

Selecting a Reduced but Representative Workload

Lieven Eeckhout Andy Georges Koen De Bosschere

Department of Electronics and Information Systems (ELIS), Ghent University
St.-Pietersnieuwstraat 41, B-9000 Gent, Belgium

{leeckhou,ageorges,kdb}@elis.UGent.be

ABSTRACT

Simulating a new microprocessor system is extremely time-consuming due to the large number of dynamic instruction counts that need to be simulated. In the computer architecture community, there are two methods available to address this issue, sampling and reduced inputs. This paper presents a methodology that is based on statistical data analysis techniques which allows us to determine whether trace sampling is to be preferred over reduced inputs. As such, a reduced but representative workload is composed. We believe this methodology is also useful in other areas such as middleware benchmarking.

1. INTRODUCTION

Designing a new microprocessor is extremely time consuming. One of the prominent reasons for this phenomenon is the huge number of instructions that need to be simulated per benchmark. These huge numbers of instructions are due to the fact that computer systems' performance continuously increases. As such, simulating one second of a real system leads to an ever increasing number of dynamic instructions. Given the large simulation slowdown, it is not exceptional that simulating one second of a real system takes 24 hours of simulation. Considering larger systems, such as multiprocessors, clusters of computers, etc., simulation time is even a bigger problem since all the individual components in the system need to be simulated simultaneously.

In this paper, we compare two techniques that are used in the computer architecture community for reducing the simulation time of uniprocessor systems. The first technique is *sampling*. Sampling refers to selecting samples from a complete program run. Since the number of sampled instructions is smaller than the total number of instructions in a complete program run, significant simulation speedups can be obtained. Important issues that need to be dealt with are: (i) which samples to select, and (ii) how to guarantee a correct hardware state at the beginning of each sample. Several authors have been doing research on sampling in the computer architecture community [1, 2, 6, 9, 10, 11, 12, 13, 14].

Next to sampling, there exists a second approach with the same goal, namely *reduced input sets*. KleinOowski *et al.* [8] propose to reduce the simulation time of the SPEC

CPU2000 benchmark suite¹ by using reduced input data sets, called MinneSPEC.² These reduced input sets are derived from the reference inputs by a number of techniques: modifying inputs (for example, reducing the number of iterations), truncating inputs, etc. The benefit of these reduced inputs is that the dynamic instruction count when simulating these inputs can be significantly smaller than the reference inputs.

Both methods, sampling and reduced input sets, thus yield significant simulation speedups. However, it is unclear which method is to be preferred in terms of accuracy and simulation speedup.

In this paper, we compare sampling versus reduced input sets for computer architecture evaluation. This is done using a methodology to reliably measure benchmark similarity [3, 4, 5]. This methodology is based on statistical data analysis techniques, namely principal components analysis and cluster analysis. The results that are obtained from this methodology can be used to compose a reduced and representative workload. Our results show that this methodology can indeed determine whether sampling is to be preferred over reduced input sets and vice versa.

We believe that the methodology evaluated here is also applicable for middleware benchmarking. In middleware benchmarking, shortening the evaluation time of a benchmark suite is an important issue, especially for middleware developers. Selecting a reduced but representative benchmark suite or workload for middleware benchmarking is thus an important issue that needs to be dealt with. The methodology presented in this paper can be a useful tool for middleware developers to evaluate the quality of their workload.

2. METHODOLOGY

Before going into detail on the results of the comparison 'sampling versus reduced inputs', we first explain our methodology which uses principal components analysis and cluster analysis.

2.1 Principal components analysis

Principal components analysis (PCA) [7] is a statistical data analysis technique that presents a different view on the measured data. It builds on the assumption that many variables (in our case, workload characteristics) are correlated and hence, they measure the same or similar properties of benchmarks. PCA takes as input the values of p variables for n cases. The variables in this study are workload

¹<http://www.spec.org>

²<http://www-mount.ee.umn.edu/~lilja/spec2000/>

characteristics such as instruction-level parallelism (ILP), cache behavior, branch behavior, etc. The cases are the various benchmarks that could be selected to be part of the workload. PCA computes new variables, called *principal components*, that are *linear combinations* of the original variables, such that all principal components are uncorrelated. In other words, PCA transforms the p variables X_1, X_2, \dots, X_p into p principal components Z_1, Z_2, \dots, Z_p with $Z_i = \sum_{j=1}^p a_{ij} X_j$. This transformation has the properties (i) $Var[Z_1] > Var[Z_2] > \dots > Var[Z_p]$ which means that Z_1 contains the most information and Z_p the least; and (ii) $Cov[Z_i, Z_j] = 0, \forall i \neq j$ which means that there is no information overlap between the principal components. Note that the total variance in the data remains the same before and after the transformation, namely $\sum_{i=1}^p Var[X_i] = \sum_{i=1}^p Var[Z_i]$.

