Regression-Based Multi-Model Prediction of Data Reuse Signature

Xipeng Shen Yutao Zhong Chen Ding

Computer Science Department, University of Rochester
{*xshen,ytzhong,cding*}*@cs.rochester.edu*

Abstract

As a locality metric, the distance of data reuses has been used in designing compiler, architecture, and file systems. Recently, Ding and Zhong described a method that predicts reuse distance histograms across all inputs of a program. In this paper we extend their method in two ways. First, we consider more than two training inputs using regression analysis. Second, we use a method called multi-model prediction to overcome the limitation due to small training inputs or coarse-grain data collection. Compared to Ding and Zhong's method, the new locality prediction can reduce about half of the prediction error, remove 95% of space cost, and use much smaller inputs and faster data collection in training.

1 Introduction

As the speed gap between CPU and main memory widens, memory hierarchy has become increasingly important in determining system performance, cost, and energy consumption. Cache performance depends on data locality in programs. A measure of locality is LRU stack distance or *reuse distance*, which is the number of distinct elements accessed between two consecutive uses of the same datum [17, 11]. Reuse distance information has been used in managing various layers of the memory system including register allocation [15], cache optimization [2, 5, 6, 22], server caching [24], and file caching [14]. Recently, Ding and Zhong shows that the reuse distance histogram, or *reuse signature*, of many programs has a consistent pattern across different inputs [11]. The pattern is a parameterized formula that for a given program input, it predicts the reuse signature for the corresponding execution. The pattern summarizes program locality and may have many uses. Zhong et al. showed that a precise signature pattern can accurately predict cache miss rates across all program inputs [23].

The pattern analysis method given by Ding and Zhong has two important limitations. First, it uses only two training runs and therefore may be misled by noises from specific executions. Second, the accuracy is limited by the precision of data collection. Accurate prediction requires using large size program inputs and fine-grained reuse distance histograms. The space and time cost of the analysis is consequently high, which makes the analysis slower and prohibits simultaneous analysis of different patterns, for example, patterns of individual data elements.

This paper presents a new set of techniques that overcome these limitations in two ways. First, we use regression to extract signature pattern from more than two training runs. Second, we employ multiple models (defined later). Next we describe the basic prediction method and the main factors affecting its accuracy.

1.1 Basic Prediction Method

Given an input of a program, we measure the locality of the execution by the histogram of the distance of all data reuses. We call it interchangeably as *reuse distance histogram* or *reuse signature* of the program execution (see Section 2 for formal definitions). The prediction method by Ding and Zhong uses a training step to construct a pattern by running two different inputs of a program. Let s and $\hat{s}$ be the sizes of the two input data. For each of the reuse distance histogram, the analysis forms 1000 groups by assigning 0.1% of all memory accesses to each group, starting from the shortest reuse distance to the largest. We denote the two sets of 1000 groups as $\langle g_1, g_2, \cdots, g_{1000}\rangle$ and $\langle \hat{g}_1, \hat{g}_2, \cdots, \hat{g}_{1000}\rangle$ and denote the average reuse distances of g_i and $\hat{g}_i$ by rd_i and $\hat{rd}_i$ respectively ($i = 1, 2, \cdots, 1000$.) Based on rd_i and $\hat{rd}_i$, the analysis classifies group i as a constant, linear, or sub-linear pattern. Group i has a constant pattern if its average reuse distance stays the same in the two runs, i.e. $rd_i = \hat{rd}_i$. Group i has a linear pattern if the average distance changes linearly with the

change in program input size, i.e. $\frac{rd_i}{\hat{rd}_i} = c + k\frac{s}{\hat{s}}$, where c and k are both constant parameters. Ding and Zhong measured the size of input data through distance-based sampling [11]. We use the same sampling method in this work.

After the training step, the reuse signature for another input can be predicted by calculating the new distance for each group according to its pattern. Interested reader can find a more detail discussion of this process in Ding and Zhong's paper [11]. Figure 1 shows the flow diagram of their prediction method. We will explain the different types of histograms in Section 2.

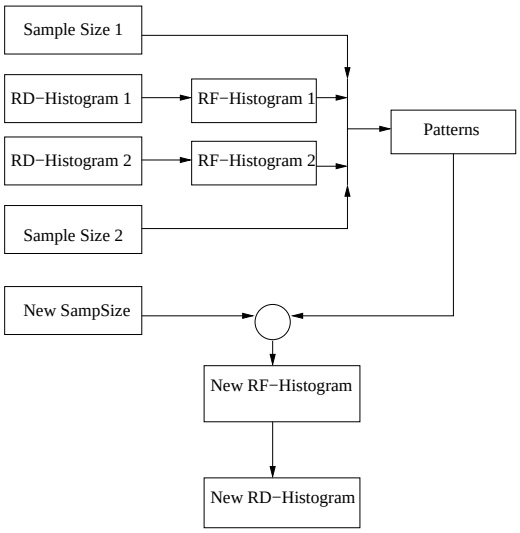


Figure 1: The flow diagram of Ding and Zhong's prediction method, which uses only two training inputs. A RD-Histogram is a reuse-distance histogram, and a RF-Histogram is a reference histogram. Sample size is the estimated input data size by sampling.

