Abstract

Locality increasingly determines system performance. As a rigorous and precise locality model, reuse distance has been used in program optimizations, performance prediction, memory disambiguation and locality phase prediction. However, the high cost of measurement has been severely impeding its uses in scenarios requiring high efficiency, e.g. product compilers, performance debugging, and run-time optimizations.

This work proposes a statistical model to approximate reuse distance histograms from easily-obtained time distance histograms. The model makes reuse distance measurement as light as measuring data access frequency. Compared to the state-of-the-art technique, this model reduces measurement overhead by 17 times on ten SPEC CPU2000 ref executions and achieves over 99% accuracy for cache reuse approximation. Furthermore, this paper presents a trace generator, which produces data access traces from a given reuse distance histogram. It is beneficial for comprehensive evaluation of the approximation technique and can serve as a general tool for other locality studies.

1. Introduction

Locality or data reuse is the basis of utilizing memory hierarchy to alleviate the notorious memory wall problem. In modern computers, more processors share memory system and the accurate and efficient locality measurement is critical for system performance. Reuse distance, also called LRU stack distance, is the number of distinct data elements accessed in a reuse interval—the interval between the current and the previous access to the same data element [9]. It provides a rigorous and precise model of program locality, increasingly used in system performance analysis [13, 8, 11, 23], performance prediction [14, 15, 24], program analysis and optimizations [5, 25, 10]. However, reuse distance is also one of the most expensive locality models to build. The high time cost has been obstructing the application of reuse distance in performance debugging during the development stage, eliminates the possible uses for efficient locality optimizations, and makes offline program analysis painful or even unfeasible for long-running applications.

Requiring the count of distinct data in every reuse interval, the naive measurement of reuse distance, storing and counting data access trace in a huge array, has time complexity of $O(T^2)$, where $T$ is the total number of memory references in the execution. Starting from Mattson et al.’s algorithm in 1970, people have used stack, vector, AVL tree, splay tree and other data structures to improve the efficiency [2, 3, 12, 16, 18, 21]. In 2003, Ding and Zhong proposed an approximation algorithm, using each node of a splay tree to store a group of references conducted in an interval. Their technique achieves the lowest time complexity as $O(T \log N)$, where $N$ is the number of distinct data elements in a program. However, the overhead of the technique still slows down a program's execution by hundreds of times: for a 1-minute execution, the measurement of its reuse distance takes more than 4 hours.

The reason for the high cost is because all previous algorithms are essentially implementing the definition of reuse distance—“counting” the number of distinct data in every reuse interval. The technique proposed in this paper is inspired by a different perspective: Can we use some easily obtained program behavior to statistically approximate reuse distance?

We choose time distance as the “cheap” behavior. Time distance is the number of data elements (no need to be distinct) accessed in a reuse interval. Time distance can be measured as easy as access frequency, with time cost linear to the execution length. However, as people conceived, time distance cannot directly serve for locality characterization. In trace 1, for example, there are billions of references to variable $b$ between the two accesses of variable $a$, i.e. the time distance of the second access to $a$ is billions large. However, the access in fact has very good locality: its reuse distance is 1, and the access will be a cache hit as long as the cache system has more than 1 cache line (assuming $a$ and $b$ are smaller than a cache line.)

\[ a \quad b \ldots \quad b \quad a \quad \text{billions of } b \text{'s} \quad (1) \]

Despite the superficial irrelevance between time distance and locality, we discovered that their connection actually is so strong that through a statistical model, we can approximate the reuse distance from the time distance with over 99% accuracy [19]. The approximation is based on the histograms of the two kinds of distance as illustrated in Figure 1. The X-axes of the two histograms are reuse distance and time distance respectively. The Y-axes are the fraction of memory references. A bar in the graph tells us what percentage of the total memory references have reuse/time distance in a certain range. The two histograms are called reuse distance histogram and time distance histogram. A reuse distance histogram summarizes the locality of a program execution and is important for cache performance prediction [15, 14, 24], reference affinity detection [25], and data reorganization [8]. The statistical model we discovered bridges the two kinds of histograms, links time and locality, and accelerates locality measurement by over 17 times.

The technique includes two stages. First, it instruments an application and builds the time distance histogram of the program’s execution. Second, it approximates the reuse distance histogram using a statistical model, which includes three steps. It first uses the
time distance histogram to calculate the probability for a randomly picked time-point to fall into a randomly picked data element’s reuse interval of any given length. From that probability, it calculates the probability for a randomly picked data element to appear in a time interval of any given length. By treating reuse distance estimation as a binomial process, the model estimates the distribution of the number of distinct data in any time interval, thus the reuse distance histogram.

