Characterizing the Unique and Diverse Behaviors in Existing and Emerging General-Purpose and Domain-Specific Benchmark Suites

Kenneth Hoste   Lieven Eeckhout

ELIS Department, Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
Email: {kehoste,leeckhou}@elis.UGent.be

Abstract

Characterizing and understanding emerging workload behavior is of vital importance to ensure next generation microprocessors perform well on their anticipated future workloads. This paper compares a number of benchmark suites from emerging application domains, such as bio-informatics (BioPerf), biometrics (BioMetricsWorkload) and multimedia (MediaBench II), against general-purpose workloads represented by SPEC CPU2000 and CPU2006. Although these benchmark suites have been characterized before, prior work did not capture the benchmark suites' inherent (microarchitecture-independent) behavior, nor did they provide a phase-level characterization.

In this paper, we characterize these existing and emerging general-purpose and domain-specific benchmark suites in terms of their inherent phase-level behaviors. Our characterization methodology identifies prominent phase behaviors across benchmarks and visualizes them in terms of their key microarchitecture-independent characteristics. From this analysis, next to revealing a detailed picture of the distinct phase-level behaviors in these benchmark suites, we also obtain a number of interesting high-level insights. For example, SPEC CPU2006 turns out to be the benchmark suite with the largest workload space coverage, i.e., it covers the largest set of distinct behaviors in the workload space. Also, our analysis provides experimental evidence supporting the intuitive understanding that domain-specific benchmark suites cover a narrower range in the workload space than general-purpose benchmark suites. Finally, the BioPerf bio-informatics benchmark suite exhibits a large fraction unique behaviors not observed in the general-purpose benchmark suites, substantially more so than the other domain-specific benchmark suites, BioMetricsWorkload and MediaBench II.

1 Introduction

Computer design and research heavily rely on a benchmarking methodology in which the performance of the future computer system is evaluated by simulating benchmarks. An important problem in the benchmarking process though is to ensure that the benchmarks are representative — designing a future computer system using yesterday's benchmarks may lead to a suboptimal design for its future workloads [27]. As such, it is extremely important that performance analysts keep pace with the evolving computer workloads as new application domains emerge. This is well recognized, and in response, performance analysts build new benchmark suites to represent future application domains. Given the expanding number of emerging benchmark suites, the number of benchmarks that a researcher or developer needs to consider increases, which in its turn increases simulation time. As such, a methodology that identifies the new behaviors in these emerging benchmark suites would be most helpful in managing the simulation time cost, while not compromising on the representativeness and accuracy of the performance evaluation.

In this paper, we analyze five benchmark suites — BioPerf (bio-informatics), BioMetricsWorkload (biometrics), MediaBench II (multimedia), SPEC CPU2000 and SPEC CPU2006 — and compare their behavioral characteristics. In order to do so, we use a workload characterization methodology that analyzes the benchmarks' behaviors on a per-phase basis using microarchitecture-independent behavioral metrics. The end goal of the methodology is to represent the benchmarks' inherent time-varying behaviors in a comprehensive way: our methodology selects a representative set of distinct phase behaviors from the various benchmarks and characterizes these prominent phase behaviors in terms of their key inherent behavioral characteristics. As a next step, we then study the unique and diverse phase behaviors observed in the existing and emerging benchmark suites.

This paper makes two major contributions:

- We characterize the uniqueness and diversity in phase-level program behavior as observed in existing and emerging benchmark suites and provide a number of novel and interesting insights. The most eye-catching insights from this analysis are that (i) the general-purpose SPEC CPU benchmark suites cover a much broader part of the workload space than the domain-specific benchmark suites, or, in other words, the domain-specific benchmark suites cover a relatively narrow part of the workload space, (ii) there is more diverse behavior to be observed in SPEC CPU2006 than in SPEC CPU2000 and any other benchmark suite, (iii) the BioPerf domain-specific benchmark suite exhibits a high fraction of unique phase-level behaviors not observed in the SPEC CPU benchmark suites, substantially more so than BioMetricsWorkload and MediaBench II.
- We show that phase-level workload characterization is feasible to do in practice, and in addition, is more informative than an aggregate workload characterization approach. The practical phase-level workload characterization methodology that we present in this paper gives an informative picture of the most significant phase behaviors observed across a set of benchmarks.

2 Phase-level characterization methodology

2.1 Motivating phase-level workload characterization

There exists a plethora of approaches to characterizing workloads. Some computer architects and performance analysts characterize workloads using hardware performance counters by running workloads on real hardware. Others use a simulation setup, typically simulating a number of microarchitectures or microarchitecture structures such as caches and/or branch predictors, and subsequently report IPC numbers and/or cache and branch predictor miss rates. Yet others advocate collecting a number of program characteristics that are microarchitecture-independent; example characteristics are instruction mix, inherent ILP, memory footprint sizes, memory access patterns, etc.

Most of these published workload characterization studies present an aggregate characterization, i.e., the characteristics are reported as average numbers across the entire program execution. This can be misleading though. Consider for example the case where a workload characterization study would report that for a given application of interest, 30% of the instructions executed are memory (load or store) instructions. This would make a computer architect conclude that about one third of the functional units being load/store units would suffice for the given program to achieve good performance. However, during the first half of the program execution, there may be only 10% memory instructions executed; and during the second half of the program execution, there may be 50% memory instructions in the dynamic instruction stream. (On average over the entire program execution, this results in 30% memory instructions.) Obviously, one third of load/store units (based on the aggregate analysis) would yield good performance for the first part of the program execution, but would yield sub-optimal and less than expected performance for the second part. A phase-level characterization showing that there are two major program phases each exhibiting different behavioral characteristics would be more accurate and more informative.

Although the notion of time-varying behavior is well known to computer architects, and although there is a lot of recent work done on detecting and exploiting phase behavior, most workload characterization papers are limited to an aggregate workload analysis and do not study time-varying program behavior. The key reason is that a phase-level characterization study involves a very large data set, up to the point where it becomes intractable. For example, in our study which involves five benchmark suites, we deal with over 1 million instruction intervals characterized using 69 program metrics. Obviously, getting insight from such a large data set is far from trivial.

