In Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS 2006).

Comparing Multinomial and K-Means Clustering for SimPoint

Greg Hamerly[‡] Erez Perelman[†] Brad Calder[†]

[‡]Department of Computer Science, Baylor University
[†]Department of Computer Science and Engineering, University of California, San Diego

Abstract

SimPoint is a technique used to pick what parts of the program's execution to simulate in order to have a complete picture of execution. SimPoint uses data clustering algorithms from machine learning to automatically find repetitive (similar) patterns in a program's execution, and it chooses one sample to represent each unique repetitive behavior. Together these samples represent an accurate picture of the complete execution of the program. SimPoint is based on the k-means clustering algorithm; recent work proposed using a different clustering method based on multinomial models, but only provided a preliminary comparison and analysis.

In this work we provide a detailed comparison of using k-means and multinomial clustering for SimPoint. We show that k-means performs better than the recently proposed multinomial clustering approach. We then propose two improvements to the prior multinomial clustering approach in the areas of feature reduction and the picking of simulation points which allow multinomial clustering to perform as well as k-means. We then conclude by examining how to potentially combine multinomial clustering with k-means.

1 Introduction

As a program executes its behavior often changes. These changes are not random, but rather are often structured as sequences of a small number of recurring behaviors, which we term *phases*. This structured behavior can be of great benefit as it allows us to intelligently sample a programs execution, as long as these recurring phases can be found, without performing full cycle-accurate simulation. This is accomplished by identifying each of the repetitive behaviors and then taking only a single sample of each behavior to serve as the representative of that behavior. All of these representative samples taken together should approximate the complete execution of the program. This is the underlying philosophy of the tool called SimPoint [24, 25, 20, 2, 16, 15].

SimPoint intelligently chooses a very small set of samples called *Simulation Points* that, when simulated and weighted appropriately, provide an accurate picture of the complete execution of the program. Simulating in detail only these carefully chosen simulation points can save hours of simulation time over random statistical sampling, while still providing the accuracy needed to make reliable decisions based on the outcome of the cycle level simulation.

The SimPoint approach uses the k-means clustering algorithm for finding the phases of a program. In the machine learning community the k-means algorithm is a time-proven technique which groups together data points into clusters based on their spatial locality to one another. Recent work by Sanghai et al. [22] proposed using multinomial mixture models instead of k-means for the clustering. They reported results for 8 SPEC benchmarks and found that their multinomial approach resulted in a similar error rate, but fewer simulation points.

In this work, we evaluate this prior multinomial approach for all of the SPEC benchmarks on all of the inputs. To perform this evaluation Sanghai et al. [22] were gracious enough to provide us with the code they used for their KDD paper. We found that the prior multinomial algorithm [22] resulted in higher max error rate and slightly higher average error rates, while using more simulation points than k-means. Even so, we see promise in their approach. We therefore focused on improving the prior multinomial clustering algorithm, and found two areas of improvement. The first improvement focuses on a better approach for reducing the dimensionality of the data for use with multinomial clustering. The second improvement focuses on an improvement for picking simulation points. With these improvements we are able to achieve similar error rates with a slight improvement in the number of simulation points using multinomial clustering over k-means. Finally, we examine a heuristic for choosing when to use the multinomial model or k-means for clustering, based upon how well the multinomial centers are represented by their corresponding simulation points, which we call 'purity'.

The remainder of the paper is laid out as follows. First, we define phase behavior and describe its related work in Section 2. Section 3 describes in depth the multinomial approach as proposed in [22]. Section 4 presents our two main optimizations over the prior approach. Section 5 compares the performance of the prior and our optimized multinomial approaches with that of SimPoint. We also propose a heuristic for choosing which clustering method to use for a program/input in this section, and provide results that validate the approach. Our findings are summarized in Section 6.

2 Defining Phase Behavior and Related Work

As described in prior work [23], program behaviors change over time on many different time scales, and even at a large

time scale one can find repeating behaviors. A program may have stable behavior for billions of instructions and then change suddenly. In addition to performance, we have found for the SPEC 95 and 2000 programs that the behavior of all of the architecture metrics (branch prediction, cache misses, etc.) tend to change in unison, though not necessarily in the same direction [23, 25]. These corresponding changes are due to underlying changes in program execution, and can result in tremendous changes across a variety of architectural metrics.

2.1 Phase Vocabulary

The underlying methodology used in this work is the ability to automatically identify these underlying program changes by grouping a program's execution into phases *without relying on architectural metrics* to identify the phases. To ground our discussion in a common vocabulary, the following is a list of terms we use to describe the analysis performed by SimPoint, and their definitions.

- Interval - A section of continuous execution (a slice in time) of a program. All intervals are assumed to be non-overlapping, so to perform our analysis we break a program's execution into contiguous non-overlapping intervals. The length of an interval is often defined as the number of instructions committed during that interval (e.g., interval lengths of 1, 10, or 100 million instructions were used in [20]). SimPoint 3.0 supports both fixed-length intervals (all intervals have the same length) and variable-length intervals (VLIs), which allows different intervals to account for different amounts of executed instructions [14].
- Phase - A set of intervals within a program's execution with similar behavior. A phase can consist of intervals that are not temporally contiguous, so a phase can re-appear many times throughout execution.
- Similarity - How close the behavior of two intervals are to one another as measured across some set of metrics. Well-formed phases should have intervals with similar behavior across various architecture metrics (e.g. IPC, cache misses, branch misprediction).
- Frequency Vector - Each interval is represented by a frequency vector, which represents the program's execution during that interval. The most commonly used frequency vector is the basic block vector [24], which represents how many times each basic block is executed in an interval. Frequency vectors can also be used to track other code structures [16] such as all branch edges, loops, procedures, registers, opcodes, data, or program working set behavior as long as tracking usage of the structure provides a signature of the program's behavior.
- Similarity Metric - Similarity between two intervals is calculated by taking the distance between the corresponding frequency vectors from the two intervals. SimPoint determines similarity by calculating the Euclidean distance between the two vectors.
- Phase Classification - Groups together intervals into phases with similar behavior, based on a similarity metric. Phase classification is applied to a program binary running a particular input (a binary/input pair).

2.2 Similarity Metric - Distance Between Code Signatures

SimPoint represents intervals with frequency vectors. A frequency vector is a one dimensional array, where each element in the array tracks usage of some way to represent the program's behavior. We focus on code structures, but a frequency vector can consist of any structure (e.g., data working sets, data stride access patterns [16]) that may provide a signature of the program's behavior. A frequency vector is collected for each interval. At the beginning of each interval we start with a frequency vector containing all zeros, and as the program executes, we update the current frequency vector as structures are used.

