious example of this in the branch prediction field is the number of static branches present in a program.
As we have seen in Chapter 5, the number of static branches present in a program has a significant effect on the amount of aliasing present in the PHT.  We believe this information is crucial to tune the size of the BHR for the best tradeoff between aliasing and correlation.

### 7.3.3 The Dependence of Correlation on Structure Size

Chapter 4 highlighted the dependence between correlation and aliasing.  It is clear now that deeper correlation entails better prediction.  It is also clear that deeper correlation entails more aliasing.  We observed that every program, depending on the amount of aliasing it experiences, has a unique sweet spot of correlation where it achieves the best tradeoff between higher prediction due to correlation and lower prediction due to aliasing.  We also observed that some programs, like li and compress, experience very little aliasing, and therefore can achieve their peak prediction performance by utilizing as much correlation as possible.
The problem is that the amount of correlation is tightly bound to the size of the predictor. Take, for example, a *gshare* structure of 1K entries (0.25KB).  The most correlation one can achieve is 10 deep.  As we mentioned before, some programs are best predicted using less than 10 bits of correlation, but this does not accommodate programs that can utilize

more than 10 bits of correlation.  Those are forced to use only the 10 bits a 1K entry *gshare* can accommodate.

The problem is accentuated in structures that attempt to reduce aliasing like the *skew* and *bi-mode* predictors.  The closest *bi-mode* structure in size to a 1KB entry *gshare* is a 0.75KB entry *bi-mode*, out of which 0.25KB entries are dedicated to each choice PHT. As a result, the most correlation that can be exploited in such a structure is a history of 8 bits deep.

It is therefore beneficial to alleviate, if not eliminate, the correlation depth dependence on the structure size.  Such de-coupling will enable the pursuit of better predictors without placing limitations on the size of the structure.

# Chapter 8 - Solutions

This chapter introduces YAGS - a new branch prediction structure that capitalizes on the criteria and observations introduced in the previous chapters. The chapter goes on to identify aliasing in the first-level table as the primary impediment of prediction accuracy. Lastly the chapter proposes a static, profile based, choice PHT to reduce aliasing in the first-level table. The profile-based choice PHT is presented in the context of YAGS, but can be incorporated into other predictors as well. The profile based choice PHT not only reduces aliasing in the first-level structure, but also facilitates transforming YAGS into a cascading predictor.

## 8.1 Yet Another Global Structure (YAGS)

In this section we introduce YAGS, a branch prediction structure that implements the global scheme and that is designed to reduce aliasing. First, we introduce the motivation behind this predictor and explain the need for a new branch prediction structure after so many have been proposed already. Then, we introduce YAGS, compare it to previous branch prediction structures, explain its advantages, and explore its design space.

### 8.1.1 Motivation

So far it has been observed that the 'third-level of adaptivity path' and 'hybrid path' are really just techniques that reduce aliasing. And therefore all branch prediction structures that implement those paths should be compared to each other and not treated as a separate path of research in the branch prediction field. Elaborate and large branch prediction structures have caused the illusion that aliasing ceases to degrade performance in branch prediction structures. Not only does aliasing still degrade performance, but it will only get worse in future chip technologies. As the disparity between wire and transistors increases, and the cycle time shrinks, predictors will have to decrease in size in order to be able to produce a prediction in one cycle. Smaller tables entail a larger number of aliasing. This problem is exacerbated by the growing code size of applications and might cause aliasing to significantly degrade performance even for simple predictors like the *bimodal*.

Despite the numerous branch predictors devised to alleviate the negative effects of aliasing on prediction accuracy, the aliasing problem is not completely solved (Section 4.4). The micro-architectural trends discussed in Section 7.1 are bound to aggravate the aliasing problem even further in the future. It is clearly necessary to try and further reduce aliasing by devising future structures that carefully consider such issues as size and delay.

### 8.1.2 YAGS

YAGS is based on the observation that some redundant information is stored in the PHT. For a predictor to be able to reach the full potential of the global branch prediction scheme, it is enough to store the branch's bias and those branch instances that do not comply with the bias. Traditionally, branch predictors have stored all branch instances. Figure 8.1 depicts a diagram of the YAGS branch predictor. We chose the *bimodal* structure to store the branches' bias and termed it, as in the *bi-mode* predictor, the choice PHT. Two direction caches were then added to store the instances of the branch that do not comply with the bias. Each entry in the direction caches contains an address tag, a history tag, and 2bc state machine. The address tag is needed to distinguish the entry as belonging to a specific branch. Notice that unlike to traditional caches, the address tag is taken from the LSBs of the branch address. The reason for this is that the cache is indexed by the branch address xored with the history register, which leads to a loss of the address information. Due to the nature of the xor function, aliasing may occur between two consecutive branches. Using the branch address as the tag will allow aliasing to occur only in the case of some spatial locality.

The reason for having two direction caches rather than a consolidated one is the aliasing-reducing property of the choice PHT, discussed in Chapter 5. Having two direction PHTs allows the choice PHT to move branch instances between them if one of the direction PHTs is overcrowded with information.

Experiments have shown that an address tag of 6 to 10 bits will suffice. Too small of an address tag might cause some branch instances to be associated with one branch, in practice they belonging to another branch. In most cases, 6 to 10 bits of address tags are enough to identify the branch. Extending the address tag to the size of the word is not cost effective. In contrast to traditional caches, this risk can be taken since the information stored in those caches is branch prediction. The worst that can happen is a wrong prediction; in traditional caches erroneous execution can result.

When comparing YAGS to the *bi-mode* branch predictor, two distinctions become apparent. First, the choice PHT is used not only for saving the branch bias but also for predicting the branch when no special instance of the branch is identified in the direction caches. Second, since the direction caches only store the special instances of the branch

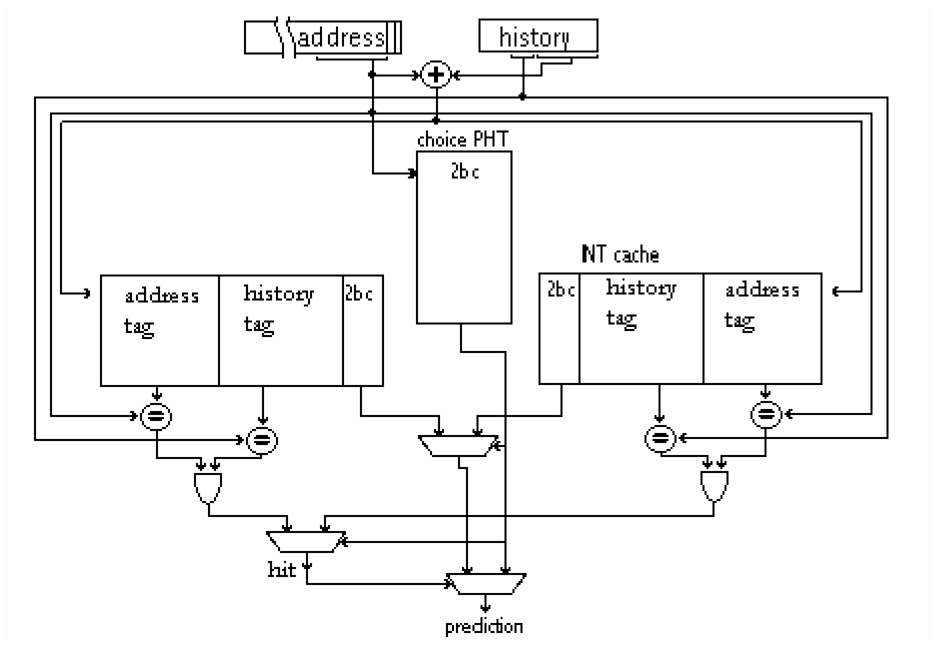


**Figure 8.1 – Diagram for the YAGS predictor**

which do not comply with the bias, the direction caches can be much smaller than the direction PHTs in terms of entries. The problem with having a smaller number of entries in the direction caches as compared to the choice PHTs is that less history is used to index the structure and as a result, less correlation is taken advantage of. On the other hand, utilizing less history means that less information is stored in the direction caches, which leads to less aliasing.

The addition of the history tag solves this problem, but even more importantly, the history tags decouple the correlation depth used by the predictor from the size of the predictor. While in previously proposed predictors the depth of correlation is bound by the size of the PHT, the introduction of the history tag in the direction caches almost completely decouples the depth of correlation from the size of the direction caches in YAGS. Obviously, the same method can be used in other predictors as well, but the size of the correlation is more tightly coupled to the size of the predictor in other branch prediction structures. Table 8.1 shows the size increase for different predictors if a history tag of increasing correlation is added to them.

| **# of bits** | ***gshare*** | ***Skew*** | ***bi-mode*** | **filter** | **agree** | **YAGS** |
|---|---|---|---|---|---|---|
| **2** | **100%** | **100%** | **66.6%** | **<100%** | **<100%** | **22%** |
| **3** | **150%** | **150%** | **100%** | **<150%** | **<150%** | **33%** |
| **4** | **200%** | **200%** | **133.3%** | **<200%** | **<200%** | **44%** |
| **5** | **250%** | **250%** | **166.6%** | **<250%** | **<250%** | **55%** |

**Table 8.1. – The relationship between correlation depth and predictor size increment**

The configuration of YAGS presented in Table 8.1 includes a 6 bits address tag and direction caches, which have a quarter the number of entries that the choice PHT has. While adding 5 bits of history tags to YAGS requires an addition of 55% in size, adding it to most other predictors would require an addition of 250% to resources. This is another of the advantages of YAGS.

When a branch shows up in the instruction stream, the choice PHT is accessed. If the choice PHT indicates “taken,” the “not taken” direction cache is accessed to check if it is a special case where the prediction does not comply with the bias. If a miss occurs in the “not taken” direction cache, the choice PHT is used for the prediction. The same happens if the choice PHT indicates “not taken” but this time the check is done in the “taken” cache. The choice PHT is addressed and updated as in the *bi-mode* choice PHT. The “not taken” cache is updated if a prediction from it was used. Further more, an entry is added to the “not taken” cache if the choice PHT indicates “taken” and the branch outcome is “not taken.” The same happens with the “taken” cache.

The classic way to reduce aliasing is to make the cache set associative. Until now, the introduction of tags was not a feasible solution. YAGS makes the introduction of tags cost effective. When making the caches set-associative, there is some extra cost for keeping a correct replacement policy. For example, in a two-way set-associative cache, one bit for every two entries will suffice to keep track of which entry was replaced last. We suggest a Least Recently Used (LRU) replacement policy with one exception: an entry in the “taken” cache which indicates “not taken” will be replaced first to avoid redundant information. The reasoning behind this is that if an entry in the “taken” cache is set to “not taken,” this information is already in the choice PHT and therefore is redundant and can be replaced.

Making the direction caches set associative pays off for selected benchmarks, but overall, is not cost effective. Reducing the amount of information stored in the direction caches reduces aliasing in the direction caches to the point that does not contribute much to performance degradation and does not justify the extra bit associated with a 2-way set associative and the added latency. Making the direction caches set-associative might help to reduce aliasing for future programs that have more static branches. The set-associative option was not investigated any further in this thesis.

Notice that when increasing the size of the direction caches, the size of the history register can be increased to better exploit correlation between branches. However, when making the direction caches two-way set-associative, one less bit is used to index them than if the direction caches were direct-mapped. This loss of correlation has a negative effect on prediction accuracy. In order to maintain the same level of correlation when comparing two-way set-associative caches against a direct-map caches YAGS of the same size, one bit of history must be added to the history tag.

#### 8.1.2.1 Prediction Accuracy

YAGS is compared against the *gshare* and *bi-mode* predictors. The *gshare* is the base predictor and is shown for reference. The *bi-mode* predictor, on the other hand, has been established as the best predictor in the ‘aliasing’ path of research, and has been compared many times to the *agree*, *filter*, and *skew* predictors. In Chapter 5, we compared the *bi-mode* to the McFarling predictor and established that the *bi-mode* predictor is more cost effective. The McFarling predictor has been established as one of the better predictors in the ‘hybrid’ path of research.

Figure 8.2 shows the prediction accuracy for *gshare*, the *bi-mode* predictor and YAGS. As can be seen, YAGS outperforms all other structures tested. As the size of the PHT increases, YAGS’ advantage over the other schemes decreases. This is expected since all structures examined implement the global branch prediction scheme. As the size of those structures increases, the aliasing problem in the PHT decreases, and therefore the performance of all predictors converges.

