Wait of a Decade: Did SPEC CPU 2017 Broaden the Performance Horizon?

Reena Panda†, Shuang Song†, Joseph Dean, Lizy K. John
The University of Texas at Austin
reena.panda@utexas.edu, songshuang1990@utexas.edu, jd45664@utexas.edu, ljohn@ece.utexas.edu

***Abstract*—The recently released SPEC CPU2017 benchmark suite has already started receiving a lot of attention from both industry and academic communities. However, due to the significantly high size and complexity of the benchmarks, simulating all the CPU2017 benchmarks for design trade-off evaluation is likely to become extremely difficult. Simulating a randomly selected subset, or a random input set, may result in misleading conclusions. This paper analyzes the SPEC CPU2017 benchmarks using performance counter based experimentation from seven commercial systems, and uses statistical techniques such as principal component analysis and clustering to identify similarities among benchmarks. Such analysis can reveal benchmark redundancies and identify subsets for researchers who cannot use all benchmarks in pre-silicon design trade-off evaluations.**

Many of the SPEC CPU2006 benchmarks have been replaced with larger and complex workloads in the SPEC CPU2017 suite. However, compared to CPU2006, it is unknown whether SPEC CPU2017 benchmarks have different performance demands or whether they stress machines differently. Additionally, to evaluate the balance of CPU2017 benchmarks, we analyze the performance characteristics of CPU2017 workloads and compare them with emerging database, graph analytics and electronic design automation (EDA) workloads. This paper provides the first detailed analysis of SPEC CPU2017 benchmark suite for the architecture community.

I. Introduction

Since its formation in 1988, SPEC has carefully chosen benchmarks from real world applications and periodically distributed these benchmarks to the semiconductor community. The last SPEC CPU benchmark suite was released in 2006 and has been widely used by industry & academia to evaluate the quality of processor designs. In the last 10+ years, the processing landscape has undergone a significant change. For instance, the size of processor components (caches, branch predictors, TLBs, etc.) and memory have increased significantly. To keep pace with technological advances and emerging application domains, the 6^{th} generation of SPEC CPU benchmarks have just been released.

The SPEC CPU2017 suite [1] consists of 43 benchmarks, separated into 4 sub-suites, corresponding to "rate" and "speed" versions of the integer and floating point programs (summarized in Table I). Benchmarks in CPU2017 have up to $\sim$10X higher dynamic instruction counts than those in CPU2006; such an increase in the program size is bound to exacerbate the simulation time problem on detailed performance simulators [2], [3], [4], [5], [6]. To keep the simulation times manageable, researchers often use a subset of the benchmarks. However, arbitrarily selected subsets can result in misleading conclusions. Understanding program behavior and their similarities can help in selecting benchmarks to represent target workload spaces. In this paper, we first conduct a detailed characterization of the CPU2017 benchmarks using performance counter based experimentation from several state-of-the-art systems and extract critical insights regarding the micro-architectural bottlenecks of the programs. Next, we leverage statistical techniques such as Principal Component Analysis (PCA) and clustering analysis to understand the (dis)similarity of benchmarks and identify redundancies in the suite. We demonstrate that using less than one-third of the benchmarks can predict the performance of the entire suite with $\geq$93% accuracy.

For the first time, SPEC has also provided separate "speed" and "rate" versions of benchmarks (see Table I) in their CPU suite. SPECspeed always runs one copy of each benchmark, and SPECrate runs multiple concurrent copies of each benchmark. We observe that the CPU2017 speed benchmarks have up to 8x higher instruction counts than their rate equivalents. SPEC's web page indicates that such benchmarks differ in terms of the workload sizes, compilation flags, etc. However, are they truly different in the performance spectrum? Our analysis indicates that most benchmarks (except a few cases, e.g., *imagick*, $fotonik3d$) have very similar performance characteristics between the rate and speed versions.

SPEC CPU2006 benchmarks [7] have long been the de facto benchmark for studying single-threaded performance. The SPEC CPU2017 benchmark suite has replaced many of the benchmarks in the SPEC CPU2006 suite with larger and more complex workloads; compared to the CPU2006 programs, it is not known whether the CPU2017 workloads have different performance demands or whether they stress machines differently. How much of the performance spectrum is lost due to benchmark removal? Do the newly added benchmarks expand the performance spectrum? We perform a detailed comparison between the two suites to identify key differences in terms of performance and power consumption.

While CPU2017 suite has introduced or expanded several application domains (e.g., artificial intelligence), many application domains have been removed (e.g., speech recognition, electronic design automation) or not included (e.g., graph analytics). We further investigate the application domain balance and coverage of the CPU2017 benchmarks using statistical techniques. Specifically, we explore whether the CPU2017 workloads have performance features that can exercise computer systems in a similar manner as emerging data-serving

†Both authors made equivalent contribution to this work.

and graph analytics workloads.

The rest of this paper is organized as follows. Section II gives an overview of the CPU2017 benchmarks and analyzes their micro-architectural performance. Section III discusses the methodology used to measure program (dis)similarity. Section IV proposes representative subsets and input sets of the programs. Section V evaluates balance in the CPU2017 suite. Finally, we discuss related work and conclude in Sections VI and VII, respectively.

II. CPU2017 Benchmarks: Overview & Characterization

In this section, we will first provide an overview of the CPU2017 benchmarks. We will also characterize their micro-architectural behvaior, while focusing on single-core CPU performance. This characterization is performed on an Intel Skylake machine (3.4 GHz, i7-6700 processor, 8MB last-level cache) running Ubuntu 14.04. Benchmarks are compiled using gcc compiler with SPEC recommended optimization flags. The performance counter measurements are carried out using the Linux perf [8] tool.

A. Benchmark Overview

Unlike its predecessors, the CPU2017 suite [1] is divided into four categories: speed integer (SPECspeed INT), rate integer (SPECrate INT), speed floating point (SPECspeed FP) and rate floating point (SPECrate FP), as shown in Table I. The SPECspeed INT, SPECspeed FP and SPECrate INT groups consist of 10 benchmarks each, while the SPECrate FP group consists of 13 benchmarks. In addition, the CPU2017 benchmarks are still written in C, C++ and Fortran languages.

Several new benchmarks and application domains have been added in the CPU2017 suite. In the FP category, nine new benchmarks have been added: *parest* implements a finite element solver for biomedical imaging; *blender* performs 3D rendering; *cam*4, *pop*2 and *roms* represent the climatology domain; *imagick* is an image manipulation application; *nab* is a floating-point intensive molecular modeling application representing the life sciences domain; *fotonik*3*d* and *cactuBSSN* represents the physics domain. In the INT category, the most notable enhancement has been made in the artificial intelligence domain with three new benchmark additions (*deepsjeng*, *leela* and *exchange*2). Two other compression-related benchmarks, *x*264 (video compression) and *xz* (general data compression) have also been added. We will analyze the application domain coverage of CPU2017 suite in detail in Section IV.

B. Performance Characterization

Table I shows the dynamic instruction count, instruction mix, and CPI of each CPU2017 benchmark. The dynamic instruction count of the benchmarks is in the order of trillions of instructions. In general, the speed benchmarks have significantly higher dynamic instruction count than the rate benchmarks. The ratio of dynamic instruction count in speed to rate categories is ~8x (avg) for the floating-point benchmarks and ~2x (avg) for the integer benchmarks. Compared to the CPU2006 FP benchmarks, the CPU2017 FP benchmarks have ~10x higher dynamic instruction count. This steep increase in instruction counts will further exacerbate the problem of benchmark simulation time on most state-of-the-art simulators [2], [3], [5].

TABLE I: Dynamic Instr. Count, Instr. Mix and CPI of the 43 SPEC CPU2017 benchmarks (Intel Skylake).

