Subsetting the SPEC CPU2006 Benchmark Suite

Aashish Phansalkar, Ajay Joshi and Lizy K. John
Department of ECE, University of Texas at Austin.
{aashish, ajoshi, ljohn}@ece.utexas.edu

Abstract

On August 24, 2006, the Standard Performance Evaluation Corporation (SPEC) announced CPU2006 – the next generation of industry-standardized CPU-intensive benchmark suite. The SPEC CPU benchmark suite has become the most frequently used suite for simulation-based computer architecture research. Detailed processor simulators take days to weeks to simulate each of the SPEC CPU programs. In order to reduce simulation to a tractable time, architects and researchers often use only a subset of benchmarks from the SPEC CPU suite to evaluate the potential of their ideas. Prior research has demonstrated that statistical techniques are most effective to find a representative subset of benchmark programs from a benchmark suite. The objective of this paper is to apply multivariate statistical data analysis techniques for selecting a representative subset of programs from the SPEC CPU2006 benchmark suite. We measure a set of performance counter based characteristics for the SPEC CPU2006 programs across a large number of architectures and apply multivariate statistical analysis techniques to find a representative subset of benchmarks and representative input sets wherever multiple input sets are provided. The results from this paper will help architects and researchers to find a smaller but representative set of programs from the SPEC CPU2006 benchmark suite, when time or resource constraints prohibit experimentation with the entire benchmark suite.

1. Introduction

SPEC, since its formation in 1988, has made many contributions in developing and distributing technically credible, portable, real-world application-based benchmarks for computer designers, architects, and consumers. SPEC CPU benchmark suite comprises of compute-intensive floating point and integer programs for measuring the performance of a computer's processor, memory, and compiler. In order to keep pace with the technological advancements, compiler improvements, and emerging workloads, new programs were added, programs that are susceptible to unfair compiler optimizations were removed, program run times were increased, and memory access intensity of programs was increased in every generation of the benchmark suite. The SPEC CPU benchmark suite, which was first released in 1989 as a collection of ten computation-intensive benchmark programs, is now in its fifth generation and has grown to 29 benchmarks. The SPEC CPU benchmark suite is primarily used by computer system manufacturers to measure and report performance of their computer systems and by microprocessor designers and researchers for evaluating design trade-offs and novel design ideas. Customers typically use the results reported to SPEC to make purchasing decisions.

As computer systems get faster the benchmark run times are increased across generations, to ensure that the benchmarks run for a long enough time to make meaningful measurements. However, longer-running benchmarks significantly increase the cost of performance evaluation, more so when simulating them on cycle accurate simulators. The response of researchers to this has been two fold – develop intelligent techniques to reduce simulation time of each benchmark and find a smaller but representative set of benchmark programs. In this paper we focus on the second approach.

A poorly chosen set of benchmark programs may not accurately depict the true performance of the processor design. On one hand, selecting too few benchmarks may not cover the entire spectrum of applications that may be executed on a computer; while on the other hand, selecting too many similar programs will increase evaluation time without providing additional information. Therefore, in order to reduce the benchmarking effort, a good workload should have programs that are well distributed within the target workload space (comprising of N axes, each representing a program characteristic) without being clustered in specific areas. Understanding similarity between programs can help in selecting benchmark programs that are distinct, but are still representative of the target workload space.

In this paper we use multivariate statistical analysis techniques such as Principal Components Analysis (PCA) and Cluster Analysis to find a subset of CPU2006 programs that is representative of the whole suite. This is a quantitative approach to selecting benchmarks based on the diversity of their characteristics. There are many workload characteristics that affect performance. It is difficult to visualize the workload space formed by these characteristics. Multivariate statistical analysis techniques help us to visualize the workload space and find representative benchmarks. The method has two main steps: The first step involves selecting the important characteristics of programs that form the workload space; in the second step we use the characteristics of all programs to perform PCA

[7] and Cluster Analysis to find a subset of programs. We will discuss each of these steps in detail in the subsequent sections. The characteristics that are used in this paper are derived from the performance counter measurements of a wide range of state-of-the-art processors. A set of workload characteristics that cover a wide enough range of the program characteristics are included to make a meaningful comparison between the programs. As the programs in SPEC CPU2006 suite are very long (some of them in the range of few trillion instructions), the cost of measurement significantly increases if we use simulation methodology. Therefore, we use performance counter based characteristics such as cache miss-rates, branch mispredictions per instruction etc. More details on the different characteristics of programs and the method used to choose them are described in section 2.

The new suite is larger than the previous, and will exercise new corners of CPUs, memory systems, and compilers – especially C++ compilers. Where CPU2000 had only 1 benchmark in C++, the new suite has 7, including one with ½ million lines of C++ code. As in the previous CPU suites, Fortran and C languages are also well represented.