One of the pitfalls of the SPEC95 benchmark suite is that most traces have a very small static branch signature [12]. For example, the compress benchmark has only 495 static branches. Those branches are executed over and over again throughout the course of the program. Because of this small static branch signature, each branch is more likely to have a unique entry in the PHT for each history instance. A small static branch

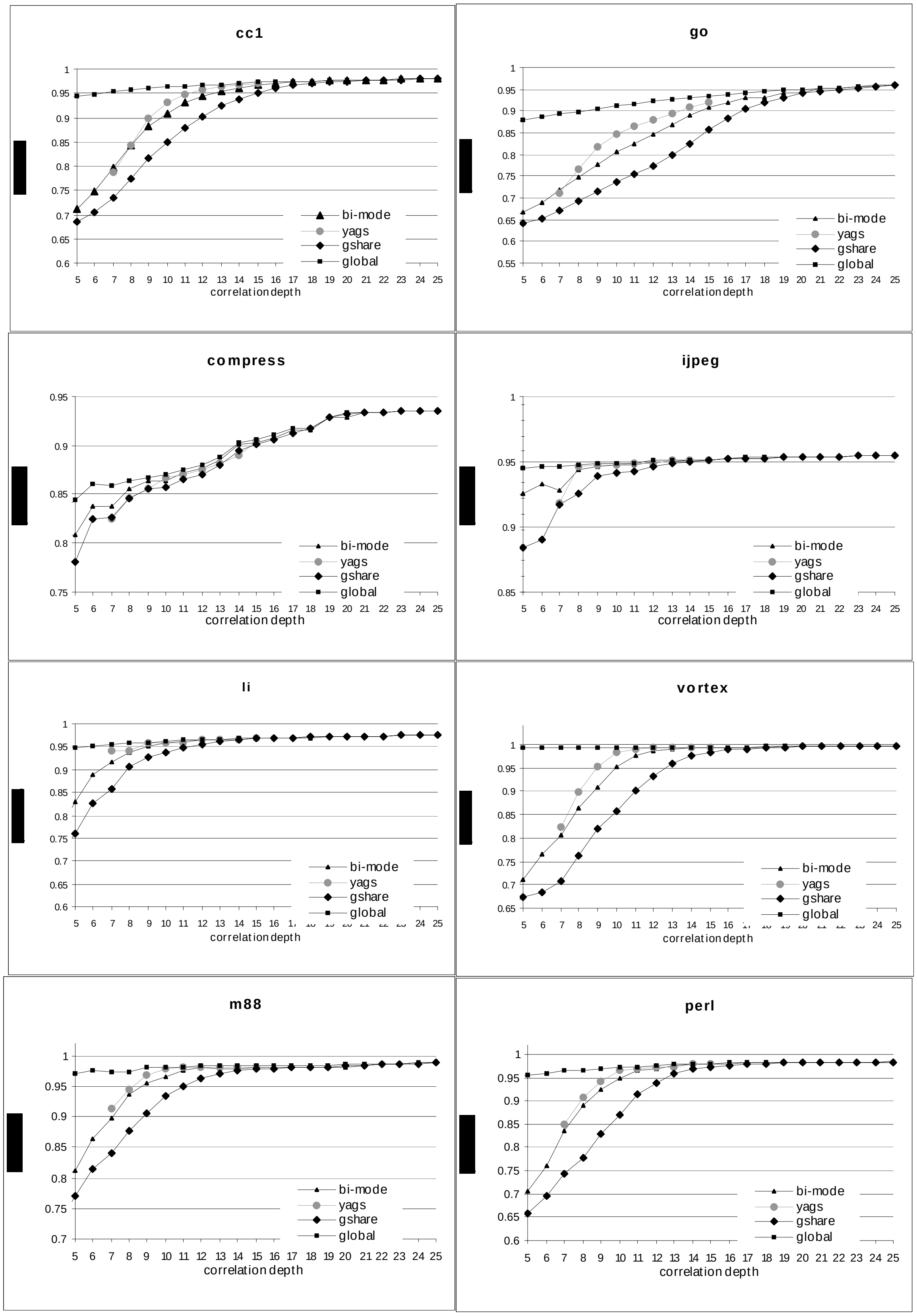


Figure 8.2 a) comparing YAGS for the SPECINT95 benchmarks

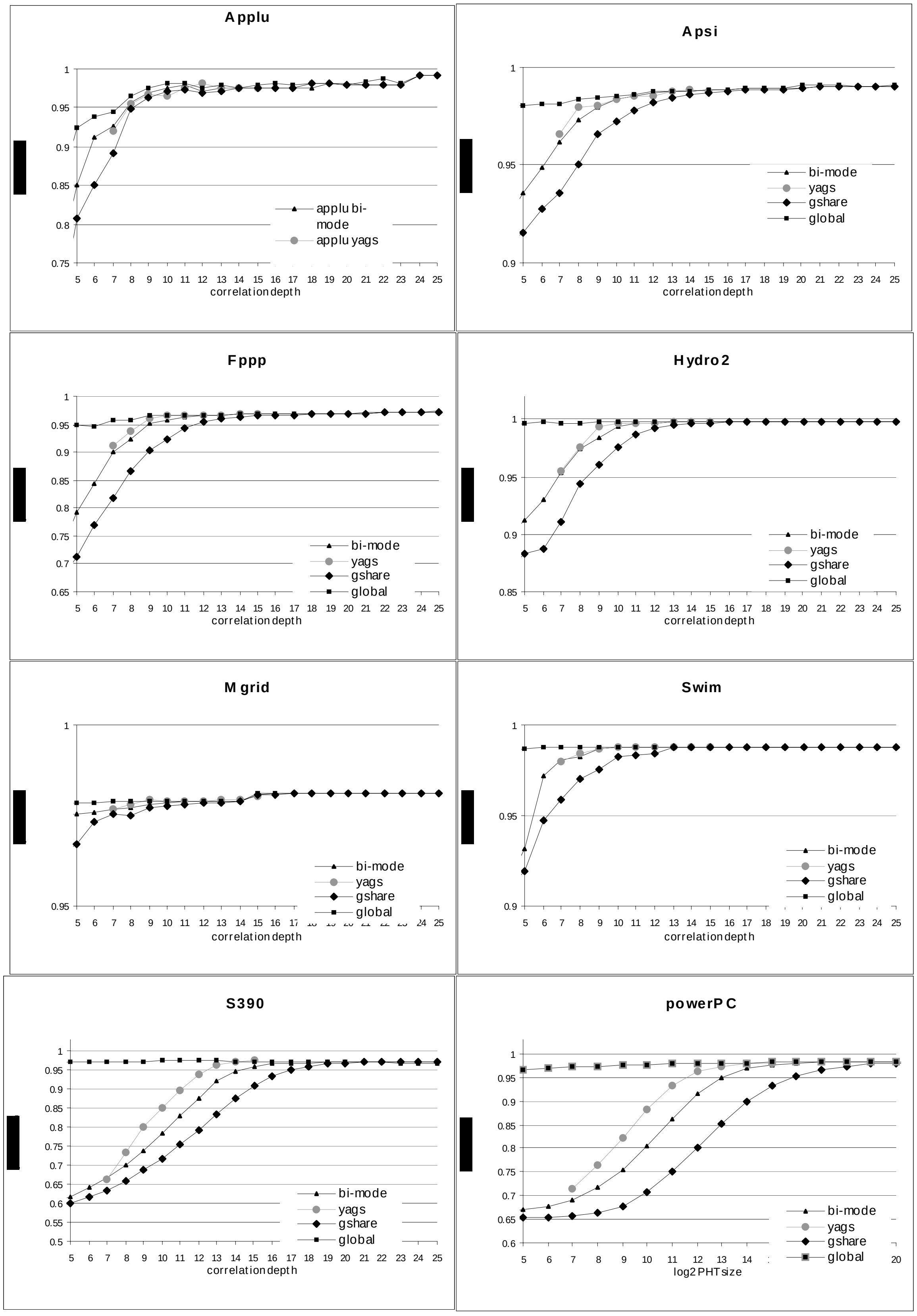


**Figure 8.2 b) comparing YAGS for the SPECFP95 and IBM benchmarks**

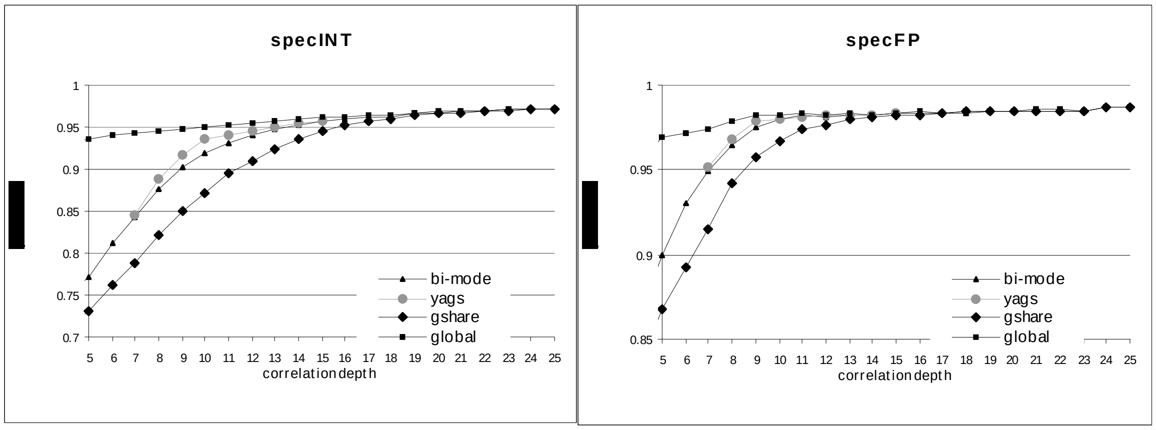


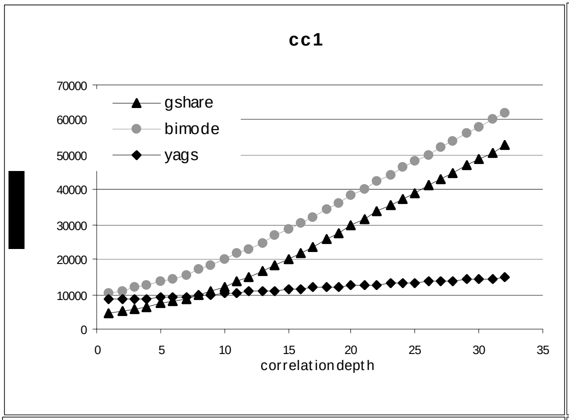
**Figure 8.2 c) comparing YAGS for the SPECINT95 and SPECFP95 averages**

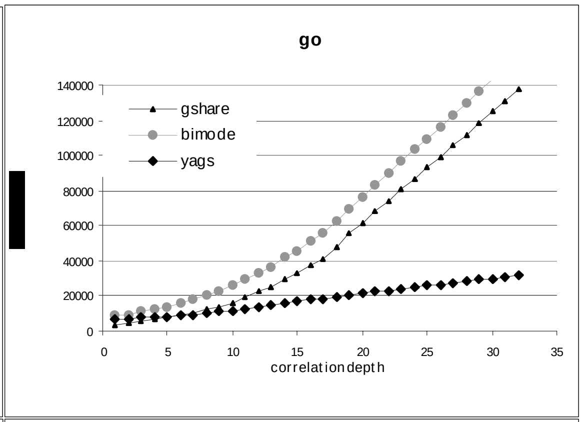
signature results in a very small amount of aliasing in the PHT, and therefore boosts the performance of the branch prediction structure.

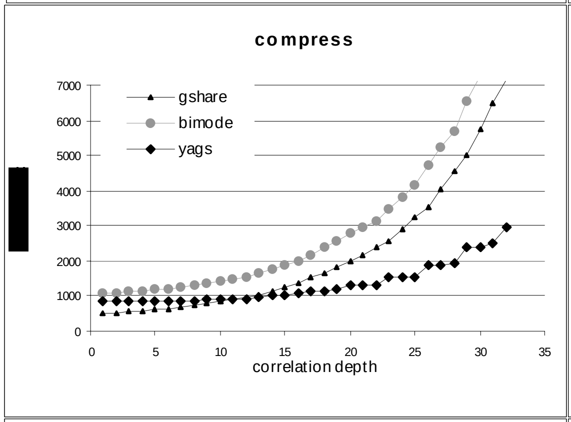
The gcc, go, and the IBM benchmarks are thus of special interest because of their large static branch signatures. As can be seen in Figure 8.2, YAGS outperforms the other structures for the go, gcc and IBM benchmarks. The benchmarks suffers considerably from destructive aliasing. The *gshare* scheme for small predictors achieves a 71.7% correct prediction accuracy while the aliasing-free potential of the global scheme is 97.4% correct prediction accuracy in the case of the S390 benchmark. For about the same amount of resources that allows *gshare* to achieve a 71.7% accuracy, *YAGS* achieves 85%. The *bi-mode,* which is designed to reduce destructive aliasing, achieves only a 78.5% accuracy.