This paper has contributions in three folds. In our technical report [19], we have described the basic algorithm of the statistical model, where the width of a bar has to be 1 in both reuse and time distance histograms. This paper presents the more general algorithm, which allows bars of any width. That extension is critical for the high efficiency of the approximation process. Secondly, this paper describes the algorithm and the implementation of a trace generator, which, given an arbitrary reuse or time distance histogram, outputs a data access sequence having the similar locality. It makes precise locality as easy to obtain as data access opportunities for program optimizations.

This work reveals the strong correlations between time and locality. It makes precise locality as easy to obtain as data access frequency, enables efficient uses of reuse distance, and opens new opportunities for program optimizations.

In the rest of the paper, Section 2 lists the concepts and notations used in the model. Section 3 describes the statistical model. Section 4 presents the trace generator. Section 5 gives the experimental results. Section 6 discusses related work, followed by a summary.

2. Terminology

This section explains the concepts and notations used in the description of the algorithm in Section 3.

A **Reuse** is the access to a memory unit that has already been accessed before. In this paper, we only care two kinds of memory units: data element, i.e. the memory word accessed by a “load” or “store” instruction; cache line, i.e. a block of data mapping to the same cache line. The former reuse is called element reuse and the latter one is named cache line reuse. All the concepts below apply to both kinds of reuses. For simplicity, we will use “data” for the memory unit.

A **reuse interval** is an interval with accesses to the same data at both ends and without accesses to that data in the between.

A **reuse distance** is also called stack distance, is defined as the number of distinct data accessed inside a reuse interval (excluding the data accessed at the two ends of the reuse interval.) The minimal value is 0.

A **reuse distance histogram** is also called reuse signature, is a histogram showing the distribution of reuse distances in an execution. The X-axis is reuse distance, and the Y-axis is the fraction of total references. Each bar in the histogram shows the fraction of total program references with certain reuse distances.

A **time distance histogram** is a histogram showing the distribution of time distances in an execution. The X-axis is time distance, and the Y-axis is the fraction of total references. Each bar in the histogram shows the fraction of total program references with certain time distances.

**L** : the total number of bars in a reuse distance histogram.

**L** : the total number of bars in a time distance histogram.

**P** : the fraction of references having reuse distance of **k**.

**P** : the fraction of references having time distance of **k**.

**P** : the fraction of references having reuse distance in range **(b_i, b_i+1)**.

**P** : the fraction of references having reuse distance in range **(b_i, b_i+1)**.

**Time distance** is defined as the number of data (no need to be “distinct”) accessed in a reuse interval, including the access at one end of the interval. The minimal value is 1.

A **time distance histogram** is a histogram showing the distribution of time distances in an execution.

Figure 1. A reuse distance histogram and time distance histogram on log scale.
3. Statistical Model for Locality Approximation

This section describes the statistical model to approximate the reuse distance histogram, \( P_R \). From a time distance histogram \( P_T \), we use \( P_T \) and the number of distinct data accessed in the execution, \( N \), which is a side-product from time distance measurement. The model is general in allowing the width of the bars in a histogram to be any value.

In the rest of this section, we first describe the analogy between reuse distance estimation and the coin tossing experiment in order to convey the intuition behind the statistical model. Then, we present the three steps to calculate the probabilities required for the approximation.

3.1 Estimate Reuse Distance

The key of the statistical model is based on the following observation:

**Observation 3.1.** Given a program’s execution, we can approximate the reuse distance histogram and the probability for any given data to appear in an interval of any given length.

We use \( P_3(b_i) \) to represent the probability for any given data to appear in a reuse interval whose length is in range \([\overline{b_i}, \overline{b_i}]\). The approximation is based on the model of Bernoulli processes. A Bernoulli process is a discrete-time stochastic process consisting of a sequence of independent random variables taking values over the set \{0, 1\}. A typical Bernoulli process is the coin tossing experiment. Suppose that a coin has a probability \( p \) showing heads when being tossed. The probability for \( k \) heads in \( n \) tosses is in a binomial distribution, which is denoted as \( f(k; n, p) \) and calculated as

\[
f(k; n, p) = \binom{n}{k} p^k (1 - p)^{n-k}. \tag{2}
\]

Assuming the probability of any data to be accessed in an interval is independent from other data, the process of estimating the number of distinct data accessed in a reuse interval becomes a Bernoulli process. For a reuse interval with time distance falling into bar \( b_i \), each distinct data in the program is like a toss of a coin with probability \( P_3(b_i) \) showing heads. The number of tosses is equal to the total number of distinct data in the program. The number of times showing heads is the number of distinct data being accessed in the interval.