2.2 Overview of the methodology

In this paper, we propose and use a tractable phase-level workload characterization methodology. The general flow of our phase-level characterization process consists of the following steps:

1. We characterize instruction intervals in all of our benchmarks in a microarchitecture-independent manner. In our setup we measure 69 characteristics and consider 100M-instruction intervals.
2. A fixed number of instruction intervals is randomly selected per benchmark across all of its inputs. In our setup we select 1,000 intervals per benchmark.
3. Applying principal components analysis (PCA) reduces the dimensionality of the data set while retaining most of the information.
4. We apply cluster analysis to identify the most representative phase behaviors, i.e., we identify 100 prominent phase behaviors which, collectively, cover most of the phase behaviors observed across the entire set of benchmarks.

category	#	description
instruction mix	10	percentage memory reads, memory writes, branches, arithmetic operations, multiplies, etc.
ILP	4	IPC that can be achieved for an idealized processor (with perfect caches and branch predictor) for a given window size of 32, 64, 128 and 256 in-flight instructions
register traffic	9	average number of register input operands per instruction, average degree od use (number of register reads per register write), distribution (measured in buckets) of the register dependency distance, or, number of instructions between the production and consumption of a register instance
memory footprint	4	number of unique 64-byte blocks and 4KB memory pages touched by the instruction and data stream
data stream strides	28	distribution of global and local strides; global stride is the difference in memory addresses between two consecutive memory accesses; local stride is similar but is restricted to two consecutive memory accesses by the same static instruction; this is measured separately for memory reads and writes; these distributions are measured in buckets
branch predictability	14	average branch transition and taken rate, along with branch misprediction rate for the theoretical PPM predictor [3]; we consider both global and local history predictors, and both per-address and global predictors; we also consider different maximum history lengths (4, 8 and 12)

Table 1. Microarchitecture-independent characteristics.

5. To ease the understanding of the behavior of these prominent phases, a genetic algorithm identifies the key microarchitecture-independent characteristics.

6. We plot all 100 prominent phase behaviors as kiviat diagrams with the axes being these key characteristics, and in addition, we also show in which benchmarks the prominent phase behaviors are observed.

This is a practical methodology that is both computationally feasible, and informative in its results. This methodology scales our prior microarchitecture-independent characterization methodology presented in [11, 12] from an aggregate to a phase-level program analysis methodology. This involves a number of novel steps to the methodology to enable efficiently handling large phase-level data sets. In particular, the novelties are interval sampling (step 2 from above), and trading off coverage versus per-cluster variability to identify prominent phase behaviors (step 4). We will now discuss each step in this methodology in more detail in the following subsections.

2.3 Microarchitecture-independent characterization

Following the observation in [11] that a microarchitecture-dependent characterization can be misleading, we use a microarchitecture-independent approach in order to capture a program's inherent behavior, independent of a particular hardware platform. Table 1 summarizes the 69 microarchitecture-independent characteristics that we measure: instruction mix, inherent ILP, register traffic, memory footprint sizes, memory access strides and branch predictability. We use PIN [20], a dynamic binary instrumentation tool, to collect these characteristics per 100M instruction interval. The MICA (Microarchitecture-Independent Characterization of Applications) PIN tool used for this analysis is available at `http://www.elis.ugent.be/~kehoste/mica/`.

2.4 Interval sampling

As a second step, we sample instruction intervals: we select a fixed number of intervals per benchmark across all of its inputs. In our setup, we select 1,000 instruction intervals per benchmark — for benchmarks with less than 1,000 instruction intervals, this means that instruction intervals will appear multiple times in the data set. This results in 77,000 instruction intervals in our setup — there are 77 benchmarks in total, as we will describe later. The motivation for the interval sampling step is to give equal weight to all the benchmarks in the subsequent analysis steps — without interval sampling, benchmarks with a larger number of inputs and/or a larger dynamic instruction count would get a higher weight in the overall analysis. This is an experimental design choice though: e.g., if the experimenter wants to give equal weight to all benchmark suites, he/she should select a fixed number of intervals per benchmark suite.

2.5 Principal components analysis (PCA)

As a third step, we apply principal components analysis (PCA) [15] to identify the key uncorrelated dimensions in the data set. We then retain the most significant dimensions which reduces the dimensionality of the data set.

The input to PCA is a matrix in which the rows are the instruction intervals and the columns the microarchitecture-independent characteristics. PCA computes new variables, called *principal components*, which are linear combinations of the microarchitecture-independent characteristics, such that all principal components are uncorrelated. In other words, PCA tranforms the p microarchitecture-independent characteristics $X_1, X_2, \ldots, X_p$ into p principal components $Z_1, Z_2, \ldots, Z_p$ with $Z_i = \sum_{j=1}^{p} a_{ij} X_j$. This transformation has the properties (i) $Var[Z_1] \geq Var[Z_2] \geq \ldots \geq Var[Z_p]$ — this means Z_1 contains the most information and Z_p the least; and (ii) $Cov[Z_i, Z_j] = 0, \forall i \neq j$ — this means there is no information overlap between the principal components.

Some principal components have a higher variance than others. By removing the principal components with the lowest variance from the analysis, we reduce the dimensionality of the data set while controlling the amount of information that is thrown away. We retain all principal components with a standard deviation greater than one; for our data set, this means 13 principal components which collectively explain 85.4% of the total variance in the data set.

In PCA, one can either work with a normalized or a non-normalized input data set — the data set is normalized when the mean of each variable is zero and its variance is one. In a workload characterization method as presented here, it is appropriate to normalize the data set prior to PCA to put all characteristics on a common scale. Also after PCA, it is appropriate to normalize the data set to put all principal components on a common scale — the principal components represent underlying program characteristics and we want to give equal weight to all underlying program characteristics. We call the resulting space the *rescaled PCA space*.