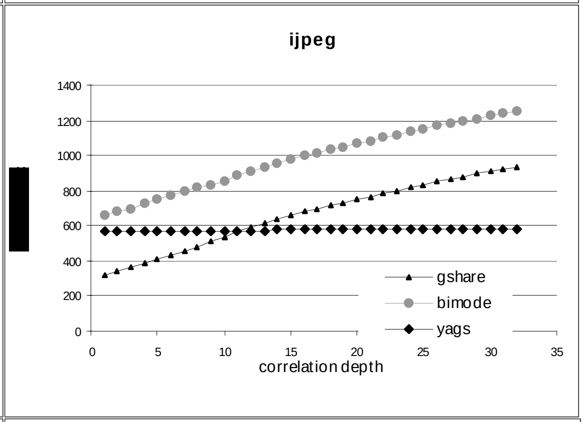
### 8.1.2.2 Amount of Information Stored in Prediction Structures

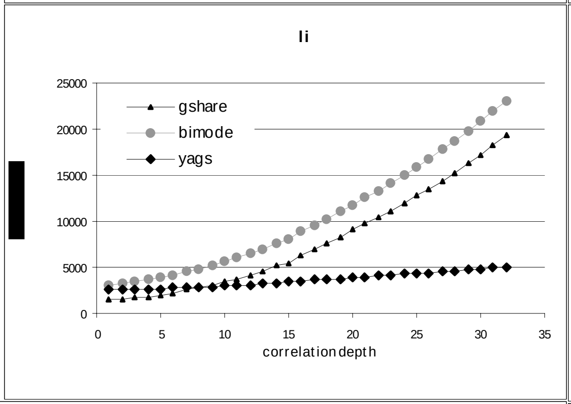
The main advantage of YAGS over other predictors is its ability to store less information without compromising the potential prediction accuracy of the global scheme, as is done by filter mechanisms. To check on that premise, we tapped the wires

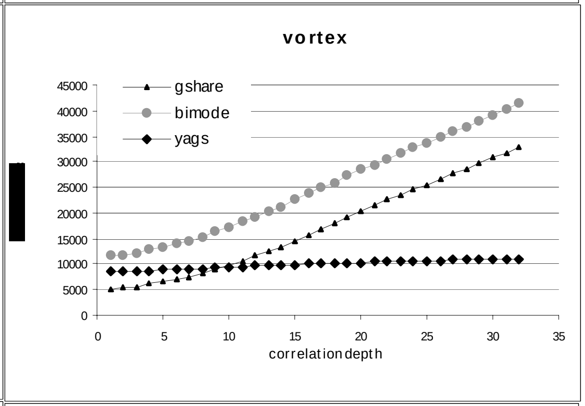


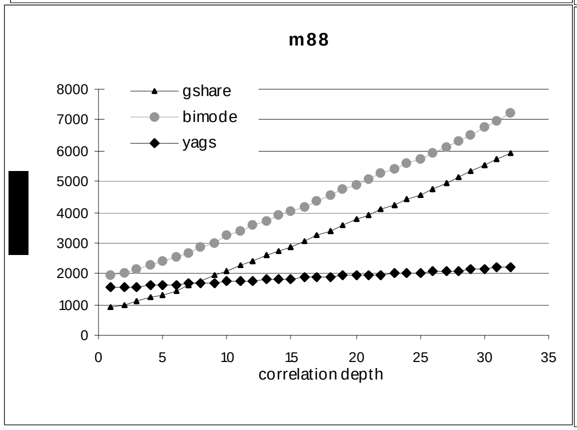












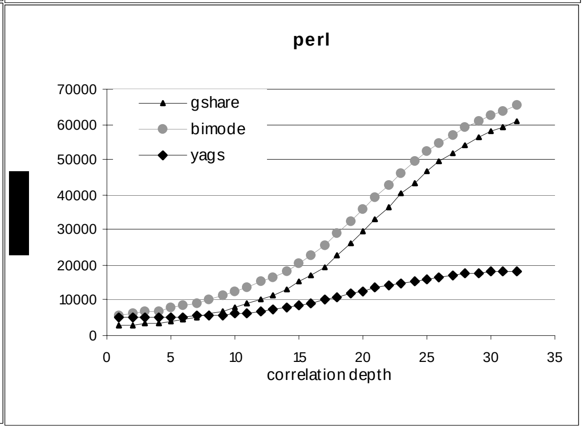


**Figure 8.3 a) Amount of information stored for the SPECINT95 benchmarks**

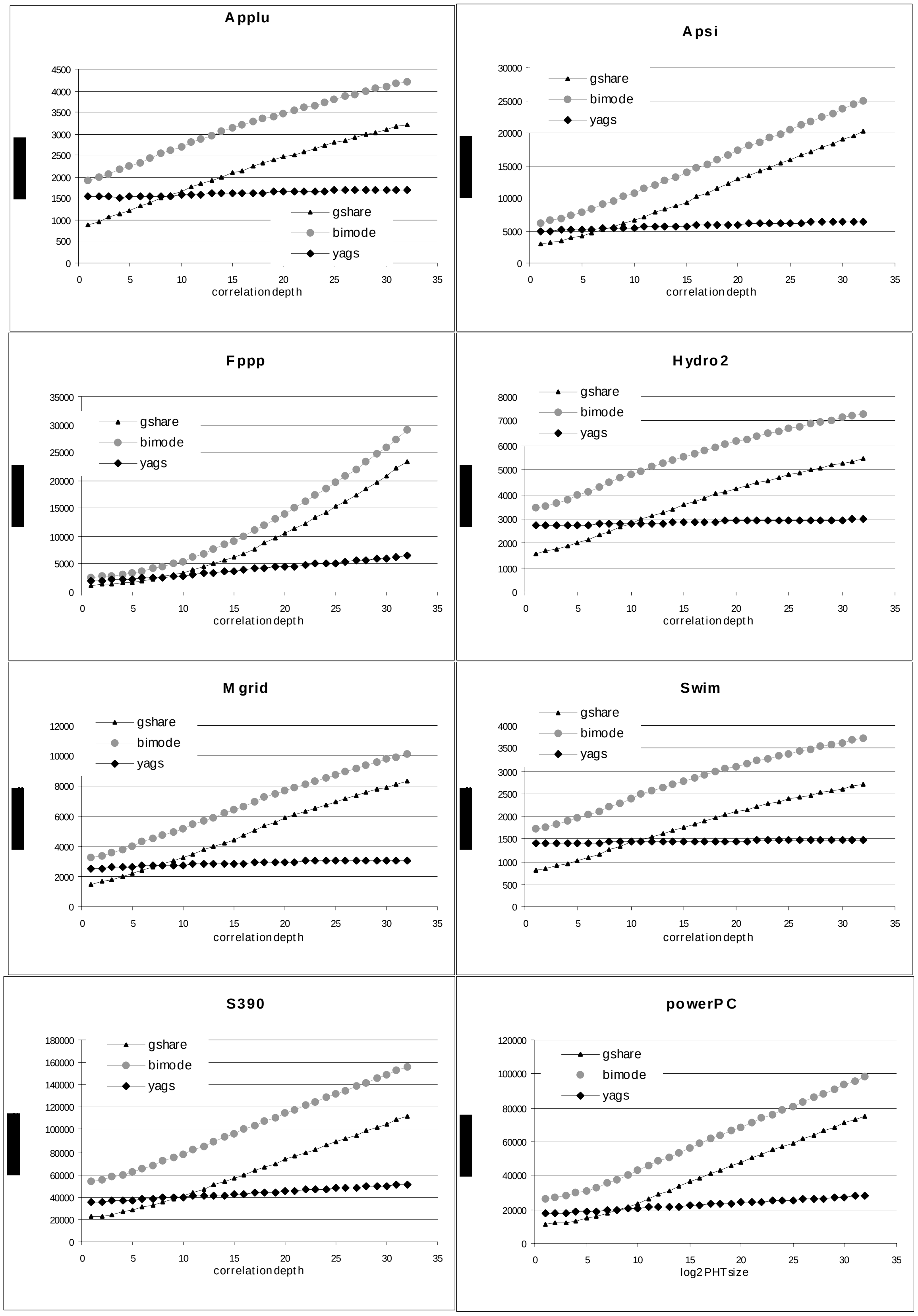


**Figure 8.3 b) Amount of information stored for the SPECFP95 and IBM benchmarks**

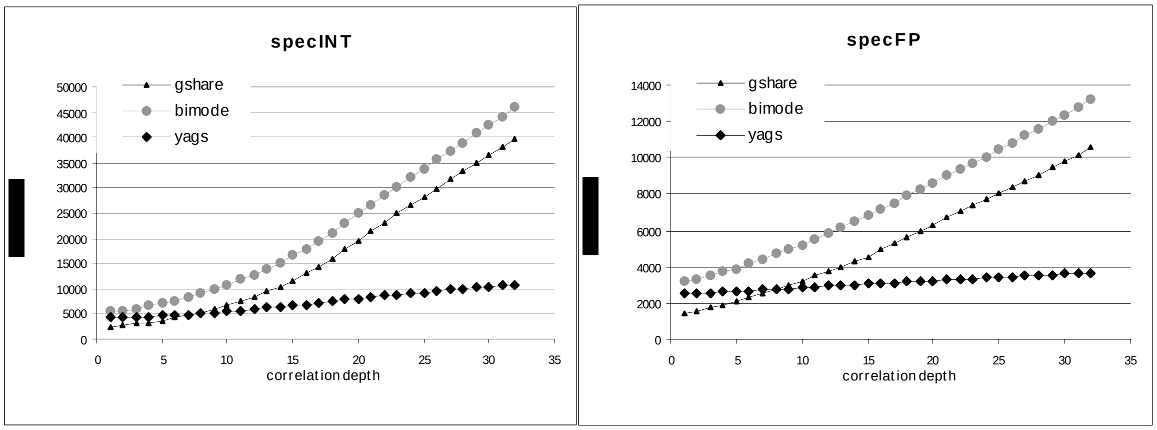


**Figure 8.3 c) Amount of information stored for the SPECINT95 and SPECFP95 averages**

used to index the different structures in the *bi-mode*, *gshare*, and YAGS predictors for distinct information stored in those structures throughout the program execution. For every piece of information stored, we checked whether it was duplicated. If the information tapped was new, we incremented the amount of information stored in the predictors. For *gshare*, the line accessing the PHT was tapped. For the *bi-mode* and YAGS, we tapped both the choice PHT and the direction PHTs or caches, respectively. Figure 8.3 shows the amount of information stored in the different predictors. YAGS consistently stores less information than the *bi-mode* and *gshare* predictors. As the depth of correlation increases in the predictors, the amount of information stored in YAGS increases very moderately compared to the amount of information stored in the *bi-mode* and *gshare* predictors. As Figure 4.5 demonstrates, YAGS main strength is that less information translates directly to less aliasing and better prediction accuracy.

It is well known that the *bi-mode* predictor is more accurate in predicting branches than the *gshare* predictor. Though this may be true, Figure 8.3 shows that the *bi-mode* predictor stored more information that the *gshare* predictor. The information gap between the two predictors does not change considerably across different correlation depths. This gap is due to the fact that the choice PHT in the *bi-mode* predictor stores the bias of the different branches on top of the correlated information stored in the PHTs. The *bi-mode* predictor achieves better prediction accuracy than the *gshare* predictor, not by reducing aliasing, but by reducing destructive aliasing only. The extra information the *bi-mode* stores on top of what the *gshare* predictor stores indicates that the *bi-mode* predictor suffers more aliasing. If we were to check for destructive aliasing, we would expect the *bi-mode* predictor to have less than the *gshare* predictor. Since YAGS has the same mechanism to reduce destructive aliasing as the *bi-mode* predictor has, this comparison holds.

#### 8.1.2.3 Testing Under Context Switching

Throughout consecutive generations of microprocessors, the amount of hardware used for the branch prediction structures has grown. Ideally, the prediction accuracy should be proportional to the amount of hardware invested in the structure. One drawback of increasing the hardware size is the time it takes the branch predictor to reach its peak performance, otherwise known as a cold start. In the presence of intensive context switching, the warm-up time of the branch prediction scheme might have a significant influence on the misprediction rate. Furthermore, due to long warm-up times, some complex structures might end up achieving less accurate predictions than less sophisticated structures. It has been shown that a hybrid predictor, composed of *gshare* and the *bimodal*, has good performance in the presence of a context switch [10]. This is due to a short warm-up time of the *bimodal* component. Each branch is mapped to only one entry in the PHT of the *bimodal* structure. Therefore, it takes only a few executions of a branch for its respective entry to reflect the information stored the branch. On the other hand, the *ghsare* structure has to execute a branch several times for each history instance before it warms up. The potentially large number of history instances, given by $2^{\text{history}}$ length, will result in a very long warm-up time and that, in return, will cause the degradation of performance in the presence of context switches. Other predictors, such as the *skew* predictors, suffer from the same problem.

On the other hand, one would expect the *bi-mode* predictor and YAGS to be more tolerant of context switches. Most of the information in the “not taken” direction PHT of the *bi-mode* predictor is “not taken”. So once the choice PHT points to the “not taken” direction PHT, the probability of a “taken” prediction is very small. Thus, only few executions of each branch are needed to warm up the choice PHT, which is essentially the *bimodal* predictor. After that, it will take more executions to warm up the branch’s history instances, which do not comply with the branch bias. But for the most part, the predictor should not perform worse than the *bimodal* predictor. The same phenomenon occurs in YAGS. This time the short warm-up time is due to the address tags. There is a low probability that the tags will match after a context switch. Therefore, until some tags match, the choice PHT will serve as the predictor.

In a sense, YAGS and the *bi-mode* predictors are hybrid predictors, which combine the *gshare* scheme with the *bi-model.* In the presence of a context switch, they should exhibit the short warm-up time of the *bimodal* predictor.

In order to simulate a context switch, a new trace file was created by interleaving all eight SPECINT95, the six SPECFP95, and the IBM benchmarks every 60,000 instructions.

This number was chosen not to reflect a real context switch interval, but to exagerate the effect of context switching on the various predictors.