We use \( P(k, b_i) \) to represent the probability of having \( k \) distinct data in a reuse interval in bar \( b_i \). Being a Bernoulli process, the probability can be calculated similarly to equation 2 as follows:

\[
P(k, b_i) = \binom{N}{k} P_3(b_i)^k (1 - P_3(b_i))^{N-k}
\]

From \( P(k, b_i) \), we can calculate the reuse distance histogram easily. The fraction of references having reuse distance of \( k \) is

\[
P_R(k) = \sum_i P(k, b_i) \cdot P_T(b_i)
\]

The calculation of the Y-axis value of a bar \( b^{R_i} \) in the reuse distance histogram is as follows:

\[
P_R(b^{R_i}) = \sum_{k=0}^{b_i} P_R(k)
\]

Therefore, the approximation problem reduces to the calculation of \( P_3(b_i) \).

3.2 Calculate \( P_3(b_i) \)

The calculation of \( P_3(b_i) \), the probability of any given data to appear in a reuse interval whose time distance is in range \([\overline{b_i}, \overline{b_i}]\), includes three steps.

3.2.1 Step 1: Calculate \( P_3(b_i) \) from \( P_T \)

In this step, we use \( P_T \) to calculate \( P_3(b_i) \). \( P_3(b_i) \) is a probability defined in this way: Given a random time-point \( t \), if we pick a data \( v \) at random from those that are not accessed at time \( t \), \( P_3(b_i) \) is the probability that \( t \) is in one of \( v \)'s reuse intervals whose time distance is in range \([\overline{b_i}, \overline{b_i}]\). The definition suggests the following formal format of \( P_3(b_i) \), where \( \overline{b_i} = \overline{\text{last access prior to } b_i} \), is to normalize \( P_1(b_i) \) into range \([0, 1]\).
Step2: Calculate $P_2(b_i)$

This step uses $P_1(b_i)$ to calculate $P_2(b_i)$, which is defined as follows: Given a random time-point $t$, if we pick a data $v$ at random from those that are not accessed at time $t$, $P_2(b_i)$ is the probability that $v$'s last access prior to $t$ is in time range $(t - b_i, t - b_i]$, i.e. the dark section in Figure 2(a).

The calculation is as follows.

Because

$$P_T(b_i) = \frac{1}{T} \sum_t \sum_v B(\bar{b}_i \leq T(v) < b_i)$$


3.2.2 Step2: Calculate $P_2(b_i)$ from $P_1(b_i)$

Thus, $P_1(b_i) = \frac{b_i + \bar{b}_i - 3}{2(N-1)} \cdot P_T(b_i)$

$P_1(b_i) = \frac{1}{T \cdot (N-1)} \sum_{v, t \neq t} B(\bar{b}_i \leq T(v) - T(v') < b_i)$

$$= \frac{1}{T \cdot (N-1)} \sum_{v, t \neq t} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq T(v) - T(v') < b_i)$$

$$= \frac{1}{T \cdot (N-1)} \sum_{v, t \neq t} (T(v, n + 1) - T(v, n) - 1) \cdot \sum_{n=1}^{N(v)} B(\bar{b}_i \leq T(v, n) - b_i)$$

Assuming the interval length is in a uniform distribution inside a bar, we can calculate $P_1(b_i)$ as follows:

$$P_1(b_i) = \frac{1}{T \cdot (N-1)} \cdot \frac{\bar{b}_i + b_i - 3}{2} \cdot \sum_{v, n=1}^{N(v)} B(\bar{b}_i \leq T(v, n + 1) - T(v, n) < b_i)$$

$$= \frac{1}{T \cdot (N-1)} \cdot \frac{\bar{b}_i + b_i - 3}{2} \cdot \sum_{v} B(\bar{b}_i \leq T(v, n + 1) - T(v, n) < b_i)$$

$$= \frac{\bar{b}_i + b_i - 3}{2(N-1)} \cdot \frac{1}{T} \sum_{v} B(\bar{b}_i \leq T(v, n + 1) - T(v, n) < b_i)$$

$$= \frac{\bar{b}_i + b_i - 3}{2(N-1)} \cdot \frac{1}{T} \sum_{v} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i)$$

$$\sum_{v} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i) = \sum_{v} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i)$$

$$\sum_{v} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i) = \sum_{v} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i)$$

$$= \frac{1}{T \cdot (N-1)} \sum_{v} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i)$$

$$= \frac{1}{T \cdot (N-1)} \sum_{v} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i)$$

The region $[T(v, n) + 1, T(v, n + 1) - 1]$ can be divided into three sections as $[T(v, n) + 1, T(v, n) + b_i - 1]$, $[T(v, n) + b_i, T(v, n) + b_i - 1]$, and $[T(v, n) + b_i, T(v, n + 1) - 1]$. Notice that the binary function $B(\bar{b}_i \leq t - T(v, n) < b_i)$ is always false on both the first and the third section. Therefore,

$$P_2(b_i) = \frac{1}{T \cdot (N-1)} \sum_{v} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i)$$

Thus, the equation can be simplified to

$$P_2(b_i) = \frac{1}{T \cdot (N-1)} \sum_{v} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i)$$

$$= \frac{1}{T \cdot (N-1)} \sum_{v} \sum_{n=1}^{N(v)} B(\bar{b}_i \leq t - T(v, n) < \bar{b}_i)$$

In the value of $T(v, n + 1)$, there are three cases.

