Probabilistic Cache Replacement

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:25620468">http://nrs.harvard.edu/urn-3:HUL.InstRepos:25620468</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Probabilistic Cache Replacement

J. Bradley Chen

TR-13-95

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts
Probabilistic Cache Replacement

J. Bradley Chen
Division of Applied Sciences
Harvard University

Abstract

Modern microprocessors tend to use on-chip caches that are much smaller than the working set size of many interesting computations. In such situations, cache performance can be improved through selective caching, use of cache replacement policies where data fetched from memory, although forwarded to the CPU, is not necessarily loaded into the cache. This paper introduces a selective caching policy called Probabilistic Cache Replacement (PCR) in which caching of data fetched from main memory is determined by a probabilistic boolean-valued function. Use of PCR creates a self-selection mechanism in which repeated misses to a word in memory increase its probability of being loaded into the cache. A PCR cache gives better reductions in instruction cache miss rate than a comparable cache configuration with a victim-cache. Instruction cache miss rates can be reduced by up to 30% for some of the SPECmarks, although the optimal probability distribution is workload dependent. This paper also presents a mechanism called Feedback PCR which dynamically selects probability values for a PCR cache. For an 16 K byte direct-mapped instruction cache, Feedback PCR with a one-entry MFB gives an average reduction in cache misses of over 11% across the SPECmarks with no significant increase in cache misses for any of the workloads, and compares favorably with other alternatives of similar hardware cost.

1. Introduction

Because of limited chip area, the on-chip caches of most microprocessors are much smaller than the working set size of typical applications. This mismatch between demand and available resources has a large negative impact on cache behavior. Figure 1 shows the references per block fetched from memory for the on-chip data cache of a DEC Alpha 26014 [Digital 92] during an execution of su2cor from the SPEC92 FP suite [SPEC92]. For su2cor, over 70% of blocks fetched into the data cache are referenced only once or twice before replacement. For these blocks, locality of reference is poor relative to cache size, and the proximity of cached data to the processor is of little benefit. Our measurements of the SPECmarks indicate that such behavior is not uncommon.

When a cache is smaller than a given working set, it doesn’t make sense to try to put everything in the cache. However, this is what current hardware attempts to do. Selective Caching is the use of caching policies which sometimes choose not to load data into the cache. Selective caching improves cache behavior by storing data which will be reused in the cache, and leaving data which will not be reused in main memory.

This paper is available from the Center for Research in Computing Technology, Division of Applied Sciences, Harvard University as technical report TR-13-95.
A selection function is a boolean-valued function which determines if a block from memory will be installed in the cache for a given cache miss. Current hardware generally uses a trivial selection function: everything goes into the cache. The tricky part of selective caching is the selection function; that is, deciding when not to load a block from memory into the cache. This paper introduces Probabilistic Cache Replacement (PCR), in which the selection function is based on a Bernoulli trial. The key feature of PCR is its self-selection behavior by which frequently referenced data has a higher probability of being loaded into the cache. Throughout this paper, $P_0$ will be used to refer to the probability that the block from memory will be loaded into the cache.

![Figure 1. Uses per fetch for su2cor.](image)

This figure shows uses per block for a 16 K byte direct-mapped data cache with 32 byte blocks. It illustrates the poor temporal locality patterns common when cache size is much smaller than working size. Note that most blocks installed in the cache are referenced only twice before replacement.

Selective caching motivates a device called a Memory Fetch Buffer (MFB), which temporarily preserves a block from memory which is not loaded into the cache. The MFB makes it possible to benefit from fine-grain locality (such as prefetch effects) without requiring that the block from memory be loaded into the cache. This paper considers one and four entry MFBs. The MFB is similar in complexity to structures such as a victim cache [Jouppi 90], but differs in management and also in performance characteristics. We compare the two devices in Section 5. In all configurations we assume that the cache and the victim cache or MFB are referenced in parallel with no impact on machine cycle time. In some cases this may not be a reasonable assumption, as the lookup in the auxiliary buffer would require an extra machine cycle.