2. Methodology

The SPEC CPU2006 suite, like its predecessors is divided into two parts: the integer component (CINT2006 benchmarks) and the floating point component (CFP2006 benchmarks). Since the results are reported separately for these two groups of programs we perform the analysis and find a subset for each of these groups separately. Measuring characteristics that are used to find similarity between benchmark programs is a very important part of the methodology. Many researchers have proposed different ways to measure these characteristics. The characteristics can be broadly classified into microarchitecture-dependent and microarchitecture-independent characteristics [4]. The microarchitecture-independent characteristics measure inherent characteristics of program at the level of Instruction Set Architecture (ISA), or at source code level. The microarchitecture-dependent characteristics such as cache miss-rate and branch misprediction rate are characteristics of a program that are dependent on the hardware configurations. These characteristics depend on the underlying microarchitecture and therefore can vary with machine configurations. Both the microarchitecture-independent and microarchitecture-dependent characteristics are dependent on the compiler and the ISA. Phansalkar *et al.*[4] show how microarchitecture-independent characteristics at ISA level can be used to categorize programs.

During the process of benchmark selection for SPEC CPU2006, benchmark similarity was used as one of the many criteria. However, the time required to measure the microarchitecture-independent characteristics proposed by Phansalkar *et al.*[4] is very high and hence cannot be completed in the short span of time. Therefore, we used an alternative approach of measuring the important characteristics using hardware performance monitoring counters from a wide range of architectures. A potential problem of using this method is that the characteristics that have lesser impact on performance nay get equal weight as the characteristics that show significant performance impact. In order to reduce such undesirable effects, four different SPEC members performed a correlation analysis of Cycles-Per-Instruction (CPI) with various characteristics on their respective machines. A common set of characteristics were chosen based on such a correlation analysis, and the experience of the hardware designers and the compiler experts. Since the different characteristics are measured on different machines, each of them forms a characteristic of a program. If we have *n* machines and we measure *m* metrics for each machine, we have *n x m* characteristics for each program. We then perform statistical analysis such as PCA and cluster analysis for these *n x m* characteristics for all programs. It is likely that some of these characteristics are correlated (for instance, consider that 2 machines have very similar microarchitectures). This correlation will be removed by PCA.

A weakness of using microarchitecture-dependent measurements is the bias towards the architecture of a particular system. However, we try to eliminate that bias by using measurements from multiple state-of-the-art computer systems. These systems have varying microarchitecture (including dynamic and statically scheduled architecture), varying degrees of out-of-order instruction issue, varying cache sizes and different cache hierarchy structures. Several different commercial-grade compilers are also used, in order to reduce the sensitivity to compiler differences. We would also like to point out that in addition to the characteristics shown in `Table` 1, new characteristics may be required to represent features in future systems that significantly affect performance. In summary, we hope that the variability in the microarchitectures/ISAs/compilers, have resulted in capturing most of the differences between the benchmarks.

`Table` 1: List of program characteristics for Integer and FP benchmarks

Integer benchmarks	**Floating point benchmarks**
Integer operations per instruction	Floating point ops per instruction
L1 I- cache misses per instruction	Memory references per instruction
Number of branches per instruction	L2 D-cache misses per instruction
Number of mispredicted branches per instruction	L2 data cache misses per L2 Accesses
L2 data cache misses per instruction	Data TLB misses per instruction
ITLB misses per instruction	L1 data cache misses per instruction

Table 1 shows the characteristics that were measured for each program on five different computer systems (four different ISAs) from SPEC member companies. Note that the important characteristics that affect performance for the CINT and CFP benchmarks are different. As mentioned before, these characteristics that form the workload space were based on the experience of hardware architects, compiler designers, and the correlation analysis from four different SPEC affiliated companies. In general, the characteristics that were chosen are also the ones that are well correlated to performance.

2.1 Principal Components Analysis

The measured data consists of a large number of variables - each performance counter characteristic for each machine is considered a separate variable. Considering the six characteristics measured on each of the five different systems, we have thirty characteristics per program. It is humanly impossible to simultaneously look at all the data and draw meaningful conclusions from them. Hence a multivariate statistical data analysis technique, namely *Principal Component Analysis* [7] is used to analyze the data. In order to isolate the effect of varying ranges of each parameter, the data is first normalized i.e. translated to zero mean and standard deviation equal to one, for each variable. This technique also allowed us to work with the SPEC member companies without disclosing the raw data about their individual state of the art machines. PCA is a classic multivariate statistical data analysis technique that helps to reduce the dimensionality of a data set while retaining most of the original information. PCA computes new variables, so called principal components (PCs), which are linear combinations of the original variables, such that all the PCs are uncorrelated. PCA transforms p variables X_1, X_2,...., X_p into p PCs $Z_1,Z_2,\ldots,Z_p$ such that:

