

# How Input Data Sets Change Program Behaviour

Lieven Eeckhout    Hans Vandierendonck    Koen De Bosschere

Department of Electronics and Information Systems (ELIS), Ghent University, Belgium

E-mail: {leeckhou, hvdieren, kdb}@elis.rug.ac.be

## Abstract

*Having a representative workload of the target domain of a microprocessor is extremely important throughout its design. The composition of a workload involves two issues: (i) which benchmarks to select and (ii) which input data sets to select per benchmark. Unfortunately, we are unable to select a huge number of benchmarks and respective input sets due to limitations on the available simulation time. In this paper, we use principal components analysis (PCA) to efficiently explore the workload space. Within this workload space, different input data sets for a given benchmark can be displayed and representative input data sets can be selected for the given benchmark. The final goal is to select a limited set of representative benchmark-input tuples that span the complete workload space.*

## 1 Introduction

The first step when designing a new microprocessor is to compose a workload that should be representative for the set of applications that will be run on the microprocessor once it will be used in a commercial product [2, 12]. A workload then typically consists of a number of benchmarks with respective input data sets taken from various benchmarks suites, such as SPEC, TPC, MediaBench, etc. This workload will then be used during the various simulation runs to perform design space explorations. It is obvious that composing a representative workload is extremely important in order to obtain a design that is optimal for the target environment of operation. The question when composing a representative workload is thus twofold: (i) which benchmarks and (ii) which input data sets to select. In addition, we have to take into account that even high-level architectural simulations are extremely time-consuming. As such, the total simulation time should be limited as much as possible to limit the time-to-market. This implies that the total number of benchmarks and input data sets should be limited without compromising the final design. Ideally, we would like to have a limited set of benchmark-input tuples span-

ning the complete workload design space, which contains a variety of the most important types of program behaviour.

Conceptually, the complete workload design space can be viewed as a  $p$ -dimensional space with  $p$  the number of important program characteristics that affect performance, e.g., branch prediction accuracy, cache miss rates, instruction-level parallelism, etc. Obviously,  $p$  will be too large to display the workload design space understandably. In addition, correlation exists between these variables which reduces the ability to understand what program characteristics are fundamental to make the diversity in the workload space. In this paper, we reduce the  $p$ -dimensional workload space to a  $q$ -dimensional space with  $q \ll p$  ( $q = 2$  or  $q = 3$  typically) making the visualisation of the workload space possible without losing important information. This is achieved by using principal components analysis (PCA). We will show that PCA can be used efficiently to explore the workload design space in general and to measure the impact of input data sets on program behaviour in particular.

Each benchmark-input tuple is a point in this  $q$ -dimensional space (obtained after PCA). We can expect that different benchmarks will be ‘far away’ from each other while different input data sets for a single benchmark will be clustered together. This representation gives us an excellent opportunity to measure the impact of input data sets on program behaviour. Weak clustering (for various inputs and a single benchmark) indicates that the input set has a large impact on program behaviour; strong clustering on the other hand, indicates a small impact.

In addition, this representation gives us an idea which input sets should be selected when composing a workload. Strong clustering suggests that a single or only a few input sets could be selected to be representative for the cluster. This will reduce the total simulation time significantly for two reasons: (i) the total number of benchmark-input tuples is reduced; and (ii) the benchmark-input tuple selected to represent a cluster can have a small dynamic instruction count compared to the other benchmark-input tuples in the cluster. The reduction of the total simulation time is an important issue for the evaluation using commercial workloads since commercial workloads tend to have large dynamic in-

struction counts in order to be representative.

This paper is organized as follows. In section 2, the program characteristics used are enumerated. Principal components analysis and its use in this context are discussed in section 3. In section 4, it is shown that PCA is useful in the context of workload characterization. In addition, we discuss how input data sets affect program behaviour. Section 5 discusses related work. We conclude in section 6.

## 2 Workload Characterization

It is important to select program characteristics that affect performance for performing PCA in the context of workload characterization. Selecting program characteristics that do not affect performance, such as the dynamic instruction count, might discriminate benchmark-input tuples on a characteristic that does not affect performance, yielding no information about the behaviour of the benchmark-input tuple when executed on a microprocessor. We have identified the following program characteristics:

