On the Use of Statistical Data Analysis Techniques in Workload Characterization

Hans Vandierendonck Koen De Bosschere

Abstract— **Workload characterization is concerned with the characterization of workloads in terms of abstract metrics, called workload characteristics. Many workload characteristics have to be measured in order to accurately model workload behavior. Furthermore, these characteristics are typically strongly correlated. It is therefore useful to use statistical data analysis techniques when comparing the workload characteristics between different programs and their inputs. One such technique, principal components analysis, is applied to several important problems in workload characterization.**

Keywords— **Workload characterization, principal components analysis**

I. Introduction

COMPUTER architectures are evaluated by running a workload on the computer and measuring its execution time. New computers are designed in the same way. However, as the computer does not exist yet, it is not possible to execute the workload. This is where workload characterization enters the picture. The goal of workload characterization is to describe the properties of a workload (its behavior) in terms of abstract performance metrics, called workload characteristics, that predict the final performance [1].

There are, however, many aspects to the behavior of a workload and each of these aspects must be adequately characterized. These aspects include the data and instruction memory access patterns, the predictability of branch instructions, the amount of instruction level parallelism, the types of ALU operations, etc. To complicate matters, the workload characteristics are strongly correlated.

For this reason, we use statistical data analysis techniques to (i) reduce the number of workload characteristics and (ii) remove the correlations. This paper uses principal components analysis [2], although there exist many other techniques that are equally applicable. Three applications of principal components analysis to workload characterization are discussed. But first, principal components analysis is explained.

II. Principal Components Analysis

Principal Components Analysis (PCA) is a statistical data analysis technique [2]. It transforms a set of p variables X_i (the workload characteristics) into a set of p new variables Z_j, in such a way that (i) the Z_j's are sorted with decreasing variance and (ii) the Z_j's are not correlated. The Z_j's are linear combinations of the X_i and they are called the principal components (PC).

Hans Vandierendonck is with the Department of Electronics and Information Systems (ELIS), Ghent University, Gent, Belgium. E-mail: hans.vandierendonck@UGent.be .

It is possible to reduce the number of characteristics by retaining only the first $q \ll p$ of the principal components and neglecting the others. When a proper choice of q is made, then the q variables still contain most of the information of the complete data set. A disadvantage of this method is that it may be very difficult to interpret the principal components, which is necessary to understand the differences in the behavior of programs.

III. Running Example

This section describes the analysis of the data memory behavior of the SPEC CPU95 and CPU2000 benchmark suites. This analysis is used during the remainder of this paper to illustrate the possible applications of PCA to workload characterization.

The data memory behavior is characterized by means of the data cache miss rate measured for various cache configurations. This data is cleaned by transforming it into ratios of miss rates, such that each of the 42 workload characteristics measures the impact of changing one specific cache parameter. The cache parameters varied in this study are the cache size, cache block size, associativity and the set index function. By varying the set index function between the baseline modulo indexing and a better XOR-based hash function, it is possible to measure whether a benchmark generates very regular and repetitive conflict misses, which can be typically removed using software techniques (e.g., code restructuring and data layout optimizations).

The result of applying PCA to the 42 workload characteristics is shown in Figure 1. Only the 3rd and 4th principal components are shown due to space limitations. The interpretation of these principal components is such that benchmarks with many repetitive conflict misses are located to the left of the graph, benchmarks that require high associativity when the block size is large are located near the top and benchmarks that require high associativity when the block size is small are located near the bottom of the plot.

IV. Applications

There are many problems in workload characterization and performance evaluation in general to which PCA can be usefully applied. Besides the topics discussed below, data analysis techniques have been applied to, e.g., selecting short traces of a benchmark that have the same behavior as the full benchmark [3] and reducing the size of a benchmark suite by pruning similar benchmarks [4].

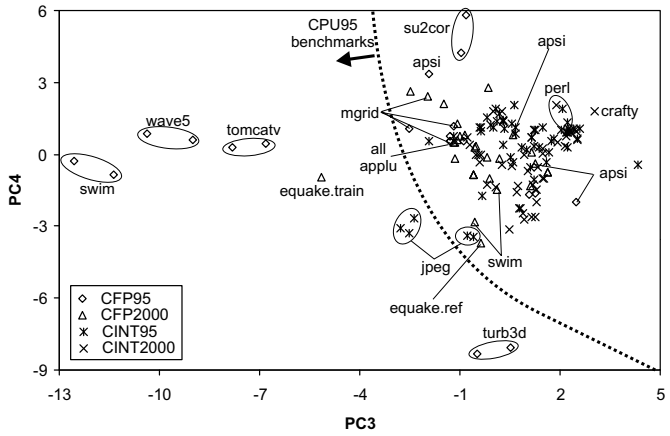

Fig. 1. Scatter plot of the 3rd and 4th principal components. Each point shows the values of PC3 and PC4 for a program and its input. The "FP" and "INT" labels signify sub-suites of CPU95 and CPU2000.