$$Z_i = \sum_{j=0}^{p} a_{ij} X_j$$

This transformation has the property $Var[Z_1] \geq Var[Z_2] \geq \ldots \geq Var[Z_p]$ which means that Z_1 contains the most information and Z_p the least. Given this property of decreasing variance of the PCs, we can remove the components with the lower values of variance from the analysis. This reduces the dimensionality of the data set while controlling the amount of information that is lost. We use a standard technique to choose PCs where only the top few PCs which have eigen-values greater than or equal to one are retained. In order to capture the entire information from p original variables, p PCs may be required, but if there are several correlated variables, a small number of PCs can capture most of the information from the original variables. This results in reduced dimensionality.

2.2 Cluster Analysis

Clustering is a statistical technique that groups programs with similar features. There are two commonly used clustering techniques called the k-means clustering and hierarchical clustering. In this paper we use hierarchical clustering since hierarchical clustering and dendrogram helps us to look at multiple clustering possibilities at the same time and the user can pick the number of clusters to be formed. Hierarchical clustering is a technique for finding relatively homogeneous clusters of items based on their measured characteristics. Given a set of N programs to be clustered and an N x N similarity matrix containing the distance between the programs using the measured workload characteristics, the hierarchical technique starts with each case (benchmark or program-input pair in our case) in a separate cluster and then combines the clusters sequentially, merging the clusters at each step until all cases merge to form one cluster. When there are N cases, this involves N-1 clustering steps, or fusions. The algorithm used for hierarchical clustering is as follows:

1. Each program is assigned to its own cluster, such that if we have N programs, we now have N clusters, each containing just one program. The distances (similarities) between the clusters equal the distances (similarities) between the programs they contain (we used complete linkage, which means that the distance between two clusters is the distance between the two farthest points in the clusters).
2. Find the closest (most similar) pair of clusters and merge them into a single cluster, so that we have one less cluster.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N.

This hierarchical clustering process can be represented as a tree or dendrogram, where each step in the clustering process is illustrated by a joint in the tree. The numbered scale corresponds to the linkage distance obtained from the hierarchical cluster analysis.

3. Subsetting the CPU2006 suite

Many CPU2006 benchmarks execute trillions of instructions, and the time required to simulate them on cycle-accurate simulators is prohibitively high. This section demonstrates the result of applying PCA and cluster analysis for selecting a subset of benchmark programs when an architect or researcher is constrained and wants to select a reduced subset of programs from the suite. The Euclidean distance between the benchmarks is used as a measure of dissimilarity and complete-linkage distance is computed to create a dendrogram as shown in Figure 1. Eight PCs are chosen which retain 97% of the variance. The dimensionality of the data is reduced from thirty (six characteristics for five machines each) to eight. This shows that there were many variables that were correlated. The dendrogram can be used by researchers to select benchmarks by using a simple technique discussed below.

In the dendrogram in Figure 1 for CINT2006, the horizontal axis shows the linkage distance indicating the dissimilarity between the benchmarks. The ordering on the y-axis

does not have particular significance, except that similar benchmarks appear together. Depending on the number of representative benchmarks to be chosen, and the degree of similarity desired, one can draw a vertical line at a location and read from there how many benchmarks are in a subset. For example, if a vertical line is drawn at linkage distance of 4.5 we get 5 clusters *(k=5)* and at about 4.3 we get 6 clusters *(k=6)*. Table 2 has two rows with each showing the representative set of benchmarks for *k=5* and *k=6*. All the benchmarks which join to form a single horizontal line that is intersected by the vertical line form a cluster. The benchmark closest to the center of the cluster is identified from the distance matrix of all the benchmarks and chosen as the representative. As we travel from right to left on the dendrogram the number of benchmarks in a subset increases. This helps the user to select appropriate benchmarks depending on the desired number of benchmarks. When forming a subset of 6 benchmarks, *429.mcf* gets added to the subset of 5 benchmarks. In both the subsets of six and five the first cluster is represented by *458.sjeng*. The distance of each of the benchmarks in the cluster to the centre of its own cluster has to be recalculated and a representative has to be chosen again as we move left but in this case *458.sjeng* remains the representative. In the next section we compare the two subsets shown in Table 2 and validate their ability to predict the average speedup of all benchmarks on 5 different machines.