- **Instruction mix.** We consider five instruction classes: integer arithmetic operations, logical operations, shift and byte manipulation operations, load/store operations and control operations.
- **Branch prediction accuracy.** We consider the branch prediction accuracy of three branch predictors: a bimodal branch predictor, a gshare branch predictor and a hybrid branch predictor. The bimodal branch predictor consists of an 8K-entry table containing 2-bit saturating counters which is indexed by the program counter of the branch. The gshare branch predictor is an 8K-entry table with 2-bit saturating counters indexed by the program counter xor-ed with the taken/not-taken branch history of 12 past branches. The hybrid branch predictor [18] combines the bimodal and the gshare predictor by choosing among them dynamically. This is done using a meta predictor that is indexed by the branch address and contains 8K 2-bit saturating counters.
- **Data cache miss rates.** Data cache miss rates were measured for five different cache configurations: an 8KB and a 16KB direct-mapped cache, a 32KB and a 64KB two-way set-associative cache and a 128KB four-way set-associative cache. The line size was set to 32 bytes.
- **Instruction cache miss rates.** Instruction cache miss rates were measured for the same cache configurations mentioned for the data cache.
- **Sequential flow breaks.** We have also measured the number of instructions between two sequential flow

breaks or, in other words, the number of instructions between two taken branches.

- **Instruction-level parallelism.** To measure the amount of ILP in a program, we consider an infinite-resource machine, i.e., infinite number of functional units, perfect caches, perfect branch prediction, etc. In addition, we schedule instructions as soon as possible assuming unit execution instruction latency. The only dependencies considered between instructions are read-after-write (RAW) dependencies through registers as well as through memory. In other words, perfect register and memory renaming are assumed in these measurements.

For this study, there are  $p = 20$  program characteristics on which PCA is performed.

## 3 Principal Components Analysis

Principal components analysis (PCA) [17] 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, program characteristics) are correlated and hence, they measure the same or similar properties of the program-input tuples. PCA computes new variables, called *principal components*, which are *linear combinations* of the original variables, such that all principal components are uncorrelated. 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  $\sum_{i=1}^p Var[Z_i]$ .

Because the total amount of variance is kept unchanged, some of the principal components will have only a small variance. Hence they have more or less the same value for all program-input tuples. By removing these components from the measurement data, we can reduce the number of workload characteristics, while controlling the amount of information that we throw away. We retain  $q$  principal components which is a significant information reduction since  $q \ll p$  in most cases, typically  $q = 2$  or  $q = 3$ . To measure the fraction of information retained in this  $q$ -dimensional space, we use the amount of variance  $(\sum_{i=1}^q Var[Z_i]) / (\sum_{i=1}^p Var[Z_i])$  accounted for by these  $q$  principal components.

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 the 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 ILP is orders of magnitude larger than the variance of the data cache miss rates.

In this study the  $p$  original variables are the program characteristics mentioned in section 2. By examining the most important  $q$  principal components, which are linear combinations in the original program characteristics, meaningful interpretations can be given to these principal components in terms of the original program characteristics. To facilitate the interpretation of the principal components, we apply the *varimax* rotation [17] in the  $q$ -dimensional space. This rotation makes the coefficients  $a_{ij}$  either close to  $\pm 1$  or zero, such that the original variables either have a strong impact on a principal component or they have no impact. Note that the rotated principal components are still uncorrelated.

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. As such, a view can be given on the workload design space and the impact of input data sets on program behaviour can be displayed, as will be discussed in the next section.

We used STATISTICA'99 edition [1], a package for statistical computations, to perform PCA. This works as follows. A 2-dimensional matrix is presented as input to STATISTICA in which the columns represent the original variables, in our case the  $p = 20$  program characteristics from section 2. The rows of this matrix represent the various program-input tuples, which will be enumerated in the next section. On this matrix, PCA is performed by STATISTICA which yields us the principal components. Once these principal components are obtained, it is up to the user to determine which principal components should be retained. This decision is made based on the amount of variance accounted for by the retained principal components. STATISTICA also computes the coefficients of the various program-input tuples as a function of the retained principal components. These data can be used to represent the program-input tuples in a  $q$ -dimensional space built up by these  $q$  retained principal components. The results of this analysis will be presented in the evaluation section of this paper.

## 4 Evaluation

### 4.1 Experimental Setup

In this study, we have used the SPECint95 benchmarks. The reason why we choose SPECint95 instead of SPECint2000 is to limit the simulation time since we wanted to work with reference inputs as much as possible.

In addition to SPECint95, we used `postgres` v6.3 running the decision support TPC-D queries over a 100MB Btree-indexed database. For `postgres`, we ran all TPC-D queries except for query 1 because of memory constraints on our simulation machine. Details on the benchmarks and their input sets are given in Table 1.

The benchmarks were compiled with optimization level `-O4` and linked statically with the `-non_shared` flag for the Alpha architecture. The characteristics measured in this paper are all dynamic characteristics. This was done by instrumenting these programs with ATOM [23], a binary instrumentation tool for the Alpha architecture.