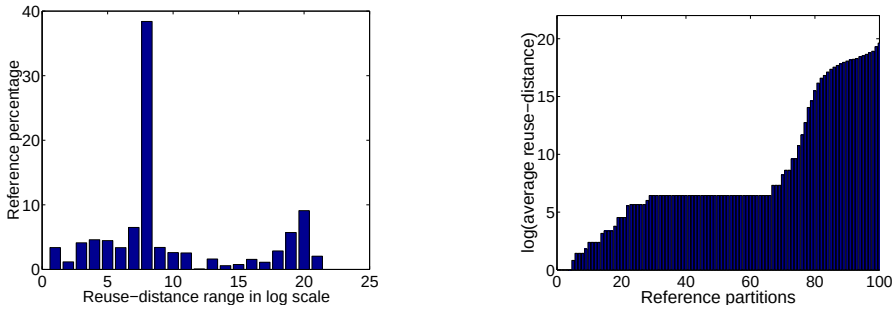
Note that not all programs have a consistent pattern, and not all patterns are predictable. However, Ding and Zhong showed that their method can find predictable patterns in a wide range of large, complex programs. The goal of this work is to improve the analysis accuracy and efficiency for programs that have a predictable pattern.

1.2 Factors Affecting Prediction Accuracy

Three factors strongly affect the prediction accuracy: the number of training inputs, the precision of data collection, and the complexity of patterns. The number of training inputs needs to be at least two, although using more inputs may allow more precise recognition of common patterns. The precision of data collection is determined by the number of groups. Since each group is represented by its average reuse distance, the more groups the analysis uses, the more precise the reuse distance information is. However, using more groups leads to slower pattern recognition and prediction since the space and time costs are proportional to the number of groups. The third factor is the complexity of patterns in each group. If we assume that the entire group has a single pattern, the analysis is a *single-model* prediction. If we assume that the group may consist of different subgroups that each may have a different pattern, the analysis is a *multi-model* prediction.

Single-model prediction has two limitations. First, the accuracy of the prediction is strictly limited by the precision of data collection, i.e. the number of groups. A large group tends to include subgroups with different patterns, which breaks the single-model assumption and causes low prediction accuracy. Second, training runs need to have a sufficient size so that the range of reuse distances in different patterns can be well separated. The larger the input data size is, the more likely different patterns is separated. If the distances of two patterns are similar, they fall into the same group, and the prediction cannot tear them apart. Because of the need for large training inputs and number of groups, single-model prediction usually incurs a large time and space cost. Multi-model prediction, however, may overcomes these two limitations by allowing sub-portions of a group to have a different pattern.

The paper presents three extensions to the basic method. The first is single-model prediction using more than two training runs. The next is a set of multi-model prediction methods using different types of reuse distance histograms. It



(a) (b)

Figure 2: Reuse-distance Histogram of *SP* with input of size 28^3. (a) distance histogram (b) reference histogram

reduces the space cost from $O(M)$ to $O(logM)$, where M is the size of program data. The last is a strategy for choosing the appropriate histograms based on analytical and experimental results.

The rest of the paper is organized as follows. Section 2 describes the types of histograms used in our prediction. Section 3 and 4 describe the new regression-based multi-model methods. Followed are the experiment results and discussions. Section 6 discusses related work, and Section 7 summarizes our findings.

2 Terminology

This section explains the two types of histograms and related terms used in later discussions.

- **A Reuse-distance Histogram (RD-Histogram):** X-axis gives reuse-distance ranges, and Y-axis gives the percentage of data accesses in each distance range. The size of distance ranges can be in linear scale, e.g. $[0k, 1k), [1k, 2k), [2k, 3k), \cdots$, or log scale, e.g. $[0k, 1k), [1k, 2k), [2k, 4k), [4k, 8k), \cdots$, or mixed linear and log scales. Figure 2(a) shows the reuse-distance histogram of SP in log scale ranges.
- **A Reference Histogram (RF-Histogram):** X-axis is the groups of data accesses, sorted by the average reuse distance. The Y-axis is the average reuse distance of each partition. Figure 2(b) is the reference histogram of SP for a partition of 100 groups. A reference histogram can be viewed as a special type of reuse-distance histogram whose distance ranges have non-uniform lengths so that each range holds the same number of program data accesses.