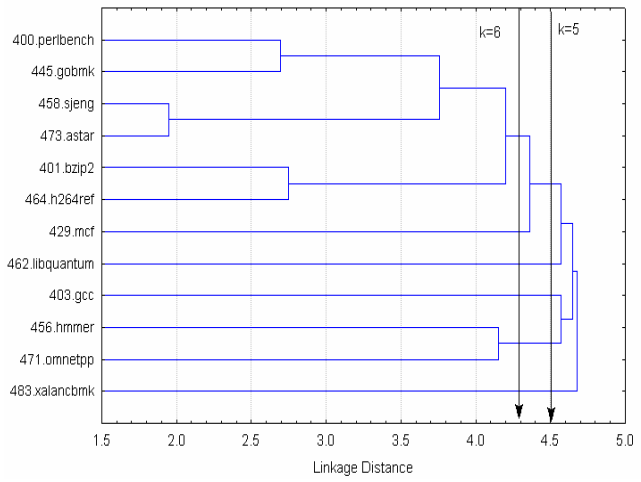

Figure 1: Dendrogram for CINT2006 benchmarks

Table 2: Subsets of CINT2006 benchmarks

Subset of 5	458.sjeng, 462.libquantum, 403.gcc, 456.hmmer, 483.xalancbmk
Subset of 6	458.sjeng, 429.mcf, 462.libquantum, 403.gcc, 456.hmmer, 483.xalancbmk

Figure 2 shows a dendrogram for the floating point benchmarks in CPU2006 suite. Seven PCs were retained after performing PCA. Similar to Figure 1, Figure2 also demonstrates the formation of a subset of 4 and 6 benchmarks by drawing vertical lines on the dendrogram at linkage distance of 5.25 and 4.5 respectively. The subsets are shown in Table 3. In the next section we validate the representativeness of the subset of benchmarks.

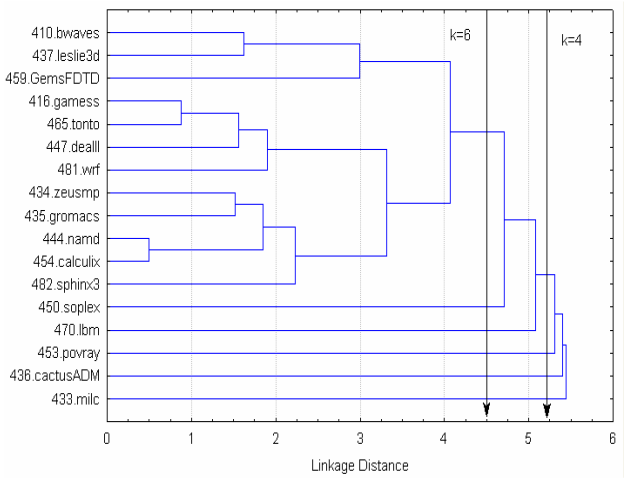

Figure 2: Dendrogram for CFP2006 benchmarks

Table 3: Subset of CFP2006 benchmarks

Subset of 4	481.wrf, 253.povray, 436.cactusADM, 433.milc
Subset of 6	481.wrf, 450.soplex, 470.lbm, 453.povray, 436.cactusADM, 433.milc

4. Validation of the subsets

The subsets that were generated in the previous section are validated in this section using a carefully chosen set of 5 different machines from the SPEC website [12]. The aim of this validation experiment is to see how well the subset of *k* benchmarks predicts the average speedup on the five machines. Figure 3 shows three bars for each machine. The first two bars for each machine are the weighted geometric mean of speedup numbers for the subsets with 5 and 6 benchmarks respectively. The third bar is the average speedup of all benchmarks measured on that machine with respect to the SPEC standard baseline machine. These numbers are obtained from the SPEC website where the scores are reported by different members. We can see that the difference in errors between the two subsets is not very significant. Figure 4 shows a similar validation experiment for the floating point benchmarks. This validation experiment shows the error that will be seen if only a subset of benchmarks is used as compared to the whole suite. Also, the errors are within reasonable limits so that they do not change the ranking of these five machines in almost all the cases.