1) When $T(v, n) + \bar{b}_i - 1 < T(v, n + 1)$, the value of

$$\sum_{n=1}^{N(v)} B(t < T(v, n + 1))$$

becomes

$$\sum_{n=1}^{N(v)} [T(v, n) + \bar{b}_i - 1 - (T(v, n) + \bar{b}_i) + 1] = \sum_{n=1}^{N(v)} (\bar{b}_i - \bar{b}_i).$$

2) When $T(v, n) + \bar{b}_i \leq T(v, n + 1) \leq T(v, n) + \bar{b}_i - 1$, the value becomes

$$\sum_{n=1}^{N(v)} [T(v, n + 1) - 1 - (T(v, n) + \bar{b}_i) + 1] = \sum_{n=1}^{N(v)} (T(v, n + 1) - T(v, n) - \bar{b}_i).$$

3) In other cases, the value is 0.

We put the results of the first two cases together through binary functions. $P_2(b_i)$ becomes
\[ P_2(b_i) = \frac{1}{T \cdot (N-1)} \sum_{v=1}^{N(v)} \sum_{n=1}^{T(v,n+1)-1} [(\overline{b_i} - b_i) \cdot \\
B(T(v,n) + b_i \leq T(v,n+1)) + \\
(T(v,n+1) - T(v,n) - \overline{b_i}) \cdot \\
B(\overline{b_i} < T(v,n+1) - T(v,n) < \overline{b_i})] \\
\]

Because \( \sum_{t=T(v,n)+1}^{T(v,n+1)-1} \frac{1}{T(v,n+1) - T(v,n) - 1} = 1 \), \( P_2(b_i) \) won't change its value after being multiplied by that term:

\[ P_2(b_i) = \frac{1}{T \cdot (N-1)} \sum_{v=1}^{N(v)} \sum_{n=1}^{T(v,n+1)-1} [(\overline{b_i} - b_i) \cdot \\
B(T(v,n) + \overline{b_i} \leq T(v,n+1)) + \\
(T(v,n+1) - T(v,n) - \overline{b_i}) \cdot \\
B(\overline{b_i} < T(v,n+1) - T(v,n) < \overline{b_i})] \\
\]

Now we replace \( T(v,n) \) by \( T_{<t}(v) \) and replace \( T(v,n+1) \) by \( T_{\geq t}(v) \). \( P_2(b_i) \) is transformed to the following:

\[ P_2(b_i) = \frac{1}{T \cdot (N-1)} \sum_{v=1}^{N(v)} \sum_{n=1}^{T(v,n+1)-1} \frac{1}{T_{\geq t}(v) - T_{<t}(v) - 1} \\
[\overline{b_i} - b_i) \cdot B(T_{<t}(v) + \overline{b_i} \leq T_{\geq t}(v)) + \\
(T_{\geq t}(v) - T_{<t}(v) - \overline{b_i}) \cdot \\
B(\overline{b_i} < T_{\geq t}(v) - T_{<t}(v) < \overline{b_i})] \\
\]

Assuming the interval length is in a uniform distribution inside a bar, therefore

\[ \sum_{t,v \neq v(t)} B(T(v,n+1) - T(V,n) = \tau_1) = \sum_{t,v \neq v(t)} B(T(v,n+1) - T(V,n) = \tau_2) \]

where, \( \tau_1 \) and \( \tau_2 \) are two randomly chosen value in range \([\overline{b_i}, \overline{b_i} - 1] \).

That leads to the following equation:

\[ \sum_{\tau = \overline{b_i}}^{\overline{b_i} - 1} \frac{1}{\tau - 1} \sum_{t,v \neq v(t)} B(T(v,n+1) - T(V,n) = \tau) = \sum_{t,v \neq v(t)} B(\overline{b_i} \leq T_{<t}(v) - T_{\geq t}(v) < \overline{b_i}) \frac{\overline{b_i} - b_i}{\overline{b_i} - b_i} \sum_{\tau = \overline{b_i}}^{\overline{b_i} - 1} \frac{1}{\tau - 1} \]

With that equation, \( P_2(b_i) \) becomes
Step 3: Calculate the access time

We have described the statistical model, which approximates the reuse distance histogram from a time distance histogram by modeling the approximation as a Bernoulli process. The time complexity of the model is \( O(L^T \times L^R) \), where \( L^T \) is the number of bars in the time distance histogram and \( L^R \) is that in the reuse distance histogram.

