6-1-1998

Avoiding Conflicts Dynamically in Direct Mapped Caches with Minimal Hardware Support

Peter N. DeSantis
Dartmouth College

Avoiding Conflicts Dynamically in Direct Mapped Caches with Minimal Hardware Support

Peter DeSantis
Senior Honors Thesis
Advisor: Thomas Cormen
Dartmouth College
Hanover, New Hampshire
Computer Science TR# PCS-TR98-339
June, 1998

Abstract

The memory system is often the weakest link in the performance of today's computers. Cache design has received increasing attention in recent years as increases in CPU performance continues to outpace decreases in memory latency.

Bershad et al. proposed a hardware modification called the Cache Miss Lookaside buffer which attempts to dynamically identify data that is conflicting in the cache and remap to pages to avoid future conflicts. In a follow-up paper, Bershad et al. tried to modify this idea to work with standard hardware but had less success than with their dedicated hardware.

In this thesis, we focus on a modification of these ideas, using less complicated hardware and focusing more on sampling policies. The hardware support is reduced to a buffer of recent cache misses and a cache miss counter. Because determination of remapping candidates is moved to software, sampling policies are studied to reduce overhead which will most likely fall on the OS. Our results show that sampling can be highly effective in identifying conflicts that should be remapped. Finally, we show that the theoretical performance of such a system can compare favorably with more costly higher-associativity caches.

1 Introduction

This paper evaluates the potential effectiveness of dynamically remapping pages in a large direct-mapped cache memory hierarchy in order to simulate associativity. Bershad et al. have investigated this technique both in hardware simulation and as a software implementation on standard hardware. While both these implementations showed reasonable success, the focus was on structure of the system, rathoer than the optimal parameters. In this paper, we offer a critique of this earlier work, and we extend the work to better identify the sampling tradeoffs of accuracy and cost.

Finally, using the results of our sampling study, we show that a recoloring policy with a direct-mapped cache should be able to perform significantly better than an equally sized 2-way associative cache.

1.1 Motivation

As computational speed continues to increase faster than memory access time, much focus is being placed on cache design. Caches are being built faster, bigger, and with greater associativity in attempts to reduce memory latency. However, there are inherent tradeoffs in cache design.

direct-mapped caches can be built faster and cheaper than n-way associative caches built with the same technology. The reason is that as associativity increases, more cache blocks have to be checked to see if the requested data is in the cache. This additional work requires additional hardware and lengthens the critical timing path, thereby increasing cost and hit times. In practice, direct-mapped caches tend to run between 2 and 12% faster than comparable 2-way associative caches. The major disadvantages of direct-mapped caches are that they tend to have much higher miss ratios, they are much more likely to exhibit terrible worst-case behavior, and they do not easily allow for parallel address translation. The advantages of parallel address translation are not significant when cache size gets large (straightforward parallel address translation requires cache size not to exceed page size times associativity). Additionally, in many modern cache architectures, virtual addressing is used, making parallel address translation unnecessary. [H88] The former two disadvantages create the most compelling case against direct-mapped cache architecture.

Many solutions to these problems have been investigated recently. One of the most interesting is dynamic page recoloring, where the goal is to identify pages that are contributing to a high quantity of misses in the cache and remap one of the pages in main memory. Remapping the page simulates associativity in the direct-mapped cache. The new page will be recolored, assigned to a different cache location, and

the data will no longer be contending for the same cache block thus eliminating future conflicts. [BCL1] For such a recoloring strategy to be effective, pages must be recolored only when the cost of recoloring is less than the cost of the averted future cache misses. To identify good recoloring candidates, a distinction must be made between various kinds of cache misses.

In a fully associative cache of size n (that is with n cache blocks), cache misses can classified as either compulsory or capacity misses. Compulsory misses are defined as the requisite, first miss on any address. Obviously if a piece of data has not been used before, then it will not be found in the cache, and the cache miss which results from the request will be compulsory. Any other miss in a fully associative cache can be considered a capacity miss. Because any piece of data can go into any cache block, for a piece of data, x, to be kicked out of the cache (assuming a least-recently replacement policy), n distinct cache blocks not containing x must have been referenced since the last reference to x. This access pattern implies that the pool of active data is greater than the capacity of the cache, and so the cache miss results directly from this insufficient capacity.

For any associativity less than full associativity, a third kind of cache miss may occur. If the same program is run on a fully associative cache of size n and also on an m-way associative cache of size n where m is less than n, any miss in the m-way associative cache that does not miss in the fully associative cache is a conflict miss. To see this behavior, the associativity set of the data item must be considered. Define the index of item x as the one of m sets in the cache to which x is mapped. A reasonable and common indexing scheme is ((Block address in main memory) modulo (cache size in blocks / m). The associativity set of an item x is the set of all data items which have the same index as x. After x has been loaded into the cache, a conflict miss occurs if m or more distinct references to cache blocks in x's associativity set not containing x have occurred since the last reference to x. [H96]

Using this distinction, the pathological case of the direct-mapped cache can be

discussed. In a direct-mapped cache, the associativity is 1; that is, each piece of data can be mapped to exactly one cache block. Since main memory tends to be many times larger than the cache, there will be many main memory blocks that map to the same cache block. Assume that a program alternately addresses two blocks in main memory that map to the same cache block. Each reference will result in a cache miss, and so the total memory latency will actually be worse than if there was no cache at all since all the cache overhead is wasted. In this case, a 2-way associative cache will perform perfectly. Even if the two pieces of data are in the same associativity set, they can both be in the cache at the same time. However, if we changed the program slightly so that it alternated between three addresses in three blocks in the same associativity set, then the 2-way associative cache performs as poorly as the direct-mapped cache in the previous example. Indeed there is a pathological case for any cache, but as the associativity increases, the likelihood of such a case actually occurring becomes increasingly unlikely.

Similarly, it is situations much like these pathological cases that cause the increased miss rates of lower associativity caches. Assume the program often finds itself toggling between references to pairs of pages that share the same cache block. Such a situation would create many conflicts in the direct-mapped cache, overusing some portions of the cache and under utilizing other parts. Without a dynamic remapping policy, the program has no way to recover from these poor page allocation situations.

In this thesis, we focus on the software sampling policies to identify pages that need remapping. The aim is to improve on the work of Bershad et al. by using less specialized hardware and reducing software overhead by employing effective sampling techniques.

1.2 The rest of the paper

In Section 2, we describe the related work on which our work builds. In Section 3, we describe the hardware on which our proposed system would run. In Section 4, we overview our sampling techniques and experiments. Section 5 presents the results of our sampling experiments. In Section 6, we look at some performance estimates of our system. Section 7 concludes with a brief discussion.

2 Related Work

As mentioned earlier, the original work in the area of dynamically identifying and recoloring cache conflicts was done by Bershad et al. The original work proposed the addition of hardware in order to monitor the cache at a very low cost. After this initial the work, the authors looked at ways to monitor the cache without the addition of costly dedicated hardware.

2.1 The CML

The Cache Miss Lookaside (CML) buffer attempted to provide a low overhead solution to identifying candidate pages for remapping. [BCL1] The device exploits the extra time available on a cache miss. On every cache miss, the address of the page that missed is sent to the CML. The CML has two sections of register pairs: HOT and LRU. Each register pair contains a page address and a miss count. When a page misses, the address is sent to the CML, and the buffer is searched. If the page is already in the buffer, its miss counter is incremented. If the page is not in the cache, then the least-recently used page in the LRU section (the register pair in the LRU section that has been accessed least recently) is replaced and its miss counter is reset to 1. Note that there is both a LRU section and a notion of an LRU entry, which must be in the LRU section. As soon as one of the LRU pairs exceeds the misses of

one of the members of HOT, then the pairs are swapped. In this way, pages that are likely to need remapping are prevented from being replaced. Finally, when a certain interrupt threshold has been met by any of the pages in the CML, an interrupt is sent to the operating system which then scans the CML remapping any pages above a recolor threshold, which is strictly less than the interrupt threshold.

Obviously, the performance of the CML is very dependent on the choice of HOT size, LRU size, the interrupt threshold, and the recolor threshold. The results of the paper suggest a maximum LRU size of 16 and a maximum HOT size of 8. Even with these sizes, the CML is a complicated hardware device. It is not clear that it will cost less to include a CML than it would to make the cache 2-way associative.

Additionally, this hardware solution has many disadvantages over a software solution. For one thing, the CML has no notion of context switching. Not preserving information across context switches could cause valuable information on the yielding process's cache misses to lost and prevent rapid identification of conflicts in the new running program.

There are many situations that could cause degradation of the performance of the CML. As an example, consider a CML with a LRU section of size of 8. Now consider a program that is doing repetitive array manipulation. Such a program might easily create misses in 9 or more pages that seem to almost cycle. So page 1 misses, followed by page 2, followed by page 3... followed by page 9, followed by page 1, followed by page 2... This cycling could continue for a very long time, but no entry in the LRU section would ever get more than one conflict. Therefore, no page would ever be identified as needing recoloring. Many such situations can be created, and because of the inflexibility of the hardware design, they will certainly create significant problems for the CML.

The performance of the CML offers much hope for this approach. Memory Cycles Per Instruction is used to compare a direct-mapped cache, a 2-way associative cache, and a direct-mapped cache with a CML. It most cases, the CML improves the

performance of the direct-mapped cache. In a few cases, the overhead of the CML actually decreases the performance of the CML. In no case does the CML perform any better than the 2-way associative cache.

2.2 Identify Cache Conflicts on Standard Hardware

In later work, the ideas of the CML were applied to standard hardware by using the system's TLB. [BCL2] By monitoring the TLB, either through page protection mechanisms or through the system's software-controlled TLB, the operating system attempts to locate pages that should be recolored. Many different software policies, divided into active and periodic policies, were tested. Active policies try to maintain the invariant that each page in the TLB must map to a different cache block. Periodic policies sample the TLB intermittently and remap pages according to some predetermined policy.

An example of a periodic policy is the "Snapshot-Delay" policy. This policy periodically invalidates the TLB and then waits for some amount of time, recording the TLB misses. Then to confirm that the pages that were suspected of conflicting should really be remapped, another snapshot is taken after a little while. Only those pages still conflicting are remapped.

None of these programs showed promise in simulation. The active policies exhibited too much TLB monitoring overhead, while the periodic policies did not detect the misses quickly enough to reduce the MCPI significantly. The problem that these approaches suffered from is that they used the TLB to predict cache misses. Merely seeing two pages in the TLB that map to the same cache block is not sufficient to determine that the pages are conflicting. The TLB is a cache itself and holds information about pages used long before. Another important factor in the unimpressive results of this technique is the high overhead involved in monitoring the TLB.

One policy tested in this paper showed noticeably better results than these other algorithms. The algorithm assumed the presence of a cache miss counter register on

the hardware. The register was used to identify periods of high cache miss activity, and during these surges, the TLB was sampled. While still not performing nearly as well as the CML, this hardware/software hybrid outperformed all the other software-only policies.

3 System Design

Looking to combine the lower overhead of the CML's dedicated hardware approach, with the decreased cost and informational benefits of compiling statistics in the OS, we felt that a system that allowed the operating system direct access to cache miss information would be ideal. The simplest implementation would be a latch that saved the conflicting page numbers on every cache miss. Such a system was patented and eventually implemented by DEC. [S95] The critical timing of such an approach is well within the time available during a cache miss. Also, the cost of such a device is insignificant when compared to the CML or the associativity hardware on a higher associativity cache. Ideally, this idea would be expanded to include a buffer of the last n cache misses, where n is the optimal sampling length. With such a buffer, the OS can stop at any time and quickly view the last n cache misses. With the simple one latch approach, to sample n misses, the operating system would have to be interrupted n times, a possibly prohibitive overhead. Additionally, a cache miss counter would be present that could be set to interrupt the OS after a certain number of cache misses. These two minor and inexpensive hardware modifications are sufficient for efficient sampling of the cache.

4 Sampling Techniques and Methodology

The work on sampling is divided into two sets of experiments. The first set of experiments, the topic of Section 5, tries to determine how well sampling does at identifying

pages whose conflicts are contributing to the largest portion of the program's cache misses. For example, how well can different sampling techniques identify the smallest set of conflicting pages that together account 90% of the program's cache misses? The effects of different sampling variables were monitored over the course of these experiments. The second set of experiments uses the results from the first set and tries to estimate the theoretical limits of improvement that are possible by employing these techniques.

4.1 Terms and Definitions

Before looking at these experiments, we need to define some terms. The standard metrics used in most of the experiments are "Prediction Rate" and "Misprediction Rate." In any cache sampling experiment, there are three critical numbers. The first is RealConflictsDetected, the number of correctly identified conflicts. RealConflictsMissed are conflicts that should have been identified but were missed. SpuriousConflictsDetected are conflicts that were identified as targeted conflicts but should not have been. The Prediction Rate is simply (RealConflictsDetected) / (RealConflictsDetected + RealConflictsMissed). The "Misprediction Rate" is (SpuriousConflictsDetected) / (RealConflictsDetected).

Many of the earlier experiments are run in relative terms. An experiment might attempt to predict misses "at the 80% level". To find the conflicts at the 80% level, one could run the program and collect the set of all conflicting pages paired with their total number of misses. These (conflict, frequency) pairs could then be sorted in decreasing order by their miss frequencies. Let totalMisses be the total number of conflicts in the run (the sum of each pair's frequency), and let targetMisses be 80% of totalMisses. To find the conflicts at the 80% level, one steps through the sorted list until the sum of all the frequencies seen to that point equals or exceeds targetMisses. The conflicts at the 80% level is the set of misses with at least a frequency of that last conflict pair. As the percentage level increases, the conflict cutoffs monotonically

	Benchmark		
Relative Cutoff	*Tomcatv*	*Swim*	*Fpppp*
95%	565,500	100,356	10,381
90%	565,500	101,263	10,381
85%	565,500	102,098	10,381
80%	565,500	102,200	10,381
75%	574,500	102,500	10,381
70%	574,500	151,828	10,381
60%	574,500	153,500	10,381
50%	574,500	204,300	18,262

Table 1: Absolute level of conflicts that correspond to the relative levels for each benchmark. This means, for example, that if all data conflicting at least 101263 times were eliminated from Swim, then there would be a 90% reduction of direct-mapped conflict misses for that program.

decrease. Obviously, the 100% level, requires all conflicts, so the conflict cutoff will be 1. The conflict miss cutoffs for various benchmark and percentage levels are shown in Table 1.

At this point it is necessary to discuss the "Prediction Cache." The Prediction Cache is the simulated cache that is the result of the sampled conflicts. The sampled conflicts are inserted into a simulated direct-mapped cache, which is cleared after every sampling period. The simulated cache has the same parameters as the hardware cache. The OS keeps track of conflicts between pages, and it makes decisions about which conflicts need remapping. This information about the sampled conflicts will be referred to as the Prediction Cache.

The "Prediction Cache Cutoff" is the number of conflicts that are necessary in

the Prediction Cache before the conflicting pages are considered to be conflicts falling into the desired category. This cutoff can be either an absolute number or a relative level, described above, as is the case in the earlier experiments. The "Sample Length" is the number of consecuctive conflicts recorded by the OS. This quantity is closely related to the "Sampling Ratio," which is the rate at which samples are taken. The two numbers together define the sampling pattern. For example a Sampling Length of 10 at a 50:1 Sampling Rate would mean that the OS records 10 cache conflicts then ignores the next 500 misses then records the next 10 cache conflicts and so on.

