Measuring Benchmark Similarity Using Inherent Program Characteristics

Ajay Joshi, *Student Member*, *IEEE*, Aashish Phansalkar, *Student Member*, *IEEE*, Lieven Eeckhout, and Lizy Kurian John, *Senior Member*, *IEEE*

Abstract—This paper proposes a methodology for measuring the similarity between programs based on their inherent microarchitecture-independent characteristics, and demonstrates two applications for it: 1) finding a representative subset of programs from benchmark suites and 2) studying the evolution of four generations of SPEC CPU benchmark suites. Using the proposed methodology, we find a representative subset of programs from three popular benchmark suites—SPEC CPU2000, MediaBench, and MiBench. We show that this subset of representative programs can be effectively used to estimate the average benchmark suite IPC, L1 data cache miss-rates, and speedup on 11 machines with different ISAs and microarchitectures—this enables one to save simulation time with little loss in accuracy. From our study of the similarity between the four generations of SPEC CPU benchmark suites, we find that, other than a dramatic increase in the dynamic instruction count and increasingly poor temporal data locality, the inherent program characteristics have more or less remained unchanged.

Index Terms—Measurement techniques, modeling techniques, performance of systems, performance attributes.

1 Introduction

Modern day benchmark suites are typically comprised of a number of application programs where each benchmark consists of hundreds of billions of dynamic instructions. Therefore, a technique that can select a representative subset of programs from a benchmark suite can translate into large savings in simulation time with little loss in accuracy. Understanding the similarity between programs is important when selecting a subset of programs that are distinct, but are still representative of the benchmark suite. A typical approach to studying the similarity between programs is to measure program characteristics and then use statistical data analysis techniques to group programs with similar characteristics.

Programs can be characterized using microarchitecture-dependent characteristics, such as cycles per instruction (CPI), cache miss-rate, and branch prediction accuracy, or microarchitecture-independent characteristics, such as temporal data locality and instruction level parallelism. Techniques that have been previously proposed to find similarity between programs primarily use microarchitecture-dependent characteristics of programs (or at least a mix of microarchitecture-dependent and microarchitecture-independent characteristics) [12], [36]. This involves measuring program performance characteristics such as instruction and data cache miss rate, branch prediction accuracy, CPI, and execution time across multiple microarchitecture configurations. However, the results obtained from these techniques could be biased by the idiosyncrasies of a particular microarchitecture configuration. Therefore, conclusions based on performance characteristics such as execution time and cache miss-rate could categorize a program with unique characteristics as insignificant only because it shows similar trends on the microarchitecture configurations used in the study. For instance, a prior study [36] ranked programs in the SPEC CPU2000 benchmark suite using the SPEC peak performance rating (a microarchitecture-dependent characteristic). The program ranks were based on their uniqueness, i.e., the programs that exhibit different speedups on most of the machines were given a higher rank as compared to other programs in the suite. In this scheme of ranking programs, the gcc benchmark ranks very low and seems to be less unique. However, this result contradicts with what is widely believed in the computer architecture community—the gcc benchmark has distinct characteristics as compared to the other programs and, therefore, is an important benchmark. This indicates that an analysis based on microarchitecture-dependent characteristics (such as the SPEC peak performance rating and speedup) could undermine the importance of a program that is really unique.

We believe that, by measuring similarity using inherent characteristics of a program, it is possible to ensure that the results will be valid across a wide range of microarchitecture configurations. In this paper, we propose a methodology to find groups of similar programs based on their inherent characteristics and apply it to study the similarity between programs in three popular benchmark suites. More specifically, we make the following contributions:

- *A. Joshi, A. Phansalkar, and L.K. John are with the Department of Electrical and Computer Engineering, University of Texas at Austin, 1 University Station C0803, Austin, TX 78712-0240. E-mail: {ajoshi, aashish, ljohn}@ece.utexas.edu.*
- *L. Eeckhout is with the Department of Electronics and Information Systems, Ghent University, Sint-Pietersnieuwstraat 41, B-900 Gent, Belgium. E-mail: Lieven.Eeckhout@elis.ugent.be.*

Manuscript received 15 June 2005; revised 27 Dec. 2005; accepted 4 Jan. 2006; published online 21 Apr. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0199-06005.

1. We motivate and present an approach that can be used to measure the similarity between programs in a microarchitecture-independent manner.
2. We use the proposed methodology to find a subset of representative programs from the SPEC CPU2000, MiBench, and MediaBench benchmark suites, and demonstrate their usefulness in predicting the average performance metrics of the entire suite.
3. We demonstrate that the subset of SPEC CPU2000 programs formed using microarchitecture-independent characteristics is representative across a wide range of machines with different instruction set architectures (ISAs), compilers, and microarchitectures.
4. We provide an insight into how the program characteristics of four generations of SPEC CPU benchmark suites have evolved.

The paper is organized as follows: Section 2 describes our characterization methodology. Section 3 describes the results from applying the proposed methodology to find subsets of programs from the SPEC CPU2000 [16], MediaBench [23], and MiBench [14] benchmark suites. Section 4 presents validation experiments to demonstrate that the subsets of programs are indeed representative of the entire benchmark suite. Section 5 uses the presented methodology to study the similarity between characteristics of programs across four generations of SPEC CPU benchmark suites. Section 6 describes the related work and Section 7 summarizes the conclusions from this study.

2 Characterization Methodology

This section describes our methodology to measure the similarity between benchmark programs. It includes a description of the microarchitecture-independent characteristics, an outline of the statistical data analysis techniques, the benchmarks used, and the tools developed for this study.

2.1 Microarchitecture-Independent Characteristics

Microarchitecture-independent characteristics allow for a comparison between programs based on their inherent properties that are isolated from features of a particular machine configuration. As such, we use a gamut of microarchitecture-independent characteristics that affect overall program performance. The characteristics that we use in this study are a subset of all the microarchitecture-independent characteristics that can be potentially measured, but we believe that our characteristics cover a wide enough range of program characteristics to make a meaningful comparison between the programs; the results in this paper in fact show that this is the case. The microarchitecture-independent characteristics that we use in this study relate to the instruction mix, control flow behavior, instruction and data stream locality, and instruction-level parallelism. These characteristics are described below.

2.1.1 Instruction Mix

The instruction mix of a program measures the relative frequency of various operations performed by a program, namely, the percentage of computation instructions, data memory accesses (load and store instructions), and branch instructions in the dynamic instruction stream of a program.

2.1.2 Control Flow Behavior

We use the following set of metrics to characterize the branch behavior of programs:

- **Basic Block Size.** A basic block is a section of code with one entry and one exit point. We measure the basic block size as the average number of instructions between two consecutive branches in the dynamic instruction stream of the program. A larger basic block size is useful in exploiting instruction level parallelism (ILP) in an out-of-order superscalar microprocessor.
- **Branch Direction.** Backward branches are typically more likely to be taken than forward branches. This characteristic computes the percentage of forward branches out of the total branch instructions in the dynamic instruction stream of the program.
- **Fraction of Taken Branches.** This characteristic is the ratio of the number of taken branches to the total number of branches in the dynamic instruction stream of the program.
- **Fraction of Forward-Taken Branches.** This characteristic is the fraction of the forward branches in the dynamic instruction stream of the program that are taken.

2.1.3 Inherent Instruction Level Parallelism

Register Dependency Distance. We use a distribution of dependency distances as a measure of the inherent ILP in the program. Dependency distance is defined as the total number of instructions in the dynamic instruction stream between the production (write) and consumption (read) of a register instance [8], [26]. While techniques such as value prediction reduce the impact of these dependencies on ILP, information on the dependency distance is very useful in understanding the inherent ILP of the program. The dependency distance is classified into six categories: percentage of total dependencies that have a distance of one instruction and the percentage of total dependencies that have a distance of up to 2, 4, 8, 16, 32, and greater than 32 instructions. Programs that have a higher percentage of large dependency distances are likely to exhibit a higher inherent ILP.

2.1.4 Data Locality

Data Temporal Locality. Several locality characteristics have been proposed in the past [5], [6], [18], [22], [32], [33], [34]; however, the algorithms for calculating them are computation and memory intensive. We selected the average memory reuse distance characteristic proposed by Lafage and Seznec [22] since it is more computationally feasible than the other characteristics that have been proposed. The data temporal locality is quantified by computing the average distance (in terms of the number of data memory accesses) between two consecutive accesses to the same address, for every unique address in the program that is executed at least twice. For every program, we calculate the data temporal locality for window sizes of

16, 64, 256, and 4,096 bytes—these windows are to be thought of as cache blocks, i.e., the data temporal locality counts the number of access between two consecutive accesses to the same window. The choice of these particular window sizes is based on the experiments conducted by Lafage and Seznec [22]. Their experimental results showed that these four window sizes were sufficient to accurately characterize the locality of the data reference stream with respect to a wide range of data cache configurations.