Benchmark	Icount (Billion)	Loads (%)	Stores (%)	Branches (%)	CPI
SPECspeed Integer — 10 benchmarks					
600.perlbench_s	2696	27.20	16.73	18.16	0.42
602.gcc_s	7226	40.32	15.67	15.60	0.58
605.mcf_s	1775	18.55	4.70	12.53	1.22
620.omnetpp_s	1102	22.76	12.65	14.55	1.21
623.xalancbmk_s	1320	34.08	7.90	33.18	0.86
625.x264_s	12546	37.21	10.27	4.59	0.36
631.deepsjeng_s	2250	19.75	9.37	11.75	0.55
641.leela_s	2245	14.25	5.32	8.94	0.80
648.exchange2_s	6643	29.61	20.22	8.67	0.41
657.xz_s	8264	13.34	4.73	8.21	1
SPECrate Integer — 10 benchmarks					
500.perlbench_r	2696	27.20	16.73	18.16	0.42
502.gcc_r	3023	34.51	16.64	14.96	0.59
505.mcf_r	999	17.42	6.08	11.54	1.16
520.omnetpp_r	1102	22.10	12.27	14.12	1.39
523.xalancbmk_r	1315	34.26	8.07	33.26	0.86
525.x264_r	4488	23.03	6.47	4.37	0.31
531.deepsjeng_r	1929	19.61	9.10	11.61	0.57
541.leela_r	2246	14.28	5.33	8.95	0.81
548.exchange2_r	6644	29.62	20.24	8.69	0.41
557.xz_r	1969	17.33	3.87	12.24	1.22
SPECspeed Floating-point — 10 benchmarks					
603.bwaves_s	66395	31.00	4.42	13.00	0.34
607.cactuBSSN_s	10976	43.87	9.50	1.80	0.68
619.lbm_s	4416	29.62	17.68	1.40	0.87
621.wrf_s	18524	23.20	5.80	9.48	0.77
627.cam4_s	15594	20	14	10.92	0.68
628.pop2_s	18611	21.71	8.41	15.13	0.48
638.imagick_s	66788	18.16	0.46	9.30	1.17
644.nab_s	13489	23.49	7.51	9.55	0.68
649.fotonik3d_s	4280	33.99	13.89	3.84	0.78
654.roms_s	22968	32.02	8.02	7.53	0.52
SPECrate Floating-point — 13 benchmarks					
503.bwaves_r	5488	34.92	4.77	9.51	0.42
507.cactuBSSN_r	1322	43.62	9.53	1.97	0.69
508.namd_r	2237	30.12	10.25	1.75	0.41
510.parest_r	3461	29.51	2.50	11.49	0.48
511.povray_r	3310	30.30	13.13	14.20	0.42
519.lbm_r	1468	28.35	15.09	1.05	0.53
521.wrf_r	3197	22.94	5.93	9.48	0.81
526.blender_r	5682	36.10	12.07	7.89	0.53
527.cam4_r	2732	19.99	8.37	11.06	0.56
538.imagick_r	4333	22.55	7.97	10.94	0.90
544.nab_r	2024	23.70	7.46	9.65	0.69
549.fotonik3d_r	1288	39.12	v12.07	2.52	0.96
554.roms_r	2609	34.57	7.57	6.73	0.48

In terms of instruction mix, we can make several interesting observations. For the integer benchmarks (rate and speed), the fraction of branch instructions is roughly ≤15%, with several benchmarks (e.g., 625.*x*264_*s*, 641.*leela*_*s*, 525.*x*264_*r*) having ≤8% branch instructions. This behavior is in contrast to the CPU2006 integer programs, which have an average of 20% branches in their dynamic instruction stream [9]. The *xalancbmk* benchmark, which is one of the four C++ programs in the INT category, has the highest fraction of branch instructions (33%). The other C++ programs (*omnetpp*, *leela* and *deepsjeng*) have ≤15% branches. For the FP categories, most benchmarks have much lower fraction of control instructions (≤9% on average) than the integer benchmarks, with several benchmarks having as low as 1% branches. The large dynamic basic block size of the FP programs can be an opportunity for the underlying micro-architectures to exploit higher degree of parallelism. In terms of memory operations, the CPU2017 benchmarks are memory-intensive, with several benchmarks

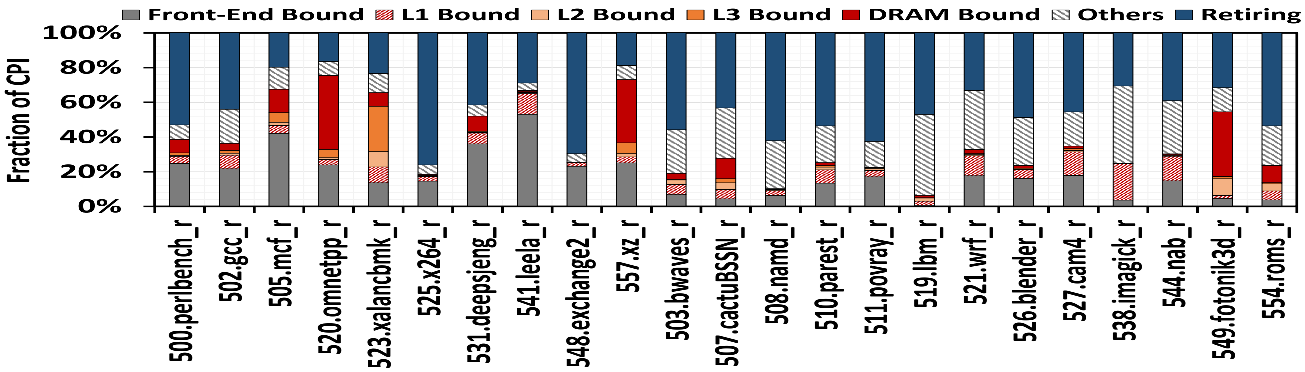


Fig. 1: Cycles per instruction (CPI) stack of CPU2017 rate benchmarks.

(e.g., 602.*gcc_r*, 507.*cactuBSSN_r*) having ∼50% fraction of memory (load and store) instructions. Later in this section, we will show that a significant fraction of the execution time of these benchmarks is spent in servicing cache and memory requests, which limits their performance.

Table II shows the range of a few performance metrics of the CPU2017 benchmarks measured using hardware performance counters on the Skylake micro-architecture. The magnitude difference between the min and max values shows that there is a lot of diversity in the performance characteristics across different benchmarks. The older SPEC CPU benchmarks have often been criticized because they do not have sufficient instruction cache miss activity as some of the emerging cloud and big-data applications [10], [11]. Interestingly, many CPU2017 benchmarks do not suffer from high instruction cache miss rates, even though the workload sizes have increased significantly.

1) Performance Bottleneck Analysis

In this section, we conduct micro-architectural bottleneck analysis of the CPU2017 applications using cycle per instruction (CPI) stack statistics. A CPI stack breaks down the execution time of an application into different micro-architectural activities (e.g., accessing cache), showing the relative contribution of each activity. Optimizing the largest component(s) in the CPI stack leads to the largest performance improvement. Therefore, CPI stacks can be used to identify sources of micro-architecture inefficiencies. We follow the top-down performance analysis methodology to collect the CPI stack information [12]. Table I also shows the actual CPI numbers for the benchmarks.

Figure 1 shows the CPI stack breakdown of the CPU2017 rate applications (see Table I for the CPI values). The front-end bound category includes the instruction fetch and branch misprediction related stall cycles. The 'other' category includes resource stalls, instruction dependencies, structural dependencies, etc. Several interesting observations can be made from the CPI stack breakdown. In most cases, more than 50% of the total execution time is spent on various types of on-chip micro-architectural activities, with mcf_r and $omnetpp\_r$ having the highest CPI among all the benchmarks. Several benchmarks (e.g., $leela\_r$, mcf_r, xz_r) spend a significant fraction of their execution time on front-end stalls as they suffer from higher branch misprediction rates. The mcf_r benchmark further suffers from high instruction cache miss rate, aggravating its front-end performance bottleneck. In general, the integer benchmarks suffer from higher branch misprediction rates than the floating-point benchmarks, leading to higher branch misspeculation related stalls. In terms of back-end (cache and memory) performance, $omnetpp\_r$, $xalancbmk\_r$, mcf_r and $fotonik3d\_r$ benchmarks spend a significant fraction of their execution time servicing cache and memory requests. For $blender\_r$ and $imagick\_r$ benchmarks, high inter-instruction dependencies are the major cause of pipeline stalls. Most speed benchmarks (not shown here due to space limit) also have similar performance correlations.

TABLE II: Range of important performance characteristics of SPEC CPU2017 benchmarks.

	Rate INT	Speed INT	Rate FP	Speed FP
Metric	**Range (Min - Max)**			
L1D$ MPKI*	∼0 - 56	∼0 - 54.7	2 - 95.4	5.5 - 98.4
L1I$ MPKI	∼0 - 5.1	∼0 - 5.2	∼0 - 11.3	0.1 - 11.6
L2D$ MPKI	∼0 - 20.5	∼0 - 20.7	∼0 - 7	0.2 - 8.6
L2I$ MPKI	∼0 - 0.9	∼0 - 0.9	∼0 - 1.2	∼0 - 1.2
L3$ MPKI	∼0 - 4.5	∼0 - 4.6	∼0 - 4.3	∼0 - 5
Branch misp. per kilo inst.	0.9 - 8.3	0.5 - 8.4	0 - 2.5	0.01 - 2.5

*MPKI stands for Misses Per Kilo Instructions.

III. Methodology

To perform a comprehensive analysis of the CPU2017 benchmark suite, we collect and use a large range of program characteristics, related to instruction and data locality, branch predictability, and instruction mix. The profiled characteristics are micro-architecture dependent, which can cause the results to be biased by features of a particular machine. Thus, in order to minimize this bias, measurements are collected on seven commercial machines with three different ISAs (machine details are summarized in Table IV). The differences in micro-architecture, ISA, and compiler help to eliminate any micro-architectural dependency and allows to capture only the true differences among the benchmarks. The performance metrics used in any subsequent analysis are listed in Table III. Some of the hardware performance counter data used in this study were measured by the authors, while other data were collected by various SPEC companies on their machines with advanced compilers.

TABLE III: Program characteristics for similarity analysis.