A common frequency vector we have used is a list of static basic blocks [24] (called a Basic Block Vector, or BBV), which we also use in this paper. If we are tracking basic block usage with frequency vectors, we count the number of times each basic block in the program has been entered in the current interval, and we record that count in the frequency vector, weighted by the number of instructions in the basic block. The intuition behind this is that the behavior of the program at a given time is directly related to the code executed during that interval [24]. We use the basic block vectors as signatures for each interval of execution: each vector tells us what portions of code are executed, and how frequently those portions of code are executed. By comparing the BBVs of two intervals, we can evaluate the similarity of the two intervals. If two intervals have similar BBVs, then the two intervals spend about the same amount of time in roughly the same code, and therefore we expect the behavior of those two intervals to be similar.

To compare two frequency vectors, SimPoint 3.0 uses the Euclidean distance, which has been shown to be effective for off-line phase analysis [25, 20]. The Euclidean distance is calculated by viewing each vector as a point in D-dimensional space, and calculating the straight-line distance between the two points.

2.3 Using k-Means for Phase Classification

Clustering divides a set of points into groups, or clusters, such that points within each cluster are similar to one another (by some metric, usually distance), and points in different clusters are different from one another. The k-means algorithm [17] is an efficient and well-known clustering algorithm which we use to split program behavior into phases. The k in k-means refers to the number of clusters (phases) the algorithm will search for.

The following steps summarize the phase clustering algorithm at a high level. We refer the interested reader to [25] for a more detailed description of each step.

1. Profile the program by dividing the program's execution into contiguous intervals, and record a frequency vector for each interval.
2. Reduce the dimensionality of the frequency vector data to a smaller number of dimensions using random linear projection.
3. Run the k-means clustering algorithm on the reduced-dimension data for a set of k values.
4. Choose from among these different clusterings a well-formed clustering that also has a small number of clusters, using a threshold on the Bayesian Information Criterion (BIC). The BIC [19] gives a measure on how well a clustering fits a set of data.
5. Select the simulation points for the chosen clustering. For each cluster (phase), we choose one representative interval that will be simulated in detail to represent the behavior of the whole cluster. By simulating *only* one representative interval per phase we can extrapolate and capture the behavior of the entire program. To choose a representative, SimPoint picks the interval in each cluster that is closest to the *centroid* (center) of each cluster. Each simulation point also has an associated weight, which reflects the fraction of executed instructions that cluster represents.
6. With the weights and the detailed simulation results of each simulation point, we compute a weighted average for the architecture metric of interest (CPI, miss rate, etc.). This weighted average of the simulation points gives an accurate prediction of the metric for the complete execution of the program/input pair.

2.4 Related Work on Phase Analysis

Several other researchers have worked on phase analysis, and we review some of the related work here. The recurring use of different areas of memory in a program was first noted by Denning and Schwarz [5] and was formalized as the idea of working sets. While working sets have driven the development of caches for decades, recently many of the more subtle implications of recurring behaviors have been explored by researchers in the computer architecture community.

Balasubramonian et al. [1] proposed using hardware counters to collect miss rates, CPI and branch frequency information for every 100,000 instructions. They used these miss rates and the total number of branches executed for each interval to dynamically evaluate the program's stability, which they then used to guide dynamic cache reconfiguration to save energy without sacrificing performance.

Dhodapkar and Smith [6, 7, 8] found a relationship between phases and instruction working sets, and showed that phase changes occur when the working set changes. They proposed performing dynamic reconfiguration of multi-configuration units in response to phase changes indicated by working set changes. Through a working set analysis of the instruction cache, data cache and branch predictor they found methods to save energy.

Hind et al. [10] provided a framework for defining and reasoning about program phase classifications, focusing on how to best define granularity and similarity to perform phase analysis.

Isci and Martonosi [11, 12] have shown the ability to dynamically identify the power phase behavior using power vectors. Deusterwald et al. [9] used hardware counters and other phase prediction architectures to find phase behavior.

These related methods offer alternative techniques for representing programs for the purpose of finding phase behaviors. They each also offer methods for using the data to find phases. Our work on SimPoint frames the problem as a clustering problem in the machine learning setting, using data clustering algorithms to find related program behaviors. This problem is a natural application of data clustering, and works well.

3 Multinomial Clustering

In this section we describe in detail the approach used in [22] to perform multinomial clustering and the choosing of simulation points.

3.1 Overview of Multinomial Clustering

A multinomial model defines the probability of a vector of counts of d mutually exclusive events that happen over time. For example, if $d = 3$, then there are three mutually exclusive events which can happen, and we can use a vector to represent how many times each of these events happen over a period of time. So if the vector contains the values (30, 20, 25), then over 75 trials the first event happened 30 times, the second event happened 20 times, and the third event happened 25 times, though the order in which the events happened is not captured. This model appears to fit well with the concept of a frequency vector, e.g. a basic block vector, which is a count of mutually exclusive execution paths over a time interval.

Rather than using just one multinomial probability model, we can combine several of them to create a mixture of multinomials. This way, multiple parts of the probability space can be covered with different models. However, we must learn an appropriate set of parameters for each multinomial in the mixture using clustering. The multinomial mixture uses a set of k multinomial models (which we also call centers in a clustering context), each of which represents a region of high probability. That is, a multinomial model should represent a cluster of frequency vectors. The proportion of frequency vectors represented by the multinomial model is kept track of by an overall weight associated with the model, called a prior probability. This weight represents the percent of vectors that are accounted for by the multinomial model. Each model represents all d dimensions of the clustered data, and each dimension for a multinomial model represents the probability of that dimension occurring across all of the vectors, weighted by the amount that each vector belongs to the cluster.

The soft-assignment clustering model considers each clustered vector to be represented by all k clusters, where each cluster represents some proportion of the vector. This is formalized as a set of k probabilities for each vector; these probabilities come naturally out of the EM clustering algorithm for multinomials. These probabilities can be used to assign a vector to a cluster, by choosing the cluster with the highest probability of generating that vector. This is the method the authors of [22] use to assign vectors to clusters.

3.2 Feature Reduction and Multinomial Representation of Data

Each clustered vector using a multinomial mixture is a vector of d counts. Those counts represent the number of times one of d mutually exclusive events has happened during a time interval. This is very appropriate for representing a frequency vector, where each element is the count of the number of times a basic block has executed (in the case of basic block vectors). However, since d may be thousands or even millions, we need to reduce the dimensionality to make clustering feasible. In particular, we need to find a lower-dimension representation of the data. Following earlier work in SimPoint and text-based machine-learning, the prior approach for multinomial clustering used random linear projection to reduce the dimension of the data. Random linear projection consists of constructing a random projection matrix, which is then used to project all the original data down to a lower-dimensional representation.

One must be careful in constructing the random projection matrix so that it does not produce negative numbers in the resulting projected data. We cannot have negative numbers in the resulting data because the multinomial probability model is applicable only to vectors of non-negative numbers. This makes sense, since it is not clear what a negative count of some event occurring would mean. Upon consultation with the authors of the [22], they have improved their random matrix projection to be the following. Their approach performs the feature reduction by using a projection matrix randomly generated from just 0 and 1s.