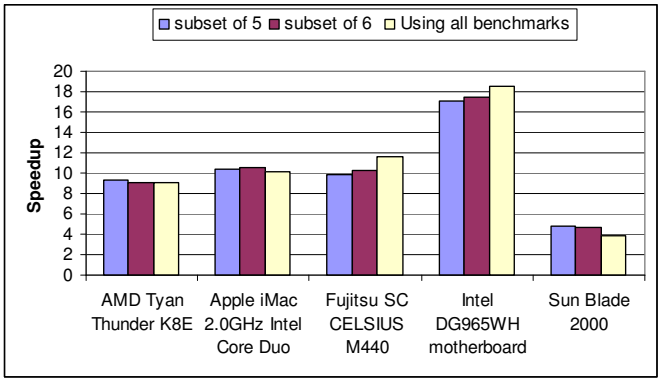

Figure 3: Validation of subsets of integer benchmarks using speedup with respect to SPEC base machine

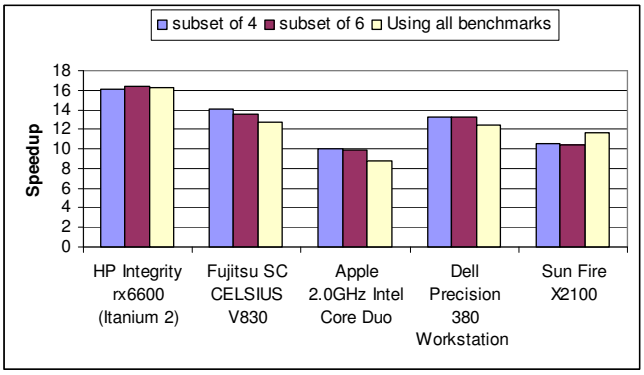

Figure 4: Validation of subsets of floating point benchmarks using speedup with respect to SPEC base machine

5. Selecting representative input sets

Many benchmarks in the CPU2006 have multiple input sets called workloads. A reportable SPEC run uses all the workloads for each benchmark; however it is possible to use PCA and clustering to identify a representative input set, helping architecture researchers to reduce simulation time/effort. Henceforth we will refer to a program input pair as a workload. In SPEC CPU2006 *403.gcc* benchmark has nine workloads. The program characteristics shown in Table 1 were measured for all the different workloads separately and for the whole benchmark. Whenever a performance data is reported for a benchmark, it is the aggregate behavior summing up all its workloads. The benchmark suffix shows the workload and the benchmark name without a suffix signifies the benchmark (aggregating all inputs).

A benchmark's workload closest to the whole benchmark in the workload space is marked as the representative workload. This can be done by looking up in the distance matrix. In CINT2006, the benchmarks that have multiple input sets are *400.perlbench, 401.bzip2, 403.gcc, 445.gobmk, 456.hmmer, 464.h264ref, 473.astar.* For each of these benchmarks we list a representative workload. Table 4 shows the list. We also do similar analysis for CFP2006. In this category there are only two benchmarks with multiple workloads. *i.e. 416.gamess* and *450.soplex.* For each of these benchmarks their representative workloads are listed in Table 4.

Table 4: List of representative input sets for SPEC CPU2006

CINT2006 benchmarks	464.h264avc - input set 2
400.perlbench - input set 1	473.astar - input set 2
401.bzip2 - input set 4	
403.gcc - input set 1	**CFP2006 benchmarks**
445.gobmk - input set 5	416.gamess - input set 3
456.hmmer - input set 2	450.soplex - input set 1

6. Subsets based on branch and memory access characteristics

Often, researchers focus on optimizing the design to take advantage of certain characteristics of programs. In this section, we look at the data access performance characteristics and branch prediction characteristics separately to identify unique programs from these perspectives. These subsets will be useful for data cache and branch predictor related stuides. When performing this analysis we put CINT and CFP workloads in the same pool *i.e.* we do not evaluate them as two separate components.

Figures 5 and 6 show the scatter plot and dendrogram respectively, based on the first 2 PCs of the branch characteristics, covering approximately 92% of the variance. The scatterplots of PC2 vs. PC1 can be easily used to visually identify the clusters as the benchmarks are plotted directly in the transformed workload space. The CFP benchmarks are clustered together but the CINT programs clearly show the diversity. A few CINT workloads overlap with the cluster of CFP workloads on the right side of the plot. Two CINT benchmarks *464.h264ref* and *456.hmmer* have branch characteristics that are not very different from the CFP programs. Majority of the floating point programs have very little diversity in their branch characteristics. The benchmarks with high or positive value of PC1, e.g. majority of CFP benchmarks and *464.h264ref* and *456.hmmer* show less misprediction rates. On the other hand CINT benchmarks are more spread out in the space and show significantly diverse behavior. CINT benchmarks like *458.sjeng* and *445.gobmk* lie on the left side of the plot and hence show much higher misprediction rate. The dendrogram shown in the Figure 6 below can be used to select benchmarks with diverse branch characteristics. A similar technique of drawing a line intersecting the axis showing the linkage distance can be used to find a subset. The benchmarks seen clustered in Figure 5 are also seen to be close in the dendrogram.