Characteristics	Metrics
Cache	L1I/D MPKI, L2I/D MPKI, L3 MPKI
TLB	L1I/D TLB MPMI†, Last level TLB MPMI‡, Page Walks per MI
Branch predictor	Branch MPKI, Branch taken MPKI
Inst Mix	Percentage of Kernel, User, INT, FP Load, Store, Branch, SIMD
Power	Core, LLC and Memory Power

TABLE IV: Hardware configurations of 7 machines (Intel, AMD, and Oracle) used in the experiments

Processor	ISA	L1(KB)	L2(KB)	LLC(MB)
Intel Core i7-6700	x86	2x32	256	8
Intel Xeon E5-2650 v4	x86	2x32	256	30
Intel Xeon E5-2430 v2	x86	2x32	256	15
Intel Xeon E5405	x86	2x32	2x6MB	N/A
SPARC-IV+ v490	SPARC	2x64	2MB	32
SPARC T4	SPARC	2x16	128	4
AMD Opteron 2435	x86	2x64	512	6

As we perform measurements on seven different machines, we treat each performance counter-machine pair as a metric. Overall, we measure 20 performance-related metrics for each benchmark on every machine, leading to a total of 140 metrics. However, it is difficult to manually look at all the data and conduct meaningful analysis. Hence, we leverage the Principal Components Analysis (PCA) technique [13], [14] to first remove any correlations among the variables (e.g., when two variables measure the same benchmark property). PCA converts i variables X_1, X_2,...,X_i into j linearly uncorrelated variables Y_1, Y_2,...,Y_j, called Principal Components (PCs). Each PC is a linear combination of various features or variables with a certain weight, known as loading factor (see Equation 1).

$$Y_1 = \sum_{k=1}^{i} a_{1k}X_k \;\; ;Y_2 = \sum_{k=2}^{i} a_{2k}X_k \; ... \qquad (1)$$

PCA transformation has many interesting properties, the first PC covers most of the variance while other PCs cover decreasing variances. Dimensionality of the data-set can be reduced by removing components with lower variance values. We use the Kaiser Criterion to choose PCs, where only top few PCs are retained, with eigenvalues ≥ 1. After performing PCA, we use another statistical technique called hierarchical clustering to analyze the similarity among benchmarks. The similarity between benchmarks is measured using the Euclidean distance of program characteristics. The results produced by this clustering technique can be presented as a tree or dendrogram. Linkage distances shown in a dendrogram represent similarity between programs (e.g. Figure 2).

IV. Redundancy in CPU2017 Benchmark Suite

A. *Subsetting the CPU2017 Benchmarks*

We discussed in Section II-B that the dynamic instruction counts of the CPU2017 benchmarks have increased up to 10x versus its predecessor. Such a significant increase in the runtime of benchmarks will make it virtually impossible

†MPMI stands for Misses Per Million Instructions.

‡Depends on the profiled machine, this can be unified or individual.

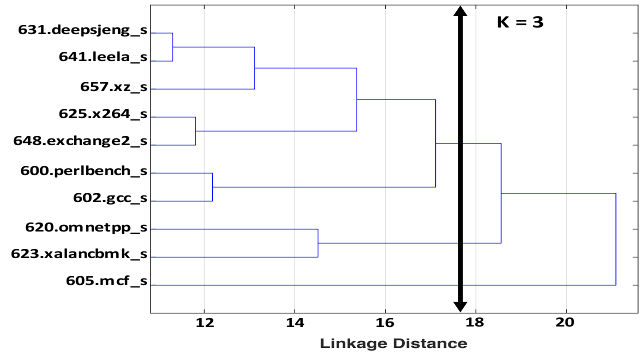


Fig. 2: Dendrogram showing similarity between SPECspeed INT benchmarks.

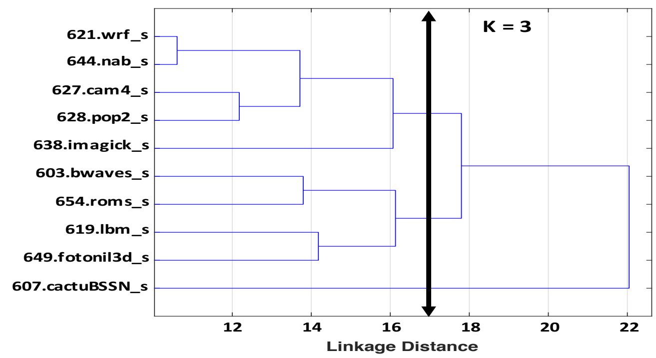


Fig. 3: Dendrogram showing similarity between SPECspeed FP benchmarks.

to perform architectural analysis for the entire CPU2017 benchmark suite on detailed performance simulators in a reasonable time. If similar information can be obtained using a subset of the CPU2017 benchmark suite, it can help architects and researchers to make faster design trade-off analysis. In this section, we study the (dis)similarities between different benchmarks belonging to the SPECrate INT, SPECspeed INT, SPECrate FP and SPECspeed INT categories individually. Linkage distance is used to identify representative subsets of the CPU2017 sub-suites.

Figure 2 shows the dendrogram plot for the SPECspeed INT benchmarks (SPECrate INT, not shown due to space considerations, has a very similar dendrogram). Seven PCs that cover more than 91% of the variance are chosen based on the Kaiser criterion. The x-axis shows the linkage distance between different benchmarks (y-axis). Smaller linkage distance between any two benchmarks indicates that the benchmarks are close, and vice versa. The ordering of benchmarks on the y-axis has no special significance. We can observe that the 605.*mcf_s* and 505.*mcf_r* benchmarks have the most distinct performance features among all the INT benchmarks. The dendrogram plot shown in Figure 2 can be used to identify a representative subset of the SPECspeed INT suite. For instance, if a researcher wants to reduce his simulation time budget to only three benchmarks for the SPECspeed INT category, a vertical line drawn at a linkage distance of 17.5 in Figure 2 can yield a subset of three benchmarks (605.*mcf_s*, 623.*xalancbmk_s* and 641.*leela_s*). For clusters having more than two benchmarks, the benchmark with the shortest linkage

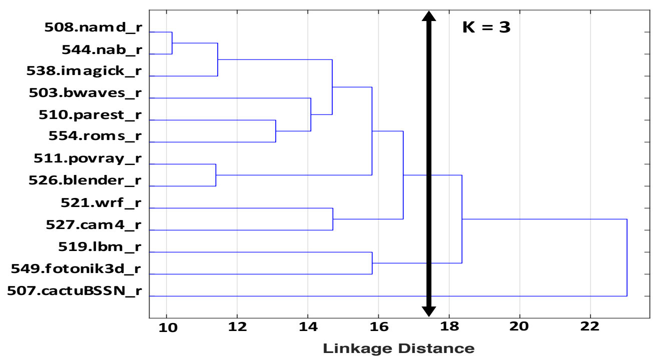


Fig. 4: Dendrogram showing similarity between SPECrate FP benchmarks.

TABLE V: Representative subsets of the CPU2017 sub-suites.

SPECspeed INT Subset of 3 Benchmarks	605.mcf_s, 641.leela_s, 623.xalancbmk_s
SPECrate INT Subset of 3 Benchmarks	505.mcf_r, 523.xalancbmk_r, 531.deepsjeng_r,
SPECspeed FP Subset of 3 Benchmarks	607.cactuBSSN_s, 621.wrf_s 654.roms_s
SPECrate FP Subset of 3 Benchmarks	507.cactuBSSN_r, 549.fotonik3d_r 544.nab_r

distance is chosen as the representative benchmark. Such analysis can be done at varying linkage distances to select the appropriate number of benchmarks when simulation time is constrained. To subset the SPECrate INT benchmark category, we use a similar approach. Overall, only simulating the suggested subsets (summarized in Table V) can reduce the total simulation time by 5.6× and 4.5× for SPECspeed INT and SPECrate INT suites, respectively.

The dendrograms for the SPECspeed FP and SPECrate FP benchmarks are shown in Figures 3 and 4 respectively. The 607.*cactuBSSN_s* and 507.*cactuBSSN_r* benchmarks have the most distinctive performance characteristics among all the FP benchmarks. Further analysis into the performance characteristics of the two benchmarks reveals that they have unique behavior in terms of their memory and TLB performance. The two vertical lines drawn in Figures 3 and 4 show the points at which 3-benchmark subsets are formed for both the FP suites. Using the benchmark subsets summarized in Table V reduces the simulation time by 4.5× and 6.3× for the SPECspeed and SPECrate FP sub-suites, respectively. It is interesting to observe that the chosen subsets contain several newly added benchmarks such as, 544.*nab_r*, 507.*cactuBSSN_r*, 654.*roms_s*, and 607.*cactuBSSN_s*. It should be noted that although this subsetting approach can identify reduced subsets in terms of hardware performance characteristics, it does not guarantee a coverage of all the different application domains of the benchmark suite.

B. Evaluating Representativeness of Subsets

Next, we evaluate the usefulness of the subsets (identified in the last section) to estimate the performance of the CPU2017 benchmark suites on commercial systems, whose results are already published on SPEC's web page.

For this analysis, we record the performance of different benchmarks on different commercial computer systems' (speedup over a *ref* machine) from SPEC's database. Then, we compute the overall performance score (geometric mean) of the benchmark subsets and compare it against the performance score (geometric mean) of all the benchmarks in that sub-suite. For example, for the SPECspeed INT category, we compute the average performance score using the 3-benchmark subset and compare it against the average performance score using all 10 benchmarks belonging to the SPECspeed INT category. Since CPU2017 suite is released very recently, very few companies have submitted the results for all speed and rate categories. Therefore, the different commercial systems used for validating the four benchmark categories are not exactly identical. But, we include all the submitted results obtained from SPEC's web page.