In previous work, we have chosen to project to 15 dimensions using SimPoint with k-means, and have found that this is an adequate number of dimensions for capturing phase behavior. Sanghai et al. [13] found this to be inadequate for some programs using multinomial clustering. Thus we use their heuristic for choosing the number of projected dimensions depending on the original number of dimensions, which we list here.

original dimension	projected dimension
< 2000	15
$2000 - 9999$	50
$10000 - 19999$	75
≥ 20000	100

Once we have a projection matrix, we can apply it to the frequency vector data to obtain a lower-dimensional version. If the frequency vector data is represented as a matrix X of n rows (vectors) by d_1 columns (original dimension), then we construct a projection matrix P of size d_1 rows by d_2 columns (low dimension). Then the projected frequency vectors are found with a matrix multiplication: $X' = XP$. The projected data X' will have n rows (vectors) and d_2 columns (dimensions). Since X is usually sparse (and P may be sparse), there are efficient ways of computing this without resorting to using every element in X and P.

After projection, we have a low-dimensional set of vectors. The prior approach [22] then used these vectors as-is in the multinomial clustering approach we describe next.

3.3 Multinomial Clustering Approach Details

The prior work used the standard multinomial clustering algorithm, which we describe in detail in this section.

The multinomial probability model assigns a probability to a set of d mutually-exclusive events occurring in any order. This set of events is represented by a vector $v = (v_1, v_2, \ldots, v_d)$ of counts of the number of times each event has happened. For example, a coin flip could represent two events: a head or a tail, so d would be 2. If we count the number of heads and tails over 8 flips, there may be 5 heads and 3 tails. For our example, $v = (5, 3)$. The probability of all the events is determined by a vector of d parameters $c = (\theta_1, \theta_2, \ldots, \theta_d)$, which represent the probability of each event happening. In our example, if the coin is fair, it would be the case that $\theta_1 = \theta_2 = 0.5$. The probability of all the events occurring according to the multinomial model is given by

$$Pr(v|c) = \frac{(\sum_{i=1}^{d} v_i)!}{\prod_{i=1}^{d} v_i!} \prod_{i=1}^{d} \theta_i^{v_i}$$

where $\theta_i \geq 0$ and $\sum_{i=1}^{d} \theta_i = 1$.

The left (fractional) part of this equation is the normalization which allows the events to occur in any permutation order. For example (using our coin example again), there are many ways that we could have obtained 5 heads (H) and 3 tails (T): HHHHHTTT, TTTHHHHH, THTHTHHH, etc. This normalization counts the possible number of permutations of these events, since any of them may have occurred, ensuring that the sum of the probability over all vectors v is equal to 1.

The right (product) part of this equation gives a probability of each of the sub-events occurring. Continuing our coin flip example, if $v_1 = 5$ is the number of heads, then $\theta_1^{v_1} = 0.5^5$ is the probability of getting 5 heads in 5 flips, which needs to be normalized by the process described in the former paragraph.

In this work, the vectors v are fixed and the parameters c are what we will be estimating through the Expectation-Maximization algorithm. We use one multinomial model to represent one cluster, so there may be a set of k clusters (multinomials) that are summed to describe all the data. The probability of a d-dimensional vector v according to the mix-

ture of k multinomials is

$$Pr(v) = \sum_{j=1}^{k} Pr(v|c_j)Pr(c_j)$$

Here c_j is the vector of d probabilities for cluster j and $Pr(v|c_j)$ is the multinomial probability as defined above. The term $Pr(c_j)$ is the prior probability of cluster j, which is related to the size of cluster j. The sum of prior probabilities over all k clusters is 1. This $Pr(v)$ is the global probability of vector v belonging to the mixture of multinomials, where the probability of v belonging to each cluster is weighted by the probability of the cluster and summed over all clusters.

To create a mixture of multinomial models, Sanghai et al. [13] used the Expectation-Maximization (EM) algorithm [4], which is a popular iterative algorithm that is much like the k-means algorithm. The algorithm begins with a random setting of the probabilities in each multinomial model and each cluster weight. In the following, x_i means the ith vector in a set of vectors. So x_i is a vector having d dimensions, which was v above. Then it proceeds by iterating two steps:

- E-step: calculate the probability $Pr(c_j|x_i)$ using Bayes' rule (see below) for each cluster c_j and each vector x_i. This *expectation value* represents the probability that cluster c_j generated vector x_i, or the probability that vector x_i belongs to cluster c_j.
- M-step: update the d probabilities in each multinomial model, and each of the k weights, so that they give the maximum probability to the data. This step depends on the expectation values computed in the E-step. For cluster c_j, the probabilities $(\theta_1, \ldots, \theta_d)$ are updated by

$$\theta_d = \frac{\sum_{i=1}^{n} Pr(c_j|x_i)x_{id} + 1}{\sum_{i=1}^{n} Pr(c_j|x_i) + d}$$

$$Pr(c_j) = \frac{\sum_{i=1}^{n} Pr(c_j|x_i) + 1}{n + k}$$

  We add 1 to each numerator and the appropriate amount to the denominator so that we avoid zero probabilities; this is a standard technique known as Laplace smoothing [21].

The EM algorithm converges once there is little change to the model in the M-step, or equivalently when the likelihood (defined below) does not change much between iterations.

The equation for $Pr(c_j|x_i)$ comes from Bayes' rule:

$$Pr(c_j|x_i) = \frac{Pr(x_i|c_j)Pr(c_j)}{Pr(x_i)}$$

where $Pr(c_j)$ is the prior probability of cluster c_j, and $Pr(x_i|c_j)$ is the multinomial probability assigned to vector x_i by the multinomial that is associated with cluster c_j. As indicated earlier, $Pr(x_i)$ can be computed as the sum over all numerators in the above equation. The output of the EM algorithm is a set of k multinomial models (the c_j's) and their weights (prior probabilities, the $Pr(c_j)$'s).

3.4 Choosing the number of clusters with the BIC

One issue that must be addressed in clustering is the number of clusters that should be used to cluster the data. The more clusters that we choose, the more accurately we can represent the entire set of vectors that are clustered. However, in this application, the number of clusters chosen, k, is the number of simulation points picked. So this directly impacts the amount of simulation time required for each program. Therefore, we want a small number of clusters so that simulation time will be low, but we want a sufficient number of clusters to accurately capture the diversity that is represented in the program.

The prior k-means approach with SimPoint addressed this problem for multinomial clustering by performing a search over many values of k. In particular, for program intervals of 100 million instructions, we searched from $k = 1$ to 15. After performing all these clusterings, we must choose a single best k that fits the data well but does not have too many clusters. To do this, we score each clustering using the Bayesian Information Criterion (BIC) [25]. The BIC is a penalized likelihood which measures how well a clustering model fits a set of clustered vectors. The BIC for a multinomial clustering model with clusters $c_1, \ldots, c_k$ is calculated as:

$$\text{BIC} = \sum_{i=1}^{n} \log\left(\sum_{j=1}^{k} Pr(x_i|c_j)Pr(c_j)\right) - \frac{kd-1}{2}\log(n)$$

The term that contains the sum of logarithms is called the log-likelihood, which tends to increase as we increase the number of clusters. From this we subtract a penalty term that gives a larger penalty for more clusters.

