Accurate Statistical Approaches for Generating Representative Workload Compositions

Lieven Eeckhout*, Rashmi Sundareswara†, Joshua J. Yi‡, David J. Lilja§ and Paul Schrater†
* ELIS Department, Ghent University, Belgium
† CS Department, University of Minnesota, Twin Cities, MN
‡ Freescale Semiconductor, Austin, TX
§ ECE Department, University of Minnesota, Twin Cities, MN

***Abstract*— Composing a representative workload is a crucial step during the design process of a microprocessor. The workload should be composed in such a way that it is representative for the target domain of application and yet, the amount of redundancy in the workload should be minimized as much as possible in order not to overly increase the total simulation time. As a result, there is an important trade-off that needs to be made between workload representativeness and simulation accuracy versus simulation speed.**

Previous work used statistical data analysis techniques to identify representative benchmarks and corresponding inputs, also called a subset, from a large set of potential benchmarks and inputs. These methodologies measure a number of program characteristics on which Principal Components Analysis (PCA) is applied before identifying distinct program behaviors among the benchmarks using cluster analysis. In this paper we propose Independent Components Analysis (ICA) as a better alternative to PCA as it does not assume that the original data set has a Gaussian distribution, which allows ICA to better find the important axes in the workload space. Our experimental results using SPEC CPU2000 benchmarks show that ICA significantly outperforms PCA in that ICA achieves smaller benchmark subsets that are more accurate than those found by PCA.

I. Introduction

Designing a new microprocessor is both a complex and time-consuming task. One of the tradeoffs that needs to be made is what benchmarks (and appropriate inputs) need to be chosen during the design process. The selection of benchmarks for inclusion in a benchmark suite is called workload composition or benchmark subsetting. A well composed workload should address two major concerns. First, the workload should be representative for a larger set of applications, *i.e.*, we want all typical program behaviors to be represented in the workload, no major program behavior should be omitted. Second, the number of benchmarks in the workload should be limited because of simulation time concerns. As such, a tradeoff needs to be made between benchmark suite representativeness (and thus simulation accuracy) and simulation time. Previous research has shown that redundancy exists across benchmarks and inputs so that the number of benchmarks and inputs can be reduced without compromising accuracy too much [1], [2], [3], [4], [5].

Prior work on workload composition and benchmark suite subsetting first measured a number of program characteristics for a large number of benchmarks and inputs. These program characteristics can be microarchitecture-dependent or microarchitecture-independent or a mix of both. In a second step, these per-benchmark/input program characteristics are analyzed through statistical data analysis. Previous work applied Principal Components Analysis (PCA) to reduce the dimensionality of the data set. Once the data set is analyzed through PCA, cluster analysis is applied to group similarly behaving benchmarks/inputs into so called clusters of distinct program behaviors. Then, a representative benchmark/input can be chosen from each cluster for inclusion in the reduced benchmark suite.

In this paper, we propose Independent Components Analysis (ICA) as a better alternative to PCA. The key difference between PCA and ICA is that PCA strives to find uncorrelated axes whereas ICA attempts to find independent axes in a multi-dimensional space—note that independent axes are also uncorrelated, *i.e.*, independence is stronger than uncorrelateness. The main reason for ICA to be better than PCA is that ICA does not assume that the original data set has a Gaussian distribution, as PCA does. When comparing behavioral characteristics between benchmarks/inputs, it could be the case however that the data set is clustered in the first place and thus is not normally distributed. As a result, ICA is better capable to find the 'important' axes in the multi-dimensional space. Through experimental evaluation using SPEC CPU2000 benchmarks we conclude that ICA outperforms PCA for benchmark subsetting in that ICA produces both (i) fewer benchmark/input pairs and (ii) more accurate performance predictions.

Many applications can benefit from this work. For instance, benchmark standardization institutions such as SPEC, TPC, EEMBC, and so forth can use this methodology for composing their benchmark suites. Obviously, when composing benchmark suites it is of primary importance that the benchmark suites being composed are representative of the real-life application domains they target. The methodology presented in this paper can be used to determine the important benchmarks and inputs for inclusion in the benchmark suite. Researchers and computer designers can also benefit from this work. The methodology presented here can be used to identify a limited set of important benchmarks and inputs from a larger set of potentially interesting benchmarks in order to reduce the total simulation time required to search a large design space.

This paper is organized as follows. We first discuss prior

work in workload composition and benchmark suite subsetting. We subsequently discuss the workload characteristics that we use as input for our methodology in Section III. We then present the two statistical data analysis techniques that we consider in this paper for workload composition: principal components analysis (PCA), as it was used in previous work, and independent components analysis (ICA), which we propose in this paper as a better alternative than PCA. After detailing our experimental setup in Section V, we evaluate the feasibility of ICA for workload composition in Section VI. Finally, we conlude in Section VII.

II. Prior work

Several different approaches have been used to measure benchmark similarity in previous studies. Saavedra and Smith [6] presented a metric that is based on dynamic program characteristics for the Fortran language. Their metric includes the instruction mix, the number of function calls, the number of address computations, etc. For measuring the difference between benchmarks they used the squared Euclidean distance. An important shortcoming of this methodology is the use of the Euclidean distance in the original workload space. Correlation and dependence between variables make the Euclidean distance an unreliable metric for quantifying benchmark similarity.

To address this shortcoming, Eeckhout *et al.* [1] proposed the use of principal components analysis (PCA) to get rid of the correlation in the data set. The methodology presented in [1] forms the basis for the work presented in this paper. A number of program characteristics are measured for a number of benchmarks/inputs on which PCA is applied. Then, cluster analysis is applied on this transformed data set in order to find the distinct program behaviors among the benchmarks.

The workload characteristics used in [1] as input to the workload analysis methodology was a mix of microarchitecture-independent and microarchitecture-dependent characteristics. The microarchitecture-independent metrics are instruction mix, inherent instruction-level parallelism, etc., while the microarchitecture-dependent metrics are cache miss rates, branch mispredict rates, etc. One potential pitfall in using microarchitecture-dependent metrics in a workload analysis methodology is that the results of the analysis might be biased by the microarchitecture-dependent metrics, *i.e.*, it is unclear how the reduced workload will perform on other microarchitectures. To address this issue, Phansalkar *et al.* [2] feed the workload analysis methodology with microarchitecture-independent metrics only. Note that these microarchitecture-independent metrics are still ISA and compiler dependent. On the opposite side of the spectrum there is some work done on benchmark subsetting using microarchitecture-dependent metrics only, see for example [3], [4].

Most of this previous work used Principal Components Analysis (PCA) prior to applying cluster analysis. In this paper we show that Independent Components Analysis (ICA) outperforms PCA. The work presented in this paper is orthogonal to this previous work. The methodology we propose here based on ICA can be applied to any data set, irrespective of whether the data set is microarchitecture-independent or not.

Yi *et al.* [5] take a different approach and propose a technique for grouping benchmarks based on how they stress the components of a processor. Their method is based on a Plackett-Burman design of experiments. A Plackett-Burman design is a technique that allows researchers to measure the impact of variables by making a limited number of measurements. For example, consider the case where we want to measure the impact of n variables where each variable can have b unique values. The total number of experiments (or in our case simulations) that need to be performed for this full factorial design is $O(b^n)$. The multifactorial Plackett-Burman design, on the other hand, is a fractional design that requires only $O(n)$ experiments to determine the impact of n variables and a subset of their interactions. This is done by varying all parameters in carefully chosen combinations to stress the critical corners of the overall design space.