As stated in the first property in the previous paragraph, some of the principal components will have a high variance while others will have a small variance. By removing the components with the lowest variance from the analysis, we can reduce the number of workload characteristics while controlling the amount of information that is thrown away. We retain q principal components which is a significant information reduction since $q \ll p$ in most cases, typically $q = 2$ to $q = 4$. To measure the fraction of information retained in this q -dimensional space, we use the amount of variance $(\sum_{i=1}^q Var[Z_i]) / (\sum_{i=1}^p Var[X_i])$ accounted for by these q principal components.

By examining the most important q principal components, which are linear combinations of the original workload characteristics, meaningful interpretations can be given to these principal components in terms of the original workload characteristics. A coefficient a_{ij} that is close to +1 or -1 implies a strong impact of the original characteristic X_j on the principal component Z_i . A coefficient A_{ij} that is close to zero on the other hand, implies no impact.

The next step in the analysis is to display the various benchmarks as points in the q -dimensional space built up by the q principal components. This can be done by computing the values of the q retained principal components for each benchmark. This representation gives us an excellent opportunity to visualize the workload design space understandably. Note that this q -dimensional space will be much easier to understand than the original p -dimensional space for two reasons: (i) q is much smaller than p and (ii) the q -dimensional space is uncorrelated.

During principal components analysis, one can either work with normalized or non-normalized data (the data is normalized when the mean of each variable is zero and its variance is one). In case of non-normalized data, a higher weight is given in the analysis to variables with a higher variance. In our experiments, we have used normalized data because of our heterogeneous data; e.g., the variance of the instruction-level parallelism (ILP) is orders of magnitude larger than the variance of the data cache miss rates.

2.2 Cluster analysis

Cluster analysis (CA) is a data analysis technique that is aimed at clustering n cases based on the measurements of q variables, in our case the principal components from PCA. The final goal is to obtain a number of clusters, containing benchmarks that have ‘similar’ behavior. A commonly used type of clustering is linkage clustering.

Linkage clustering starts with a matrix of distances between the n cases or benchmarks. As a starting point for the algorithm, each benchmark is considered as a cluster. In each iteration of the algorithm, the two clusters that are most close to each other (with the smallest distance, also called the *linkage distance*) will be combined to form a new cluster. As such, close clusters are gradually merged until finally all cases will be in a single cluster. This can be represented in a so called *dendrogram*, which graphically represents the linkage distance at each iteration of the algorithm. Having obtained a dendrogram, it is up to the user to decide how many clusters to take. This decision can be made based on the linkage distance. Indeed, small linkage distances imply strong clustering and thus high similarity, while large linkage distances imply weak clustering or dissimilar behavior.

Cluster analysis is heavily dependent on an appropriate distance measure. In our analysis, the distance between two benchmarks is computed as the Euclidean distance in the transformed q -dimensional space obtained after PCA for the following reason. The values along the axes in this space are uncorrelated. The absence of correlation is important when calculating the Euclidean distance because two correlated variables—that essentially measure the same thing—would contribute a similar amount to the overall distance as an independent variable; as such, these variables would be counted twice, which is undesirable.

Next to defining the distance between two benchmarks, we also need to define the distance between two clusters. One way to compute the distance between two clusters is the *weighted pair-group average* strategy in which the distance between two clusters is computed as the weighted average distance between all pairs of benchmarks in the two different clusters. The weighting of the average is done by considering the cluster size, i.e., the number of benchmarks in the cluster.

3. COMPARING SAMPLING VERSUS REDUCED INPUTS

The introduction discussed two techniques to reduce the length of a benchmark simulation run, namely sampling and reduced input sets. However, it is unclear which method is to be preferred. In other words, which method yields a reduced benchmark that most closely resembles the reference benchmark while yielding a significant simulation speedup.

To compare sampling versus reduced input sets, we can make use of the methodology presented in the previous section:

- First, we measure a number of important program characteristics for a set of benchmarks. In this study we used a collection of 20 program characteristics concerning the branch behavior, instruction mix, instruction-level parallelism (ILP), cache behavior, etc.
- Second, we apply PCA to this dataset. The result is a transformed uncorrelated space in which the various benchmarks can be displayed.
- Third, cluster analysis is applied to graphically represent the similarity between the various benchmarks.