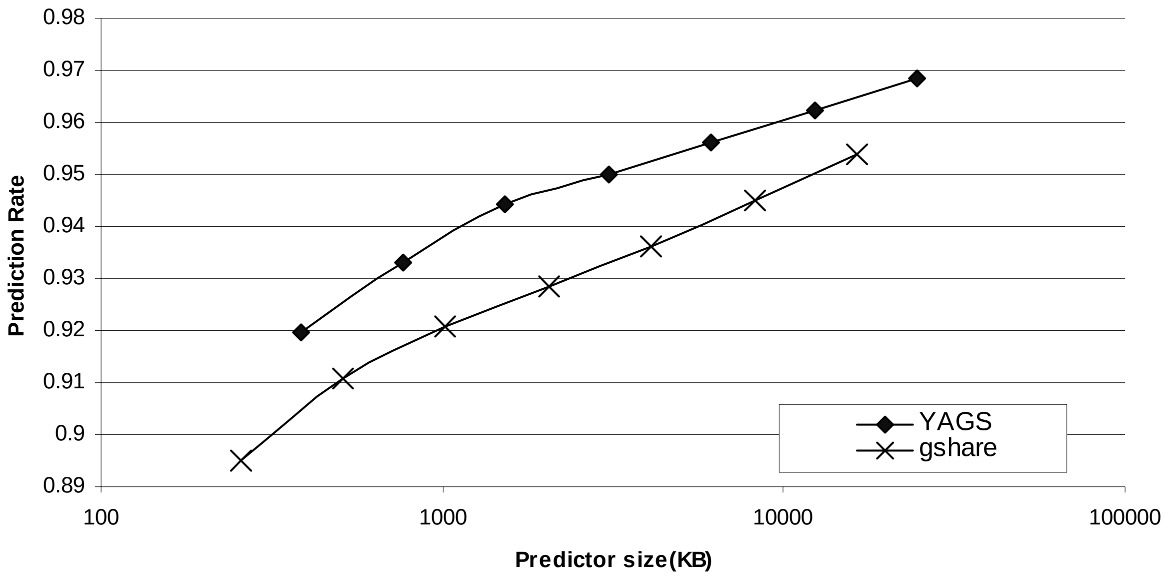


**Figure 8-4 – Context Switching Effect on the Different Predictors**

Figure 8.4 shows the performance of the predictors tested in the presence of context switches. As expected, YAGS performs much better than *gshare* because of its short warm-up times. The difference between the accuracy of the different predictors is much more pronounced in the presence of context switches. The *gshare* structure would converge with YAGS only if the PHT was large enough to accommodate most of the branch instances from all the SPEC95 benchmarks. Without context switches, the predictor's performance would converge if the *gshare* PHT were big enough to accommodate the benchmark with the largest branch signature.

## 8.2 In-Depth Analysis

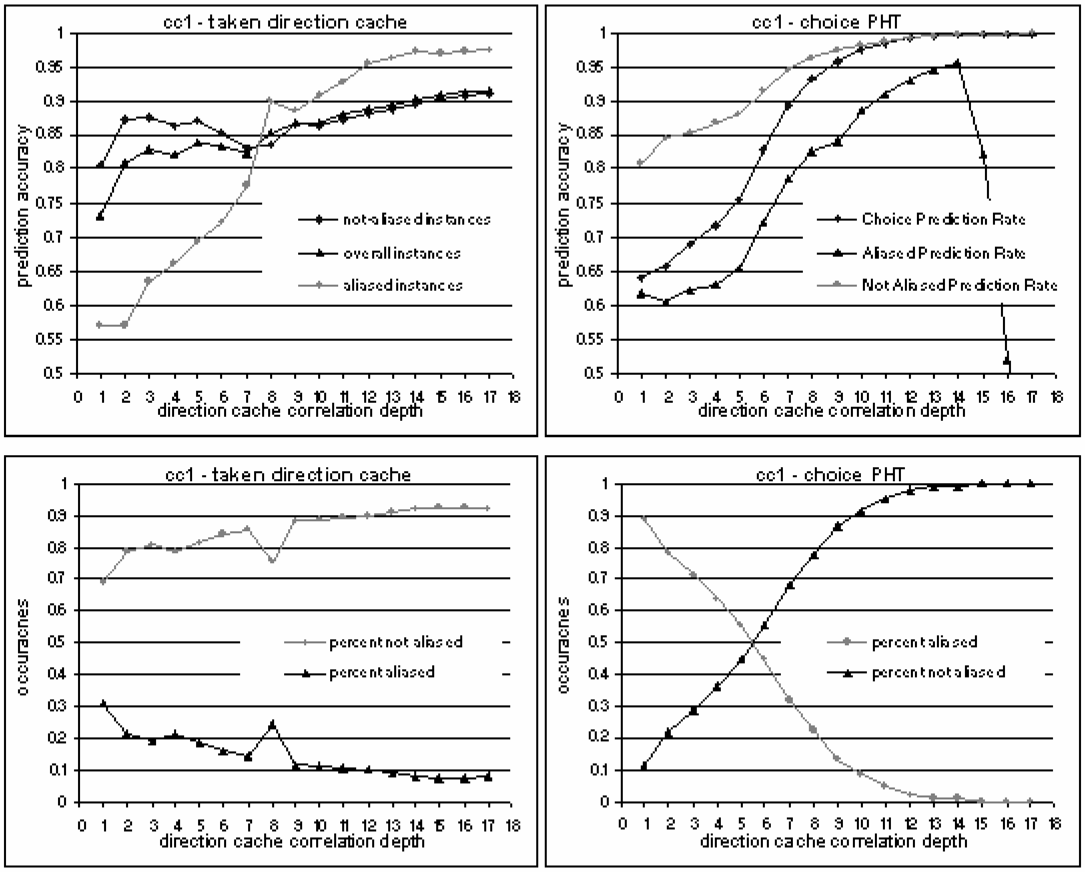


**Figure 8.5 – aliased and not-aliased instances usage and prediction accuracy for the choice PHT and taken direction cache**

YAGS outperforms leading branch predictors because of its ability to reduce aliasing. How much aliasing still exist in each of YAGS' structures and the inter-working of the choice PHT with the direction caches is not clear. We next present a series of studies that gives an insight to the inter-working of YAGS. We chose to show only the results for the gcc benchmark because results for other benchmarks are similar and do not result in a better insight into YAGS.

Figure 8.5 shows the prediction accuracy and the usage of the taken direction cache and the choice PHT for the gcc benchmark in a grid of four graphs. Results for the not taken direction cache are not shown for they are very similar to the result of the taken direction cache.

For the choice PHT, prediction accuracy of not-aliased instances is expectedly higher than the overall prediction accuracy and the prediction accuracy of the aliased instances. As the predictor size increases not-aliased prediction accuracy and overall prediction accuracy converges. The reason is that the percentage of aliased instances is approaching zero. As the size of the predictor increases, the number of choice PHT hits is decreasing. The reason is that increase predictor size comes with increased correlation depth. As a result the number of instances, which do not comply with the bias, increase, and the direction caches are utilized more heavily. Because a hit in the choice PHT only occur when there is a miss in the direction cache, the increase utilization of the direction caches result in a smaller utilization of the choice PHT.

In the direction cache, the number of hits increases as the predictor increase in size. The reason is an increase utilization of correlation depth and was discussed above. As in the choice PHT the prediction accuracy of the not-aliased instances is greater than the prediction accuracy overall for small predictors. The prediction accuracy of the two converges for larger predictors. Interestingly enough, the prediction accuracy of the aliased instances increases as the predictor size increases. At certain predictor size it surpass the prediction accuracy of the not-aliased instances. However, the increase in accuracy as little effect on the overall prediction accuracy because the number of aliased instances decreases as the predictor size increases. The explanation for that is hinted in chapter 4. Notice that aliased instances in some why implement the history only scheme. In chapter 4 it was shown that with larger correlation the history scheme surpass the global scheme in prediction accuracy.  In contrast to what this result might suggest, a YAGS version that implement the history only scheme doesn't work as well and the YAGS that implement the global scheme.

## 8.3 And Yet More Aliasing

The introduction of the two-level branch predictor [22] made the *bimodal* branch predictor obsolete in the eyes of researchers.  The promising potential of the two-level branch predictors led to an outpouring of research into correlating branch predictors. When aliasing was discovered to degrade performance in the two-level branch predictors, its negative effect on one-level branch predictors, such as the *bimodal* structure, was no longer a priority.  While it was established that aliasing in the *bimodal* structure did not occur often as in the second-level PHT, it appears that its adverse effect on prediction was never investigated, and no solution to aliasing in the first-level table was ever proposed.

The effect of aliasing in one-level branch predictors is of interest for few reasons.  For one, micro-architectural trends, such as increasing code size and decreasing state reachable in one cycle, might force processors to scale down their branch predictors to one-level branch predictors.  Even if it is possible to avoid the *bimodal* structure, some aliasing reducing structures has a *bimodal structure* embedded in them.  Examples for such predictors are the *bi-mode,* YAGS and the selection mechanism in most hybrid predictors.  If aliasing degrades performance in the *bimodal* structure, it ought to degrade the performance of those branch predictors too.

In the *bimodal* scheme, each branch needs only one entry, compared to the global scheme with a BHR of size 10, where each branch theoretically needs 1024 entries.  This suggest there will be much less aliasing in the *bimodal* predictor compared to the *gshare* predictor.  However, since each aliasing instance in the *bimodal* predictor adversely effects all instances of this branch, we expect each aliasing instance in the *bimodal*