Citron [7] performed a survey on the SPEC CPU benchmarks used in the ISCA, Micro and HPCA computer architecture conferences. He observed that some benchmarks are more popular than others, *i.e.*, researchers tend to use some benchmarks more frequently than others. The question however is whether these subsets are representative for the whole SPEC CPU2000 benchmark suite. The methodology proposed in this paper could be used to identify a representative subset.

Reducing the simulation time has received a lot of attention in the recent literature. Various research groups have proposed various approaches for reducing the simulation time. Examples are reduced input sets [8], [9], sampling [10], [11], [12], [13], and statistical simulation [14], [15], [16]. These approaches however, are orthogonal to benchmark suite subsetting as discussed in this paper. When a reduced workload is composed, any of these simulation speedup approaches can be used to reduce the overall simulation time even further.

III. Program characteristics

The workload space could be viewed as a p-dimensional space in which the dimensions are determined by a set of workload characteristics. The individual benchmarks can then be displayed by a p-dimensional vector within this space.

An important issue is the choice of the dimensions in this workload space. Intuitively, the program characteristics that should be used are those that affect performance. As mentioned before, one could either choose a set of microarchitecture-dependent characteristics, such as cache miss rates, branch mispredict rates, IPC numbers, etc., or one could choose a mix of microarchitecture-dependent and microarchitecture-independent characteristics, or finally, one could choose a set of microarchitecture-independent metrics only. The program characteristics that are selected in this paper to build our data set are a mix of microarchitecture-dependent and microarchitecture-independent characteristics. The program characteristics that we consider in this paper are shown in Table I; in fact, these are the same data set of

No.	Category	Program characteristic
1	Instruction mix	Percentage integer arithmetic operations
2		Percentage logical operations
3		Percentage shift and byte manipulation operations
4		Percentage load/store operations
5		Percentage control operations
6	Branch predictability	Branch prediction accuracy for a hybrid branch predictor selecting among an 8K-entry bimodal predictor and an 8K-entry gshare predictor (history of 12 branches); meta predictor contains 8K entries
7	Control flow	Number of instructions between two sequential flow breaks, or the number of instructions between two taken branches
8	Data stream behavior	Miss rate for an L1 8KB direct-mapped D-cache
9		Miss rate for an L1 16KB direct-mapped D-cache
10		Miss rate for an L1 32KB 2-way set-associative D-cache
11		Miss rate for an L1 64KB 2-way set-associative D-cache
12		Miss rate for an L1 128KB 4-way set-associative D-cache
13	Instruction stream behavior	Miss rate for an L1 8KB direct-mapped I-cache
14		Miss rate for an L1 16KB direct-mapped I-cache
15		Miss rate for an L1 32KB 2-way set-associative I-cache
16		Miss rate for an L1 64KB 2-way set-associative I-cache
17		Miss rate for an L1 128KB 4-way set-associative I-cache
18	Instruction-level parallelism (ILP)	ILP on an infinite-resource processors, *i.e.*, assuming an infinite number of functional units, infinite decode/issue/reorder width, infinite window size, perfect caches, perfect branch prediction, unit execution latency. In other words, only read-after-write dependencies are considered through registers as well as through memory.

TABLE I

PROGRAM CHARACTERISTICS USED FOR CHARACTERIZING THE BENCHMARK/INPUT PAIRS.

characteristics as used in [1]. This set consists of 18 metrics measuring microarchitecture-independent characteristics such as instruction mix, number of sequential fbw breaks and inherent instruction-level parallelism (ILP), and microarchitecture-dependent metrics such as instruction and data cache miss rates and branch mispredict rates. These metrics were chosen in such a way that they cover a large spectrum of program characteristics. Note that a good choice of the important metrics for quantifying benchmark behavior is extremely important for a workload composition methodology. For example, using the characteristics from Table I for selecting benchmarks and inputs for doing research on megabyte L2 caches might be inappropriate because the caches being considered here were small L1 caches. Note however that the appropriate selection of program characteristics is outside the scope for this paper—the focus in this paper is on the statistical analysis.

These program characteristics are measured for all benchmark/input pairs considered in this study. There are 63 benchmark/input pairs and they are given in Section V. As such we obtain a 63×18 matrix in which the rows are the benchmark/input pairs and in which the columns are the 18 program characteristics. Prior to doing any statistical analysis we first normalize this data set, *i.e.*, each variable (program characteristic) in the data set is transformed so that it has a mean of zero and a variance of one over all benchmark/input pairs. The reason for doing the normalization prior to statistical analysis is the heterogeneity of the data set. Some program characteristics, such as ILP, vary in the range of tens, whereas other program characteristics vary in the range of fractions smaller than 1, such as the cache miss rates. Normalization puts all the program characteristics on a common scale. In the above example, measuring benchmark similarity using unnormalized data would give a higher weight to the ILP metric than to the cache miss rate metrics. This 63×18 normalized data matrix is used as input for both PCA and ICA.

IV. STATISTICAL DATA ANALYSIS

This section discusses both principal components analysis (PCA) and independent components analysis (ICA). There are basically two important reasons why we want to apply these statistical data reduction techniques for workload composition. First, a statistical data analysis technique attempts to find the key 'dimensions' in a data set. Both PCA and ICA try to find important axes in the original workload space so that important underlying (latent) program metrics become apparent in the axes obtained from the analysis. The second reason for applying a statistical data analysis technique is to reduce the dimensionality of the data set. Reducing the dimensionality of a data set increases its understandability.

A. Principal Components Analysis (PCA)

Principal components analysis [17] builds on the assumption that many variables (in our case, program characteristics) are correlated and hence measure the same or similar properties of the various inputs (which, in our case, are the benchmarks/input pairs). PCA computes new variables called principal components that are linear combinations of the original variables such that all principal components are uncorrelated. PCA transforms the p variables $X_1, X_2, \ldots, X_p$ into p principal components $Z_1, Z_2, \ldots, Z_p$ with $Z_i = \sum_{j=1}^{p} a_{ij} \cdot X_j$. This transformation has the following important properties: (i) $Var[Z_1] \geq Var[Z_2] \geq \ldots \geq Var[Z_p]$ 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 variation in the data remains the same before and after the transformation, namely: $\sum_{i=1}^{p} Var[X_i] = \sum_{i=1}^{p} Var[Z_i]$. Mathematically speaking, PCA actually solves the eigenvalue problem over the correlation matrix.

As stated in the first property in the previous paragraph, some of the principal components have a high variation while others have a small variation. By removing the components with the lowest variation from the analysis, we can reduce the number of program 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. To measure the fraction of information retained in this q-dimensional space, we use the amount of variation $\sum_{i=1}^{q} Var[Z_i] / \sum_{i=1}^{p} Var[X_i]$ accounted for by these q principal components. Typically 85% to 90% of the total variation should be explained by the retained principal components.

Recall that the p original variables are the program characteristics that build up the original workload space. By examining the most important q principal components, which are linear combinations of the original program characteristics, meaningful interpretations can be given to these principal components in terms of the original program 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 0 on the other hand, implies no impact. We use STATISTICA [18] for applying PCA.