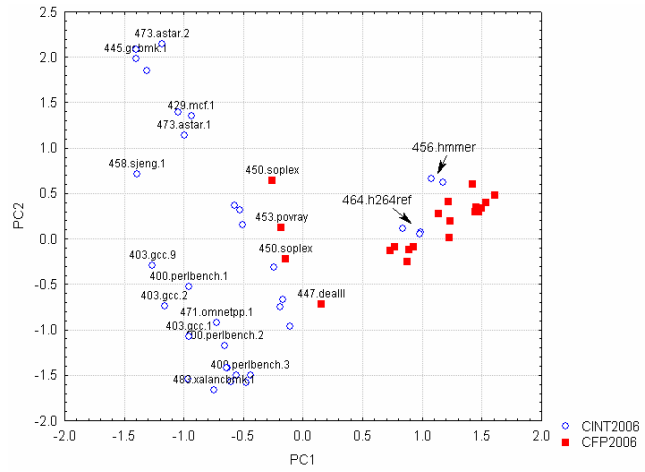

Figure 5: The CINT and CFP programs plotted in the PC space using only branch predictor characteristics

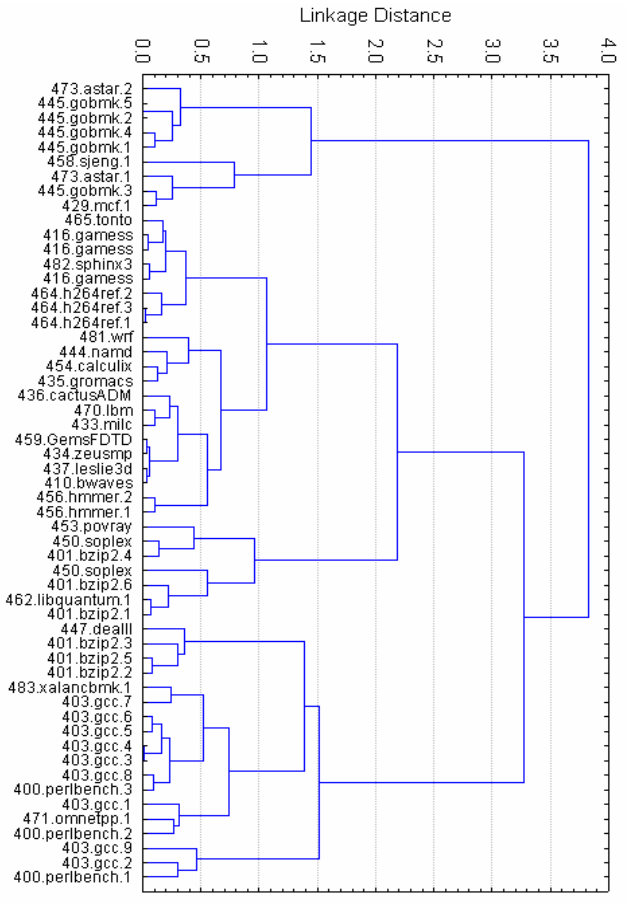

Figure 6: Dendrogram of CINT and CFP benchmarks based on branch characteristics only

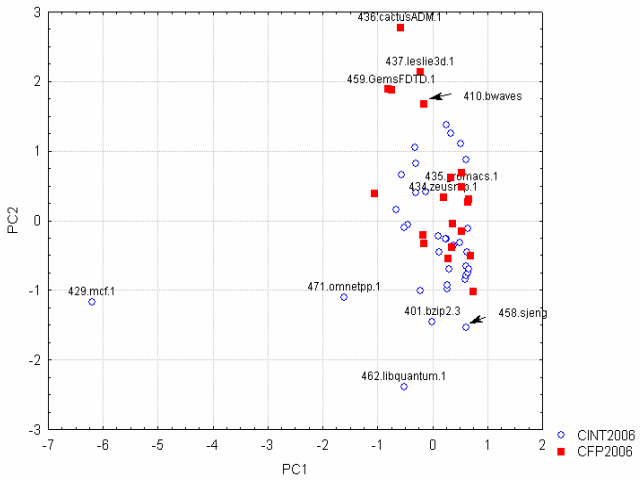

Figure 7: The CINT and CFP programs plotted in the PC space using only data access characteristics (PC2 Vs PC1)