TABLE VI: Accuracy comparison among proposed subsets and random subsets.

	Identified subsets	Rand set1	Rand set2
SPECspeed INT	¡1%	28.2%	23.4%
SPECrate INT	7%	22.4%	21.7%
SPECspeed FP	3%	49.7%	25.6%
SPECrate FP	4.5%	39.1%	27.1%

Figure 5 shows the validation results for the SPECspeed INT and SPECrate INT sub-suites. The average error for the SPECspeed INT category is $\leq$1% across 4 systems. For the SPECrate INT category, using a subset of 3 benchmarks achieves an average error of 7% (maximum 12.9%) in terms of speedup as compared to using all the benchmarks. Figure 6 shows similar validation results for the FP categories. Using 3 out of the 10 SPECspeed FP benchmarks produces an average error of 3%, and 3 out of the 13 SPECrate FP benchmarks leads to a 4.5% speedup estimation error. To further evaluate the effectiveness of the proposed subsets, we compare their speedup estimation accuracy with respect to two randomly selected subsets. Results are shown in Table VI, where random sets 1 and 2 result in an average error of 34.85% and 24.45% respectively.

The above analysis shows that the identified subsets can accurately predict the performance speedup of the entire benchmark suite. Including more benchmarks in the subset can reduce the prediction error, but will also increase the simulation time significantly. However, only a third of the benchmark suite can be used to predict the performance of the entire benchmark suite reasonably well.

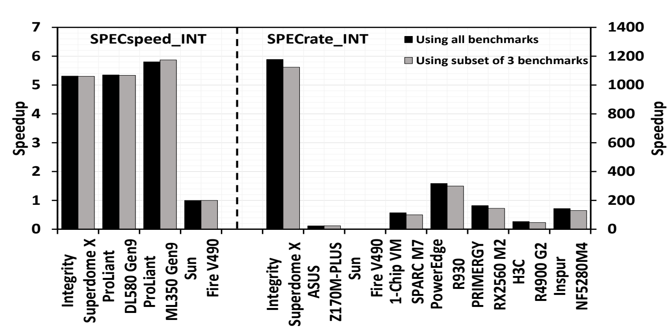


Fig. 5: Validation of SPECspeed and SPECrate INT subsets using performance scores of commercial systems from SPEC's web page.

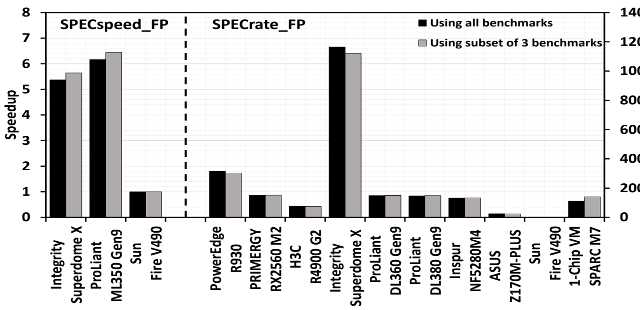


Fig. 6: Validation of SPECspeed FP and SPECrate FP subsets using performance scores of commercial systems from SPEC's web page.

C. Selecting Representative Input Sets

Similar to CPU2006 benchmarks, many CPU2017 benchmarks have multiple input sets. For example, 502.*gcc_r* and 525.*x264_r* benchmarks have five and three different input sets, respectively. For a reportable run of such benchmarks, SPEC requires aggregating results across all the different input sets. However, simulating all possible input sets for a benchmark for design trade-off studies can take a prohibitive amount of time. In this section, we want to systematically evaluate differences among the performance characteristics of different input sets belonging to the same benchmark. Such analysis can help researchers to select representative input sets of each benchmark for their evaluation studies, rather than choosing an input set in an ad hoc manner.

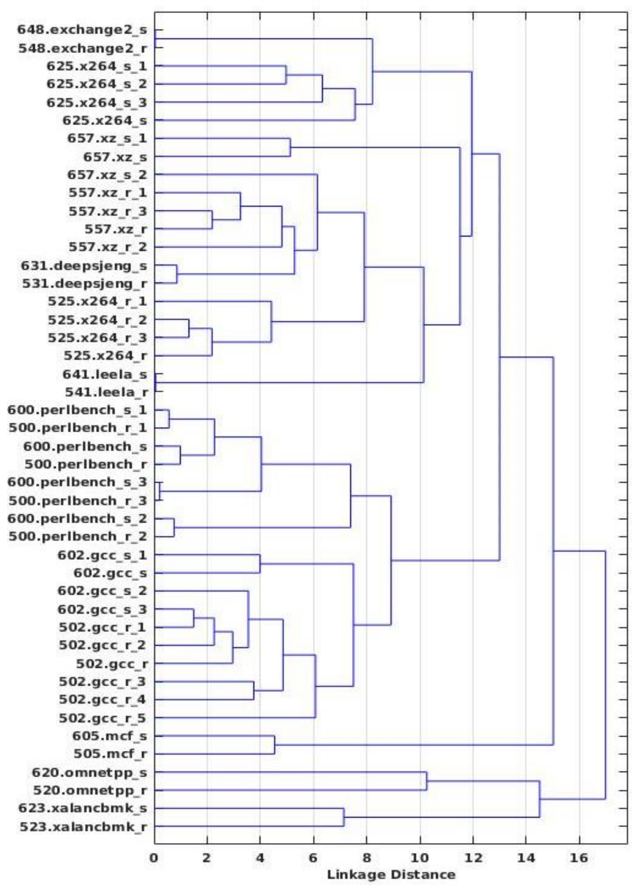


Fig. 7: Dendrogram showing similarity between program input sets of each SPEC 2017 INT benchmark.

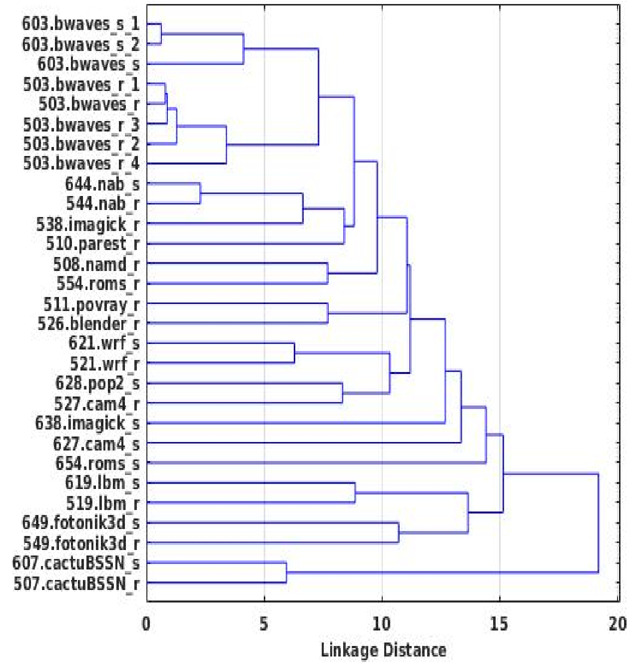


Fig. 8: Dendrogram showing similarity between input sets of each SPEC 2017 FP benchmark.

TABLE VII: List of representative input sets of CPU2017 benchmarks.

SPECrate INT benchmarks	**SPECspeed INT benchmarks**
500.perlbench_r - input set 1	600.perlbench_s - input set 1
502.gcc_r - input set 2	602.gcc_s - input set 1
525.x264_r - input set 3	625.x264_s - input set 3
557.xz_r - input set 1	657.xz_s - input set 1
SPECrate FP benchmarks	**SPECspeed FP benchmarks**
503.bwaves_r - input set 1	603.bwaves_s - input set 1

Figure 7 shows the dendrogram plot for different INT benchmarks and their input sets. Benchmarks having a single input set are represented by their original names, while benchmarks with multiple input sets are numbered based on the output of the *specinvoke* tool. For this analysis, ten PCs are chosen covering 94% of variance using the Kaiser criterion. We can see that for all the benchmarks, different input sets have very similar characteristics. For example, the five different input sets of 502.*gcc_r* are clustered together in the dendrogram plot. This is in contrast to more pronounced variations between the various inputs for gcc benchmark in the CPU2006 [14].

We perform similar analysis on the different input sets of the floating-point benchmarks. The 603.*bwaves_s* and 502.*bwaves_r* benchmarks are the only two floating-point benchmarks with multiple input sets. Figure 8 shows the similarity between different input sets of the FP programs for both rate and speed categories. Twelve PCs covering 94% of the variance are used for this analysis. To identify the most representative input set of each benchmark, we choose the input set that is closest to the aggregated benchmark run. The most representative input set of each benchmark is summarized in Table VII. This analysis can help researchers in selecting the most representative input set for each benchmark.

D. Are Rate and Speed Benchmarks Different?

So far, our analysis has considered the rate and speed benchmarks separately. With the exception of a few benchmarks (508.*namd_r*, 510.*parest_r*, 511.*povray_r*, 526.*blender_r* and 628.*pop2_s*), most benchmarks are included in both rate and speed categories. Based on the information provided on SPEC's web page, rate and speed benchmarks differ in terms of the workload sizes, compilation flags and run rules. For example, SPEC's web page suggests that the 603.*bwaves_s* benchmark has a memory usage of 11.2 GB versus the 0.8GB usage of the 503.*bwaves_r* benchmark. Similarly, the 605.*mcf_s* and 649.*fotonik3d_s* benchmarks also have significantly higher memory usage than their rate versions. Furthermore, the speed benchmarks have much higher dynamic instruction counts and runtime than the rate benchmarks. However, do these differences translate into low-level micro-architectural performance variations?

