Analyzing and Improving Clustering Based Sampling for Microprocessor Simulation

Yue Luo, Ajay Joshi, Aashish Phansalkar, Lizy John, and Joydeep Ghosh
Department of Electrical and Computer Engineering
University of Texas at Austin
{luo, ajoshi, aashish, ljohn, ghosh}@ece.utexas.edu

Abstract

We propose a set of statistical metrics for making a comprehensive, fair, and insightful evaluation of features, clustering algorithms, and distance measures in representative sampling techniques for microprocessor simulation. Our evaluation of clustering algorithms using these metrics shows that CLARANS clustering algorithm produces better quality clusters in the feature space and more homogeneous phases for CPI compared to the popular k-means algorithm. We also propose a new micro-architecture independent data locality based feature, Reuse Distance Distribution (RDD), for finding phases in programs, and show that the RDD feature consistently results in more homogeneous phases than Basic Block Vector (BBV) for many SPEC CPU2000 benchmark programs.

1. Introduction

Cycle-accurate microarchitecture simulation is one of the most important tools in computer architecture research. However, it is often prohibitively time-consuming. Researchers have proposed taking advantage of the well-observed phase behavior in program execution to reduce simulation time. A *phase* can be defined as a portion of dynamic execution of a program for which most of the performance metrics such as Cycle Per Instruction (CPI), show very little variance. Because the performance metrics remain stable in a phase, simulating only one chunk of instructions from each phase to represent the execution of the whole program can greatly reduce simulation time with little loss of simulation information. We call this type of technique *phase based representative sampling*, or *representative sampling* hereafter. Three recently proposed schemes fall into this category [2][3][4]. These techniques estimate some target metric (e.g. CPI, energy per instruction, or cache miss rate) by taking advantage of the phase behavior. To identify phases, they divide the dynamic instruction stream into chunks of instructions, and for each chunk, measure a feature vector that is distinguishable between phases. We refer to such a feature vector as a *phase classification feature vector*, or simply a *feature* as it is often called in the data mining community. Then, cluster analysis is performed to group the chunks into clusters. Each cluster corresponds to a phase because the chunks in the same cluster exhibit very similar phase classification feature.

The major characteristics of the three representative sampling techniques are summarized in Table 1. Out of the three techniques, SimPoint [2] is the most popular. It uses Basic Block Vector (BBV) as the phase classification feature. BBV is a vector whose elements are frequencies of dynamic execution of static basic blocks in the program. Because of its popularity and BBV's advantage of microarchitecture independence, we base our study mainly on the approach of SimPoint[1].

Table 1. Recently proposed phase based representative sampling techniques

Technique	Target Metric	Phase classification feature	Clustering algorithm	Chunk size (million instructions)
SimPoint	IPC	BBV	k-means	100
SPEClite [3]	29 Performance Metrics	Performance Counter Data	k-means	1
Lafage and Seznec [4]	Data Cache Miss-Rate	Data Reuse Distance	Hierarchical	1

Although representative sampling is becoming popular for microprocessor simulation, the design space has not been well explored and many questions are still unanswered. This research addresses the following important questions:

This research is supported by the National Science Foundation under grant numbers 0429806 and 0307792, the IBM Center for Advanced Studies (CAS), an IBM SUR grant, and by Intel and AMD Corporations.

[1] Variance SimPoint and Early SimPoint [11] extended the original SimPoint, but they require explicit warm-up. In this paper we focus on the original SimPoint, but our methodology is applicable to all representative sampling techniques.

Choice of clustering algorithms and distance measures: It is well known in the data mining community that no single clustering algorithm or distance measure is well suited for all applications. Many clustering algorithms (k-means, hierarchical etc.) and distance measures (Euclidean distance, Manhattan distance, cosine distance etc.) have been proposed for different application domains. However, previous research does not compare the quality of various clustering algorithms and distance measures for identifying simulation points using representative sampling.

Evaluation methodology: How to fairly evaluate the effectiveness of new clustering algorithms, new distance measures, and new phase classification features in representative sampling is another question that has not been studied previously. Of course, the final error in target metric can be used to compare different approaches. However, as we will show in Section 2, this is not a reliable method. A fair and comprehensive evaluation methodology for representative sampling is needed.