Because the EM algorithm begins with a random initialization, it can find different solutions depending on the initialization. Therefore, for each k, the prior multinomial approach by Sanghai et al. [22] ran the clustering algorithm multiple times from different random starting points (random seeds). They performed 100 random initializations for each k. Then as the representative for that k, they choose the clustering that achieved the highest BIC score. When they find the best clustering for each k, they then need to choose one particular clustering. Thus they use the BIC score again, choosing the clustering with the highest score across all the k considered. This is the approach we also use in this work for multinomial clustering.

3.5 Picking of Simulation Points

Once we have a set of multinomial models and prior probabilities for the chosen k, for the purposes of SimPoint phase analysis we need to pick a set of simulation points. Sanghai *et al.* [22] proposed doing the following:

- Assign each projected vector a label according to the highest probability cluster (multinomial model) for that vector. That is, give label j to vector x_i for the highest value of $Pr(C_j|x_i)$ over all multinomial models j. This is called

hard assignment. This means that a vector is assigned to the multinomial model (center) that most represents it.

- For each cluster c_j, compute the centroid m_j of all vectors with label j. We define C_j as the set of indexes of all the vectors that are assigned label j, and we compute m_j as

$$m_j = \frac{1}{|C_j|} \sum_{i \in C_j} x_i$$

  where $|C_j|$ is the size of C_j (the number of vectors in the cluster). Recall that c_j is the vector of probabilities that represents the mean of cluster j. Thus m_j is the average of all the vectors x_i that have been assigned to belong to cluster j, and this is the centroid of that cluster. Note that c_j and m_j are similar, but they are different. The c_j is based on soft assignment calculations, and m_j is based on the hard assignment calculations (as in k-means).

- For each cluster c_j, choose the vector that is closest to m_j, as measured by Euclidean distance. Only choose from vectors that have a label of j. The interval corresponding to this vector will be the simulation point for that cluster.

With this method, there may be some clusters which have no representative chosen, for there may be no vector for which $Pr(c_j|x_i)$ is highest at j. In other words, there can be multinomial clusters that don't represent any particular vector very well, and thus no vector belongs more to that cluster more than to all other clusters. Therefore, even if we choose k multinomial models, we may choose fewer than k simulation points.

4 Improving Multinomial Clustering

In this section we describe two enhancements to the multinomial clustering approach in [22]. The two parts of the algorithm we focus on for improvement are in the feature reduction and the picking of the simulation points.

4.1 Random Projection

We examine using our prior random projection approach from SimPoint [25] to provide the feature reduction for multinomial clustering. To construct a low-dimensional set of non-negative valued frequency vectors using random linear projection, we can use random linear projection like usual. But we construct the projection matrix with the constraint that no entry in the projection matrix may be negative. This is done by filling the projection matrix with uniform random numbers between 0 and 1. This corresponds to the types of projections we have used for SimPoint with k-means. However, in our prior work we used random values between -1 and 1.

After projection, we have a low-dimensional set of vectors. Previous work [22] has used these vectors as-is, but our approach normalizes the vectors so that they sum to 1, and then multiplies by a scaling factor (e.g. 10,000). We normalize the vectors because we are dealing with fixed-length intervals, and we want each vector to represent the same amount of time from the program execution. Also, the magnitude of the sum of the vector has an impact on the probabilities given by the multinomial model, and after projection we may have vectors with very different sums. Therefore, it makes sense to normalize the vectors. This is the first important change we examine over the prior technique.

After we perform normalization, we need to scale the vectors for two reasons. The most important is that the values of a vector being used in a multinomial model are supposed to represent a count. If we simply normalize them to sum to 1, then all elements are within the range of 0 to 1, so none of them really represents a count. The second reason is that data x of small magnitude will have high probability in every multinomial model, since θ^x goes to 1 as x goes to 0, for all $\theta > 0$. Thus the clustering algorithm will not be sensitive enough to changes in the data. In other words, for every vector, every multinomial model will give high probability to that vector, so that a vector will belong about equally to all multinomial clusters. Thus the vectors will not be divided up well among clusters. Further, when the count $x < 1$, counterintuitive things happen with the probabilities; for example, $0.5^{0.5} > 0.5^{0.8}$, while $0.5^5 < 0.5^8$; so we really must scale the counts beyond 1 to get appropriate probability behavior.

For the results in this paper, we examine applying this normalization and scaling factor to both our random projection using a projection matrix filled with random numbers between 0 and 1, and the prior approach of using random projection using a projection matrix filled with only 0s and 1s.

4.2 Picking of Simulation Points

The prior approach [22] for choosing the simulation points used the cluster assignment from the multinomial model. We examine the following alternative method for choosing simulation points:

- Assign each projected vector a label according to the highest probability cluster (multinomial model) for that vector (i.e. give label j to vector x_i for the model j that gives the highest value of $Pr(c_j|x_i)$).
- For each cluster c_j, compute the centroid m_j of all vectors with label j, as shown in the previous section.
- For each interval, reassign the interval to the cluster c_j that it is closest to using the Euclidean distance between the vector and the centroid m_j. This step is new over what was done in the prior approach [22], and it can change the cluster memberships of some vectors, since membership is now based on Euclidean distance rather than multinomial probability.
- For each cluster c_j, choose the vector that is closest to m_j. Only choose from intervals that have a label of j. This interval will be the simulation point for that cluster.

This method is the same as that of the previous section except for the third step. We found that the third step is important because the prior approach [22] might not pick the

interval that is closest to the centroids that are calculated in the second step above. In step one above, the intervals were assigned to a cluster based on its highest probability of being represented by that multinomial model. But the problem is that in step two above, once all of the cluster centers are calculated, the algorithm transitions from using a multinomial model to a Euclidean distance-based model. In doing so, we should pick the interval that is closest to the centroid. Whereas, the prior technique does not permit labels to change from the multinomial labels in picking the simulation point. This misses some opportunity to pick intervals closer to the centroids with respect to the changed metric.

We also examined a third way of choosing the simulation points, where we use the same model for clustering as the model for choosing simulation points. In other words, we do not use Euclidean distance at all. In this approach the simulation point for each multinomial model (center) is chosen to be the interval that maximizes $Pr(x_i|c_j)$ for cluster c_j. Due to space we don't provide results for this, since it had higher error rates than the above approach and the prior approach.

5 Comparing Multinomial and K-Means Clustering

In this section we compare the results for using multinomial clustering in comparison to k-means, and examine a purity metric to determine when to possibly use the multinomial over the k-means approach.

5.1 Methodology

We performed our analysis for the complete set of SPEC 2000 programs for multiple reference inputs using the Alpha binaries on the SimpleScalar website. We collect all of the frequency vector profiles using SimpleScalar [3]. To generate our baseline fixed length interval results, all programs were executed from start to completion using SimpleScalar. The baseline microarchitecture model used in our prior work [20]. All processor performance data we present in this work assumes perfect warmup. For the multinomial comparison of the work in [22], Sanghai et al. generously shared their code for us to replicate their approach.