The next section presents background information including observations based on a simple analytic model of PRC which motivate the strategy and give insight into why Bernoulli Trials might be useful for creating selection functions. In Section 3 we discuss the experimental system used to evaluate PCR. Sec-
tion 4 presents simulation results for PRC starting with experimental results for PCR with fixed probabilities (Static PCR) and continuing with a development of Feedback PCR, a mechanism for adjusting PCR probabilities dynamically.

Section 5 compares the Feedback PCR cache to other configurations (such as a victim cache) of similar hardware cost and continues with other considerations to explore the generality of the method. Section 6 concludes.

2. Background

Cache behavior is by its nature a stochastic phenomena. Fundamentally, the good behavior of caches depends on avoiding the statistical anomalies that could cause many active memory blocks to map to the same cache location. Although cache conflicts will always occur, we generally assume that cache activity will be dominated by average-case behavior. In this sense, all caching systems depend on probabilistic behavior and the law of large numbers [Ross 84] to achieve good performance. This probabilistic nature of cache behavior suggest that it might be well adapted to probabilistic schemes for cache management.

We now develop a recurrence equation that expresses the probability under Static PCR that a block will have been loaded into the cache after $n$ misses to that block. If a memory block $X$ has never been referenced, then it has zero probability of being in the cache:

$$F[0] = 0$$

Consider a memory reference to block $X$. With a traditional cache, the block from memory is always loaded into the cache. With PCR, $X$ will be loaded into the cache according to the following boolean-valued function:

$$F[R] = (R < P_0)$$

where $R$ is a uniform random variable on $[0,1]$ and $P_0$ is a probability on $[0,1]$. On the first miss, block $X$ will be loaded into the cache with probability $P_0$:

$$F[1] = P_0$$

If $X$ is never referenced again, then with probability $(1-P_0)$ the PCR cache will make the right choice and not load it into the cache.

Suppose that block $X$ is referenced twice. After the second reference, the probability that block $X$ has been loaded into the cache is the sum of:

- the probability it was loaded into the cache the first time: $P_0$, and
- the probability it was not loaded the first time and was loaded the second time: $(1-P_0)P_0$

which gives

$$F[2] = P_0 + (1 - P_0)P_0$$
$$= F[1] + (1 - F[1])P_0$$

After three references, the probability that block $X$ will have been loaded into the cache is


In general, after $n$ references, the probability that $X$ has been installed in the cache is given by the following recurrence:

$$F[n] = F[n-1] + (1 - F[n-1])P_0$$
$$= F[n-1] + P_0 \cdot P_0(F[n-1])$$
\[
P = P_0 + (1 - P_0)F[n-1]
\]

Figure 2 shows a plot of this recurrence with \( P_0 \) varying from 0.1 to 1.0. The key observation from Figure 2 is that with only a small number of repeated references, the probability that a word has been installed in the cache becomes high. Even at \( P_0 = 0.5 \), only four references to a block are required before its probability of having been loaded into the cache reaches 0.9. The value of the probability recurrence increases rapidly with repeated references, and creates a self-selection behavior where blocks that miss repeatedly have a higher probability of being loaded into the cache than blocks that miss only once. This behavior is the basis of PCR.

To summarize, PCR has two properties which make it an interesting choice as a selection function:

- The more you reference a block of data, the more likely it will end up in the cache.
- Some of the data referenced only once will not be loaded into the cache, avoiding the replacement of useful cached data.

With an MFB, data not loaded into cache is preserved until flushed from the MFB by subsequent non-caching misses. The MFB provides a degree of associativity, similar to a victim cache, which might also be of benefit without PCR. The benefit of PCR beyond that of the MFB is examined in Section 5.

PCR has the potential to introduce variation in cache behavior between runs because it employs randomness. Fortunately, the reference patterns of interesting computations tend to be long and repetitive, so the variation in behavior due to PCR should be minimal. Section 5.3 explores repeatability issues for Feedback PCR caches.

![Figure 2: Probability Recurrence for PCR](image)

This figure shows plots of the recurrence \( F[n] = P_0 + (1 - P_0)F[n-1] \) for values of \( P_0 \) ranging from 0.1 to 1.0. Values of \( n \) (number of cache misses) increase along the x-axis. This figure shows that with repeated references, the probability that a block from memory is loaded into the cache becomes very high.