Choice of phase classification feature: The third question we study is which phase classification feature to use in representative sampling. For modern microprocessors, the data access latency is one of the most important factors that determine the performance. Therefore, is it possible to get better result by designing a data locality based feature?

The contributions of this paper are three-fold:

1) We propose a systematic method to fairly evaluate new clustering algorithms, new distance measures, and new phase classification features for representative sampling. Our methodology also helps the user to gain better understanding of the sampling technique.
2) We investigate the effectiveness of using different clustering algorithms and different distance metrics.
3) We propose a new microarchitecture-independent data locality based feature, Reuse Distance Distribution, for identifying phases in a program. We show that for a set of benchmarks it consistently produces more homogenous phases than BBV.

The paper is structured as follows. Our evaluation methodology is proposed in Section 2. Different clustering algorithms and distance measures are evaluated in Section 3. In Section 4 we propose our new phase feature, RDD, and compare it with BBV. In Section 5 we draw conclusions from this study.

2. Evaluation Methodology

The accuracy of a representative sampling based simulation technique depends on the correlation between the phase classification feature and the target metric, the clustering algorithm, and the choice of the data point used to represent the phase.

Therefore, our evaluation methodology consists of three components in both the feature and target metric space. First, we examine the cohesiveness of clusters in the phase classification feature space. Then, we measure the homogeneity of target metric in each phase. And lastly, we look at the final sampling error. In this study, we present results for all cluster numbers between 4 and 10, which covers most of the range of the number of clusters used in SimPoint.

In the phase feature space, the cohesiveness of clusters can be measured by the Average Distance (AD) from each data point to the representative data point of the cluster it belongs to.

$$AD = \sum_{x} distance(x, c_i) / n,$$

where $\boldsymbol{x}$ is a data point and $\boldsymbol{c_i}$ is the representative data point for the cluster that $\boldsymbol{x}$ belongs to and $\boldsymbol{n}$ is the total number of data points. This metric can be used to compare different clustering algorithms with the same phase classification feature and the same distance measure. A better clustering algorithm will give tighter clusters and thus a smaller average distance.

We use the Normalized Standard Deviation (NSD) metric, defined as follows, to evaluate the homogeneity of phases in the target metric space.

$$NSD = \sqrt{(\sum_{i=1}^{k} \frac{n_i}{n} S_i^2)} / S,$$

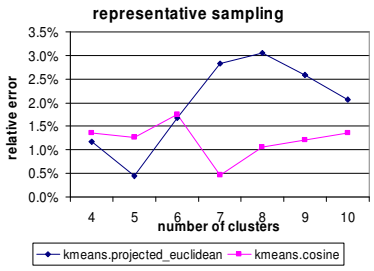
where $\boldsymbol{n_i}$ is the size of cluster $\boldsymbol{i}$. $\boldsymbol{S_i}$ is the standard deviation of the target metric in cluster $\boldsymbol{i}$, n is the total number of data points, and $\boldsymbol{S}$ is the standard deviation of the target metric for all data points. NSD reflects the tightness of the cluster in the target metric space. NSD measures the benefit of doing cluster analysis as compared to simple random sampling (see [6][13] for details). The lower the normalized standard deviation gets, the more homogeneous the phases are. A value much smaller than 1 and close to 0 is desirable. Since the calculation of NSD only involves the target metric, it can be used to compare different clustering algorithms, different distance measures, and different phase classification features.

We also examine the final Relative Error (RE) in target metric compared to full cycle-accurate simulation, which is defined as,

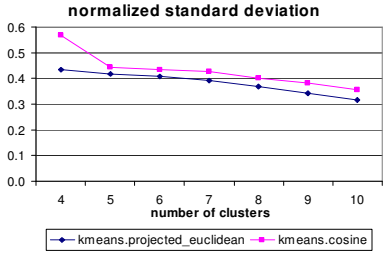
$$RE = \left| \sum_{i=1}^{k} \frac{n_i}{n} y_i - \bar{y} \right| / \bar{y},$$

where y_i is the target metric of the representative data point in cluster i. $\bar{y}$ is the true target metric of the benchmark. The relative error is determined not only by the quality of the clustering and the correlation between the feature and target metric, but also by how close the target metric for the representative data point is to the mean target metric of a cluster. Therefore, the error in a representative sampling based simulation technique is the result of several factors lumped together, and provides little insight into the relative advantages and disadvantages of each technique.