Compared to the model described in our technical report [19], the major difference is that this model allows the time distance histogram to have bars of arbitrary width. That extension is both critical and non-trivial. Without it, the number of bars in the time distance histogram is equal to the length of the entire execution, which could be billions and makes the model too slow to use for long-running programs.

4. Generate Traces from Locality

In Section 5, we will demonstrate the effectiveness of the approximation technique on 10 SPEC CPU2000 programs. However, the histograms of those programs fall into several categories and cover only a small portion of the entire histogram space. To test the approach comprehensively, it is ideal to have a tool able to generate data access sequences from any given reuse distance histograms.

The trace generator developed in this work is such a tool, whose input includes a reuse distance histogram \( P_R \) of any distribution, the number of distinct data \( N \) and the number of total data accesses \( T \) in the trace to be generated. The generator creates the corresponding data trace by choosing data to fill each time point of the trace through a stochastic process based on the given \( P_R \), \( N \) and \( T \).

Notice, in the trace generator, all bars’ width is 1 in all histograms. The purpose is to better test the approximation technique since wider bars could hide the approximation error inside a bar. We assume that \( T \) is much larger than \( N \), which holds for most long-running programs.

4.1 Stochastic Process for Trace Generation

This section presents the algorithm of the trace generator. Figure 3(a) illustrates a reuse distance histogram with \( N = 7 \) distinct data.

The first step is to construct the cumulative distribution. We use \( C_R(i) \) to represent the cumulative probability of a distance \( i \) and calculate it as follows:

\[
C_R(i) = \sum_{j=0}^{i} P_R(j)
\]

where \( P_R(j) \) is the histogram value of reuse distance \( j \). The crosses, “x”, in Figure 3(b) show the cumulative distribution of the histogram in Figure 3(a). As shown by the horizontal broken lines in Figure 3(b), the Y-axis values of those crosses partition range \([0, \frac{1}{N}]\) into \( N \) cumulative sections: \([0, C_R(0)]\), \([C_R(0), C_R(1)]\), \( \ldots \), \([C_R(N-2), 1]\).

In the second step, we simply fill the first \( N \) positions of the trace with the \( N \) distinct data. The order of the data is not important.
void GenTrace (double RDH [], int N, int T, double *trace)
// build cumulative distribution from reuse distance histogram
double * CD;
BuildCD(RDH, CD);

// fill first N positions with all distinct data
for (int i=0; i<N; i++)
trace[i] = i;

// fill other positions
for (int i=N; i<T; i++) {
    a = random();
    s = findCumulativeSection (a, CD);
    rd = RDH[s];
    // find the data with reuse distance of rd in trace[0 — i-1]
    trace[i] = findDataWithRD(trace,0,i-1,rd)
}

Figure 4. Algorithm to generate a data trace from a reuse distance histogram.

The last step is to fill the other positions of the trace one by one. At each time, a random number generator produces a number evenly distributed in range [0, 1]. We then find out the cumulative section which α falls into, represented by s(α). Suppose r(s(α)) is the reuse distance corresponding to section s(α). We fill the current position of the trace with the data that has reuse distance of r(s(α)) from its previous access. For example, if the current data trace is “a b c b c” and r(s(n)) is 2, the next data should be “a”. The rationale of the generation is that the probability of random number α to fall into a cumulative section s is equal to the reuse distance histogram value of section s. Figure 4 shows the algorithm.

Finding the data with reuse distance of r(s(α)) takes O(log N) time when the most recent distinct data are organized in a splay tree. The time complexity of the whole trace generation is O(TlogN).

The similar model can be applied to time distance histograms and yields another trace generator which produces the corresponding trace given a time distance histogram. We implement the generator for locality study of other purposes.

5. Evaluation

This section presents experiment results on both generated traces and the programs from SPEC CPU2000 suite. Generated traces make it easy for us to test the approximation model on histograms of different distributions. We use the SPEC programs to demonstrate the approximation efficiency and accuracy in real uses. Different from [19], this work uses the ref runs of SPEC benchmarks which are longer and have more complex reuse histograms than the test and train runs used before.

All experiments were run on Intel(R) Xeon(TM) 2.00GHz Processors with 2 GB of memory running Fedora Core 3 Linux. We use PIN 3.4 for instrumentation with gcc version 3.4.4 as our compiler. All compilations have the highest level optimization with “-O3” turned on.