Data Spatial Locality. Caches exploit spatial locality through the use of cache blocks, i.e., programs that have a good spatial locality will benefit from a large cache block. Therefore, a program that exhibits good spatial locality will show a significant reduction in the value of the temporal data locality characteristic, i.e., average memory reuse distance, as the window size is increased. In contrast, for a program with poor spatial locality, the value of the temporal data locality characteristic will not significantly reduce as the window size is increased. We capture the spatial locality of a program by computing the ratio of the data temporal locality characteristic for window sizes of 64, 256, and 4,096 bytes to the data temporal locality characteristic for a window size of 16 bytes. The values of these three ratios characterize the spatial locality of the program. A smaller ratio for a higher window size indicates that the program exhibits good spatial locality.

2.1.5 Instruction Locality

Instruction Temporal Locality. The instruction temporal locality is quantified by computing the average distance (in terms of the number of instructions) between two consecutive accesses to the same static instruction, for every unique static instruction in the program that is executed at least twice. Similar to the data temporal locality characteristic, we calculate the instruction temporal locality characteristic for window sizes of 16, 64, 256, and 4,096 bytes.

Instruction Spatial Locality. The spatial locality of the instruction stream is characterized by the ratio of the instruction temporal locality for window sizes of 64, 256, and 4,096 bytes, to the instruction temporal locality characteristic for a window size of 16 bytes—this is similar to how the data spatial locality characteristic is computed.

2.2 Statistical Data Analysis

There are several variables (29 microarchitecture-independent characteristics) and many cases (benchmarks) involved in our study. It is humanly impossible to simultaneously look at all the data and draw meaningful conclusions from them. Therefore, we use multivariate statistical data analysis techniques, namely, *Principal Component Analysis* and *Cluster Analysis*, to compare and discriminate programs based on the measured characteristics and understand the distribution of the programs in the workload space.

Principal components analysis (PCA) [10] is a classic multivariate statistical data analysis technique that is used to reduce the dimensionality of a data set while retaining most of the original information. We use PCA to remove the correlation between the measured variables and reduce the dimensionality of the data set. After performing PCA, we use clustering algorithms to find groups of programs with similar characteristics. There are two very popular clustering algorithms, *k-means* and *hierarchical* clustering [17]. In this paper, we use both the clustering approaches. For k-means clustering, we generate different random seeds to find the best initial placement of centers and then use the BIC (Bayesian Information Criterion) explained in [29] to find the best fit of k for the data, i.e., the optimal number of clusters in k-means clustering algorithm. The readers are referred to [20] for an overview of the PCA and clustering analysis techniques.

2.3 Benchmarks

We use programs from the SPEC CPU [16], MediaBench [23], and MiBench [14] benchmark suites in this study. Due to the differences in libraries, data type definitions, pointer size conventions, and known compilation issues on 64-bit machines, we were unable to compile some programs (mostly from old suites—SPEC CPU89 and SPEC CPU92). The programs were compiled on a Compaq Alpha AXP-2116 processor using the Compaq/DEC C, C++, and the FORTRAN compiler. The details of the programs and the input sets that we used in this study are listed in [20]. Although the characteristics that we measure are microarchitecture-independent, they are dependent on the instruction set architecture (ISA) and the compiler. However, in Section 4.2, we show that the subsets are reasonably valid across various compilers and ISAs.

2.4 Tools

SCOPE. The workload characteristics were measured using a custom-grown analyzer called *SCOPE*. *SCOPE* was developed by modifying the *sim-safe* functional simulator from the *SimpleScalar v3.0* tool set [1]. *SCOPE* analyzes the dynamic instruction stream and generates statistics related to the instruction mix, instruction and data locality, branch predictability, basic block size, and ILP. Essentially, the back-end of *sim-safe* is interfaced with custom developed analyzers to obtain the various microarchitecture-independent characteristics.

Statistical Data Analysis. We use STATISTICA software version 6.1 for performing PCA and hierarchical clustering. For k-means clustering, we use the *SimPoint* software [30]. However, we do not apply random projection before applying k-means clustering as done by default in the SimPoint software. Instead, we perform clustering in the transformed PCA space.

3 Subsetting Benchmark Suites

In order to find a subset of representative benchmark programs from a suite, we first measure the microarchitecture-independent characteristics, as described in Section 2, for all the benchmark programs. We then apply the PCA technique to remove correlation between the measured characteristics and to reduce the dimensionality of the data set and then use the k-means clustering algorithm and the Bayesian Information Criterion (BIC) to group the programs into k distinct clusters. A subset of representative programs is then composed by selecting one program from each cluster. In our study, we select the program that is closest to the center of its cluster as a representative of that group. For clusters with just two programs, any program can be chosen

TABLE 1
Optimal Clusters for SPEC CPU2000 Programs Based on the Overall Program Characteristics

Cluster 1	applu, mgrid
Cluster 2	bzip2, gzip
Cluster 3	crafty, equake
Cluster 4	**fma3d**, ammp, apsi, galgel, swim, vpr, wupwise
Cluster 5	**mcf**
Cluster 6	**twolf** , lucas, parser, vortex
Cluster 7	**mesa**, art, eon
Cluster 8	**gcc**

as the representative. We apply the subsetting methodology to the SPEC CPU2000, MiBench, and MediaBench benchmark suites. For each benchmark suite, we compose two subsets of programs, the first based on their overall characteristics and the second based just on their data locality characteristics.

3.1 Subsetting of SPEC CPU2000 Programs Using Overall Program Characteristics

In this section, we find a subset of representative programs from the SPEC CPU2000 benchmark suite based on the similarity between the overall characteristics of the programs. All 29 microarchitecture-independent program characteristics for 21 programs from the SPEC CPU2000 benchmark suite are used as input to the data analysis. After performing PCA and using BIC with k-means clustering, we obtain eight clusters as the best fit for the measured data set. Table 1 shows the eight clusters and their members. The programs in boldfaced font are chosen to be the representatives (closest to the center of the cluster) of that particular group.

Citron [4] presented a survey on the use of SPEC CPU2000 benchmark programs in papers from four recent ISCA conferences. He observed that some programs are more popular than the others among computer architecture researchers. The list of popular integer benchmarks in their decreasing order of popularity is: gzip, gcc, parser, vpr, mcf, vortex, twolf, bzip2, crafty, perlbmk, gap, and eon. For the floating-point benchmarks, the list in decreasing order of popularity is: art, equake, ammp, mesa, applu, swim, lucas, apsi, mgrid, wupwise, galgel, sixtrack, facerec, and fma3d. The clusters that we obtained in Table 1 suggest that the most popular programs in the listing provided by Citron [4] do not form a truly representative subset of the benchmark suite (based on their inherent characteristics). For example, subsetting SPEC CPU2000 integer programs using gzip, gcc, parser, vpr, mcf, vortex, twolf, and bzip2 will result in three uncovered clusters, namely, 1, 3, and 7. We also observe that there is a lot of similarity in the characteristics of the popular programs. The three popular benchmarks parser, twolf, and vortex belong to the same cluster, Cluster 6, and, hence, are not likely to provide any additional information. The results from Table 1 suggest that using applu, gzip, gcc, equake, fma3d, mcf, mesa, and twolf as a representative subset of the SPEC CPU2000 benchmark suite would be a better practice.

We observe that gcc is in a separate cluster by itself and, hence, has characteristics that are significantly different from other programs in the benchmark suite. After inspecting the characteristics, we observe that gcc has a peculiar instruction temporal locality behavior (large reuse distance for the instruction stream) and, hence, stands out from the rest of the programs. However, in the ranking scheme used in a prior study [36], gcc is ranked very low and does not seem to be a very unique program. Their study uses a microarchitecture-dependent characteristic, namely, the SPEC peak performance rating, and, hence, a program such as gcc that shows similar speedup on most of the machines will be ranked lower. This example shows that the results from analyzing microarchitecture-independent characteristics can identify redundancy more effectively.

3.2 Subsetting of Embedded Programs Using the Overall Program Characteristics

The MiBench and MediaBench benchmark suites represent the typical workloads used in embedded computing. MiBench suite consists of benchmarks that are representative of the workloads used in automotive, consumer devices, network, security, office automation, and telecommunications applications. The benchmarks in the MediaBench suite are representative of embedded multimedia and communication workloads. In this section, we compose a subset of representative embedded programs from MiBench and MediaBench benchmark suites, based on their overall program characteristics. We use the same procedure as described in the previous section, i.e., performing PCA on all 29 microarchitecture-independent characteristics followed by k-means clustering, to divide benchmarks into groups of similar programs. Using BIC with k-means clustering, we found five clusters as the best fit for this data.