RE is the final metric that the user cares about. But unlike NSD, which is calculated from all n data points, the relative error is affected by only k representative data points. Since $k<<n$, it is far less stable than NSD. Consider, for example, *bzip2-source* from SPEC CPU2000 benchmark suite. Figure 1b shows NSD in CPI for two distance measures, Euclidean distance with random projection and cosine distance. As expected, when we divide the data points into more clusters, the overall homogeneity improves. In contrast, Figure 1a shows the relative error for CPI of the simulation using representative sampling. The error curve does not have a clear trend and is much more "messy", making comparison very difficult.



(a) Relative Error in CPI for representative sampling



(b) Normalized Standard Deviation in CPI for representative sampling

Figure 1. Comparing projected Euclidean distance with cosine distance for *bzip2-source*.

Based on the above analysis, we use NSD as our main evaluation metric in the target metric space.

All three metrics (AD, NSD, and RE) are needed to evaluate a representative sampling technique, and every metric provides a different insight. Improving one metric may not automatically make other metrics better. Suppose we use a better clustering algorithm and get smaller average distances, but NSD does not improve, then we know it is because the correlation between the phase classification feature and the target metric is not strong enough. We need to search for a better phase classification feature. If we get more homogeneous CPI in each phase (i.e. smaller NSD) but the final error remains large, then it indicates that the error introduced by picking the central data point dwarfs our improvement in homogeneity.

3. Evaluation and analysis of clustered sampling using BBV

3.1 Comparing clustering algorithms and distance measures

Although k-means clustering algorithm is popular in representative sampling, it is very sensitive to outliers and performs well only on clusters that are spherical and have the same variance. In addition, the centroid of a cluster rarely coincides with a real data point. Therefore, we would like to evaluate a different algorithm against the k-means algorithm. We choose k-medoid method because it overcomes two limitations of k-means. First, k-medoid methods are less strict about the distribution of the data points and are robust to the existence of outliers. Second, in k-medoid methods, the medoid, which is a real data point, naturally represents the cluster. There are several k-medoid methods. We choose to use CLARANS algorithm, proposed by Ng and Han [7][8], because of its lower computation cost, which is basically linearly proportional to the number of data points.

In all three representative sampling techniques we reviewed, Euclidean is used to measure the dissimilarity between the chunks of instructions. In this study the cosine distance is of special interest. It has been successfully used in automatically clustering documents into different topics. Documents are often represented as vectors, where each element is the frequency with which a particular term occurs in the document. If we compare BBV with a text document vector, we can see their similarity. An element in BBV is the number of times a specific static basic block is executed, which corresponds to the number of times a specific word occurs in a document. In addition, both are very high dimensional vectors (thousands to more than a hundred thousand dimensions). Therefore, it is very interesting to see whether cosine distance can be applied to representative sampling. If p and q are two vectors, then the cosine distance is defined as

$$cosine\_distance(p, q) = 1 - \frac{p \bullet q}{\|p\| \cdot \|q\|},$$

where $\bullet$ indicates vector dot product, and $\|p\|$ is the length of vector p. Because the result is divided by the norm of the vectors, the cosine distance is really a measure of the angle between p and q. If the angle is 0°, then the two vectors are the same except for the magnitude. The cosine distance will be 0, which is the minimum value. If the angle is 90°, then the two vectors do not share any elements. In other words, the code in the two chunks of instructions are complete different because they do not share any basic blocks. In this case, the cosine distance reaches the maximum value of 1.

3.2 Experiment setup

We use 8 programs with the reference data set from the SPEC CPU2000 benchmark suite. The programs and the number of instructions are listed in Table 2.