We used a number of long running SPEC CPU95 benchmarks (*m88ksim*, *vortex*, *go* and *compress*) as well as a database

Table 1: Comparing sampling versus reduced inputs: the method of choice (shown in bold) depends on the benchmark. The two rightmost columns represent the dynamic instruction count (in millions) and the obtained simulation speedup, respectively.

benchmark	input/sampling regime	dyn (M)	speedup
postgres	TPC-D query 16	82,228	
	10%-sampled trace	8,222	10
	1%-sampled trace	822	100
	0.1%-sampled trace	82	1,000
m88ksim	ref	71,161	
	train	24,959	2.9
	10%-sampled trace	7,116	10
	1%-sampled trace	711	100
vortex	ref	92,555	
	train	3,244	28.5
	10%-sampled trace	9,255	10
	1%-sampled trace	925	100
go	0.1%-sampled trace	92	1,000
	ref1 (50 21 9stone21.in)	35,758	
	10%-sampled ref1 trace	3,575	10
	1%-sampled ref1 trace	357	100
	0.1%-sampled ref1 trace	35	1,000
	ref2 (50 21 5srone21.in)	35,329	
	10%-sampled ref2 trace	3,532	10
	1%-sampled ref2 trace	353	100
	0.1%-sampled ref2 trace	35	1,000
	train	593	60.3
compress	ref (14,000,000 e 2231)	60,102	
	10%-sampled trace	6,010	10
	1%-sampled trace	601	100
	0.1%-sampled trace	60	1,000
	10,000,000 e 2231	42,936	1.4
	5,000,000 e 2231	21,495	2.8
	1,000,000 e 2231	4,342	13.8
	500,000 e 2231	2,182	27.5
	100,000 e 2231	423	142

postgres running a decision support TPC-D query, namely query 16. For each of these benchmarks we have considered three sampled traces: one with a sample rate of 10%, a second one with a sample rate of 1% and a third one with a sample rate of 0.1%. These sampled traces were obtained using the reference inputs. We assumed a periodic sampling regime and samples of 1 million instructions each. Next to these sampled traces, we also considered a number of reduced input sets. For m88ksim, vortex and go, we considered the train input. For compress, we generated a number of reduced inputs by modifying the reference input.

The methodology presented in the previous section is used to determine benchmark similarity. If a sampled trace is closer to the reference input than reduced inputs we conclude that sampling yields more representative benchmark behavior than reduced inputs. In this case, the sampled trace will be included in the workload. In the opposite case where a reduced input is more representative, the reverse will be true.

The results from these analyses can be represented in a dendrogram which represents the (dis)similarity between points in the workload design space. From Figure 1, we can make the following conclusions for the various benchmarks:

- For some benchmarks, sampling results in benchmark behavior that is more similar to the reference input than reduced inputs. Indeed, for m88ksim and vortex the linkage distances between the points corresponding to the sampled traces and the reference input are much

shorter than between the reduced inputs and the reference input. As such, for these benchmarks, sampling is definitely a better option than reduced inputs.

- For other benchmarks on the other hand, for example go and compress, reduced inputs result in program behavior that is more similar to the reference input than what can be attained through sampling. For compress, we used a variety of reduced inputs. Note however, that not all reduced inputs result in benchmark behavior that is similar to the reference input. The smallest reduced input 100,000 e 2231 (which is derived from the reference input 14,000,000 e 2231) results in very dissimilar behavior. The other reduced input sets, varying from 500,000 e 2231 to 10,000,000 e 2231, yield similar behavior. As such, we conclude that the reduced input should not be smaller than 500,000 e 2231.

Table 1 summarizes the conclusions from this analysis. The method of choice, sampling versus reduced input sets, is shown in bold for each benchmark. In addition, the simulation time speedup that is obtained for each method is displayed as well.

As a general conclusion from this section, we can state that for some benchmarks sampling is more accurate than reduced input sets. For other benchmarks, the opposite is true. As such, we recommend researchers working with either sampled traces or reduced input sets to verify whether they are using an approach that yields representative behavior. The methodology presented here is an excellent tool for this purpose.

4. CONCLUSION

This paper discussed a methodology that allows researchers to select a reduced and representative workload in a reliable way. As a case study we considered sampling versus reduced input sets for computer architecture evaluation. However, we believe that this methodology can be used more widely in other areas such as middleware benchmarking.

Acknowledgements

Lieven Eeckhout is a Postdoctoral Fellow of the Fund for Scientific Research – Flanders (Belgium) (F.W.O. Vlaanderen). Andy Georges is supported by the SEESCOA project sponsored by the Flemish Institute for the Promotion of the Scientific-Technological Research in the Industry (IWT – Vlaanderen).

5. REFERENCES

- [1] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *Proceedings of the 1996 International Conference on Computer Design (ICCD-96)*, pages 468–477, October 1996.
- [2] P. K. Dubey and R. Nair. Profile-driven sampled trace generation. Technical Report RC 20041, IBM Research Division, T. J. Watson Research Center, April 1995.
- [3] L. Eeckhout, A. Georges, and K. De Bosschere. How Java programs interact with virtual machines at the microarchitectural level. In *Proceedings of the 18th Annual ACM SIGPLAN Conference on*

