

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 [9, 10]. 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 [9]. 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 “Sim-point [10].” By simulating a program using only the Sim-points 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

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

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 [11, 12, 13]. It has been shown that when K-means is applied to cluster text data using Euclidean distance, the results are marginal [14]. 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 [15, 12].

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 [16]. 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:

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

where the α_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:

$$\mathcal{L} = \sum_{i=1}^N \log \left\{ \sum_{j=1}^K \alpha_j p(y_i|\theta_j) \right\} \quad (2)$$

It is difficult to directly optimize (2), so we can use the Expectation-Maximization (EM) [17] 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 [16]. 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 [18].

The optimal number of components K is selected by:

$$\hat{K} = \operatorname{argmax}_K (2\mathcal{L} - m \log(N)) \quad (3)$$

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 [19, 20, 21]. 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

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 Simpoin

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 *Simpoin*. Each of the Simpoin obtained is weighted by the priority of the corresponding class. This ensures that the results obtained for each Simpoin is weighted in proportion to its contribution to the overall program performance.

Having obtained the Simpoin and their corresponding weights, the Simpoin 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 Simpoin. The IPC metrics are weighted based on the cluster contributions of each individual Simpoin (i.e., cluster frequency).

5. RESULTS

Table 1 shows the multiple Simpoin (i.e., cluster representative) for the eleven benchmark programs used in this study. The numbers in the multiple Simpoin 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 Simpoin, the program needs to be fast forwarded to 100 million times the numeric value of the Simpoin 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 Simpoin 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 Simpoin by as much as 45% as compared to the Simpoin obtained by [3]. This suggests that we will need fewer points to simulate, and as the simulation time becomes proportional to the number of Simpoin, it consequently reduces the simulation time further. We thereby show that by incorporating our clustering algorithm within the Simpoin 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 Simpoin 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

We wish to express our sincere thanks to Shi Zhong who provided us with a MATLAB version of EM on a mixture of multinomials model. This research was supported by NSF Grant No. IIS-0347532.

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. Sezec. Choosing representative slices of program execution for microarchitecture simulations: a preliminary application to the data

- 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] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 84–97, New York, NY, USA, 2003. ACM Press.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques, 2000.
- [12] M. Meil and D. Heckerman. An experimental comparison of model-based clustering methods. *Mach. Learn.*, 42(1-2):9–29, 2001.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, New York, 2000.
- [17] 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.
- [18] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.
- [19] 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.
- [20] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [21] 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.
- [22] J. Dy and C. Brodley. Feature selection for unsupervised learning. *Journal of Machine Learning Research*, 5:845–889, August 2004.
- [23] 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.
- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] C. Burger and T. Austin. The simplescalar tool set, version 2.0., June 1997.