Table 2. Number of instructions and simulation time of selected SPEC CPU 2000 benchmarks with reference data set. The data set name is appended to the benchmark name.

Benchmark-Input Pair	Number of instructions (million)
art-110	41,798
bzip2-source	108,878
equake	131,518
gcc-166	46,917
lucas	142,398
mcf	61,867
vortex-1	118,976
vpr-route	84,068

Following SimPoint, we divide the instruction stream of each program into intervals of 100 million instructions. In all our experiments we use CPI as the target metric. To evaluate the result in the target space, we simulate all 8 benchmarks in *sim-outorder* [1] to collect CPI for each chunk. The processor configuration used in the simulation is shown in Table 3. The same configuration has been used in study on cache warm-up [9] and in validation of *SimPoint* [10]. K-means and CLARANS clustering algorithms are each evaluated using projected Euclidean and cosine distance measures. Because clustering algorithms are less effective at high dimensional Euclidean space, the dimensionality of BBV is reduced to 15 through random projection just as in SimPoint. Thus it is given the name "projected Euclidean distance". Both k-means and CLARANS may give different results with different random seeds. Therefore, we run each experiment 5 times with different random seeds and the data shown below are the average results.

Table 3. Processor configuration.

Pipeline	
Issue Width	8 instructions/cycle
Decode Width	8 instructions/cycle
Register Update Unit	128 entries
Load-Store Queue	32 entries
Commit Width	8 instructions/cycle
Cache Hierarchy	
L1 Data	16KB; 4-way assoc., 32B lines, 2-cycle hit
L1 Instruction	8KB; 2-way assoc., 32B lines, 2-cycle hit
L2 Unified	1MB; 4-way assoc., 64B lines, 20-cycle hit
Memory Access Latency	151 cycles
Combined Branch Predictor	
Bimodal	8192 entries
PAg	8192 entries
Return Address Stack	64 entries
Branch Target Buffer	2048 entries; 4-way assoc.
Misprediction Latency	14 cycles

3.3 Experiment results

Following the evaluation methodology proposed in Section 2, we first evaluate the two clustering algorithms in the BBV space. Since average distance metric cannot be compared between different distances, the result for projected Euclidean distance and cosine distance are drawn separately. Figure 2 compares the average distance of k-means and CLARANS algorithms for projected Euclidean distance for 2 program-input pairs (see [13] for complete graphs for Euclidean distance and cosine distance). CLARANS clearly produces tighter clusters than k-means in most cases. For some benchmarks, such as *equake* and *gcc-166* the reduction in average distance is significant. At 10 clusters, for projected Euclidean distance, CLARANS reduces the distance by over 50%, while for cosine distance the reduction is almost 90%.

We then examine the normalized standard deviation, which is shown in Figure 3 (the graphs for remaining 6 programs can be found in [13]). The NSD shows a downward trend. Therefore, as we increase the number of phases, the CPI in each phase shows lesser variance. In general, CLARANS algorithm produces more homogeneous CPI phases in most cases. Table 4 shows the clustering algorithm and distance measure with the overall lowest normalized standard deviation. In 5 (or 6) out of 8 benchmarks, CLARANS is better than k-means. The cosine distance we experimented with, on the other hand, does not seem to perform better than the projected Euclidean distance. Cosine distance is the best for only *vortex-1*.

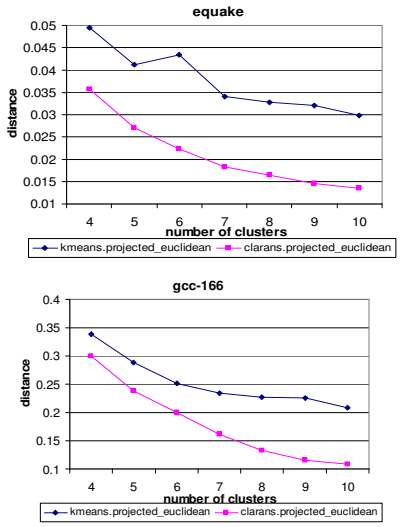


Figure 2. Average distance for different clustering algorithms and distance measures