2.6 Clustering

The fourth step in our workload analysis is to apply cluster analysis (CA) [15] in the rescaled PCA space. The k-means clustering algorithm is an iterative process that first randomly selects k cluster centers, and then works in two steps per iteration. The first step is to compute the distance of each point in the multi-dimensional space to each cluster center. In the second step, each point gets assigned to the closest cluster. As such, new clusters are formed and new cluster centers are to be computed. This algorithm is iterated until convergence is observed, i.e., cluster membership ceases to change across iterations.

We evaluate a number of randomly chosen initial cluster centers and then retain the clustering with the highest *Bayesian Information Criterion (BIC)* score. The BIC score is a measure that trades off goodness of fit of the clustering to the given data set versus the number of clusters. The end result of this clustering algorithm is k clusters with the highest BIC score. For each cluster we then select the instruction interval that is closest to the cluster center as the cluster representative. Each cluster representative then gets a weight assigned that is proportional to the number of instruction intervals it represents.

The final goal of this clustering step is to yield a limited number of representative phases that collectively cover a sufficiently large fraction of the total entire set of benchmarks. This involves making a trade-off between coverage and variability within a cluster. To illustrate this, consider the following example. Say the ultimate goal for the workload characterization study is to come up with 100 prominent phases. One option would be to apply k-means clustering with $k = 100$; this will yield 100 prominent phases with a 100% coverage, i.e., all instruction intervals in the data set will be represented by a phase representative. Another option is to apply k-means clustering for $k > 100$; in this case, the 100 most prominent phases will account for less than 100% coverage. However, the variability within each cluster will be substantially smaller than for the $k = 100$ case. In other words, we can select the most prominent phase behaviors that collectively account for a large fraction of the entire benchmark suite while minimizing the variability represented by each prominent phase. In our setup, we set $k = 300$ and select 100 prominent phases; these 100 prominent phases collectively cover 87.8% of the entire set of benchmarks.

2.7 Genetic algorithm

After the clustering algorithm, we are left with a number of prominent phase behaviors. The end goal now is to visualize these phase behaviors in terms of their key program characteristics. This is challenging though. Visualizing a phase behavior as a point in a 69-dimensional space (in terms of all the microarchitecture-independent characteristics), is difficult and likely not very helpful. One solution may be to plot these phase behaviors in terms of the most significant principal components; the greatly reduced dimensionality will facilitate the understanding of the data set. However, the problem with a characterization based on principal components is that interpreting the principal components may be difficult, although the dimensionality is much smaller than the original data set. The principal components are linear combinations of the original characteristics which complicates intuitive reasoning. Instead we use a genetic algorithm, following [11, 12], to select a limited number of key microarchitecture-independent characteristics. We then visualize the prominent phase behaviors in terms of these key characteristics.

A genetic algorithm is an evolutionary optimization method that starts from populations of solutions. For each solution in each population, a fitness score is computed and the solutions with the highest fitness score are selected for constructing the next generation. This is done by applying mutation, crossover and migration on the selected solutions from the previous generation. Mutation randomly changes a single solution; crossover generates new solutions by mixing existing solutions; and migration allows solutions to switch populations. This algorithm is repeated, i.e., new generations are constructed, until no more improvement is observed for the fitness score.

A solution in our case, is a series of 69 0's and 1's, one per microarchitecture-independent characteristic. A '1' selects a program characteristic and a '0' excludes a program characteristic. The fitness score equals the Pearson corre-

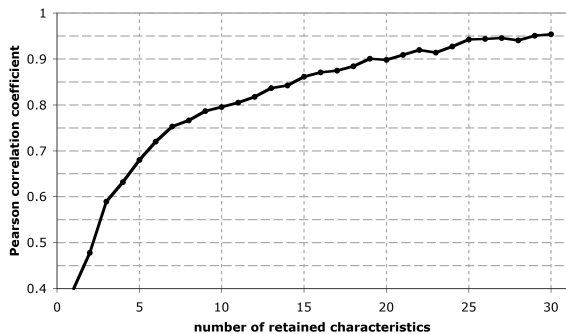

Figure 1. Correlation coefficient of the distance in the workload space built from the retained characteristics through the genetic algorithm versus the distance in the workload space built from all characteristics.

1	average branch transition rate
2	branch misprediction rate for the GAs PPM predictor with 4-bit history
3	percentage string instructions
4	percentage shift instructions
5	instruction memory footprint at 64-byte block level
6	data memory footprint at 64-byte block level
7	probability for a local store stride $\leq 32K$
8	probability for a local store stride ≤ 64
9	probability for a global load stride $\leq 256K$
10	probability for a global load stride ≤ 64
11	average degree of use of register values
12	average number of register operands

Table 2. The microarchitecture-independent characteristics retained by the genetic algorithm.

lation coefficient of the distances between the prominent phases in the original data set (with all microarchitecture-independent characteristics) versus the distances between the prominent phases in a reduced data set with only the selected microarchitecture-independent characteristics, i.e., only the characteristics with a '1' assigned are included in the reduced data set. Computing the distance in the original data set as well as in the reduced data set is done through PCA, i.e., we normalize both data sets, apply PCA on both data sets, retain the principal components with a variance greater than one, normalize the principal components and finally compute the distances between the prominent phases in the rescaled PCA spaces. The reason for this additional PCA step when computing the distance is to discount the correlation between program characteristics in the data set from the distance measure. The end result of the genetic algorithm is a limited number of program characteristics that allow for an accurate and intuitive microarchitecture-independent program characterization of the most prominent phase behaviors.

It is up to the experimenter to determine how many microarchitecture-independent characteristics to retain. This can be done based on the desired correlation coefficient of the distances in the reduced dimensionality space versus the distances in the original space. Figure 1 shows this correlation coefficient as a function of the number of selected characteristics. We pick 12 key microarchitecture-independent characteristics which result in a 0.82 correlation coefficient. These 12 key microarchitecture-independent characteristics are shown in Table 2. The key microarchitecture-independent characteristics include a range of behavioral properties covering instruction mix, branch predictability, register traffic, memory footprint and memory access patterns.

