Evaluating Benchmark Subsetting Approaches

Joshua J. Yi[1], Resit Sendag[2], Lieven Eeckhout[3], Ajay Joshi[4], David J. Lilja[5], and Lizy K. John[4]

[1] – Networking and Computing Systems Group
Freescale Semiconductor, Inc.
Austin, TX
joshua.yi@freescale.com

[2] - Department of Electrical and Computer Engineering
University of Rhode Island
Kingston, RI
sendag@ele.uri.edu

[3] - Department of Electronics and Information Systems
Ghent University
Ghent, Belgium
leeckhou@elis.ugent.be

[4] - Department of Electrical and Computer Engineering
University of Texas
Austin, TX
{ajoshi, ljohn}@ece.utexas.edu

[5] - Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN
lilja@ece.umn.edu

Abstract

To reduce the simulation time to a tractable amount or due to compilation (or other related) problems, computer architects often simulate only a subset of the benchmarks in a benchmark suite. However, if the architect chooses a subset of benchmarks that is not representative, the subsequent simulation results will, at best, be misleading or, at worst, yield incorrect conclusions. To address this problem, computer architects have recently proposed several statistically-based approaches to subset a benchmark suite. While some of these approaches are well-grounded statistically, what has not yet been thoroughly evaluated is the: 1) Absolute accuracy, 2) Relative accuracy across a range of processor and memory subsystem enhancements, and 3) Representativeness and coverage of each approach for a range of subset sizes. Specifically, this paper evaluates statistically-based subsetting approaches based on principal components analysis (PCA) and the Plackett and Burman (P&B) design, in addition to prevailing approaches such as integer vs. floating-point, core vs. memory-bound, by language, and at random. Our results show that the two statistically-based approaches, PCA and P&B, have the best absolute and relative accuracy for CPI and energy-delay product (EDP), produce subsets that are the most representative, and choose benchmark and input set pairs that are most well-distributed across the benchmark space. To achieve a 5% absolute CPI and EDP error, across a wide range of configurations, PCA and P&B typically need about 17 benchmark and input set pairs, while the other five approaches often choose more than 30 benchmark and input set pairs.

1 Introduction

Despite the introduction of reduced-time simulation techniques such as SimPoint [11], SMARTS [15], and MinneSPEC [7], and increasingly faster platforms to simulate or run on, computer architecture researchers frequently only simulate a subset of the benchmarks in a benchmark suite [1]. Computer architects also subset a group of benchmarks when selecting benchmarks for a new benchmark suite, which allows them to capture the most important behavior and characteristics of the application space with fewer benchmarks and/or eliminate redundancy. Since the benchmarks in a benchmark suite ostensibly represent the programs that are commonly used in a particular application domain, at the very least, subsetting the suite means that the architect will not use all of the programs that are representative of the application space. In the worst case, using a subset of benchmarks can result in different conclusions than if the architect used the entire suite. Additionally, when architects use different approaches to subset a benchmark suite, inevitably, they evaluate different subsets, which makes it more difficult to compare the results across papers.

To help computer architects subset benchmark suites more intelligently, researchers have proposed several approaches, including: subsetting benchmark suites based on their most significant characteristics (*e.g.*, *principal components analysis* (PCA) [2] or based on their performance bottlenecks using a *Plackett and Burman* (P&B) *design* [16]. Both of these benchmark subsetting approaches are statistically-based and can be microarchitecturally independent. Other benchmark subsetting approaches that are less statistically-rigorous, but perhaps as effective, include subsetting benchmarks based on 1) The percentage of floating-point instructions (*i.e.*, integer vs. floating-point benchmarks) in the benchmarks, 2) If the benchmark is computationally-bound or memory-bound, and 3) Language (*e.g.*, C vs. C++ vs. FORTRAN) or infrastructure (*e.g.*, Availability of reduced input sets, language-specific compilers, *etc.*). Finally, computer architects also frequently choose benchmarks from a benchmark suite at random. Ideally, the subset of benchmarks should have good *absolute* and *relative accuracy*, as compared to the entire benchmark suite, as well as adequately covering the space of the suite's characteristics.