### 4.2 Results

In this section, we will first perform PCA on the data for the various input sets of `gcc`. Subsequently, the same will be done for `li` and `postgres`, respectively. Finally, PCA will be applied on the data for all the benchmark-input tuples of Table 1.

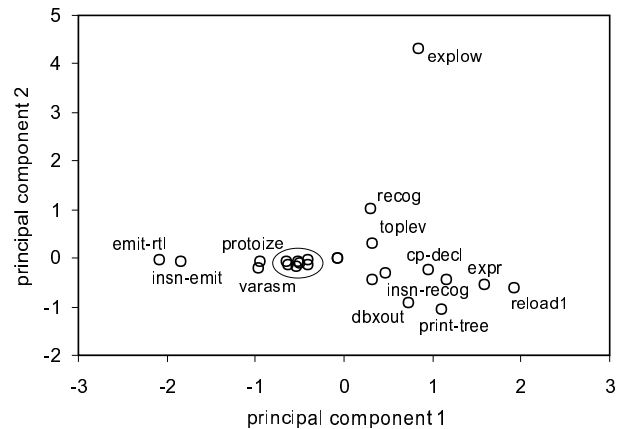


Figure 1. Gcc.

**Gcc.** PCA extracted two principal components from the data of `gcc`. These two principal components together account for 88.3% of the total variance. After varimax rotation, the first component is positively dominated, see Table 2, by the branch prediction accuracy, the percentage of arithmetic and logical operations; and negatively dominated by the I-cache miss rates. The second component is positively dominated by the D-cache miss rates, the percentage of shift and control operations; and negatively dominated by the ILP, the percentage of load/store operations and the number of instructions between two taken branches. Figure 1 presents the various input sets of `gcc` in the 2-dimensional space built up by these two components. Data

benchmark	input	dyn (M)	stat	mem (K)
gcc	amptjp	835	147,402	375
	c-decl-s	835	147,369	375
	cccp	886	145,727	371
	cp-decl	1,103	143,153	579
	dbxout	141	120,057	215
	emit-rtl	104	127,974	108
	explow	225	105,222	280
	expr	768	142,308	653
	gcc	141	129,852	125
	genoutput	74	117,818	104
	genrecog	100	124,362	133
	insn-emit	126	84,777	199
	insn-recog	409	105,434	357
	integrate	188	133,068	199
	jump	133	126,400	130
	print-tree	136	118,051	201
	protoize	298	137,636	159
	recog	227	123,958	161
	regclass	91	125,328	117
	reload1	778	146,076	542
	stmt-ptoize	654	148,026	261
	stmt	356	138,910	250
	toplev	168	125,810	218
	varasm	166	139,847	168
postgres	Q2F	227	57,297	345
	Q3F	948	56,676	358
	Q4F	564	53,183	285
	Q5F	7,015	60,519	654
	Q6F	1,470	46,271	1,080
	Q7a	9.6	34,103	189
	Q8a	11.8	34,125	192
	Q9a	9.5	32,843	189
	Q10F	1,794	62,564	681
	Q11a	6.6	34,126	186
	Q12a	6.3	34,294	185
	Q13a	5.9	32,725	184
	Q14a	5.9	35,404	184
	Q15F	5.5	35,138	183
	Q16F	82,228	58,067	389
	Q17F	183	54,835	366

**Table 1. Characteristics of the benchmarks used with their inputs, dynamic instruction count (in million), static instruction count (number of instructions executed at least once) and data memory footprint in 64-bit words (in thousands).**

points in this graph with a high value along the first component, have high branch prediction accuracies and high percentages of arithmetic and logical operations compared to the other data points; in addition, these data points also have low I-cache miss rates. Note that these data are normalized. Thus, only relative distances are important. For example, `emit-rtl` and `insn-emit` are relatively closer to each other than `emit-rtl` and `cp-decl`.

Figure 1 shows that `gcc` executing input set `explow` exhibits a different behaviour than the other input sets. This is due to its high D-cache miss rates, its high percentage of shift and control operations, and its low ILP, its low percentage of load/store operations and its low number of instructions between two taken branches. The input sets `emit-rtl` and `insn-emit` have a high I-cache miss rate, a low branch