2.8 Kiviat plots

Our final step is to visualize the prominent phase behaviors in terms of their key microarchitecture-independent characteristics. To this end, we use kiviat plots which are helpful to visualize multidimensional data. In addition to these kiviat plots, we also show pie charts displaying which benchmarks are represented by each phase behavior and by what fraction. Together, the kiviat plots and the pie charts, give an insightful understanding of the prominent phase behaviors and how they represent the various benchmarks in the workload.

2.9 Discussion

The phase-level characterization methodology presented in this paper requires a number of settings to be determined in the various steps of the methodology, such as determining how many principal components to retain, how many prominent phases to select, and how many key microarchitecture- independent characteristics to select through the genetic algorithm for building the kiviat plots. Basically, these parameter settings trade off coverage versus accuracy. We made such a trade-off in our experiments when boiling down more than 1 million intervals to 100 representative intervals, however, making the appropriate parameter settings is up to the experimenter when making the coverage/accuracy trade-off.

Note also that in this paper, we consider 100M instruction intervals, however, our methodology can be applied to any interval granularity of interest, and could even be applied to variable-length intervals. A smaller instruction interval size would result in a more fine-grain phase-level characterization; a larger instruction interval size would result in a more coarse-grain characterization. We use 100M-instruction intervals because this interval size is an appropriate interval size for detailed simulation. At the 100M-instruction interval granularity (and larger), microarchitecture state warmup is less of an issue [24]. This is also the

suite	benchmark	cnt
BioMetricsWorkload	face	13,534
	finger	27,296
	gait	3,718
	hand	10,789
	speak	2,847
BioPerf	blast	1,902
	ce	42
	clustalw	2,709
	fasta	169,911
	glimmer	338
	grappa	43,013
	hmmer	5,330
	phylip	20,172
	predator	7,127
	t-coffee	2,741
MediaBench II	h263	224
	h264	1,505
	jpeg2000	14
	jpeg	2
	mpeg2	177
	mpeg4	23
	mpeg4-mmx	8

suite	benchmark	cnt
SPECint2000	bzip2	2,871
	crafty	1,852
	eon	2,047
	gap	2,220
	gcc	1,298
	gzip	3,215
	mcf	529
	parser	3,353
	perlbmk	3,318
	twolf	2,843
	vortex	2,963
	vpr	2,076
SPECfp2000	ammp	3,578
	applu	3,495
	apsi	4,548
	art	1,516
	equake	1,525
	facerec	3,366
	fma3d	3,121
	galgel	3,689
	lucas	2,458
	mesa	2,882
	mgrid	4,822
	sixtrack	7,042
	swim	2,285
	wupwise	4,286

suite	benchmark	cnt
SPECint2006	astar	13,530
	bzip2	23,144
	gcc	11,739
	gobmk	16,927
	h264ref	36,123
	hmmer	31,765
	libquantum	39,490
	mcf	3,782
	omnetpp	7,704
	perlbench	23,056
	sjeng	23,531
	xalancbmk	12,348
SPECfp2006	bwaves	21,826
	cactusADM	30,466
	calculix	74,593
	dealII	22,701
	gamess	56,550
	gemsFDTD	19,242
	gromacs	25,597
	lbm	18,455
	leslie3d	17,387
	milc	12,150
	namd	31,371
	povray	12,234
	soplex	8,292
	sphinx3	30,246
	tonto	35,062
	wrf	31,737
	zeusmp	22,285

Table 3. The benchmarks used in this paper, along with the number of 100M instruction intervals.

reason why industry uses relatively large interval sizes, see for example the Intel PinPoints approach [22].

3 Experimental setup

In our experimental setup we consider five benchmark suites, namely SPEC CPU2000 (ref), SPEC CPU2006 (ref), BioPerf (medium) [1], BioMetricsWorkloads (s100) [4] and MediaBench II [19]. There are 77 benchmarks in total, and 1,103,953 100M-instruction intervals in total (or over 110 trillion dynamically executed instructions), see also Table 3. All of these benchmarks are compiled using the Intel compiler kit version 9.1.051 on a 32-bit Linux/x86 Intel Pentium 4 machine with the `-static -O2` compiler flags; BioMetricsWorkload's gait and SPEC CPU2000's gap were compiled using the GNU C compiler v4.1.2 with `-O2` because of compilation problems with the Intel compiler on our platform.

4 Phase-Level Characterization of Existing and Emerging Benchmark Suites

Before analyzing the coverage, uniqueness and diversity of the benchmark suites, we first discuss the fine-grain phase-level results that are obtained from our methodology. In addition, we also discuss how these results enable gaining insight into individual phase behaviors.

4.1 Workload visualization

Figures 2 and 3 show the 100 most prominent phases when applying the proposed phase-level workload characterization methodology on the benchmark suites considered in this paper.

Each prominent phase is characterized by the following:

- **Cluster weight.** The weight that each cluster has in the overall analysis is shown above each cluster representation. For example, the top-left cluster representation in Figure 2 covers 4.87% of the overall analysis, i.e., 4.87% of the total execution of all the benchmarks is represented by this cluster. These cluster weights sum up to 87.8%, which is the coverage of the 100 prominent phases in the entire data set.
- **Kiviat plot.** The axes of the kiviat plots represent the twelve key microarchitecture-independent characteristics, see the legend shown at the bottom of Figure 2. The various rings within a kiviat plot represent the mean value minus one standard deviation, the mean value, and the mean value plus one standard deviation along each dimension; the center point and the outer ring represent the minimum and maximum, respectively[1]. The prominent phase behaviors are characterized by connecting their key characteristics to form a

[1]The maximum value can be smaller than the mean value plus one standard deviation, and likewise, the minimum value can be larger than the mean value minus one standard deviation, as is the case for some characteristics, see the legend of Figure 2.

dark gray area; this dark gray area visualizes a phase's inherent behavior.

- **Benchmark list.** Next to each kiviat plot we also show a box with a list of the benchmarks that are represented by the given cluster. The percentage between brackets denotes the fraction of the benchmark's execution that is represented by the given cluster. For example, see the top-left cluster representation in Figure 2, 31.56% of Fasta's execution is represented by the given cluster. The benchmarks for which this fraction is less than 2% are grouped under the 'other' label.