### 2.1 Related Work

Prior work to improve performance of small caches has focused on deterministic methods. Associativity is the standard technique. Miss buffers and victim caches [Jouppi 90] avoid conflicts in a direct mapped cache by augmenting it with a small associative buffer which is very similar to the MFB used in PCR. A more aggressive approach is to try to put things into the cache before they are needed. Stream
buffers [Jouppi 90] and prefetch instructions [Farkas & Jouppi 94, Chen & Baer 94] hide memory system latency by issuing loads from memory before the data is needed.

Selective caching differs from other methods in that it tries to keep things out of the cache in order to use the space for data which will be reused. Dynamic exclusion [McFarling 92] is a selective caching mechanism for eliminating conflict misses in direct mapped instruction caches. The selection function uses a two-bit state machine for each cache line as input. The state machine recognizes commonly occurring cache-miss patterns where storing a line from memory harms performance. Another project proposed a bypass cache [Hoevel & Milutinovic 89] as a part of a scheme to prevent infrequently used memory blocks from returning to the cache. This scheme uses compiler heuristics to determine when a load should bypass the cache.

Computers with cache memories generally support non-caching reads and writes to I/O devices, but their purpose is correctness rather than cache performance. Machines with memory-mapped frame buffers often bypass the cache in frame-buffer writes. In this case, the purpose can be to avoid displacing cache contents [McCormick 91]; however the selection function is completely determined by the memory reference address and is not probabilistic.

There are many applications of randomization to resource allocation in computer software systems. The Mach 3.0 operating system uses a random policy to select the physical page to back a given virtual page [Chen & Bershad 93]. Lottery Scheduling uses randomization as part of a sort of lottery contest to schedule CPU resources [Waldspurger & Weihl 94].

Feedback PCR uses miss rate information from the running program to dynamically adjust the probability distribution used for PCR. Dynamic page mapping strategies [Bershad et al. 94, Romer et al. 94] also use feedback based on cache miss rates or TLB misses. Dynamic page mapping techniques differ fundamentally from PCR in that they begin by assuming that the cache is big enough to hold the computation’s working set and then eliminate cache conflict misses by relocating or recoloring pages within the cache. Dynamic page-mapping strategies perform poorly when a cache is much smaller than the computation’s working set because there isn’t a good candidate for the destination of the recolor. In contrast, PCR starts with the assumption that the cache is too small for the current working set.

3. Methodology

Trace driven simulation was used to simulate the performance of PCR and standard caches. The experiments used traces of instruction and data reference activity for workloads in the SPEC92 suite [SPEC92]. The simulations were conducted on DEC Alpha AXP workstations using the Atom software instrumentation system from Digital Equipment Corporation [Digital 92, Srivastava & Eustace 94]. Split instruction and data caches are assumed throughout. All caches are read-allocate/write around with 64 byte blocks. Writes to blocks which are not in the cache have no effect on cache contents. The simulations assume virtually indexed caches. For small caches, this does not affect the generality of the results. Data cache miss rates are given as data cache misses per instruction.

Operating system activity was not included for these experiments. Prior research has shown that interaction between user and system activity has minimal impact on the behavior of caches as small as those measured in this study [Chen & Bershad 93]. A separate question of interest is the potential beneficial effect of PCR on system reference streams, although this is outside the scope of the current work.

The SPEC92 benchmark suite was used to test PCR behavior under a diverse range of memory reference patterns. Although these workloads have serious limitations when evaluating the performance of
complete memory systems or large caches due to small working set size [Chen 94], many of them do provide interesting miss rates for caches of the size considered in this paper. Table 1 gives some baseline measurements for the SPEC92 suite.