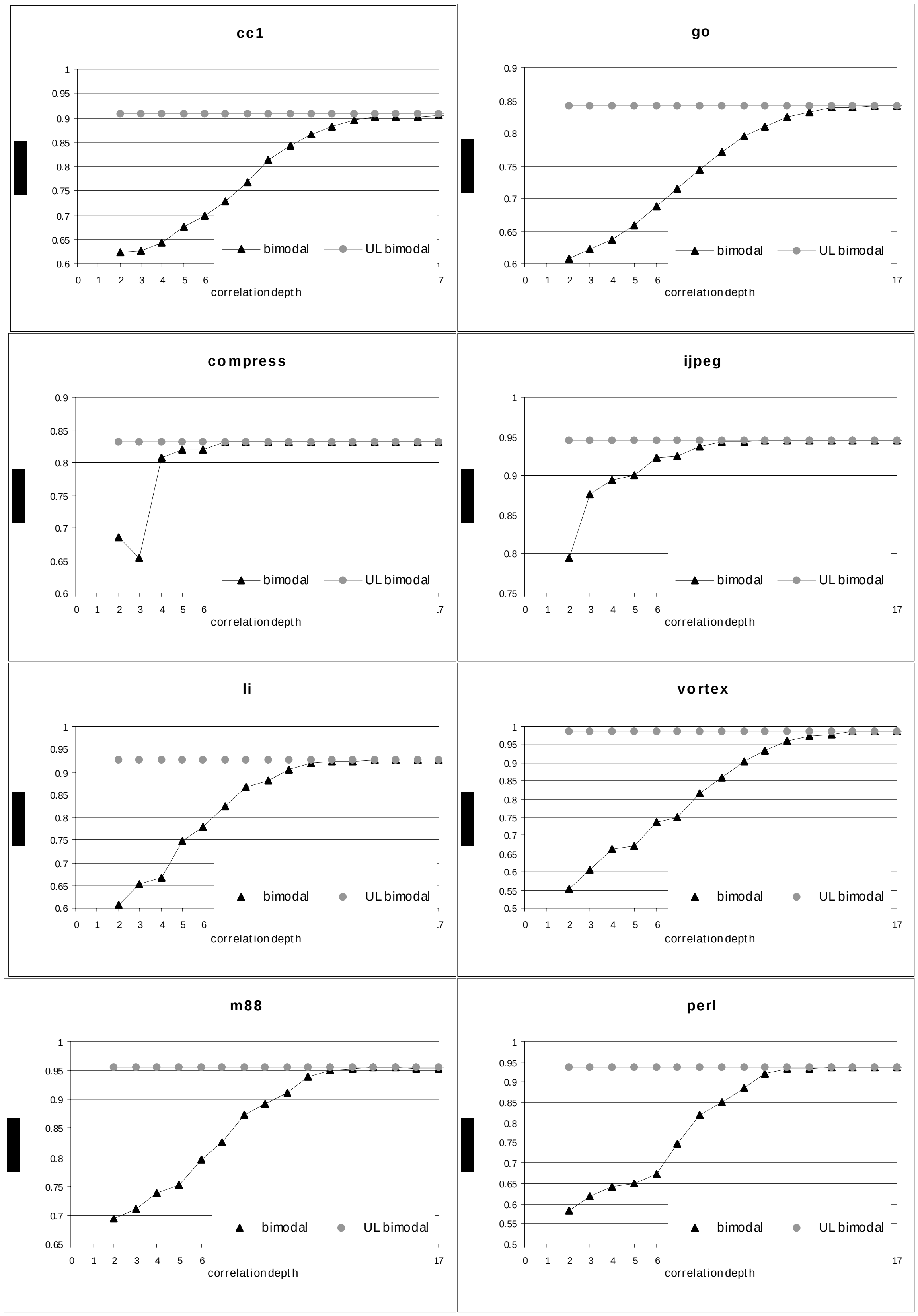


**Figure 8.6 a) Aliasing in the bimodal predictor for the SPECINT95 benchmarks**

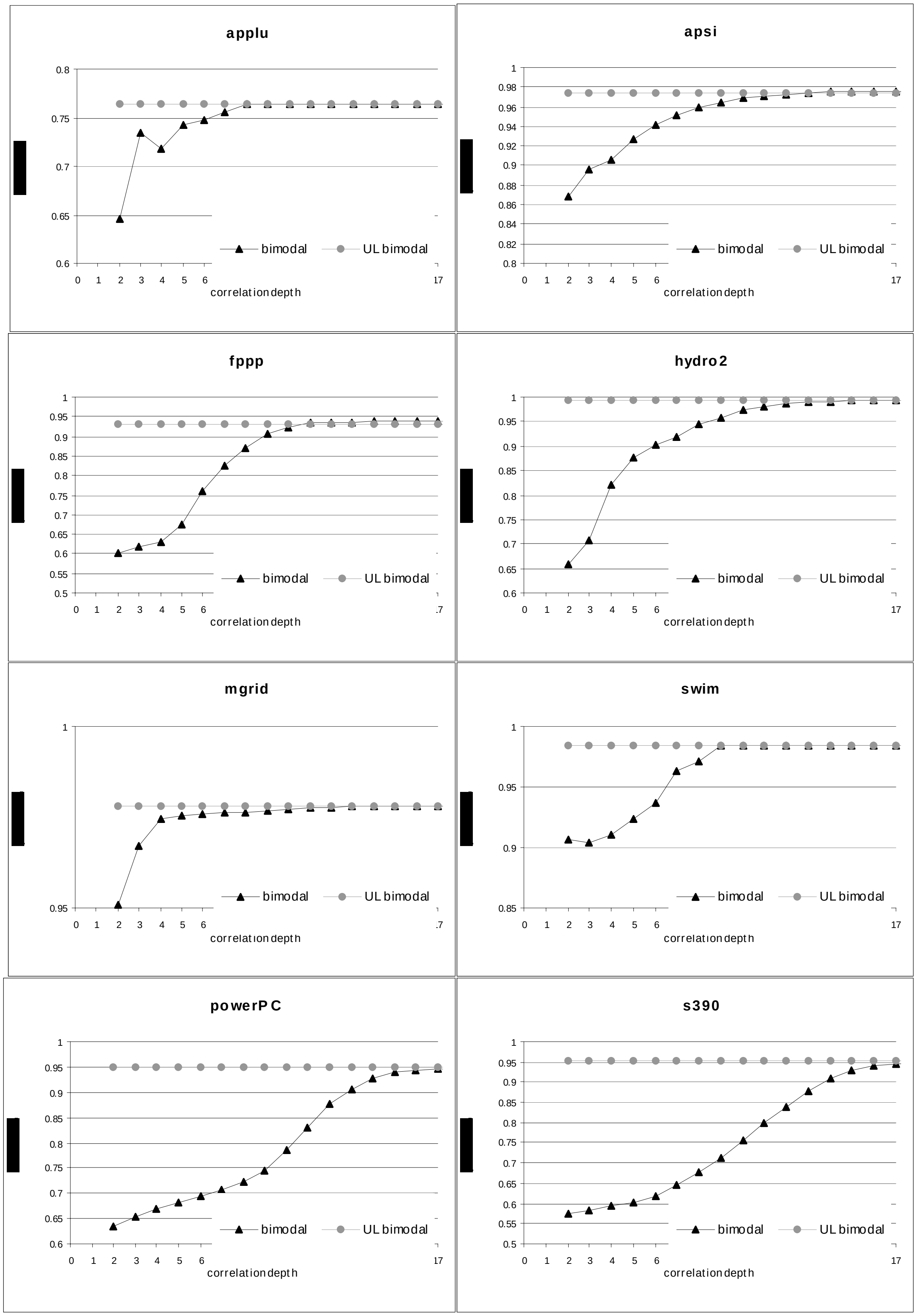


**Figure 8.6 b) Aliasing in the bimodal predictor for the SPECFP95 and IBM benchmarks**

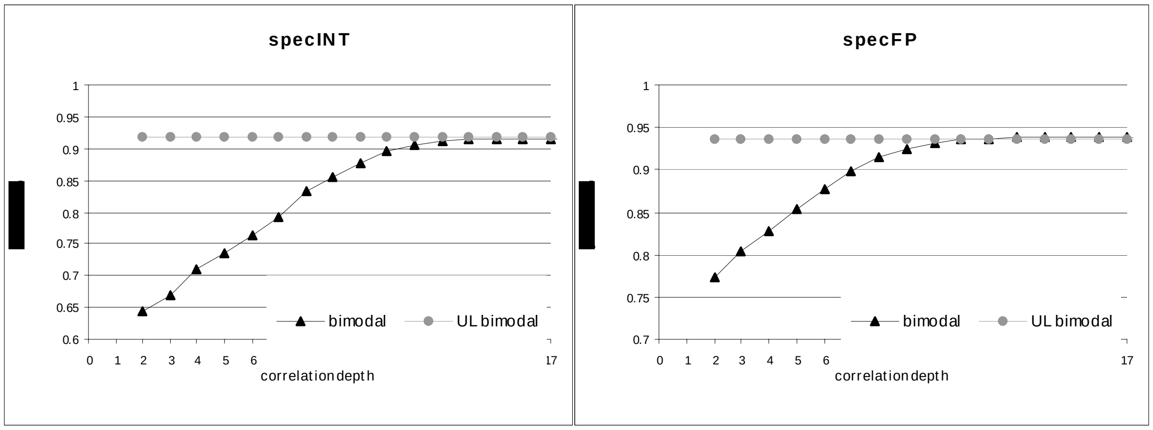


**Figure 8.6 c) Aliasing in the bimodal predictor for the SPECINT95 and SPECFP95 averages**

predictor to have a much more destructive effect than an aliasing instance in the gshare predictor.

Figure 8.6 depicts the adverse effects of aliasing in the *bimodal* structure. It compares a *bimodal* implementation that has a dedicated entry for each branch (UL- bimodal) to a regular *bimodal* structure. The x-axis represents log to the base 2 of the number of entries for the *bimodal* structure, but has no significant for the UL-bimodal. For small tables, aliasing degrades performance for all benchmarks. However, the point where aliasing ceases to be a problem varies significantly across benchmarks. While for the compress benchmark, aliasing does not degrade performance for tables as small as 64 bytes, for the s390 benchmark, aliasing persists as a problem even for tables as large as 16KB. If s390 and other benchmarks with large number of static branches represent future programs, aliasing in one-level branch predictors will significantly degrade prediction accuracy. But even for the average of the SPECINT95, aliasing degrades prediction accuracy for 2KB tables.