- **Pie chart.** The pie chart next to each kiviat plot shows the weight that each represented benchmark has in the cluster. For example, the top-left cluster representation in Figure 2 shows that Fasta accounts for 100% of the cluster, i.e., there are no other benchmarks that it represents. As another example, the bottom-right cluster representation in Figure 3 shows that GemsFDTD accounts for 37% of the cluster, while soplex accounts for 61%, and 6 other benchmarks account for 2% collectively.

4.2 Getting insight

This visualization provides a number of interesting insights. To facilitate the discussion and to illustrate the level of insight these kiviat plots can provide, we organized the cluster representations in Figures 2 and 3 along three groups. We make a distinction between *benchmark-specific* clusters (clusters which represent phase behaviors for a single benchmark), *suite-specific* clusters (clusters which represent phase behaviors from multiple benchmarks from a single benchmark suite), and *mixed* clusters (clusters which represent phase behaviors from multiple benchmarks from multiple benchmark suites).

The benchmark-specific clusters represent unique behaviors not observed in other benchmarks. The BioPerf and SPEC CPU2006 benchmark suites, and to a lesser extent also the SPECfp2000 and BioMetricsWorkload benchmark suites, exhibit a number of unique behaviors (see Figure 2) and the kiviat plots provide insight into why these behaviors are unique. For example, most of BioPerf's Grappa execution exhibits a large number of string operations along with a large number of global small-distance strides (smaller than 64).

These kiviat plots also present an interesting view on per-benchmark phase behavior. For example, SPEC CPU2006's astar seems to be partitioned across two prominent phase behaviors (see the top-left cluster representation in the SPECint2006 subgroup of Figure 2 and the rightmost cluster representation on the fourth row of the mixed clusters group in Figure 3). In particular, 23.22% and 62.08% of the dynamic instruction stream of astar is represented by these clusters, respectively. The kiviat plots provide an easy-to-understand view on how these phase behaviors differ from each other: the mixed-cluster phase in astar shows significantly better data locality (see the global load stride probabilities) and better branch predictability, along with a smaller transition rate compared to the benchmark-specific phase. In fact, the benchmark-specific phase in astar shows the worst branch predictability overall.

The mixed clusters show prominent phase behaviors observed across various benchmarks (from various benchmark suites); in general, these mixed clusters represent more average behavior than the benchmark-specific and suite-specific clusters. One interesting mixed cluster example is the 'hmmer' cluster which includes the SPEC CPU2006 integer hmmer benchmark and the BioPerf Hmmer benchmark (see the rightmost cluster representation on the second row of the mixed clusters group in Figure 3 and the Hmmer cluster in the BioPerf benchmark-specific group in Figure 2, respectively). A major part (68.06%) of the SPEC CPU2006 version resembles only a small part (5.07%) of the BioPerf version. The remaining major part (59.44%) of the BioPerf version shows dissimilar behavior compared to the SPEC CPU2006 version in terms of branch predictability and number of register operands, see Figure 2.

5 Comparing the Uniqueness and Diversity of Benchmark Suites

We now use the results obtained from the phase-level characterization to study the coverage, diversity and uniqueness of the various benchmark suites — for this purpose, we now consider all 300 phase-level behaviors, not just the 100 most prominent phase behaviors as done in the previous section. Subsequently, we then discuss the implications of these observations for simulation-based performance evaluation.

5.1 Coverage and diversity

As a first step, we quantify a benchmark suite's *workload space coverage*, or how much a benchmark suite covers the entire workload space. Figure 4 shows the number of clusters (out of the 300) that represent some part of the benchmark suite, e.g., 136 and 158 of the clusters represent (at least one interval from) SPECint2006 and SPECfp2006, respectively. We observe that both the SPEC CPU2000 and SPEC CPU2006 benchmark suites cover the largest part of the workload space; and CPU2006 does even more so than CPU2000, for both the integer and floating-point benchmarks. This reflects SPEC CPU's property of being a general-purpose application benchmark suite. The emerging benchmark suites, BioPerf, BioMetricsWorkload and

benchmark-specific clusters

BioPerf

weight: 4.87% Fasta: 31.56%
weight: 3.46% Fasta: 22.43%
weight: 1.19% Grappa: 30.54%
weight: 0.95% Grappa: 24.45%
weight: 0.66% Grappa: 16.89%
weight: 0.64% Grappa: 16.32%
weight: 0.43% Grappa: 10.93%
weight: 0.29% $Hmmer_{BioPerf}$: 59.44%
weight: 1.40% Phylip: 76.35%
weight: 0.35% Predator: 53.51%
weight: 0.20% Predator: 31.14%

SPECfp2000

weight: 0.24% ammp: 73.39%
weight: 0.29% mgrid: 65.84%
weight: 0.63% sixtrack: 98.67%

SPECint2006

weight: 0.29% astar 23.22%
weight: 0.39% $bzip2_{int2006}$: 18.72%
weight: 0.60% gobmk: 38.77%
weight: 0.47% gobmk: 30.89%
weight: 0.33% gobmk: 21.33%
weight: 1.68% libquantum: 46.76%
weight: 1.14% libquantum: 31.92%
weight: 2.13% sjeng: 99.79%
weight: 0.61% xalancbmk: 54.57%

SPECfp2006

weight: 1.55% bwaves: 78.48%
weight: 0.24% bwaves: 11.97%
weight: 2.75% cactusADM: 99.49%
weight: 3.99% calculix: 58.96%
weight: 1.51% calculix: 22.34%
weight: 0.93% calculix: 13.78%
weight: 0.35% dealII: 16.98%
weight: 0.30% dealII: 14.44%
weight: 0.25% dealII: 12.34%
weight: 1.11% gamess: 21.67%
weight: 0.96% gamess: 18.80%
weight: 0.94% gromacs: 40.46%
weight: 0.69% gromacs: 29.65%
weight: 1.66% lbm: 99.39%
weight: 0.47% milc: 42.17%
weight: 1.96% namd: 68.17%
weight: 0.45% zeusmp: 22.28%
weight: 0.35% zeusmp: 17.07%