5.2 Performance of Multinomial Improvements

For all of these results, we cluster basic block vectors using a fixed length interval length of 100 million instructions. This was the interval length used in the prior multinomial clustering paper [22]. All of the CPI error rates are calculated with respect to simulating the baseline program/input from start to end. For all of the results we use the same multinomial clustering parameters as in [22], which examines clusters of size 1 to 15 and uses the BIC to choose the best one, we use 100 random restarts, and we use the number of dimensions as specified in the Table in Section 3.2.

Figures 1 and 2 show the relative error in CPI and the number of simulation points found across several configurations of the multinomial approach as well as standard k-means SimPoint. The baseline multinomial approach (KDD baseline) used in [22], is the leftmost entry in both plots. The other multinomial configurations in this plot are based on the following naming schema, and all start with the label "multi":

- sparse: Projection matrix is based on the approach in [22] described in Section 3.2, which is a matrix randomly filled with 0s and 1s. The additional change is that the projected vectors are normalized and scaled by a factor of either 1,000 or 10,000 as described in Section 4.1.
- uniform: Projection matrix is based on the SimPoint method, where the values are uniformly picked between 0 and 1 as described in Section 4.1.
- same: The simulation points are picked from the cluster assignments given by the multinomial clustering, as described in Section 3.5.
- reassign: The simulation points are picked from reassigning the intervals to their closest centroids derived from the multinomial clustering of the data, as described in Section 4.2.

The last field on each "multi" entry is the scaling coefficient (1,000 or 10,000) used on the projected and normalized vectors, which was described in Section 4.1. The standard SimPoint entries in this plot are labeled k-means-maxK-N, where N=6..10. For these configurations, we use the SimPoint 3.0 algorithm where the maximum number of clusters considered, maxK, was set to each of the values of N and the best clustering based on the BIC threshold was picked.

The results show that the KDD baseline has the highest maximum CPI error across the other configurations, but maintains a comparable average CPI error. It also picks as many or more simulation points as all the other configurations. In comparison to the results presented in [22], there were a few robustness issues with the clustering methodology in the prior work which were confirmed by the authors of that work. We got a version of their code with those issues fixed. With those fixes the results for the 8 benchmarks examined in the prior work changed so that their multinomial approach was no longer superior to k-means, sometimes it was worse, sometimes it was better.

In adding the normalization and the scaling factor to the feature reduction (multi-sparse results), we see a consistent drop in the maximum CPI error rate and average CPI error rates over the baseline KDD algorithm. The reason is that these additions ensure that each interval has the same influence on the clustering algorithm, which has an impact on the multinomial models generated as described in Section 4.1. In using our uniform random projection (multi-uniform), we actually see slightly worse error rates than when using the multi-sparse approach, but we see a nice reduction in the number of simulation points.

When adding in the re-assignment of clusters as described in Section 4.2 (these are the results with reassign in their name), we see a small decrease in CPI error rate for the multi-sparse results, and a much larger decrease in error rate

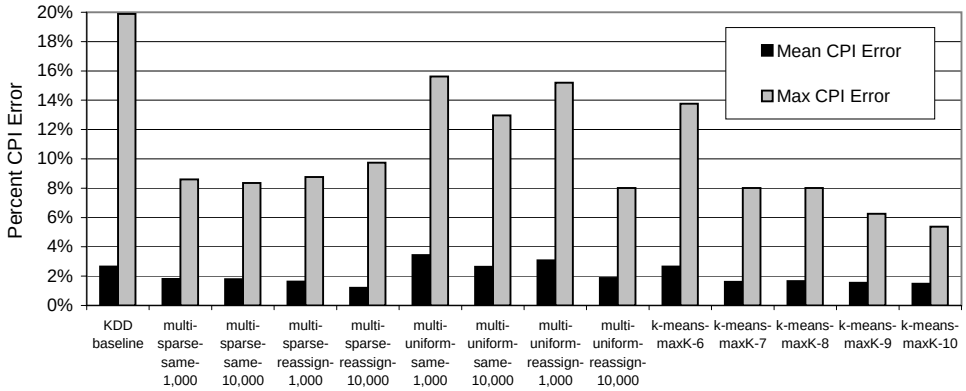

Figure 1: Average and maximum CPI error over all of the SPEC benchmarks and inputs.

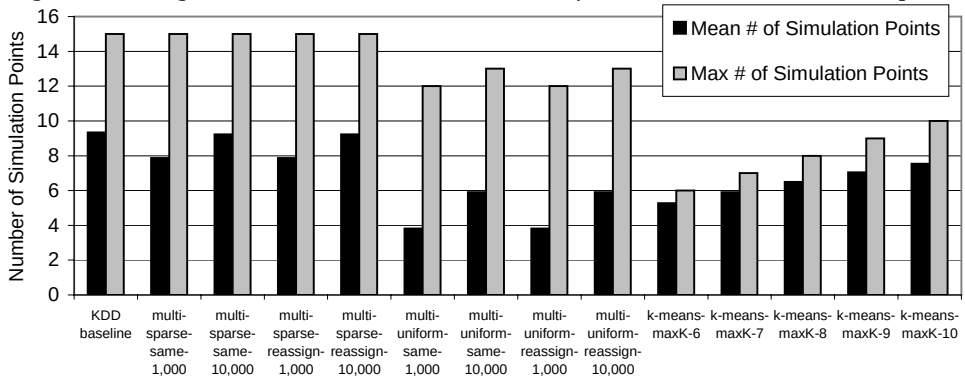

Figure 2: Average and maximum number of simulation points over all of the SPEC benchmarks and inputs.

in the multi-uniform results. The best performing multinomial technique in terms of balancing accuracy and number of simulation points was the multi-uniform-reassign-10K.

The last four sets of results in Figures 1 and 2 provide a comparison against standard SimPoint 3.0 with max K set to 6-10 using 5 random seeds. The results show that k-means with maxK of 7 and higher tend to provide the lowest error rates of all the results in these figures. Even so, in comparison to the multi-uniform-reassign-10K approach, the error rates and number of simulation points are similar. Figures 3 and 4 show the CPI error rates and number of simulation points for all of the SPEC INT and FP programs and their inputs. Due to space we only show results for multi-uniform-reassign-10K with and without the cluster purity chooser (described below) and for k-means with a maxK of 7 and 10.

The one potential advantage of the multinomial approach is that in the multi-uniform-reassign-1K result in Figure 2, we see on average 2 fewer simulation points than k-means with maxK of 7. The downside is that it has a higher error rate, but if what we care about are *relative* errors, which should be consistent across the design space exploration as shown in [20], then this may be attractive to use.