4.2 Methods

All experiments are run on three SPEC95 floating-point benchmark programs. All the benchmarks were written in fortran and are described in detail by SPEC. Tomcatv is a vectorized mesh generation program. Swim solves a system of shallow water equations using finite difference approximation on a 512 by 512 grid. Fpppp performs two electron integral derivations which occur in GausianXX series of programs. The runs are of varying lengths with Tomcatv running nearly 10 times as long as the other two benchmarks. Fpppp has no capacity misses whereas the other two benchmarks have significant capacity misses. Tomcatv performs much better on a 2-way associative cache than on a direct-mapped coche, but Fpppp and Swim have very similar direct-mapped cache and 2-way associative cache performance but perform much better on fully associative caches. Table 2 presents summary statistics for the various benchmarks.

It is now possible to describe the sampling experiments in greater detail. In the first set of experiments, three factors are investigated: the correlation between the actual cache misses and the Prediction Cache, the effect of Sample Length on accuracy, and the effect of Sampling Rate on accuracy. Experiment 1 attempts to predict the actual conflicts at the 80% level. Different relative Prediction Cache Cutoffs and Sample Lengths are tested and Prediction and Misprediction Rates are recorded.

		Non-Conflict Misses		Fully Associative Cache	
Benchmark	*Total References*	*Cumpulsory*	*Capacity*	*Total Misses*	*Miss Rate*
Tomcatv	66,830,110,156	14,455	32,068,821	32,083,276	0.048007%
Swim	8,557,192,275	14,485	4,112,278	4,126,763	0.048226%
Fpppp	7,460,196,512	213	0	213	0.0%

	2-way Associative Cache			Direct-Mapped Cache		
Benchmark	*Total Misses*	*Miss Rate*	*Conflict Misses*	*Total Misses*	*Miss Rate*	*Conflict Misses*
Tomcatv	72,235,279	0.108088%	40,152,003	1,434,342,851	2.146252%	1,402,259,575
Swim	36,644,270	0.428228%	32,517,507	41,719,487	0.487537%	37,592,724
Fpppp	1,530,548	0.020516%	1,530,335	1,798,451	0.024107%	1,798,238

Table 2: Summary statistics for the three benchmarks used in the experiments.

The goal is to see hom well the actual cache at the 80% level is predicted by the prediction cache at relative cutoffs of about 80%. If good correlation is found between the prediction cache and the actual cache at the 80% levels than the prediction cache is doing a good job at finding high frequency misses. This experiment is repeated at the 90% prediction level. Experiment 2 investigates the effects on accuracy of different sampling ratios. The Sample Length is fixed at 100 and three different sampling ratios are tested (50 to 1, 100 to 1, and 500 to 1).

In the second set of experiments, the focus turns away from testing the predictive ability of sampling and toward identifying conflicts for remapping. In experiment 3, various sampling patterns are tested in identifying absolute thresholds. Absolute thresholds are conflicts that conflict more than some fixed number of times. The accuracy using absolute thresholds is important to remapping schemes because it is necessary to make decisions based on absolute levels sinces relative levels are not known till execution completes. In experiment 4,this information is used to pick an effective sampling strategy and a remapping simulation is run. An upper limit for the performance of our system is found and compared to the conventional cache architectures.

All simulations were done on DEC Alpha workstations. The object code was instrumented with ATOM. Every memory reference was instrumented by ATOM,

System Cache Assumptions	
Cache Size	256K
Block Size	1K
Indexing	Virtual

Table 3: Parameters of simulated system cache.

resulting in a call to a user-defined function. [S94] [DEC93] [DEC95] This function both traced the actual system cache and performed the appropriate sampling routine. Table 3 shows the parameters of the simulated system cache. These parameters follow the earlier work of [BCL1]. After the trace completed, the predicted cache results were compared to the targeted conflicts in the simulated system cache.

5 Testing Sampling Accuracy

5.1 Experiment 1

The first set of experiments attempts to correlate the Prediction Cache with the actual simulated system cache. Relative cutoffs are used in order to test the consistency of the Prediction Cache relative the actual cache. By comparing the accuracy of the various relative Prediction Cache Cutoffs against the relative cutoffs in the simulated system cache, it is possible to see how closely the simulation mirrors reality and make generalizations about the under or over-representation of the most significant cache conflicts in the prediction cache. Also the relative effectiveness of the various Sample Lengths can be compared.

Figures 1 - 3 show the accuracy of predicting at the 80% level. For both Tomcatv and Swim, the Prediction Cache Cutoffs of between 70 and 80% do the best job of estimating. Fpppp has perfect performance between the 65 and 90% Prediction

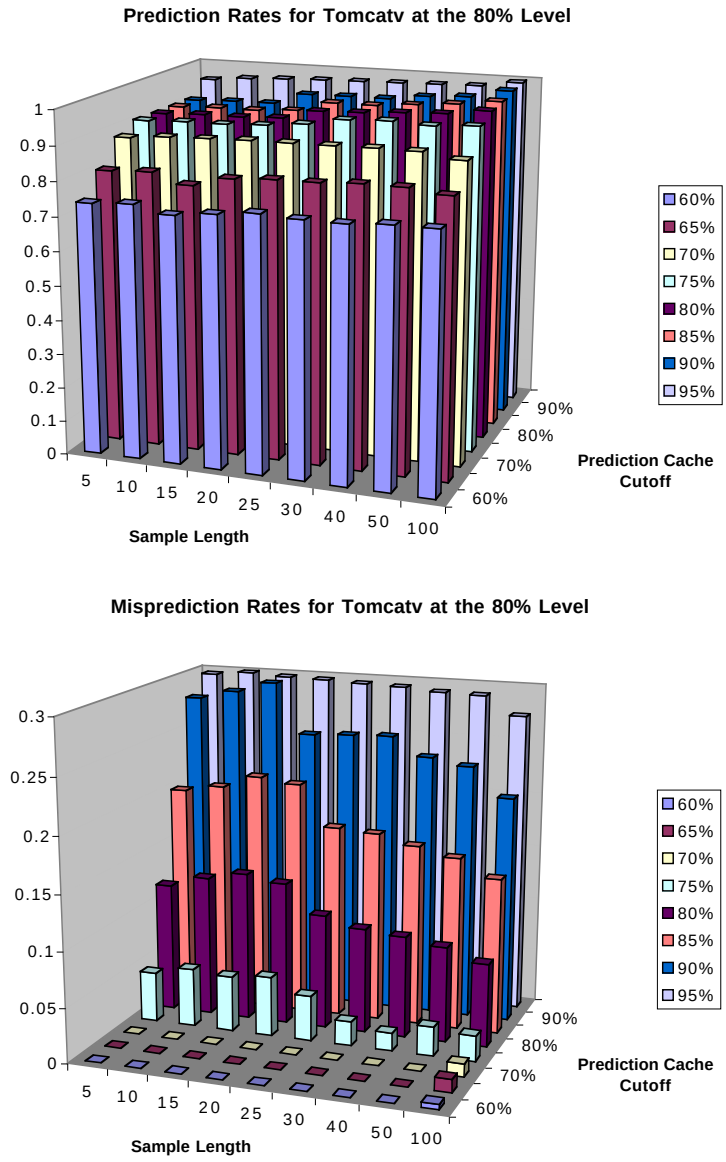


Figure 1: Experiment 1 - Prediction and Misprediction Rates for various Sample Lengths and Relative Prediction Cache Cutoffs for conflicts at the 80% Level in Tomcatv with Sampling Ratio of 50 to 1

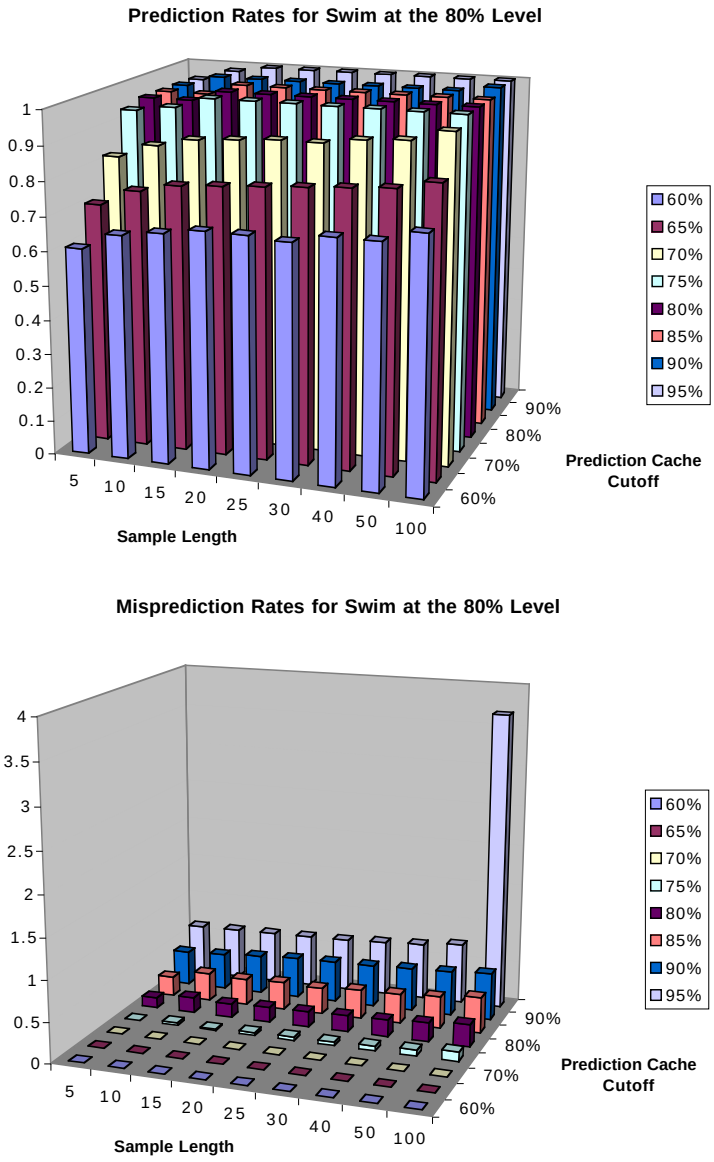


Figure 2: Experiment 1 - Prediction and Misprediction Rates for various Sample Lengths and Relative Prediction Cache Cutoffs for conflicts at the 80% Level in Swim with Sampling Ratio of 50 to 1

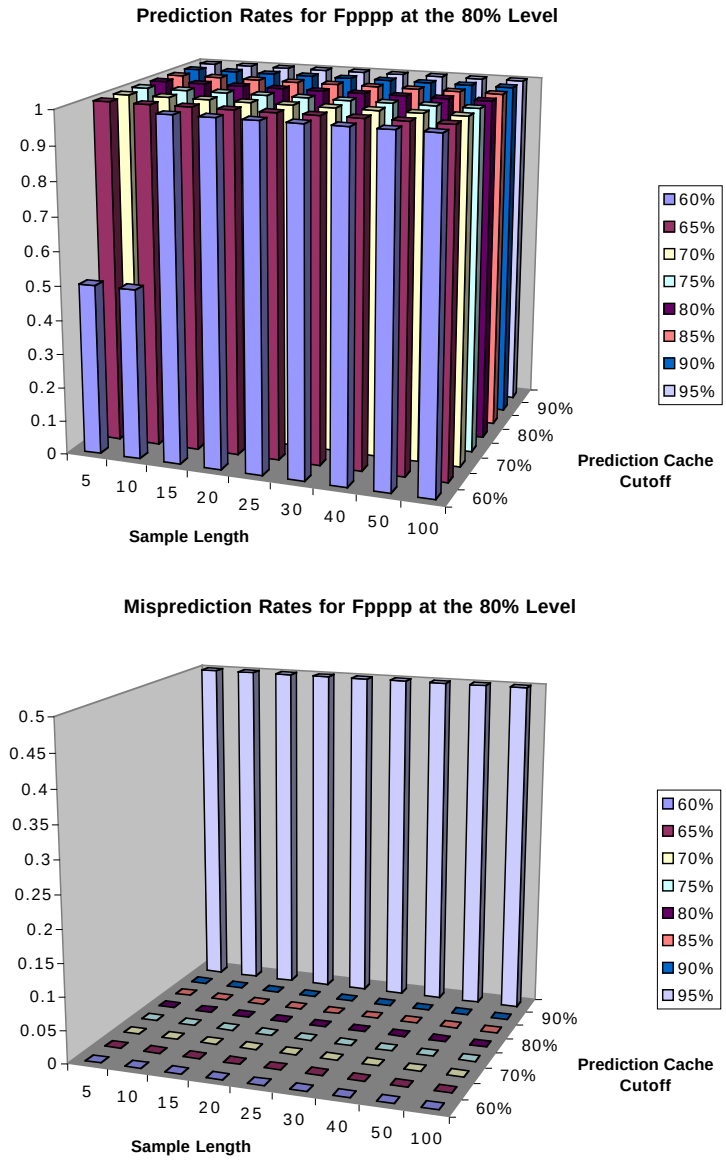


Figure 3: Experiment 1 - Prediction and Misprediction Rates for various Sample Lengths and Relative Prediction Cache Cutoffs for conflicts at the 80% Level in Fpppp with Sampling Ratio of 50 to 1

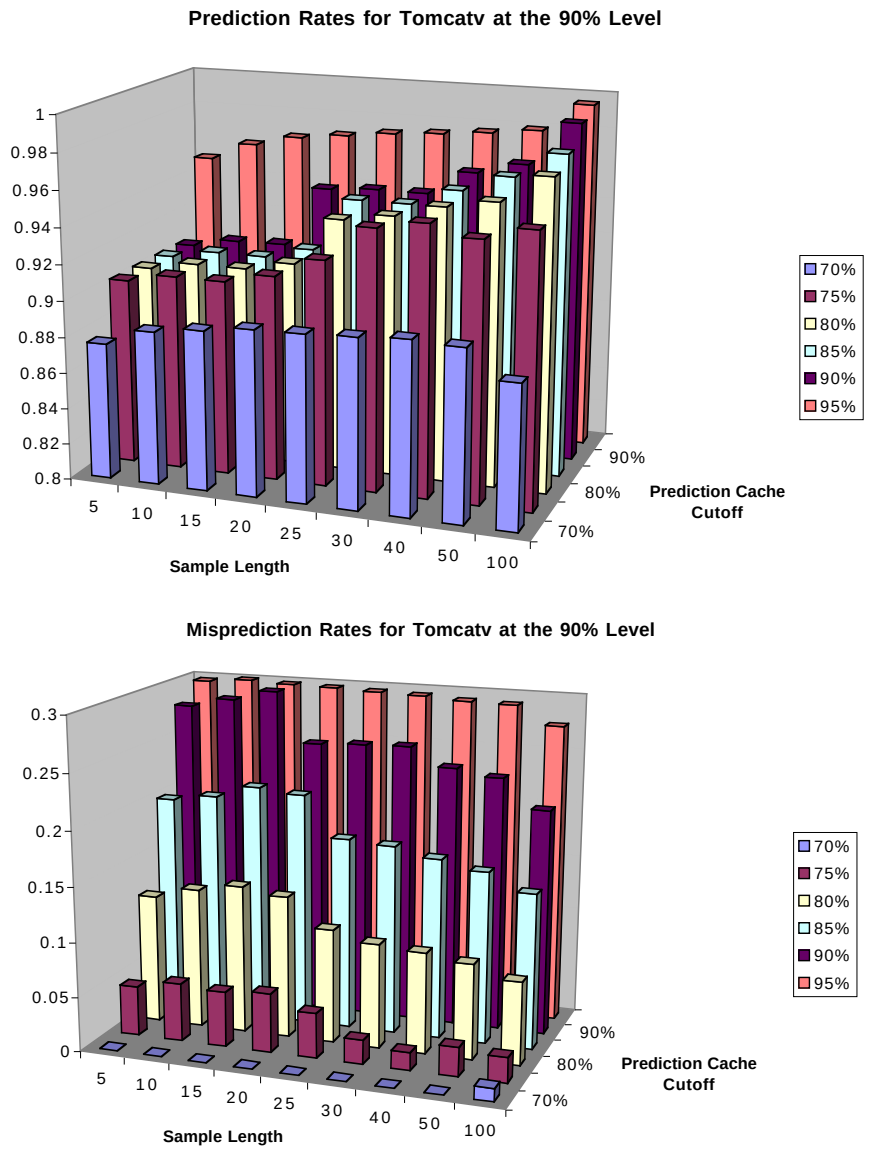


Figure 4: Experiment 1 - Prediction and Misprediction Rates for various Sample Lengths and Relative Prediction Cache Cutoffs for conflicts at the 90% Level in Tomcatv with Sampling Ratio of 50 to 1