BMW

weight: 1.13% face: 92.10%
weight: 0.30% hand: 30.99%

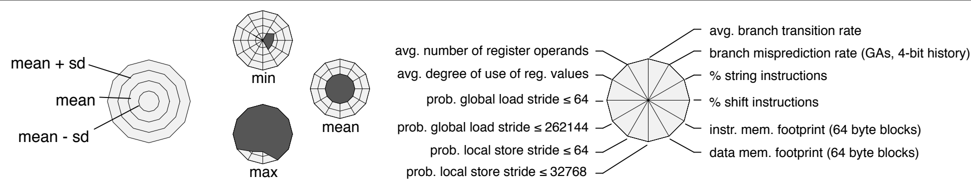

Figure 2. Kiviat plots (part I) presenting the phase-level workload characterization.

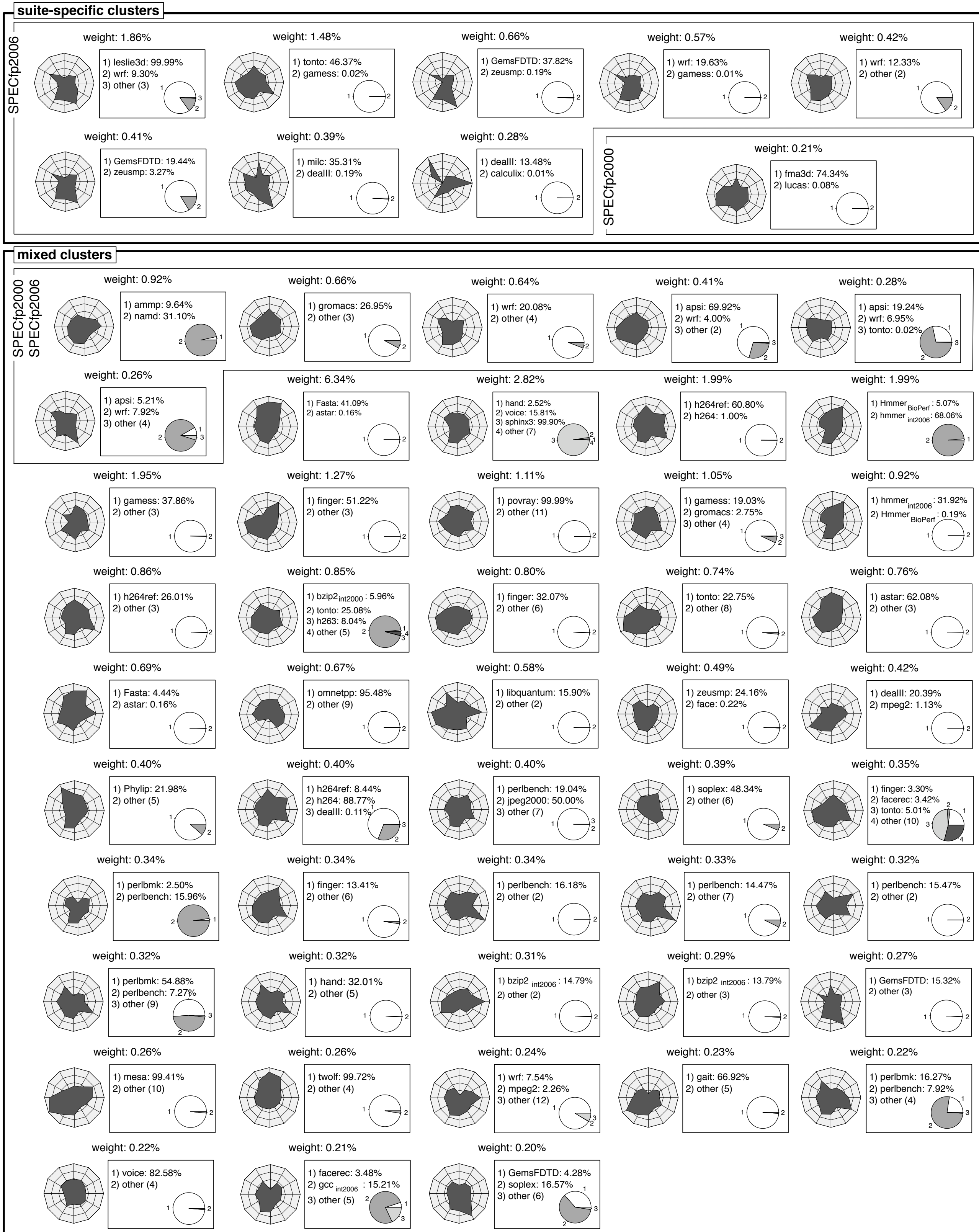

Figure 3. Kiviat plots (part II) presenting the phase-level workload characterization.

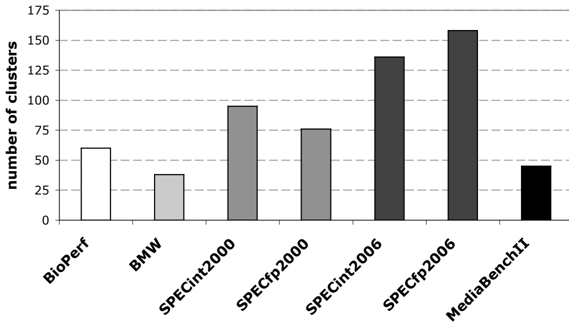

Figure 4. Workload space coverage per benchmark suite.

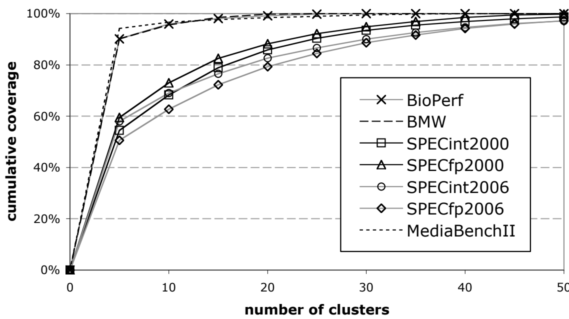

Figure 5. Cumulative coverage per benchmark suite as a function of the number of clusters.