Table 2 shows the five different groups of embedded benchmark programs. The program-input pairs marked in boldfaced font are the cluster representatives. We observe that, although MiBench and MediaBench are two different suites, they still have three common programs, namely, cjpeg, djpeg, and ghostscript. Although the cjpeg and djpeg benchmarks from the MiBench and MediaBench suites have different input sets, they reside in the same cluster (Clusters 1 and 3). This suggests that the input set does not affect the program behavior of the jpeg compression/decompression benchmarks. Also, the ghostscript benchmarks from the MiBench and MediaBench suites exhibit similar program characteristics.

Interestingly, the six automotive benchmarks from the MiBench suite show very little similarity between each other and are distributed in four out of five clusters. However, the benchmarks from the telecommunication and networking application domains are relatively very similar to each other. The bitcount automotive benchmark forms a singleton cluster (Cluster 5) and is, therefore, the most unique program in the two benchmark suites. The encoder and decoder versions of the MediaBench programs g.721, adpcm, and mpeg2 are also very similar to each other. From these observations, we can conclude that a large number of

TABLE 2
Optimal Clusters for Embedded Programs Based on the Overall Characteristics

Cluster 1	**mediabench_cjpeg**, mediabench_unepic, mediabench_ghostscript, mibench_consumer_cjpeg, mibench_office_ghostscript, mibench_office_rsynth
Cluster 2	**mediabench_rasta**, mediabench_mesa, mibench_automotive_qsort, mibench_network_dijkstra, mibench_network_patricia, mibench_office_stringsearch, mibench_security_sha, mibench_telecomm_CRC32
Cluster 3	**mibench_telecomm_invFFT**, mediabench_epic, mediabench_g721_decoder, mediabench_g721_encoder, mediabench_djpeg, mediabench_mpeg2_decoder, mediabench_mpeg2_encoder, mibench_automotive_basicmath, mibench_automotive_susan2, mibench_automotive_susan3, mibench_consumer_djpeg, mibench_consumer_typeset, mibench_ telecomm_FFT, mibench_telecomm_gsm
Cluster 4	**mediabench_adpcm_decoder**, mediabench_adpcm_encoder, mibench_automotive_susan1
Cluster 5	**mibench_automotive_bitcount**

programs from the MiBench and MediaBench suites show very similar program behavior and only five benchmarks, namely, cjpeg, rasta, invFFT, adpcm, and bitcount, are required to represent the 32 embedded benchmark programs from the two suites.

3.3 Subsetting of SPEC CPU2000 Programs Using the Data Locality Characteristics

In Section 3.1, we selected a representative subset of SPEC CPU2000 programs based on their overall program characteristics. However, architects and researchers often use cache simulations when performing studies related to the data memory hierarchy of a microprocessor. In order to select a representative subset of programs for such studies, one needs to understand the similarity between programs just based on their data locality characteristics.

In this analysis, we find a subset of the SPEC CPU2000 benchmark suite by only considering the seven characteristics of SPEC CPU2000 programs that are related to their temporal and spatial data locality. We use the same methodology, i.e., PCA followed by k-means and BIC, to group the programs into an optimal number of clusters. Table 3 shows the groups of SPEC CPU2000 programs that have similar data locality characteristics. We observe that a large number (nine out of 21) of SPEC CPU2000 programs are grouped together in one cluster (Cluster 3) and, hence, exhibit very similar data locality characteristics. Surprisingly, the floating-point benchmarks art, ammp, applu, and mgrid have similar temporal and spatial data locality characteristics as the integer benchmarks crafty, eon, parser, twolf, vortex, and vpr. Also, the floating-point benchmark mesa shows similar data locality characteristics as the integer benchmark gcc. We conclude that only three integer programs, gzip, mcf, and bzip2, and six floating-point programs, ammp, equake, mesa, fma3d, galgel, and wupwise, are representative of the data locality characteristics exhibited by programs in the SPEC CPU2000 benchmark suite.

TABLE 3
Optimal Clusters for SPEC CPU2000 Programs Based on Data Locality Characteristics

Cluster 1	**gzip**
Cluster 2	**mcf**
Cluster 3	**ammp**, applu, crafty, art, eon, mgrid, parser, twolf, vortex, vpr
Cluster 4	**equake**
Cluster 5	**bzip2**
Cluster 6	mesa, gcc
Cluster 7	**fma3d**, swim, apsi
Cluster 8	galgel, lucas
Cluster 9	**wupwise**

3.4 Subsetting Embedded Benchmarks Using Data Locality Characteristics

We now select a subset of representative embedded programs from the MiBench and MediaBench benchmark suites based on their similarity in data locality characteristics. Table 4 shows the eight groups of media programs that differ in their data locality behavior.

We observe that all the automotive benchmarks from the MiBench benchmark suite, susan, bitcount, basicmath, and qsort, reside in different clusters, suggesting that they have very different data locality characteristics. Particularly, the benchmark susan exhibits different data locality characteristics depending on the input set used. Also, susan forms a singleton cluster for inputs sets 1 and 2 and, therefore, has the most unique data locality characteristics of all the embedded programs. Interestingly, except for the epic benchmark, all the other pairs of compress/decompress and encoder/decoder benchmarks, namely, adpcm, g.721, jpeg, and mpeg2, show very similar data locality. One key conclusion that we can draw is that the combined set of embedded benchmark programs from the MiBench and MediaBench suites can be represented by six programs from the MiBench suite, namely, susan (input sets 1 and 2), djpeg, sha, qsort, ghostscript, and one program from the MediaBench suite, namely, adpcm. In other words, except for the

TABLE 4
Optimal Number of Clusters for Embedded Programs Based on the Data Locality Characteristics

Cluster 1	**mibench_automotive_susan1**
Cluster 2	**mibench_automotive_susan3**
Cluster 3	**mibench_ consumer_djpeg,** mediabench_epic, mediabench_cjpeg, mediabench_djpeg, mediabench_mpeg2_decode, mediabench_mpeg2_encoder, mibench_consumer_cjpeg, mibench_consumer_typeset, mibench_telecomm_FFT, mibench_telecomm_invFFT
Cluster 4	mediabench_adpcm_decoder**,** mediabench_adpcm_encoder
Cluster 5	**mibench_security_sha_large** ,mediabench_mesa, mediabench_rasta,mibench_ automotive_susan2, mibench_network_dijkstra, mibench_office_stringsearch,
Cluster 6	automotive_basicmath_large**,** network_patricia_large
Cluster 7	**mibench_automotive_qsort,** mediabench_unepic, mediabench_g721_decoder, ediabench_g721_encoder, mibench_automotive_bitcount, mibench_office_rsynth, mibench_telecomm_CRC32, mibench_telecomm_gsm
Cluster 8	mediabench_office_ghostscript, mibench_office_ghostscript

adpcm program, the data locality characteristics of the MediaBench programs are a subset of the data locality characteristics exhibited by programs from the MiBench suite.

4 Validating the Representativeness of Benchmark Subsets

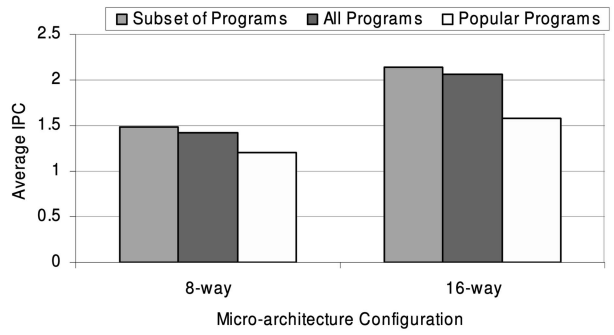
It is important to understand whether the subsets of programs we have created are meaningful and are indeed representative of the original benchmark suite. Therefore, we use the subset of programs, composed using our proposed methodology, to estimate the average benchmark suite IPC, L1 data cache miss-rate, and speedup. We then compare our results to those obtained by using the entire benchmark suite.

4.1 Estimating IPC through Subsetting

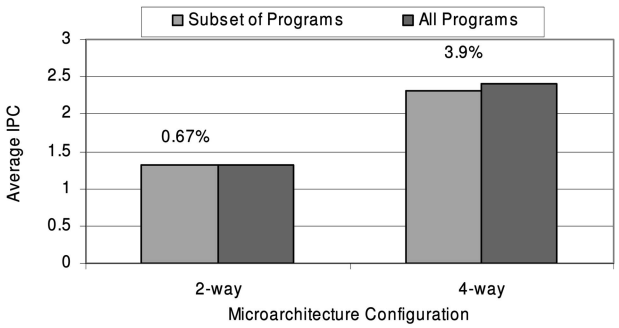
Using the subset of programs based on overall program characteristics from the SPEC CPU2000, MiBench, and MediaBench benchmark suites, we estimated the average IPC of the entire suite for two different superscalar configurations. For the SPEC CPU2000 benchmark programs, we used an 8-way and a 16-way issue superscalar microprocessor configuration, whereas, for the MiBench and MediaBench programs, we used a 2-way and a 4-way issue configuration. The details of the configurations are listed in [20]. From Table 1 and Table 2, we observe that each cluster has a different number of programs and, hence, the weight assigned to each representative program should depend on the number of programs that it represents, i.e., the number of programs in its cluster. For example, from Table 1, the weight for fma3d (Cluster 4) is 7. Similarly, we assign a weight to each representative program and, using these weights, we calculate the weighted harmonic mean of the IPC for the entire suite.