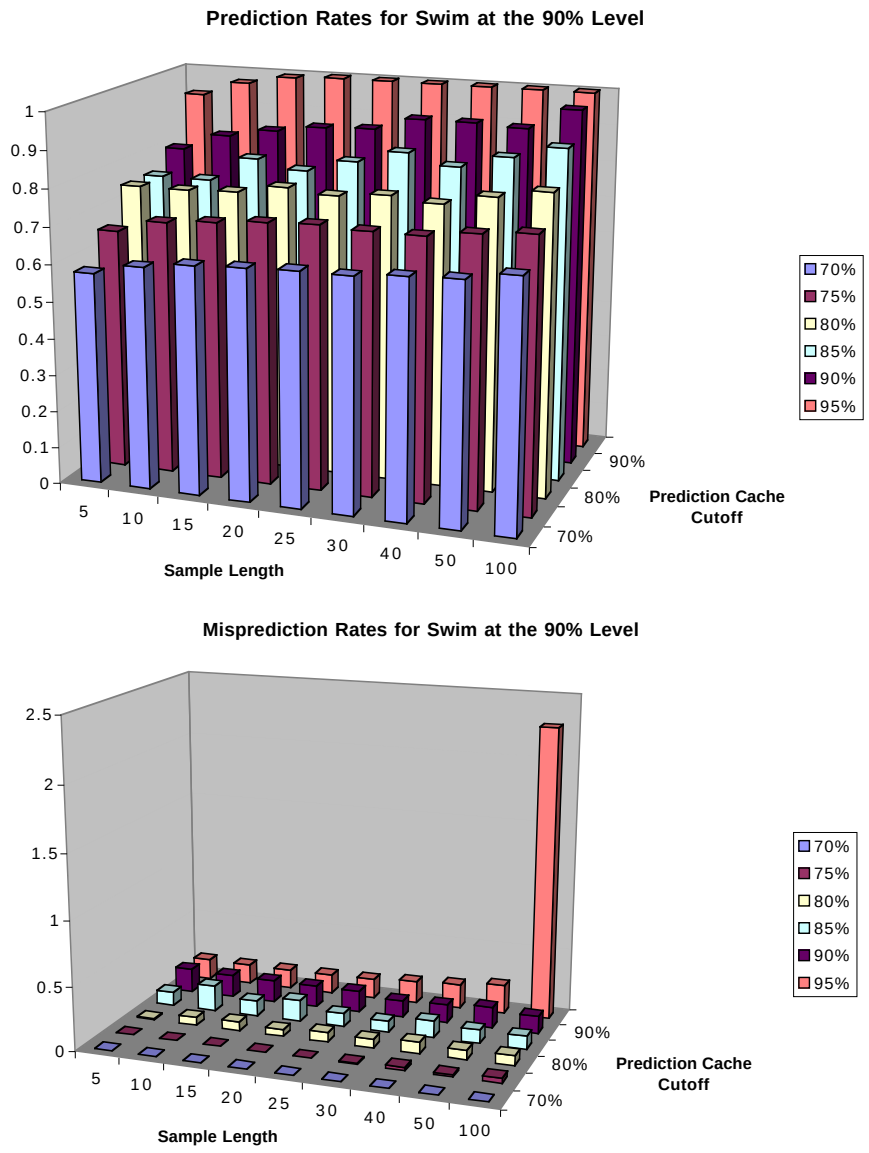


Figure 5: Experiment 1 - Prediction and Misprediction Rates for various Sample Lengths and Relative Prediction Cache Cutoffs for conflicts at the 90% Level in Swim with Sampling Ratio of 50 to 1

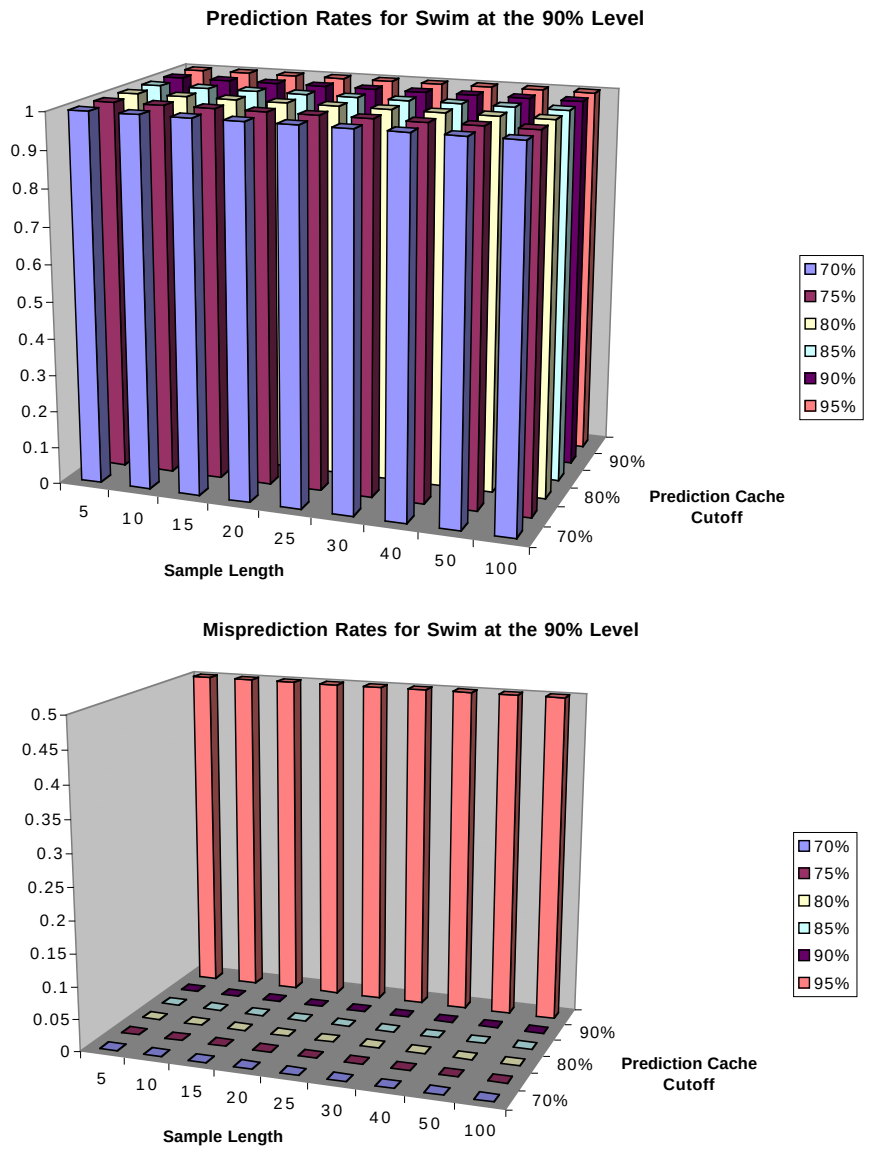


Figure 6: Experiment 1 - Prediction and Misprediction Rates for various Sample Lengths and Relative Prediction Cache Cutoffs for conflicts at the 90% Level in Fpppp with Sampling Ratio of 50 to 1

Cache Cutoff levels. These early results show that the Prediction Cache is doing a very good job of paralleling the system cache with very high frequency cache misses.

As seen in Figures 4 - 6, the results at the 90% level are less comprehensive. For the Tomcatv benchmark, the 65-80% Prediction Cache Cutoff levels seem to be the best fit. Swim at the 90% level is estimated very closely in the 85-95% level. Finally, Fpppp is once again fit perfectly with all cutoffs between 70% and 90%. These results still show good fits, but there is more volatility at this higher level. Fortunately, it is unlikely that we would want to estimate at the 90% because we will want to remap at much lower thresholds.

Except for the spikes in the Misprediction Rate at the 95% level in Figures 2 and 5, the longer sample lengths of 50 and 100 tend to show higher Prediction Rates and lower Misprediction Rates across the board.

5.2 Experiment 2

The second set of experiments locks the sampling length in at 100 and varies the sampling rate. See Figures 7 - 12. The results are mixed. While Tomcatv performs almost identically at all three sampling rates, Swim's Misprediction rates are very responsive to changes in the sampling rate at the 95% level. It is also interesting to note that both the Prediction Rate and especially the Misprediction Rates are much more responsive to changes in the Prediction Cache Cutoff at the 50 to 1 sampling rate than at the other rates.

This first set of experiments shows that relatively infrequent sampling can yield highly accurate results. Also, longer sample lengths performed almost universally better in both higher Prediction Rates and lower Misprediction Rates. Finally, some programs appear to be very responsive to changes in sampling rates while others do not. This behavior is likely the result of differences in data-access patterns, but it merits closer inspection.

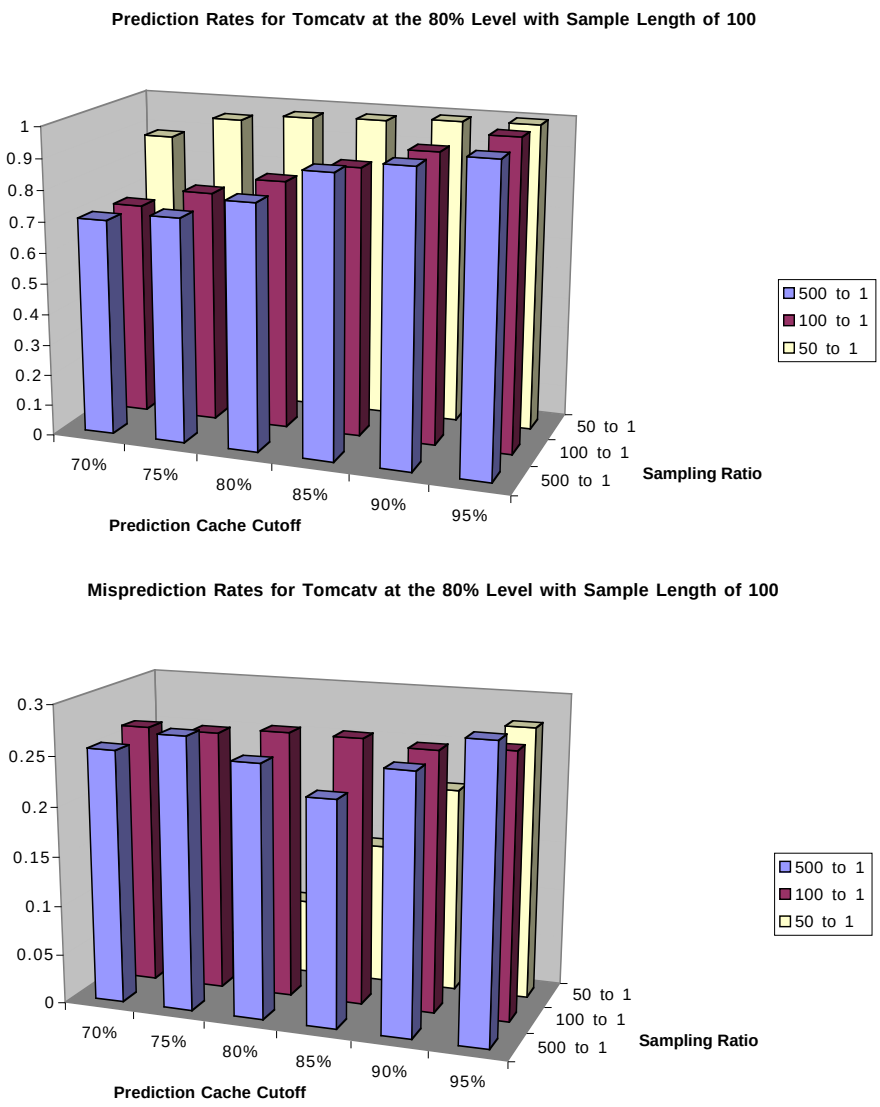


Figure 7: Experiment 2 - Prediction and Misprediction Rates for various Sampling Ratios and Relative Prediction Cache Cutoffs for conflicts at the 80% Level in Tomcatv and fixed Sample Length of 100

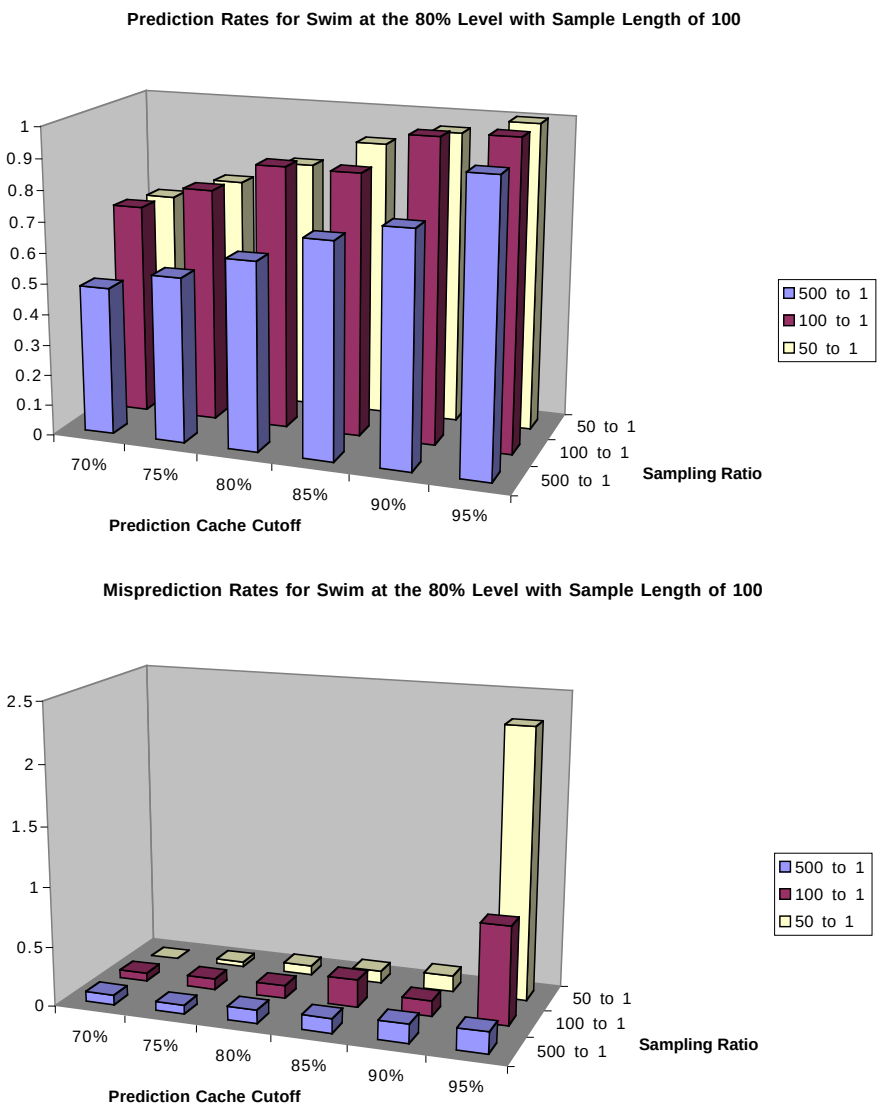


Figure 8: Experiment 2 - Prediction and Misprediction Rates for various Sampling Ratios and Relative Prediction Cache Cutoffs for conflicts at the 80% Level in Swim and fixed Sample Length of 100