In this section, we use PCA and hierarchical clustering analysis to compare performance characteristics of the rate and speed benchmarks. We will use the dendrogram plots in Figures 7 and 8 for performing this analysis. From the dendrogram plot for the INT benchmarks in Figure 7, we can observe that most benchmarks belonging to the rate and speed categories have very similar performance characteristics. Only three benchmarks (620.*omnetpp_s*, 623.*xalancbmk_s* and 625.*x264_s*) have higher linkage distances to their respective rate versions. On the other hand, for the FP benchmarks, many benchmarks have significant differences between the rate and speed versions. The most notable example is the 638.*imagick_s* benchmark, which has $\geq$30% higher misses in all cache levels than the 538.*imagick_r* benchmark, resulting in the largest linkage distance between the two. Also, the high memory usage of 603.*bwaves_s* makes its cache performance significantly different from its rate version. FP benchmarks such as 644.*nab_s*, 621.*wrf_s*, 607.*cactuBSSN_s* etc. have similar performance as their rate equivalents. It should be noted that we consider only single-core performance of the rate and speed benchmarks (we suppress all OPENMP directives in the speed benchmarks).

E. Benchmark Classification based on Branch and Memory Behavior

So far, we have looked at the aggregate performance characteristics of CPU2017 benchmarks based on all the metrics shown in Table III. However, many times, researchers are interested in studying only particular aspects of program performance, e.g., the control-flow predictor performance, cache performance etc. In this section, we compare different CPU2017 benchmarks in terms of the branch characteristics, data cache and instruction cache performance. This similarity analysis can help to identify important programs of interest when performing branch predictor or cache related studies. We analyze all the CPU2017 benchmarks from the speed and rate categories without classifying them into integer and floating-point groups.

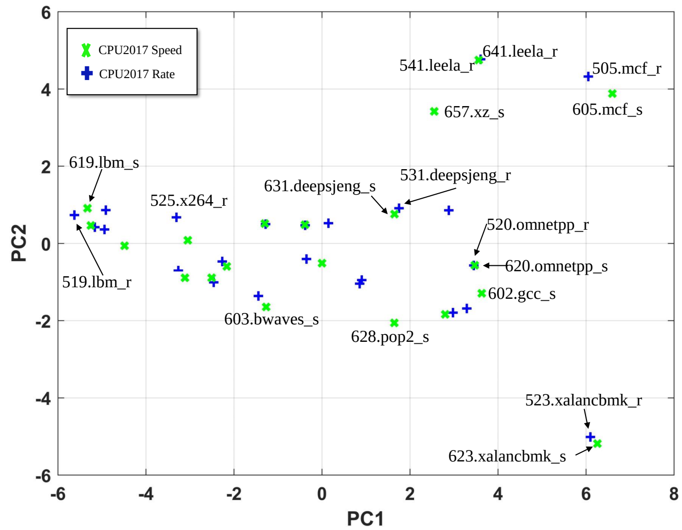
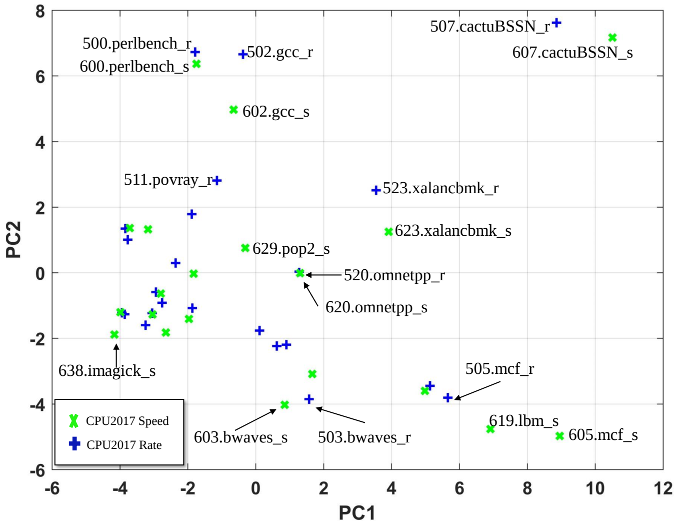


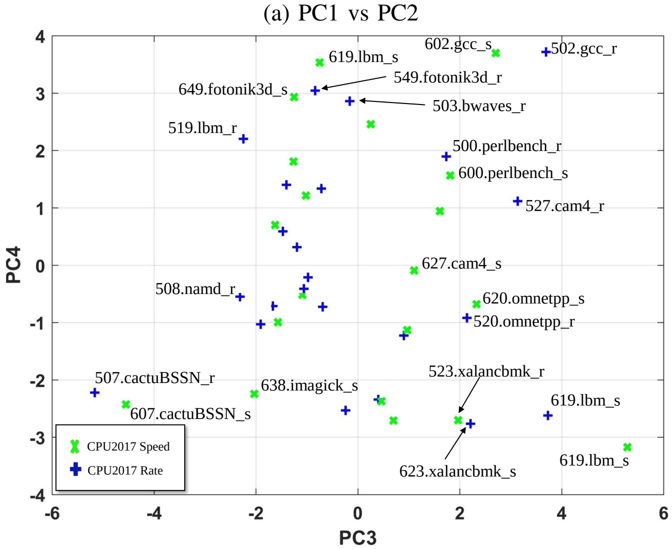
Fig. 9: Comparing CPU2017 benchmarks in the PC workload space based on branch performance metrics.

Figure 9 shows the scatter-plot based on the first two PCs of the branch characteristics, covering over 94% of the variance. PC2 is dominated by branch mispredictions per kilo instructions and PC1 is dominated by the fraction of branch instructions and fraction of taken branches. The 541.*leela_r*, 641.*leela_s*, 505.*mcf_r* and 605.*mcf_s* benchmarks have a higher fraction of difficult-to-predict branches, and thus suffer from the highest branch misprediction rates among the different CPU2017 programs. In the CPU2017 suite, 505.*mcf_r*, 605.*mcf_s*, 502.*gcc_r* and 602.*gcc_r* benchmarks have the highest fraction of taken branches. It is also interesting to observe that a majority of C++ benchmarks (e.g., 623.*xalancbmk_s*, 523.*xalancbmk_r*, 620.*omnetpp_s*, 520.*omnetpp_r*) have a higher fraction of taken branches. Also, most floating-point benchmarks are clustered together, while the integer programs show greater diversity in terms of control-flow behavior.

The PC1 values are dominated by high L1 and L2 data cache miss rates. Thus, benchmarks having higher PC1 values have poor data locality. The benchmarks that experience the highest data cache miss rates among the CPU2017 suite are 605.*mcf_s*, 505.*mcf_r*, 607.*cactuBSSN_s*, 507.*cactuBSSN_r*, 649.*fotonik3d_s* and 549.*fotonik3d_r*. Out of these benchmarks, the *cactuBSSN* and *fotonik3d* benchmarks have been recently introduced in the CPU2017 suite. The PC2 values are dominated by high data cache accesses. The 500.*perlbench_r*, 600.*perlbench_s* and 607.*cactuBSSN_s*, 507.*cactuBSSN_r* benchmarks from CPU2017 suite have a high number of data cache accesses. In the PC3-PC4 plot (see Figure 10), the PC4 values are dominated by instruction cache accesses and misses. SPEC CPU benchmarks have often been criticized as they do not have as much instruction cache activity and misses as some of the emerging big-data and cloud workloads [10], [15], [11]. In general, CPU2017 benchmarks also do not have very high instruction cache miss rates (instruction cache MPKI ranges between 0-11). Nonetheless, the



(a) PC1 vs PC2



(b) PC3 vs PC4

Fig. 10: CPU2017 (rate and speed) benchmarks in the PC workload space using data and instruction cache characteristics

500.*perlbench_r*, 600.*perlbench_s*, 502.*gcc_r* and 602.*gcc_r* benchmarks have the highest instruction cache access and miss activity.

Although this analysis helps in identifying benchmarks that exercise a certain performance metric, care should be exercised when selecting benchmarks for any particular study so that the chosen benchmarks cover the entire workload space. Selecting outlier benchmarks will only emphasize the best-case or worst-case performance behavior, which may lead to misleading conclusions.

F. Difference Between Benchmarks from Same Application Area

In this section, we classify the CPU2017 benchmarks based on their application domain (see Table VIII) and seek to find (dis)similarities between different benchmarks belonging to the same category. The benchmarks that are marked in bold in the table have distinct performance behaviors and should be used to cover the performance spectrum for their respective application domain. For those benchmarks which have similar performance behavior in the rate and speed mode, we mark only the rate versions in the table (as they are short-running). For example, in the compiler/interpreter application domain, 502.*gcc_r* and 500.*perlbench_r* have distinct performance characteristics, but are similar to their respective speed equivalents. Thus, running the 502.*gcc_r* and 500.*perlbench_r* benchmarks can represent the performance spectrum of that application domain. As we discussed before, many CPU2017 benchmarks exhibit different behaviors in the rate and speed versions. For example, for the fluid dynamics and climatology domains, both speed and rate versions of the *bwaves*, *roms*, *lbm* benchmarks should be used to achieve comprehensive domain coverage.