In summary, the reason why we see a consistent reduction in average error rate when performing the vector reassignment and why it is hard for the multinomial approach to beat k-means is that the SimPoint approach assigns each interval to only one specific cluster. The k-means algorithm is a hard assignment algorithm, so it specifically optimizes for this case. In terms of the multinomial algorithm used for picking simulation points, this also performs a hard assignment of a given interval to the highest probability multinomial center after clustering has finished. However, the multinomial clustering model uses soft assignment *during* clustering, where each interval is represented by multiple cluster centers. So soft-assignment EM multinomial clustering is not as well suited as k-means for a final hard assignment, and in the end choosing simulation points comes down to performing a hard assignment of a single simulation point to the cluster.

We tried a hard-assignment variant of the EM algorithm with a multinomial mixture model. The difference was that instead of using soft assignment during the expectation step, we did a hard assignment like k-means. However, this approach performed similarly to the soft-assignment with our reassignment improvement for choosing simulation points, both in error rate and number of simulation points chosen. We do not report these results here, but with the other evidence we have shown they indicate that for this application, hard assignment is a key aspect to obtaining good performance for the phase classification task. Even though EM and k-means are similar algorithms when viewed from a high level, the key differences appear to come from the use of soft

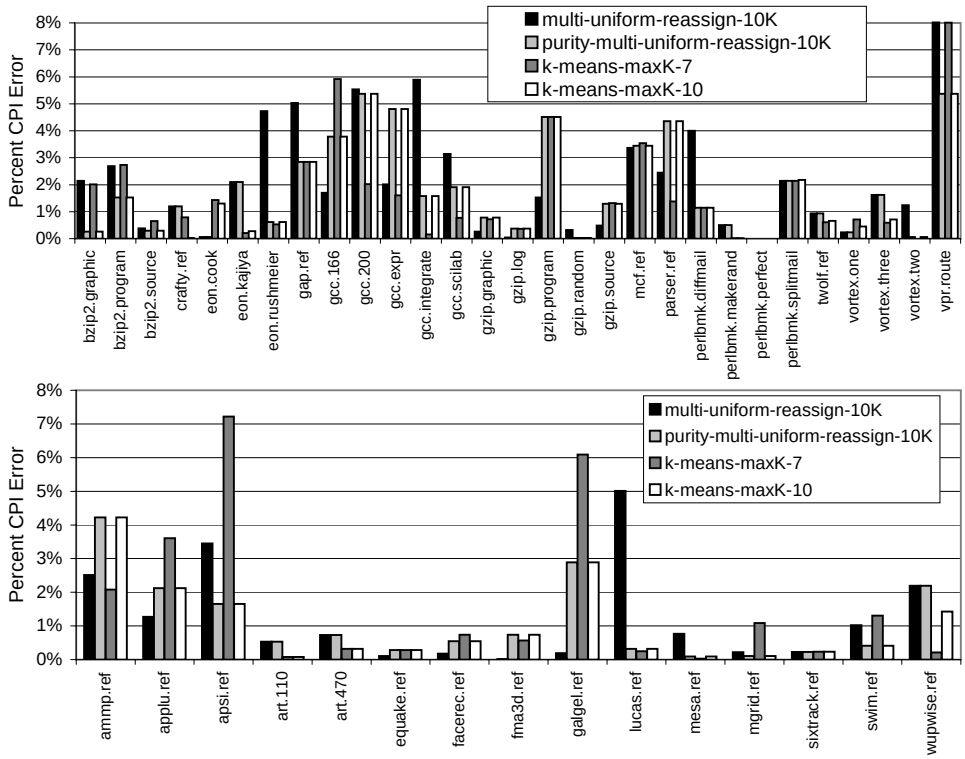

Figure 3: CPI error rates for all of the SPEC INT and FP program. Results are shown for multinomial clustering with uniform projection and reassignment of cluster labels with a scaling factor of 10,000 with and without the purity chooser. Results are also shown for k-means with maxK of 7 and 10.

or hard assignment in the algorithms.

5.3 Measuring the Purity of Multinomial Clusters

Each vector that is clustered by a mixture of multinomials with soft assignment can belong fractionally to each of the k clusters. A good question to ask then is how well can we represent each cluster with a single vector chosen from that cluster? We can use the probability $Pr(c_j|x_i)$ to quantify this. If an interval x_i is selected to represent cluster c_j, then we hope that $Pr(c_j|x_i)$ is as large (close to 1.0) as possible. We could investigate this measurement for the highest-ranked vector of each cluster. For example, if we have 3 clusters and three associated vectors, then the simulation points might have expectation values of:

	cluster 1	cluster 2	cluster 3
vector 1	0.80	0.05	0.15
vector 2	0.15	0.85	0.00
vector 3	0.00	0.05	0.95

Here $Pr(\text{cluster 1}|\text{vector 1}) = 0.80$, for example. It makes sense to look at how much each interval chosen belongs to the cluster it was chosen from, as compared to other clusters. We call this a measurement of the "purity" of that clustering. To measure this, for each cluster c_j we choose the interval x_i that maximizes $Pr(x_i|c_j)$. Call this interval s_j; note that s_j is not necessarily the simulation point, since we choose simulation points in a different manner. However, s_j is the interval that is most highly representative of cluster c_j according to the multinomial probabilities. Then we compute the purity metric as:

$$\text{purity} = \frac{\sum_{j=1}^{k} Pr(c_j|s_j)}{k}$$

The purity score for the above example would be (0.80 + 0.85 + 0.95) / 3 = 0.867. The purity score will always take a value between 0 and 1. A score of 1 means that each selected interval belongs 100% to the cluster it represents.

We can use this metric to investigate how pure a clustering is, with the expectation that a higher purity means that the data was more well-clustered with a mixture of multinomials. We found a strong correlation: when the purity is low, the multinomial approach usually has a much higher than average error rate. Note that this purity metric is based on only looking at the clustering model; no performance metrics are examined.

We propose that this measure of cluster purity can be used to select when the multinomial algorithm will do a good job of clustering the data, and when it will not. We therefore examine an approach that only uses the multinomial simulation points if the purity score is high (i.e. above 0.95), because we know that each multinomial cluster has a very strong representative. Otherwise, we resort to using the standard k-means SimPoint approach, since there is a multinomial cluster that