MediaBench II, cover a much narrower part of the workload space, reflecting the fact that these benchmark suites are tied to a specific application domain.

As a subsequent step, we quantify the *diversity within a benchmark suite*. This is done by computing the cumulative number of clusters needed to represent a given fraction of the given benchmark suite. The results are shown in Figure 5: the cumulative coverage is shown per benchmark suite as a function of the number of clusters. For example, this graph shows that about 20 clusters are required to cover 80% of the SPECfp2006 benchmark suite; or, only 5 clusters are required to cover 90% of the BioPerf benchmark suite. The lower the curve for a given benchmark suite, the more clusters are required to cover a given percentage of the entire benchmark suite, and thus the higher the diversity. We observe that the domain-specific benchmark suites show a relatively low diversity compared to the general-purpose benchmark suites.

We thus conclude from this section that BioPerf, BioMetricsWorkload and MediaBench II cover a much narrower part of the workload space than SPEC CPU, and in addition, the number of distinct behaviors within these benchmark suites is much smaller than for SPEC CPU. This analysis thus provides experimental evidence for the intuitive understanding that domain-specific benchmark suites represent a smaller part of the workload space than general-purpose benchmark suites do.

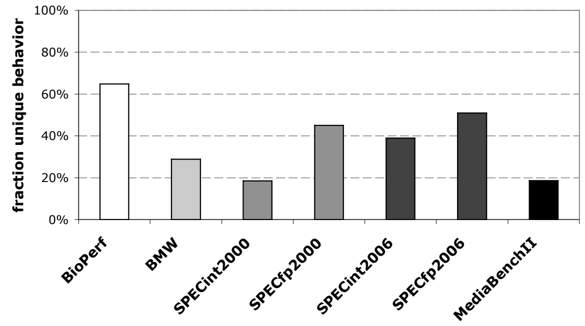

Figure 6. Fraction of a benchmark suite that represents unique program behavior not observed in the other benchmark suites.

5.2 Uniqueness

We now quantify a benchmark suite's *uniqueness* with respect to the other benchmark suites. To do so, we compute the fraction of a given benchmark suite that is represented by clusters that contain data for the given benchmark suite only, see Figure 6. For example, 65% of the BioPerf benchmark suite execution is represented by either benchmark-specific or suite-specific clusters. In other words, 65% of the BioPerf benchmark suite exhibit unique program behavior not observed in other benchmark suites. This is the highest fraction observed among the benchmark suites analyzed. Also SPEC CPU2006 exhibits a fairly large fraction of unique program behavior, namely 39% and 51% for SPECint2006 and SPECfp2006, respectively. The floating-point SPEC CPU benchmark suites exhibit more unique behavior than the integer SPEC CPU benchmark suites, for both CPU2000 and CPU2006. The other domain-specific benchmark suites, MediaBench II and BioMetricsWorkload, represent substantially less unique program behavior with fractions of unique program behavior, namely 19% and 29%, respectively.

5.3 Implications

The results obtained in the previous sections present a number of implications to performance evaluation. First, since SPEC CPU2006 exhibits a slightly larger diversity than its predecessor (see Figure 5), this implies that only a slightly larger number of representative samples or sim-

ulation points need to be simulated for CPU2006 as for CPU2000 in order to cover all major phase-level behaviors in the benchmark suite. Second, BioPerf shows a large fraction unique behavior not observed in the other benchmark suites. For this reason, BioPerf is a good candidate benchmark suite for performance evaluation, i.e., because the MediaBench II and BioMetricsWorkload benchmark suites represent much less unique behaviors than CPU2006 and BioPerf, in case one is pressed on simulation time, it may not be worth the effort to simulate MediaBench II and BioMetricsWorkload; the additional insights may not pay off the additional simulation time.

6 Related work

A large body of related work has been done which we summarize in this section.

6.1 Phase behavior

It is well known that a computer program typically goes through a number of phases during its execution. A program phase is characterized by its relatively homogeneous behavior within the phase while showing distinct behavior across phases. Different approaches have been proposed in the recent literature to identify program phases. Duesterwald et al. [7] identify and predict program phase behavior based on microarchitecture-dependent characteristics such as IPC, cache miss rates, branch misprediction rates, etc. Other researchers characterize program phase behavior in a microarchitecture-independent manner. The advantage of a microarchitecture-independent characterization is that it applies across different microarchitectures — this is especially valuable when using program phase behavior to drive software or hardware optimization. Several approaches to characterizing program phase behavior in a microarchitecture-independent manner have been proposed. Dhodapkar and Smith [5] track the instruction footprint to detect program phase transitions. Sherwood et al. [24] compute Basic Block Vectors (BBV) — a BBV computes the number of times a basic block has been executed in a given instruction interval. Lau et al. [17] found that BBVs correlate strongly with performance characteristics. Huang et al. [14] relate phase behavior to methods and loops being executed. Yet other papers identify phase behavior based on other microarchitecture-independent characteristics such as memory access patterns [18, 23] or a wide variety of program characteristics [8] — the advantage of these approaches over instruction footprints and BBVs is that they can be used to compare phase behaviors across benchmarks.

Various researchers have proposed to exploit phase behavior for a variety of applications. One application to phase analysis is hardware adaptation for energy saving [2, 6, 14, 25]. The idea there is to adapt the hardware on a per-phase basis so that energy consumption is reduced while not affecting overall performance. Another application is software profiling and optimization [10, 21]. Yet another application is simulation acceleration [8, 22, 24] by picking and simulating only one representative simulation point per phase. SimPoint [22, 24] identifies simulation points by analyzing program behavior within a single benchmark. Eeckhout et al. [8] identify simulation points across benchmarks by exploiting phase-level program similarity across benchmarks — this results in a smaller overall number of representative phases than using SimPoint.