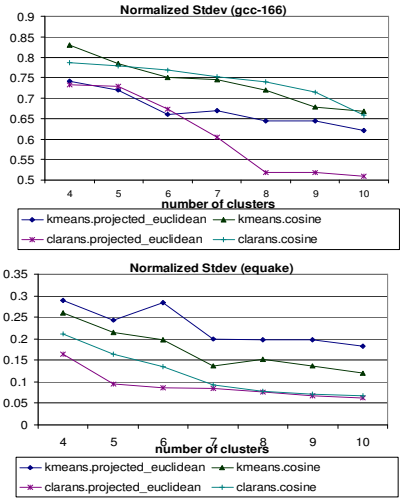


Figure 3. Normalized standard deviation for different clustering algorithms and distance measures

Figure 4 shows the relative error of representative sampling using different clustering algorithms and distance measures. For brevity, only two benchmarks are shown, the complete set of graphs can be found in [13]. Since the normalized standard deviation decreases as more phases are identified, the errors are expected to follow suit. However, they do not show a general trend and vary a lot, crossing each other multiple times, making it almost impossible to identify the one with consistently lowest error. If we only focus on projected Euclidean distance (used in SimPoint), then CLARANS consistently results in smaller error for *equake* and *vortex-1*. For the remaining 6 benchmarks, they are comparable.

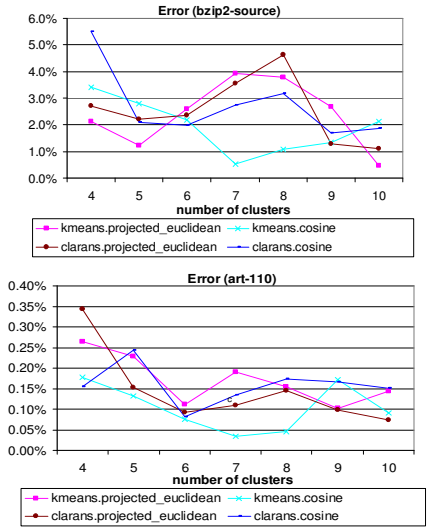


Figure 4. Relative error in the CPI from representative sampling for different clustering algorithms and distance measures for *bzip2-source* and *art-110* programs.

Table 4. Clustering algorithm and distance measure with the lowest normalized standard deviation (i.e. the most homogeneous phases).

Benchmark	Best clustering algorithm and distance measure
art-110	Kmeans with projected Euclidean distance
bizp2-source	CLARANS with projected Euclidean distance
equake	CLARANS with projected Euclidean distance
gcc-166	CLARANS with projected Euclidean distance
lucas	No clear winner
mcf	CLARANS with projected Euclidean distance
vortex-1	CLARANS with cosine distance
vpr-route	Tie between k-means and CLARANS with projected Euclidean distance

4. Reuse Distance Distribution - A new feature for phase classification

Due to the gap between processor and memory performance, data access latency is one of the most important factors that determine program performance in modern day microprocessors. Data access latency is a function of the inherent locality in the data address stream of a program. Therefore, we feel that a feature based on the data locality of a program will be able to find phases in a program that have similar data locality, and that show similar performance. It is important that the locality feature should be microarchitecture independent. This increases the confidence that the phases identified by the feature will be valid across different microarchitectures.

The BBV feature does not capture the properties of the data locality of the program. Therefore, it is

possible that a static section of code in a program has different memory reference patterns at different points of time in its execution. A feature based on the data locality in a program will be able to capture this behavior, and may therefore perform better than BBV in finding more homogeneous phases.