TABLE VIII: Classification of benchmarks based on application domains.

INT Benchmarks	
App domain	SPEC 2017
Compiler	**502.gcc_r**, 602.gcc_s **500.perlbench_r**, 600.perlbench_s
Compression	**525.x264_r**,**557.xz_r**, **625.x264_s**, 657.xz_s
AI	**531.deepsjeng_r**, 631.deepsjeng_s, **541.leela_r**, 641.leela_s, **548.exchange2_r**, 648.exchange2_s
Combinatorial optimization	**505.mcf_r**, 605.mcf_s
DE Simulation	**520.omnetpp_r**, **620.omnetpp_s**
Doc Processing	**523.xalancbmk_r**, **623.xalancbmk_s**
FP Benchmarks	
App domain	SPEC 2017
Physics	**507.cactuBSSN_r**, **549.fotonik3d_r**, 607.cactuBSSN_s, **649.fotonik3d_s**
Fluid dynamics	**519.lbm_r**, **503.bwaves_r**. **619.lbm_s**, **603.bwaves_s**
Molecular dynamics	**508.namd_r**,**544.nab_r**, 644.nab_s
Visualization	**511.povray_r**,526.blender_r, **538.imagick_r**,**638.imagick_s**
Biomedical	**510.parest_r**
Climatology	**521.wrf_r**, 527.cam4_r, 628.pop2_s, **554.roms_r** 621.wrf_s, 627.cam4_s, **654.roms_s**

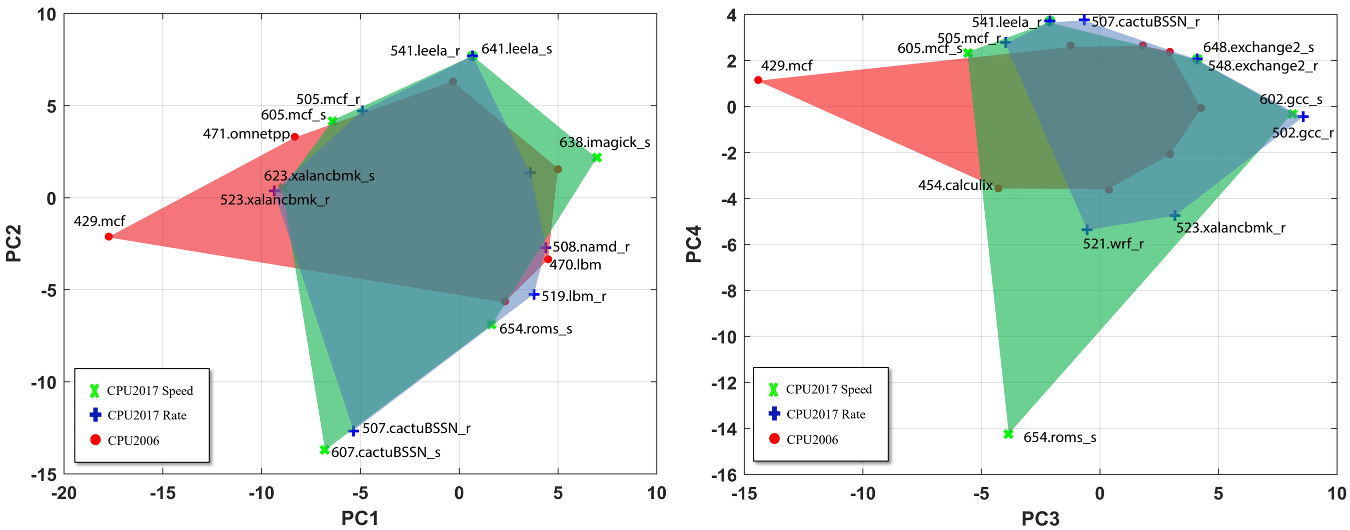
V. Balance in the SPEC CPU2017 Benchmark Suites

This section compares the CPU2017 benchmarks with the CPU2006 benchmarks and with popular workloads from other domains, such as graph analytics, EDA and data-serving applications. Finally, we also analyze the sensitivity of the CPU2017 benchmarks to different micro-architectural performance characteristics.

A. Comparing Performance Spectrum of CPU2017 & CPU2006 Suites

The CPU2017 suite has revamped many of the benchmarks in the SPEC CPU2006 suite or replaced them with larger/more complex workloads in order to allow stress-testing of powerful modern-day processors and their successors. However, it is not known whether these workloads have different performance demands or whether they stress machines differently compared to CPU2006 benchmarks. Have the new CPU2017 benchmarks managed to expand the workload design-space beyond the CPU2006 benchmarks? Did removing or replacing any CPU2006 benchmarks cause a loss in coverage of the performance spectrum?

Figure 11 shows the scatterplot comparing the CPU2006 and CPU2017 benchmarks based on the top four PCs (covering



(a) PC1 vs PC2 (b) PC3 vs PC4

Fig. 11: CPU2017 (rate and speed) and CPU2006 benchmarks in the PC workload space.

80% of the variance), using the performance metrics shown in Table III. In terms of the PC1-PC2 spectrum, CPU2017 only slightly expands the coverage area; however, more than 25% of the CPU2017 benchmarks fall outside the space covered by the CPU2006 programs. In terms of PC3-PC4 spectrum, the 2017 benchmarks cover twice as much area as the 2006 benchmarks. From these results, we can conclude that the CPU2017 benchmarks are spread farther in the workload space as compared to the CPU2006 benchmarks in terms of performance characteristics, thereby expanding the envelope of the workload design space. The newly added benchmarks, such as 507.*cactuBSSN_r*, 654.*roms_s*, 638.*imagick_s*, 641.*leela_s*, etc., contribute significantly to this increased diversity.

It is also interesting to note that with the exception of a few CPU2017 programs (e.g., 520.*omnetpp_r* and 503.*bwaves_r*), which have been retained from the CPU2006 suite, most benchmarks have quite different overall performance characteristics as compared to their predecessors. This implies that the benchmarks have been changed to not only have a higher instruction count and bigger data footprint, but they have also undergone changes in control-flow, instruction and data locality behavior. As an exception, the 429.*mcf* benchmark from the CPU2006 suite, a highly popular benchmark to evaluate cache and memory behavior, exerts the data caches (all cache-levels) more than the mcf benchmarks from the CPU2017 suite (the 505.*mcf_r* and 605.*mcf_s* programs).

B. Comparison of Application Domains

Comparing the application domains of the CPU2017 (see Table VIII) and CPU2006 benchmarks, we can see that many new application domains have been introduced or greatly expanded in the CPU2017 suite. For example, the artificial intelligence domain has been expanded in the CPU2017 suite to include three new benchmarks. Similarly, 510.*parest_r* benchmark is added to represent the biomedical category. On the other hand, many application domains from the CPU2006 suite have been omitted as well: speech recognition (483.*sphinx*3), linear programming (450.*soplex*), quantum chemistry (e.g., 416.*gamess*, 465.*tonto*), etc.

Loss of an application domain does not necessarily imply a loss in the performance spectrum. Any two benchmarks from different application domains may have similar behavior if they stress similar micro-architectural structures. Similarly, two benchmarks from the same application domain can have very different performance characteristics. Using PCA and hierarchical clustering (see cluster plots in Figure 11), we analyzed every benchmark of the CPU2006 suite, which have been removed from the CPU2017 suite and identify those CPU2006 benchmarks whose performance characteristics are not covered by the CPU2017 benchmarks. Interestingly, we find that only three benchmarks (429.*mcf*, 445.*gobmk* and 473.*astar*) are not covered. The workload space of the remaining removed benchmarks is covered by the CPU2017 benchmarks.

C. Comparing Power Consumption

Next, we compare the power characteristics of the CPU2017 and CPU2006 benchmarks. Power is measured by using RAPL counters available on three different Intel-based micro-architectures (Skylake, Ivybridge, and Broadwell). Figure 12 shows the scatter-plot based on first two PCs (covering more than 84% of the variance). PC1 is dominated by the power spent in DRAM memory and PC2 is dominated by the power spent in the processor cores. Overall, we observe that the CPU2017 benchmarks have much higher coverage space as compared to the CPU2006 benchmarks. It should be noted that many newly added benchmarks (e.g., 648.*exchange*2*_s*, 548.*exchange*2*_r*, 641.*leela_s*, 554.*roms_r*, 557.*xz_r*, and 538.*imagick_r*) contribute to this broader coverage. In general, CPU2006 benchmarks exhibit greater diversity in the PC1 spectrum as compared to the PC2 spectrum. On the other hand, over 20 benchmarks from the CPU2017 suite have significant variations in terms of core power consumption. To

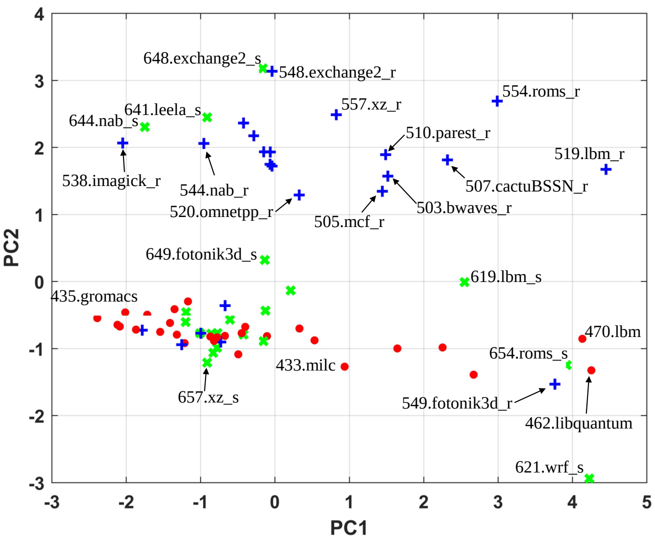


Fig. 12: CPU2017 (rate and speed) benchmarks in the PC workload space using power characteristics.

the best of our knowledge, CPU2017 benchmarks are more computationally-intensive. This results in the higher diversity in the core power consumption. Therefore, we can conclude that CPU2017 benchmarks can be more useful than CPU2006 benchmarks for power/energy efficiency related studies.