Fig. 1a shows the weighted average (harmonic mean) IPC of the entire SPEC CPU2000 benchmark suite, the estimated IPC from the subset of programs from Table 1, and the average (harmonic mean) IPC calculated using the list of popular programs published by Citron [4]. We obtained the IPC performance data for an 8-wide and 16-wide superscalar out-of-order microarchitecture for every program in the SPEC CPU2000 benchmarks from Wenisch et al. [38].

The error in weighted average IPC computed using the subset of programs in Table 1 for both 8-way and 16-way issue widths is less than 5 percent. We observe that the average IPC calculated using the list of popular programs published by Citron in [4] shows high errors (-15 percent



(a)



(b)

Fig. 1. Estimating average IPC using a subset of programs from the (a) SPEC CPU2000 and (b) MiBench and MediaBench benchmark suites.

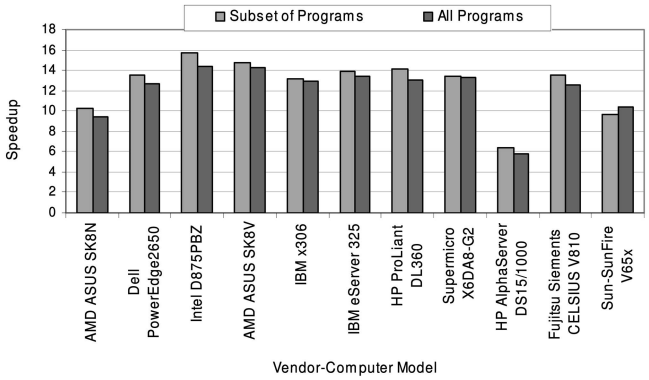


Fig. 2. Estimating average speedup using a subset of programs from the SPEC CPU2000 benchmark suite.

and -23.4 percent, respectively, for the 8-way and 16-way issue configurations, respectively). The two main reasons why we see a higher error from the subset of popular programs are: 1) The popular subset of programs is not selected by using a formal methodology to find similarity/dissimilarity with the rest of the programs and 2) there is no method to assign weights to the programs in the subset.

Fig. 1b shows the average IPC (harmonic mean) of all the embedded programs from the MiBench and MediaBench benchmark suites and the estimated average IPC (weighted harmonic mean) using the subset of programs shown in Table 2. We find that the error in estimating the average IPC using the subset of programs is very small for both the configurations (-0.67 percent for 2-way issue and -3.9 percent for 4-way issue).

Since the IPC of the entire suite can be estimated with reasonable accuracy using the subsets formed using our methodology, we feel that it is a good validation for the usefulness of the subsets.

4.2 Estimating Speedup of SPEC CPU2000 Benchmarks through Subsetting

In the previous section, we evaluated the usefulness of the subset to accurately estimate the overall IPC in a single design point. However, in the early stages of the design cycle, relative accuracy, i.e., the ability to predict speedup, is even more important. We now demonstrate the usefulness of the subset of programs from the SPEC CPU2000 suite to estimate the speedup of 11 machines from different vendors with respect to the base machine (Sun Ultra5_10 with 300MHz processor) that SPEC uses to calculate the SPEC CPU rating. Fig. 2 shows the estimated weighted average (geometric mean) speedup of the entire suite using the subset based on overall program characteristics and the average speedup (geometric mean) of the entire suite for computers from various manufacturers.

The speedup numbers for the SPEC CPU2000 programs were directly obtained from their execution times published by SPEC [42]. The maximum error in the speedup estimated using the subset is 9.1 percent. Since the machines used in this experiment have different ISAs, microarchitecture, and compiler settings, we can conclude that the subset of programs composed using inherent program characteristics is valid across different microarchitectures, ISAs, and compilers.

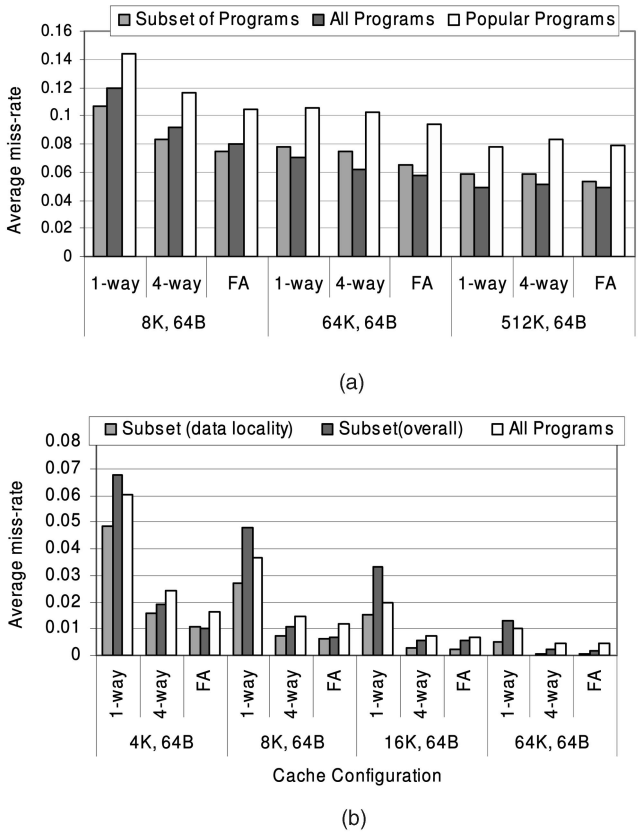


Fig. 3. Estimating the average data cache miss rate using a subset of programs from the (a) SPEC CPU2000 and (b) MiBench and MediaBench suites.

4.3 Estimating Average Data Cache Miss-Rate through Subsetting

In this section, we evaluate the usefulness of the subset of programs, formed using the data locality characteristics, in estimating the average data cache miss-rate of the entire suite. Similarly to the procedure described in the earlier section, we assign a weight to every representative program. Fig. 3a shows the weighted average (harmonic mean) L1 data cache miss-rate of the SPEC CPU2000 benchmark suite estimated using the subset of programs shown in Table 3 (based on data locality characteristics), the estimated average (harmonic mean) L1 data cache miss-rate using the entire benchmark suite, and the estimated average L1 data cache miss-rate using the list of popular programs published by Citron in [4]. We obtained the miss-rates for nine different L1 data cache configurations from Cantin and Hill [3]. The average absolute error in estimating the L1 data cache miss-rate of the entire suite using the subset of programs shown in Table 3 is 0.8 percent. The average absolute error in estimating the L1 data cache miss-rate using the set of popular programs is 3 percent. From these results, we can conclude that the program subset derived in Table 3 is indeed representative of the data locality characteristics of programs in the SPEC CPU2000 benchmark suite.

Fig. 3b shows the average (harmonic mean) L1 data cache miss-rate of the entire set of embedded programs and the

TABLE 5
Optimum Number of Clusters for the Four Generations of SPEC CPU Benchmark Programs Using the Overall Program Characteristics

Cluster 1	gcc(95), gcc(2000)
Cluster 2	mcf(2000)
Cluster 3	**turbo3d (95)**, applu (95), apsi(95), swim(2000), mgrid(95), wupwise(2000)
Cluster 4	**hydro2d(95**), hydro2d(92), wave5(92), su2cor(92), succor(95), apsi(2000), tomcatv(89), tomcatv(92), crafty(2000), art(2000), equake(2000), mdljdp2(92)
Cluster 5	**perl(95)**, li (89), li(95), compress(92), tomcatv(95), matrix300(89)
Cluster 6	**nasa7(92)**, nasa(89), swim(95), swim(92), galgel(2000), wave5(95), alvinn(92)
Cluster 7	applu(2000), mgrid(2000)
Cluster 8	**doduc(92**), doduc(89), ora(92)
Cluster 9	mdljsp2(92), lucas(2000)
Cluster 10	**parser(2000)**, twolf(2000), espresso(89), espresso(92), compress(95), go(95), ijpeg(95), vortex(2000)
Cluster 11	**fppp(95)**, fppp(92), eon(2000), vpr(2000), fppp(89), fma3d(2000), mesa(2000), ammp(2000)
Cluster 12	bzip2(2000**)**, gzip(2000)

estimated weighted average (harmonic mean) L1 data cache miss-rate using the subset of programs shown in Table 2 (all characteristics) and Table 4 (only data locality characteristics). We use 12 different cache configurations (sizes of 4KB, 8KB, 16KB, and 64KB, each with a direct-mapped, 4-way set associative, and fully associative configurations) to validate the representativeness of the subset of programs. The average absolute error in estimating L1 data cache miss-rate using the subset based on overall program characteristics is 0.6 percent and using the subset based on data locality characteristics is 0.5 percent. Again, our results show that the subset of programs is very effective in estimating the data cache miss-rate of the entire suite.