In the experiments, we use Ding and Zhong’s implementation of their technique (using PIN) [9] to measure the real reuse distance histograms for comparison (the version with 0.1% relative error limit.) To our knowledge, that is the fastest tool for reuse distance collection.

The accuracy of approximated reuse distance histogram \( \hat{RDH} \) is calculated as follows:

\[
\text{accuracy} = 1 - \frac{1}{2} \sum_{i} |B_i - \hat{B}_i| 
\]  

Where, \( B_i \) is the Y-axis value of the \( i \)’th bar in real histogram \( RDH \) and \( \hat{B}_i \) is the one in \( \hat{RDH} \). The division by 2 is to normalize the accuracy to [0,1].

5.1 Evaluation of Trace Generator

In [19], we have shown the effectiveness of the statistical model on generated traces with three different reuse distance histograms: histograms of random distribution, pulse-like distribution, and exponential distribution. The accuracy of reuse distance approximation is over 98%.

In this section, rather than reporting more approximation results, we focus on the evaluation of the trace generator instead of the approximation technique. We measure how close the reuse distance histogram of the generated trace is from the given histogram. The closer the better.

We choose histograms in Gaussian and exponential distributions for the evaluation. By changing the variance, the distributions change from one with a high peak in a small region to one with nearly uniform distribution. Figure 5 shows two examples with
Table 1. Accuracy of reuse distance of the generated traces.

<table>
<thead>
<tr>
<th>distr.</th>
<th>gaussian (variance=) 20</th>
<th>60</th>
<th>100</th>
<th>140</th>
<th>180</th>
<th>exp.</th>
<th>avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc. (%)</td>
<td>98.2</td>
<td>96.7</td>
<td>96.4</td>
<td>96.2</td>
<td>96.0</td>
<td>97.4</td>
<td>96.8</td>
</tr>
</tbody>
</table>

The accuracy of the reuse distance histograms of the generated trace compared to the given one. All 6 distributions have accuracy greater than 96%. For Gaussian distributions, as the variance increases, the accuracy decreases, which is because the histogram value becomes close to zero for distributions with large variance as shown in Figure 5(b). The exponential distribution is as follows: \( P_{RD}(k) \propto e^{-0.02\times k} \). Overall, for all those distributions, the histograms of the generated traces have less than 3% difference from the given histogram.

### 5.2 Locality Approximation on SPEC ref Runs

We use the ref runs of SPEC CPU2000 benchmarks to evaluate the effectiveness of the reuse distance histogram approximation technique. In [19], we presented the results on test and train runs. On 10 benchmarks used in this experiment, the ref runs on average are 8 times longer than the train runs, and have much wider range of reuse distance and time distance.

We choose the 10 benchmarks from the 12 used in [19]. There are 6 integer benchmarks and 4 floating-point benchmarks.

#### 5.2.1 Accuracy

Table 2 shows the reuse distance approximation accuracy on both element and cache line reuses. The accuracy is measured on both linear and log scales: A bar in a linear-scale histogram covers the reuse distance of 1K; the bars in a log-scale histogram have the range as \([0 \text{K}), [1\text{K}, 2\text{K}), [2\text{K}, 4\text{K}), [4\text{K}, 8\text{K}), \ldots\) The accuracy calculation is through Equation 3.

The accuracy for element reuses is 86.3% and 87.4% on average. Benchmark mcf gives the lowest accuracy. A possible reason for the inaccuracy is the independence assumption made in the statistical model: we assume that each variable has a probability to appear in an interval and that probability is independent of the other variables. Although that assumption holds in most cases, it could result in inaccuracy of the reuse distances as in the mcf case. However, a larger granularity removes the error almost completely. A common use of reuse distance is to study cache behavior through cache line reuse histograms, for which the occasionally low accuracy of element reuses does not matter.

The approximation accuracy for cache line reuses is 98.8% and 99.0% for linear and log scale respectively. The worst accuracy is on benchmark ammp, but it is still as accurate as 94.0% and 95.6%. Seven of the ten benchmarks have over 99% accuracy.

These results compare favorably to our previous results [19]. The cache line accuracy for those tests were 99.3% and 99.4% for linear and log scales respectively. However, the element accuracies are 91.8% and 94.1% for linear and log scales on the test and train inputs, compared to 86.3% and 87.4% for the ref input. This drop is due to the more complex histograms of ref runs.