benchmark	input	dyn (M)	stat	mem (K)
li	boyer	226	9,067	36
	browse	672	9,607	39
	ctak	583	8,106	18
	dderiv	777	9,200	16
	deriv	719	8,826	15
	destru2	2,541	9,182	16
	destrum2	2,555	9,182	16
	div2	2,514	8,546	19
	puzzle0	2	8,728	19
	tak2	6,892	8,079	16
	takr	1,125	8,070	36
	triang	3	9,008	15
jpeg	vigo.ppm	817	16,037	1,273
	specmun.ppm	730	15,952	1,136
	penguin.ppm	790	16,128	1,227
go	50 9 2stone9.in	593	55,894	45
	50 21 9stone21.in	35,758	62,435	57
	50 21 5stone21.in	35,329	62,841	57
compress	5000000 e 2231	21,495	4,494	1,715
	1000000 e 2231	4,342	4,490	433
	500000 e 2231	2,182	4,496	272
	100000 e 2231	423	4,361	142
m88ksim	train.in	24,959	11,306	4,834
perl	jumble	2,945	21,343	5,951
vortex	train	3,244	78,766	1,266

prediction accuracy and a low percentage of arithmetic and logical operations; for `reload1` the opposite is true. The strong cluster in the middle of the graph contains the input sets `gcc`, `genoutput`, `genrecog`, `jump`, `regclass`, `stmt` and `stmt-ptoize`. Note that although the characteristics mentioned in Table 1 (i.e., dynamic and static instruction count, and data memory footprint) are significantly different, these input sets result in quite similar program behaviour.

**Li.** PCA extracted two principal components from the data of the lisp interpreter. These two principal components together account for 75.6% of the total variance. After varimax rotation, the first component is positively dominated, see Table 2, by the percentage of memory operations, the

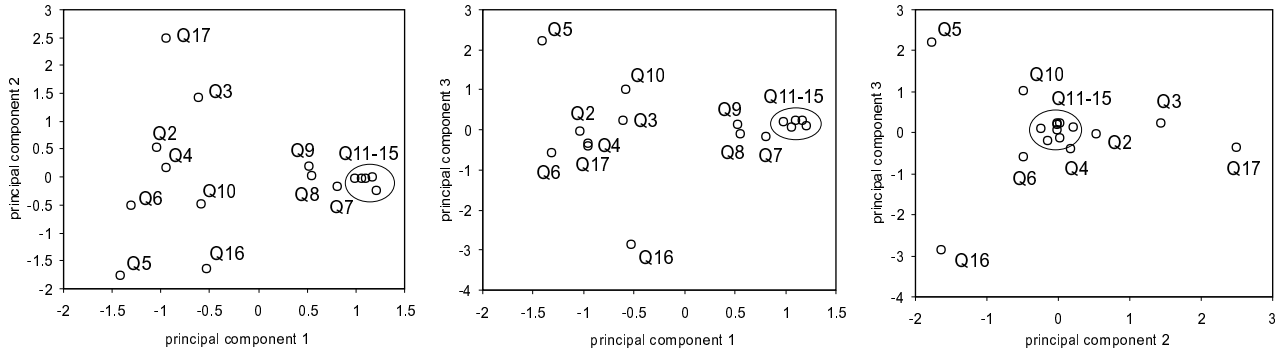


Figure 3. Postgres.

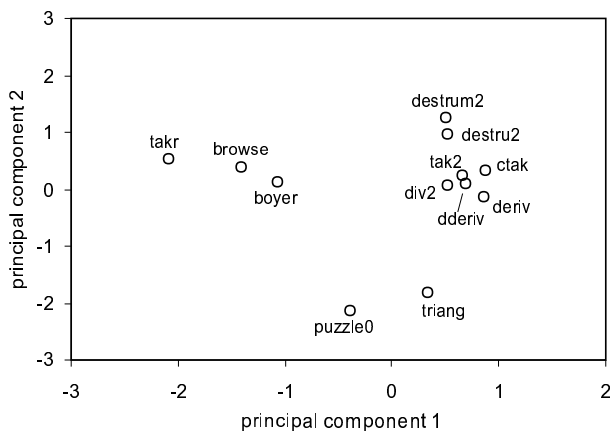


Figure 2. Lisp interpreter.

number of instructions between two taken branches and the miss rate for the 8KB I-cache; and negatively dominated by the percentage of logical and shift operations, and the miss rates for D-caches larger than 32KB. The second component is positively dominated by the amount of ILP and the miss rate for the D-caches smaller than 16KB; and negatively dominated by the percentage of arithmetic operations and the miss rate for I-caches larger than 32KB.