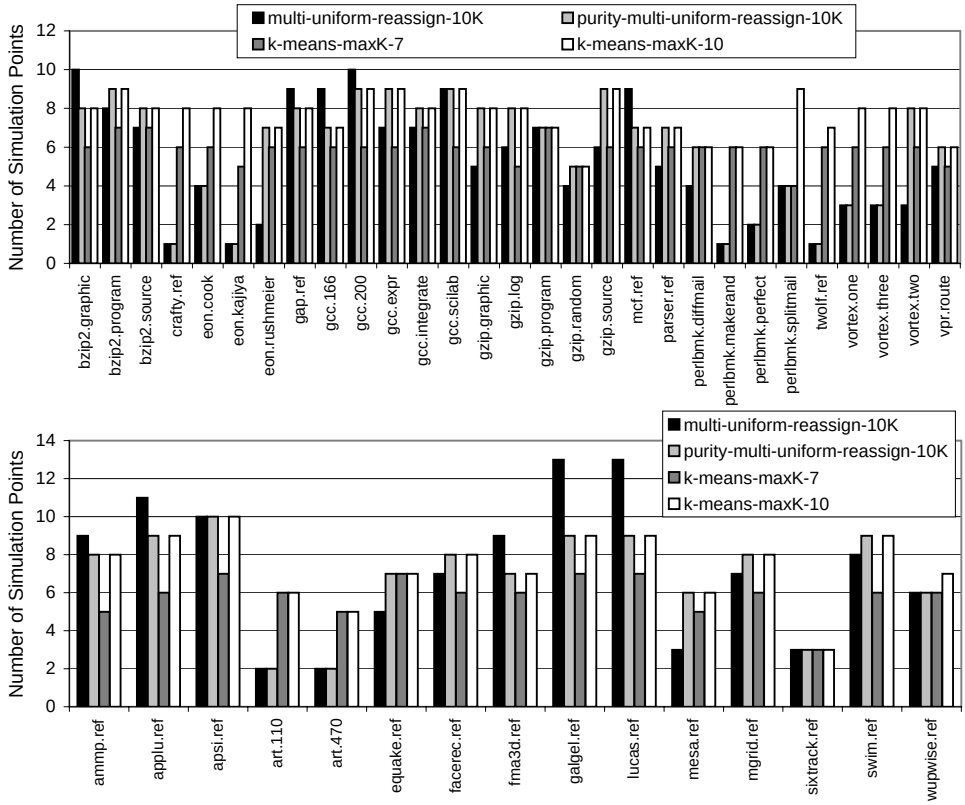

Figure 4: Number of simulation points chosen for all of the SPEC INT and FP program. Results are shown for multinomial clustering with uniform projection and reassignment of cluster labels with a scaling factor of 10,000 with and without the purity chooser. Results are also shown for k-means with maxK of 7 and 10.

does not have a strong multinomial representative. We chose 0.95 as the purity threshold because we want to be sure that when we do decide to use multinomial clustering, it is based on a very pure clustering. Looking at the data (not shown here due to space), we found that for a lower purity threshold, it may be that the resulting prediction is still good, but above 0.95 almost all the predictions were good.

Figures 5 and 6 show the average and maximum CPI error and number of simulation points for the best multinomial configuration from Figure 1 and its performance with the *purity chooser* to decide if the multinomial simulation points should be used or the k-means simulation points. The purity results represent using the multinomial approach if the purity metric is above 95%, otherwise using k-means with maxK set to 10. In these plots we also include standard k-means SimPoint with maxK=7 and maxK=10 as a comparison. The detailed results of these statistics are seen in Figures 3 to 4, where the Spec 2K FP and INT benchmarks are plotted for the same configurations. For the purity-multi-uniform-reassign-10K results, 29% of the programs were chosen to be clustered with multinomials, and the remaining with k-means. For the purity-multi-sparse-reassign-10K results, 25% of the programs were chosen to be clustered with multinomials.

The benefit of using the purity score can be seen from these results. The average CPI stays about the same when using the purity chooser, but we found that the programs with high error rates are flagged, and a lower error rate is then achieved by using k-means. This improvement in accuracy is a result of finding the programs where there are clusters without good representation in the multinomial approach based on the purity score. SimPoint has better accuracy in general, and can do a better job at representing them although it may require more simulation points. By combining the two techniques we can achieve a better accuracy and fewer simulation points over either solo method.

5.4 Timing Comparison

One difference we did notice between the multinomial approach and k-means is the running time of the two algorithms. We found the multinomial approach to be around 10 times slower than using k-means. Both techniques have linear run-time in the size of the input data and the number of clusters used. However, performing multinomial clustering with EM requires many more floating point operations, and it cannot be optimized nearly as well as k-means. Because k-means uses hard assignment throughout the algorithm, it can be heavily optimized so that not all computations need to be fully performed. For example, k-means relies heavily on finding the shortest distance between a vector and all the k

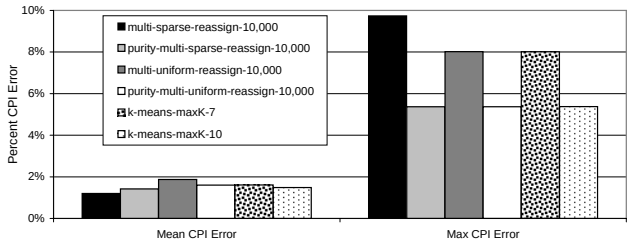

Figure 5: Average and maximum CPI error over all of the SPEC programs and inputs for using the full multinomial approach, using the full k-means approach, and then using the purity chooser. The purity results represent using that multinomial approach if the purity metric is above 95%, otherwise using k-means with maxK set to 10.

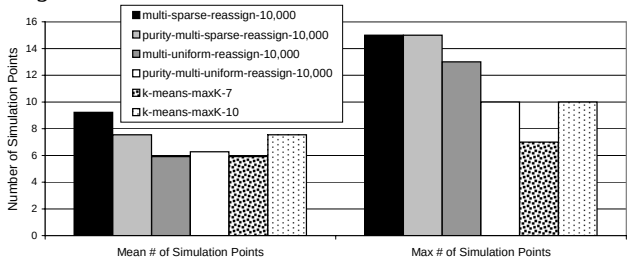

Figure 6: Average and maximum simulation points over all of the SPEC programs and inputs for using the full multinomial approach, using the full k-means approach, and then using the purity chooser. The purity results represent using that multinomial approach if the purity metric is above 95%, otherwise using k-means with maxK set to 10.

centers. If you have already computed the distance between an interval and center c_1 as 3.0 and you are computing the distance to c_2, you may find that partway through the distance computation the distance will be greater than 3.0. If this is the case, then you can stop the computation for c_2, since you only care about the minimum distance, and clearly c_2 will not be the minimum. This example is called partial distance search [18]. The hard assignment of k-means makes it a great candidate for these types of optimizations. Conversely, because the EM algorithm must calculate the probability of every interval according to every model, there is little that can be done to optimize it.

Figure 5.4 shows the time in seconds for running k-means and the multinomial algorithm varying the number of vectors as we vary the number of clusters (value of k). For this experiment, the interval vectors are randomly generated from uniformly random noise in 15 dimensions.

