Evaluating a Model for Cache Conflict Miss Prediction

Apan Qasem Ken Kennedy
Department of Computer Science
Rice University
Houston, TX
{qasem,ken}@cs.rice.edu

October 4, 2005

Abstract

Cache conflict misses can cause severe degradation in application performance. Previous research has shown that for many scientific applications majority of cache misses are due to conflicts in cache. Although, conflicts in cache are a major concern for application performance it is often difficult to eliminate them completely. Eliminating conflict misses requires detailed knowledge of the cache replacement policy and the allocation of data in memory. This information is usually not available to the compiler. As such, the compiler has to resort to applying heuristics to try and minimize the occurrence of conflict misses. In this paper, we present a probabilistic method of estimating cache conflict misses for set-associative caches. We present a set of experiments evaluating the model and discuss the implications of the experimental results.

1 Introduction

Cache conflict misses can cause severe degradation in application performance. Previous research has shown that for many scientific applications majority of cache misses are due to conflicts in cache [**?**]. Although, conflicts in cache are a major concern for application performance it is often difficult to eliminate them completely. Eliminating conflict misses requires detailed knowledge of the cache replacement policy and the allocation of data in memory. This information is usually not available to the compiler. As such, the compiler has to resort to applying heuristics to try and minimize the occurrence of conflict misses. In this paper, we present a probabilistic method of estimating cache conflict misses for set-associative caches. We present a set of experiments evaluating the model and discuss the implications of the experimental results.

2 Conflict Miss Prediction Model

We extend the cache associativity model described by Mark and Hill in [2]. We compute the probability of a cache line being evicted before it is reused based on the size and associativity of the cache and the reuse distance.

Let,

r_1 and r_2 = references to the same cache line

m = reuse distance between r_1 and r_2

s = number of sets in cache

a = associativity

If we assume, each line from m is equally likely to be mapped to any of the sets then

$$\begin{aligned} Pr[a \text{ lines landing in line occupied by } r_1] &= Pr[conflict\ miss\ on\ r1] \\ &= \sum_{i=a}^{m} \binom{m}{i} \left[\frac{1}{s}\right]^i \left[\frac{s-1}{s}\right]^{m-i} \\ &= 1 - \sum_{i=0}^{a-1} \binom{m}{i} \left[\frac{1}{s}\right]^i \left[\frac{s-1}{s}\right]^{m-i} \end{aligned}$$

Now, we introduce a tolerance term T that expresses how high a probability of a conflict miss we are willing to accept. We then have,

$$T \geq Pr[conflict\ miss\ on\ r1] = 1 - \sum_{i=0}^{a-1} \binom{m}{i} \left[\frac{1}{s}\right]^i \left[\frac{s-1}{s}\right]^{m-i}$$

From this inequality we can derive an upper bound on m for a given value of T.

$$m \leq E(a, s, T)$$

Here, $E(a, s, T)$ is the maximum integral m such that $Pr[conflict\ miss\ on\ r1] \leq T$.

Now, given a tolerance term T and the size and associativity of a cache at level k we can express our formula for *effective cache capacity (ECC)* in the following manner:

$$ECC(L_k) = E(a_k, s_k, T) \tag{1}$$

where, s_k and a_k refer to the size and associativity of the cache at level k.

Now, using Eq. 1, we can derive an upper bound for the miss rate for a given cache configuration and a tolerance term T. If the number of reused references in a program is n and all reuse distances in the program are less than the *effective cache size* then according to our model,

$$Expected\ Misses \leq nT$$

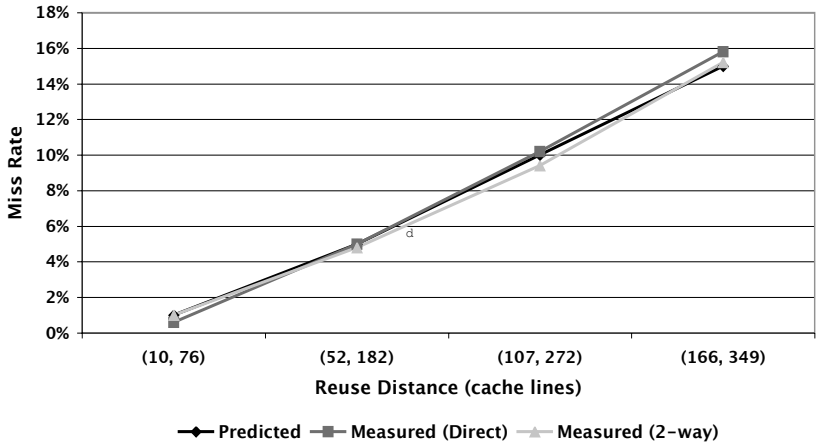

Figure 1: Accuracy of Cache Miss Model on `randaccess`

From the above we get,

$$\textit{Expected Miss Rate} = \frac{\textit{Expected Misses}}{\textit{Total Reused References}} \leq nT/n = T$$

3 Accuracy of the Model and its Implications

The cache miss model presented in the previous section makes the assumption that memory accesses between any two reused references are essentially random. In this section, we present experimental results that compare the accuracy of the miss rate predicted by our model against measured miss rates on three different programs.