The purpose of a reference histogram is for the trade-off between information loss and computation/space efficiency. For dense regions in the reuse distance histogram, where a large portion of memory accesses have similar reuse distances, the reference histogram uses short range to increase accuracy. For sparse regions in the reuse distance histogram, the reference histogram uses large ranges to reduce the total number of ranges.

3 Single-Model Multi-Input Prediction

Using more than two training inputs may produce a better prediction, because it reduces mainly two kinds of noises. One is brought by the reuse distance measurement. Ding and Zhong used approximation to trade accuracy for efficiency [11]. The approximation brings errors to the reuse distance histogram. The second kind of noise is brought by the input size. Although distance-based sampling [11] finds a size reflecting the size of a program input, the sampled size is not always accurate. These noises reduce the accuracy of the prediction.

According to the regression theory, more data can reduce the effect of noises and reveal a pattern closer to the real pattern [19]. Accordingly, we apply a regression method on more than two training inputs. The extension is straightforward. For each input, we have an equation as follows.

$$d_i = c_i + e_i * f_i(s) \quad (1)$$

where d_i is the average reuse distance of i^{th} reference group when the input size is s, c_i and e_i are two parameters to be determined by the prediction method, and f_i is one of the following functions of s:

$0;\ \ s;\ \ s^{1/2};\ \ s^{1/3};\ \ s^{2/3}$

Given the histograms of two training runs, Ding and Zhong could solve a linear equation, determine the two unknowns for each group, and calculate the reuse distance histogram for a new input given its input size. Using two training inputs is sufficient because there are only two unknowns in each model (Equation 1).

While the previous method has two equations, we have more than two equations because of more training inputs. We use *Least square regression* [19] to determine the best values for the two unknowns. We use 3 to 6 training inputs in our experiment. Although more training data can lead to better results, they also lengthen the profiling process. We will show that a small number of training inputs is sufficient to gain high prediction accuracy.

4 Multi-Model Prediction

A multi-model method assumes that memory accesses in a group can have different models. For example in a histogram, a bar (group) may contain both a constant model and a linear model.

Figure 3 gives a graphical illustration of the multi-model prediction. We arbitrarily pick one of the training inputs as the *standard input*. In this example, s_0 is the size of the standard input (the other training inputs are not showed in the figure.) Its reuse distance histogram, called *standard histogram*, has 12 groups, and each group consists of two models—constant and linear models. Together, they form the histogram of s_0. Using regression technique on all training histograms, the standard histogram in Figure 3(a) is decomposed into constant and linear models in Figure 3(b) and 3(c). The decompose process is described below. During prediction process, the two histograms change to Figure 3(d) and 3(e) respectively according to the size of the new input $8 * s_0$. Constant histogram keeps unchanged, and the distance of each data in linear histogram increases to 8 times long. The X-axis is in log scale, so each bar in linear histogram moves 3 ranges right-toward. The reuse distance histogram for the new input is the combination of the new constant and linear histograms, see Figure 3(f).

Formally, the reuse distance function of a group is as follows.

$$h_i(s) = \varphi_{m_1}(s, i) + \varphi_{m_2}(s, i) + \cdots + \varphi_{m_j}(s, i) \quad (2)$$

where, s is the size of input data, $h_i(s)$ is the Y-axis value of the i^{th} bar/group for input of size s, and $\varphi_{m_1}...\varphi_{m_j}$ are the functions corresponding to all possible models.

Each $h_i(s)$ can be represented as a linear combination of all the possible models of the standard histogram:

$\varphi_{m_1}(s_0, 1), \varphi_{m_1}(s_0, 2), \cdots, \varphi_{m_1}(s_0, G), \varphi_{m_2}(s_0, 1), \varphi_{m_2}(s_0, 2), \cdots, \varphi_{m_2}(s_0, G),$
$\cdots, \varphi_{m_j}(s_0, 1), \varphi_{m_j}(s_0, 2), \cdots, \varphi_{m_j}(s_0, G)$

where, G is number of groups in standard histogram.

For example, a program has both constant and linear patterns. For easy description, we assume:

range 0: [0,1); range 1: [1,2); range 2: [2,4); range 3: [4,8), $\cdots$

For another input of size $s_1 = 3 * s_0$, we calculate the Y-axis value of range $[4, 8)$ as follows:

$h_3(s_1) = \varphi_0(s_0, 3) + \varphi_1(s_0, r)$

where, r is range $[\frac{4}{3}, \frac{8}{3})$. This is because the constant model of range $[4, 8)$ in the standard histogram gives entire contribution, and the linear model of $h_3(s_1)$ comes from the linear portions in range $[\frac{4}{3}, \frac{8}{3})$ of standard histogram. $\varphi_1(s_0, r)$ can be calculated as follows:

$\varphi_1(s_0, r) = \varphi_1(s_0, r_1) + \varphi_1(s_0, r_2)$

where, $r_1 = [\frac{4}{3}, 2)$ and $r_2 = [2, \frac{8}{3})$.