An important application for a data reduction technique such as PCA is the visualization of the workload space. Indeed, the benchmark/input pairs can be displayed 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. As such, a view can be given on the workload space and benchmark similarity can be studied. The projection in the q-dimensional space is much easier to understand than a view on the original p-dimensional space for two reasons: (i) q is much smaller than p: $q \ll p$, and (ii) the q-dimensional space is uncorrelated.

PCA makes the following assumptions: (i) linear combinations of the data dimensions are valid, (ii) the statistical structure of the data can be captured by the mean and covariance, *i.e.*, the components have a Gaussian distribution. This implies that all statistical dependence between components can be captured via correlation. When these assumptions hold, PCA is a good tool to use. But in many cases, not all assumptions are valid. In particular, the notion that all statistical dependence is captured by correlation is frequently too restrictive. Nevertheless, previous work using PCA has shown that PCA is fairly accurate (even if the data is not Gaussian distributed), however, the contribution of this paper is to show that ICA is even more accurate. Note also that ICA is normally used for source separation in signals—an example of which is the cocktail party problem, *i.e.*, that is given a recording of many signals, how can you separate out the various sources? The problem being addressed here is not exactly source separation. As a result, ICA did not seem to be a solution for identifying benchmark similarity in previous work. However, on seeing the distribution of the underlying data, ICA becomes apparent as a viable solution.

B. Independent Components Analysis (ICA)

Independent Components analysis (ICA) is a data analysis technique that relaxes the assumption that the underlying data be Gaussian distributed. Instead, ICA tries to produce components that are statistically independent, which is a stronger requirement than "uncorrelatedness" as discussed before. It is similar to PCA in that it also produces a linear representation (although non-linear forms exist—we restrict the discussion and implementation to the linear form though).

a) Statistical Independence and Uncorrelatedness: Two variables y_1 and y_2 are defined as being "statistically independent" if the joint probability density $p(y_1, y_2)$ can be expressed as the following:

$$p(y_1, y_2) = p(y_1)p(y_2). \tag{1}$$

If two variables are independent, they are said to be uncorrelated. However, the inverse is not true. Two variables that are uncorrelated can still be dependent on each other. For example, two random variables y_1 and y_2 are said to be uncorrelated if their covariance is zero, *i.e.* [19]:

$$E\{y_1, y_2\} = E\{y_1\}E\{y_2\}. \tag{2}$$

ICA estimation procedures consist of estimating a mixing matrix A, and its inverse W, such that the measured data consisting of column vectors x_i are related in the following way to the independent components, s_i:

$$x = As \tag{3}$$

and

$$s = Wx \tag{4}$$

Next, we discuss one method of estimating the matrix A (and consequently its inverse W). At this point, we have a representation that is similar to PCA, *i.e* the matrix A and its inverse W are linear representations of the data x. The only difference being that the space that the column vectors (axes) of A (or W) describe need not necessarily be an orthogonal space. In PCA, it is always the case that the column vectors of A be orthogonal. Vectors s are the independent components that are mutually independent of each other and are non-gaussian distributed.

b) Negentropy: Since ICA is interested in extracting the dimensions that highlight the non-gaussianity of the data, we need a measure of the non-gaussianity. An important measure is negentropy. To explain negentropy, we need to revisit the idea of entropy. The measure of entropy is an important concept in information theory. The more "random" a variable is, the larger its entropy. For a random variable x, its entropy, $H(x)$ can be defined by

$$H(x) = -\int f(x) log(f(x)) dx \tag{5}$$

If x is a random variable whose distribution is Gaussian, then it is proven that x's entropy is high. To define a distribution of a random variable as non-Gaussian, we need to define

its distribution with reference to a Gaussian distribution. Therefore, negentropy (J) is defined as

$$J(x) = H(x_{gauss}) - H(x) \tag{6}$$

where x_{gauss} is a Gaussian distribution with the same covariance matrix as x. This approximation is hard to obtain in practice because one would need an estimate of the probability density function (PDF) of x which might not always be easy to obtain analytically for arbitrary distributions. Therefore, negentropy is practically estimated in the following way [19], [20]:

$$J(x) \approx \sum_{i=1}^{p} k_i[E\{G(x_i)\} - E\{G(v)\}]^2 \tag{7}$$

where k_i are some positive constants, v is a gaussian random variable with mean 0 and variance 1 and G_i are some non-quadratic functions of which some examples are:

$$G_1(u) = \frac{1}{a_1} logcosha_1 u, \qquad G_2(u) = -exp(\frac{-u^2}{2}) \tag{8}$$

where $1 \leq a_1 \leq 2$.

The function G should be chosen such that it does not grow too fast—this allows for a more robust estimator. In our analysis, we used the publicly available software, FastICA [20] to implement ICA. FastICA is based on a fixed point algorithm for finding a maximum of non-Gaussianity of $w^T x$ as estimated by Eqn. 7. The basic form of the algorithm chooses initial w vectors and iterates until Eqn. 7 (the derivation of Eqn. 8 is actually used) has converged for all the w vectors.

For an in-depth discussion of ICA and FastICA, please refer to [19], [21].

C. Cluster Analysis

As mentioned previously, the next step in our methodology is to apply cluster analysis [17] on the transformed data set—note that we can apply cluster analysis in both the PCA and ICA space. Cluster analysis is a data analysis technique that is aimed at clustering n cases, in our case benchmark/input pairs, based on the measurements of q variables, in our case the dimensions obtained from PCA or ICA. The final goal is to obtain a number of groups containing various benchmarks that exhibit 'similar' behavior.

One particular approach to cluster analysis is K-means clustering. K-means clustering produces exactly K clusters with the greatest possible distinction. The K-means clustering algorithm works as follows. In each iteration, the distance is calculated for each case to the center of each cluster. A case then gets assigned to the closest cluster. As such, new cluster centers can be computed. This algorithm is iterated until no more changes are observed. It is well known that the result of K-means clustering can be dependent on the choice of the initial cluster centers. Therefore we consider multiple randomly chosen initial cluster centers and then consider the clustering with the optimal BIC criterion. The Bayesian Information Criterion (BIC) is a score that indicates how well the data fits the model (number of clusters in this case). The algorithm uses a Bayesian approach and estimates a posterior probability for each cluster conditioned on the data. The model with the highest probability is returned. For applying the K-means clustering we use the SimPoint v2.0 software[1] [11], [12].

progr	input	insns
vpr	ref	94,331
	smred	6
	mdred	92
	lgred	857
	train	10,457
vortex	ref1	118,977
	ref2	128,678
	ref3	133,044
	smred	88
	mdred	415
	lgred	1,154
	train	17,813
	test	9,808
twolf	ref	346,485
	smred	92
	mdred	259
	lgred	973
	train	13,200
parser	ref	546,748
	smred	269
	mdred	612
	lgred	4,527
	train	13,433
	test	4,203
bzip2	ref-source	108,878
	ref-program	124,927
	ref-graphic	143,565
	lgred-source	1,820
	lgred-program	2,159
	lgred-graphic	2,644
	train	61,128
	test	8,822

progr	input	insns
gcc	ref-scilab	62,031
	ref-integrate	13,164
	ref-expr	12,086
	ref-200	108,670
	ref-166	46,918
	smred	97
	mdred	551
	lgred	5,117
	test	2,016
gzip	ref-source	84,367
	ref-random	82,167
	ref-program	168,868
	ref-log	39,527
	ref-graphic	103,706
	smred-source	1,486
	smred-random	1,362
	smred-program	4,025
	smred-log	602
	smred-graphic	4,984
	mdred-source	1,552
	mdred-random	1,362
	mdred-program	2,732
	mdred-log	597
	mdred-graphic	1,209
	lgred-source	1,583
	lgred-random	1,361
	lgred-program	2,858
	lgred-log	593
	lgred-graphic	1,786
	train	57,970
	test	3,367

TABLE II

THE SPEC CPU2000 BENCHMARKS AND THEIR INPUTS USED IN THIS PAPER, ALONG WITH THEIR DYNAMIC INSTRUCTION COUNT (IN MILLIONS).