5 Similarity across Four Generations of SPEC CPU Benchmark Suites

We now use the methodology presented in this paper for analyzing how benchmark programs evolve with time. The Standard Performance Evaluation Corporation (SPEC) CPU benchmark suite, which was first released in 1989 as a collection of 10 computation-intensive benchmark programs (average size of 2.5 billion dynamic instructions per program), is now in its fourth generation and has grown to 26 programs (average size of 230 billion dynamic instructions per program). So far, SPEC has released four CPU benchmark suites: in 1989, 1992, 1995, and 2000.

In this section, we use our collection of microarchitecture-independent characteristics, described in Section 2, to characterize the generic behavior of four generations of SPEC CPU benchmark programs. In these experiments, we use the same compiler to compile programs from all four suites. The data is analyzed using PCA and cluster analysis to understand the changes in the CPU workloads over time. First, we use all the characteristics and perform k-means clustering to find optimal number of clusters for all the four generations of SPEC CPU benchmarks. In the subsequent sections, we analyze each important characteristic separately for all the generations. In order to visualize the workload space, we plot the scores for the first two PCs for 60 programs on a two-dimensional graph and also plot a dendrogram showing the similarity between the programs.

5.1 Overall Characteristics

In order to understand the (dis)similarity between programs across SPEC CPU benchmark suites, we perform a cluster analysis in the PCA space as described in Section 3. Clustering all the 60 benchmarks yields 12 optimum clusters, which are shown in Table 5; the benchmarks in boldfaced font are the cluster representatives.

A detailed analysis of Table 5 gives us several interesting insights. First, out of all the benchmarks, gcc (2000) and gcc (95) are together in a separate cluster. We observe that instruction locality for gcc is worse than any other program in all four generations of the SPEC CPU suite. Because of this, the gcc programs from the SPEC CPU 95 and 2000 suites reside in their own separate cluster. Due to its peculiar data locality characteristics, mcf (2000) resides in a separate cluster (cluster 2), and bzip2 (2000) and gzip (2000) form one cluster (cluster 12). SPEC CPU2000 programs exist in 10 out of 12 clusters, as opposed to SPEC CPU95 in seven clusters, SPEC CPU92 in six clusters, and SPEC CPU89 in five clusters. This shows that the SPEC CPU2000 benchmark suite is more diverse than its ancestors.

5.2 Instruction Locality

We perform PCA on the raw data measured for the instruction locality characteristics, which yields two principal components explaining 68.4 percent and 28.6 percent of the total variance. Fig. 4 shows the benchmarks in the PCA space. In order to visualize the relative positions of the benchmarks in the workload space, we also present a tree, or dendrogram, using hierarchical clustering. Fig. 5 shows the dendrogram obtained from applying hierarchical clustering to the data set in the PCA space. The horizontal scale of the dendrogram lists the benchmarks and the vertical scale corresponds to the

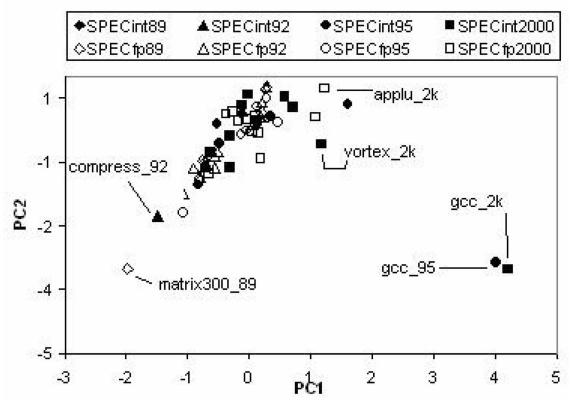


Fig. 4. PCA space built from the instruction locality characteristics.

linkage distance obtained from the hierarchical clustering analysis. The shorter the linkage distance, the closer, i.e., the more similar, the benchmarks are to each other in the workload space.

For example, in Fig. 5, the gcc (2000) and gcc (95) benchmarks combine into a cluster at a linkage distance of 0.2 and the cluster containing the two gcc benchmarks combines into a cluster containing all the other programs at a linkage distance of 6.2. This means that the gcc benchmarks from the SPEC CPU95 and SPEC CPU2000 benchmark suites are more similar to each other than to all the other programs.

PC1 represents the instruction temporal locality and PC2 represents the instruction spatial locality of the benchmarks, i.e., the benchmarks with a higher value along PC1 show poor temporal locality for the instruction stream, and the benchmarks with a higher value along PC2 show good spatial locality in the instruction stream. Fig. 4 and Fig. 5 show that programs from all the SPEC CPU generations overlap. The biggest exception is gcc in SPECint2000 and SPECint95 (the two dark points on the plot on the extreme right). The gcc benchmark from the SPECint2000 and SPECint95 suites exhibits poor instruction temporal locality. It also shows very low values for PC2 due to poor spatial locality. The floating-point program matrix300 from the SPEC CPU89 suite and compress from SPEC CPU92 show very good temporal and spatial locality. The benchmark program applu from SPEC CPU2000 shows a very high value for PC2 and would therefore benefit a lot from an increase in block size. The fppp benchmarks from the SPEC CPU89, SPEC CPU92, SPEC CPU95 suites, and the bzip2 and gzip benchmarks from the SPEC2000 suite show similar instruction locality.

In general, we observe that, although the average dynamic instruction count of the benchmark programs has increased by a factor of x100, the static instruction count has remained more or less constant. This suggests that the dynamic instruction count of the SPEC CPU benchmark programs have simply been scaled—more iterations through the same instructions.

5.3 Branch Characteristics

For studying the branch behavior, we include the following characteristics in our analysis: the percentage of branches in the dynamic instruction stream, the average basic block size, the percentage forward branches, the percentage taken branches, and the percentage forward-taken branches. From PCA analysis, we retain two principal components explaining 62 percent and 19 percent of the total variance, respectively. Fig. 6 plots the various SPEC CPU benchmarks in this PCA space and Fig. 7 is a dendrogram showing the linkage distance between the programs based on the branch characteristics.

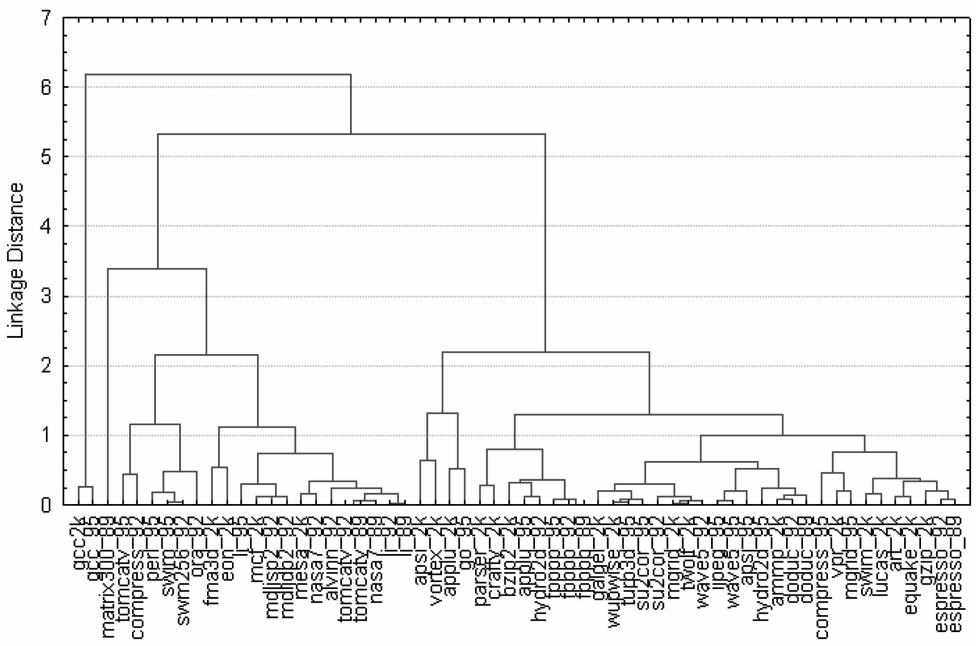


Fig. 5. Dendrogram showing the linkage distance between programs based on the instruction locality characteristics.