#### 5.2.2 Time Cost

Table 3 shows the time cost of the statistical model compared with the fastest reuse distance measurement in the past developed by Ding and Zhong. The second column gives the basic overhead of the instrumentation, \( T_{inst} \). It is the running time of the benchmarks after being inserted an invocation of an empty function at each

| speedup = \( (T_{RD} - T_{inst})/(T_{RD} + T_{conv} - T_{inst}) \)

<table>
<thead>
<tr>
<th>Runs</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.74</td>
</tr>
<tr>
<td>20</td>
<td>1.52</td>
</tr>
<tr>
<td>30</td>
<td>1.39</td>
</tr>
<tr>
<td>40</td>
<td>1.27</td>
</tr>
<tr>
<td>50</td>
<td>1.15</td>
</tr>
<tr>
<td>60</td>
<td>1.03</td>
</tr>
<tr>
<td>70</td>
<td>0.91</td>
</tr>
<tr>
<td>80</td>
<td>0.80</td>
</tr>
<tr>
<td>90</td>
<td>0.69</td>
</tr>
<tr>
<td>100</td>
<td>0.58</td>
</tr>
</tbody>
</table>

The speedup of our technique compared to Ding and Zhong’s. In order to avoid the effects of different instrumentors, we subtract the instrumentation overhead from both kinds of measure time when calculating the speedup as follows:

### 6. Related Work

Compiler analysis has been successful in understanding and improving locality in basic blocks and loop nests. McKinley and Temam carefully studied various types of locality within and between loop nests [17]. Cascaval presented a compiler algorithm that measures reuse distance directly [6]. Allen and Kennedy discussed the subject comprehensively in their book [1]. Thabit identified data often used together based on their access frequency [22]. Chlibimi used grammar compression to find hot data streams and reorganized data accordingly [7].

As a precise locality model, reuse distance has been used to study the limit of register reuse [13] and cache reuse [11, 8, 23], to evaluate the effect of program transformations [8, 4], to predict program performance [14, 24] and program phases [20], and to guide data placement [5, 25] and memory disambiguation [10].

Because of the importance of reuse distance, its measurement has been studied for several decades. In 1970, Mattson et al. published the first measurement algorithm [16] using a list-based stack. The time complexity is \( O(TN) \), where \( T \) is the execution length and \( N \) is the size of program data. Space cost is \( O(N) \). Bennett and Kruskal used a vector and built an \( m \)-ary tree on it [3] to speedup the measurement. That reduces time complexity to \( O(TlogT) \) but increases space complexity to \( O(T) \). In 1981, Olken implemented the first tree-based method using an AVL tree [18] with time complexity \( O(TlogN) \) and space overhead \( O(N) \). Sugumar and Abraham in 1993 showed that a splay tree has better memory performance [21] and developed the widely used cache simulator, Chetah.

In 1991, Kim et al. proposed the first imprecise analysis method [12]. Their method marks \( S \) ranges in the list that store program data and counts the number of distances that fall inside each range. Its time complexity is \( O(TS) \) and space complexity \( O(C) \), where \( C \) is the furthest marker. Ding and Zhong further proposed an approximation algorithm through dynamic tree compression in 2003 [9]. The approximation algorithm uses a tree node to record a group of data accessed closely and assigns a single time memory access. Much of that overhead could be saved with a more efficient instrumentor, e.g. a source-code level instrumentor through a compiler. The left half of the rest of the table shows the result of element reuses, and the right half gives that of cache line reuses (in this case a cache line is 128-byte wide.) Within each part, the first column is the time of Ding and Zhong’s technique, \( T_{RD} \); the sum of the next two columns is the total time of our technique, which includes the time to measure time distance, \( T_{RD} \), and the time to convert time distance histograms to reuse distance histograms, \( T_{conv} \). The next column shows the speedup of our technique compared to Ding and Zhong’s. In order to avoid the effects of different instrumentors, we subtract the instrumentation overhead from both kinds of measure time when calculating the speedup as follows:

<table>
<thead>
<tr>
<th>Runs</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.74</td>
</tr>
<tr>
<td>20</td>
<td>1.52</td>
</tr>
<tr>
<td>30</td>
<td>1.39</td>
</tr>
<tr>
<td>40</td>
<td>1.27</td>
</tr>
<tr>
<td>50</td>
<td>1.15</td>
</tr>
<tr>
<td>60</td>
<td>1.03</td>
</tr>
<tr>
<td>70</td>
<td>0.91</td>
</tr>
<tr>
<td>80</td>
<td>0.80</td>
</tr>
<tr>
<td>90</td>
<td>0.69</td>
</tr>
<tr>
<td>100</td>
<td>0.58</td>
</tr>
</tbody>
</table>

The speedup of our technique compared to Ding and Zhong’s. In order to avoid the effects of different instrumentors, we subtract the instrumentation overhead from both kinds of measure time when calculating the speedup as follows:
Figure 5. Illustration of the evaluation of trace generator. Solid lines show the given reuse distance histograms and the broken lines show the reuse distance histograms of the generated traces. Both given histograms are in Gaussian distribution. The width of each bar is 1. We connect the points into lines for better legibility.