V. EXPERIMENTAL SETUP

In this paper we use a number of SPEC CPU2000 integer benchmarks; we used the Alpha binaries from the SimpleScalar website[2]. We use multiple inputs for each benchmark—there are 63 benchmark-input pairs in total; Table 2 lists both the benchmarks and input sets. Some of the inputs are taken from the SPEC distribution, others are taken from MinneSPEC. MinneSPEC is a set of reduced inputs for various SPEC CPU2000 benchmarks [8], [9]. These reduced input sets are derived from the reference inputs using a number of techniques: modifying inputs (for example, reducing the number of iterations), truncating inputs, etc. A distinction is made between three sets of reduced inputs: smred for short simulations (100 million instructions), mdred for medium length simulations (500 million instructions) and lgred for full length, reportable simulations (1 billion instructions). We chose these seven SPEC CPU benchmarks for the following reasons. First, for some other SPEC CPU benchmarks we obtained different instruction counts in our simulation and instrumentation infrastructures—this is due to system call

[1] http://www.cs.ucsd.edu/~calder/simpoint

[2] http://www.simplescalar.com

	Program characteristic	K-S	S-W
1	% arithmetic ops	$p < 0.01$	$p = 0.0018$
2	% logical ops	$p < 0.20$	$p = 0.0156$
3	% shift and byte ops	$p < 0.01$	$p = 0.0000$
4	% load/store ops	$p < 0.01$	$p = 0.0008$
5	% branch ops	$p < 0.01$	$p = 0.0001$
6	branch predictability	$p < 0.15$	$p = 0.0198$
7	control flow	$p < 0.10$	$p = 0.4069$
8	miss rate 8KB D-cache	$p < 0.01$	$p = 0.0000$
9	miss rate 16KB D-cache	$p < 0.01$	$p = 0.0000$
10	miss rate 32KB D-cache	$p < 0.05$	$p = 0.0000$
11	miss rate 64KB D-cache	$p < 0.01$	$p = 0.0000$
12	miss rate 128KB D-cache	$p < 0.01$	$p = 0.0000$
13	miss rate 8KB I-cache	$p < 0.01$	$p = 0.0000$
14	miss rate 16KB I-cache	$p < 0.01$	$p = 0.0000$
15	miss rate 32KB I-cache	$p < 0.01$	$p = 0.0000$
16	miss rate 64KB I-cache	$p < 0.01$	$p = 0.0000$
17	miss rate 128KB I-cache	$p < 0.01$	$p = 0.0000$
18	ILP	$p < 0.01$	$p = 0.0000$

TABLE IV
TESTING FOR NORMALITY USING THE KOLMOGOROV-SMIRNOV (K-S) AND THE SHAPIRO-WILKS' W TEST (S-W).

effects. Second, we limited the number of benchmarks because of the long instrumentation and simulation times that were required for collecting the data. Note also that the goal of this paper is to focus on the statistical techniques rather than to select the representative SPEC CPU2000 benchmarks.

The program characteristics that we measure in this paper are obtained using ATOM [22] which is a binary instrumentation tool for the Alpha architecture. ATOM allows for statically instrumenting executables at the function, basic block and instruction level. Executing the instrumented binary then yields the desired program characteristics.

The CPI numbers used in this paper for the reference inputs are taken from [23]; we simulated the other inputs ourselves using the same simulator. We report CPI prediction errors for predicting the CPI of an 8-issue as well as a 16-issue machine. Table III summarizes the important processor model parameters. The CPI numbers are obtained using SimpleScalar/Alpha v3.0 [24].

VI. EVALUATION

In this section we first evaluate whether the assumption made by PCA on the normality of the data set is valid. We subsequently evaluate the ability of ICA to subset the benchmark programs compared to those obtained by PCA. We do this in a number of experiments. We first vary the subset size looking for the optimal subset size. We subsequently inspect the subset and the representative benchmark/input pairs. We finally provide some results on simulation-centric workload subsetting.

A. Normality of the data set

As mentioned before, PCA assumes that the data has a Gaussian or normal distribution, whereas ICA does not make a similar assumption. In order to verify whether or not our data set has a Gaussian distribution we use two well known statistical tests for normality, namely the Kolmogorov-Smirnov test and the Shapiro-Wilks' W test—in fact, the Shapiro-Wilks' W test is the preferred test of normality. The null hypothesis for these tests is that the data is Gaussian distributed. We use STATISTICA [18] to apply these tests. The p-value reported by these tests should be greater than 0.05 to accept the null hypothesis and thus the assumption of normality with a 95% confidence. Table IV shows the outcome of these tests. These p-values are very low and thus provide evidence that assuming that the data has a Gaussian distribution is invalid. These initial results give us a first intuition why ICA might be a better and more robust approach than PCA.

B. Comparing PCA vs. ICA for subsetting

In order to compare PCA and ICA for the purpose of subsetting we provide the same data set as input to both PCA and ICA; this is the data set as detailed in Section 3. PCA and ICA then determine the important dimensions in this data set and project the data set on the lower dimensional workload space. These are the PCA-based and ICA-based workload spaces. In this reduced workload space, we subsequently apply K-means clustering in order to determine the groups of distinct program behaviors. This is done for a range of k values; we vary k from 5 up to 30 (recall the total number of program-input pairs is 63). For each value of k we determine 25 clusterings; this is done by considering 25 random seeds for choosing the initial (furthest-first) cluster centers. As discussed in Section 4.3, we then retain the clustering with the maximum BIC value. For this optimal clustering for a given value of k we then compute the CPI prediction error of the selected subset compared to t he complete set of benchmarks/inputs in our data set. The CPI prediction error is computed as follows. Our reference CPI is computed as the average CPI over all the benchmarks tabulated in Table II. The estimated CPI for the reduced workload is computed by taking a representative per cluster being the benchmark-input pair that is closest to the cluster centroid and computing the weighted average CPI over all clusters with the weights being the number of benchmark/input pairs in the cluster. The CPI prediction error then is the percentage difference between the reference CPI and the estimated CPI; the reported errors are absolute error numbers.

Figures 1 and 2 show the CPI prediction error as a function of the value of k (the subset size) chosen in the K-means algorithm for PCA and ICA, respectively. Figure 3 shows the delta CPI prediction error which is computed as the CPI prediction error for PCA minus the CPI prediction error for ICA—a positive delta CPI prediction error measures ICA is better than PCA. Various curves are shown while varying the dimensionality of the reduced workload space from 2 up to 6. As expected, these graphs show that the CPI prediction error generally decreases for larger subset sizes. These graphs also clearly show that ICA outperforms PCA, especially for the 3-, 4- and 5-dimensional spaces. The CPI prediction error for ICA is significantly lower across nearly all subset sizes. The error is usually below 5% for both the 8- and 16-issue machine; for PCA and the same dimensionality of the workload space frequently exceeds the 5% and often goes up to 10%.