D. Case Study on EDA Applications

Applications from the Electronic Design Automation (EDA) domain were included in early SPEC CPU benchmark suites (e.g., CPU2000). However, EDA benchmarks were removed from the CPU2006 suite. Nonetheless, it has been shown by prior research that CPU2006 suite contains several benchmarks that show similar behavior as the EDA benchmarks [14], which makes the CPU2006 suite balanced even without the EDA applications. No EDA application is included in the CPU2017 suite either. Do the CPU2017 benchmarks cover the performance spectrum of the EDA applications? To answer this, we select two benchmarks from the CPU2000 suite: 175.*vpr* and 300.*twolf*. Figure 13 shows the dendrogram plot comparing the CPU2017 benchmarks, EDA benchmarks and several graph analytics and database applications (which we will discuss next). From the figure, we can clearly see that the EDA benchmarks are close to many CPU2017 applications (especially 505.*mcf_r* and 605.*mcf_s*). Therefore, although the EDA application domain is still not included in new CPU2017 suite, the hardware behavior of the EDA applications are well covered.

E. Case Study on Database Applications

The big-data revolution has created an unprecedented demand for efficient data management solutions. While the traditional data management systems were primarily driven by relational database management systems based on the structured query language (SQL), recent years have seen a rise in the popularity of NoSQL databases. Several prior research studies have compared the CPU2006 benchmarks with the database applications and have concluded that their perfor-

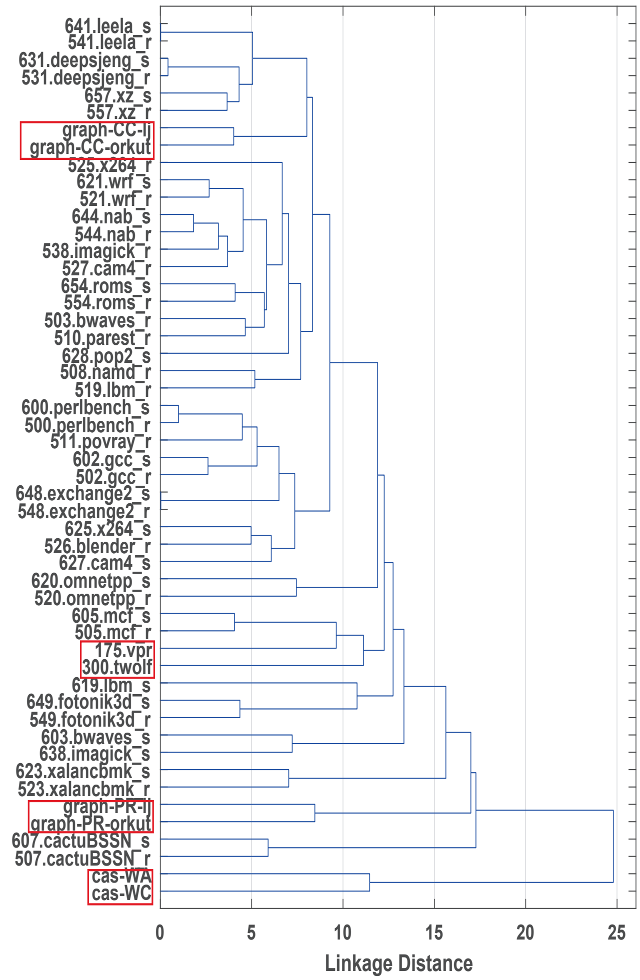


Fig. 13: Similarity among CPU2017, EDA, graph analytics, and database applications.

mance characteristics are highly different [15], [16], [10]. In this section, we compare the performance of the CPU2017 benchmarks with a popular NoSQL database, Cassandra [17] running the Yahoo! Cloud Serving Benchmark (YCSB) [18] benchmarks. Figure 13 shows that the database applications ($cas-WA$ and $cas-WC$) also have very different characteristics than the CPU2017 benchmarks. Deep diving into their performance characteristics, we can see that the difference between the two application classes is primarily caused by their instruction cache and instruction TLB performance.

F. Case Study on Graph Applications

Graph processing workloads [19], [20], [21], [22], [23] have recently gained attention from both system and architecture researchers. Many architects have proposed various hardware accelerators [24], [25] to solve the problem of random memory access from hardware side, as it is one of the major bottlenecks for most graph workloads. To test the balance of SPEC 2017 benchmarks, we compare two popular graph analytics workloads with two real-world graphs. Figure 13 shows that pagerank (pr) has distinct program characteristics with both graph inputs, having high linkage distance due to

TABLE IX: Sensitivity to branch misprediction rate, L1 D-cache miss rate and TLB miss rate. Benchmarks with low sensitivity are not listed.

Branch Prediction	
High	603.bwaves_s, 503.bwaves_r
Medium	544.nab_r, 521.wrf_r, 511.povray_r, 527.cam4_r, 648.exchange2_s, 623.xalancbmk_s, 621.wrf_s, 602.gcc_s, 627.cam4_s, 628.pop2_s
L1 D-cache	
High	549.fotonik3d_r, 649.fotonik3d_s
Medium	548.exchange2_r, 505.mcf_r, 519.lbm_r, 648.exchange2_s, 627.cam4_s, 607.cactuBSSN_s & 628.pop2_s,
L1 D TLB	
High	503.bwaves_r, 507.cactuBSSN_r, 557.xz_r, 511.povray_r, 657.xz_s, 649.fotonik3d_s, 607.cactuBSSN_s
Medium	526.blender_r, 544.nab_r, 508.namd_r, 549.fotonik3d_r, 500.perlbench_r, 521.wrf_r, 541.leela_r, 527.cam4_r, 531.deepsjeng_r 631.deepsjeng_s, 621.wrf_s, 641.leela_s, 600.perlbench_s, 603.bwaves_s,

high L1 TLB activity caused by random data requests [26], [27]. However, Connected Components (*cc*) has very similar hardware performance behavior to SPEC benchmarks, such as the speed and rate versions of *leela*, *deepsjeng* and *xz*. This shows that the newly added benchmarks improve the balance of the suite. Therefore, missing graph applications in CPU2017 suite have not significantly impacted the overall balance of the CPU2017 suite.

G. Sensitivity of CPU2017 Programs to Performance Characteristics

In this section, we present a classification of different CPU2017 programs based on their sensitivity to branch predictors, data cache and TLB configurations across four different machines. To measure the sensitivity of a program to different branch predictor, cache and TLB configurations, we ranked the different CPU2017 programs based on these characteristics on every machine. The difference in ranks of the same benchmark across all machines is used as an indicator of the sensitivity of the benchmark for a specific characteristic.

Table IX shows the classification of different CPU2017 programs based on their sensitivity to branch predictor, L1 data cache and TLB configurations. For every characteristic, benchmarks are categorized into low, medium and highly sensitive categories. The most important observations are as follows: both 503.*bwaves_r* and 603.*bwaves_s* show a lot of variation in terms of branch performance. In terms of data cache performance, 549.*fotonik3d_r* and 649.*fotonik3d_s* show significant performance variability across different machines. In terms of the data TLB performance, the 503.*bwaves_r*, 507.*cactuBSSN_r*, 557.*xz_r*, 511.*povray_r*, 649.*fotonik3d_s* and 607.*cactuBSSN_s* benchmarks experience the greatest variability. One should note that having the highest sensitivity to a parameter does not imply that the benchmark has the worst/best behavior in terms of that parameter. For example, 541.*leela_s*, 641.*leela_r*, 657.*xz_s* and 605.*mcf_s* benchmarks have low sensitivity to branch predictors, because they perform similarly poor across the different machines. In fact, they suffer from the highest misprediction rates across all the systems.

VI. Related Work

Vandierendonck and Bosschere [28] analyzed the CPU2000 benchmarks and identified a smaller benchmark subset that can accurately predict the performance of the entire suite. Similarly, Giladi and Ahituv [29] found that reducing the SPEC89 suite into 6 programs does not affect the SPEC rating. Phansalkar et al. [14] analyzed the redundancy and benchmark balance in CPU2006 suite. Eeckhout et al. [30], [31] leverage PCA and clustering analysis to select representative program inputs for processor design space exploration. Sherwood et al. [32] proposed to use basic block distribution to find representative simulation points for SPEC CPU 2000 benchmarks. Nair et al. [33] leverage this method to generate simpoints for SPEC CPU 2006 benchmark suite. Moreover, Eeckhout et al. [34] studies the (dis)similarity among these benchmarks to reduce the simulation time for entire suite.