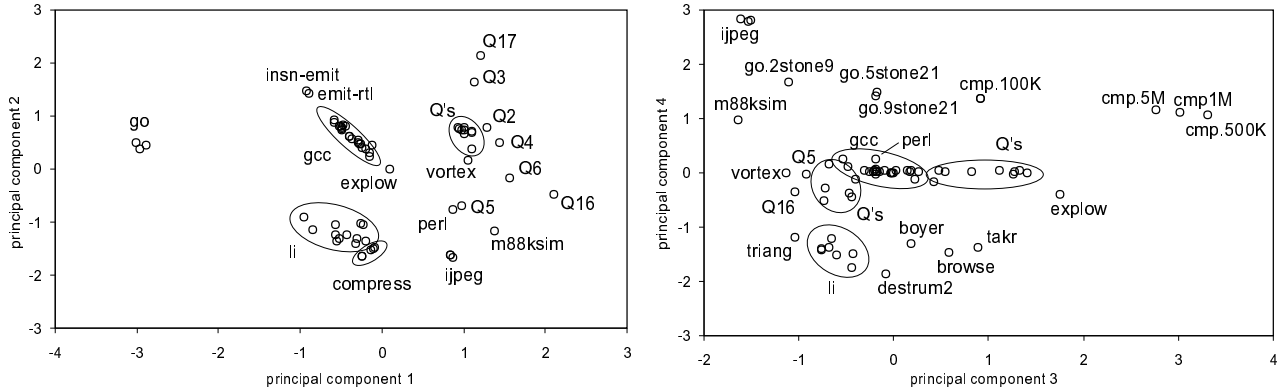
Figure 2 presents the various input sets of li in the 2-dimensional space built up by its two principal components. Five input sets result in a behaviour that is different from the other input sets located at the right of the graph. Three of these, namely takr, browse and boyer, have a higher miss rate for larger D-caches, a higher percentage of logical and shift operations, a lower percentage of load/store operations, a lower number of instructions between two taken branches and a lower miss rate for the 8KB I-cache. Two of these input sets, namely puzzle0 and triang, have a higher I-cache miss rate, a smaller miss rate for small D-caches, a higher percentage of arithmetic operations and a lower

amount of ILP.

**TPC-D.** PCA extracted three principal components from the data of postgres running the TPC-D queries, accounting for 90.2% of the total variance. After varimax rotation, the first component is positively dominated, see Table 2, by the amount of ILP, the percentage of arithmetic operations and the D-cache miss rate; and negatively dominated by the branch prediction accuracy of the gshare branch predictor and the percentage of logical operations. The second component is positively dominated by the I-cache miss rate and negatively dominated by the percentage of shift and byte manipulation operations. The third component is positively dominated by the number of instructions between two taken branches and negatively dominated by the branch prediction accuracy and the percentage of control operations.

Figure 3 shows the data points of postgres running the TPC-D queries in the 3-dimensional space built by the three (rotated) components. In this graph, there is a strong cluster that contains queries 11, 12, 13, 14 and 15. From this graph, we can also conclude that queries 5, 16 and 17 exhibit a significantly different behaviour than the other queries. Query 17 has significantly higher I-cache miss rates and a significantly lower percentage of shift operations. Queries 5 and 16 differ from each other due to a high and low number of instructions between two taken branches, respectively; and a low versus high branch prediction accuracy and percentage of control operations, respectively.

**Workload Space.** Performing PCA on all benchmark-input tuples as described in Table 1, yields four principal components accounting for 89.2% of the total variance. After varimax rotation, the first component is positively dominated, see Table 2, by the branch prediction accuracy and the percentage of logical operations. The second principal component is positively dominated by the I-cache miss rates. The third component is positively dominated by the D-cache miss rates. The fourth component is positively



**Figure 4. Workload space: first component vs. second component on the left and third vs. fourth component on the right.**

dominated by the percentage of arithmetic operations and the number of instructions between two taken branches; and negatively dominated by the percentage of control operations.

Figure 4 shows these benchmark-input tuples in the four-dimensional workload space built up by its principal components. The benchmarks `go`, `jpeg` and `compress` are somewhat isolated points in this 4-dimensional space. This is due to the high D-cache miss rates for `compress`, the high percentage of arithmetic operations, the low percentage of control operations and the high number of instructions between two taken branches for `jpeg` and the low branch prediction accuracy and the low percentage of logical operations for `go`.

It is also interesting to note that the data points corresponding to the `gcc` benchmark are strongly clustered, except for the input sets `emit-rtl`, `insn-emit` and `explore`. This suggests that for the cluster `gcc` only a small number of input sets should be selected for the workload to represent the `gcc` benchmark, ultimately reducing the total simulation time of `gcc` compared to the simulation of all the input sets.

The data points corresponding to the lisp interpreter are strongly clustered as well, except for the following five input sets: `triang`, `tahr`, `destrum2`, `browse` and `boyer`. The variety within these five input sets is caused by the data cache miss rate measured by the third principal component.