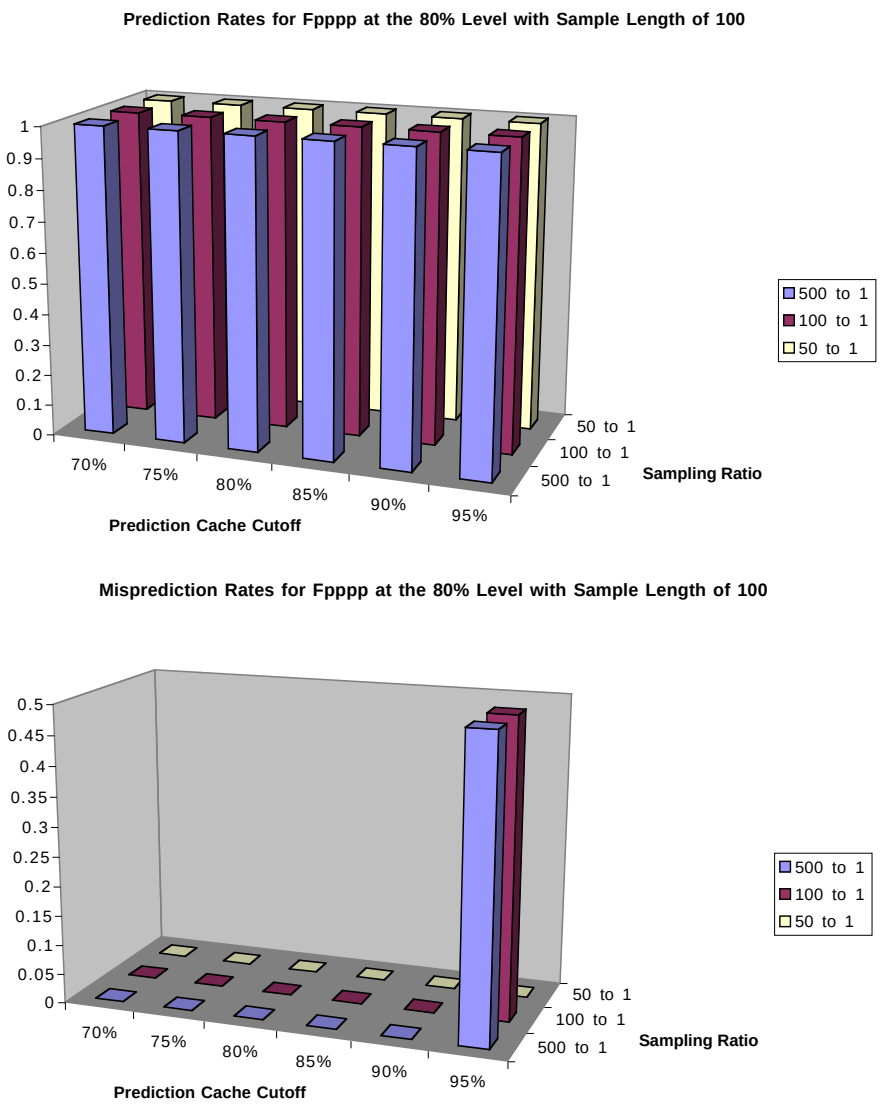
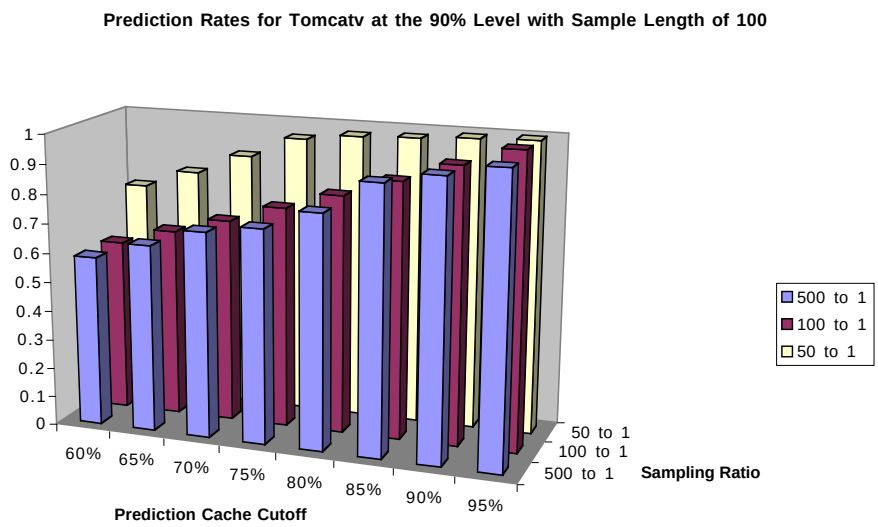


Figure 9: Experiment 2 - Prediction and Misprediction Rates for various Sampling Ratios and Relative Prediction Cache Cutoffs for conflicts at the 80% Level in Fpppp and fixed Sample Length of 100



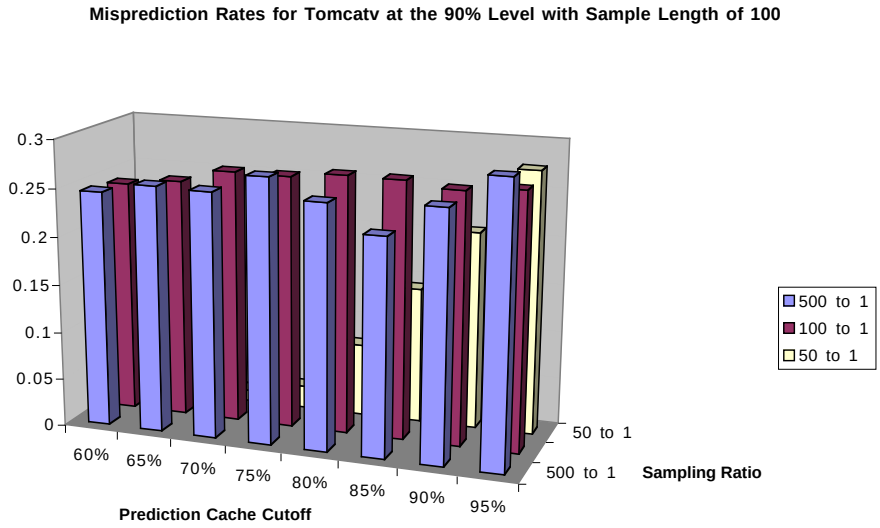


Figure 10: Experiment 2 - Prediction and Misprediction Rates for various Sampling Ratios and Relative Prediction Cache Cutoffs for conflicts at the 90% Level in Tomcatv and fixed Sample Length of 100

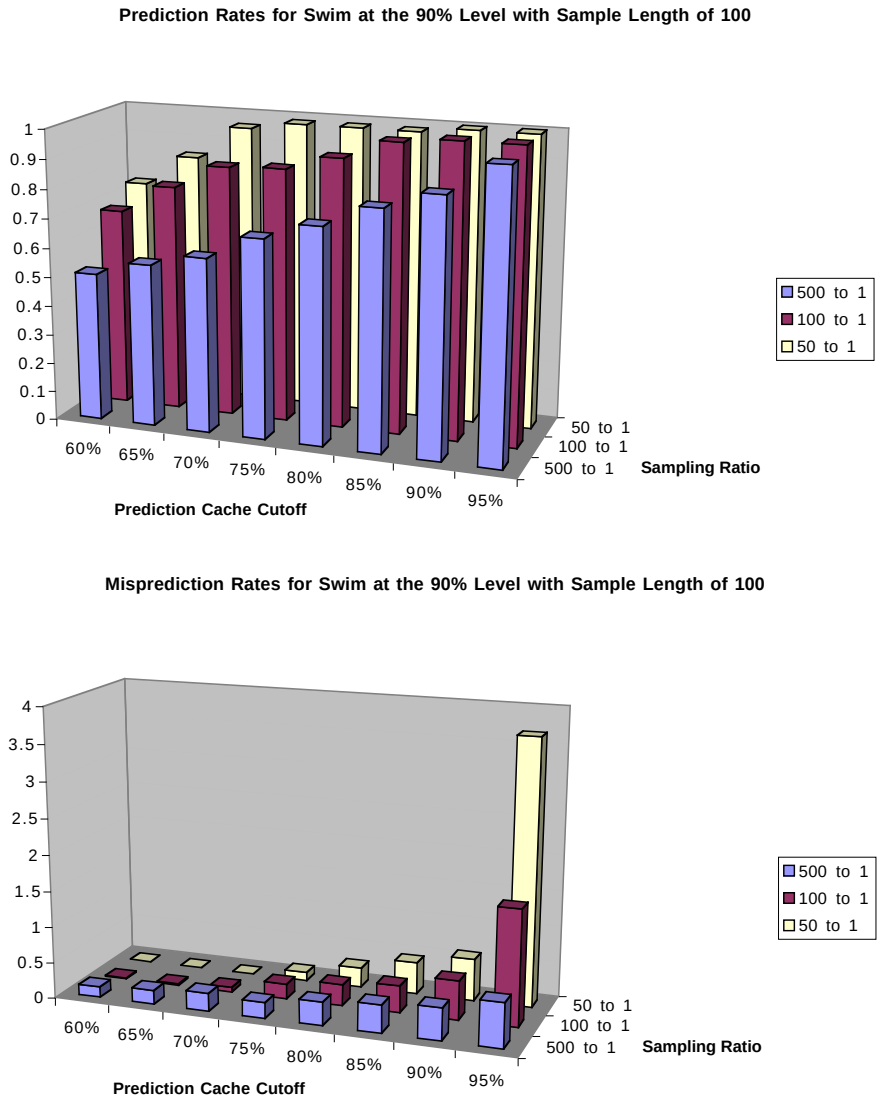
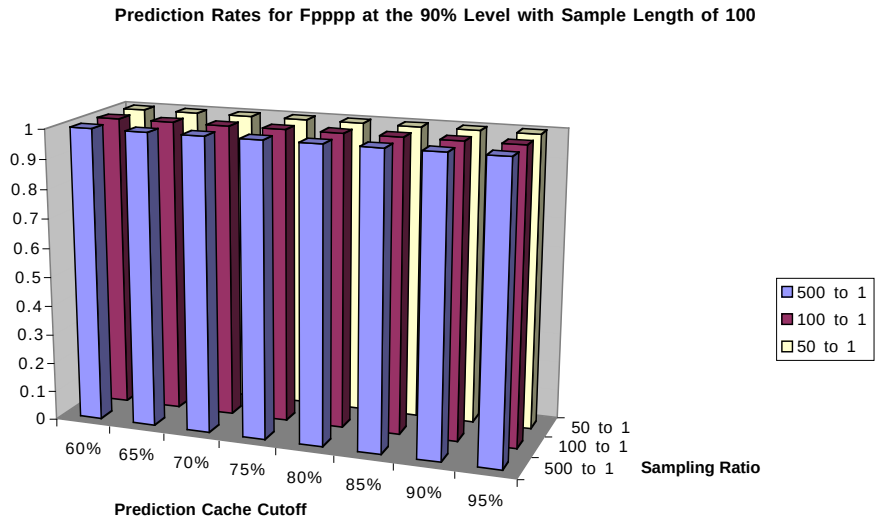


Figure 11: Experiment 2 - Prediction and Misprediction Rates for various Sampling Ratios and Relative Prediction Cache Cutoffs for conflicts at the 90% Level in Swim and fixed Sample Length of 100



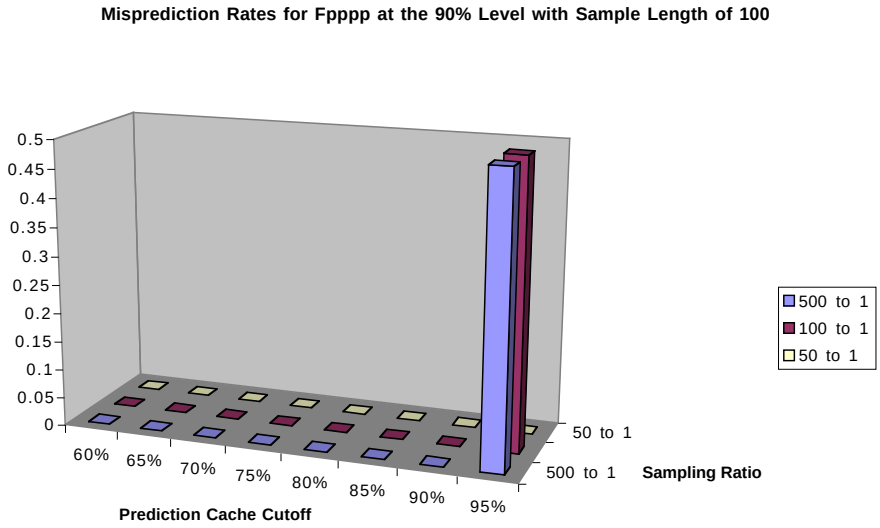


Figure 12: Experiment 2 - Prediction and Misprediction Rates for various Sampling Ratios and Relative Prediction Cache Cutoffs for conflicts at the 90% Level in Fpppp and fixed Sample Length of 100

6 Performance Estimates

6.1 Experiment 3

In order to estimate potential speedups, our attention must first be turned to finding good absolute prediction results. Relative results do little good until a program has completed running, too late for page remapping to make a difference. The first experiment shows the effectiveness of different Prediction Cache Cutoffs, Sample Lengths, and Sampling Rates to Prediction and Misprediction Rates for actual conflict cutoff levels of 100, 500, 1000, 5000, and 10000. See Figures 13 - 27. Each chart is useful by itself, but when used with the other charts it provides additional insight into the Prediction and Misprediction Rates of the other thresholds.

Prediction Cache Cutoffs of 10 almost always lead to a strong possibility of very high Misprediction Rates even at the 100 miss threshold. This relationship implies that such a low cutoff may cause us to remap many conflicts that do not even conflict 100 times. On the other hand, the difference in Prediction and Misprediction rates for cutoffs of 50 and 100 are marginal at best. So choosing a threshold of 100 might cause us to wait too long to remap, thereby lessening the possible gains. The longer 100 sample length once again yields significantly higher Prediction Rates at the expense of only marginally higher Misprediction Rates at the higher thresholds. This is particularly true with the 50 to 1 Sampling Ratio. The combination of these factors led us to choose a Sample Length of 100, a Sampling Ratio of 50 to 1, and a Prediction Cache Cutoff of 50 as the parameters for the final experiment.