4.1 Reuse Distance Distribution (RDD) definition

Reuse distance is the number of memory addresses accessed between the accesses to the same memory line. A memory line is analogous to a cache block. We define the RDD feature as the relative frequencies of the different reuse distances in the data address stream of a program. The reuse distances can have a large number of unique values. To make the distribution more manageable, we group similar reuse distances together. The width of each interval in the histogram is exponentially distributed – i.e. a reuse distance of r is classified into interval $\lceil \log_e r \rceil$. In general, interval i consists of reuse distances from e^{i-1} to e^i. We can represent the RDD feature as a vector with n elements, where element i is the relative frequency of the number of reuse distances for interval i. For a given memory line size, the RDD feature characterizes the temporal locality of the data memory address stream. Information about the spatial locality of a memory address stream can be characterized by measuring the RDD feature for a range of different memory line sizes. The example in Section 7.1 of our technical report [13] is a simple illustration of how the RDD feature is calculated for an address stream.

Similar concepts based on reuse distance of memory accesses have been used in many areas such as cache warm-up [9] and program execution and performance modeling [14][15]. Our definition of reuse distance is a little different than the one in [15]. In [15], reuse distance is equal to number of unique addresses accessed between the accesses to the same memory address. But this reuse distance measure is very computation intensive so we use the one described earlier in this section. Two pervious works were in representative sampling. Lau et al. [5] extensively examined different phase features based on data accesses. Their features are related to the access patterns, but do not directly measure the data locality. Since it is the data locality that impacts program performance, these phase features may not be a strong enough indicator of the program performance. Lafage et al. [4] used average memory reuse distance (RDI) to identify phases for simulation for the data cache miss-rate. We experimented with the average reuse distance and found that BBV performed better in all the benchmarks we considered. By measuring the distribution of the reuse distance instead of the aggregated average, we have greatly improved the accuracy over RDI.

4.2 Comparing RDD and BBV features for phase classification

As described in Section 2, normalized standard deviation is a more reliable and insightful metric than the final error in CPI for comparing two phase classification features. We therefore used normalized standard deviation as the performance metric for comparing the RDD and BBV features. We used the 8 programs listed in Table 1. RDD feature (for memory line sizes of 16, 64, 256, and 4096 bytes) and the BBV features are measured for every interval of 100 million instructions. For clustering the points in the RDD feature space, we used the CLARANS and k-means clustering algorithm each with Euclidean and cosine distance measures. The processor configuration in Table 3 was simulated.

In order to make a meaningful comparison between the two features, for every program, we selected the best (smallest NSD in CPI) algorithm-distance pair for the RDD feature, and compared it with the best algorithm-distance pair for the BBV feature. Figure 5 shows a plot of the best algorithm-distance pair for RDD and BBV features for the 8 program-input pairs used in this study.

From these graphs we observe that, irrespective of the number of clusters, the RDD feature gives lower normalized standard deviation in CPI than the BBV feature for *gcc-166, lucas, mcf, vpr-route,* and *art-110* programs. For *lucas*, the normalized standard deviation for 4 clusters formed using the RDD feature is even smaller than the normalized standard deviation from forming 10 clusters using the BBV feature. This shows that *lucas* benefits far more from the RDD feature than from the BBV feature. Forming more than 4 clusters using the BBV feature does not significantly benefit *mcf*. The normalized standard deviation given by the BBV feature for 4 to 10 clusters is not very different, showing that the benefit from increasing the number of clusters is minor. However, the normalized standard deviation for *mcf* using RDD feature is not only smaller than that of BBV, but also drops from 0.45 to 0.33 as the number of clusters are increased from 4 to 10. The normalized standard deviation for *art-110* is very small (0.25 for BBV feature and 0.22 for RDD feature) even when only 4 clusters are fromed. This shows that *art-110* substantially benefits from clustering using the BBV or RDD feature. Although the normalized standard deviation for *art-110* is small for the BBV feature, the RDD feature approximately improves it by additional 14%