The data points corresponding to `postgres` running the TPC-D queries on the other hand, are weakly clustered except for queries 11 through 15 (not clearly seen on the graph). This suggests that for the TPC-D benchmark, several queries need to be considered in order to be representative. Note that all queries result in an above-average branch prediction accuracy (high value along the first principal component). The spread along the second principal component is very large and covers about 80% of the range

of the second component. Therefore, a wide range of different I-cache behaviour can be observed when running the TPC-D queries. When inspecting the third principal component, we see that the TPC-D queries fall apart into two groups: those with relatively high data cache miss rates and those with relatively low D-cache miss rates.

The difference in behaviour between the program-input tuples for `go` and `compress` is mainly due to the difference in the data cache miss rates. For `jpeg`, all three input sets seem to result in more or less the same behaviour.

In general, we can conclude that the variation between programs is larger than the variation between input sets for the same program. Thus, when composing a workload, it is more important to select different programs with a well chosen input set than to include various inputs for the same program. For example, the program-input tuples for `gcc` (except for `explore`) and `jpeg` are strongly clustered in the workload space. In some cases however, for example for `postgres` running different TPC-D queries, the input set has a relatively high impact on program behaviour.

## 5 Related work

KleinOowski *et al.* [14] propose to reduce the simulation time of the SPEC 2000 benchmark suite by using reduced input data sets. Instead of using the reference input data sets provided by SPEC, which result in unreasonably long simulation times, they propose to use smaller input data sets that accurately reflect the behaviour of the full reference input sets. For determining whether two input sets result in more or less the same behaviour, they used the chi-squared statistic based on the function-level execution profiles for each input set. Note that a resemblance of function-level execution profiles does not necessarily imply a resemblance of other program characteristics which are probably more di-

	li		gcc		TPC-D			all			
	PC1	PC2	PC1	PC2	PC1	PC2	PC3	PC1	PC2	PC3	PC4
ILP	0.11	<b>0.74</b>	0.43	<b>-0.70</b>	<b>0.87</b>	-0.13	0.33	-0.58	0.20	0.39	0.59
bimodal	0.38	-0.59	<b>0.94</b>	0.30	0.04	0.05	<b>-0.93</b>	<b>0.92</b>	0.20	-0.03	-0.18
gshare	0.44	0.09	<b>0.94</b>	0.22	<b>-0.70</b>	-0.10	-0.66	<b>0.78</b>	-0.35	0.01	-0.48
hybrid	0.67	-0.17	<b>0.95</b>	0.19	-0.45	0.07	<b>-0.83</b>	<b>0.80</b>	-0.07	-0.02	-0.56
ld/st	<b>0.91</b>	0.30	-0.58	<b>-0.80</b>	-0.59	0.14	0.66	-0.66	-0.34	-0.21	-0.58
int arithmetic	-0.25	<b>-0.96</b>	<b>0.94</b>	-0.06	<b>0.84</b>	0.48	0.10	-0.17	0.33	0.04	<b>0.81</b>
logical	<b>-0.84</b>	-0.41	<b>0.82</b>	0.12	<b>-0.90</b>	-0.39	-0.01	<b>0.95</b>	0.11	-0.08	0.01
shift	<b>-0.97</b>	0.17	-0.04	<b>0.86</b>	-0.04	<b>-0.72</b>	-0.47	0.52	-0.14	0.35	0.54
control	-0.23	0.64	0.24	<b>0.89</b>	-0.41	-0.32	<b>-0.80</b>	0.07	0.26	0.06	<b>-0.89</b>
seq. flow break	<b>0.82</b>	0.51	-0.14	<b>-0.95</b>	0.40	0.11	<b>0.89</b>	-0.26	-0.25	-0.12	<b>0.87</b>
I\$ 8KB	<b>0.80</b>	-0.51	<b>-0.72</b>	-0.61	-0.04	<b>0.96</b>	-0.13	0.24	<b>0.88</b>	-0.13	-0.26
I\$ 16KB	0.32	<b>-0.87</b>	<b>-0.76</b>	-0.55	-0.02	<b>0.98</b>	-0.13	0.28	<b>0.94</b>	-0.05	-0.01
I\$ 32KB	0.04	<b>-0.86</b>	<b>-0.88</b>	-0.41	0.20	<b>0.93</b>	0.06	0.11	<b>0.97</b>	0.04	0.03
I\$ 64KB	-0.05	<b>-0.92</b>	<b>-0.90</b>	-0.24	0.21	<b>0.86</b>	0.14	-0.08	<b>0.93</b>	0.00	0.05
I\$ 128KB	-0.04	<b>-0.92</b>	<b>-0.85</b>	-0.14	0.52	<b>0.70</b>	0.25	-0.27	<b>0.75</b>	-0.04	0.06
D\$ 8KB	0.09	0.63	0.27	<b>0.93</b>	0.67	0.66	0.27	-0.18	0.64	0.59	-0.08
D\$ 16KB	-0.46	<b>0.75</b>	0.43	<b>0.88</b>	<b>0.83</b>	0.37	0.28	-0.14	0.46	<b>0.84</b>	0.01
D\$ 32KB	<b>-0.93</b>	0.30	0.50	<b>0.86</b>	<b>0.95</b>	0.15	0.18	-0.04	-0.03	<b>0.99</b>	0.00
D\$ 64KB	<b>-0.94</b>	0.27	0.57	<b>0.80</b>	<b>0.96</b>	0.06	0.13	0.04	-0.17	<b>0.97</b>	0.01
D\$ 128KB	<b>-0.97</b>	0.17	0.63	<b>0.73</b>	<b>0.96</b>	0.00	0.10	0.12	-0.20	<b>0.95</b>	0.07