<table>
<thead>
<tr>
<th>workload</th>
<th>instructions</th>
<th>data references</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compres</td>
<td>87296503</td>
<td>19310270</td>
<td>file compression utility</td>
</tr>
<tr>
<td>eqntott</td>
<td>1800890167</td>
<td>224725209</td>
<td>build truth table from boolean expression</td>
</tr>
<tr>
<td>espress</td>
<td>118766783</td>
<td>24821900</td>
<td>boolean function minimization</td>
</tr>
<tr>
<td>gcc1</td>
<td>23094634</td>
<td>4956750</td>
<td>compile pre-processed C to MC68000 assembler</td>
</tr>
<tr>
<td>li</td>
<td>6772634774</td>
<td>1741681716</td>
<td>xlisp solving nine queens</td>
</tr>
<tr>
<td>sc</td>
<td>115491031</td>
<td>24695609</td>
<td>spreadsheet calculation</td>
</tr>
<tr>
<td>alvinn</td>
<td>5233314204</td>
<td>1410937848</td>
<td>train a neural network by backpropagation</td>
</tr>
<tr>
<td>doduc</td>
<td>1114333020</td>
<td>247081503</td>
<td>thermohydraulic simulation of nuclear reactor</td>
</tr>
<tr>
<td>ear</td>
<td>13104739758</td>
<td>2792463422</td>
<td>simulate sound propagation in inner ear</td>
</tr>
<tr>
<td>fpppp</td>
<td>4220204349</td>
<td>1394657621</td>
<td>two electron integral derivative</td>
</tr>
<tr>
<td>hydro2d</td>
<td>6258486249</td>
<td>1431804466</td>
<td>solve Navier Stokes equations (astrophysics)</td>
</tr>
<tr>
<td>mldjp2</td>
<td>2930268743</td>
<td>424603679</td>
<td>solve for motion in atom system - double precision</td>
</tr>
<tr>
<td>mldjsp2</td>
<td>3687060319</td>
<td>757405499</td>
<td>solve for motion in atom system - single precision</td>
</tr>
<tr>
<td>nasa7</td>
<td>6170324098</td>
<td>1760347996</td>
<td>seven floating point kernels</td>
</tr>
<tr>
<td>ora</td>
<td>5783034488</td>
<td>915199289</td>
<td>optical simulation</td>
</tr>
<tr>
<td>spice</td>
<td>17651286445</td>
<td>4781043817</td>
<td>analog circuit simulator</td>
</tr>
<tr>
<td>su2cor</td>
<td>4707800071</td>
<td>1043044014</td>
<td>compute masses of elementary particles</td>
</tr>
<tr>
<td>swm256</td>
<td>11039022292</td>
<td>2850202583</td>
<td>shallow water model using finite difference approximations</td>
</tr>
<tr>
<td>tomcatv</td>
<td>910175498</td>
<td>247331311</td>
<td>vector mesh generation</td>
</tr>
<tr>
<td>wave5</td>
<td>3452161308</td>
<td>723069400</td>
<td>relativistic simulation of plasma phenomena</td>
</tr>
</tbody>
</table>

Table 1: Baseline measurement values for the SPEC92 Suite.

4. Experimental Results

This section begins with base results for a 16 K byte Static PCR cache. These base results show the potential of PCR and give some intuition for understanding PCR behavior. Next we present a development of Feedback PCR including a discussion of considerations for eliminating undesirable PCR behavior. Section 5 explores issues related to the general applicability of PCR.
4.1 Static PCR

Figure 3 shows a block diagram of a Static PCR cache. The Static PCR cache requires an MFB, a register to hold $P_0$, a comparator, and a random number generator, all of which are straightforward to implement. The MFB is similar to a miss cache [Jouppi 90] in the way it is accessed, although its management differs. Random number generators can be implemented using a linear feedback register [McCluskey 86].

Figure 4 shows Static PCR cache miss rates for the SPEC92 suite as $P_0$ is varied from 0.1 to 1.0 in increments of 0.1. Static PCR reduces cache miss rates by up to 40% (tomcatv data). In the instruction cache, only 3 of 20 workloads perform best at $P_0 = 1.0$, ora, swm256, and hydro2d. All three of these workloads have very low I-cache miss rates (less than 0.00005 misses/instruction). Two workloads, fpppp and alvinn, perform best with $P_0 = 0.2$. This suggests a situation where a very long basic block overlaps with itself, or where in-cache execution of a loop is periodically disrupted by straight-line code.

