A Multinomial Clustering Model for Fast Simulation of Computer Architecture Designs

Kaushal Sanghai, Ting Su, Jennifer Dy, David Kaeli
Northeastern University
ECE Department
Boston, MA 02115, USA
{ksanghai, tsu, jdy, kaeli}@ece.neu.edu

ABSTRACT

Computer architects utilize simulation tools to evaluate the merits of a new design feature. The time needed to adequately evaluate the tradeoffs associated with adding any new feature has become a critical issue. Recent work has found that by identifying *execution phases* present in common workloads used in simulation studies, we can apply clustering algorithms to significantly reduce the amount of time needed to complete the simulation. Our goal in this paper is to demonstrate the value of this approach when applied to the set of industry-standard benchmarks most commonly used in computer architecture studies. We also look to improve upon prior work by applying more appropriate clustering algorithms to identify phases, and to further reduce simulation time.

We find that the phase clustering in computer architecture simulation has many similarities to text clustering. In prior work on clustering techniques to reduce simulation time, K-means clustering was used to identify representative program phases. In this paper we apply a mixture of multinomials to the clustering problem and show its advantages over using K-means on simulation data. We have implemented these two clustering algorithms and evaluate how well they can characterize program behavior. By adopting a mixture of multinomials model, we find that we can maintain simulation result fidelity, while greatly reducing overall simulation time. We report results for a range of applications taken from the SPEC2000 benchmark suite.

Categories and Subject Descriptors

C.1 [**Processor Architectures**]: Single Data Stream Architectures; I.5.3 [**Pattern Recognition**]: Clustering

Keywords

Program Phase, simulation, clustering, EM, K-means, mixture of multinomials

1. INTRODUCTION

In the field of Computer Architecture, it is typical to evaluate new microprocessor designs using quantitative simulation studies. A software-based simulation model of the proposed architecture is developed, and then benchmark programs are executed on the model to evaluate the merits of new microprocessor *design features* (i.e., incremental changes to the base microprocessor design that are made to increase performance, increase reliability, or lower power). Examples of performance metrics commonly used to evaluate the merit of a new design feature include instructions executed per cycle (IPC), cache miss rates, and branch prediction accuracy.

As the complexity of a microprocessor architecture design grows, the time needed to evaluate a single design feature has become prohibitive [1]. A number of techniques have been proposed to reduce this simulation time [2, 3, 4, 5, 6]. One approach that has been shown to be particularly effective exploits the distinct *phases* that are present in program execution [3].

A phase in a program is defined as a time interval or execution slice where processor resources are utilized uniformly. Each program phase is comprised of on the order of 10-100 million instructions, as compared to the full program execution, which can include many trillions of instructions. During the simulation of a microprocessor design, if we could use a representative mix of these program phases to drive our simulation, we should be able to reduce the time to perform these needed evaluations, without sacrificing simulation accuracy.

Once we can identify the distinct phases of a program, clustering algorithms can be applied to identify similarities between phases. The fundamental observation is that the behavior of industry standard benchmark programs can be effectively characterized by a rather small number of representative phases [3, 7]. By running simulations on just those representative phases, and then weighting the individual results by the frequency that the phase occurs in the full application, we can obtain very similar simulation results as compared to running a simulation using the entire program. This approach can lead to significant reductions in overall simulation time.

Previous work on clustering phases [3] applies random projection, followed by K-means clustering, and uses the Euclidean distance as the measure for dissimilarity. To capture characteristic program behavior, phases are represented as a vector of frequency counts. The count represents the

number of times a particular basic block has been executed.[1] Basic block execution profiles possess similar characteristics as text data. Thus, we expect that models more appropriate for handling frequency vectors, such as a mixture of multinomials, should outperform K-means. In this paper, we propose to first use random projection, followed by clustering using a mixture of multinomials. Our experimental results confirm that a mixture of multinomials can reduce the error introduced by sampling by almost 50%.