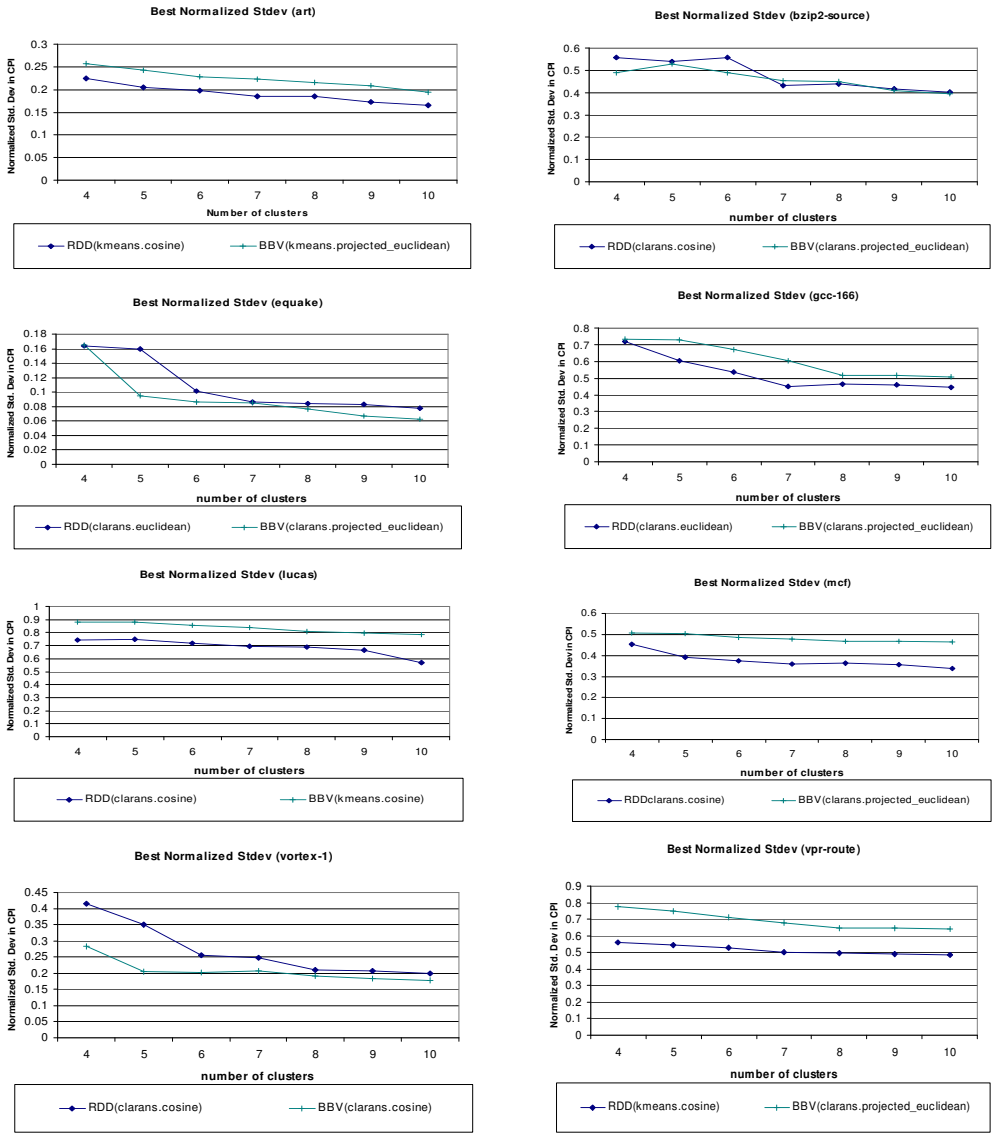


Figure 5. Normalized standard deviation in CPI for BBV and RDD feature for 8 program-inputpairs from SPEC CPU2000 benchmark suite.

irrespective of the number of clusters. For *bzip2-source*, the graphs for both the features cross each other and hence we conclude that neither of the features clearly outperforms the other. However, for *vortex-1* and *equake*, the BBV feature always gives a lower normalized standard deviation in CPI than the BBV feature.

For this processor configuration, the RDD feature is consistently better than the BBV feature for 5 out of the 8 benchmark programs. However, it is possible that the RDD feature is better than BBV just for the microarchitecture configuration selected for this experiment. Therefore, we felt that it is important to evaluate RDD and BBV features on more microarchitectures. We selected two additional configurations, same as those used for evaluation of SMARTS sampling methodology [12]. The complete results can be found in [13]. We again observe that the RDD feature gives lower normalized standard deviation in CPI than the BBV feature for the same five programs, *gcc-166, art, lucas, mcf,* and *vpr-route*. For *bzip2-source*, neither of the feature clearly outperformed the other. The validation using three different microarchitecture configurations has

increased our confidence that the results are independent of the microarchitecture and are generally applicable.