6.2 Experiment 4

The final part of this thesis, by tracing each of the programs one additional time, uses the above parameters to estimate a maximum speedup that can be realized. This time, as soon as the Prediction Cache finds two addresses to conflict more than 50 times, the

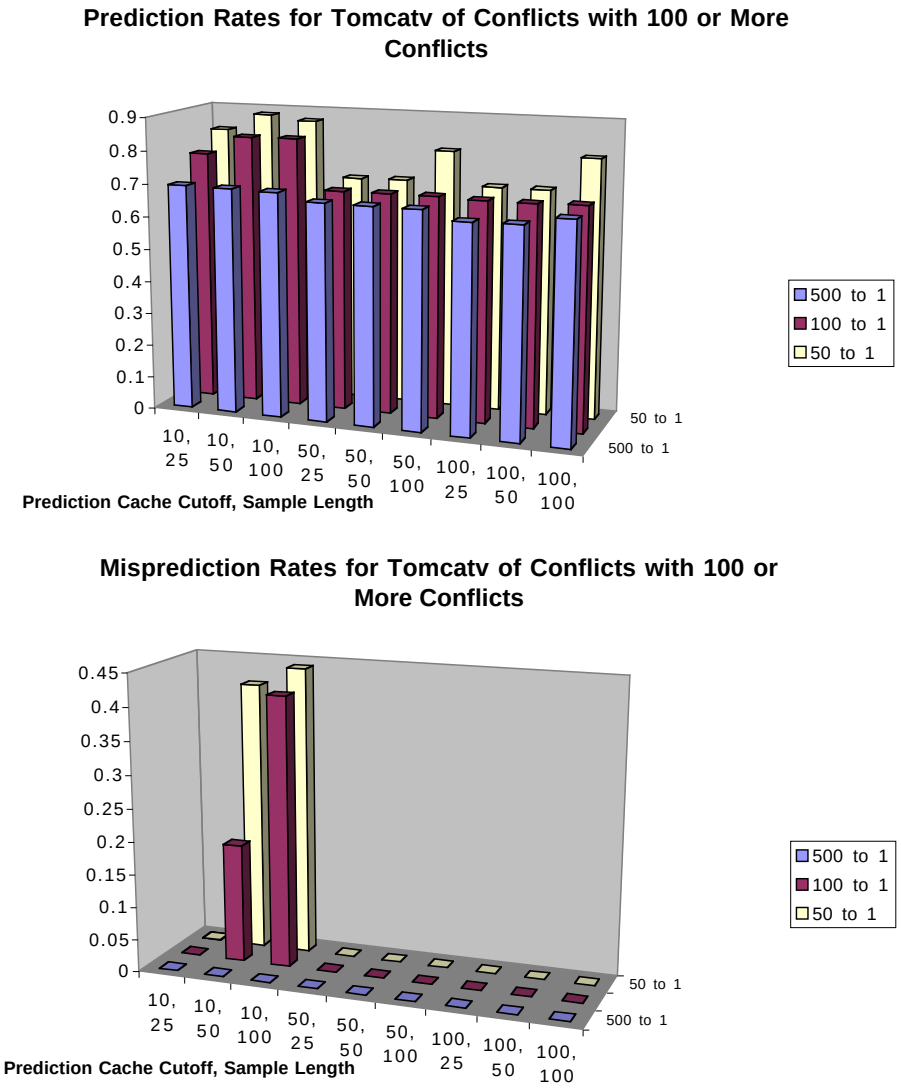


Figure 13: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 100 or more in Tomcatv

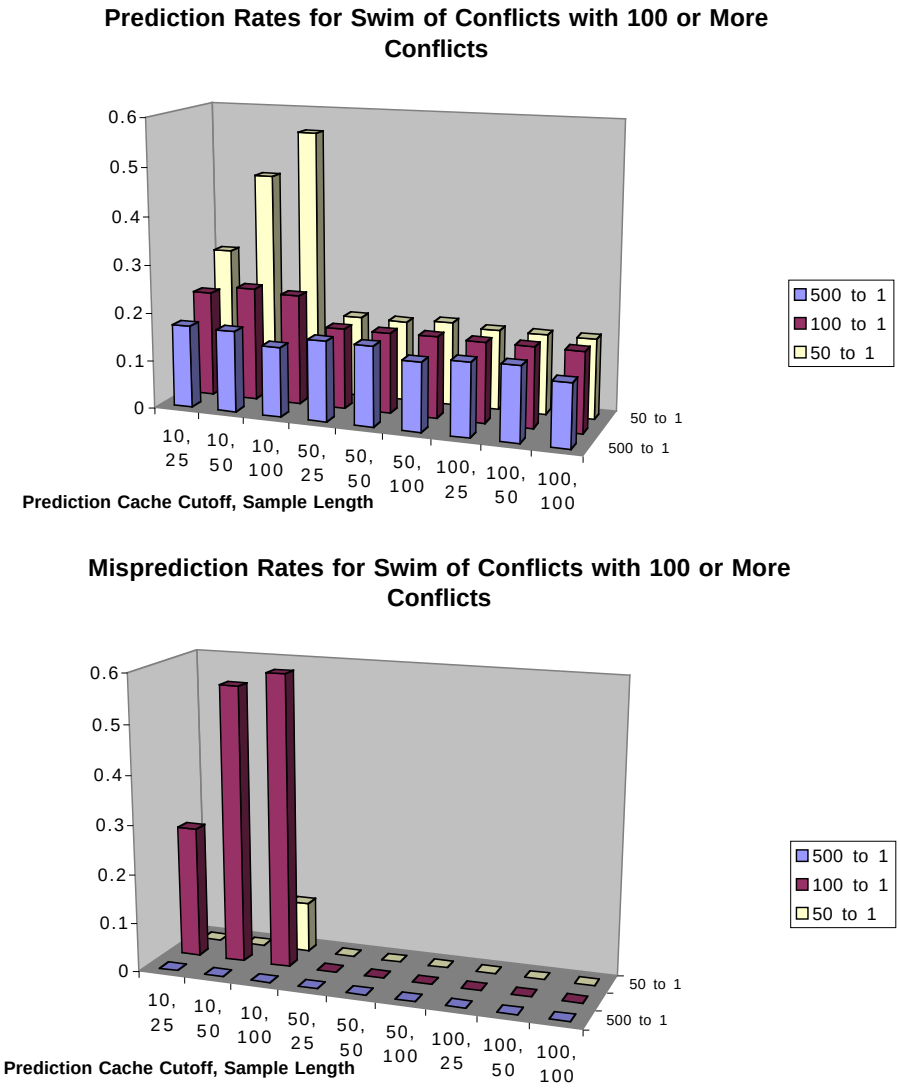


Figure 14: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 100 or more in Swim

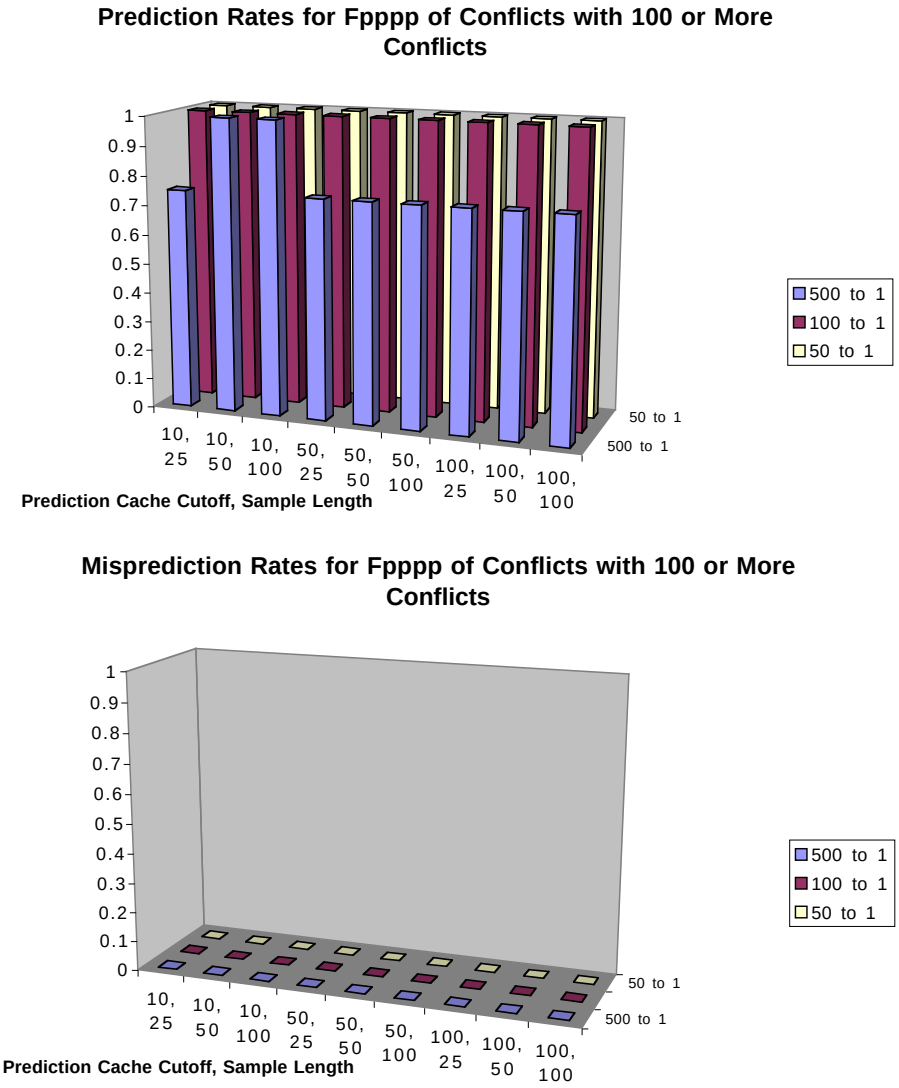


Figure 15: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 100 or more in Fpppp

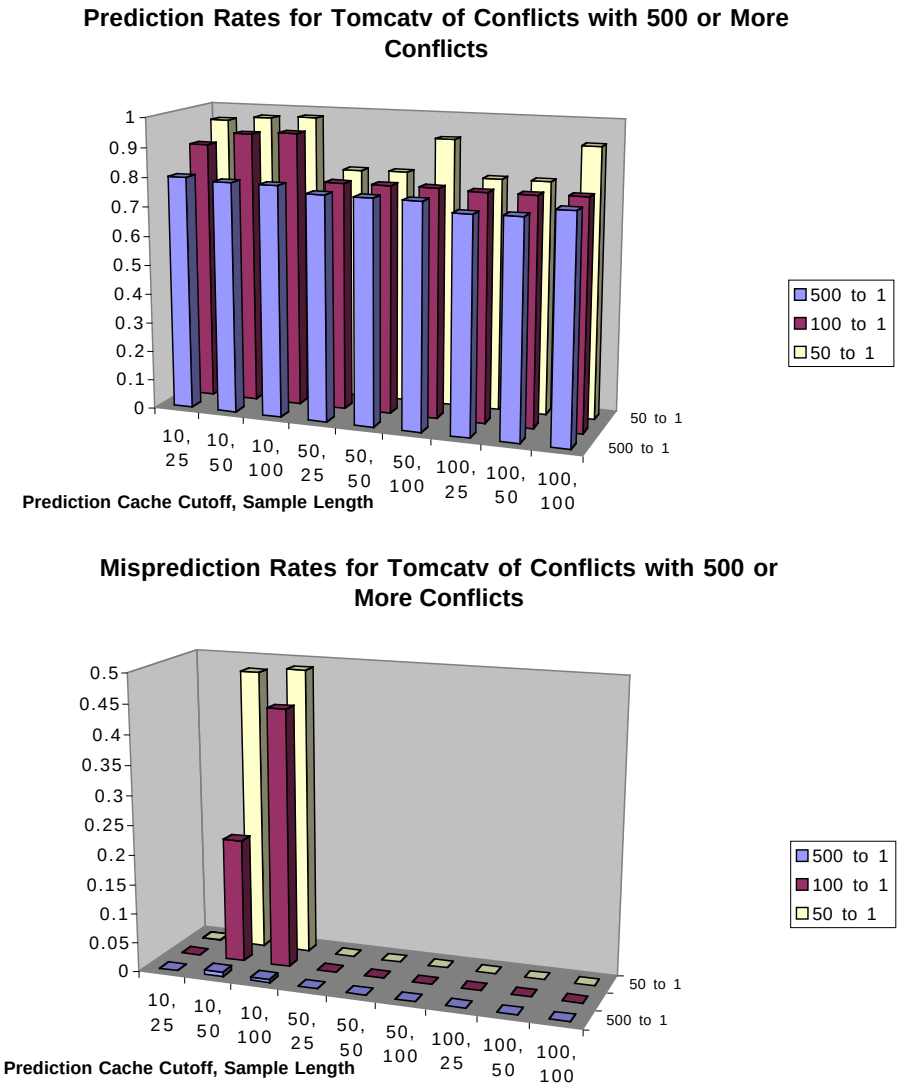


Figure 16: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 500 or more in Tomcatv

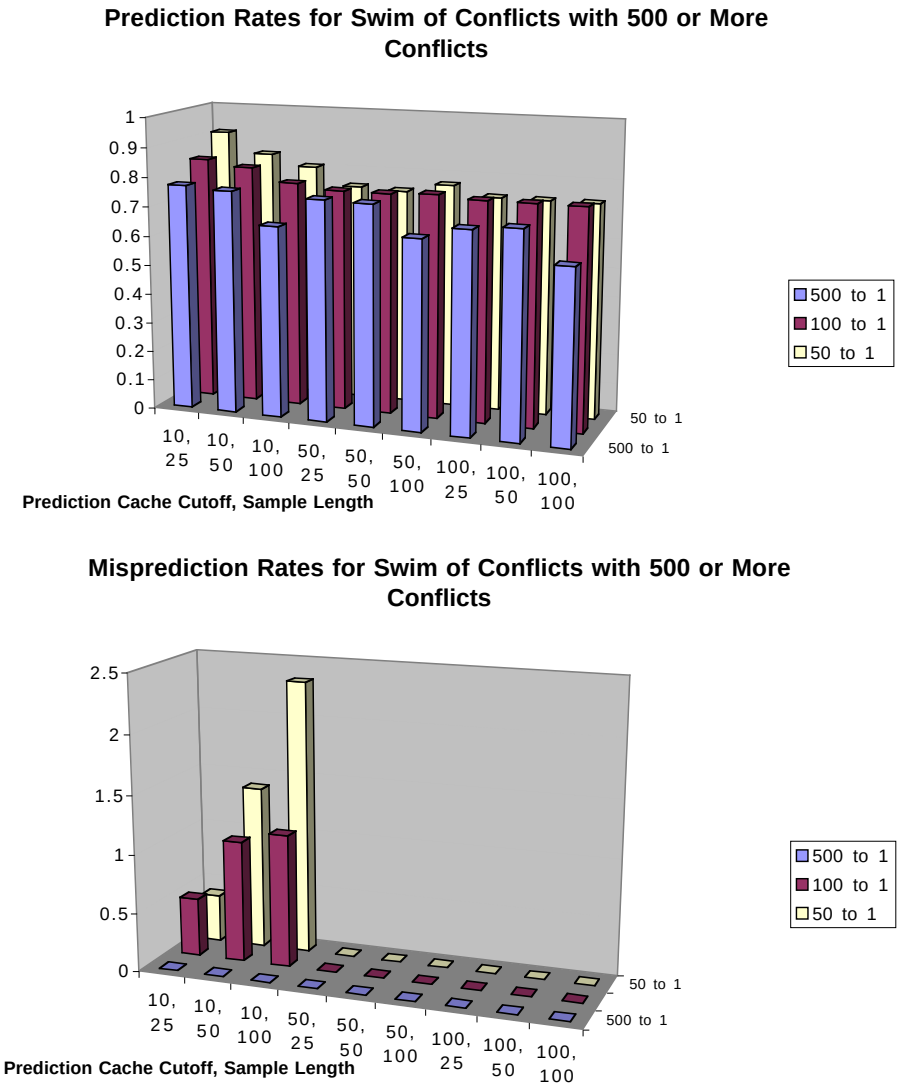
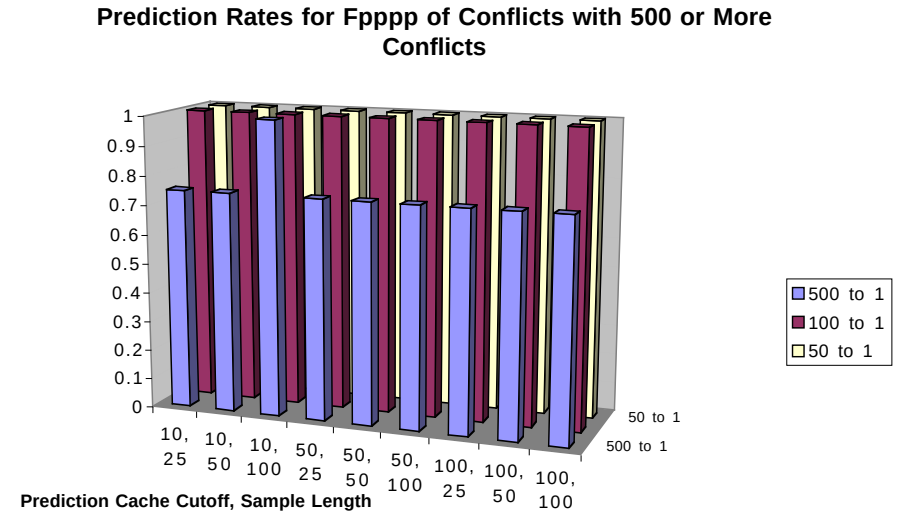