These graphs also show that neither of these two statistical

	8-issue machine	16-issue machine
RUU/LSQ	128/64	256/128
Cache hierarchy	32KB L1 I/D-caches, 1MB L2, 16-entry store buffer	64KB L1 I/D-caches, 2MB L2, 32-entry store buffer
Latencies	1/12/100 cycles	2/16/100 cycles
Branch predictor	Hybrid 2K tables, 7 cycle front-end pipeline	Hybrid 8K tables, 10 cycle front-end pipeline
Processor width	8-wide	16-wide

TABLE III

PROCESSOR MODELS CONSIDERED IN THIS STUDY.

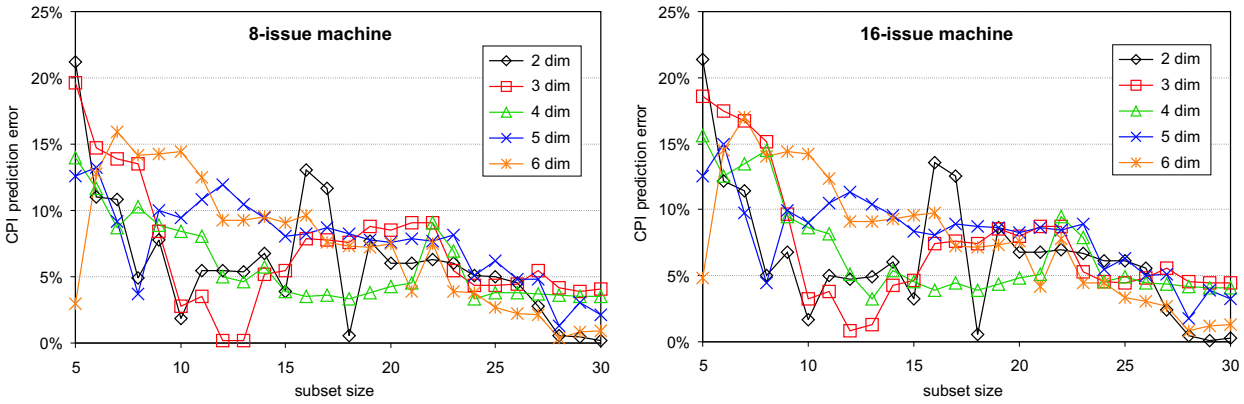

Fig. 1. Evaluating the CPI prediction error for PCA as a function of subset size for the 8-issue machine on the left and the 16-issue machine on the right.

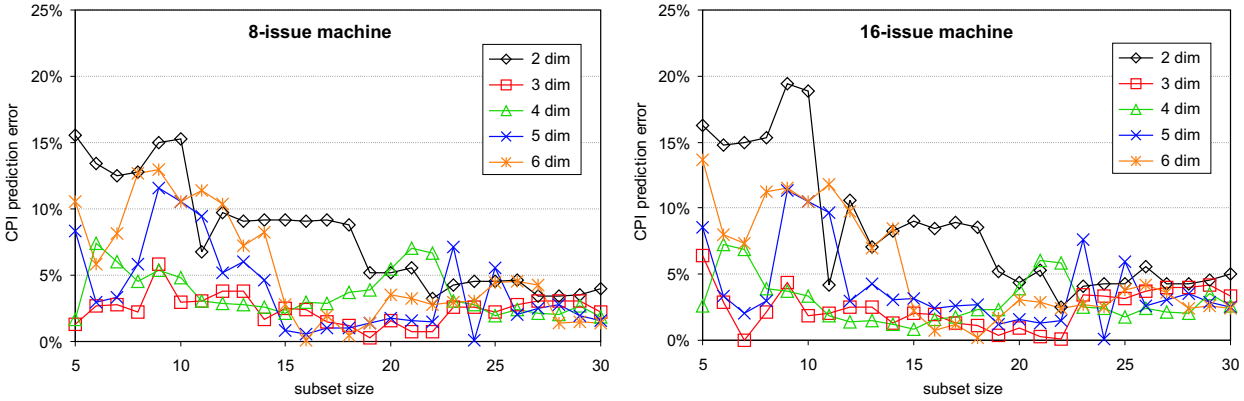

Fig. 2. Evaluating the CPI prediction error for ICA as a function of subset size for the 8-issue machine on the left and the 16-issue machine on the right.

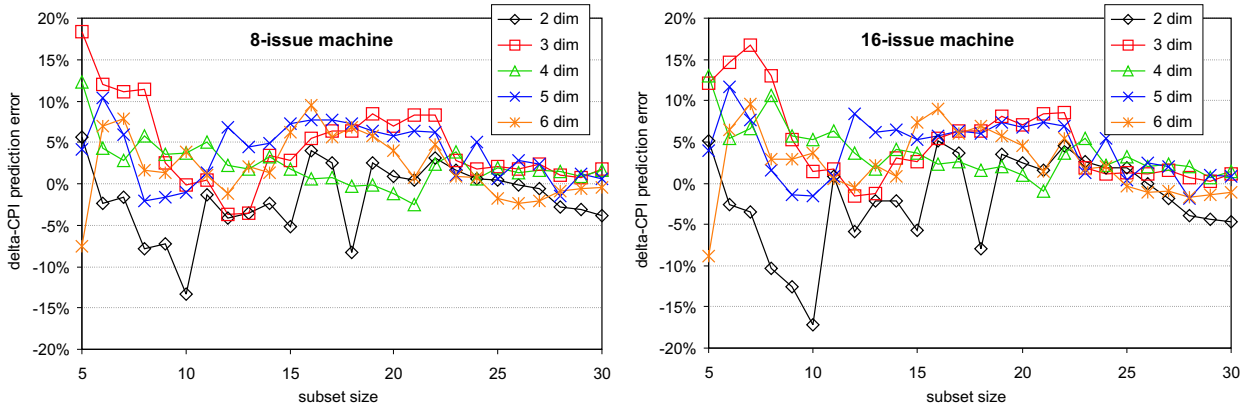

Fig. 3. The CPI prediction error for PCA minus the CPI prediction error for ICA as a function of subset size for the 8-issue machine on the left and the 16-issue machine on the right.

analysis techniques are capable of determining an accurate subset in a 2-dimensional space. Apparently, the 2-dimensional workload space does not contain enough information from the original space in order to discriminate distinct program behaviors. The 6-dimensional workload space on the other hand, contains too much information which results in higher CPI prediction errors. We observed similar results for higher dimensionality data—not shown here to preserve the readability of the graphs. The reason is that some of the higher dimensional axes inflate some of the rather unimportant underlying program characteristics in the original data set. This phenomenon, reduced clustering quality for a higher dimensionality in the data, is well known as the *curse of dimensionality*.

We thus conclude that a 3- to 5-dimensional space results in the best subsets in conjunction with ICA. These 3, 4 and 5 dimensions collectively account for 74.1%, 83.1% and 89.2% of the total variation, respectively. Using 2 and 6 dimensions account for 60.4% and 93.1% of the total variation, respectively. As such, our recommendation for applying this methodology for other benchmark suites would be to select a number of dimensions so that they account for 70% to 90% of the total variation. Using fewer or more principal components is likely to result in poorer performing subsets.