Che et al. [35] compared GPU benchmarks from the Rodinia suite to contemporary CMP benchmarks. Sharkawi et al. [36] performed performance projection of HPC applications using the SPEC CFP06 suite. Woodlee [37] compared the SPEC CPU06 suite with SPEC OMP01 suite to study the transferability between them. Goswami et al. [38] and Ryoo et al. [39] performed comprehensive analysis to explore GPGPU workloads, analyzed their performance spectrum and studied the similarity among different GPGPU benchmark suites. Several research studies [40], [15], [10], [16] characterized big-data benchmarks and found that these benchmarks cannot fully represent real world big data workloads. Wu et al. [41], [42] proposed benchmark suites for emerging mobile platform and perform comprehensive studies in terms of energy, thermal and performance.

VII. Conclusion

In this paper, we studied the similarities and redundancies among the CPU2017 benchmarks using performance counter based characterization on several state-of-the-art machines. Our analysis shows that using a subset of 3 programs can accurately predict the performance of SPECrate INT, SPECspeed INT, SPECrate FP, and SPECspeed FP sub-suites with $\geq$93% accuracy. Moreover, we evaluated the representativeness of different input sets of CPU2017 benchmarks, and identified the most representative inputs. We also observed that rate and speed versions of most benchmarks (except *imagick*, *fotonik3d* etc.) have very similar performance characteristics.

To evaluate the balance in the CPU2017 suite, we compared the application domain coverage, the performance and power spectrum of CPU2017 benchmarks to the CPU2006 benchmarks. We observed that the included CPU2017 programs expand the workload coverage area in terms of both performance and power, especially due to the addition of new benchmarks. Furthermore, an analysis from the perspective of program characteristics shows that the CPU2017 programs offer characteristics broader than the EDA programs' space, some overlap with graph analytics, but do not cover the characteristics from the Cassandra workloads. We believe that our comprehensive analysis can guide the usage of this suite and benefit the architecture community.

VIII. Acknowledgements

We enjoyed the opportunity to interact with the SPEC CPU Subcommittee and to provide clustering analysis to SPEC for selection of CPU2017 programs. The authors express their gratitude to John Henning for providing technical support/data and to Jeff W. Reilly for facilitating the clustering analysis. The authors would also like to thank Texas Advanced Computing Center (TACC) at UT Austin for providing compute resources. The authors of this work are supported partially by National Science Foundation (NSF) under grant numbers 1725743 and 1745813. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or other sponsors.

References

[1] "SPEC CPU2017." https://www.spec.org/cpu2017.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, Aug. 2011.

[3] A. Patel, F. Afram, S. Chen, and K. Ghose, "Marssx86: A full system simulator for x86 cpus," 2011.

[4] C. Jiang, Z. Yu, L. Eeckhout, H. Jin, X. Liao, and C. Xu, "Two-level hybrid sampled simulation of multithreaded applications," *ACM Trans. Archit. Code Optim.*, vol. 12, pp. 39:1–39:25, Nov. 2015.

[5] T. E. Carlson, W. Heirmant, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *SC*, 2011.

[6] C. Jiang, Z. Yu, H. Jin, C. Xu, L. Eeckhout, W. Heirman, T. E. Carlson, and X. Liao, "Pcantorsim: Accelerating parallel architecture simulation through fractal-based sampling," *ACM Trans. Archit. Code Optim.*, vol. 10, pp. 49:1–49:24, Dec. 2013.

[7] "SPEC CPU2006." https://www.spec.org/cpu2006.

[8] "Linux perf tool." https://perf.wiki.kernel.org/index.php/Main_Page.

[9] A. Phansalkar, A. Joshi, L. Eeckhout, and L. John, "Measuring program similarity: Experiments with spec cpu benchmark suites," *ISPASS*, pp. 10–20, 2005.

[10] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *ASPLOS*, (New York, NY, USA), pp. 37–48, ACM, 2012.

[11] W. Gao, Y. Zhu, Z. Jia, C. Luo, L. Wang, Z. Li, J. Zhan, Y. Qi, Y. He, S. Gong, X. Li, S. Zhang, and B. Qiu, "Bigdatabench: a big data benchmark suite from web search engines," *CoRR*, vol. abs/1307.0320, 2013.

[12] A. Yasin, "A top-down method for performance analysis and counters architecture," *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, vol. 00, pp. 35–44, 2014.

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

[14] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the spec cpu2006 benchmark suite," in *ISCA*, 2007.

[15] R. Panda, C. Erb, M. Lebeane, J. Ryoo, and L. K. John, "Performance characterization of modern databases on out-of-order cpus," in *IEEE SBAC-PAD*, 2015.

[16] R. Panda and L. K. John, "Data analytics workloads: Characterization and similarity analysis.," in *IPCCC*, pp. 1–9, IEEE, 2014.

[17] "Cassandra." wiki.apache.org/cassandra/FrontPage.

[18] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *SoCC*, pp. 143–154, 2010.

[19] M. LeBeane, S. Song, R. Panda, J. H. Ryoo, and L. K. John, "Data partitioning strategies for graph workloads on heterogeneous clusters," in *SC*, 2015.

[20] S. Song, M. Li, X. Zheng, M. LeBeane, J. H. Ryoo, R. Panda, A. Gerstlauer, and L. K. John, "Proxy-guided load balancing of graph processing workloads on heterogeneous clusters," in *ICPP*, 2016.

[21] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *OSDI*, 2012.

[22] Z. Ai, M. Zhang, Y. Wu, X. Qian, K. Chen, and W. Zheng, "Squeezing out all the value of loaded data: An out-of-core graph processing system with reduced disk i/o," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, (Santa Clara, CA), pp. 125–137, USENIX Association, 2017.

[23] M. Zhang, Y. Wu, K. Chen, X. Qian, X. Li, and W. Zheng, "Exploring the hidden dimension in graph processing," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, (Savannah, GA), pp. 285–300, USENIX Association, 2016.

[24] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "Graphpim: Enabling instruction-level pim offloading in graph computing frameworks," in *HPCA*, 2017.

[25] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[26] J. H. Ryoo, N. Gulur, S. Song, and L. K. John, "Rethinking tlb designs in virtualized environments: A very large part-of-memory tlb," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, (New York, NY, USA), pp. 469–480, ACM, 2017.

[27] Y. Marathe, N. Gulur, J. H. Ryoo, S. Song, and L. K. John, "Csalt: Context switch aware large tlb," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, (New York, NY, USA), pp. 449–462, ACM, 2017.

[28] H. Vandierendonck and K. De Bosschere, "Many benchmarks stress the same bottlenecks," in *Workshop on Computer Architecture Evaluation Using Commercial Workloads*, vol. 2, pp. 57–64, 2004.

[29] R. Giladi and N. Ahitav, "Spec as a performance evaluation measure," *Computer*, vol. 28, pp. 33–42, Aug 1995.

[30] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere, "Workload design: Selecting representative program-input pairs," *PACT*, vol. 0, p. 83, 2002.

[31] M. B. Breughe and L. Eeckhout, "Selecting representative benchmark inputs for exploring microprocessor design spaces," *ACM Trans. Archit. Code Optim.*, vol. 10, pp. 37:1–37:24, Dec. 2013.

[32] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *PACT*, pp. 3–14, 2001.

[33] A. A. Nair and L. K. John, "Simulation points for spec cpu 2006," in *2008 IEEE International Conference on Computer Design*, pp. 397–403, 2008.

[34] L. Eeckhout, J. Sampson, and B. Calder, "Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation," in *IEEE IISWC*, pp. 2–12, 2005.

[35] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads," in *IISWC*, 2010.

[36] S. Sharkawi, D. DeSota, R. Panda, R. Indukuru, S. Stevens, V. Taylor, and X. Wu, "Performance projection of hpc applications using spec cfp2006 benchmarks," in *IPDPS*, 2009.

[37] E. Ould-Ahmed-Vall, K. A. Doshi, C. Yount, and J. Woodlee, "Characterization of spec cpu2006 and spec omp2001: Regression models and their transferability," in *ISPASS*, 2008.

[38] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring gpgpu workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *IISWC*, 2010.

[39] J. H. Ryoo, S. J. Quirem, M. Lebeane, R. Panda, S. Song, and L. K. John, "Gpgpu benchmark suites: How well do they sample the performance spectrum?," in *ICPP*, pp. 320–329, 2015.

[40] W. Xiong, Z. Yu, Z. Bei, J. Zhao, F. Zhang, Y. Zou, X. Bai, Y. Li, and C. Xu, "A characterization of big data benchmarks," in *2013 IEEE International Conference on Big Data*, 2013.

[41] D. Pandiyan, S. Y. Lee, and C. J. Wu, "Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite - mobilebench," in *IISWC*, pp. 133–142, 2013.

[42] C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, and C. J. Wu, "A study of mobile device utilization," in *ISPASS*, pp. 225–234, 2015.