This paper is organized as follows. In section 2, we define the terminology used in this problem, and describe our methodology. In section 3, we motivate our choice of using a mixture of multinomials for clustering. In this section, we also explain how we apply feature reduction techniques before performing clustering. In section 4, we describe our experimental setup and discuss results. Finally, we conclude the paper in section 6, and discuss directions for future work.

2. TERMINOLOGY AND METHODOLOGY

2.1 Basic block generation

A basic block is defined as a program section comprising of a set of instructions which have only one entry and one exit point. To be more precise, there are no branch instructions within a basic block. There have been different program characteristics proposed in the literature to identify the distinct phases of complex programs [8, 9]. Of all these techniques, basic blocks frequencies have been shown to be the most effective in capturing phase transitions. Basic block vectors (BBVs) (a BBV contains the execution frequency of each basic block during a program phase) exhibit a high degree of stability within a phase [8]. Therefore, in this study we use the number of times each basic block has been executed (i.e., BBV) as the feature to identify the distinct phases of a program.

2.2 Simulation Points

As defined earlier, a phase is an interval during program execution where the program behaves uniformly in terms of utilizing processor resources. The full execution of a program can be characterized by a set of distinct phases in which a particular phase can repeat several times during program execution. The representative phase representing each distinct phase is called a "simulation point" or "Simpoint [9]." By simulating a program using only the Simpoints instead of the entire program, we expect to obtain reasonable simulation accuracy, while significantly reducing simulation runtime.

Industry standard benchmarks are commonly used in the evaluation of microprocessor performance. Presently, the most commonly used benchmark suite is SPEC2000. The number of basic blocks present in individual benchmarks in this suite range from 1000 to 30, 000. The full execution of just a single benchmark program can surpass one trillion instructions.

3. SIMILARITIES BETWEEN PROGRAM PHASES AND TEXT DOCUMENTS

The phases found in benchmark programs look similar to the "bag-of-words" representation of text documents. Basic

blocks present in a phase are analogous to the words present in a paragraph. We can represent a program phase/document based on the frequency of basic blocks/words. To our best knowledge, in the computer architecture literature, only the K-means algorithm has been applied [3] to this problem. This is in contrast to the amount of previous work done on clustering text, where several data mining techniques have been examined [10, 11, 12]. It has been shown that when K-means is applied to cluster text data using Euclidean distance, the results are marginal [13]. The similarity between program phases and text inspires us to investigate clustering techniques that have been shown to be effective for text applications. In particular, we investigate the mixture of multinomials [14, 11].

Next, we describe mixture of multinomial models, along with the feature reduction algorithm that we apply.

3.1 Mixture models

Clustering, using finite mixture models, is a well-known generative method [15]. When applying this method, one assumes that data y is generated from a mixture of K component density functions, in which the component density function $p(y|\theta_j)$ represents cluster j for all $j's$, where θ_j is the parameter (to be estimated) for cluster j. The probability density of data y, is expressed by:

$$p(y) = \sum_{j=1}^{K} \alpha_j p(\mathbf{y}|\theta_j) \qquad (1)$$

where the $\alpha'_j s$ are the mixing proportions of the components (subject to: $\alpha_j \geq 0$ and $\sum_{j=1}^{K} \alpha_j = 1$). The log-likelihood of the N observed data points is then given by:

$$\mathcal{L} = \sum_{i=1}^{N} \log\{\sum_{j=1}^{K} \alpha_j p(\mathbf{y}_i|\theta_j)\} \qquad (2)$$

It is difficult to directly optimize (2), so we can use the Expectation-Maximization (EM) [16] algorithm to find a (local) maximum likelihood or maximum a posteriori (MAP) estimation of the parameters for the given data set. The EM algorithm iterates between an E-step and a M-step until convergence which are defined in section 3.2.