C. Optimal subset size

So far we considered a range of subset sizes and evaluated the performance for each of them. However, in practice it is undesirable to evaluate a range of subset sizes as done in the previous section using CPI prediction error numbers—this would mean we need to simulate all benchmarks and this is exactly what we are trying to avoid in this paper. As such, it would be desirable if we could find a well performing subset without having to evaluate every subset size based on detailed simulation results.

The data presented in Figures 1 and 2 provide evidence that a well performing subset exists. For example, small subset sizes are unlikely to perform well since they are unable to cover the range of program behavior adequately. Larger subsets on the other hand, are likely to be more accurate, although more simulation time will be required since there are more benchmarks in the subset.

In order to find a well performing benchmark subset in terms of accuracy and simultion time required, we propose to use the BIC criterion used during K-means clustering. During cluster analysis we evaluated the BIC scores for all the subset sizes k from 5 up to 20 (and for each k we evaluate 25 random seeds for finding the initial clustering) and then picked the clustering that results in the highest BIC score. Figure 4 shows evidence that the BIC score indeed is a good metric for the subset quality. This graph shows the subset CPI prediction error as a function of the BIC score obtained through clustering for the 3-dimensional PCA and ICA spaces. We clearly observe that higher BIC scores correspond to lower CPI prediction errors.

When picking the clustering with the optimal BIC score over a range of k values, we obtain the results given in Table V. This table shows the percentage CPI prediction error for the 8- and 16-issue machines for PCA and ICA. We observe that ICA achieves better accuracy with smaller subsets for the 3-, 4- and 5-dimensional workload spaces. For example, in the 3-dimensional space, ICA yields a subset size of 11 with a CPI prediction error in the range of 2% to 3%; PCA yields a subset size of 18 with a CPI prediction error in the range 7% to 8%.

So far we considered the optimal BIC score for determining the optimal clustering, *i.e.*, we considered a range of k values and for each k we considered a number of random seeds; the optimal clustering is then determined as the one with the highest BIC score. In general, the clustering with the highest BIC score is likely to be a clustering with a large number of subsets. However, we could for example relax the constraint on the optimal BIC score and pick the clustering that corresponds to a given *BIC percentage* (*e.g.*, 90%) of the observed BIC range. Figure 5 quantifies the impact of the BIC percentage on prediction accuracy and subset size. We conclude that as the BIC percentage increases, the prediction error tends to decrease while the subset size increases. It is also interesting to note that a low-dimensional space seems to be less sensitive to the BIC percentage compared to a high-dimensional space. As a result, if one is interested in a rather small workload, it is better to consider a low-dimensional space than a high-dimensional space with a small BIC percentage. If on the other hand, a fairly large reduced workload works fine for a given purpose, a high-dimensional space and a high BIC percentage would be recommended.

D. Benchmark clusters

Tables VI and VII show the clusters and the benchmarks/inputs per cluster obtained from applying K-means clustering in a 3-dimensional ICA space and PCA space, respectively. The representatives per cluster are shown in bold. There are 11 clusters in total and thus there are 11 representatives. The representative is the one closest to the cluster centroid. In case a cluster only contains two benchmarks/inputs, either one can be chosen as the cluster representative.

We first analyze the clustering result based on ICA. The results given in Table VI show that for many benchmarks, several inputs result in similar program behavior. For example, for twolf all inputs reside in one single cluster; for gzip many of its inputs reside in a single cluster, although a few inputs, especially program, seem to result in different behavior. This clustering result also provides valuable information about the representativeness of the MinneSPEC reduced inputs. For example for vortex only the mdred input seems to result in a program behavior that is similar to the reference inputs; the smred and lgred inputs do not. Similarly, for gcc only the lgred input results in program behavior similar to the reference inputs; the other reduced inputs do not.

The results we obtain from PCA are quite different, see Table VII. When comparing Table VII for PCA versus Table VI for ICA there seems to be only two clusters that are identical, namely clusters 10 and 11. There are a number of clusters

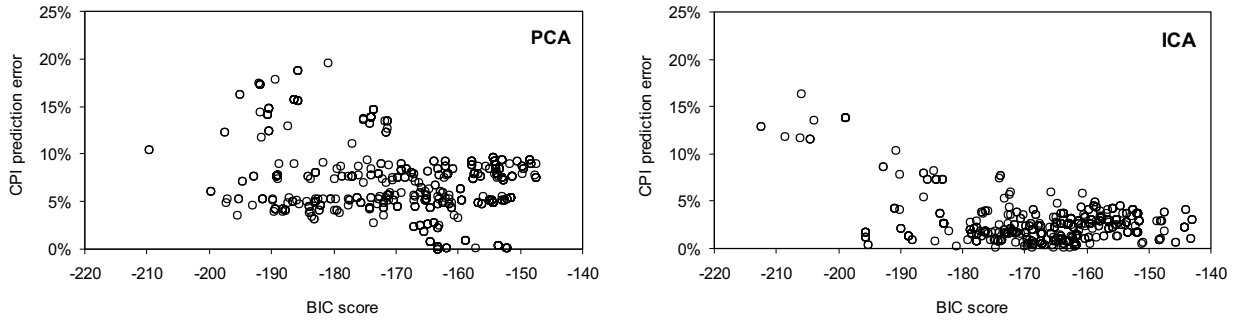

Fig. 4. CPI prediction error for the 8-issue machine as a function of the BIC score. This is for the 3-dimensional PCA space on the left and the 3-dimensional ICA space on the right.

	PCA			ICA		
	err. 8-issue	err. 16-issue	subset size	err. 8-issue	err. 16-issue	subset size
2-dim	5.5%	4.7%	12	12.5%	14.9%	7
3-dim	7.6%	7.5%	18	3%	2%	11
4-dim	3.4%	3.9%	18	2.8%	1.5%	13
5-dim	7.8%	8.6%	19	1.8%	1.6%	20
6-dim	7.5%	7.6%	20	3.5%	3.1%	20

TABLE V

CPI PREDICTION ERROR AND SUBSET SIZE FOR OPTIMAL SUBSETS OBTAINED THROUGH PCA AND ICA.

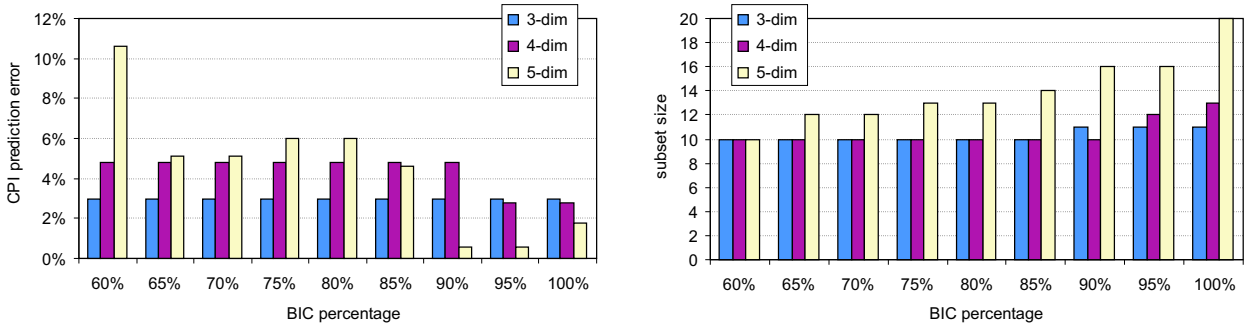

Fig. 5. CPI prediction error for the 8-issue machine (on the left) and subset size (on the right) as a function of the BIC percentage for the ICA space.

that show some similarities between PCA and ICA, namely clusters 7 up to 9. The remaining clusters, 1 to 6, show large disagreements between PCA and ICA.