6.2 Workload characterization

Many workload characterization studies characterize workload behavior in terms of microarchitecture-dependent metrics such as IPC, cache miss rates, branch misprediction rates, etc. To do so, most of these studies employ hardware performance counters or rely on simulation [26]. We advocate characterizing workload behavior in terms of microarchitecture-independent workload characteristics, see also our prior work [11, 12, 13, 16]. These characteristics can then serve as input to a PCA-based workload analysis methodology to select a limited set of representative benchmarks to represent a much broader spectrum of benchmarks [8, 9, 16]. Such a methodology has a number of applications, such as program behavior characterization [11], simulation time reduction [8], benchmark suite composition [9], performance prediction [13] and analyzing benchmark drift [16]. This paper differs from this prior work in its aim at visualizing microarchitecture-independent program behavior at the phase level in an easy-to-understand manner. There are two primary reasons motivating this approach. For one, a phase-level characterization provides more information compared to an aggregate analysis. Second, the characterization data at the phase level is valuable as a complement to simulation results to gain insight into how inherent program characteristics affect performance — because simulating a complete benchmark execution is too time-consuming, simulation is done at the phase level anyway.

7 Conclusion

Phase-level workload characterization is more informative than aggregate program analysis because it provides insight into a program's time-varying behavior. This paper described a methodology to characterize general-purpose and domain-specific benchmark suites at the phase level in a microarchitecture-independent manner. The key feature of this phase-level workload characterization methodology is that it gives insight into the inherent behavior of the most

prominent phase behaviors across a set of benchmarks. Applying this methodology to characterize existing and emerging benchmark suites (SPEC CPU2000, SPEC CPU2006, BioPerf, BioMetricsWorkload and MediaBench II) leads to a number of interesting conclusions. For one, SPEC CPU2006 has a larger workload space coverage than its predecessor, SPEC CPU2000, and any of the other benchmark suites included in our setup. Second, the domain-specific benchmark suites cover a narrower subspace in the workload space than general-purpose benchmark suites. Third, the BioPerf domain-specific benchmark suite exhibits a large fraction unique phase-level behaviors, more so than any other benchmark suite included in this work.

Acknowledgements

We would like to thank the anonymous reviewers for their thoughtful comments and valuable suggestions. Kenneth Hoste is supported by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). Lieven Eeckhout is a Postdoctoral Fellow with the Fund for Scientific Research – Flanders (Belgium) (FWO Vlaanderen). This work is also supported in part by the FWO projects G.0160.02 and G.0255.08, and the HiPEAC Network of Excellence.

References

[1] D. A. Bader, Y. Li, T. Li, and V. Sachdeva. BioPerf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. In *IISWC*, pages 163–173, Oct. 2005.

[2] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *MICRO*, pages 245–257, Dec. 2000.

[3] I. K. Chen, J. T. Coffey, and T. N. Mudge. Analysis of branch prediction via data compression. In *ASPLOS*, pages 128–137, Oct. 1996.

[4] C.-B. Cho, A. V. Chande, Y. Li, and T. Li. Workload characterization of biometric applications on pentium 4 microarchitecture. In *IISWC*, pages 76–86, Oct. 2005.

[5] A. Dhodapkar and J. E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *ISCA*, pages 233–244, May 2002.

[6] A. S. Dhodapkar and J. E. Smith. Comparing program phase detection techniques. In *MICRO*, pages 217–227, Dec. 2003.

[7] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *PACT*, pages 220–231, Oct. 2003.

[8] L. Eeckhout, J. Sampson, and B. Calder. Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In *IISWC*, pages 2–12, Oct. 2005.

[9] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Workload design: Selecting representative program-input pairs. In *PACT*, pages 83–94, Sept. 2002.

[10] A. Georges, D. Buytaert, L. Eeckhout, and K. De Bosschere. Method-level phase behavior in Java workloads. In *OOPSLA*, pages 270–287, Oct. 2004.

[11] K. Hoste and L. Eeckhout. Comparing benchmarks using key microarchitecture-independent characteristics. In *IISWC*, pages 83–92, Oct. 2006.

[12] K. Hoste and L. Eeckhout. Microarchitecture-independent workload characterization. *IEEE Micro*, 27(3):63–72, May 2007.

[13] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. De Bosschere. Performance prediction based on inherent program similarity. In *PACT*, pages 114–122, Sept. 2006.

[14] M. Huang, J. Renau, and J. Torrellas. Positional adaptation of processors: Application to energy reduction. In *ISCA*, pages 157–168, June 2003.

[15] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, fifth edition, 2002.

[16] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John. Measuring benchmark similarity using inherent program characteristics. *IEEE Transactions on Computers*, 55(6):769–782, June 2006.

[17] J. Lau, J. Sampson, E. Perelman, G. Hamerly, and B. Calder. The strong correlation between code signatures and performance. In *ISPASS*, pages 236–247, Mar. 2005.

[18] J. Lau, S. Schoenmackers, and B. Calder. Structures for phase classification. In *ISPASS*, pages 57–67, Mar. 2004.

[19] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *MICRO*, pages 330–335, Dec. 1997.

[20] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *PLDI*, pages 190–200, June 2005.

[21] P. Nagpurkar, C. Krintz, and T. Sherwood. Phase-aware remote profiling. In *CGO*, pages 191–202, Mar. 2005.

[22] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. In *MICRO*, pages 81–93, Dec. 2004.

[23] X. Shen, Y. Zhong, and C. Ding. Locality phase prediction. In *ASPLOS*, pages 165–176, Oct. 2004.

[24] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, pages 45–57, Oct. 2002.

[25] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *ISCA*, pages 336–347, June 2003.

[26] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *HPCA*, pages 281–291, Feb. 2003.

[27] J. J. Yi, H. Vandierendonck, L. Eeckhout, and D. J. Lilja. The exigency of benchmark and compiler drift: Designing tomorrow's processors with yesterday's tools. In *ICS*, pages 75–86, June 2006.