We assume the reuse distance has uniform distribution in each range. Thus,

$$\varphi_1(s_0, r_1) = (\frac{2-4/3}{2-1})\varphi_1(s_0, 1) = \frac{2}{3}\varphi_1(s_0, 1)$$
$$\varphi_1(s_0, r_2) = (\frac{8/3-2}{4-2})\varphi_1(s_0, 2) = \frac{2}{3}\varphi_1(s_0, 2)$$

Finally, we calculate $h_3(s_1)$ as follows:

$$h_3(s_1) = \varphi_0(s_0, 3) + \frac{2}{3}\varphi_1(s_0, 1) + \frac{2}{3}\varphi_1(s_0, 2)$$

After we represent each $h_i(s)$ of all training inputs in the above manner, we obtain an equation group. The unknown variables are the models in the standard histogram. The equations correspond to the groups in all training histograms. Regression techniques can solve the equation group. This completes the decomposition process and also completes the construction of reuse distance predictor. During prediction process, for any input, each of its reuse distance group can be calculated as a linear combination of standard histogram models in the same manner as in decomposition process. Then, its reuse distance histogram can be obtained by the combination of all the groups.

One important assumption is that the percentage of memory accesses in each model remains unchanged for different inputs. There is no guarantee that this is the case, although Ding and Zhong showed that it is an acceptable assumption for a range of programs including those used in this paper.

A multi-model method does not depend on the type of histograms. It can use distance histograms with log or $linear$ size groups. It can also use reference histograms. The equations are constructed and solved in the same manner.

We now describe three methods of multi-model prediction. They differ by the type of reuse distance histograms. The first two methods use reuse distance histograms with log and log-$linear$ scale ranges respectively. The first 13 ranges in the log-$linear$ scale is in log scale (power-of-two) and the rest have length 2048. The purpose of the log part is to distinguish groups with short reuse distances. The space and time cost of the second method is $O(M)$, where M is the size of program data in training runs. The space and time cost of the first method is $O(logM)$, which saves significant space and time because it has much fewer equations and variables. However, the linear scale has higher precision, which can produce better results especially when using small size training runs.

The third multi-model method uses a reference histogram, for example, with 1000 groups. Unlike the first two methods, in this method, the number of equations and variable is the same as the number of groups. We can choose an arbitrary number. This provides freedom but also raises a problem: how to choose the best number of groups. In fact, the last method represents as many methods as the maximal number of groups, which is $O(N)$, where N is the number of memory accesses in the smallest training run. We will see in the evaluation section that the prediction accuracy depends heavily on the choice of groups.

5 Evaluation

5.1 Experimental Setup

We compare five prediction methods: the single-model two-input method given by Ding and Zhong, the single-model multi-input regression described in Section 3, and the three multi-model methods described in Section 4. The multi-input methods use 3 to 6 training inputs. We measure accuracy by comparing the predicted histogram with the measured histogram for a test input. The definition of accuracy is the same as Ding and Zhong's [11]. Let x_i and y_i be the size of ith groups in the predicted and measured histograms. The cumulative difference, E, is the sum of $\mid y_i - x_i \mid$ for all i. The accuracy A is $(1 - E/2)$, which intuitively is the overlap between the two histograms.

Table 1 lists the names of six test programs, their descriptions, and the sources. Table 2 and Figure 4 show the accuracy of the five approaches on six benchmarks when training and testing inputs are large. In the table, Max. Inputs Num. is the maximal number of inputs among all the five methods for each benchmark. In our experiment, for each benchmark, the size of the biggest training input is the same for all five methods. This is to make the comparison fair.

5.2 Results on Large Inputs

Using a large input has two benefits. First, different models stay separated from each other. For example, suppose constant and linear models co-exist in a range r when the input size is s_0. For a larger input whose size is $1024 * s_0$, the linear model will move far out of range r, but constant model remains unchanged. Thus, there will not be an overlap of models. The separation of models is important for the two single-model methods since they assume that only one model exists in each range. The second benefit is that the percentage of individual models is more likely to remain constant when the input size is large. This is required by both single-model and multi-model based methods.