Table 2. Approximation accuracy of data element and cache line reuse distance histograms on ref runs.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>Number of data elements</th>
<th>Number of mem. accesses</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>linear</td>
</tr>
<tr>
<td>CINT</td>
<td>crafty Game playing: chess</td>
<td>485K</td>
<td>117B</td>
<td>91.8</td>
</tr>
<tr>
<td></td>
<td>gcc GNU C programming language compiler</td>
<td>22.8M</td>
<td>32.9B</td>
<td>89.0</td>
</tr>
<tr>
<td></td>
<td>gzip GNU compression using Lempel-Ziv coding</td>
<td>70.4K</td>
<td>634M</td>
<td>99.0</td>
</tr>
<tr>
<td></td>
<td>mcf Combinatorial optimization</td>
<td>20.0M</td>
<td>25.0B</td>
<td>42.6</td>
</tr>
<tr>
<td></td>
<td>twolf Place and route simulator</td>
<td>7.38K</td>
<td>137B</td>
<td>88.2</td>
</tr>
<tr>
<td></td>
<td>vortex Object-oriented Database</td>
<td>2.76M</td>
<td>9.63B</td>
<td>93.2</td>
</tr>
<tr>
<td>CFP</td>
<td>ammp Modeling molecule system</td>
<td>3.02M</td>
<td>167B</td>
<td>95.8</td>
</tr>
<tr>
<td></td>
<td>equake Seismic wave propagation simulation</td>
<td>9.63M</td>
<td>76.7B</td>
<td>57.6</td>
</tr>
<tr>
<td></td>
<td>mesa 3-D graphics library</td>
<td>1.78M</td>
<td>139B</td>
<td>97.3</td>
</tr>
<tr>
<td></td>
<td>wupwise Physics/Quantum Chromodynamics</td>
<td>38.4M</td>
<td>168B</td>
<td>90.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86.3</td>
</tr>
</tbody>
</table>

Table 3. Comparison of the time of reuse distance approximation and measurement on ref runs.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Instrument overhead (sec.)</th>
<th>Data element level</th>
<th>Cache line level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reuse dist. measure (sec.)</td>
<td>Time dist. measure (sec.)</td>
<td>Time to approximate (sec.)</td>
</tr>
<tr>
<td>CINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>crafty 2935</td>
<td>114099</td>
<td>11688</td>
</tr>
<tr>
<td></td>
<td>gcc 932</td>
<td>25484</td>
<td>2233</td>
</tr>
<tr>
<td></td>
<td>gzip 21</td>
<td>553</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>mcf 1038</td>
<td>28138</td>
<td>4397</td>
</tr>
<tr>
<td></td>
<td>twolf 3530</td>
<td>122424</td>
<td>23662</td>
</tr>
<tr>
<td></td>
<td>vortex 239</td>
<td>8127</td>
<td>606</td>
</tr>
<tr>
<td>CFP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ammp 2431</td>
<td>147999</td>
<td>12596</td>
</tr>
<tr>
<td></td>
<td>equake 116/</td>
<td>5685</td>
<td>4361</td>
</tr>
<tr>
<td></td>
<td>mesa 2936</td>
<td>112254</td>
<td>7104</td>
</tr>
<tr>
<td></td>
<td>wupwise 2674</td>
<td>143689</td>
<td>9620</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
stamp to all of them. The time complexity becomes $O(T \log \log N)$ and the space overhead is $O(\log N)$.

This work tackles the problem from a unique perspective. Instead of "counting" distinct data as all previous methods do, it discovers the statistical connection between time distance and reuse distance and achieves 17 times speedup than the state-of-the-art approach. The complexity for measuring distance time is $O(T)$.

7. Conclusions

This paper presents an algorithm converting time distance histograms to reuse distance histograms. The algorithm is general in allowing the histograms to have bars of any width. Furthermore, the paper describes a data trace generator, which produces data access traces from a given reuse distance or time distance histogram. The generator helps the evaluation of the locality approximation technique and can serve for other locality studies. The experiments on ref runs of ten SPEC CPU2000 benchmarks demonstrate over 99% accuracy for cache line reuse approximation and 17 times speedup over the past techniques. The technique makes precise locality as easy to obtain as data access frequency, enables efficient uses of reuse distance, and opens new opportunities for program optimizations.

Acknowledgments

This material is based upon work supported by a grant from The National Science Foundation grant numbers EIA-0080124, EIA-0205061 (subcontract, Keck Graduate Inst), NSF EF-0328363 and NSF CNS-0509270. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of above named organizations.

We thank Chen Ding for the excellent suggestion on the ABA-scheme for the implementation of distance measurement.

References