We used Eq. 1 to generate *effective cache size* for a fixed size cache with varying degrees of associativity. For each cache the tolerance term T was set to four different values: 0.01, 0.05, 0.10 and 0.15. We then forced the maximum reuse distance in the program to be less than the computed *effective cache size* by picking a small enough data set size.

To get accurate measurements without interference from other architectural features we conducted experiments using a cache simulator instead of a real machine. We used the Simplescalar [1] cache simulator to simulate several different cache configurations. Here, we present and analyze results from a direct-mapped and a 2-way set associative cache both 32KB in size with 32B lines.

The first set of experiments was performed on `randaccess`, a synthetic benchmark we wrote to validate our model. In `randaccess` we iterate over an array several times. Between each iteration we access m distinct random locations in memory each of which land in a different cache line and none of which overlap with locations in the array. Thus the reuse distance for each reused reference in `randaccess` is exactly m cache lines and the reuse pattern

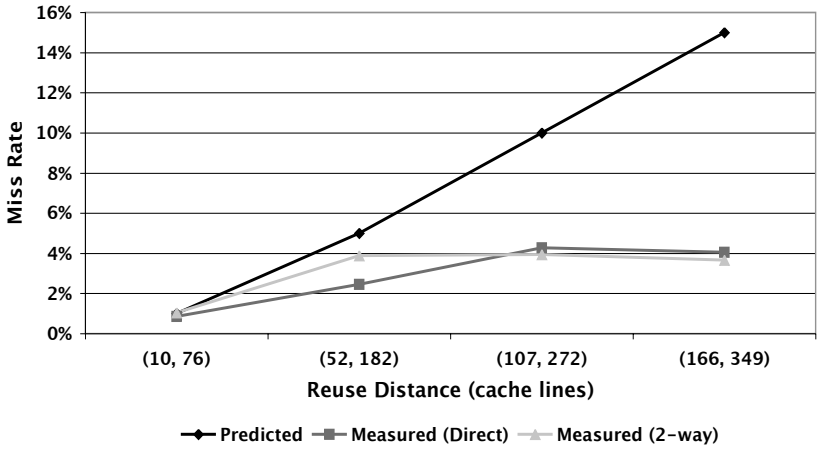

Figure 2: Accuracy of Cache Miss Model on `erlebacher`

conforms to the assumption that each cache line accessed within the reuse distance is equally likely to cause a conflict miss. Hence, for `randaccess` we expect our predicted miss rate to closely match the measured miss rate.

The results from experiments on `randaccess` are presented in Fig. 1. The measured miss rates reported in the graph are average miss rates over 100 runs of the program. The data points on the x axis correspond to the *effective cache size* computed by our model for the two cache configurations. As expected, when we enforce random access of memory locations between reuses of the same location the model is able to predict the miss rate in the program quite accurately. For both the direct-mapped and 2-way cache the difference between the predicted rate and the measured rate is never more than 1%.

We realize however, that the reuse pattern in a real application is unlikely to be totally random. Hence, we wanted to determine how much the accuracy of our model drops when the assumption of random access to memory locations is violated. We performed the second set of experiments on a C-implementation of `erlebacher`. The experimental results are presented in Fig. 2. In this case, we see that our prediction does not match the measured miss rates as closely. For smaller reuse distances the discrepancy is not that high but as we move on to larger data sets with more variability in reuse distances the difference between the predicted rate and the measured rate goes up to as high as 10%. We observe however, that the predicted miss rate is always greater than the measured miss rate. Thus as an *upper bound* for the miss rate the model still works well. In most cases, a conservative estimate of the *effective cache size* is good enough to make the right fusion choices through empirical tuning. However, in some cases a conservative estimate may lead to an underutilized cache and also have adverse effects on other transformations such as *tiling*. We address these issues later in the section.