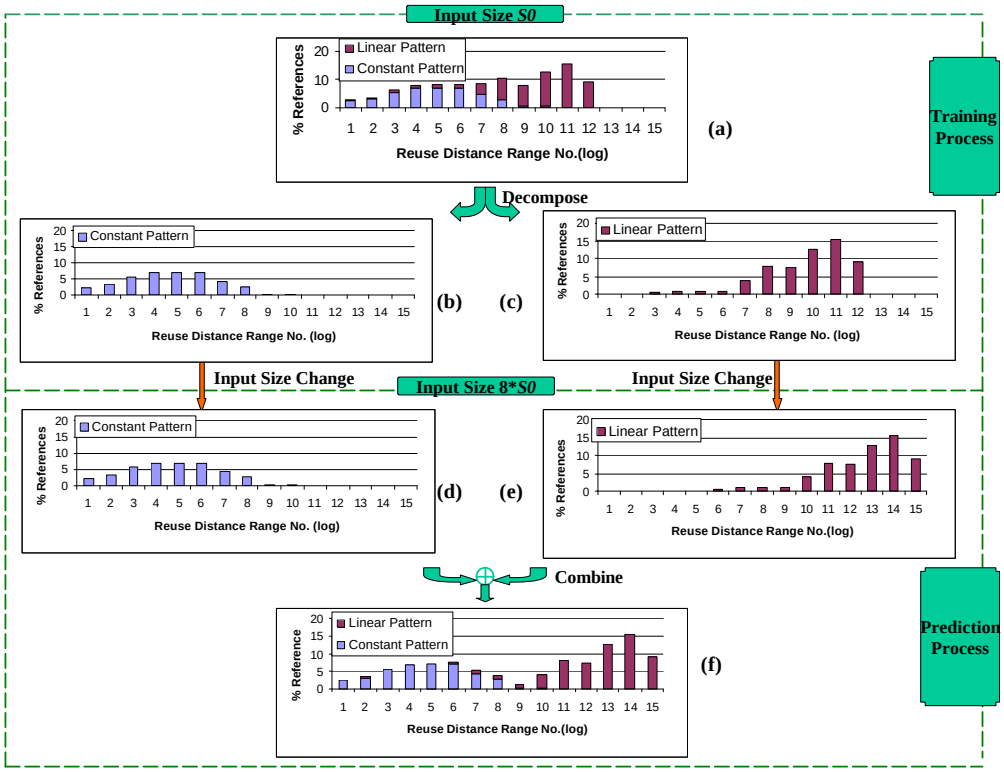


Figure 3: An example for multi-model reuse signature prediction. Figure (a) is the reuse distance histogram of the execution on standard input s_0. By using regression technique on all training histograms, the standard histogram is decomposed into two histograms—constant and linear histograms in Figure (b) and (c). During prediction process, the two histograms change to Figure (d) and (e) respectively according to the size of the new input $8 * s_0$. Constant histogram keeps unchanged, and the distance of each data in linear histogram increases to 8 times long. The X-axis is in log scale, so each bar in linear pattern moves 3 ranges right-toward. The reuse distance histogram for the new input is the combination of the new constant and linear histograms, showed in Figure (f).

Table 1: Six benchmark programs

Benchmark	Description	Suite
Applu	solution of five coupled nonlinear PDE's	Spec2K
SP	computational fluid dynamics (CFD) simulation	NAS
FFT	fast Fourier transformation	
Tomcatv	vectorized mesh generation	Spec95
GCC	based on the GNU C compiler version 2.5.3	Spec95
Swim	finite difference approximations for shallow water equation	Spec95

Table 2: Comparison of prediction accuracy by five methods

Bench-mark	Single Model		Multi-model			Max. Inputs Num.∗
	2 inputs RF-Hist.	>2 inputs RF-Hist.	Log RD-Hist.	log-Linear RD-Hist.	Fixed RF-Hist.	
Applu	70.49	97.40	93.65	93.90	90.83	6
SP	91.08	96.69	94.20	94.37	90.02	5
FFT	73.28	93.30	93.22	93.34	95.26	3
Tomcatv	92.32	94.38	94.70	96.69	88.89	5
GCC	98.50	97.95	98.83	98.91	93.34	4
SWIM	93.89	94.05	84.67	92.20	72.84	5
Average	**86.59**	**95.63**	**93.21**	**94.90**	**88.53**	**4.7**

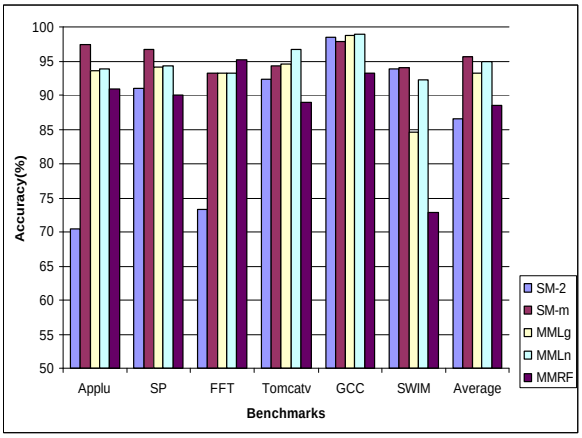


Figure 4: Prediction accuracy shown as a bar graph.