Given all of these approaches, there are several unanswered questions. First, which approach yields the most accurate/representative subset of benchmarks, both from the absolute and relative points-of-view? Not only does measuring the accuracy of the various subsetting approaches have its own intrinsic merit, but determining which subsetting approach is the most accurate could be the first step towards creating a "standard" subset of benchmarks to facilitate cross-paper comparison. Second, in addition to their accuracy, how well does each of these approaches cover the space of characteristics of the entire benchmark suite? Finally, which of these approaches yields the most accurate subset of benchmarks, with the least subsetting, *i.e.*, profiling cost?

To answer these questions, we compare the aforementioned subsetting approaches across the range of subset sizes (*i.e.*, number of benchmark and input set pairs in the subset) and for several different processor configurations. More specifically, we compare the absolute accuracy of each subsetting approach against the entire suite for CPI and energy-delay product (EDP) and evaluate the relative accuracy of the subsetting approaches for three microarchitectural enhancements to determine if using subsets can yield accurate estimates of the speedup. Finally, we quantify the respresentativeness by measuring how well the benchmarks of the subset are distributed over the entire space (of microarchitectural metrics).

The contributions of this paper are as follows:

1. This paper compares the absolute accuracy (CPI and EDP) of the seven most prevalent or emerging benchmark subsetting approaches across a wide range of processor configurations and benchmark subset sizes.
2. This paper compares the relative accuracy of these subsetting approaches for three processor core and memory subsystem enhancements across a wide range of both processor configurations and benchmark subset sizes.
3. This paper compares the representativeness and coverage of each subsetting approach.
4. This paper presents the most accurate subsetting results for the entire SPEC CPU 2000 benchmark suite that computer architects can use "out-of-the-box".

The remainder of this paper is organized as follows: Section 2 describes the subsetting approaches and the specific permutations of each that this paper examined and some relevant related work. Section 3 describes the simulation methodology that we followed in this paper, Section 4 presents the results and our analysis, and, Section 5 discusses the subsetting cost of the statistically-based approaches. Finally, Section 6 summarizes.

2 Description of Benchmark Subsetting Approaches

This section describes the seven benchmark subsetting approaches that we examined in this paper. The two statistically-based approaches subset a benchmark suite based on the: 1) Benchmark's principal components using PCA and 2) Performance bottlenecks that the benchmark induces using the P&B design. The next four approaches, which are not statistically-based, subset a benchmark suite based on: 3) The percentage of floating-point instructions (*i.e.*, integer vs. floating-point), 4) If the benchmark is core-bound or memory-bound, 5) The programming language (*e.g.*, C vs. FORTRAN), and 6) Randomly selecting the benchmarks. Finally, the seventh and last subsetting approach subsets a benchmark suite based on the benchmark's frequency of use, as reported in [1], in HPCA, ISCA, and MICRO.

2.1 Statistically-Based Subsetting Approaches

2.1.1 Subsetting by using Principal Components Analysis

Eeckhout *et al*. [2, 3, 4] and Phansalkar *et al.* [9] proposed using PCA[1] as a statistically-rigorous way of subsetting a benchmark suite. The first step to use PCA as a subsetting tool is to profile the benchmarks to determine their key characteristics. In this paper, we profile the benchmarks for the same program characteristics that were used in [3], which include: 1) Instruction mix (Loads, stores, control transfers, arithmetic operations, integer multiplies and floating-point operations), 2) ILP assuming an idealized processor, but a limited (32, 64, 128, and 256) instruction window, 3) Register characteristics (Average degree of use, average number of times a register instance is consumed since it was originally written, and number of instructions between a write to and read from that register), 4) Working set size (Number of unique 4KB pages for both instruction and data and 32-byte blocks that were touched), 5) Data stream strides (Number of memory addresses between temporally adjacent memory accesses for the same static load/store instruction and for different static load/store instructions), and 6) Branch predictability.

After profiling the benchmarks to determine their key program characteristics, the second step is to use PCA to compute the principal components, which are linear combinations of the original variables, such that all principal components are uncorrelated. PCA builds on the assumption that many variables – in our case, program characteristics – are correlated and hence, they measure the same or similar properties of the benchmark and input pairs. Using PCA to transform the program characteristics to principal components has two key advantages over using the original program characteristics to subset the benchmarks. First, since the principal components are uncorrelated, using them to subset the benchmarks produces better clusters, since two correlated program characteristics that essentially measure the same thing are not counted twice. Second, using principal components reduces the dimensionality of the data by determining the most important program characteristics, which also yields a better clustering result.