Selecting the number of clusters in a mixture model is a difficult task, and remains an open problem. A number of methods for selecting the number of clusters are discussed in [15]. We cannot simply apply the maximum likelihood criterion to determine the number of clusters, because this will lead to a clustering where each data point is a cluster. Some form of penalty for model complexity is needed. Here, we utilize the popular Bayesian information criterion (BIC) approach [17].

The optimal number of components K is selected by:

$$\hat{K} = argmax_K(2\mathcal{L} - m\log(N)) \qquad (3)$$

where m is the number of parameters for the model.

3.2 Multinomial Model

In this paper, we make a naive Bayes assumption which has gained popularity in text classification and clustering due to its simplicity and good performance [18, 19, 20]. The assumption is that each feature (i.e., the number of occurrence of basic blocks in our case) is independent of any other feature, given the class label. Given this assumption, a data

---

[1]A basic block is an instruction execution sequence with one entry point and one exit point.

point (i.e., a program phase in our case) is generated according to a multinomial probability distribution, given the class label, i.e., $p(y_i|\theta_j)$ is a multinomial distribution for all j's: $P(y_i|\theta_j) = \prod_l^D P_j(b_l)^{n_{il}}$, where $P_j(b_l)$ is the probability of the feature l occurring in cluster j, and hence is subject to $\sum_l^D P_j(b_l) = 1$, where n_{il} is the number of occurrences of feature l appearing in data y_i, and D is the number of features. To avoid zero values in $P_j(b_l)$, we utilize the MAP estimate with a Dirichlet prior. The use of this type of prior is also known as Laplace smoothing [20]. The updated equations for computing the MAP estimator are as follows:
E-step:
Ez_{ij}, the posterior probability of data y_i belonging to cluster j is given by:

$$Ez_{ij} = \frac{\alpha_j P(y_i|\theta_j)}{\sum_{j'}^K \alpha_{j'} P(y_i|\theta_{j'})} \quad (4)$$

M-step:

$$P_j(b_l) = \frac{1 + \sum_i^N Ez_{ij} n_{il}}{D + \sum_i^N \sum_{l'}^D Ez_{ij} n_{il'}} \quad (5)$$

where, $\alpha_j = \frac{1 + \sum_i^N Ez_{ij}}{N + K}$

When we compute the BIC for a mixture of multinomial models using 3, the number of parameters m is $K - 1 + K(D - 1)$.

3.3 Feature Reduction

A phase is typically comprised of $2,000$ to $30,000$ features (i.e., the number of basic blocks). Due to this high dimensionality (which may impact the performance of the clustering [21]), we apply a random projection method, since it has been shown that random projection can obtain reasonable performance [22], and is much faster than other feature reduction methods (e.g., faster than principal component analysis) [23, 24].

In random projection, the original D-dimensional data is projected to a q-dimensional $(q < D)$ subspace, using a random D by q matrix R. More specifically, let $X_{D \times N}$ stands for the original set of N D-dimensional observations. The projected data on the q dimensional subspace is given by $X\prime_{N \times q} = X_{N \times D} R_{D \times q}$. The rationale behind this computationally efficient method is the Johnson-Lindenstrauss lemma [25]. There are many choices for selecting the distribution of R_{ij} (the elements of the random projection matrix). We could choose to select a Gaussian or uniform distribution[26]. We choose to use the equation described below, that was proposed by [22], because it is simple and computationally efficient. Moreover, this equation has shown to perform well on text data [23].

$$R_{ij} = \sqrt{3} + 1 \quad \text{with probability } 1/6,$$
$$-1 \quad \text{with probability } 1/6,$$
$$0 \quad \text{with probability } 2/3$$

In our implementation, we omit the $\sqrt{3}$ multiplication, since it will not affect the separation between data points.