**Table 2. The factor loadings of the principal components after varimax rotation for li, gcc, postgres and all the benchmark-input tuples, from left to right respectively. For example, this means that the first principal component for li is defined as follows:  $PC1 = 0.11 \cdot ILP + 0.38 \cdot bimodal + 0.44 \cdot gshare + \dots$ . Factor loadings greater than 0.70 are shown in bold.**

rectly related to performance, such as instruction mix, cache behaviour, etc. The latter approach was taken in this paper for exactly that reason. KleinOowski *et al.* also recognized that this is a potential problem. The methodology presented in this paper can be used as well for selecting reduced input data sets. A reference input set and a resembling reduced input set will be situated close to each other in the  $q$ -dimensional space built up by the principal components.

Another important research topic that is related to this paper is trace sampling [5, 6, 13, 16]. In trace sampling, several samples are taken from a program execution so that the total number of instructions in the samples is significantly less than the total number of instructions of a complete execution. In order to make viable design decisions based on these sampled traces, a sampled trace should be representative for the complete program execution. Iyengar *et al.* [11] propose an R-metric for measuring the representativeness of a sampled trace. Lafage and Seznez [15] propose to choose representative samples using a data reduction technique, namely cluster analysis. The methodology presented here could also be used to validate sampled traces. Indeed, a sampled trace that is situated close to its reference trace in the workload space could be considered

as being representative.

Only recently, a new fast simulation technique was introduced, namely statistical simulation [3, 7, 8, 19, 21, 20]. In statistical simulation, a statistical profile is extracted from a program execution which is subsequently fed into a synthetic trace generator. The synthetic trace being generated can then be executed on a trace-driven simulator which yields performance estimates. Due to the statistical nature of the technique, the total number of instructions in a synthetic trace can be limited since the performance characteristics while simulating a synthetic trace quickly converge. Typically, no more than one million instructions need to be simulated to obtain a stable performance estimate. Statistical simulation is related to the research topic presented in this paper, since the success of both techniques relies on choosing relevant program characteristics to be incorporated in the analysis. For statistical simulation, relevant program characteristics are needed to obtain a high accuracy; for the technique presented in this paper, relevant program characteristics are needed to construct a reliable workload space.

Another possible application of using a data reduction technique such as principal components analysis, is to com-

pare different workloads. In [4], Chow *et al.* used PCA to compare the branch behaviour of Java and non-Java workloads. The interesting aspect of using PCA in this context is that PCA is able to identify on which point two workloads differ.

Huang and Shen [10] evaluated the impact of input data sets on the bandwidth requirements of computer programs.

Changes in program behaviour due to different input data sets are also important for profile-guided compilation [22], where profiling information from a past run is used by the compiler to guide its optimisations. Fisher and Freudenberger [9] studied whether branch directions from previous runs of a program (using different input sets) are good predictors of the branch directions in future runs. Their study concludes that branches generally take the same directions in different runs of a program. However, they warn that some runs of a program exercise entirely different parts of the program. Hence, these runs cannot be used to make predictions about each other. By using the average branch direction over a number of runs, this problem can be avoided. Wall [24] studied several types of profiles such as basic block counts and the number of references to global variables. He measured the usefulness of a profile as the speedup obtained when that profile is used in a profile-guided compiler optimisation. Seemingly, the best results are obtained when the same input is used for profiling and measuring the speedup. This implies that every input is different in some sense and leads to different compiler optimisations.

## 6 Conclusion

In microprocessor design, it is important to have a representative workload to make correct design decisions. This paper proposes the use of principal components analysis to efficiently explore the workload space. In this workload space, benchmark-input tuples can be displayed. This representation can be used to measure the impact of input data sets on program behaviour. An interesting application for this technique is the selection of representative input data sets. Indeed, simulating a single or only a few representative input sets per benchmark instead of simulating a large number of input sets ultimately reduces the total simulation time.