Figure 17: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 500 or more in Swim



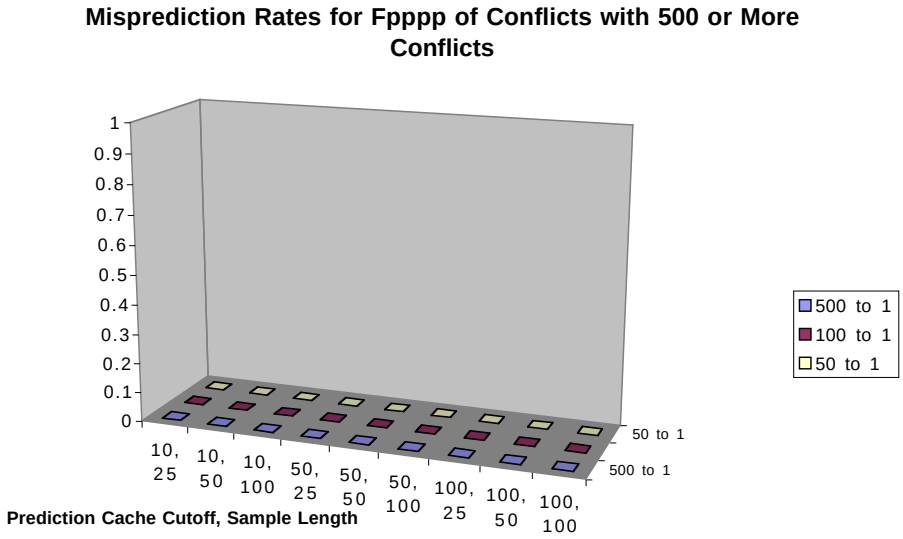


Figure 18: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 500 or more in Fpppp

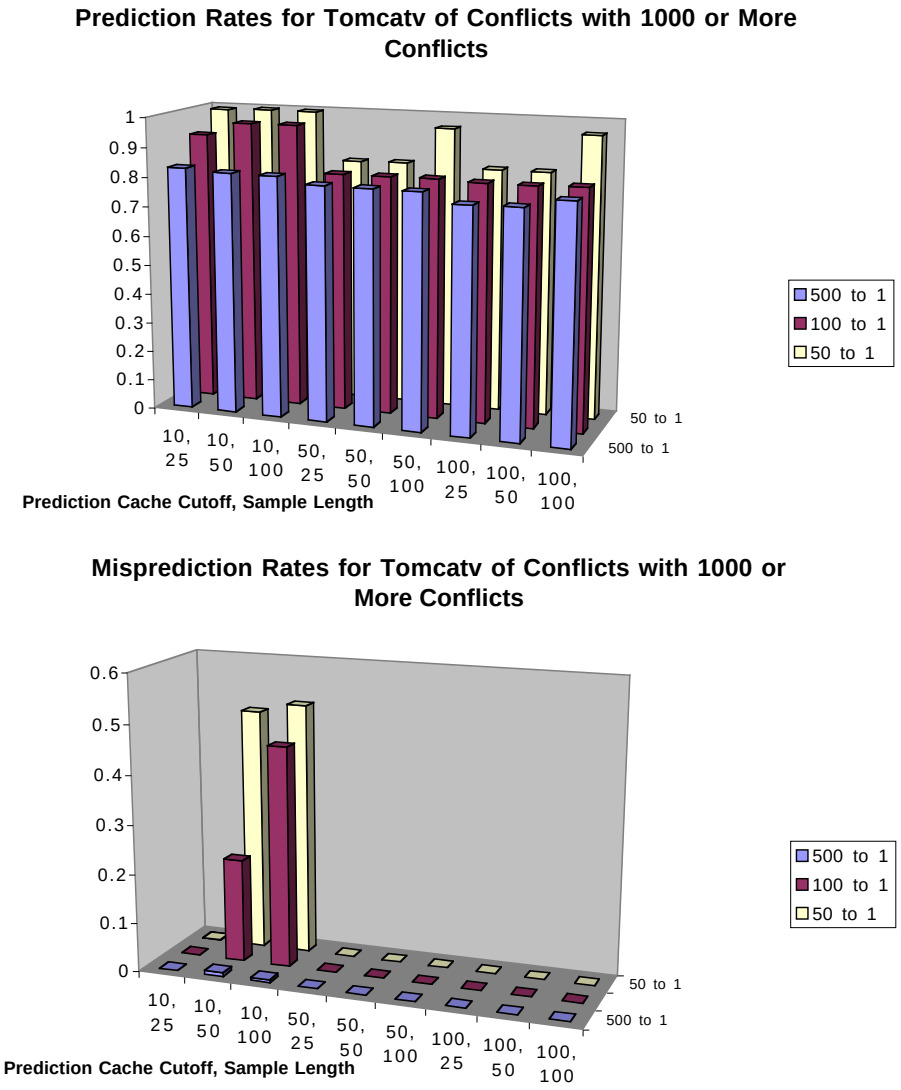


Figure 19: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 1000 or more in Tomcatv

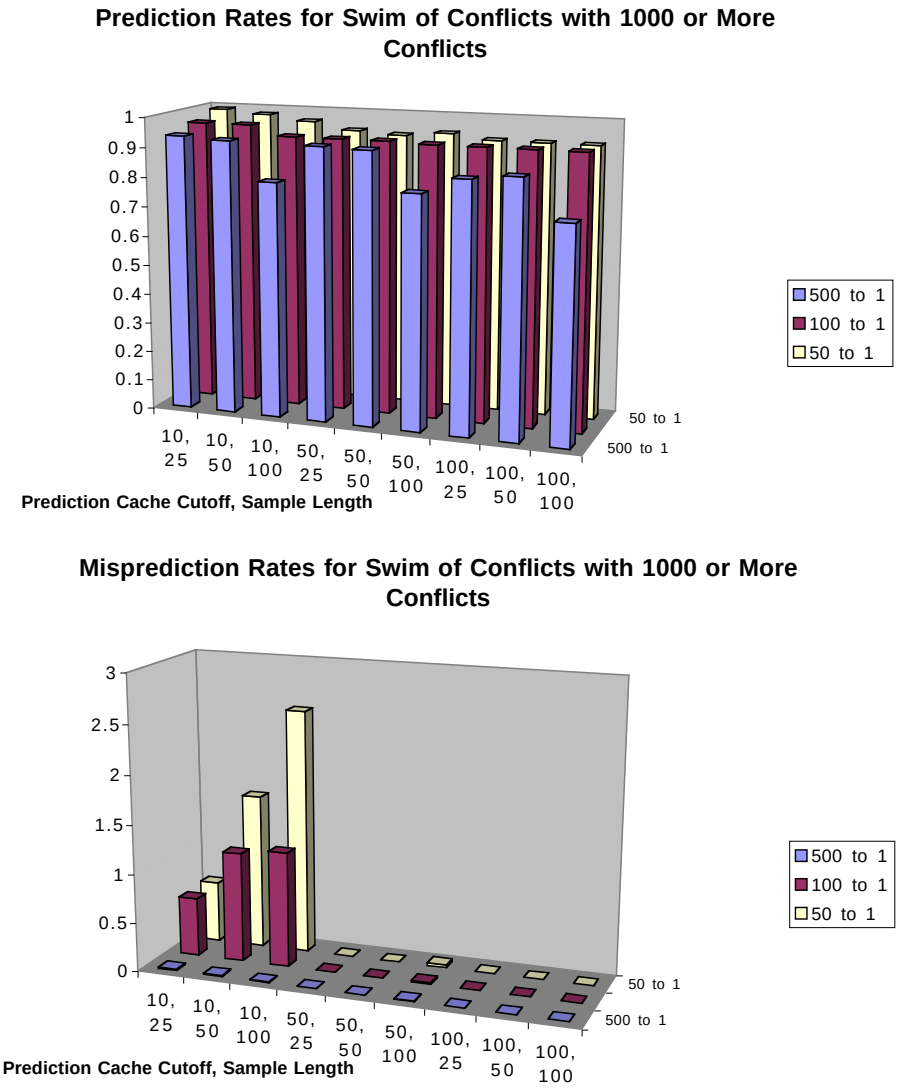


Figure 20: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 1000 or more in Swim

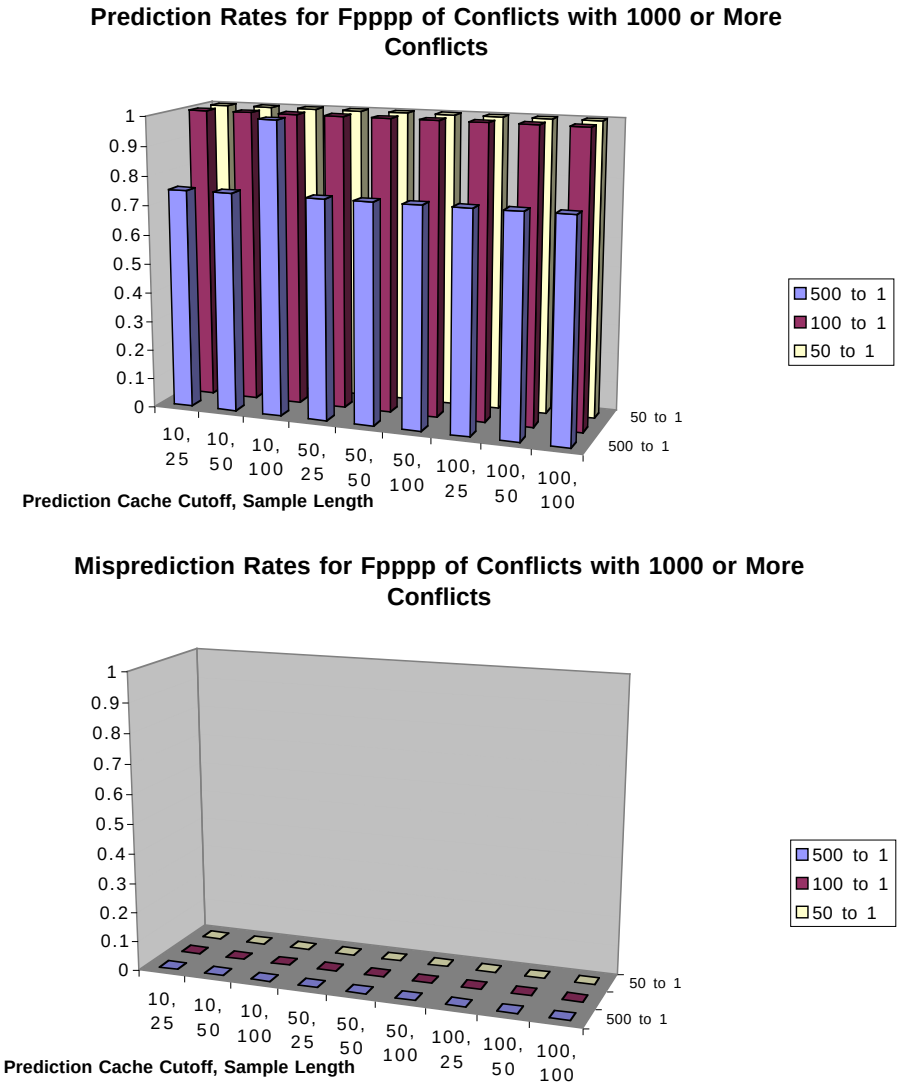


Figure 21: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 1000 or more in Fpppp

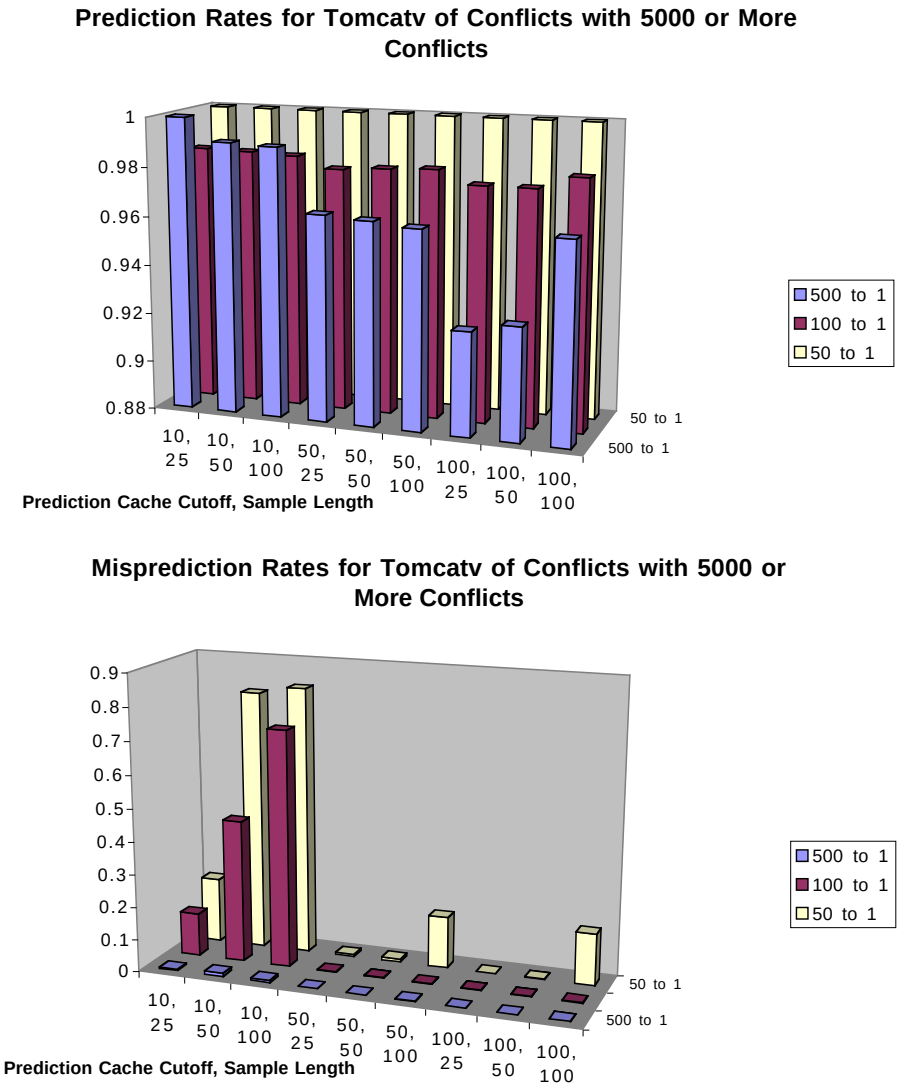


Figure 22: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 5000 or more in Tomcatv