The best data cache performance for most workloads occurs with $P_0$ around 0.8 and 0.9. Only one workload, spice, performs best at $P_0 = 1.0$. PCR has very little effect on data cache activity for compres. This is a situation where data is streaming through the processor with good spatial locality but no temporal locality, for which caching is of little benefit.

It should be noted that the comparison of the Static PCR cache in Figure 4 puts the standard cache at an unfair disadvantage; the PCR cache benefits from the additional associativity provided by the MFB. Section 5.1 explores this issue by comparing a PCR cache to a cache that uses a similar MFB with deterministic management.

Overall, the measurements for Static PCR demonstrate the potential for improving cache miss rates. Unfortunately, there is no single value of $P_0$ that gives optimal performance for all workloads. The best value for $P_0$ depends on the workload in question. This is our motivation for the development of Feedback PCR in the next section. An important observation about the overall shape of the curves is that they gener-
ally tend to be concave upward with no local minima apart from the global minimum, a property that will be useful in the development of Feedback PCR.

### 4.2 Feedback PCR

This section describes a feedback mechanism which continuously monitors cache miss activity and adjusts the value of $P_0$ accordingly. The basic goals of the feedback mechanism are:

- Adjust probabilities to track program behavior over time.
- Keep $P_0 = 1.0$ when this gives best performance.

Typically we think of cache behavior as a single miss rate for a given workload, but this is deceptive. Any realistic computation is composed of multiple stages of activity each with its own particular patterns of data reference activity. It follows that each of these stages of activity has an optimal value for $P_0$. Intuition suggests that a PCR with a feedback mechanism that can respond to changes in program behavior should give better performance than a Static PCR cache. The challenge in designing a feedback mechanism is to respond to measurements of steady state behavior and ignore cache miss activity when it is dominated by transients. The basic algorithm for the feedback system is given in Figure 5. The good behavior of such a feedback mechanism depends on the assumption that the variation of miss rates with respect to $P_0$ is well

---

**Figure 4.** Experimental Results for Static PCR.

This figure shows miss rates for a 16 K byte Static PCR cache for the SPEC92 suite. The experiments used split instruction and data caches with a block size of 64 bytes, and read-allocate policy for the data cache.
behaved with no local minima apart from the global minimum. The results presented in Figure 4 and the simulation results for Feedback PCR indicate that, in general, this condition holds true.

Although the feedback mechanism is simple and straightforward to implement in hardware, there are a number of subtleties, all related to the problem of distinguishing between transient and steady-state behavior.

- **interval** must be large enough to smooth out most transient behavior. A value of 2000000 instructions is used for these simulations.
- **dmag** must be large enough, as small variations in $P_0$ will have no noticeable performance impact. $dmag = 5$ for these simulations.
- **delta** should be relatively small, to provide good granularity in tuning probabilities and avoid making big mistakes. $delta = 0.02$ for these simulations.
- The sampling mechanism should vary $P_0$ during measurement intervals so as to be able to identify and ignore transient behavior.

- Start with initial values for $P_{base}$, $delta$, $dmag$, and **interval**.

$P_0 := P_{base}$;

$base1 :=$ misses during next **interval** instructions;

$P_0 := P_{base} + dmag \times delta$

test := misses during next **interval** instructions

$P_0 := P_{base}$

$base2 :=$ misses during next **interval** instructions

if $(base1 > test)$ and $(base2 > test)$

$P_{base} := P_{base} + delta$

if $(base1 < test)$ and $(base2 < test)$

$delta := (-delta)$

$P_{base} := P_{base} + delta$

else test was inconclusive; don’t change $P_{base}$.

**Figure 5.** Feedback mechanism for Feedback PCR.

The feedback mechanism described in Figure 5 is the result of a series of trial-and-error experiments searching for an appropriate mechanism. There are many possible pitfalls when designing such a mechanism. For example, a mechanism that similar to that of Figure 5 but in which $P_0$ takes the values \{ $P_{base} - delta$, $P_{base}$, $P_{base} + delta$ \} performs badly because it fails to discriminate between changing and steady-state miss behavior. Using \{ $P_{base}$, $P_{base} - delta$, $P_{base} + delta$ \} works well for most workloads, but it causes a significant performance penalty in the data cache for *spice*.