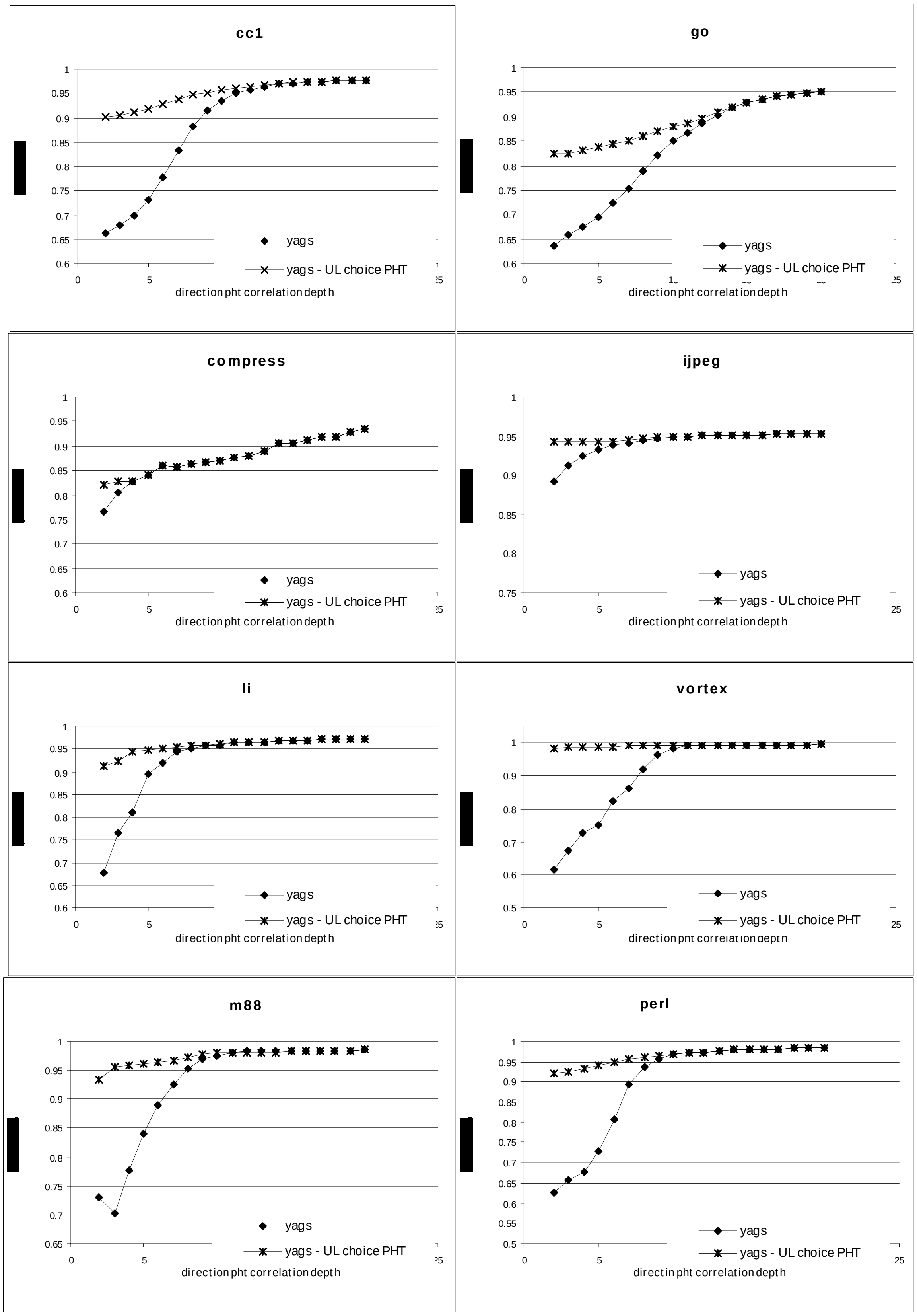


**Figure 8.7 a) Choice PHT aliasing effect in YAGS for the SPECINT95 benchmarks**

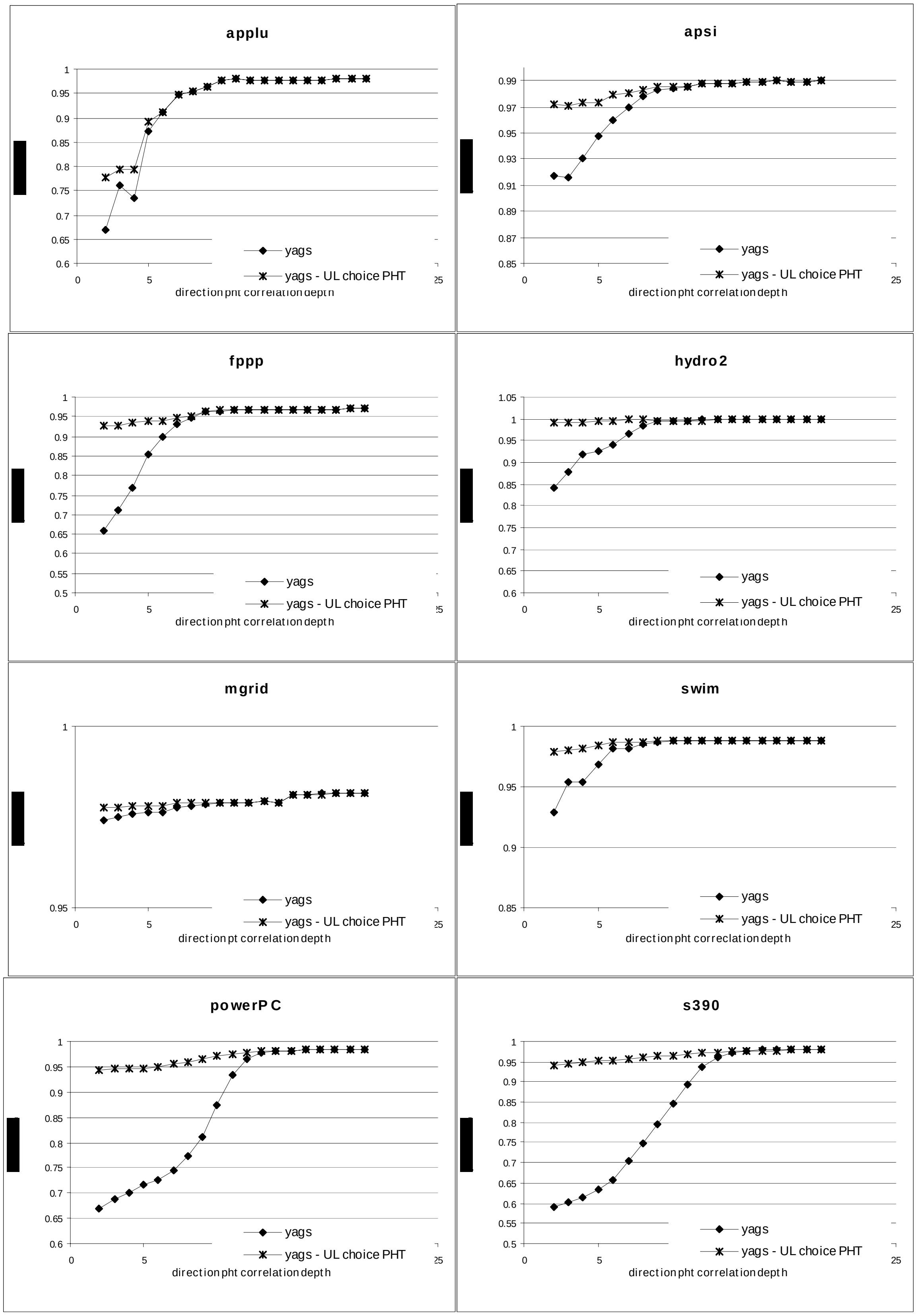


**Figure 8.7 b) Choice PHT aliasing effect in YAGS for the SPECFP95 and IBM benchmarks**

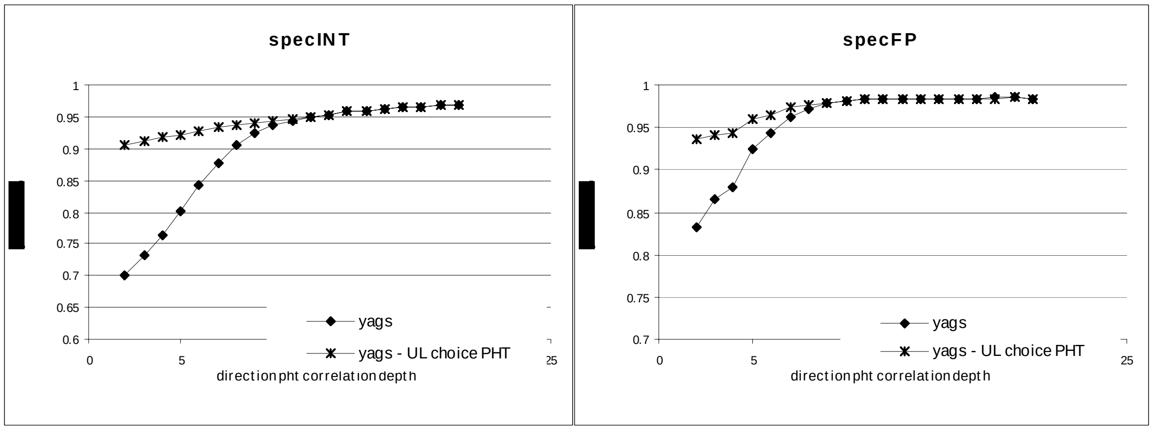


**Figure 8.7 c) Choice PHT aliasing effect in YAGS for the SPECINT95 and SPECFP95 averages**

Because aliasing in the *bimodal* predictor degrades prediction accuracy, it could be expected to similarly degrade the prediction accuracy in two-level branch predictors that have an embedded *bimodal* structure. This premise is checked in the context of the proposed YAGS predictor. Figure 8.7 compares the basic YAGS structure against a hypothetical YAGS structure with no resource limit on the choice PHT. Notice that the direction caches in this hypothetical YAGS are still restricted in size. Comparing this hypothetical YAGS to the regular YAGS enables us to quantify the adverse effects that aliasing in the choice PHT has on the prediction accuracy of YAGS. As can be seen, the prediction accuracy degradation due to aliasing in the choice PHT is significant, and for benchmarks with large numbers of static branches, like the s390, aliasing in the bimodal structure is only resolved for choice PHTs of 4KB in size. For smaller predictors, the adverse effects of aliasing in the choice PHT overwhelms the benefits the predictor achieves from implementing the global scheme via the direction caches.

Obviously, aliasing in the *bimodal* structure is a prominent source of prediction accuracy degradation that has so far been overlooked. Finding ways to remove some of the aliasing from the *bimodal* structure could bring significant benefits to prediction, especially for smaller size predictors.

## 8.4 Profile YAGS

Removing aliasing from the choice PHT of YAGS is important for two reasons. First, it will increase prediction accuracy. Second, it will facilitate a reduction in the size of the choice PHT compared to the direction caches. This, in turn, facilitates turning YAGS into a cascading predictor where a small choice PHT is used when only one cycle is available for prediction. When more than one cycle is available for prediction, the direction caches can then be accessed.

Attempts to create a dynamic structure that alleviates aliasing in the first-level structure, in this dissertation, have failed. This failed attempt is described next. Borrowing from the *bi-mode* structure, the choice PHT was split into two choice PHTs. One serves branches that are mostly biased to be taken and the other serves branches that are biased to be not taken. The determining factor of which choice PHT will server each branch was whether the branch is a forward or a backward branch. The reasoning behind this choice is similar to the static branch prediction scheme that states "backward taken, forward not taken." However, this attempt failed.

Next, a profiled version of YAGS is presented. The direction caches are similar to the ones in YAGS. The choice PHT, on the other hand, is replaced by a bit in the branch instruction that indicates the branch's bias. Using a profiling bit to help in the prediction of a two-level branch predictor was described before in the context of an *agree* like predictor [67], however, this does not exclude profile YAGS as a novel predictor.

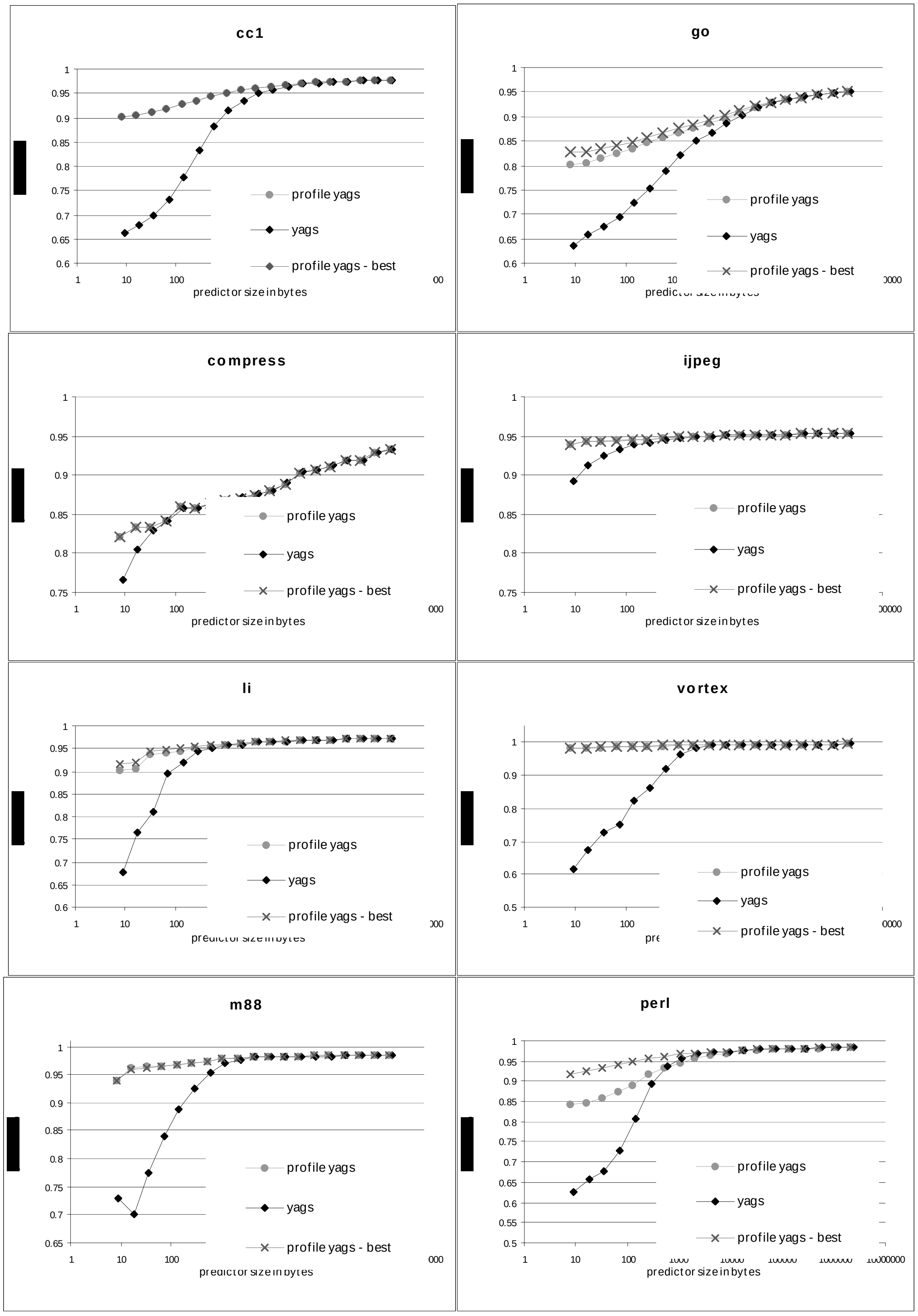


**Figure 8.8 a) Profile YAGS for the SPECINT95 benchmarks**

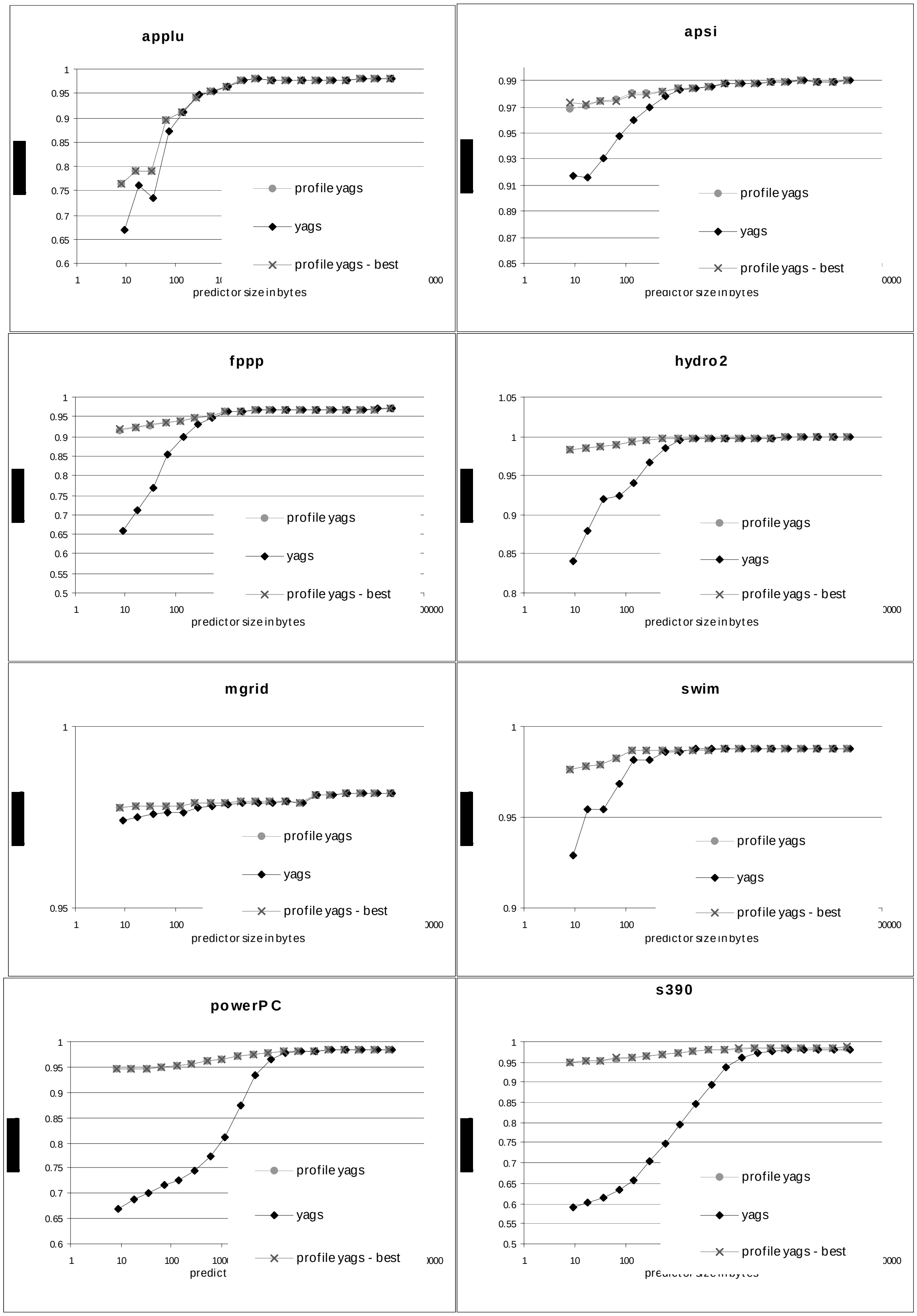


**Figure 8.8 b) Profile YAGS for the SPECFP95 and IBM benchmarks**

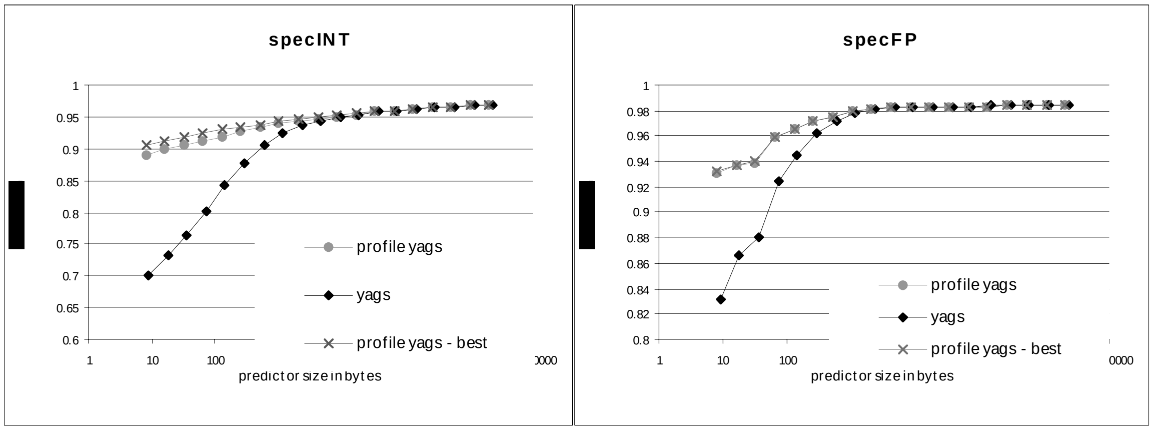


**Figure 8.8 c) Profile YAGS for the SPECFP95 and SPECINT95 averages**

Figure 8.8 compares the regular YAGS with profile YAGS when a test dataset is used for profiling (profile-yags) and when the same dataset is used for profiling and real simulations (profile-yags-best). The later is an overestimation and marks the upper bound that can be achieved by profiling. Surprisingly, the omniscient profile YAGS and the feasible profile YAGS produce very similar results.