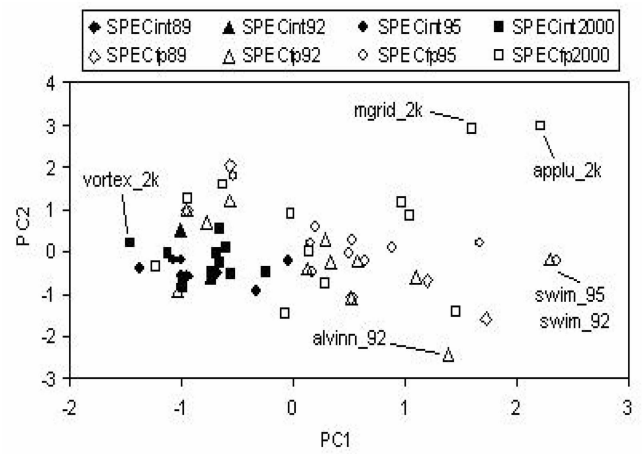


Fig. 6. PCA space built from the branch characteristics.

We observe that the integer benchmarks are clustered in an area. We also observe that the floating-point benchmarks typically have a positive value along the first principal component (PC1), whereas the integer benchmarks have a negative value along PC1. The reason is that floating-point benchmarks typically have fewer branches and, thus, have a larger basic block size; also, floating-point benchmarks typically are very well structured and have a smaller percentage of forward branches and fewer forward-taken branches.

In other words, floating-point benchmarks tend to spend most of their time in loops. The two prominent outliers in the top right corner of this graph are SPEC 2000's mgrid and applu programs due to their extremely large average basic block sizes, 273 and 318 instructions, respectively. The two outliers on the right are swim benchmarks from SPEC92 and SPEC95 suites, due to their large percentage of taken branches and small percentage of forward branches. On the extreme left of the PCA space is vortex from SPEC2000, which shows a very low average basic block size. Due to a significant overlap seen in the plot, we can conclude that the branch characteristics of the SPEC CPU programs did not significantly change over the past four generations of SPEC CPU programs. Fig. 7 also suggests that the branch behavior of programs has not significantly changed for the last four generations—doduc, espresso, fppp, hydro2d, li, and tomcatv are examples of programs whose branch characteristics have not changed across generations of SPEC CPU benchmark suites.

5.4 Instruction-Level Parallelism

In order to study the instruction-level parallelism (ILP) of the SPEC CPU suites we used the interinstruction register dependency characteristic. This characteristic is closely related to the intrinsic ILP available in an application. Long dependency distances generally imply a high ILP. The first two principal components explain 96 percent of the total variance. The PCA space is plotted in Fig. 8 and Fig. 9 shows the dendrogram with the linkage distance between the programs based on their ILP characteristics.

We observe that the integer benchmarks typically have a high value along PC1, which indicates that these benchmarks have a higher percentage of short dependency distances. The floating-point benchmarks typically have larger dependency distances. We observe no real trend in this graph. The intrinsic ILP did not change over the four benchmark suites except for the fact that several floating-point programs from the SPEC CPU89 and SPEC CPU92 suites (and no SPEC CPU95 or SPEC CPU2000 benchmarks) exhibit relatively short dependencies compared to other floating-point benchmarks; these overlap with integer benchmarks in the range $-0.1 < PC1 < 0.6$.

In the top left corner, we can see two outliers, mgrid and applu, that are quite far from a lot of other programs and

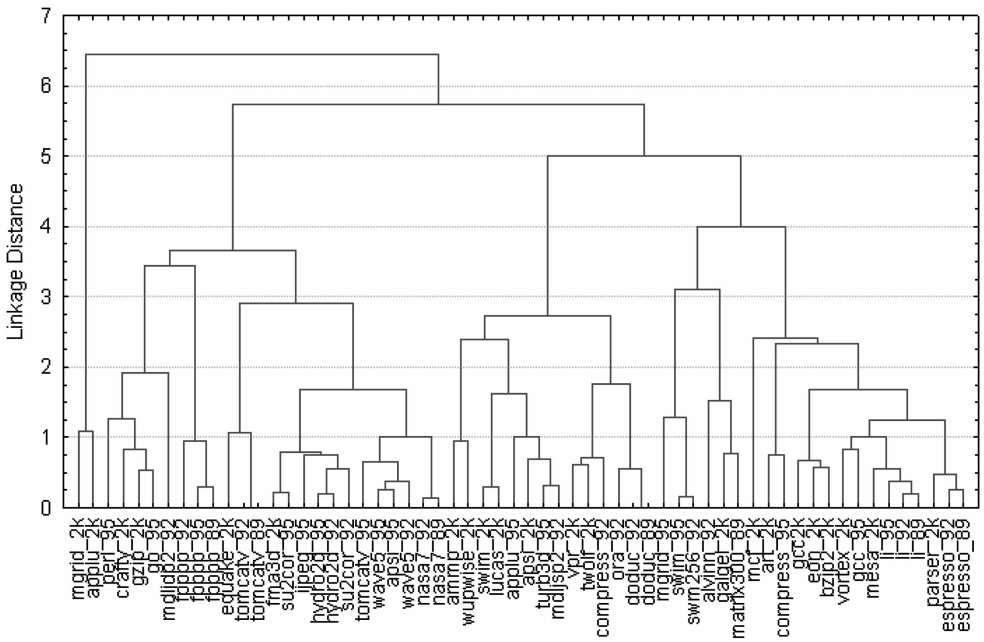


Fig. 7. Dendrogram showing linkage distance between programs based on the branch characteristics.

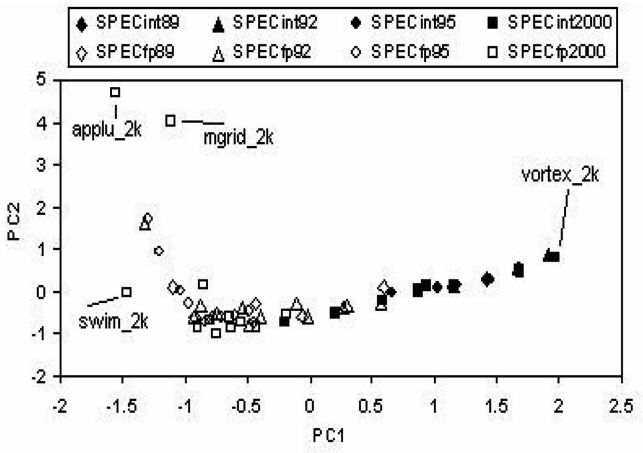


Fig. 8. PCA space built from the ILP characteristics.

show large dependency distances, which implies better ILP. The program swim from the SPEC CPU2000 suite also shows large dependency distances. The majority of the programs on the right side of the PCA space are integer programs, with vortex from SPEC 2000 being the one with the largest number of short dependency distances. In Fig. 9, we observe that a lot of floating-point programs across various generations, e.g., fppp, tomcatv, nasa7, li, and doduc, form a tight cluster. Hence, we can conclude that there is a lot of similarity between the ILP characteristics exhibited by the floating-point programs across all four generations of the SPEC CPU suites.

5.5 Data Locality

For studying the temporal and spatial locality behavior of the data stream, we used the locality characteristics described in Section 2. Recall that the characteristics by themselves quantify temporal locality, whereas the ratios between them are a measure of the spatial locality. Fig. 10 shows a plot of the benchmarks in the PCA space built from these data locality characteristics and Fig. 11 shows the linkage distance between various programs.

In Fig. 10, the first principal component measures temporal locality, i.e., a more positive value along PC1 indicates poorer temporal locality. The second principal component measures spatial locality. Therefore, benchmarks with a high value along PC2 will thus benefit more from an increased cache line size. From this figure, we conclude that several SPEC CPU2000 and CPU95 benchmark programs, namely, bzip2, gzip, mcf, and wupwise, from CPU2000, and gcc, turbo3d, applu, and mgrid from CPU95, exhibit a temporal locality that is significantly worse than the other benchmarks. Concerning spatial locality, most of these benchmarks exhibit a spatial locality that is relatively higher than that of the remaining benchmarks, i.e., increasing the *window sizes* improves the performance of these programs more than it does for the other benchmarks.

Programs like gzip, bzip2, and mcf show poor spatial locality. There are a lot of programs in all four generations of the SPEC CPU suites that overlap. This indicates that, although the objective of SPEC is to worsen the data stream locality behavior of subsequent CPU suites, several benchmarks in recent suites exhibit a locality behavior that is similar to older versions of SPEC CPU. Moreover, several CPU95 benchmarks, like wave, perl, compress, apsi, and CPU2000 benchmarks, like equake, galgel, lucas, and swim that show a temporal locality behavior that is better than some CPU89 and CPU92 benchmarks.