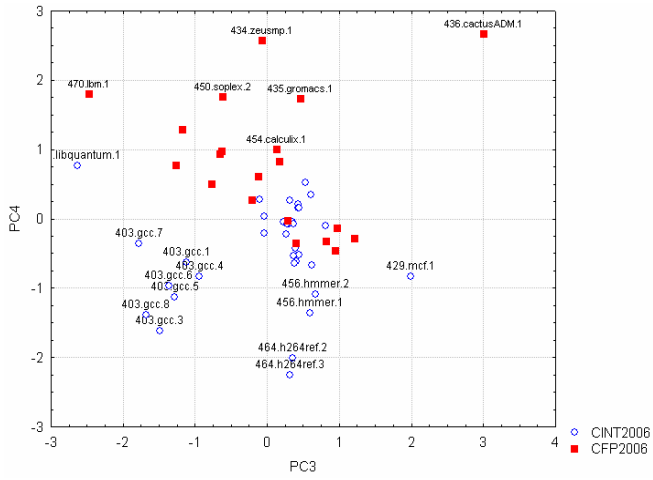

Figure 8: The CINT and CFP programs plotted in the PC space using only data access characteristics (PC4 Vs PC3)

In Figure 7 the benchmarks that show negative value of PC1 *e.g. 429.mcf, 471.omnettp, 462.libquantum* show higher miss-rates. The CFP benchmarks that are located at the top show very high percentage of memory accesses and hence should also be considered while picking benchmarks for cache studies. In Figure 8, benchmarks that show a positive value of PC3 and PC4 show a higher DTLB misses and percentage of memory accesses. The scatter plots can be used to visually identify the similarity between benchmarks but the dendrogram should be used to pick a subset of benchmarks in the same way as shown in Figure 1. Figure 9 shows the dendrogram of CINT and CFP benchmarks based on only the data accesses behavior. This can be used by researcher to find a subset of diverse benchmarks for cache studies.

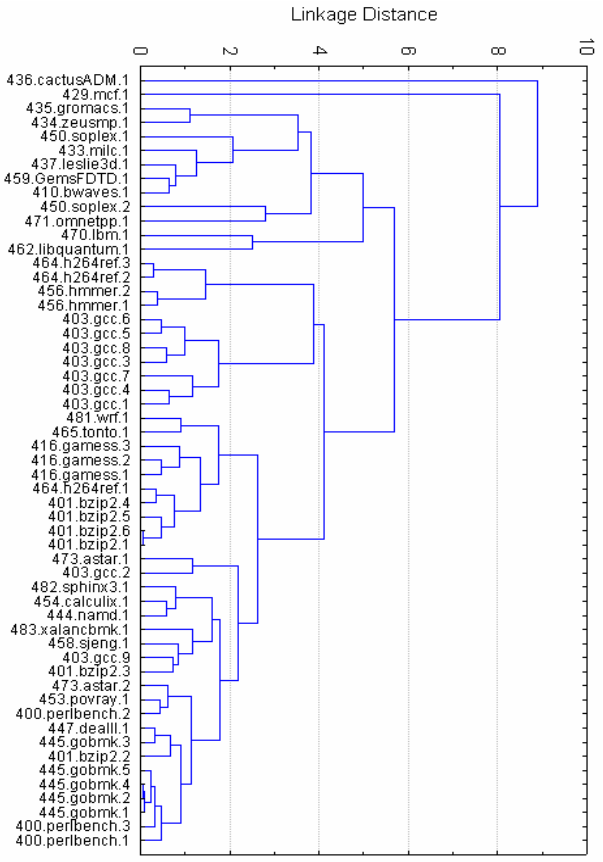

Figure 9: Dendrogram for CINT2006 and CFP2006 benchmarks based on data access characteristics only

7. Related Work

Eeckhout *et.al.* [1,2] measure many different workload characteristics such as instruction mix, branch prediction accuracy, the data and instruction cache miss rates, average number of instructions in a basic block, maximum amount of parallelism inherent to programs. In their work [1,2], two programs are considered to be similar if they show similar workload characteristics and hence lie close to each other in the workload space built from these characteristics. The authors use PCA and Cluster Analysis to find similar programs. The characteristics chosen are a mix of microarchitecture-dependent and microarchitecture-independent metrics.

Vandierendonck and Bosschere [3] primarily used microarchitecture-dependent metrics such as execution time or SPEC CPU peak performance rating for classifying programs. The technique aims at finding redundant programs which are inherently similar to the others and do not add any information while using benchmarks to rank machines. Vandierendonck and Bosschere [3] analyzed the SPEC CPU2000 benchmark suite peak results on 340 different machines representing eight architectures, and used PCA to identify the redundancy in the benchmark suite. In [3], the authors quantify redundancy as the inability of a program to show different speedup on two different machines. The programs that do not show very different speedups are considered redundant. In other words [3] concludes that we do not need such redundant programs to rank the predecided 340 machines. The main drawback of that method is that it cannot be used if the performance scores for many machines are not available especially at the time of designing a benchmark suite. Giladi and Ahituv [6] also had a similar approach towards finding a subset of programs. They found that the ten programs of SPEC89 suite could be reduced to six without affecting the SPEC rating. They also found that for SPECint92, three instead of six programs are sufficient and for SPECfp92 seven out of fourteen are enough to get a similar SPEC rating on all the machines at that time.