4. EXPERIMENTAL FRAMEWORK

To evaluate the merit of using different clustering algorithms, we compare the instructions per cycle (IPC) ob-

tained by running a full simulation with using the simulation points identified by the different clustering algorithms. The clustering approaches we compare are:

1. K-means algorithm combined with random projection of the dataset, and

2. EM applied to a mixture of multinomials, combined with random projection.

The random matrix is generated as described in section 3.3. Since the EM and K-means algorithms may result in local maxima, we apply ten random restarts to initialize all the clustering algorithms in our experiments for each K. To find K, we ran a mixture of multinomials for $K = 1$ to $kmax = 15$. We then picked the clustering result with the largest BIC score. The number of Simpoints obtained for each data is shown in figure 2 and will be discussed in section 5. The original dimensions were reduced by random projection. For datasets with an original dimension in the range of $1000 - 2000$, we reduced them to 15 dimensions. For datasets with dimensions greater than 20000, we reduced them to 100 dimensions. Several techniques have been proposed to determine the number of retained dimensions, with Principal Component Analysis (PCA) being one of them. But PCA is computationally expensive to compute for data sets possessing high dimensionality. Therefore, we utilized the heuristics described above to set the number of dimensions for random projection. In the following subsections, we describe how an actual simulation is conducted.

4.1 Program Simulation Methodology

The simulations are performed on a popular cycle accurate simulator, the Simplescalar toolset [27]. Simplescalar consists of various toolsets, of which sim-fast and sim-outorder are used for this work. A modified version of the sim-fast Simplescalar simulator available from [9] is used to generate basic block vectors.

The results are evaluated for a set of industry standard SPEC2000 benchmark programs. The programs in the SPEC 2000 suite are frequently used to drive performance evaluations of new and enhanced microprocessor architectures in industry and in research. The SPEC2000 programs consist of a set of floating-point and integer programs. Our experiments are conducted using both floating-point and integer benchmark programs. The integer benchmark programs exhibit less deterministic program behavior as compared to floating point benchmark programs. Therefore, it is more difficult to capture the behavior of integer programs (as we will discuss in section 5). The programs selected in our study exhibit the most non-deterministic behavior in the entire suite.

4.2 Generating the BBVs

The BBVs are generated with the help of the basic block vector generation tool provided by [9]. Every phase of 100 million instructions forms a vector, which contains the frequency of each basic block executed within that phase. The program is run to completion to generate a matrix, where each row represents a fixed interval of size 100 million instructions, and each column captures the frequency of each basic block executed during the phase. Thus the phases represent a point in the entire execution space, and each basic block adds a dimension to it. A different phase size could

also be used, though a phase size of 100 million instructions reasonably captures the phase transitions in a program. It would also be interesting to have a variable-length phase, but we have restricted our study to fixed-length phases. The dataset is generated for nine programs, with some using different input data sets, to form eleven benchmark programs.

4.3 Obtaining Simpoints

The dataset generated is the input to our clustering algorithm. After clustering the dataset, we obtain the mean of each cluster. We then select the program phase which has the shortest Euclidean distance to the mean of each cluster. The selected point becomes a *Simpoint*. Each of the Simpoints obtained is weighted by the priority of the corresponding class. This ensures that the results obtained for each Simpoint is weighted in proportion to its contribution to the overall program performance.

Having obtained the Simpoints and their corresponding weights, the Simpoints are tested by *fast-forwarding* (i.e., executing the program without performing any cycle-accurate simulation, as described in [3]) up to the simulation point, and then running a cycle accurate simulation for 100 million instructions. The sim-outorder tool provides a convenient way to test programs in the manner described above. Fast-forwarding a program implies only a functional simulation and avoids any time consuming detailed cycle accurate measurements. The IPC (instructions executed per cycle) is recorded for each Simpoint. The IPC metrics are weighted based on the cluster contributions of each individual Simpoint (i.e., cluster frequency).