It is difficult to predict exactly what kind of reuse pattern we will observe in

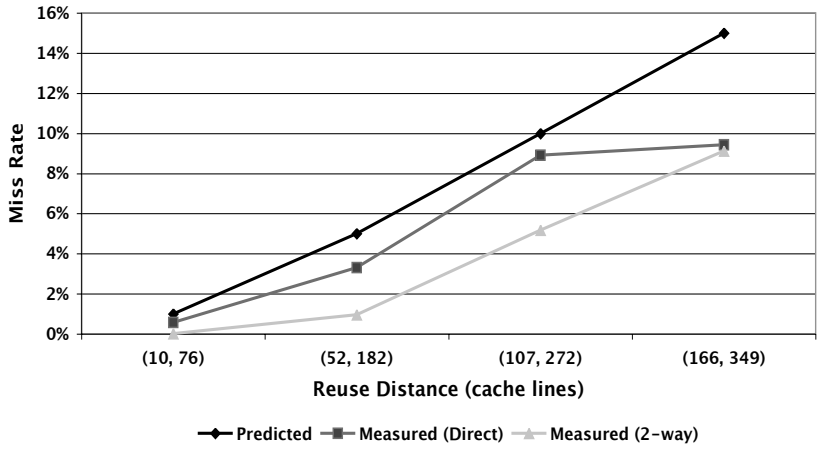

Figure 3: Accuracy of Cache Miss Model on `arraysweep`

any given program. However, for many scientific applications it can be generally summarized as a set of loops sweeping through a number of contiguous arrays. We wrote our third benchmark `arraysweep` with this reuse pattern in mind. `arraysweep` sweeps through k arrays a fixed number of times producing reuse in each of the k arrays. The inner loop is *tiled* with respect to the outer so that the reuse distance of each reference can be controlled by varying the tiling factor. For our experiments, we derive the sweep length using the following formula:

$$Sweep\ Length = Effective\ Cache\ Size/k$$

For `arraysweep` we also wanted to eliminate the effects of the compiler's allocation strategy on the conflict miss rate of a program. For this reason, we control the allocation of each array in `arraysweep` and ensure that each array starts off at a random location in memory. The experimental results with `arraysweep` using 4 arrays are presented in Fig. 3. As was the case with `erlebacher` our model *over predicts* the miss rate for `arraysweep`. However, the discrepancy between the predicted rate and the measured rate is smaller in this case. This increased accuracy in the predicted miss rate is explained by two factors. First, the reuse distance for each reference in `arraysweep` is the same number of cache lines. If there are reused references with small reuse distances that are not affected by the *effective cache size* then those references will have a positive impact on the miss rate regardless of what *effective cache size* is chosen. Thus it is likely that the inner loop carried dependences in `erlebacher` contributed to the larger gap between the predicted miss rate and the measured miss rate. The second factor that may have improved the accuracy of our prediction for `arraysweep` is the allocation strategy we used. We notice in Fig. 2 that in some cases the measured miss rate for `erlebacher` decreases slightly as we increase the data set size. One factor that could explain this phenomenon is some optimized allocation strategy

used by the compiler for the larger data sets. Forcing the arrays in `arraysweep` to start in random locations results in more predictable behavior. We observe a steady increase in the measured miss rate as the data set size is increased. Since, our model does not account for any optimizations in the allocation strategy it fares better on `arraysweep` then on `erlebacher`.

4 Conclusions

In this paper, we have presented a probabilistic model for predicting cache conflict misses for set associative caches. The experimental results suggest that our model is able to predict an upper bound for the conflict miss rate with reasonable accuracy. However, the predicted upper bound for the miss rate may be significantly greater than the actual miss rate of the program. Although a conservative estimate suffices for profitability estimates of transformations such as fusion it is important to consider its implications on other transformations. A key transformation for improving memory performance in numerical applications is *tiling*. If we use our conflict miss model with *tiling* then the *effective cache capacity* would directly determine the tile factor for a given loop nest. In that case, a conservative estimate would imply choosing a smaller tile size which in turn may lead to lost reuse in inner loops. Therefore, in such situations we need a cache miss model that is able to predict the cache miss rate more accurately. We are currently working on such a model. Our new model incorporates the effects of *tiling* and also considers the layout of arrays in memory.

References

[1] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical report, Department of Computer Science, University of Washington, June 1997.

[2] M. D. Hill and A. J. Smith. Evaluating associativity in cpu caches. *IEEE Trans. Comput.*, 38(12), 1989.

[3] K. S. McKinley and O. Temam. Quantifying loop nest locality using SPEC'95 and the Perfect benchmarks. *ACM Transactions on Computer Systems*, 17(4):288–336, 1999.