Table 2 gives simulation results to evaluate the feedback scheme with respect to the best Static PCR performance for SPEC92 FP Data Cache behavior. The feedback mechanism consistently selects uncached to cached ratios which are close to the optimal value for $P_0$ indicated by the Static PCR experiments (Figure 4). For most workloads, the cache miss rate is slightly higher for Feedback PCR. This represents the overhead of the search mechanism, constantly increasing and decreasing $P_0$ in a search for the optimal value, a behavior which only has a measurable impact when cache miss rates are very low.

An interesting behavior occurs for *nasa7*. This workload has seven different localities corresponding to seven different floating point kernels, each with its own data reference pattern. The feedback mechanism
appears to have adapted to these localities independently, thereby achieving a lower overall miss rate for the program run. Wave5 behaves similarly. These results indicate that the feedback scheme does a reasonable job of dynamically determining the appropriate value for $P_0$.

<table>
<thead>
<tr>
<th>workload</th>
<th>Optimal $P_0$</th>
<th>average $P_0$</th>
<th>Static PCR misses/inst</th>
<th>Feedback PCR misses/inst</th>
</tr>
</thead>
<tbody>
<tr>
<td>alvinn</td>
<td>0.8</td>
<td>0.815</td>
<td>0.00877942</td>
<td>0.00899719</td>
</tr>
<tr>
<td>doduc</td>
<td>0.9</td>
<td>0.951</td>
<td>0.01589523</td>
<td>0.01608509</td>
</tr>
<tr>
<td>ear</td>
<td>0.6</td>
<td>0.597</td>
<td>0.00298130</td>
<td>0.00294562</td>
</tr>
<tr>
<td>fpppp</td>
<td>0.9</td>
<td>0.833</td>
<td>0.00573750</td>
<td>0.00605873</td>
</tr>
<tr>
<td>hydro2d</td>
<td>0.9</td>
<td>0.927</td>
<td>0.02223732</td>
<td>0.02283202</td>
</tr>
<tr>
<td>mdljdp2</td>
<td>0.9</td>
<td>0.963</td>
<td>0.00731129</td>
<td>0.00751677</td>
</tr>
<tr>
<td>mdljsp2</td>
<td>0.9</td>
<td>0.811</td>
<td>0.00297516</td>
<td>0.00313707</td>
</tr>
<tr>
<td>nasa7</td>
<td>0.6</td>
<td>0.549</td>
<td>0.08124870</td>
<td>0.07834882</td>
</tr>
<tr>
<td>ora</td>
<td>0.2</td>
<td>0.233</td>
<td>0.00067332</td>
<td>0.00068475</td>
</tr>
<tr>
<td>spice</td>
<td>1.0</td>
<td>0.864</td>
<td>0.06403922</td>
<td>0.06549688</td>
</tr>
<tr>
<td>su2cor</td>
<td>0.7</td>
<td>0.682</td>
<td>0.06581248</td>
<td>0.06618603</td>
</tr>
<tr>
<td>swm256</td>
<td>0.9</td>
<td>0.923</td>
<td>0.01217696</td>
<td>0.01242938</td>
</tr>
<tr>
<td>tomcatv</td>
<td>0.8</td>
<td>0.772</td>
<td>0.02692122</td>
<td>0.02807343</td>
</tr>
<tr>
<td>wave5</td>
<td>0.9</td>
<td>0.906</td>
<td>0.00873827</td>
<td>0.00788964</td>
</tr>
</tbody>
</table>

Table 2. Evaluation of Feedback Mechanism

This table provides data for simulations using the feedback mechanism in Figure 5. The experiments used a 16 K byte cache with 64 byte blocks. The comparable values for optimal static $P_0$ vs. average $P_0$ from the Feedback PCR run and the comparable miss rates for optimal Static and Feedback PCR demonstrate that the feedback mechanism succeeds at selecting good values for $P_0$ over the range of workloads.

