Introduction

The efficiency of a computer system in executing a program is highly dependent on the effectiveness of the memory hierarchy to supply requested data quickly. However, since the structure of the processor's memory hierarchy is almost always fixed, the overall efficiency depends on how well the program reference pattern matches the given cache structure. Characterizing cache behavior has generally taken the form of varying the cache characteristics and measuring behavior for a given program with a particular data input set. The architecturally defining features of a cache designare its size, associativity, and cache line (block) size

A fourth dimension little discussed is how cache behavior for a given cache configuration (size, associativity, and line size) varies as the program data set varies. . The three-dimensional space varies cache size on one axis, associativity on another, and program data set size on a third, for a fixed line size. There are many works on cache characterization techniques that address how to quickly explore the planar space i.e, the size and associativity axes.We focused on understanding a work related to exploration along the program data set size in this project. The method proposed in that work utilizes *data reuse signature patterns*, a fundamental quality of program behavior .

Data reuse signature patterns are defined by measurements of a program's *data reuse distance*. In sequential execution, reuse distance is the number of distinct data elements accessed between two consecutive references to the same element. The reuse distance is a measure of the capacity that a fully associative cache must have for the subsequent access to hit in the cache. Thus, reuse distance accurately describes access behavior to a fully associative cache. The reuse pattern of cache blocks are analyzed as a prestep for cache miss prediction.

This method predicts program miss rates across a wide range of program data set and cache sizes. This technique uses reuse distance information from *two* input data sets of different sizes, then extracts a parameterized model of the program's fundamental data reuse pattern.. Then the data reuse distance information is converted into cache miss rates for any input size and for any size of *fully associative cache* .

Along with the model they also built a visualisation tool that uses a three dimensional plot to show miss rate variation across program input sizes ,cache sizes and cache associativity. This tool identifies critical input data set sizes where the miss rate changes dramatically for a given cache. Second, these critical input data sizes can be well beyond the size of some of the available benchmark reference data sets; thus, the critical data set sizes are unlikely to be discovered by profiling or runtime sampling methods. The predicted miss rate curves provide insight into program behavior that may not be practical to generate by any other means.

Why we have chosen Cache Miss-Rate Prediction?

Today’s computing centers often use thousands of processors to run a few large applications. Rather than having to simulate numerous inputs and extrapolating how the system would perform from those runs, designers could save both time and money by being able to see the change in miss rates as cache size changes.Miss -rate prediction facilitates evaluating the cache performance of data sets too large to simulate on any existing machine.With accurate estimates of cache performance across numerous inputs, benchmark design may be improved to allow faster evaluation of systems and optimizations.

Cache Miss Rate Estimation

In this method program locality is based on the concept of *reuse distance*. Reuse distance is independent of the granularity of data elements and   each distinct cache block is treated as a basic data element.This approach to predict the cache miss ratio for a given program across different data inputs consists of two main steps. The first step is to model the program locality as predictable reuse distance patterns; the second step is to estimate the miss rate according to the obtained patterns.

Reuse Distance

The reuse distance of a reference is defined as the number of distinct memory references between itself and its reuse.In this method the whole program is represented as a histogram describing reuse distance distribution, where each bar consists of the portion of memory references whose reuse distance falls into the same range.The data size of an input is defined as the largest reuse distance .Given two histograms with different data sizes, the locality patterns of a third data size are predictable in a selected set of benchmarks. The pattern recognition step generates the histogram
for the third input.