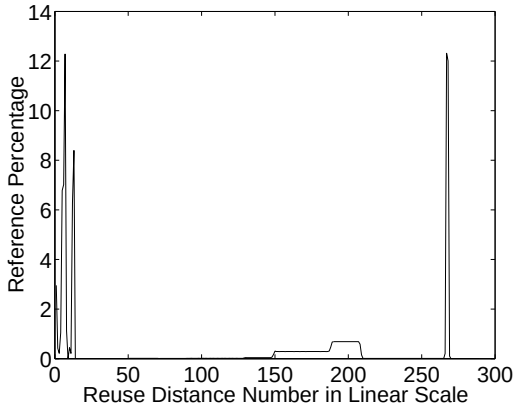


Figure 5: The reuse distance histogram curve of *SWIM*

The first column of Table 2 gives results obtained by Ding and Zhong's original method. Other columns show the accuracy of the new methods. All methods are based on histograms given by the same reuse-distance analyzer and the input sizes given by the same distance-based sampler. The numbers of the first column is slightly different from Ding and Zhong's paper [11], because they used a different reuse-distance analyzer than we do. Different analyzers lose precision in slightly different ways. The sampled input size is also slightly different because of this. From Table 2, we make the following observations:

- For most programs, all four new approaches produce better results than Ding and Zhong's method. Therefore, regression on multiple inputs indeed improves prediction accuracy.
- Except for *SWIM*, multi-model logarithmic scale method is comparable to the best predictors, although it uses 95% less storage space in most analysis. It is the most efficient among all methods.
- The performance of multi-model log-linear scale method is slightly better than multi-model logarithmic scale method for the first four benchmarks and much better for *SWIM*. However, log-linear scale costs more than 20 times in space and computations than logarithmic scale for most programs.
- The multi-model method based on reference histograms outperforms single-model two-input method for two out of six programs. It gives the highest accuracy for FFT. As we explained in Section 4, this approach is very flexible and its performance depends heavily on the number of groups. In our experiment, we tried 7 different numbers of groups for each benchmark and presented the highest accuracy, but finding the maximal accuracy requires trying thousands of choices. The result for FFT shows the potential of this method, but the overhead of finding the best result is prohibitively high.

SWIM is a special program. The multi-model logarithmic scale has poor result for *SWIM*, but multi-model log-linear scale and single-model methods give very accurate predictions. Figure 5 shows the reuse distance histogram of *SWIM*. Note it has a high peak in a very small reuse distance range. Multi-model logarithmic scale uses log scale ranges. It assumes that the reuse distance is evenly distributed in each range, which brings significant noise for the analysis of *SWIM*. Log-linear scale methods alleviate the problem because their histograms are much more precise.

5.3 Results on Small Inputs

As we explained at the beginning of Section 5.2, different patterns may overlap with each other when the input size is small. In this case, single-model methods are not expected to perform well, while multi-model methods should work as well as in large input sizes. But these methods still require that the percentage of each model keeps unchanged for different input for each reuse distance range. Table 3 shows the performance of the four methods on small size inputs of SP benchmark (We do not show the results of the multi-model method using reference histograms because it is difficult to tune). The results show that multi-model log-linear scale method is significantly more accurate than other methods. The good accuracy shows that the percentage of each model remains unchanged even for small inputs.

The performance of multi-model logarithmic scale method is worse than the log-linear scale method because of the low precision in logarithmic scale histograms. Although multi-model log-linear scale method needs more computation and more space than the logarithmic scale method, this cost is less an issue for small-size inputs.

Table 3: Accuracy for SP with small-size inputs

largest training input size	testing input size	single-model 2 inputs	single-model >2 inputs	multi-model log scale	multi-model log-linear scale
8^3	10^3	79.61	79.61	85.92	89.5
	12^3	79.72	75.93	79.35	82.84
	14^3	69.62	71.12	74.12	85.14
	28^3	64.38	68.03	76.46	80.3
10^3	12^3	91.25	87.09	84.58	90.44
	14^3	81.91	83.20	78.52	87.23
	16^3	77.28	77.64	76.01	84.61
16^3	28^3	75.93	74.11	77.86	83.50

5.4 Comparison

We compare the five methods in Table 4, which uses the following **notations**:

SM-2: Ding and Zhong's original method

SM-m: Extended version of SM-2 on multiple inputs

MMLg: Multi-model *log* scale method

MMLn: Multi-model log-linear scale method

MMRF: Multi-model on reference histogram

Table 4: Features of Various Reuse Distance Prediction Methods

Approach	*SM-2*	*SM-m*	*MMLg*	*MMLn*	*MMRF*
Input No.	2	>2	>2	>2	>2
Model No. per Range	1	1	≥1	≥1	≥1
Histogram	Ref.	Ref.	Dist.	Dist.	Ref.
Granularity	log-linear	log-linear	log	log-linear	log-linear

A **comparison** of the four new approaches is as follows.