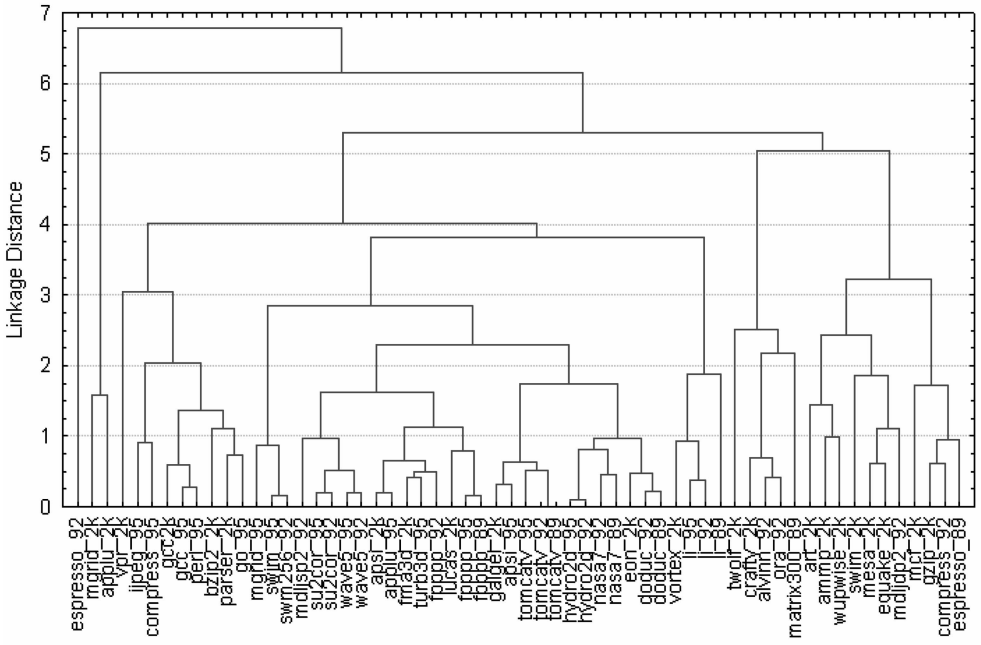


Fig. 9. Dendrogram showing the linkage distance between programs based on the ILP characteristics.

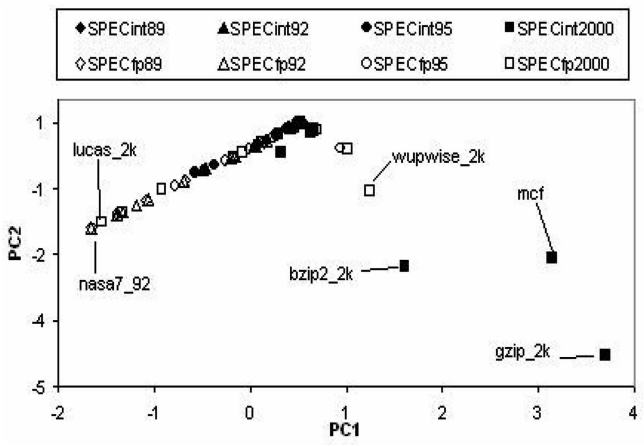


Fig. 10. PCA space built from the data locality characteristics.

6 Related Work

Weicker [37] used characteristics such as statement distribution in programs, distribution of operand data types, and distribution of operations, to study the behavior of several stone-age benchmarks. Saveedra and Smith [28] characterized Fortran applications in terms of the number of various fundamental operations and predicted their execution time. They also developed a measure for program similarity that makes it possible to classify benchmarks with respect to a large set of characteristics.

Prior work in studying benchmark characteristics has typically taken the approach of measuring microarchitecture-dependent characteristics, e.g., cycles per instruction, cache miss rate, branch prediction accuracy, etc., on various microarchitecture configurations that offer a different mixture of bottlenecks [11], [12], [36], [41]. The variation in these characteristics is then used to infer the generic program behavior.

There has been prior research to find redundancy in benchmark suites. Dujmovic and Dujmovic [9] developed a quantitative approach to evaluate benchmark suites. They used the execution time of a program on several machines to calculate measures that quantify the size, completeness, and redundancy of the benchmark space. Vandierendonck and De Bosschere [36] analyzed the SPEC CPU2000 benchmark suite peak results on 340 different machines representing eight ISAs and used PCA to identify the redundancy in the benchmark suite. In [36], the authors quantify redundancy as the ability of a program to show different speedup on two different machines. The programs that do not show very different speedups are considered redundant. They conclude that only a subset of programs from SPEC CPU2000 benchmark programs is required to accurately predict the ranks of these 340 machines.

There has been some research on microarchitecture-independent locality and ILP characteristics. For example, locality models researched in the past include working set models, least recently used stack models, independent reference models, temporal density functions, spatial density functions, memory reuse distance, locality space, etc. [5], [6], [18] [22], [32], [33], [34]. Generic measures of parallelism were used by Noonburg and Shen [26] and Dubey et al. [8] based on a profile of dependency distances in a program. Microarchitecture-independent characteristics, such as true computations versus address computations and overhead memory accesses versus true memory accesses, have been proposed by several researchers [15], [19]. The methodology presented in this paper can benefit from more microarchitecture-independent characteristics, but we believe that the characteristics we have used cover a wide enough range of the program characteristics to make a meaningful comparison between the programs.

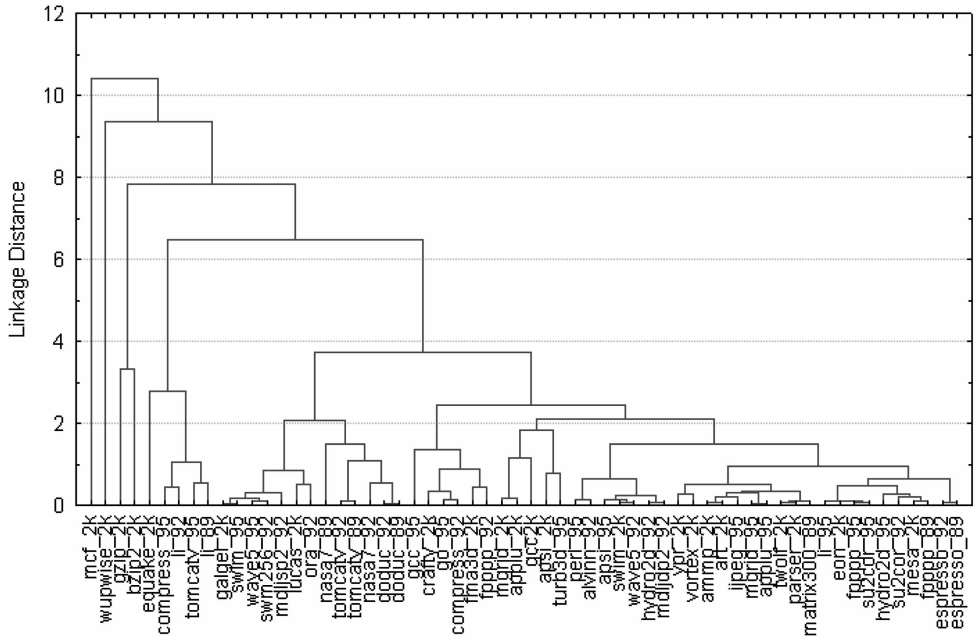


Fig. 11. Dendrogram showing the linkage distance between programs based on the data locality characteristics.

Another stream of work reduces the simulation time of benchmarks by finding representative phases within a program [29], [30], [40]. These techniques are orthogonal to the one presented in this paper and can be used to further reduce the simulation time of the subset of programs selected from the suite.

7 Conclusion

In this paper, we proposed a method to measure the similarity between programs based on their inherent microarchitecture-independent characteristics and we demonstrated the use of this technique to subset programs from the SPEC CPU2000, MiBench, and MediaBench benchmark suites. We validated the usefulness of the subsets obtained using our methodology by demonstrating that the average IPC, data cache miss rate, and speedup of the entire suite could be estimated with reasonable accuracy by just simulating the subset of programs. Based on our results and validation experiments, we recommend that, if the time required to simulate the entire SPEC CPU benchmark suite is prohibitively high, the following set of programs should be used as a representative subset: applu, equake, fma3d, gcc, gzip, mcf, mesa, and twolf.

From our study on the similarity between the four generations of SPEC CPU benchmark suites, we find that no single characteristic has changed as dramatically as the dynamic instruction count. Our analysis shows that the branch and ILP characteristics have not changed much over the last four generations, but the temporal data locality of programs has become increasingly poor.

The methodology presented in this paper can be used to select representative programs for the characteristics of interest should the cost of simulating the entire suite be prohibitively high. This technique can also be used during the benchmark design process to compose a benchmark suite from a group of candidate program.

Acknowledgments

This paper is an extended version of [27], published at the International Symposium on Performance Analysis of Systems and Software (ISPASS), 2005. This research is supported in part by US National Science Foundation grants 0113105, 0429806 and IBM and Intel Corporations. Lieven Eeckhout is a postdoctoral fellow of the Fund for Scientific Research-Flanders (Belgium) (F.W.O Vlaanderen).