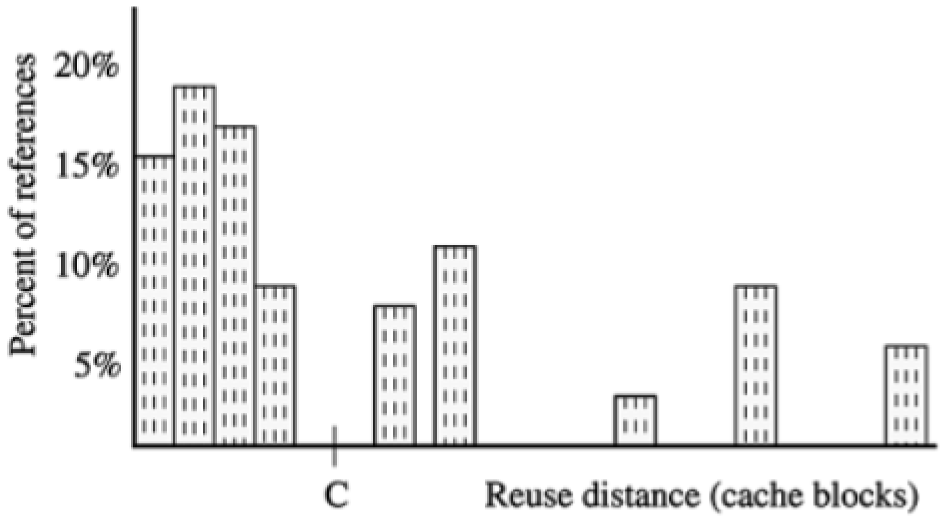
Identifying Locality Pattern

The reuse histogram depicts the locality behavior of a program showing the percentage of memory accesses with a given reuse distance. Pattern recognition is used to discover the parameterized general model for program locality behavior from the reuse histograms generated during profiling.

2.a.1 Categorizing into Reference Groups

The reuse distance histogram is composed of references to cache blocks that often exhibit a pattern. The pattern might be a constant reuse distance, e.g., sequential accesses to a block while stepping through an array. The histogram can include patterns that are proportional to the data set size, where the data set size is the number of unique cache lines accessed by the program.The histogram is an aggregation of all accesses and, thus, includes many types of patterns. The modeling technique classifies portions of the histogram into groups and models each group by a function type.

The difficulty is how to tease apart which references are represented by the different function types. In this technique, the reuse histograms from two runs of the program having two different sizes of data sets are compared. For each histogram *1000* groups are formed by assigning 0.1 percent of the histogram to a group, starting from the shortest reuse distance and moving toward the largest. The dotted lines in qualitatively show a partitioning. An implicit assumption is that references of like reuse distance can be modeled by a similar function type.



Estimating reuse distance functions for each group. (a) Partitionings histogram into groups. (b) Pairing groups.

We justify this pairing of groups with observations that reuse patterns are a function of data size, *s*, and this component dominates in determining the reuse distance. For a nontrivial data size, the result is that the groups become sorted from left to right by functions of increasing power of *s*, e.g., reuse distance increases more rapidly for groups based on *s* than for those based on $\sqrt{s}$.

If we show three groups and the reuse distance histogram for two runs. In the second run, the movement of the groups is shown relative to the first run. The estimated function types are also shown. The first group, g_i, does not move between the runs, so its reuse distance is independent of the data set size (*constant*). On the other hand, g_j has a square root relationship to the data size, $O(\sqrt{s})$ and g_k has a linear relationship, $O(s)$. Dividing the histogram into 1, 000 groups limits the error from any single estimated model. The final model is a parameterized set of 1, 000 functions.

Identifying Individual Patterns

Given two histograms with different data sizes, the pattern recognition step constructs a formula for each group of references. Let d1i be the distance of the ith group in the first histogram, d2i be the distance of the ith group in the second histogram, s_1 be the data size of the first run, and s_2 the data size of the second run. The pattern for this group is a linear fitting based on the data size. Specifically, we want to find the two coefficients, c_i and e_i, that satisfy the following two equations:

$d1i=ci+ei*fi(s1).(1)$

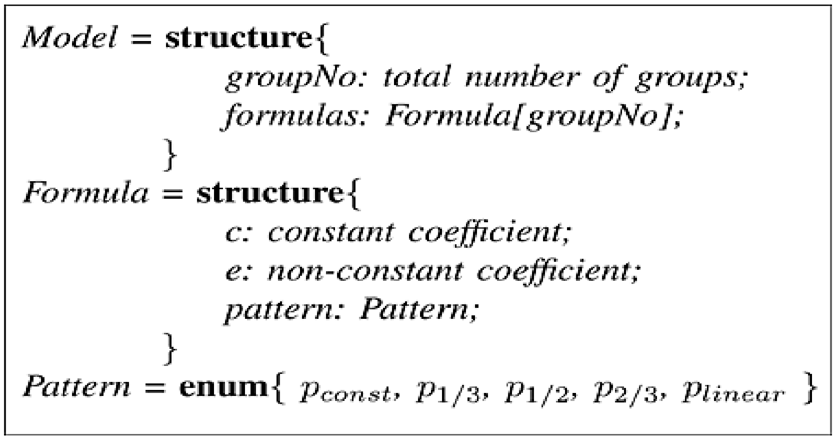
$d2i=ci+ei*fi(s2).(2)$

Assuming the function fi is known, the two coefficients uniquely determine the distance for any other data size. The pattern is more accurate if more profiles are collected for the linear fitting. The minimal number of inputs is two.

The formula is based on an important fact about reuse distance: In any program, the largest reuse distance cannot exceed the size of program data. Therefore, the function fi can be linear at most, so the pattern is a linear or sublinear function of data size and not a general polynomial function. Following are the choices for fi : The first is pconst(s) = 0. A constant formula pattern because reuse distance does not change with data size. The second is plinear(s) = s, a linear pattern. The constant and linear patterns are the lower and upper bounds of the distance patterns. Between them are sublinear patterns.For each group pair (g1i,g2i), we calculate the ratio of their average distance, d2i/d1i, and pick fi to be the pattern function, p, such that p(s2)/p(s1) is closest to d2i/d1i.

According to the regression theory, more data can reduce the effect of noise and reveal a pattern closer to the true pattern . Using more than two training inputs in the analysis may produce a better prediction because it reduces the noise from imprecise reuse distance measurement and reference histogram construction .The extension is straightforward. For each input, there is an equation as shown. For each bin, instead of two linear equations for two unknowns, there are as many equations as the number of training runs. Although more training data can lead to better results, they also lengthen the profiling process.

$$di=ci+ei\ast fi(s)$$

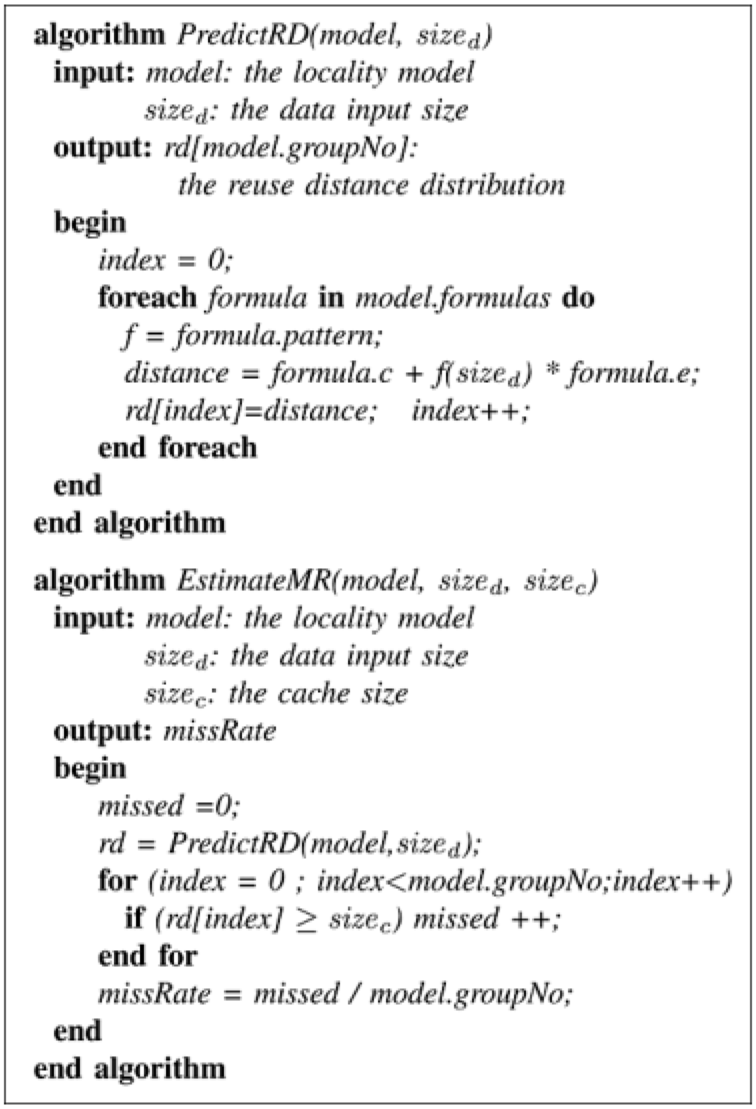


```
Model = structure{
                groupNo: total number of groups;
                formulas: Formula[groupNo];
          }
Formula = structure{
                c: constant coefficient;
                e: non-constant coefficient;
                pattern: Pattern;
          }
Pattern = enum{ $p_{const}$, $p_{1/3}$, $p_{1/2}$, $p_{2/3}$, $p_{linear}$ }
```