5. Evaluation

5.1 Comparison

This section compares a standard direct-mapped cache with three alternatives which use additional hardware to eliminate cache conflict misses:

- a PCR cache
- a Deterministic Memory Fetch Buffer
- a victim cache

All three of these methods have similar hardware requirements. The Deterministic Memory Fetch Buffer provides a basis for evaluation of the benefit of the associativity of the MFB independent of PCR. In the DMFB cache, all misses go first into the MFB; when replaced in the MFB they go into the cache. The PCR and DMFB caches manage their MFB as a simple FIFO. In the victim cache, a hit in the auxiliary buffer
causes it to be swapped with the a block from the cache, so that the most recently referenced block is always in the cache rather than the auxiliary buffer.

Figure 6. Miss rate improvements for PCR, DMFB, and Victim Caching techniques.

This figure show miss rates for PCR, DMFB, and victim cache in terms of (misses)/(misses for standard cache). Results are for a 16 K byte direct-mapped cache with a block size of 64 bytes and a single entry MFB. Across the SPEC92 suite, the PCR cache is more effective at reducing instruction cache misses, while the victim cache is more effective in the data cache.

We start by considering variants with an auxiliary buffer that holds a single block from memory. Figure 6 shows instruction cache performance for the three schemes in terms of percentage of total cache misses eliminated. All schemes give significant performance improvements over the standard cache. For most workloads, PCR gives better instruction-cache performance than the other two variants. In particular, Feedback PCR gives superior performance for the four workloads with the worst instruction cache performance (see Figure 4): fpppp, doduc, gcc, and sc. Feedback PCR has negative instruction cache performance for three workloads: hydro2d, nasa7, and ora. However, because the instruction cache miss rates for these workloads are very small, the impact on overall performance is not significant.

In the Data cache, the configuration with a victim cache consistently gives the best performance. In many cases, the DMFB cache gives superior performance to the PCR cache. This occurs when the self-selection effects of PCR are not significant enough to compensate for the higher average cache miss penalty required to get useful data into the cache. The benefit of the additional associativity provided by the MFB becomes apparent. These results reflect a difference in the locality properties of instruction and data reference streams: instruction blocks are more likely to be referenced once without reuse. Two examples of specific instruction reference patterns where instructions are not reused are:
Another plausible explanation for greater PCR applicability to instruction reference streams is that the PCR cache does not perform well with higher rates of miss activity as in the data cache. Results in the next section will show that this is not the case, and that the effectiveness of PCR does not decrease as the mismatch between instruction cache size and working set size becomes more severe.

We now consider PCR, Victim, and DMFB instruction caches with four-entry auxiliary buffers. The PCR and DMFB caches use FIFO management to select victims in the MFB. The victim cache is slightly more complex. It uses LRU replacement in the auxiliary buffer in addition to the swap on a victim cache hit. With four entry associative buffers, (Figure 7), the PCR cache has less of an advantage over the victim cache than in the one entry case. Still, the three workloads with the worst instruction cache performance (gcc1, doduc, and fpppp) benefit more from PCR than victim caching. Of the seven the workloads for which the victim cache works better than PCR, five of them (compress, hydro2d, mdljsp2, nasa7, and tomcatv) have instruction cache miss rates that are very low (0.0002 misses per instruction or less). The other two (li and sc) still get substantial benefit from PCR, though not quite as much as with the victim cache. The next section shows how the victim cache looses its advantage for these two workloads when the size of the cache is reduced.

These results suggest that some combination of PCR with a victim cache might achieve the advantages to each. We have explored variants on the victim cache using probabilistic (rather than deterministic) swapping, and also variants on the PCR cache with probabilistic promotion of blocks from the MFB to the cache on an MFB hit. Neither of these schemes showed significant advantages; at best they began to approximate the behavior of the victim cache while loosing the beneficial effects of PCR. We are continuing to explore enhancements for PCR.

### 5.2 Impact of Cache Size

This section considers the impact of cache size on PCR cache behavior. Two concerns motivate this study. The first and most obvious is that on-chip cache sizes vary between machine implementations. The study of various cache sizes is necessary to understand how the results for 16 K byte caches generalize to...
other cache sizes. Secondly, the 16 K byte instruction cache is large enough to contain the working sets for many workloads in the SPEC92 suite (see Figure 4). In decreasing the cache size we put more pressure on instruction-cache, and this gives an indication of how PCR will behave when the mismatch between cache size and working set size is more severe.