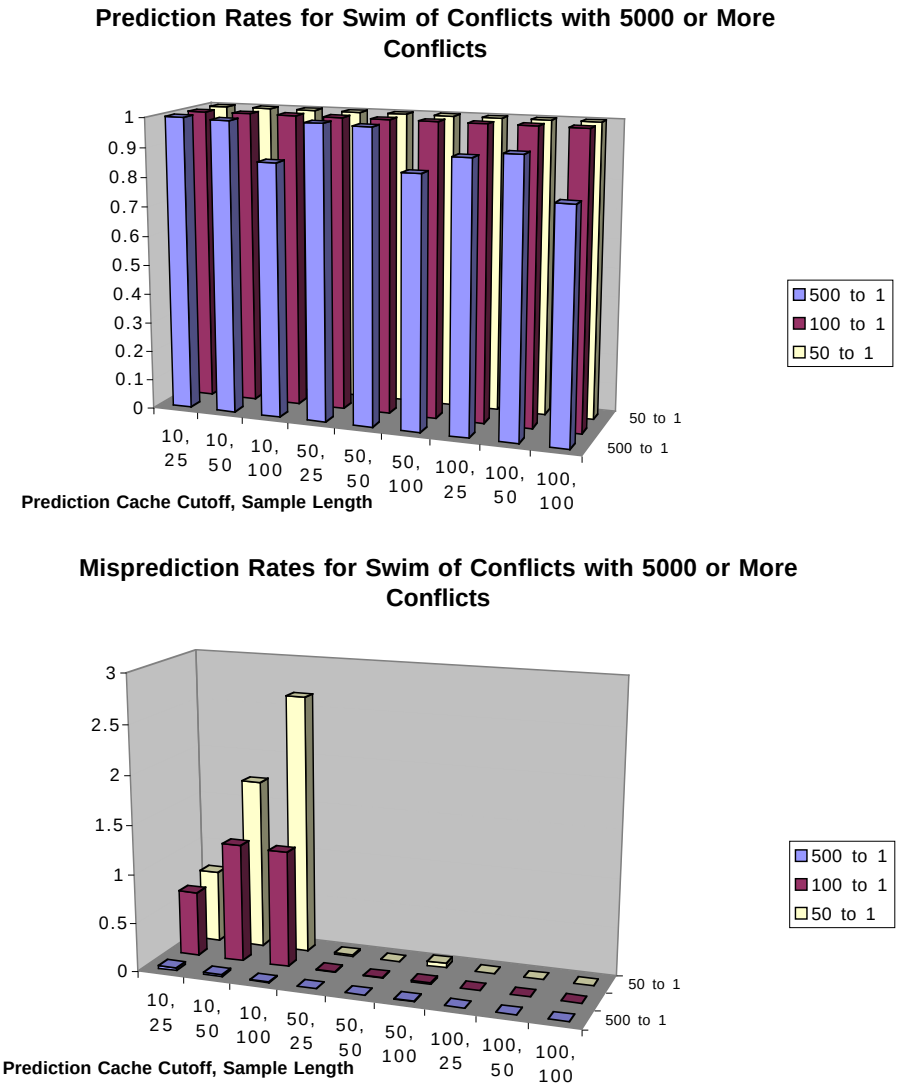


Figure 23: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 5000 or more in Swim

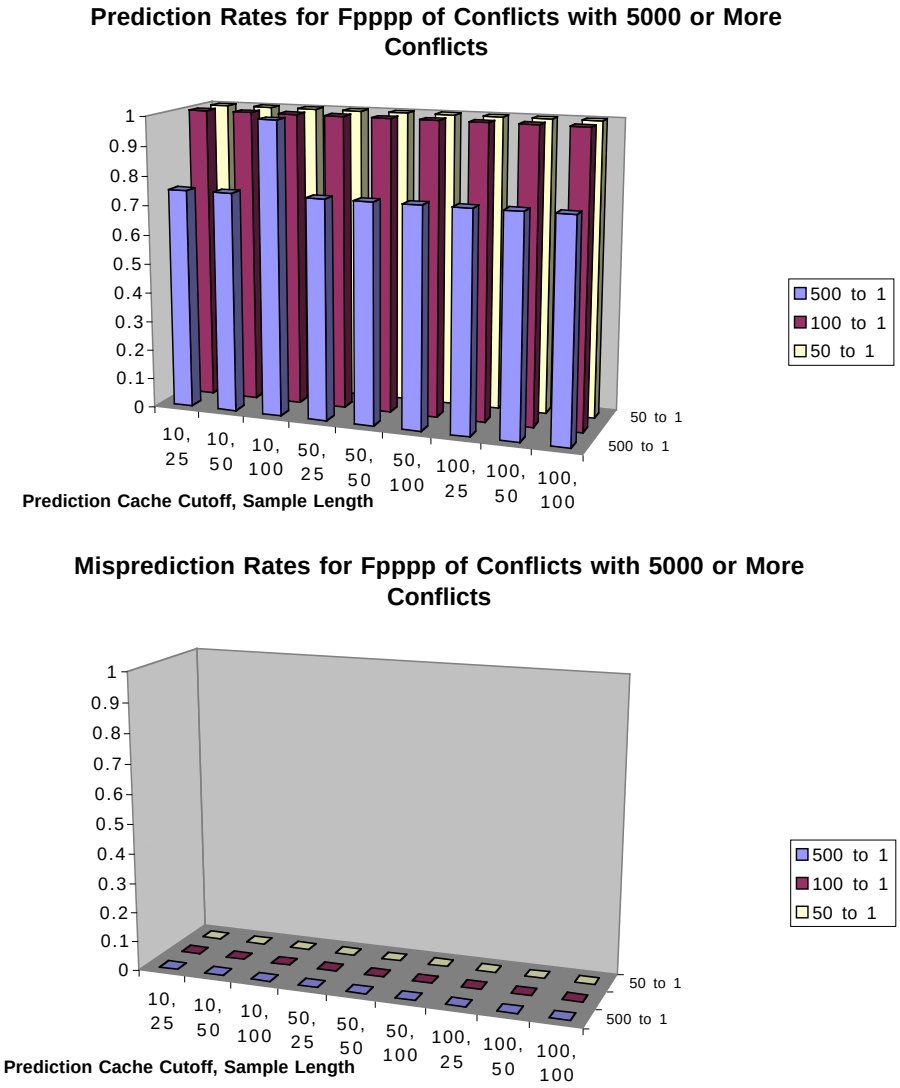


Figure 24: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 5000 or more in Fpppp

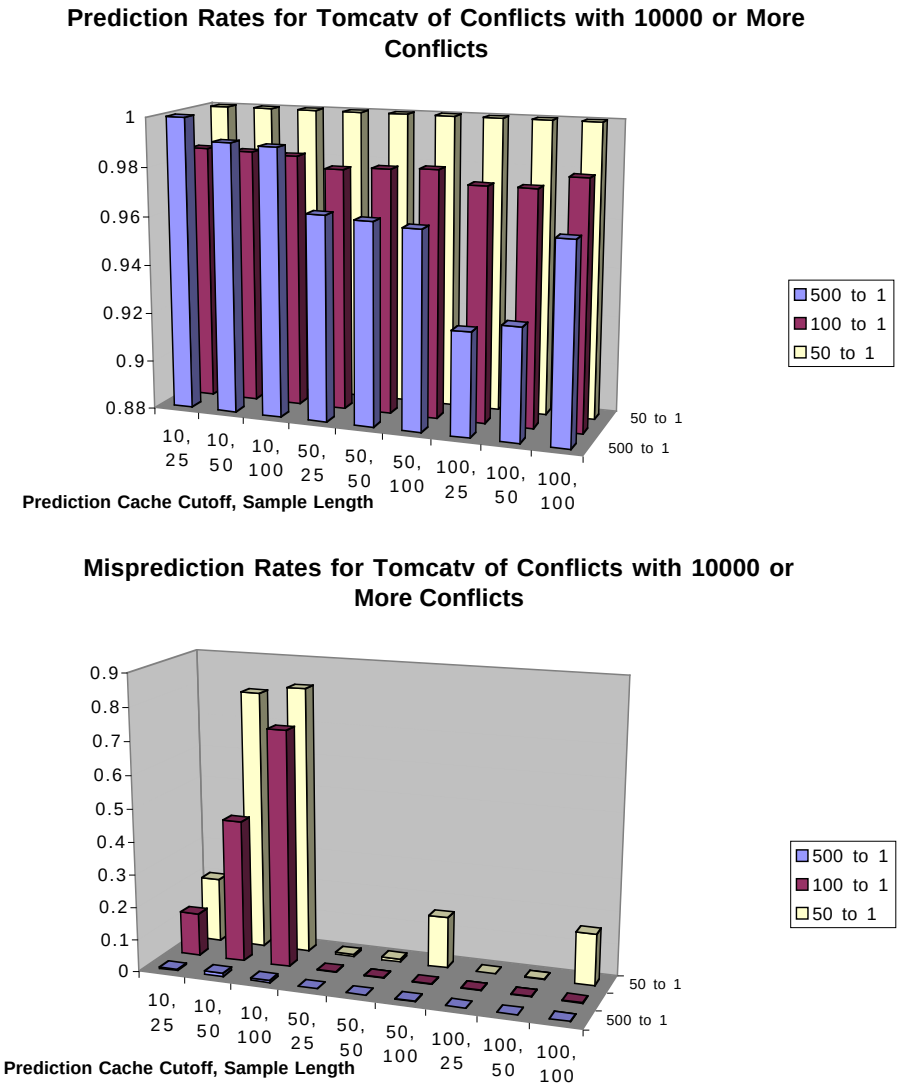


Figure 25: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 10000 or more in Tomcatv

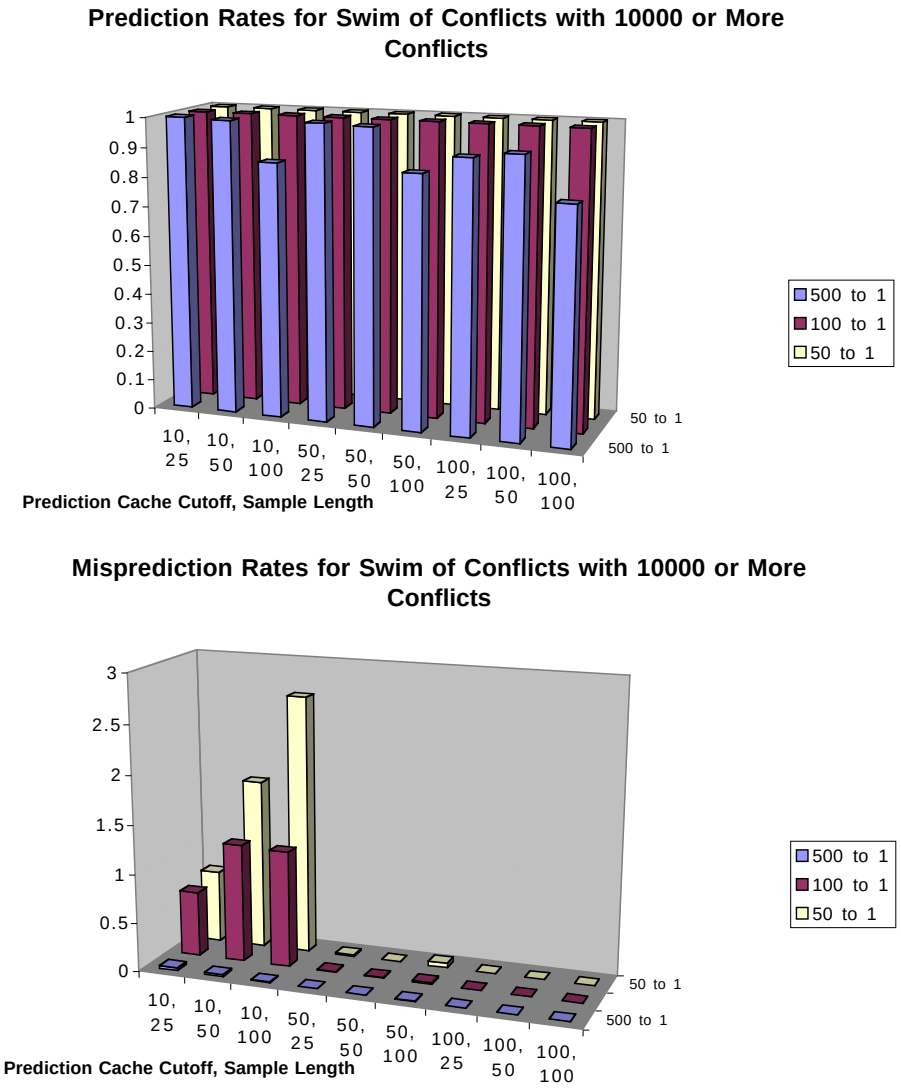


Figure 26: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 10000 or more in Swim

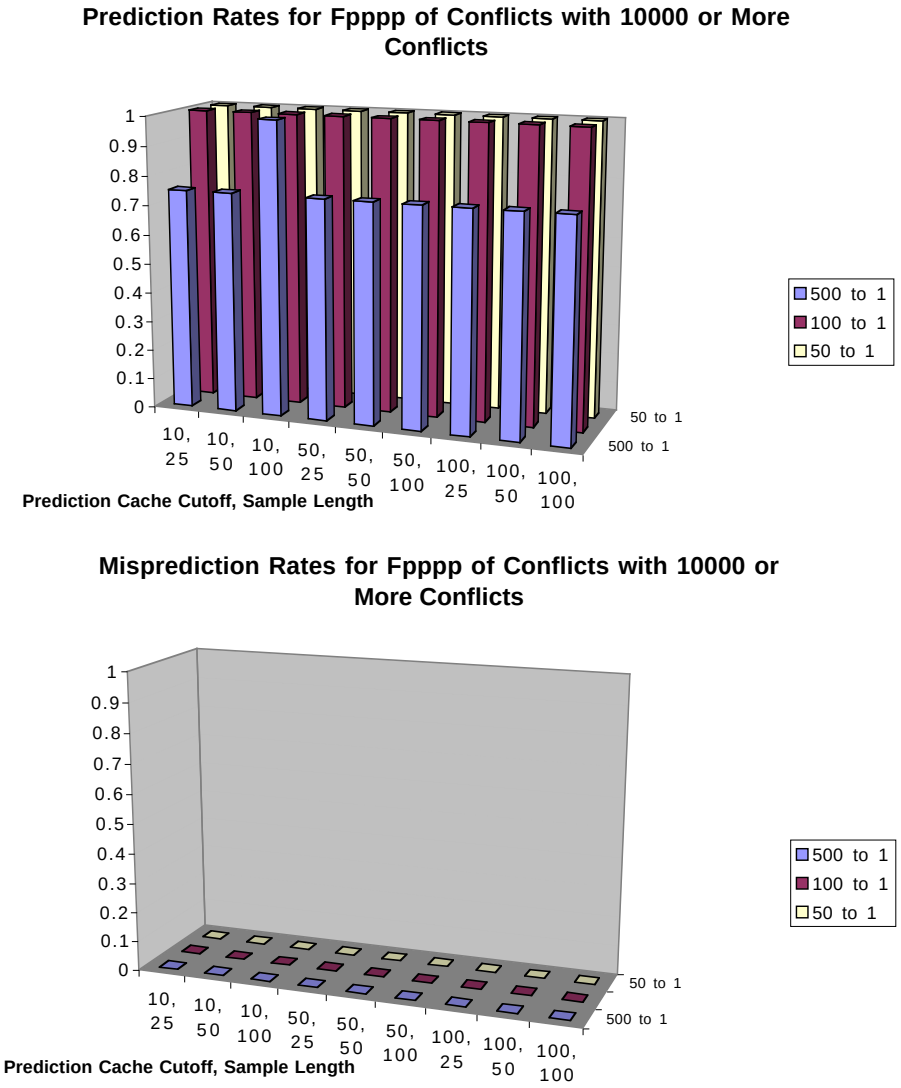


Figure 27: Experiment 3 - Prediction and Misprediction Rates for various Sampling Ratios and (Absolute Prediction Cache Cutoff, Sample Length) pairs for conflicts of 10000 or more in Fpppp

page is "recolored." No attention is given to the recoloring algorithms, as it is beyond the scope of this paper. Instead, once a remapping occurs, it is assumed that it will no longer conflict with the page it previously conflicted with, and, additionally, it will not conflict with other pages after remapping. This assumption is quite unrealistic, especially with the benchmarks that have high rates of capacity misses, but serves to give us a good notion of an upper bound on performance.