References

[1] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer,* vol. 35, no. 2, pp. 59-67, Feb. 2002.

[2] L. Barroso, K. Ghorachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," *Proc. Int'l Symp. Computer Architecture,* pp. 3-14, 1998.

[3] J. Cantin and M. Hill, "Cache Performance for SPEC CPU2000 Benchmarks," http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/, 2003.

[4] D. Citron, "MisSPECulation: Partial and Misleading Use of SPEC CPU2000 in Computer Architecture Conferences," *Proc. Int'l Symp. Computer Architecture,* pp. 52-61, 2003.

[5] T. Conte and W. Hwu, "Benchmark Characterization for Experimental System Evaluation," *Proc. Hawaii Int'l Conf. System Science,* vol. I, Architecture Track, pp. 6-18, 1990.

[6] P. Denning, "The Working Set Model for Program Behavior," *Comm. ACM,* vol. 2, no. 5, pp. 323-333, 1968.

[7] K. Dixit, "Overview of the SPEC Benchmarks," *The Benchmark Handbook,* chapter 9. Morgan Kaufmann, 1998.

[8] P. Dubey, G. Adams, and M. Flynn, "Instruction Window Size Trade-Offs and Characterization of Program Parallelism," *IEEE Trans. Computers,* vol. 43, no. 4, pp. 431-442, Apr. 1994.

[9] J. Dujmovic and I. Dujmovic, "Evolution and Evaluation of SPEC Benchmarks," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 26, no. 3, pp. 2-9, 1998.

[10] G. Dunteman, *Principal Component Analysis.* Sage Publications, 1989.

[11] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Designing Computer Architecture Research Workloads," *Computer,* vol. 36, no. 2, pp. 65-71, Feb. 2003.

[12] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Quantifying the Impact of Input Data Sets on Program Behavior and Its Applications," *J. Instruction Level Parallelism,* vol. 5, pp. 1-33, 2003.

[13] R. Giladi and N. Ahituv, "SPEC as a Performance Evaluation Measure," *Computer,* vol. 28, no. 8, pp. 33-42, Aug. 1995.

[14] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proc. Fourth Ann. Workshop Workload Characterization,* 2001.

[15] D. Hammerstrom and E. Davdison, "Information Content of CPU Memory Referencing Behavior," *Proc. Int'l Symp. Computer Architecture,* pp. 184-192, 1997.

[16] J. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millenium," *Computer,* vol. 33, no. 7, pp. 28-35, July 2000.

[17] A. Jain and R. Dubes, *Algorithms for Clustering Data.* Prentice Hall, 1988.

[18] L. John, P. Vasudevan, and J. Sabarinathan, "Workload Characterization: Motivation, Goals and Methodology," *Workload Characterization: Methodology and Case Studies,* L.K. John and A.M.G. Maynard, eds., IEEE CS Press, 1999.

[19] L. John, V. Reddy, P. Hulina, and L. Coraor, "Program Balance and Its Impact on High Performance RISC Architecture," *Proc. Int'l Symp. High Performance Computer Architecture,* pp. 370-379, Jan. 1995.

[20] A. Joshi, A. Phansalkar, L. Eeckhout, and L. John, "Measuring Benchmark Similarity Using Inherent Program Characteristics," Laboratory of Computer Architecture Technical Report TR-060201, The Univ. of Texas at Austin, Feb. 2006.

[21] A.J. KleinOswoski and D. Lilja, "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research," *Computer Architecture Letters,* pp. 10-13, 2002.

[22] T. Lafage and A. Seznec, "Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream," *Proc. Workshop Workload Characterization (WWC-2000),* Sept. 2000.

[23] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems," *Proc. Int'l Symp. Microarchitecture,* 1997.

[24] N. Mirghafori, M. Jacoby, and D. Patterson, "Truth in SPEC Benchmarks," *Computer Architecture News,* vol. 23, no. 5, pp. 34-42, Dec. 1995.

[25] S. Mukherjee, S. Adve, T. Austin, J. Emer, and P. Magnusson, "Performance Simulation Tools," *Computer,* vol. 35, no. 2, Feb. 2002.

[26] D. Noonburg and J. Shen, "A Framework for Statistical Modeling of Superscalar Processor Performance," *Proc. Int'l Symp. High Performance Computer Architecture,* pp. 298-309, 1997.

[27] A. Phansalkar, A. Joshi, L. Eeckhout, and L. John, "Measuring Program Similarity—Experiments with SPEC CPU Benchmark Suites," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2005.

[28] R. Saveedra and A. Smith, "Analysis of Benchmark Characteristics and Benchmark Performance Prediction," *ACM Trans. Computer Systems,* vol. 14, no. 4, pp. 344-384, 1996.

[29] T. Sherwood, E. Perelman, and B. Calder, "Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* pp. 3-14, 2000.

[30] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. Int'l Conf. Architecture Support for Programming Languages and Operating Systems,* pp. 45-57, 2002.
[31] K. Skadron, M. Martonosi, D. August, M. Hill, D. Lilja, and V. Pai, "Challenges in Computer Architecture Evaluation," *Computer,* pp. 30-36, Aug. 2003.
[32] E. Sorenson and J. Flanagan, "Cache Characterization Surfaces and Prediction of Workload Miss Rates," *Proc. Int'l Workshop Workload Characterization,* pp. 129-139, Dec. 2001.
[33] E. Sorenson and J. Flanagan, "Evaluating Synthetic Trace Models Using Locality Surfaces," *Proc. Fifth IEEE Ann. Workshop Workload Characterization,* pp. 23-33, Nov. 2002.
[34] J. Spirn and P. Denning, "Experiments with Program Locality," *Proc. The Fall Joint Conf.,* pp. 611-621, 1972.
[35] Standard Performance Evaluation Corp., http://www.spec.org/benchmarks.html, 2005.
[36] H. Vandierendonck and K. De Bosschere, "Many Benchmarks Stress the Same Bottlenecks," *Proc. Workshop Computer Architecture Evaluation Using Commerical Workloads (CAECW-7),* pp. 57-71, 2004.
[37] R. Weicker"An, Overview of Common Benchmarks," *Computer,* vol. 23, no. 12, pp. 65-75, Dec. 1990.
[38] T. Wenisch, R. Wunderlich, B. Falsafi, and J. Hoe, "Applying SMARTS to SPEC CPU2000," CALCM Technical Report 2003-1, Carnegie Mellon Univ., June 2003.
[39] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proc. Int'l Symp. Computer Architecture,* pp. 24-36, June 1995.
[40] J. Wunderlich, R. Wenisch, B. Falfasi, and J. Hoe, "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," *Proc. Int'l Symp. Computer Architecture,* pp. 84-95, 2003.
[41] J. Yi, D. Lilja, and D. Hawkins, "A Statistically Rigorous Approach for Improving Simulation Methodology," *Proc. Int'l Conf. High-Performance Computer Architecture,* pp. 281-291, 2003.
[42] "All Published SPEC CPU200 Results,"http://www.spec.org/cpu2000/results/cpu2000.html, 2005.

Ajay Joshi received the BE degree in instrumentation engineering from the University of Pune, India, in 1998 and the MS degree in electrical and computer engineering from Ohio State University in 2001. He is currently a PhD candidate at the University of Texas at Austin, where he is investigating techniques to automatically generate representative synthetic workloads. His research interests include computer performance evaluation, benchmarking, workload characterization, and computer architecture. He is a student member of the IEEE, the IEEE Computer Society, and the ACM.

Aashish Phansalkar is a PhD candidate in the Department of Electrical and Computer Engineering at the University of Texas at Austin. He received the MS degree in engineering from the University of Texas at Austin. His research interests include workload characterization, performance evaluation, and computer architecture. He is currently working on microarchitecture independent characterization of programs. He is a student member of the IEEE and the IEEE Computer Society.

Lieven Eeckhout received the engineering degree and the PhD degree in computer science from Ghent University, Belgium, in 1998 and 2002, respectively. He is currently a postdoctoral fellow of the Fund for Scientific Research-Flanders (Belgium) (F.W.O.-Vlaanderen). His research interests include computer architecture, performance analysis, and workload characterization.

Lizy Kurian John received the PhD degree in computer engineering from Pennsylvania State University in 1993. She joined the faculty at the University of Texas at Austin (UT Austin) in the fall of 1996, where she is currently an associate professor and Engineering Foundation Centennial Teaching Fellow in the Electrical and Computer Engineering Department. Her current research interests are in computer architecture, high-performance microprocessors and computer systems, high-performance memory system, workload characterization, performance evaluation, compiler optimization techniques, reconfigurable computer architectures, etc. She has received several awards, including the Texas Exes teaching award, the UT Austin Engineering Foundation Faculty award, the Halliburton Young Faculty award, and the US National Science Foundation CAREER award. She is a senior member of the IEEE and a member of the IEEE Computer Society.