We also evaluated the effectiveness of different clustering algorithms and distance measures for RDD. The results agree with our observation in Section 3. In the feature space, CLARANS clustering algorithm shows a lower AD and generally produces more homogeneous phases in CPI.

5. Conclusions

In this paper, we proposed a set of statistical metrics for evaluating representative sampling techniques, and showed that these metrics are reliable, insightful, and provide a deeper understanding of the quality of clustering in representative sampling. We used these metrics to evaluate the benefit from using CLARANS clustering algorithm and the cosine distance measure. We proposed and evaluated a new data locality based microarchitecture independent feature, RDD, for phase classification in a program.

Our experiments showed that in the feature space for both BBV and RDD features, for all benchmarks, CLARANS produces more cohesive clusters than the k-means clustering algorithm. CLARANS algorithm also results in more homogeneous phases in CPI for many, but not all, benchmarks. Therefore, a better clustering algorithm can improve the quality of clustering in the feature space, but the benefit obtained in the target metric space also depends on the correlation between the feature and the target metric.

The new feature that we propose, RDD, is consistently better than BBV for phase classification in 5 out of 8 programs on three different microarchitectures. Therefore, the best feature for finding phases is program dependent, but often holds true on different microarchitecture configurations. This helps the user to choose the best feature for more efficient microprocessor simulation. The user can select the best feature for a program using one microarchitecture, and apply the same feature on different microarchitectures.

References

[1] D. Burger, and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin-Madson, June 1997.

[2] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (October 2002), 45-57.

[3] R. Todi. SPEClite: Using Representative Samples to Reduce SPEC CPU2000 Workload. IEEE 4th Annual Workshop on Workload Characterization. 2001.

[4] T. Lafage and A. Seznec, "Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream", Kluwer Academic Publishers, pp. 145-163, September 2000.

[5] Jeremy Lau, Stefan Schoenmackers, and Brad Calder, Structures for Phase Classification, 2004 IEEE International Symposium on Performance Analysis of Systems and Software, March 2004.

[6] W. G. Cochran. Sampling Techniques, 3rd ed. John Wiley & Sons, 1977.

[7] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. Proceedings of the 20th VLDB Conference. Santiago, Chile. 1994.

[8] Raymond T. Ng, Jiawei Han. CLARANS: A Method for Clustering Objects for Spatial Data Mining. IEEE Transactions on Knowledge and Data Engineering archive Volume 14, Issue 5, 2002. pp. 1003 – 1016.

[9] J. W. Haskins, Jr. and K. Skadron. "Memory Reference Reuse Latency: Accelerated Warm-up for Sampled Microarchitecture Simulation." In Proceedings of the International Symposium on Performance Analysis of Systems and Software, Mar. 2003.

[10] Example Error Rates for SimPoint, http://www.cs.ucsd.edu/~calder/simpoint/error-rates.htm

[11] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (September 2003), 244-255.

[12] R. Wunderlich, T. Wenisch, B. Falsafi, and J.Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In 30th Annual International Symposium on Computer Architecture, June 2003.

[13] Y. Luo, A. Joshi, A. Phansalkar, L. John, and J. Ghosh. Analyzing and Improving Clustering Based Sampling for Microprocessor Simulation. Technical Report TR-050301-01, Laboratory for Computer Architecture, The University of Texas at Austin. March 2005. http://www.ece.utexas.edu/projects/ece/lca/ps/TR-050301-01.pdf

[14] V. Phalke and B. Gopinath. An Inter-Reference Gap Model for Temporal Locality in Program Behavior. 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems. 1995

[15] C. Ding and Y. Zhong. Predicting Whole-Program Locality through Reuse-Distance Analysis. ACM SIGPLAN Conference on Programming Language Design and Implementation. 2003.