A. The Effect of Inputs on Program Behavior

The behavior of some programs is strongly effected by its input. E.g., when compiling a C program, the code constructs used in the program determine which code fragments in the compiler are activated. As such, the input effects the behavior of the compiler [5].

From the scatter plot (Figure 1), one can easily see how the input effects the behavior of the program. When the program-input pairs for the same program are clustered together, then the input has a minor effect on the program behavior. E.g., the inputs to *swim*, *wave5* and *tomcatv* in the left half plane, *su2cor* at the top right and *applu* near (-1,1) are clustered, so the inputs have little influence on program behavior. In contrast, the train and reference inputs for *equake* radically change its behavior.

Applications include: selecting inputs for profile-guided compiler optimizations and composing a workload for simulation purposes [5], [6] as well as validating shortened inputs (i.e., inputs that have been trimmed in order to limit execution times) [7].

B. Comparing Workloads

The *Standard Performance Evaluation Corporation* (SPEC) constructs benchmark suites with the goal of making objective performance comparisons. The CPU benchmark suites measure the combined performance of the processor, the memory hierarchy and the compiler. As computers become larger and more powerful, so do the programs we wish to run on them. Therefore, SPEC renews the CPU suite every 3 to 4 years.

The behavioral differences between successive versions of the CPU suite can be judged from the scatter plot. The most obvious change in behavior is the importance of repetitive conflict misses: CPU95 has several benchmarks which are very prone to this type of misses, while this program property is almost absent in CPU2000. It can be observed that the left half of the scatter plot contains only CPU95 benchmarks, except for *equake.train* (Figure 1). This area is marked by the dashed line.

C. Eccentric Benchmarks

The example also illustrates that some benchmarks have a notably different behavior as the majority of the benchmarks. Such benchmarks are called *eccentric* benchmarks. They are excellent candidates for a case study, as they stress some behavioral property to a large extent. Such benchmarks are also good candidates to include in a benchmark suite, as they cover behavioral types that are not present in other benchmarks.

Eccentric benchmarks can also be *erratic*, i.e., they have a different behavior but this behavior can be fixed by relatively simple optimizations of the source code. Erratic benchmarks are not useful, as they measure the performance of a system on a badly coded program. If one wants high performance, then one must invest an effort too and optimize the program for the machine at hand.

The *swim* benchmark in CPU95 shows that erratic benchmarks do occur in practice. The CPU95 version of swim is located at the left of the scatter plot and is eccentric: there are few benchmarks that are so sensitive to repetitive conflict misses. On the other hand, the CPU2000 version of *swim* is near (0,0), which means it is an average benchmark. Thus, the behavior of CPU95's *swim* can be fixed, meaning that it is an erratic benchmark. *Wave5* and *tomcatv* are also erratic benchmarks.

V. Conclusion

Workload characterization is concerned with the characterization of workload performance in terms of abstract metrics. The application of principal components analysis to workload characterization problems is discussed. It is applied to gauge the effect of an input on program behavior, to detect behavioral differences between benchmark suites and to detect eccentric benchmarks.

Acknowledgments

Hans Vandierendonck is supported by the Flemish Institute for the Promotion of Scientific-Technological Research in the Industry (IWT).

References

[1] T. M. Conte and W.-m. W. Hwu, "Benchmark characterization," *IEEE Computer*, vol. 24, no. 1, pp. 48–56, Jan. 1991.

[2] G. H. Dunteman, *Principal Components Analysis*, SAGE Publications, 1989.

[3] T. Lafage and A. Seznec, "Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream," in *Workshop on Workload Characterisation (WWC-2000)*, Sept. 2000.

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

[5] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Workload design: Selecting representative program-input pairs," in *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, 9 2002, pp. 83–94.

[6] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Quantifying the impact of input data sets on program behavior and its applications," *Journal of Instruction-Level Parallelism*, vol. 5, pp. 1–33, 2 2003.

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