![Figure 8. Miss rates improvements for an 8 K byte instruction cache.](image)

This figure shows miss rates for PCR, DMFB, and victim cache in terms of \(\text{misses}/(\text{misses for standard cache})\). Results are for a 8 K byte direct-mapped cache with a block size of 64 bytes and a single entry MFB. Overall, the advantage of PCR over victim caching increases with decreasing cache size.

Figure 8 shows cache miss rate improvements for PCR, DMFB, and victim caching for an 8 K byte instruction cache, each with a single-entry auxiliary buffer. In most cases the PCR cache gives the best performance improvement. Also note that only one workload (\textit{hydro2d}) has a worse cache miss rate with PCR, although the miss rate for \textit{hydro2d} is still very low (about 0.00009 misses per instruction with PCR). For all other workloads, PCR gives performance that is as good as or better than performance for the single-entry victim cache. Although the percent of cache misses eliminated is often smaller for the 8 K byte cache as compared to the 16 K byte case, the miss rates are also higher, and the absolute benefit of PCR is greater.

### 5.3 Repeatability

<table>
<thead>
<tr>
<th>workload</th>
<th>instruction mean</th>
<th>instruction max - min</th>
<th>instruction SD</th>
<th>data mean</th>
<th>data max - min</th>
<th>data SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>doduc</td>
<td>0.0126540</td>
<td>0.0000076</td>
<td>0.0000025</td>
<td>0.0137159</td>
<td>0.0000787</td>
<td>0.0000302</td>
</tr>
<tr>
<td>gcc1</td>
<td>0.0183888</td>
<td>0.0000482</td>
<td>0.0000177</td>
<td>0.0106708</td>
<td>0.0003090</td>
<td>0.0001120</td>
</tr>
<tr>
<td>hydro2d</td>
<td>0.0000508</td>
<td>0.0000028</td>
<td>0.0000010</td>
<td>0.0202824</td>
<td>0.0000546</td>
<td>0.0000197</td>
</tr>
<tr>
<td>li</td>
<td>0.0032380</td>
<td>0.0000125</td>
<td>0.0000040</td>
<td>0.0164781</td>
<td>0.0000077</td>
<td>0.0000029</td>
</tr>
</tbody>
</table>

**Table 5. Repeatability**

This figure shows data from instruction cache miss rates for five independent runs of each workload, using a 16 K byte PCR instruction cache with a four entry MFB.

As it relies on non-deterministic phenomena, PCR has the potential to introduce unwanted variability into program execution time and overall system performance. We measured behavior of \textit{doduc, gcc1},
hydro2d, and 1i during multiple runs to understand how PCR affects repeatability. Table 5 shows the mean, range and standard deviation for instruction and data cache miss rates during five independent runs of the benchmarks. In no case does PCR introduce significant variance into cache miss rates. Only for gcc1 data is the range more than 2% of the mean miss rate. We conclude that repeatability for a PCR cache is good.

6. Conclusions

Probabilistic Cache Replacement has significant potential for improving the performance of direct mapped instruction caches. Our mechanism for Feedback PCR succeeds in adjusting the PCR parameter \( P_0 \) such that an appropriate value is selected for the reference pattern in question. For most workloads, use of Feedback PCR gives superior instruction cache performance to a victim cache of comparable size. This is true in particular for the workloads with the most elevated instruction cache miss rates. PCR cache performance remains stable as the mismatch between cache size and working set size becomes more severe and does not introduce significant variability into cache miss behavior. This assures the relevance of PCR for future workloads and machines. Overall, PCR is an effective mechanism for reducing cache miss rates when the mismatch between cache size and working set size is extreme.

Acknowledgments

This work benefitted from discussion with Alan Eustace throughout the project. Thanks to Alan Eustace and Trevor Blackwell for comments on earlier drafts of this paper. The author would also like to thank Digital Equipment Corporation for their generous support.
Bibliography