5. RESULTS

Table 1 shows the multiple Simpoints (i.e., cluster representative) for the eleven benchmark programs used in this study. The numbers in the multiple Simpoints column show the start of the simulation point where a cycle accurate simulation should be performed for a length of 100 million instructions. To reach each of the Simpoints, the program needs to be fast forwarded to 100 million times the numeric value of the Simpoint and then start the accurate simulation run for 100 million instructions. It should be noted here that running the full simulation for the benchmark programs chosen takes about $4 - 5$ days, whereas with the methodology described here, it would take less than 5 hours for any program. As was discussed previously, the full execution of a benchmark program results in up to 1000 phases with each phase of size 100 million instructions. Now, if we simulate for just $2 - 5$ phases versus 1000 phases, then we can reduce simulation time to less than 0.5% of the original time. Therefore, simulation time is proportional to the number of Simpoints obtained for a particular program.

Figure 1 shows the results obtained in terms of the percent error in Instructions per Cycle (IPC) for the 11 programs, when considering the two clustering techniques. As we can see, our method performs better in 9 out of the 11 programs chosen. This is mainly due to the fact that we model the dataset more appropriately. By using a more appropriate clustering algorithm (random projection followed by mixture of multinomials), we reduce the percentage error to 1.65% as compared to 3.24% obtained by previous techniques (random projection, followed by K-means) for the programs that we tested. The average error reported in [3] is 3%, though this was for all the SPEC2000 bench-

mark programs, whereas we are comparing only a subset of the programs that are more interesting. Also, as is shown in Figure 2, we are able to reduce the number of Simpoints by as much as 45% as compared to the Simpoints obtained by [3]. This suggests that we will need fewer points to simulate, and as the simulation time becomes proportional to the number of Simpoints, it consequently reduces the simulation time further. We thereby show that by incorporating our clustering algorithm within the Simpoint tool [3], we can obtain better results. There are other programs in the SPEC2000 benchmark suite, but we focused our study to the 11 most challenging programs to characterize. Basically, the programs we have chosen have comparatively more phase transitions and therefore it is more difficult to capture the phase behavior of the program.

6. CONCLUSION

The cost to evaluate the merit of a new design feature continues to grow rapidly with each new generation of microprocessor. To reduce this cost, a sampled version of a benchmark program's execution can be used to drive the simulation study.

In prior work on this problem, K-means clustering was used to identify representative program phases. We observed that the workload characterization problem has many similarities to text clustering. In this paper, we have illustrated the benefits of utilizing an alternative clustering scheme (specifically, using a mixture of multinomials).

To summarize, this paper makes the following contributions:

1. We are able to accurately capture the distinct phases of the more complex SPEC2000 benchmark programs. The IPC error we obtained is less than 1.65%, as compared to 3.24% obtained by the techniques used in prior work.

2. Our algorithm results in a smaller number of clusters as compared to K-means clustering, and reduces the number of clusters by 45% on average over the 11 programs studied from the SPEC2000 benchmark suite.

3. Our toolset is the current state-of-the-art, as compared to the existing tools described in [3], for acquiring multiple Simpoints in the program. By simply incorporating EM into our new mixture of multinomials in the existing toolset [3], we can obtain higher fidelity simulation results and provide greater benefits to the computer architecture research community.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] S. Mukherjee, S. Adve, T. Austin, J. Emer, and P. Magnusson. Performance simulation tools. *IEEE Computer Magazine*, pages 38–49, February 2002.

[2] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: a preliminary application to the data

Benchmark program	length(in 100 million)	Multiple Simpoints obtained by mixture of multinomials			
art-110	417	331(100)	-	-	-
bzip2-source	1087	1075(33.2)	1198(19.1)	1163(1.5)	285(46.2)
crafty	1918	1857(14.5)	931(29.0)	121(24.3)	1120(32.2)
equake	1315	191(14.7)	772(85.3)	-	-
gcc-166	469	147(33.7)	319(66.3)	-	-
gcc-200	1086	489(2.8)	835(13.4)	243(76.7)	840(7.0)
gzip-log	395	95(17.2)	222(82.7)	-	-
gzip-random	876	376(52.3)	687(16.6)	234(31.0)	-
gzip-source	843	206(18.6)	185(13.7)	165(17.1)	584(50.7)
lucas	1423	593(4.4)	341(67.0)	228(28.5)	-
mcf	618	162(47.6)	409(52.3)	-	-