- *SM-m* and *MMRF* are based on reference histograms while *MMLg* and *MMLn* are based on reuse distance histograms. Thus, *MMLg* and *MMLn* do not need to transform between the two histograms as *SM-m* and *MMRF* do.
- *MMLg* saves 20 times in space and computation compared to *SM-m* and *MMLn*. *MMRF* can also save cost because it can freely select the number of groups, but it is hard to pick the right number.
- *MMLg* loses information because it assumes a uniform distribution in large ranges. It affects the prediction accuracy for programs like $SWIM$, which has a high peak in a very small range. In that case, *SM-m* and *MMLn* produce much better results because they use shorter ranges.
- *MMLn* predicts with higher accuracy than *SM-m* does if multiple models overlap. Overlapping often happens for inputs of small size, for which *MMLg* cannot perform well because of its loss of information.

Summary The experiments show that regression on more than two training inputs give significant improvement than the method using two inputs. Single-model multi-input, and multi-model logarithmic and log-linear scale methods produce comparable results for most programs when the input size is large. Their overall performance is the best among all five approaches. The multi-model method using reference histograms method is flexible but hard to control. Multi-model log-linear scale method can produce better results than multi-model logarithmic scale method. But the former needs over 20 times of more space and time than the latter, and the performance is not significantly different in most programs when the input size is large. For input of small size, the log-linear scale method is clearly the best among all methods.

6 Related Work

Data reuse analysis can be performed mainly in three ways: by a compiler, by profiling or by run-time sampling. Compiler analysis can model data reuse behavior for basic blocks and loop nests. An important tool is dependence analysis. Allen and Kennedy's recent book [1] contains a comprehensive discussion on this topic. Various types of array sections can measure data locality in loops and procedures. Such analysis includes linearization for high-dimensional arrays by Burke and Cytron [7], linear inequalities for convex sections by Triolet *et al.* [20], regular sections by Callahan and Kennedy [8], and reference list by Li *et al.* [16]. Havlak and Kennedy studied the effect of array section analysis on a wide range of programs [12]. Cascaval extended dependence analysis to estimate the distance of data reuses [9]. Other locality analysis includes the matrix model by Wolfe and Lam [21], the memory order by McKinley et al. [18], and a number of recent studies based on more advanced models.

Balasundaram et al. presented a performance estimator for parallel programs [4]. A set of kernel routines include primitive computations and common communication patterns are used to train the estimator. While their method trains for different machines, our scheme trains for different data inputs. Compiler analysis is not always accurate for programs with input-dependent control flow and dynamic data indirection. Many types of profiling analysis have been used to study data access patterns. However, most past work is limited to using a single inputs or measuring correlation among a few executions. The focus in this paper is to predict changes for new data inputs.

Run-time analysis often uses sampling to reduce the overhead. Ding and Kennedy sampled program data [10], while Arnold and Ryder sampled program execution [3]. Run-time analysis can identify patterns that are unique to a program input, while training-based prediction cannot. On the other hand, profiling analysis can analyze all accesses to all data.

Finally we discuss previous work in using the reuse distance. Reuse distance has been used by architecture and compiler researchers because it gives more information about program locality than a miss rate. For example, Huang and Shen used time distance between reuses to composite *value reuse profile* and study the bandwidth requirement of programs [13]. Li et al. defined reuse distance on values to study the limit of register reuse [15]. Beyls and D'Hollander used profiling to collect reuse distance information and generate placement hints for data [6]. They reported a performance improvement of 7% on an Itanium processor. Reuse distance prediction would extend these studies to all program inputs, not just the profiled ones.

For software managed cache, Jiang and Zhang developed an efficient buffer cache replacement policy, *LIRS*, based on the assumption that the reuse distance of cache blocks is stable over certain time [14]. Zhou et al. divided the second-level server cache into multiple buffers dedicated to blocks of different reuse intervals [24]. Both studies showed that reuse distance based management outperforms single LRU cache. Reuse distance prediction may be used in this context to classify file access traces based on their access patterns.

7 Conclusions

We draw the following conclusions.