Here p1/3,p1/2,p2/3 are sub linear functions of s .ex. p2/3(s)=s^⅔.

Cache Miss Rate Estimation from predicted reuse distances

After generating the locality model of a program, locality behavior for any data input size *s can be predicted* . The reuse distance for each group *g_i* can be predicted via their particular formula. The overall reuse distance distribution is the aggregation of the reuse distance of each group. For any given size of a fully associative LRU cache, the capacity miss rate is directly estimated from the reuse distance distribution. The groups with a reuse distance larger than or equal to the given cache size will be capacity misses. The number of these groups gives an estimate of the miss rate. The detailed algorithms for reuse distance prediction and cache miss rate estimation are described as follows

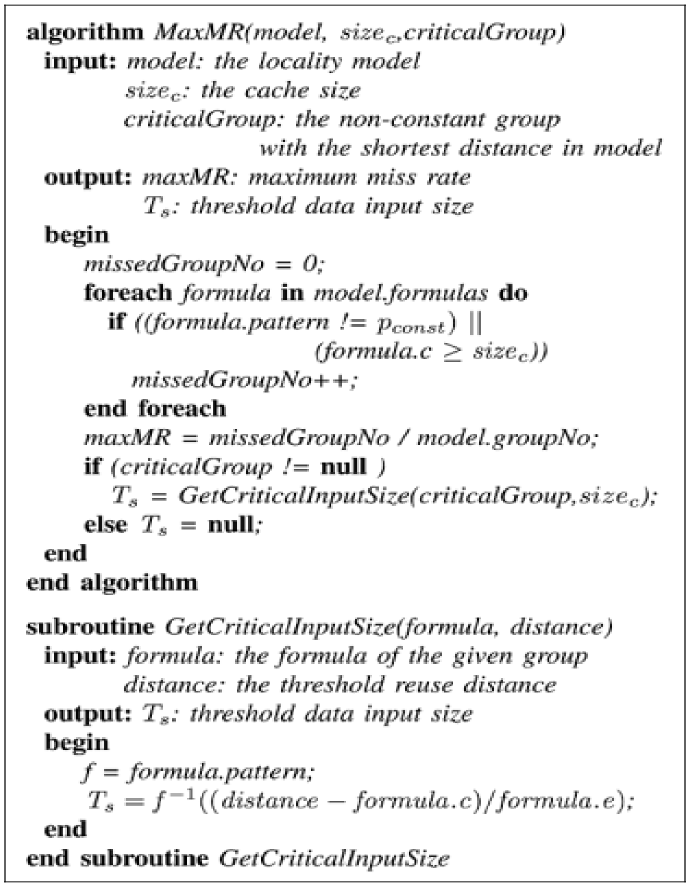


```
algorithm PredictRD(model, $size_d$)
  input: model: the locality model
         $size_d$: the data input size
  output: rd[model.groupNo]:
            the reuse distance distribution
  begin
      index = 0;
      foreach formula in model.formulas do
        f = formula.pattern;
        distance = formula.c + f($size_d$) * formula.e;
        rd[index]=distance;    index++;
      end foreach
  end
end algorithm

algorithm EstimateMR(model, $size_d$, $size_c$)
  input: model: the locality model
         $size_d$: the data input size
         $size_c$: the cache size
  output: missRate
  begin
      missed =0;
      rd = PredictRD(model,$size_d$);
      for (index = 0 ; index<model.groupNo;index++)
        if (rd[index] $\geq$ $size_c$) missed ++;
      end for
      missRate = missed / model.groupNo;
  end
end algorithm
```