The results show that as the number of vectors and clusters increases, so does the amount of time required to cluster the data. The first graph shows that for 100,000 vectors and $k = 128$, it took about 3.5 minutes for k-means to perform the clustering. For multinomial clustering it took roughly 64 minutes to cluster the same amount of data. Both of these timings are for one clustering for one random initialization. It is clear that the number of vectors clustered and the value of

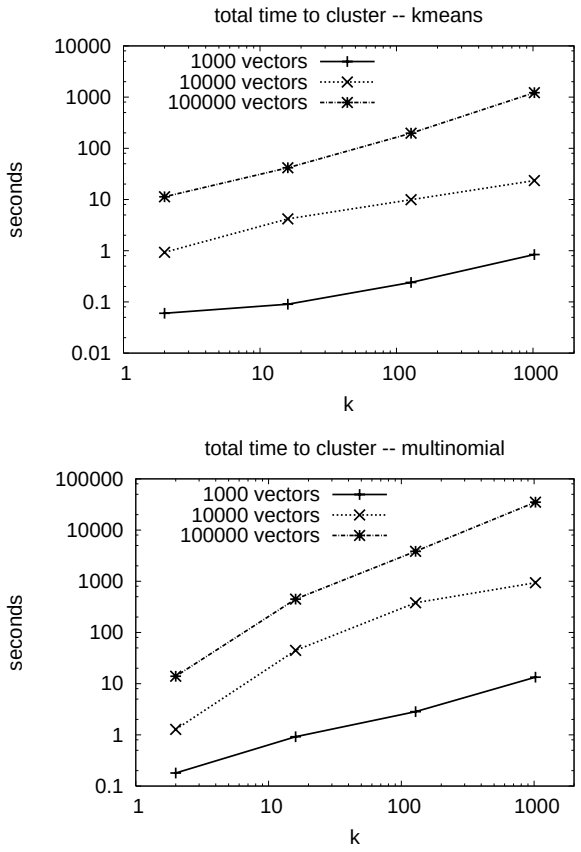

Figure 7: Timing results for clustering with k-means and a mixture of multinomials.

k both have a large effect on the run-time of k-means and the multinomial algorithm: the run-time changes linearly with the number of clusters and the number of vectors. Finally, we find that k-means gives good performance with only 5 random initializations per k, whereas following [22] we find we need 100 random initializations per k for multinomial mixture clustering in order to find a consistently good clustering. The need for more random initialization further increases the timing gap between the two algorithms.

6 Summary

Characterizing the behavior of program execution has become crucial for modern computer architecture research, especially in the application of processor simulation. SimPoint automates the process of picking simulation points using a phase classification algorithm based on k-means clustering, which significantly reduces the amount of simulation time required. By simulating only a handful of intelligently picked sections of the full program, the simulation points provide an accurate picture of the complete execution of a program, which gives a highly accurate estimation of performance.

A different clustering algorithm is explored in this work, based on the multinomial model [22]. We examine the base-

line approach as proposed in [22] and compare its performance to k-means SimPoint [20, 25] and find that k-means has better performance, both in accuracy and number of simulation points.

We then propose two optimizations to the baseline approach [22]. First we find that normalizing and scaling the projected data results in a better clustering of the data. Secondly, we propose a different method for picking the simulation points from the clusters. We use the multinomial model for grouping the intervals into clusters and to calculate those clusters' centroids, just as in the prior technique. But after this step we choose the interval that is closest to a centroid from any intervals, not restricting them based upon the multinomial model labeling. We found that this helps reduce the maximum error rates seen for the SPEC benchmarks.

Finally we propose a new metric, cluster-purity, for determining how effective the multinomial algorithm is at characterizing a program. The cluster-purity metric is used to score the multinomial clustering, and based on its score we decide if this is a good representation. The benefit with such a score is that we can use it to choose between the two clustering techniques. Based on the purity score we can decide whether we should use the multinomial representation or if we should use the SimPoint k-means representation instead. The multinomial approach does not excel in accuracy as SimPoint does, but on average it picks fewer simulation points. This combination of the two techniques results in a small improvement in accuracy (mainly helping the programs that had a high error rate) with a small reduction in the average number of simulation points.

Acknowledgments

We would like to thank David Kaeli, Kaushal Sanghai, and the anonymous reviewers for providing helpful feedback on this paper. This work was funded in part by NSF grant No. CCF-0342522, NSF grant No. CCF-0311710, a UC MICRO grant, and a grant from Intel and Microsoft.

References

[1] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *33rd International Symposium on Microarchitecture*, pages 245–257, 2000.

[2] M. Van Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *IEEE International Symposium on Performance Analysis of Systems and Software*, March 2004.

[3] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997.

[4] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:185–197, 1977.

[5] P.J. Denning and S. C. Schwartz. Properties of the working-set model. *Communications of the ACM*, 15(3):191–198, March 1972.

[6] A. Dhodapkar and J. E. Smith. Dynamic microarchitecture adaptation via co-designed virtual machines. In *International Solid State Circuits Conference*, February 2002.

[7] A. Dhodapkar and J. E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *29th Annual International Symposium on Computer Architecture*, May 2002.

[8] A. Dhodapkar and J.E. Smith. Comparing program phase detection techniques. In *36th Annual International Symposium on Microarchitecture*, December 2003.

[9] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *12th International Conference on Parallel Architectures and Compilation Techniques*, October 2003.

[10] M. Hind, V. Rjan, and P. Sweeney. Phase shift detection: A problem classification. Technical report, IBM, August 2003.

[11] C. Isci and M. Martonosi. Identifying program power phase behavior using power vectors. In *Workshop on Workload Characterization*, September 2003.

[12] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *36th International Symposium on Microarchitecture*, December 2003.

[13] D. Kaeli. *Issues in Trace-Driven Simulation*. Lecture Notes in Computer Science No. 729, Performance Evaluation of Computer and Communication Systems,L. Donatiello and R. Nelson eds., Springer-Verlag, 1993, pp. 224-244., 1990.

[14] J. Lau, E. Perelman, G. Hamerly, T. Sherwood, and B. Calder. Motivation for variable length intervals and hierarchical phase behavior. In *IEEE International Symposium on Performance Analysis of Systems and Software*, March 2005.

[15] J. Lau, J. Sampson, E. Perelman, G. Hamerly, and B. Calder. The strong correlation between code signatures and performance. In *IEEE International Symposium on Performance Analysis of Systems and Software*, March 2005.

[16] J. Lau, S. Schoenmackers, and B. Calder. Structures for phase classification. In *IEEE International Symposium on Performance Analysis of Systems and Software*, March 2004.

[17] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. LeCam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, CA, 1967. University of California Press.

[18] J. McNames. Rotated partial distance search for faster vector quantization encoding. *IEEE Signal Processing Letters*, 7(9), 2000.

[19] D. Pelleg and A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the 17th International Conf. on Machine Learning*, pages 727–734, 2000.

[20] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *International Conference on Parallel Architectures and Compilation Techniques*, September 2003.

[21] Eric Sven Ristad. A natural law of succession. Technical Report TR-495-95, Princeton University, 1995.

[22] Kaushal Sanghai, Ting Su, Jennifer Dy, and David Kaeli. A multinomial clustering model for fast simulation of computer architecture designs. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 808–813, New York, NY, USA, 2005. ACM Press.

[23] T. Sherwood and B. Calder. Time varying behavior of programs. Technical Report UCSD-CS99-630, UC San Diego, August 1999.

[24] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *International Conference on Parallel Architectures and Compilation Techniques*, September 2001.

[25] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *10th International Conference on Architectural Support for Programming*, October 2002.