1. Regression significantly improves the accuracy of reuse distance prediction, even with only a few training inputs.

2. The multi-model method using logarithmic histograms can save 95% space and computations and still keep the best accuracy in most programs, although it is not as consistent as those methods using log-linear histograms. Space efficiency is necessary for our future work to analyze individual patterns of different program data.

   It is a good choice when efficiency is important.

3. The multi-model method using log-linear scale histograms is the best for small input sizes, where different models tend to overlap each other. It is also efficient because the input size is small.

4. The single-model multi-input method has the highest accuracy, but it cannot accurately model small-size inputs. It is the best choice when one can tolerate a high profiling cost.

Reuse distance prediction allows locality analysis and optimization to consider program inputs other than profiled ones. we have presented an extensive study and a careful comparison of major prediction methods. The techniques discussed in this work may also help to solve other prediction problems, especially in program analysis and transformation.

References

[1] R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. Morgan Kaufmann Publishers, October 2001.

[2] G. Almasi, C. Cascaval, and D. Padua. Calculating stack distances efficiently. In *Proceedings of the first ACM SIGPLAN Workshop on Memory System Performance*, Berlin, Germany, June 2002.

[3] M. Arnold and B. G. Ryder. A framework for reducing the cost of instrumented code. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, Snowbird, Utah, June 2001.

[4] V. Balasundaram, G. Fox, K. Kennedy, and U. Kremer. A static performance estimator to guide data partitioning decisions. In *Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Williamsburg, VA, April 1991.

[5] K. Beyls and E.H. D'Hollander. Reuse distance as a metric for cache behavior. In *Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, August 2001.

[6] K. Beyls and E.H. D'Hollander. Reuse distance-based cache hint selection. In *Proceedings of the 8th International Euro-Par Conference*, Paderborn, Germany, August 2002.

[7] M. Burke and R. Cytron. Interprocedural dependence analysis and parallelization. In *Proceedings of the SIGPLAN '86 Symposium on Compiler Construction*, Palo Alto, CA, June 1986.

[8] D. Callahan, J. Cocke, and K. Kennedy. Analysis of interprocedural side effects in a parallel programming environment. *Journal of Parallel and Distributed Computing*, 5(5):517–550, October 1988.

[9] G. C. Cascaval. *Compile-time Performance Prediction of Scientific Programs*. PhD thesis, University of Illinois at Urbana-Champaign, 2000.

[10] C. Ding and K. Kennedy. Improving cache performance in dynamic applications through data and computation reorganization at run time. In *Proceedings of the SIGPLAN '99 Conference on Programming Language Design and Implementation*, Atlanta, GA, May 1999.

[11] C. Ding and Y. Zhong. Predicting whole-program locality with reuse distance analysis. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, CA, June 2003.

[12] P. Havlak and K. Kennedy. An implementation of interprocedural bounded regular section analysis. *IEEE Transactions on Parallel and Distributed Systems*, 2(3):350–360, July 1991.

[13] S. A. Huang and J. P. Shen. The intrinsic bandwidth requirements of ordinary programs. In *Proceedings of the 7th International Conferences on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, October 1996.

[14] S. Jiang and X. Zhang. LIRS: an efficient low inter-reference recency set replacement to improve buffer cache performance. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Marina Del Rey, California, June 2002.

[15] Z. Li, J. Gu, and G. Lee. An evaluation of the potential benefits of register allocation for array references. In *Workshop on Interaction between Compilers and Computer Architectures in conjuction with the HPCA-2*, San Jose, California, February 1996.

[16] Z. Li, P. Yew, and C. Zhu. An efficient data dependence analysis for parallelizing compilers. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):26–34, January 1990.

[17] R. L. Mattson, J. Gecsei, D. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM System Journal*, 9(2):78–117, 1970.

[18] K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems*, 18(4):424–453, July 1996.

[19] J. O. Rawlings. *Applied Regression Analysis: A Research Tool*. Wadsworth and Brooks, 1988.

[20] R. Triolet, F. Irigoin, and P. Feautrier. Direct parallelization of CALL statements. In *Proceedings of the SIGPLAN '86 Symposium on Compiler Construction*, Palo Alto, CA, June 1986.

[21] M. E. Wolf and M. Lam. A data locality optimizing algorithm. In *Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation*, Toronto, Canada, June 1991.

[22] Y. Zhong, C. Ding, and K. Kennedy. Reuse distance analysis for scientific programs. In *Proceedings of Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, Washington DC, March 2002.

[23] Y. Zhong, S. G. Dropsho, and C. Ding. Miss rate preidiction across all program inputs. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, New Orleans, Louisiana, September 2003.

[24] Y. Zhou, P. M. Chen, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *Proceedings of USENIX Technical Conference*, June 2001.