An interesting note that could be made here is that ICA seems to group the benchmarks more on a per-benchmark basis, *i.e.*, multiple inputs for the same benchmark seem to be grouped in a single cluster. This is less the case for PCA; several clusters contain inputs from multiple benchmarks. This gives some additional intuitive support for the ICA approach over the PCA approach, next to the higher accuracy obtained for the ICA approach. Indeed, it is to be expected intuitively that there is more similarity across inputs for a given benchmark program than there is across different benchmark programs. This is also shown in Figures 6 and 7 where the 3-dimensional workload space is shown for PCA and ICA, respectively. In these graphs, the benchmark/input pairs are shown in the workload space; the axes in the workload space are the principal components. In the ICA space the points seem to be grouped more on a per-benchmark basis; and this seems to be more the case than in the PCA space. This explains the clustering results of Tables VI and VII. Based on this observation we can conclude that ICA is especially interesting for selecting representative inputs from a set of potential inputs for a given benchmark. Note that the simulation speedup obtained through ICA comes from two sources: (i) cross-program similarity—there are a few clusters that contain multiple benchmarks, and (ii) more importantly, cross-input similarity—ICA selects a reduced input as a representative for the reference input.

E. Simulation-time centric workload composition

In the above clustering results, a representative is chosen as the benchmark/input pair that is closest to the centroid for that cluster. Simulating only these benchmark/input pairs instead of the complete set of benchmarks results in a CPI prediction error of 3% and 1.9% for the 8-issue and 16-machine, respectively, when applying the clustering in the 3-dimensional ICA space. The simulation speedup that is obtained by only simulating these selected benchmarks is 12.2X compared to simulating the complete set of benchmark/input pairs.

ID	Size	Clusters and their representatives
1	2	vortex-smred/lgred
2	2	parser-smred/mdred
3	5	gzip-program-smred/mdred/**lgred**/ref, gzip-graphic-smred
4	3	gcc-test/smred/**mdred**
5	6	gcc-ref-scilab/integrate/**expr**/200/166, gcc-train
6	5	twolf-train/smred/mdred/**lgred**/ref
7	17	gzip-train, gzip-test, gzip-smred-source/random/log, gzip-mdred/lgred/**ref**-source/random/**log**/graphic
8	5	parser-train/test/ref/**lgred**, bzip2-test
9	7	bzip2-lgred/**ref**-source/program/**graphic**, bzip2-train
10	4	vpr-train/ref/mdred/**lgred**
11	7	vpr-smred, vortex-**train**/test/mdred/ref-1/ref-2/ref-3

TABLE VI

CLUSTERING RESULT IN THE 3-DIM ICA SPACE; THE CPI PREDICTION ERROR IS 3% AND 2% FOR THE 8-ISSUE AND 16-ISSUE MACHINE, RESPECTIVELY.

ID	Size	Clusters and their representatives
1	2	twolf-train/ref
2	3	gzip-graphic-smred, gzip-**program**-ref/**lgred**
3	5	gcc-**mdred**/test/smred, vortex-smred/lgred
4	6	gzip-source-smred/**mdred**/lgred/ref, gzip-program-smred/mdred
5	2	gcc-ref-166/integrate
6	4	gcc-train, gcc-ref-**scilab**/expr/200
7	13	gzip-train/test, gzip-program-smred, gzip-random-mdred/lgred/ref, gzip-**log**-smred/mdred/**lgred**/ref, gzip-graphic-mdred/lgred/ref
8	7	parser-train/test/ref/smred/mdred/**lgred**, twolf-lgred
9	10	bzip2-**lgred**/ref-source/program/**graphic**, twolf-test/smred
10	4	vpr-train/ref/mdred/**lgred**
11	7	vpr-smred, vortex-**train**/test/mdred/ref-1/ref-2/ref-3

TABLE VII

CLUSTERING RESULT IN THE 3-DIM PCA SPACE; THE CPI PREDICTION ERROR IS 3.5% AND 3.7% FOR THE 8-ISSUE AND 16-ISSUE MACHINE, RESPECTIVELY.

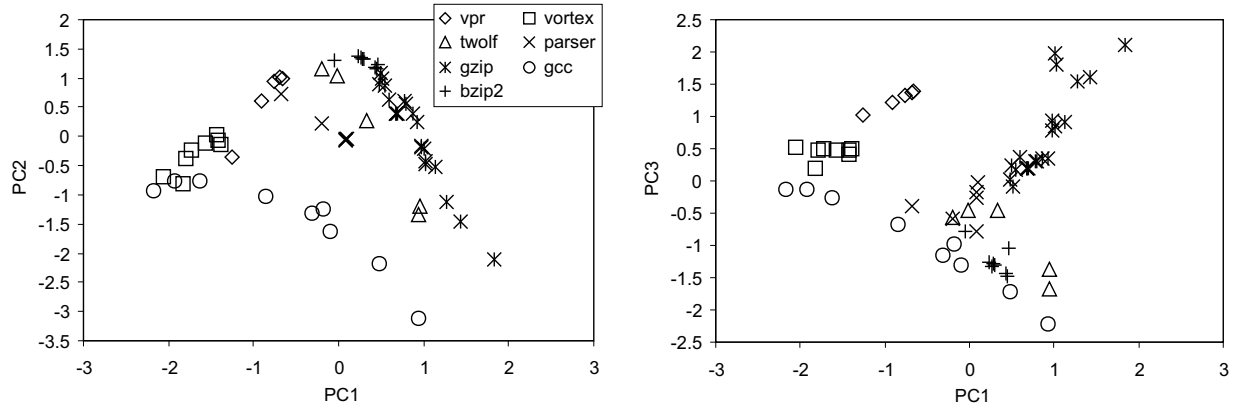

Fig. 6. The 3-dimensional PCA space: second vs. first principal component on the left, and third vs. first principal component on the right.

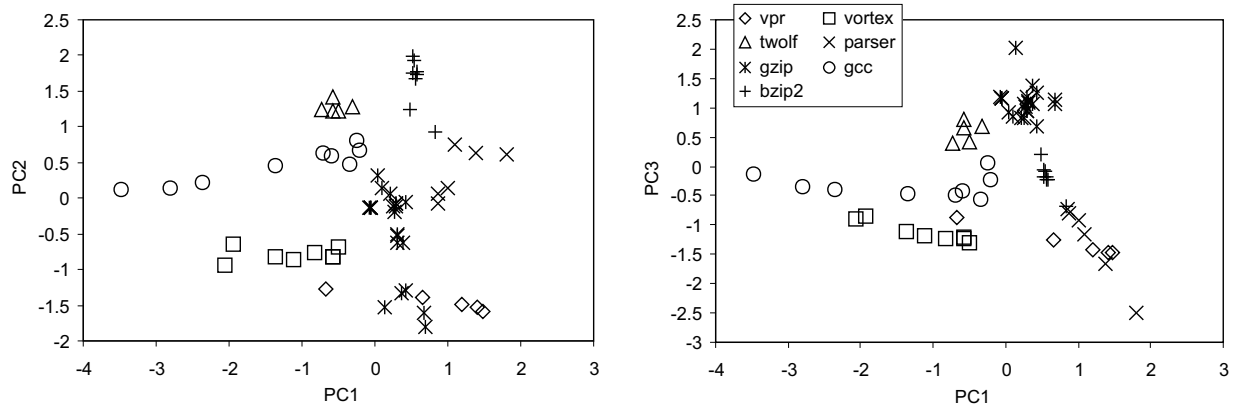

Fig. 7. The 3-dimensional ICA space: second vs. first principal component on the left, and third vs. first principal component on the right.