## Acknowledgements

Lieven Eeckhout and Hans Vandierendonck are supported by a grant from the Flemish Institute for the Promotion of the Scientific-Technological Research in the Industry (IWT).

## References

- [1] StatSoft, Inc. (1999). STATISTICA for Windows. Computer program manual. <http://www.statsoft.com>.
- [2] P. Bose and T. M. Conte. Performance analysis and its impact on design. *IEEE Computer*, 31(5):41–49, May 1998.
- [3] R. Carl and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Workshop on Performance Analysis and its Impact on Design*, June 1998.
- [4] K. Chow, A. Wright, and K. Lai. Characterization of Java workloads by principal components analysis and indirect branches. In *Proceedings of the Workshop on Workload Characterization (WWC-1998), held in conjunction with the 31st Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-31)*, pages 11–19, Nov. 1998.
- [5] 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)*, Oct. 1996.
- [6] P. K. Dubey and R. Nair. Profile-driven sampled trace generation. Technical Report RC 20041, IBM Research Division, T. J. Watson Research Center, Apr. 1995.
- [7] L. Eeckhout and K. De Bosschere. Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, pages 25–34, Sept. 2001.
- [8] L. Eeckhout, K. De Bosschere, and H. Neefs. Performance analysis through synthetic trace generation. In *The IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2000)*, pages 1–6, Apr. 2000.
- [9] J. Fisher and S. Freudenberger. Predicting conditional branch directions from previous runs of a program. In *Proc. of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 85–95, 1992.
- [10] A. S. Huang and J. P. Shen. The intrinsic bandwidth requirements of ordinary programs. In *Proc. of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 105–114, Oct. 1996.
- [11] V. S. Iyengar, L. H. Trevillyan, and P. Bose. Representative traces for processor models with infinite cache. In *Proceedings of the Second International Symposium on High-Performance Computer Architecture (HPCA-2)*, pages 62–73, Feb. 1996.
- [12] L. K. John, P. Vasudevan, and J. Sabarinathan. Workload characterization: Motivation, goals and methodology. In L. K. John and A. M. G. Maynard, editors, *Workload Characterization: Methodology and Case Studies*. IEEE Computer Society, 1999.
- [13] R. E. Kessler, M. D. Hill, and D. A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, 43(6):664–675, June 1994.
- [14] A. J. KleinOsowski, J. Flynn, N. Mearns, and D. J. Lilja. Adapting the SPEC 2000 benchmark suite for simulation-based computer architecture research. In *IEEE 3rd Annual Workshop on Workload Characterization (WWC-2000) held*

*in conjunction with the International Conference on Computer Design (ICCD-2000)*, Sept. 2000.

- [15] T. Lafage and A. Sez nec. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In *IEEE 3rd Annual Workshop on Workload Characterization (WWC-2000) held in conjunction with the International Conference on Computer Design (ICCD-2000)*, Sept. 2000.
- [16] G. Lauterbach. Accelerating architectural simulation by parallel execution of trace samples. Technical Report SMLI TR-93-22, Sun Microsystems Laboratories Inc., Dec. 1993.
- [17] B. F. J. Manly. *Multivariate Statistical Methods: A primer*. Chapman & Hall, second edition, 1994.
- [18] S. McFarling. Combining branch predictors. Technical Report WRL TN-36, Digital Western Research Laboratory, June 1993.
- [19] D. B. Noonburg and J. P. Shen. A framework for statistical modeling of superscalar processor performance. In *Proceedings of the third International Symposium on High-Performance Computer Architecture (HPCA-3)*, pages 298–309, Feb. 1997.
- [20] S. Nussbaum and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, pages 15–24, Sept. 2001.
- [21] M. Oskin, F. T. Chong, and M. Farrens. HLS: Combining statistical and symbolic simulation to guide microprocessor design. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, pages 71–82, June 2000.
- [22] M. Smith. Overcoming the challenges to feedback-directed optimization (keynote talk). In *Proc. of ACM SIGPLAN workshop on Dynamic and adaptive compilation and optimization*, pages 1–11, 2000.
- [23] A. Srivastava and A. Eustace. ATOM: A system for building customized program analysis tools. Technical Report 94/2, Western Research Lab, Compaq, Mar. 1994.
- [24] D. Wall. Predicting program behavior using real or estimated profiles. In *Proceedings of the 1991 International Conference on Programming Language Design and Implementation (PLDI-1991)*, pages 59–70, 1991.