From our locality model, we can also derive the maximum possible miss rate of a program for a given cache size. Here, we consider two important properties of the reuse distance model. First, only references with a reuse distance larger than or equal to the cache size are cache misses. References with a shorter reuse distance will hit in cache because of temporal locality. The second property is that reuse distances of references having linear or sublinear patterns monotonically increase with the input size. The distances of references having a constant pattern, on the other hand, are independent of the input size. From these two facts, we can infer that, for a certain cache size, if we increase the program data input size, the cache miss rate may remain the same or increase, but it will never decrease. Furthermore, the maximum miss rate will be reached when the input size is large enough so that all references with a nonconstant pattern have a reuse distance larger than the cache size.

Figure below shows the main algorithm to predict the maximum cache miss rate and the corresponding threshold input size for a given cache of *size$_c$* blocks. Suppose, among the total *G* groups (*G = 1, 000* for all our results), the number of reference groups with a pattern dependent on data size *s* is *G$_s$* and the number of groups having a pattern independent of data size (constant) with a fixed reuse distance greater than the capacity of the cache is



```
algorithm MaxMR(model, $size_c$,criticalGroup)
  input: model: the locality model
         $size_c$: the cache size
         criticalGroup: the non-constant group
                 with the shortest distance in model
  output: maxMR: maximum miss rate
          $T_s$: threshold data input size
  begin
     missedGroupNo = 0;
     foreach formula in model.formulas do
        if ((formula.pattern != $p_{const}$) ||
                         (formula.c $\geq$ $size_c$))
          missedGroupNo++;
     end foreach
     maxMR = missedGroupNo / model.groupNo;
     if (criticalGroup != null )
        $T_s$ = GetCriticalInputSize(criticalGroup,$size_c$);
     else $T_s$ = null;
  end
end algorithm

subroutine GetCriticalInputSize(formula, distance)
  input: formula: the formula of the given group
         distance: the threshold reuse distance
  output: $T_s$: threshold data input size
  begin
     f = formula.pattern;
     $T_s = f^{-1}((distance - formula.c)/formula.e)$;
  end
end subroutine GetCriticalInputSize
```

Fig. 5. Max miss rate and threshold input prediction.

G$_c$, the maximum miss rate is calculated as *Miss Rate$_{max}$ = (G$_s$ + G$_c$)/G*.

The *threshold input size (T$_s$)* is the smallest input size in which the cache miss rate reaches the maximum value for the program and the given cache configuration. To estimate its value, we only need to consider the single non constant reference group having the shortest reuse distance. The maximum miss rate occurs when this group, *g$_j$*, has a reuse distance greater than or equal to the cache size, *$d_j \geq size_c$*.We can directly calculate the required input data size from this condition:

$$T_s=f_{(-1)j}((C-c_j)/e_j).(4)$$

If the model only contains the constant pattern, the cache miss rate is the same for all inputs. In this case, e_j is zero and generates a meaningless null threshold input. The threshold input size T_s is useful since it is the smallest input that generates the worst case hit ratio for a fully associative cache.

*Model accuracy verification:*Only two input sets are necessary to generate a miss rate model for a program. However, if more input sets are measured, multiple models can be generated from pairwise groupings and the predictions evaluated for the excluded data set sizes. For example, for three data set sizes, *A*, *B*, and *C*, three combinations of pairs are possible, *(A, B)*, *(A, C)*, and *(B, C)*. The data sets used for verification would be *C*, *B*, and *A*, respectively. Thus, a degree of confidence for the model can be measured with limited exploration of the data set space.

References

1)Y. Zhong, S. Dropsho, and C. Ding.“Miss Rate Prediction Across Program Inputs and Cache Configurations”,IEEE TRANSACTIONS ON COMPUTERS, VOL. 56, NO. 3, MARCH 2007.

2)Changpeng Fang , Steve Carr ,Soner Onder and Zhenlin Wang” Reuse-distance-based Miss-rate Prediction on a Per Instruction Basis”

3)C. Ding and Y. Zhong, “Predicting Whole-Program Locality with Reuse Distance Analysis,” Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation, June 2003.