That static selection mechanism work so well with YAGS as opposed to the serving as the selection mechanism for the McFarling predictor, stems from the underlying information profiled. With the McFarling predictor, the profiled information indicates which predictor better predicts the branch. This information is not balanced and is heavily biased toward either the local or the global component, depending on the benchmark. On the other hand, with YAGS, the profiled information indicates the branch's bias. This tends not to change for different datasets and is fairly balanced between branches that are biased to be taken, and branches that are biased not to be taken. Figure 8.8 shows that profiling the branches' bias for YAGS entails large prediction benefits for small predictors. The reasons are twofold. First, the cost of the choice PHT is zero because this information is stored in the branch instruction. Second, there is no aliasing in the first-level structure. Aliasing was shown in Section 8.2 to considerably degrade performance of global two-level branch predictors.

As the size of the predictor increases, the benefits of using profile YAGS over YAGS diminish until they are nonexistent for most benchmarks. It is worthwhile to note the predictor size where this benefit diminishes. For the applu benchmark the critical size is as small as 128 Bytes. For the s390 benchmark it occurs for predictors as large as 256KB in size. On average for the SPECINT95, the critical point is 18KB. This is by far larger than the conservative estimate of the PHT of 1KB that would be accessible in one cycle for the 35nm technology available in the year 2012 [64]. For a 1KB PHT, the prediction accuracy of profile YAGS is 93.9% for the SPECINT95 average compared to only 92.5% prediction accuracy for YAGS. If the more conservative estimate for future technology is considered, a PHT of size 0.125KB will be available in the year 2012 and the prediction accuracy of profile YAGS will be 92.7% compared with only 87.6% prediction accuracy for YAGS.

The benefit of using profiling to determine the branches' bias in future technologies is obvious. However, profiling is not as easy implemented and some major hurdles were overlooked in the past when profiling was studied in branch predictors. The next section addresses those problems.

## 8.5 The Future of Profiling

Profiling has been used for static methods in branch prediction for as long as the branch prediction field of research has existed. Profiling allows the acquisition of information before the program is run.  In contrast, dynamic methods acquire their information while the program runs.  This difference between dynamic and static methods imposes less time constraints on static profiling than on dynamic methods.  The profiled information is conveyed to the microarchitecture via the ISA.  A well-known pitfall of profiling is that the ISA must have a mechanism to convey the profile information to the processor.  This mechanism usually comes in the form of dedicated bits in the branch instructions.  The lack of such a mechanism requires a change in the ISA.  A change to the ISA is not trivial in most cases and might cause problems with backward compatibility.

Next, three other pitfalls are discussed that, we think, will force other means of profiling to be devised.  The creators of the SPEC benchmark suite, recognized that different datasets must be supplied.  Most studies, therefore, profile with one dataset and then use the profiled information to measure the prediction accuracy when running on a different dataset.  More than 97% of the static branches are profiled with the test dataset, and therefore when running the real dataset those branches already have prediction information.

### 8.5.1 Profiling Pitfalls

The first profiling pitfall is that SPEC95 benchmarks are very small programs compared to today’s software.  Additionally, the active regions of the program are used regardless of the dataset.  This is not the case for large programs such as MS-WORD or EXCEL.  The amount of code for those applications is enormous compared to the SPEC95 benchmarks and the active regions of such program will change considerably with different datasets. As a result, coming up with a dataset that will profile most of the static branches in the program is an almost impossible task. Those un-profiled regions of the program are not necessary less used. It might be the case that those regions will be used over and over again by the same user. As a result, this user will experience considerable slowness when running those un-profiled portions of the applications.

The second pitfall relates to legacy code. Legacy code refers to programs that were compiled and are running on older implementations of the same ISA. When legacy code is installed on a new implementation of the ISA that relies on profiling for branch prediction, it can considerably degrade its performance. This might lead to the disconcerting situation that an older processor will run legacy code faster than a newer implementation of the same ISA.

The last pitfall of profiling is the commitment to a branch prediction implementation. Suppose one version of the ISA is implemented with the McFarling branch predictor where the selection mechanism relies on profiling. Programs are compiled after profiling sample datasets for optimal accuracy for the McFarling predictor. A new implementation of the same ISA cannot change the McFarling predictor to the *bi-mode* predictor because previous profiled and compiled programs will suffer considerable performance degradation. The same problem can occur even if the new processor implements the McFarling predictor with a different PHT size than the older version.

Those overlooked pitfalls of profiling in branch prediction can be easily generalized to profiling in different structures of a microarchitecture. This suggests that dynamic structures should be used because they do not suffer from the pitfalls mentioned above. However, as we have seen before, due to micro-architectural trends, dynamic structure will be forced to decrease in size and will therefore suffer a reduction in performance. Profiling, on the other hand, is not limited in size and does not suffer from aliasing.

### 8.5.2 Dynamic Profiling

The solution to this problem might be a different way of profiling. Profiling needs to be done during run time rather then compile time to avoid all the above pitfalls. This can be done either by the processor or the OS. A simple example might be the bias of a branch. A bit in the branch instruction indicates whether the branch was profiled or not. If the branch has not yet been profiled, when the branch is evicted from the BTB, the processor could interject an instruction to write the branch back into memory with the profile information obtained by a 2bc attached to the BTB. The first time the program is run, it might encounter delays due to large dynamic structure. The next time the program is executed, branches with profile information in the ISA obtained in the previous run will be predicted by this profile information. This bypasses the delay due to a large dynamic structures and at the same time avoids the above mentioned pitfalls. This approach was proposed before albeit serious structure limitations [62].

The profiling done by the processor is restricted to very simple profiling. Some profiling needs elaborate data structures that are not cost effective to implement in hardware. If profiling is done by the OS, in a similar manner described for processor profiling, elaborate data structures can be used and the cost of on-chip dynamic structures can be avoided. This will require an upgrade to the OS for each processor, but can be easily done.  The OS will maintain a list of profiled programs and on every program that was not profiled, the OS will invoke the profiling module.

Dynamic profiling is not an easy idea to implement.  It requires the cooperation of the microarchitecture and the OS.  It is clear that future microprocessors will have to be more tightly coupled with the OS in order to maintain increasing processor performance.

# Chapter 9 - Conclusion

## 9.1 Contributions

4The initial objective of the research leading to this dissertation was to improve branch prediction accuracy by combining different advances made in the branch prediction field. In the process of investigating the feasibility of this approach, several myths and misconceptions were debunked. Throughout this dissertation, those misconceptions have been clarified, and ways were devised to improve the accuracy of branch predictors.

The branch prediction research community has taken three different paths in its attempts to improve branch prediction. In an initial attempt to combine the benefits of a hybrid branch predictor with the benefits of a branch prediction structure which reduces aliasing, we observed that both predictors showed improved branch prediction for the same reason– namely reducing aliasing. In a series of studies we consolidated the three different branch prediction paths by reducing the hybrid and the third-level paths to the ‘aliasing path’. This reduction is done by showing that hybrid predictors improve prediction by reducing aliasing, and by showing that the third-level of adaptivity is a filtering mechanism that also reduces aliasing.

In the process of investigating what makes the hybrid path work, we shed light on some myths in the ongoing debate between static and dynamic selection mechanisms in hybrid predictors. We found that known dynamic selection mechanisms fail to choose the best component for each branch dynamically and we questioned whether a branch changes its best predictor during execution. It was found that the strength of dynamic selection mechanisms does not lie in dynamically choosing the best predictor for each branch, but in reduction aliasing. The dynamic selection mechanism serves as a load balancer between the two components of the hybrid predictor. Once one of the components is congested, the selection mechanism moves some branches to the other component. This load balancing process reduces aliasing, which in turn translates into better prediction accuracy. While this role of the dynamic selection mechanism in reducing aliasing has not been noted before, it has been well established that a static selection mechanism reduces aliasing by not taking any hardware resources for the selection mechanism and by reducing information for the reason that only one component is updated for each branch. For static selection mechanism, we showed that profiling must be done with aliasing in mind. If profiling is focused on the benefits of using the hybrid scheme instead of aliasing aware profiling, the degradation in performance can be considerable.

After concluding that reducing aliasing is the only way to improve prediction in two-level branch predictors, we categorized the different ways reducing aliasing was done in the past. The different ways are reducing the information stored in the PHT (and its simplified case – filtering), reducing negative aliasing, and pseudo-associativity. Studies conducted for this dissertation were then used to list the attributes found to help prediction

We laid out the tradeoff of increasing the size of the BHR, which has been empirically observed before but never adequately explained. We further showed that the size of the BHR and the number of static branches, present in a program, have more or less equal effect on the extent of aliasing. This is true in spite of the fact that the size of the BHR is the dominant term in the theoretical equation dictating the amount of information stored in the PHT (Section 4.5). By explaining the relationship between correlation and aliasing, we noted misinterpretations made in the past when two branch predictors that reduce aliasing, were compared. To take advantage of this observation, we proposed a way to decouple correlation from the size of the PHT even when the size of the PHT has fewer than $2^{\text{BHR Size}}$ entries. While this method is not successful in most cases, it was the basis for further improvement of YAGS. YAGS is a new branch predictor proposed that takes advantage of lessons learned while gathering the results for this dissertation.

Aliasing in the second level table of two-level branch predictors, and structures to alleviate the aliasing problem has been the subject of extensive research. However, to our knowledge, aliasing was never considered in the bimodal structure. We discovered it degrades performance. Bimodal aliasing currently degrades prediction for programs with large branch signatures. As program size increases, and the amount of state accessible in a cycle decreases, this problem is aggravated and most programs will suffer its negative effects. Since the bimodal structure is used as the choice PHT in the *bi-mode* and YAGS branch prediction structures and as a selection mechanism for different hybrid branch prediction structures, the aliasing problem in the bimodal structure can be expected to degrade performance for all those structures. We verified this for YAGS.

We propose a static selection mechanism to replace the choice PHT in the YAGS branch predictor based on profiling. Because we considered aliasing during the profiling process, and most branches' bias does not change between different datasets, profiling YAGS works extremely well even for small size allocation.

The strength of profiling YAGS is not only in a more cost effective use of resources, but also in its ability to eliminate the choice PHT to nothing. In the future, this ability can be used to transform YAGS and the *bi-mode* predictors into cascading predictors as a mean of tolerating the decreasing state accessed in a cycle.

This dissertation does not paint an optimistic picture of the branch prediction field. Even with generous resource allocation and no limit on the amount of state reached in one cycle, branch prediction is expected to create the most limiting bottleneck in future processors [1]. Moreover, code size is increasing [2] and the amount of state reached in a cycle is decreasing [63][64]. This means that future branch predictors will need to be smaller, and to predict more branches with better accuracy. This dissertation took the branch prediction field a step backward, in one sense, by consolidating three different paths of research into one, and as a result showed that advantages previously assumed were misleading and will not result in a better prediction. In addition to identifying these flaws, this thesis has identified and presented the new problem of aliasing in the bimodal structure. This problem will only get worse with the micro-architectural trends discussed above.

We hope that these observations will help to better direct future research in the branch prediction field. Since most of the myths uncovered in this dissertation could have been avoided by performing an adequate limit study, we look forward to seeing more studies that give insight into why a particular branch prediction structure/scheme works, moving beyond ad hoc empirical results which show that a particular branch predictor is more accurate than previously proposed ones.

On the positive side, we introduced YAGS, a new branch prediction structure that attacks the aliasing problem. YAGS utilized the criteria that we introduced (Chapter 7). Those techniques include but are not limited to decoupling correlation depth from PHT size, reducing the amount of information stored in the PHT, and load balancing. We facilitated the use of YAGS as a cascading predictor by utilizing profiling to determine the branch's bias. We stopped short of testing YAGS as a cascading predictor.

We acknowledge that YAGS, similar to previous branch prediction structures, does not completely solves the aliasing problem nor eliminates control dependency from being the bottleneck in future processors. However, any improvement in branch prediction accuracy will help to open up the control dependency bottleneck. For that, we believe that further research and novel branch predictor structures are needed in order to facilitate faster processors.

## 9.2 Future Work

Trends in microarchitecture and software development dictate that control dependency will continue to be a problem in the foreseeable future. This provides an exciting opportunity for future research.

Note that while we have shown that hybrid predictors improve prediction accuracy by eliminating aliasing, we have not ruled out hybrid schemes altogether. Rather we have pointed out that current hybrid predictors are unable to fulfill this potential. Once aliasing is eliminated, or a way is found to separate those two competing potentials, predictors can start taking advantage of the hybrid scheme potential.