Phansalkar *et.al.* [4] characterized benchmarks using microarchitecture-independent metrics to find a representative subset. The metrics are independent of microarchitecture but dependent on compiler used to compile benchmarks. It takes a significantly long time to generate the microarchitecture-independent metrics. This methodology is suitable for the problems discussed in this paper. But very long runtime of the benchmarks with some of them about few trillion instructions long makes the methodology in [4] impractical. Citron [9] presented subsets based on use by the computer architecture research community. The subsets had similarity to the subsets presented by Phansalkar *et .al*, however, there were differences as well.

8. Conclusion

The process of subsetting may result in loss of information and should only be used when it is difficult to run the complete SPEC CPU suite. This paper provides a quantitative approach to guide selection of benchmarks when a researcher cannot use all the benchmarks in the process of performance evaluation and helps to keep the loss of information very low. This methodology is based on using performance counters to measure different characteristics of programs on different systems, and using statistical analysis techniques such as PCA and clustering to visualize the workload space. A weakness of using microarchitecture-dependent measurements is the bias towards the architecture of a particular system. However, we try to eliminate that bias by using measurements from multiple computer systems with four different ISAs. Other than the characteristics of programs used in this paper, users may be interested in measuring more characteristics relevant to their study. Users should use their expertise to select characteristics specific to their objectives. In this paper, two sets of dendrograms are presented, one that illustrates the similarity between benchmarks based on overall performance characteristics, and the other which includes only branch and data access metrics. The paper also shows the methodology to find the input set that represents the whole run of the benchmark with all the input sets. This is very useful in academic research since a lot of research papers show the use of only a single input set per

benchmark. In such a case the input set that is most representative of the whole run of the benchmark should be chosen.

9. Disclaimer

All the observations and analysis done in this paper on SPEC CPU2006 benchmarks are the authors' opinions and should not be used as official or unofficial guidelines from SPEC in selecting benchmarks for any purpose. This paper only provides guidelines for researchers and academic users to choose a subset of benchmarks should the need be.

10. Acknowledgements

The authors express their gratitude to the following SPEC member companies (IBM, Sun Microsystems, Intel, AMD, Apple, HP, SGI) for providing performance counter data from their systems, for the analysis presented in this paper and/or the clustering analysis during the selection of the CPU2006 suite. We enjoyed the opportunity to interact with the SPEC CPU Subcommittee and provide our analysis regarding the similarity of the programs. The researchers are supported in part by National Science Foundation under grant number 0429806. Ajay Joshi is supported by an IBM fellowship.

11. References

[1] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Designing computer architecture research workloads", *IEEE Computer*, 36(2), pp. 65-71, Feb 2003.

[2] 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, 2003.

[3] H. Vandierendonck, K. Bosschere, "Many Benchmarks Stress the Same Bottlenecks", *Proc. of the Workshop on Computer Architecture Evaluation using Commerical Workloads (CAECW-7)*, pp. 57-71, 2004.

[4] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John, "Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites". IEEE International Symposium on Performance Analysis of Systems and Software. March 2005

[5] K. M. Dixit: Overview of the SPEC Benchmarks. "The Benchmark Handbook", Chapter 9, 1993

[6] R. Giladi and N. Ahituv, "SPEC as a Performance Evaluation Measure", IEEE Computer, Aug 1995, Vol. 28, No. 8, pp 33-42

[7] G. Dunteman, *Principal Components Analysis*, Sage Publications, 1989

[8] L. John, P. Vasudevan and J. Sabarinathan, "Workload Characterization: Motivation, Goals and Methodology", In Workload Characterization: Methodology and Case Studies, Edited by L. John and A. M. G. Maynard, IEEE Computer Society, pp. 3-14, November 1998

[9] D. Citron, J. Hennessy, D. Patterson, G. Sohi, "The Use and Abuse of SPEC: An ISCA panel", Proceedings of the 30th Annual International Symposium on Computer Architecture, pp. 73-77, June 9-11, 2003.

[10] J. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium" IEEE Computer, July 2000 Vol. 33, No. 7, pp. 28-35

[11] A. Joshi, A. Phansalkar, L. Eeckhout, and L. John, "Measuring Benchmark Similarity using Inherent Program Characteristics" IEEE Transactions on Computes, vol 55(6), pp. 769-782, June 2006.

[12] SPEC CPU2006 published results page : http://www.spec.org/cpu2006/results/