Table 1: This table lists the 11 programs from the SPEC2000 benchmark suite that were used in this study. The second column gives the total number of instructions (in 100 million) executed for the full simulation of the program. The Simpoints give the starting point of the Simpoints to simulate in the program. The program should be simulated for 100 million instructions by fast-forwarding the program until the start of each Simpoint. The value in parentheses indicates the weight (i.e., α_j) of the Simpoints in proportion to the overall program behavior. The weight is the mixing proportion of the components.
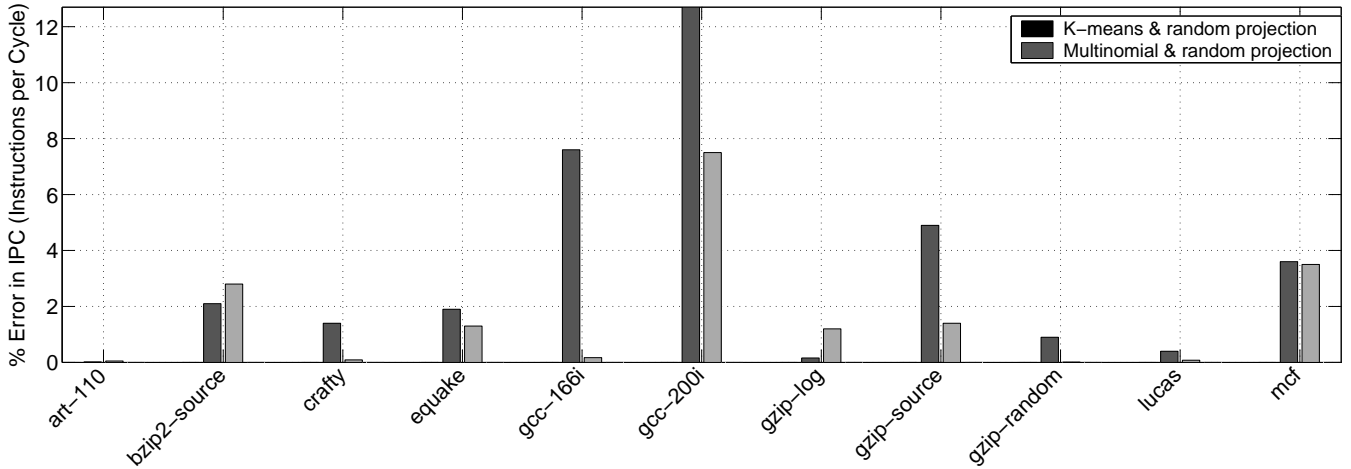

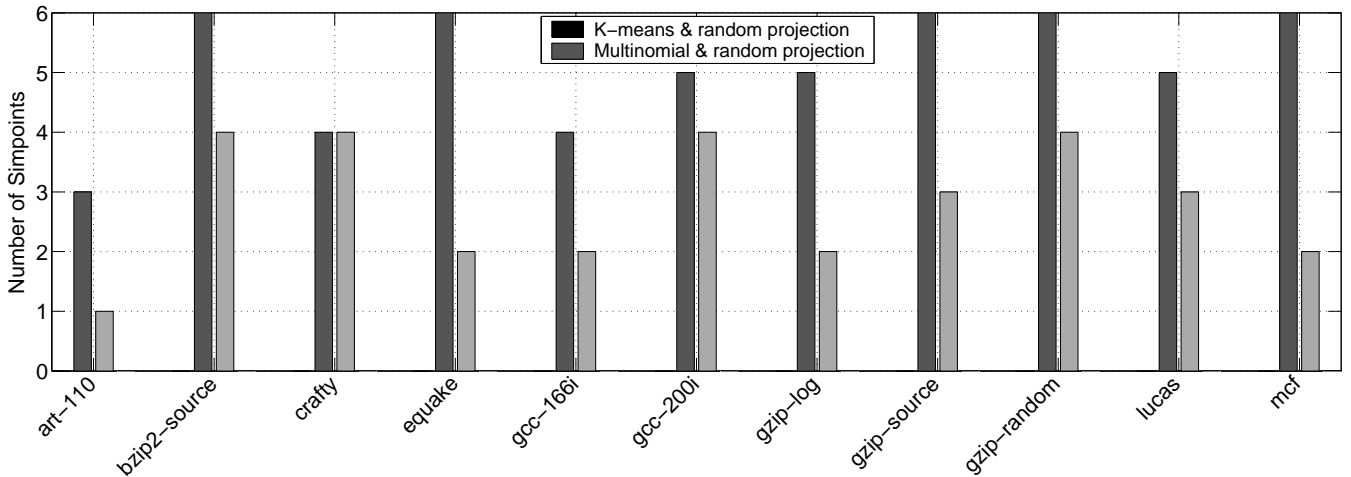Figure 1: % error in IPC values for the two clustering techniques.