The only existing study pertaining to cascading predictors [65] used the *gshare* structure and specifically mentioned that it would be beneficial to incorporate a structure that reduces aliasing with a cascading predictor. One criticism of the cascading predictor proposed is that it stores redundant information. We believe in the potential of YAGS to serve as a cascading predictor where the choice PHT is eliminated by the use of profiling information. When another cycle is available for prediction, the direction caches can than be accessed. While not pursuing this in its entirety, we made the first step by allowing the first cycle prediction to have no latency, while still taking advantage of the two-level structure when a second cycle is available for the prediction of the branch. Note that in order for the *bi-mode* structure to serve as a cascading branch predictor, the choice PHT will need to supply predictions, taking the *bi-mode* structure one step closer to the YAGS branch predictor.

We predict that compile/profile time information will need to be more closely coupled with dynamic predictors. This trend is not new and different ISAs incorporate bits in the ISA to do just that. For example in the IA64 ISA there is one bit in the branch instructions to indicate whether dynamic or static prediction is to be used for filtering purposes. Another bit is present to indicate the prediction in case a static prediction is used. This last bit can be easily used to indicate the branch's bias in profiling YAGS. Such compile and profile time cooperation will need to be increased in order for branch predictors to keep up with future processors. This might require ISA changes, and as painful as that might be to implement, we feel that it will be necessary. An obvious example is the loop instruction in the PowerPC ISA, which other ISAs are lacking.

Happy hunting.

# Bibliography

[1]. P. Ranganathan, and N. Jouppi. The relative impact of memory latency, bandwidth and branch limit to microprocessor performance. In *Proceedings of the 1st Workshop on Mixing Logic and DRAM: Chips that Compute and Remember (held in conjunction with the 1997 International Symposium on Computer Architecture),* June 1997.

[2]. R. Uhlig, D. Nagle, T. Mudge, S., Sechrest, and J. Emer. Instruction fetching: Coping with code bloat. In *Proceedings of the 22nd Annual International Symposium on Computer Architectur*e, June 1995, pp. 345-356.

[3]. M. Lam, and R. Wilson. Limits of control flow on parallelism. In *Proceedings of the 19th Annual International Symposium on Computer Architectur*e, May 1992, pp. 46-57.

[4]. A. Uht. A theory of reduced and minimal procedural dependencies. *IEEE Transaction on Computer*s, vol. 40, no. 6, June 1991, pp. 681-693.

[5]. E. Dijkstra. Guarded commands, non-determinacy, and formal derivation of programs. *Communication AC*M, vol.18, August 1975, pp.453-457.

[6]. P. Hsu, and E. Davidson. Highly concurrent scalar processing. In *Proceedings of the 13th Annual International Symposium on Computer Architectur*e, June 1986, pp. 386-395.

[7]. D. Pnevmatikatos, and G. Sohi. Guarded execution and branch prediction in dynamic ILP processors. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, April 1994, pp.120-129.

[8]. M. Lam, and R. Wilson. Limits of control flow on parallelism. In *Proceedings of the 19th Annual International Symposium on Computer Architectur*e, May 1992, pp. 46-57.

[9]. E. Sprangle, R. Chappell, M. Alsup, and Y. Patt. The Agree predictor: A mechanism for reducing negative branch history interference. In *Proceedings of the 24th Annual International Symposium on Computer Architectur*e, May 1997.

[10]. P. Chang, M. Evers, and Y. Patt. Improving branch prediction accuracy by reducing pattern history table interference. In *Proceedings of the International Conference Parallel Architecture and Compilation Technique*s. October 1995.

[11]. P. Michaud, A. Seznec, and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architectur*e, May 1997.

[12]. C. Lee, I. Chen, and T. Mudge. The bi-mode branch predictor. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitectur*e, 1997.

[13]. P. Chang, E. Hao, T. Yeh, and Y. Patt. Branch classification: A new mechanism for improving branch predictor performance. In *Proceedings of the 27th Annual ACM/IEEE International Symposium on Microarchitectur*e, November 1994.

[14]. S. McFarling. Combining branch predictors. *WRL Technical Note TN-3*6, June 1993.

[15]. A. Talcott, M. Nemirovsky, and R. Wood. The influence of branch prediction table interference on branch prediction scheme performance. In *Proceedings of the 3rd International Conference on Parallel Architecture and Compilation Techniqu*es, June 1995.

[16]. C. Young, N. Gloy, M. and Smith. A comparative analysis of schemes for correlated branch prediction. In *Proceedings of the 22nd Annual International Symposium on Computer Architectur*e, June 1995.

[17]. C.C. Lee. Optimizing high performance dynamic branch predictors. *Ph. D. Dissertation, Univ. of Michigan, Ann Arbor*, November 1997.

[18]. M. Evers, S. Patel, R. Chappell, and Y. Patt. An analysis of correlation and predictability: What makes two-level branch predictors work. In *Proceedings of the 25th Annual International Symposium on Computer Architectur*e, 1998, pp. 52-61.

[19]. S. Sechrest, C. Lee, and T. Mudge. Correlation and aliasing in dynamic branch predictors. In *Proceedings of the 23rd Annual International Symposium on Computer Architectur*e, May 1996.

[20]. M. Tarlescu, K. Theobald, and G. Gao. Elastic history buffer: A low-cost method to improve branch prediction accuracy. In *Proceedings of the 1997 International IEEE Conference on Computer Desig*n, 1997, pp. 82-87.

[21]. T. Juan, S. Sanjeevan, and J. Navarro. Dynamic history-length fitting: A third level of adaptivity for branch prediction. In *Proceedings of the 25th Annual International Symposium on Computer Architectur*e, 1998, pp. 155-166.

[22]. T. Yeh, and Y. Patt. Two-level adaptive training branch prediction. In *Proceedings of the 24th Annual ACM/IEEE International Symposium on Microarchitectur*e, November 1991, pp. 51-61.

[23]. T. Yeh, and Y. Patt. Alternative implementations of two-level adaptive branch predictions. In *Proceedings of the 19th Annual International Symposium on Computer Architectur*e, May 1992, pp. 124-134.

[24]. T. Yeh, and Y. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *Proceedings of the 20th Annual International Symposium on Computer Architectur*e, May 1993.

[25]. C. Yound and M. Smith. Improving the accuracy of static branch prediction using branch correlation. In *Proceedings of the 6thInternational Conference on Architectural Support for Programming Languages and Operating System*s, October 1994, pp. 232-241.

[26]. R. Nair. Dynamic path-based branch correlation. In *Proceedings of the 28th Annual ACM/IEEE International Symposium on Microarchitectur*e, 1995, pp. 15-23.

[27]. L. Gwennap. Intel's P6 uses decoupled super-scalar design. *Microprocessor Repor*t, vol. 9, no.2, February. 16, 1995.

[28]. L. Gwennap. Digital 21264 sets new standard. *Microprocessor Repor*t, vol. 10, no. 14, October. 28, 1996.

[29]. D. Pnevmatikatos, and G. Sohi. Guarded execution and branch prediction in dynamic ILP processors. In *Proceedings of the 21st Annual International Symposium on Computer Architectur*e, April 1994.

[30]. I. Chen, C. Lee, M. Postiff, and T. Mudge. Tag-less two-level branch prediction schemes. *Technical Report CSE-TR-306-96, University of Michigan*, 1996.

[31]. S. Sechrest, C. Lee, and T. Mudge. The role of adaptivity in two-level adaptive branch prediction. In *Proceedings of the 28thAnnual ACM/IEEE International Symposium on Microarchitectur*e, 1995.

[32]. W. Hwu, T. Conte, and P. Chang. Comparing software and hardware schemes for reducing the cost of branches. In *Proceedings of the 16th Annual ACM/IEEE International Symposium on Microarchitectur*e, May 1989.

[33]. K. Driesen, and U. Hölzle. Accurate indirect branch prediction. In *Proceedings of the 25th Annual International Symposium on Computer Architectur*e, 1998, pp. 167-178.

[34]. K. Driesen, and U. Hölzle. Improving indirect branch prediction with source- and parity-based classification and cascaded prediction. *Technical Report TRCS98- 07, Department of Computer Science, University of California, Santa Barbara*, March 1998.

[35]. J. Smith. A study of branch prediction strategies. In *Proceedings of the 8th Annual International Symposium on Computer Architectur*e, May 1981, pp. 135-148.

[36]. R. Nair. Optimal 2-bit branch predictors. *IEEE Transaction on Computer*s, vol. 44, no. 5, May 1995.

[37]. S. McFarling, and J.L. Hennessey. Reducing the cost of branches. In *Proceedings of the 13th Annual International Symposium on Computer Architectur*e, June 1986.

[38]. J. Fisher, and S. Freudenberger. Predicting conditional branch directions from previous runs of a program. *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System*s, 1992.

[39]. D. Kaeli, and P. Emma. Improving the accuracy of history-based branch prediction. *IEEE Transaction on Computer*s, April 1994.

[40]. C. Young, and M. Smith. Improving the accuracy of static branch prediction using branch correlation. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating System*s, 1994, pp. 232-241.

[41]. J. DeRosa, and H. Levy. An evaluation of branch architectures. In *Proceedings of the 14th Annual International Symposium on Computer Architectur*e, June 1987, pp. 10-16.

[42]. J. Lee, and A. Smith. Branch prediction strategies for branch target buffer design. *IEEE Computer,* Jan. 1984, pp.6-22.

[43]. SPEC CPU'95, *Technical Manual*, August 1995.

[44]. S. Pan. K. So, and J. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System*s, 1992, pp. 76-84.

[45]. Q. Jacobson, E. Rotenberg, and J. Smith. Path-based next trace prediction. In *Proceedings of the 30th International Symposium on Microarchitectur*e, 1997.

[46]. S. Raches and S. Weiss. Implementation and analysis of path history in dynamic branch prediction schemes. *IEEE Transaction on Computer*s, vol. 47, no. 8 Aug. 1998.

[47]. A. Eden and T. Mudge. The YAGS branch predictor. In *Proceedings of the 31st International Symposium on Microarchitectur*e, December 1998.

[48]. P. Chang, E. Hao, and Y. Patt. Alternative implementations of hybrid branch predictors. In *Proceedings of the 28th International Symposium on Microarchitectur*e, 1995, pp. 252-257.

[49]. M. Exers, P. Chang, and Y. Patt. Using hybrid predictor to improve branch prediction accuracy in the presence of context switches. In *Proceedings of the 22nd International Symposium on Computer Architectur*e, 1996, pp. 3-11.

[50]. A. Seznec and F. Bodin. Skewed-associative caches. In *proceedings of PARLE'93*, May 1993.

[51]. D. Grunwald, D. Lindsay, and B. Zorn. Static methods in hybrid branch prediction. In *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques,* 1998.

[52]. M. Evers. Improving branch prediction by understanding branch behavior. *Ph.D. Thesis, The University of Michigan*, 1999.

[53]. D. Burger, T. Austin. The SimpleScalar Tool Set, Version 2.0. *Technical Report TR 1342, University of Wisconsin*, June 1997.

[54]. J. Stark, M. Evers, and Y. Patt. Variable length path branch prediction. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operation Systems.* October, 1998.

[55]. E. Hao, P-Y. Chang , and Y. Patt. The effect of speculatively updating branch history on branch prediction accuracy, revisited. Proceedings of the 27th Annual International Symposium on Microarchitecture. November 1994, pp. 228-232.

[56]. S. Jourdan, T-H. Hsing, J. Stark, and Y. Patt. The effects of mispredicted-path execution on branch prediction structures. In *Proceedings of the of the 4th International Conference on Parallel Architecture and Compilation Techniques,* 1996, pp 58-67.

[57]. B. Calder, D. Grunwald, and J. Emer. A system level perspective on branch architecture performance. In *proceedings of the 28th Annual ACM/IEEE International Symposium on Microarchitecture,* pp. 199-206, 1995.

[58]. J. Rypley and D. Holloway. Performance tradeoff in sequencer design on a new G4 PowerPC microprocessor. In Proceedings of the 1st International Symposium on Performance Analysis of Systems and Software, 2000, pp. 88-94.

[59]. Virtual OCR reference.

[60]. L. Gwennap. Intel's MMX Speeds Multimedia. *Microprocessor Report, Vol. 10, No. 3,* March 1996.

[61]. J. Brown, S. Persels, and J. Meyer. Branch prediction unit for high-performance processor. *US Patent 5,394,529,* Feb 28, 1995. [First filed in 1990.]

[62]. T.M. Conte, B.A. Patel and J.S. Cox. Using branch handling hardware to support profile-driven optimization. In *Proceedings of the 27th International Symposium on Microarchitecture*, 1994 pp.12-21.

[63]. M. Horwitz, R. Ho and K. Mai. The future of wires. *In Semiconductor Research Corporation Workshop on Interconnects for Systems on a Chip*, May 1999.

[64]. V. Agarwal, M.S. Hirishikesh, S.W. Keckler, and D. Burger. Clock rate versus IPC: The end of the road conventional microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture.* June 2000, pp. 248-259.

[65]. D.A. Jimenez, S.W. Keckler, and C. Lin. The impact of delay on the design of branch predictors. *Proceedings of the 33rd International Symposium on Microarchitecture.* December 2000.

[66]. T.H. Heil, Z. Smith and J.E. Smith. Improving branch predictors by correlating on data values. In *Proceedings of the 32nd International Symposium on Microarchitectur*e, December 1999.

[67]. H.P. agree patent