An alternative to selecting as the representative the one benchmark closest to the centroid would be to select the benchmark/input pair with the smallest dynamic instruction count in that cluster. This might be an attractive alternative when simulation speed is a primary concern. By doing so, we achieve a simulation speed of 180.6X over simulating all benchmark/input pairs, which is a substantial simulation speedup over selecting a representative closest to the centroid. Obviously, by doing so we trade simulation speed for accuracy. In our case, the CPI prediction error now is 6.3% and 6.1% for the 8-issue and 16-issue machine, respectively. In future work it would be interesting to study how simulation speed and accuracy can be traded-off more carefully so that a major part of the 180.6X simulation speedup is preserved while reducing the prediction error as much as possible.

VII. Conclusion

Composing a workload is crucial throughout the design cycle of a microprocessor. The workload should be representative of the target application domain and at the same time the redundancy within the workload should be minimized. The goal is to make an intelligent trade-off between workload representativeness (and thus simulation accuracy) versus simulation speed.

Prior work on benchmark subsetting or workload composition typically measured a number of program characteristics and subsequently performed Principal Components Analysis (PCA) on these inputs in order to remove the correlation in the data set. Cluster Analysis is then performed on this transformed data set. This paper proposed Independent Components Analysis (ICA) as a better alternative for selecting the key dimensions in the input data than PCA. The primary difference between PCA and ICA is that PCA focuses on finding uncorrelated axes in a multi-dimensional space whereas ICA tries to find the (stronger in a statistical sense) independent axes. Furthermore, we observe that the data is not Gaussian across all 18 original dimensions. As a result, PCA is not capable of capturing the true succinct representation. The ICA algorithm is estimated by measuring the non-gaussianity of the data, which is more appropriate for the measured data. These differences makes ICA a more robust technique than PCA for quantifying benchmark similarity.

Applying ICA for benchmark subsetting or workload composition leads to both smaller and more accurate subsets than PCA. Our experimental results using SPEC CPU2000 have shown that ICA is indeed capable of identifying reduced but representative workloads. For example, in a 3-dimensional space, ICA identifies a subset of 11 benchmark/input pairs to be optimal whereas PCA finds a subset of 18 benchmark/input pairs. Additionally, the CPI prediction error for ICA is in the range of 2% to 3% whereas the error for PCA is in the range of 7% to 8%.

Acknowledgements

Lieven Eeckhout is a Postdoctoral Fellow of the Fund for Scientific Research – Flanders (Belgium) (F.W.O. Vlaanderen). This research is also supported by Ghent University, the European HiPEAC network of excellence, IWT, the Minnesota Supercomputing Institute.

References

[1] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Quantifying the impact of input data sets on program behavior and its applications," *JILP*, vol. 5, Feb. 2003, http://www.jilp.org/vol5.

[2] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John, "Measuring program similarity: Experiments with spec cpu benchmark suites," in *ISPASS'05*, Mar. 2005, pp. 10–20.

[3] J. J. Dujmovic and I. Dujmovic, "Evolution and evaluation of SPEC benchmarks," *ACM Performance Evaluation Review*, vol. 26, no. 3, pp. 2–9, Dec. 1998.

[4] H. Vandierendonck and K. De Bosschere, "Experiments with subsetting benchmark suites," in *WWC-7*, Oct. 2004, pp. 55–62.

[5] J. J. Yi, D. J. Lilja, and D. M. Hawkins, "A statistically rigorous approach for improving simulation methodology," in *HPCA-9*, Feb. 2003, pp. 281–291.

[6] R. H. Saavedra and A. J. Smith, "Analysis of benchmark characteristics and benchmark performance prediction," *ACM TOCS*, vol. 14, no. 4, pp. 344–384, Nov. 1996.

[7] D. Citron, "MisSPECulation: Partial and misleading use of SPEC CPU2000 in computer architecture conferences," in *ISCA-31*, June 2003, pp. 52–59.

[8] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Designing workloads for computer architecture research," *IEEE Computer*, vol. 36, no. 2, pp. 65–71, Feb. 2003.

[9] A. J. KleinOsowski and D. J. Lilja, "MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research," *Computer Architecture Letters*, vol. 1, no. 2, pp. 10–13, June 2002.

[10] T. M. Conte, M. A. Hirsch, and K. N. Menezes, "Reducing state loss for effective trace sampling of superscalar processors," in *ICCD-96*, Oct. 1996, pp. 468–477.

[11] E. Perelman, G. Hamerly, and B. Calder, "Picking statistically valid and early simulation points," in *PACT-2003*, Sept. 2003, pp. 244–256.

[12] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ASPLOS-X*, Oct. 2002, pp. 45–57.

[13] R. E. Wunderlich, T. F. Wenisch, B. Falsafi , and J. C. Hoe, "SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling," in *ISCA-30*, June 2003, pp. 84–95.

[14] L. Eeckhout, R. H. Bell Jr., B. Stougie, K. De Bosschere, and L. K. John, "Control fbw modeling in statistical simulation for accurate and effi cient processor design studies," in *ISCA-31*, June 2004, pp. 350–361.

[15] S. Nussbaum and J. E. Smith, "Modeling superscalar processors via statistical simulation," in *PACT-2001*, Sept. 2001, pp. 15–24.

[16] M. Oskin, F. T. Chong, and M. Farrens, "HLS: Combining statistical and symbolic simulation to guide microprocessor design," in *ISCA-27*, June 2000, pp. 71–82.

[17] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*, 5th ed. Prentice Hall, 2002.

[18] StatSoft, Inc., STATISTICA for Windows. Computer program manual. 1999. http://www.statsoft.com, 1999.

[19] A. Hyvärinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

[20] A. Hyvärinen, "Fast and robust fi xed-point algorithms for independent components analysis," *IEEE Transactions on Neural Networks*, vol. 10, no. 3, pp. 626–634, 1999.

[21] ——, "Survey on independent component analysis," *Neural Computing Surveys*, vol. 2, pp. 94–128, 1999.

[22] A. Srivastava and A. Eustace, "ATOM: A system for building customized program analysis tools," Western Research Lab, Compaq, Tech. Rep. 94/2, Mar. 1994.

[23] T. F. Wenisch, R. E. Wunderlich, B. Falsafi , and J. C. Hoe, "Applying SMARTS to SPEC CPU2000," Computer Architecture Lab at Carnegie Mellon University, Tech. Rep. 2003-1, June 2003.

[24] D. C. Burger and T. M. Austin, "The SimpleScalar Tool Set," Computer Architecture News, 1997, see also http://www.simplescalar.com for more information.