Figure 2: The number of Simpoints obtained for the two clustering techniques.

stream. In *Proceedings of Workload characterization of emerging computer applications*, pages 145–163. Kluwer Academic Publishers, 2001.

[3] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 45–57. ACM Press, 2002.

[4] S. Girbal, G. Mouchard, A. Cohen, and O. Temam. Dist: a simple, reliable and scalable method to significantly reduce processor architecture simulation time. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 1–12. ACM Press, 2003.

[5] T. Conte, M. Hirsch, and K. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *ICCD '96: Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*, pages 468–477. IEEE Computer Society, 1996.

[6] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. How input data sets change program behaviour. In R. Clapp, K. Keeton, and A. Nanda, editors, *Proceedings of the Fifth Workshop on Computer Architecture Evaluation using Commercial Workloads, held in conjunction with the Eighth International Symposium on High Performance Computer Architecture (HPCA-8)*, pages 39–47, 2 2002.

[7] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 336–349. ACM Press, 2003.

[8] A. Dhodapkar and J. Smith. Comparing program phase detection techniques. In *MICRO 36: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, page 217. IEEE Computer Society, 2003.

[9] 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.

[10] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques, 2000.

[11] M. Meil and D. Heckerman. An experimental comparison of model-based clustering methods. *Mach. Learn.*, 42(1-2):9–29, 2001.

[12] S. Zhong and J. Ghosh. Generative model-based document clustering. Technical report, Dept. of Computer Science and Engineering. Florida Atlantic University, Boca Raton, FL, 2004.

[13] A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Proceedings of the 17th National Conference on Artificial Intelligence: Workshop of Artificial Intelligence for Web Search (AAAI 2000), 30-31 July 2000, Austin, Texas, USA*, pages 58–64. AAAI, July 2000.

[14] S. Vaithyanathan and B. Dom. Model-based hierarchical clustering. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 599–608, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[15] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, New York, 2000.

[16] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal statistical Society, Series B*, 39(1):1–38, 1977.

[17] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.

[18] D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15. Springer Verlag, Heidelberg, DE, 1998.

[19] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[20] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabelled document using em. *Machine Learning*, 39(2/3):103–134, 2000.

[21] J. Dy and C. Brodley. Feature selection for unsupervised learning. *Journal of Machine Learning Research*, 5:845–889, August 2004.

[22] D. Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281. ACM Press, 2001.

[23] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM Press, 2001.

[24] D. Fradkin and D. Madigan. Experiments with random projections for machine learning. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–522. ACM Press, 2003.

[25] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. In *In Conference in modern analysis and probability*, pages 189–206, 1984.

[26] S. Dasgupta. Experiments with random projection. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 143–151. Morgan Kaufmann Publishers Inc., 2000.

[27] C. Burger and T. Austin. The simplescalar tool set, version 2.0., June 1997.