At the end of the trace a new statistic is generated, conflictMissesAvoided. This number, as well as data from Table 2, experimental results from Table 5 and some empirical evidence summarized in Table 4 is used to compute the maximum achievable speed up. For each cache architecture, the total number of cycles lost to cache overhead is computed. For the Direct-Mapped Cache, the 2-Way Associative Cache and the Fully Associative Cache, this is simply the total number of cache misses multiplied by the number of cycles per cache miss. For the new system, total cycles spent on cache overhead is ((number of direct mapped cache misses - conflictMissesAvoided) * (cycles per cache miss)) + (number of sampling sessions * overhead per sampling session) + (number of pages remapped * overhead of remapping). These results are summarized in Figure 28 - 30.

Both Fpppp performed almost as well as a fully associative cache. Swim shows performance almost as good as the 2-Way Associative cache. This sort of speedup was never demonstrated with the earlier systems. The Tomcatv performance, while greatly improved, does not even approach the performance of the 2-Way Associative cache. This is a surprising result because the CML improved performance on this same benchmark (on much shortor runs) to levels very close to the 2-Way Associative cache. This could be attributed to the very close temporal locality of cache misses in Tomcatv. The CML should perform best under these circumstances, and the CML has much lower sampling overhead then our approach.

In the Tomcatv and Swim benchmark, the overhead of sampling accounts for signicant portions of the time spent servicing the modified Direct-Mapped Cache. In

System Cost Assumptions	
Activity	*Cost in Cycles*
Cache Miss	25
Samling Session	11,000
Page Remapping	5,000

Table 4: Various Meterics used to create Figure 28, Figure 29, and Figure 30. The length of the sampling sessions was estimated empirically. The cost of remapping pages and of the cache miss were adapted from [BCL1] and [BCL2].

Experimental Results			
Benchmark	*Number of Sampling Sessions*	*Number of Pages Remapped*	Cache Misses Avoided
Tomcatv	21,406	2,841	1,325,170,360
Swim	2,214	230	30,422,749
Fpppp	3	4	1,785,218

Table 5: Ewpirical results from final experiment.

the Tomcatv benchmark, sampling accounts for about 75% of the total time. These numbers are surprising and suggest that some adjusttments might be benefitial. First, the overhead of each sampling session can be reduced. No time was spent optimizing this routine, and significant improvements might result from hand tuning this code. Second, a higher Sampling Rate might improve performance. While accuracy might decrease slightly, as shown in the earlier experiments, the significant savings in overhead might justify the reduced accuracy.

7 Discussion and Conclusions

Sampling was shown to be a very effective and efficient means of making the necessary decisions to facilitate dynamic page remapping. While there is a degree of variabil-

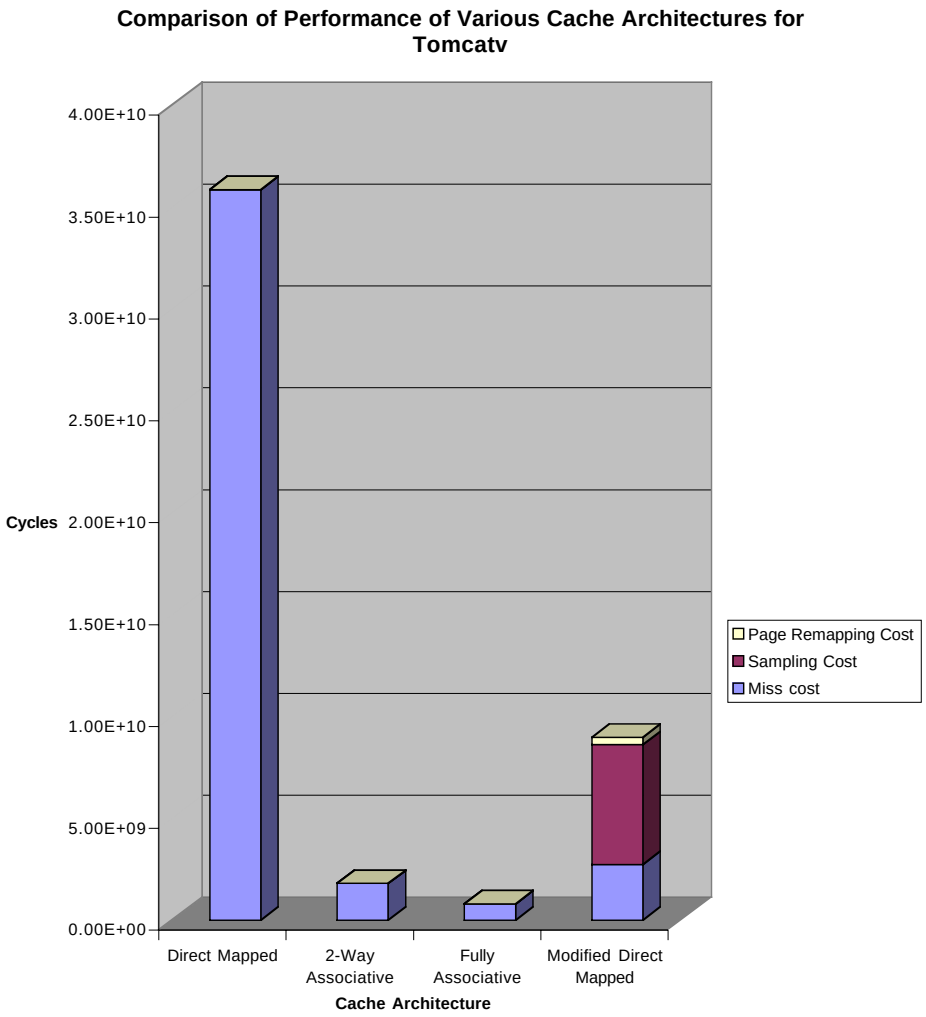


Figure 28: Experiment 4 - This graph shows the total cost of servicing the cache for the Tomcatv spec with four different cache schemes: Direct-Mapped, 2-Way Associative, Fully Associative, and the lower limit of the Direct-Mapped Cache with software sampling and page remapping. The cost of the first three caches is simply the cycles spent servicing misses. The modified Direct-Mapped cache includes the cost of servicing cache misses, the cost of sampling, and the cost of remapping pages.

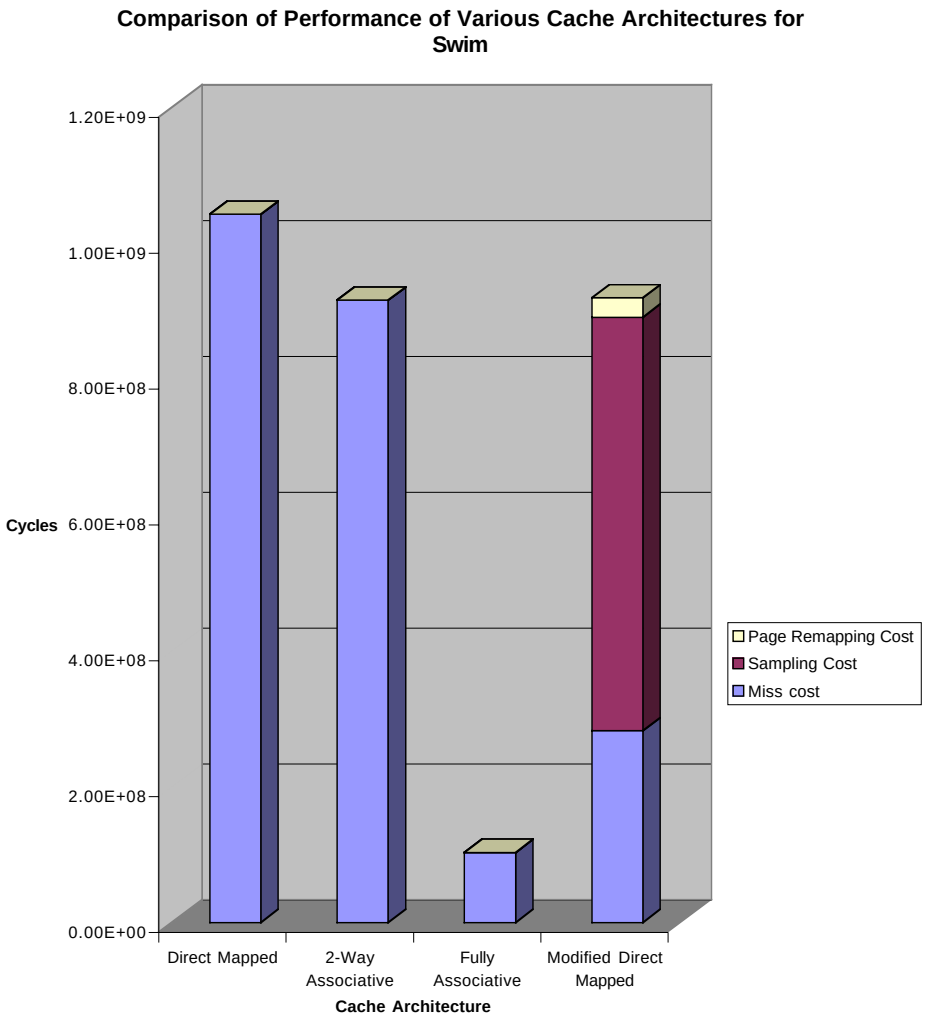


Figure 29: Experiment 4 - This graph shows the total cost of servicing the cache for the Swim spec with four different cache schemes: Direct-Mapped, 2-Way Associative, Fully Associative, and the lower limit of the Direct-Mapped Cache with software sampling and page remapping. The cost of the first three caches is simply the cycles spent servicing misses. The modified Direct-Mapped cache includes the cost of servicing cache misses, the cost of sampling, and the cost of remapping pages.

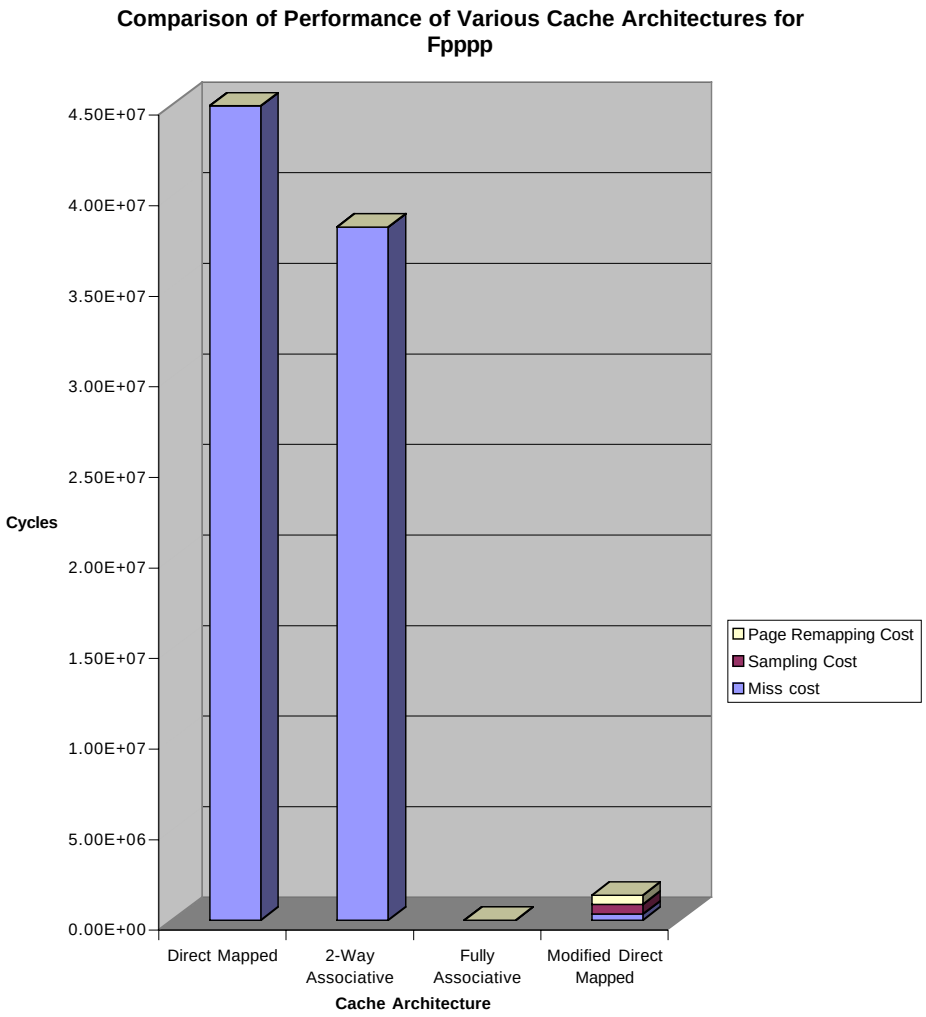


Figure 30: Experiment 4 - This graph shows the total cost of servicing the cache for the Fpppp spec with four different cache schemes: Direct-Mapped, 2-Way Associative, Fully Associative, and the lower limit of the Direct-Mapped Cache with software sampling and page remapping. The cost of the first three caches is simply the cycles spent servicing misses. The modified Direct-Mapped cache includes the cost of servicing cache misses, the cost of sampling, and the cost of remapping pages.

ity, our test show that selection of intelligent Sample Lengths, Sampling Rates, and Prediction Cache Cutoffs can greatly reduce this variability.

We demonstrated that sampling can effectively identify pages for remapping. The speedups that can be realized easily rival those of the CML. This approach also avoids the pitfalls identified earlier with the CML. Additionally, our approach is cheaper to implement in hardware and can be tested in hardware at a much lower cost.

There are also many potential benefits to having cache miss data in the OS. Eventually, it should be possible to augment this blind sampling with data from the OS and perhaps even supplement it with hints about data access patterns generated by compilers. This ability to use both dynamic cache miss information and data from the OS and compilers is unique to our implementation.

Further research should include expanding experiment 4 to study the effects of higher Sampling Rates on overall performance. Additionally, a larger suite of benchmarks should be tested. The sampling code should be optimized. Finally, intelligent remapping strategies should be tested in conjunction with sampling. If these results continue to show promise, the system should be implemented on the DEC machines that have a cache conflict counter and a cache conflict lache. This system will be able to give really accurate empirical performance results.

References

[DEC93] "ATOM: Reference Manual", Digitial Equiptment Corperation, December 1993.

[DEC95] "ATOM: User Manual" Digital Equiptment Corperation, June 1995.

[BCL1] Bershad, B. N., Chen, J. B., Lee, Dennis, and Romer, T. H. "Avoiding Cache Misses Dynamically in Large Direct-Mapped Caches." *Proc. 6th Interna-*

tional Conference on Architectural Support Programming Languages and Operating Systems, pages 158-170. ACM, 1994.

[BCL2] Bershad, B. N., Chen, J. B., Lee, Dennis, and Romer, T. H. "Dynamic Page Mapping Policies for Cache Conflict Resolution on Standard Hardware." *Proceedings of the First Symposium on Operating Systems Design and Implementation*, Usenix, November 1994.

[H96] Hennessy, John, and Patterson, David. *Computer Architecture A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., San Fransisco, California, 1996.

[H88] Hill, Mark D. "A Case For Direct Mapped Caches'." IEEE, December 1988.

[S95] Sites, Richard L. "Method and Apparatus for Cache Miss Reduction by Simulating Cache Associativity.", *United States Patent 5,442,571*. August, 1995.

[S94] Srivastava, Amitabh, and Eustace, Alan. "ATOM: A System for Building Customized Program Analysis Tools." *SIGPLAN '94 Conference on Programming Language Design and Implementation.*, 1994.