After determining the principal components of the

[1] In this paper, we do not compare *independent component analysis* (ICA) [3] as, for the same input data, ICA produces the same results as PCA, as shown in the correction note to [4] available at http://www.elis.ugent.be/~leeckhou/.

benchmarks, the next step is to cluster the benchmarks based on their Euclidean distances away from each other and to form the subsets by choosing one benchmark from each cluster. For more information on how to use PCA to subset benchmark suites, see [2].

In this paper, to evaluate the efficacy of PCA as a benchmark subsetting approach, we cluster and subset the benchmarks based on the 4, 5, 6, 7, and 8 most significant principal components.

2.1.2 Subsetting by Performance Bottlenecks, using the Plackett and Burman Design

Yi *et al.* in [16] proposed using the P&B design to characterize the performance bottlenecks that a benchmark induces when running on a processor. The P&B design specifies a very wide range of processor configurations that the architect needs to run the benchmarks for. After running the benchmarks, the architect calculates the magnitude of each performance bottleneck by using the P&B design matrix and the results (*e.g.*, execution time, energy-per-instruction, *etc.*); the larger the magnitude, the more significant that performance bottleneck is, and vice-versa. For more information about using the P&B design and potential applications in computer architecture, see [16].

After characterizing the performance bottlenecks, Yi *et al.* proposed clustering and subsetting the benchmarks based on their performance bottlenecks [16]. The downside of this approach, compared to PCA, is that it requires an inordinate amount of simulation time. In our case, the P&B design required 88 simulations per-benchmark with a detailed out-of-order superscalar processor simulator while PCA required only six instrumented runs, which could be consolidated into a single, monolithic instrumented run.

In this paper, we characterized the performance bottlenecks by using processor configurations that were similar to those specified in [16]. We used *foldover* [8] to remove the effects of two-bottleneck interactions from each of the single bottlenecks. We calculated the magnitude of each performance bottleneck based on the CPI from each of the 88 processor configurations. We clustered the benchmarks based on the: 1) Ranks of each performance bottleneck, 2) Magnitudes of each, 3) Percentage of the total variation accounted for by each single bottleneck, and 4) Percentage of the total variation accounted for by each single bottleneck and all interactions. Finally, we clustered the benchmarks based all (43) bottlenecks and based on the Top 3, 5, and 7 bottlenecks across all benchmarks and for each benchmark. For the latter clusterings (Top 3, 5, and 7), the value of the bottlenecks that are not in the Top 3, 5, or 7 were set to the lowest possible value.

2.2 Prevailing Subsetting Approaches

This subsection describes the subsetting approaches that computer architects frequently use. The following list of subsetting approaches and specific permutations of each were based on the papers that we have read over the years and the subsetting rationale in those papers. For each of the approaches that we describe in this subsection, to ensure that our results are not a by-product of the specific benchmark and input sets pairs, we randomly form thirty subsets for each subset size and average the results.

2.2.1 Prevalence of Floating-Point Instructions (Integer vs. Floating-Point)

One of the most popular approaches that computer architects use to subset a benchmark suite such as SPEC CPU is to evaluate the benchmarks that are in the integer or floating-point subsuites only. Furthermore, for a benchmark suite such as SPEC CPU 2000, the architect may even subset the SPECint or SPECfp subsuites. One rationale that architects frequently give for restricting the subsets to the integer or floating-point benchmarks only are the memory access patterns and branch predictabilities of the benchmarks within each subsuite.

In this paper, for this approach, we randomly select X benchmark and input set pairs from the subsuite, where X is the number of benchmark and input set pairs in the subset. The value of X ranges from 1 to N-1, where N is the maximum number of benchmark and input set pairs in each subsuite. (There are 31 and 15 benchmark and input sets pairs for SPECint 2000 and SPECfp 2000.)

2.2.2 Core-Bound vs. Memory-Bound

Another popular approach is to subset the benchmark suite based on if the benchmarks are core-bound (compute-bound) or memory-bound. Obviously, simulating benchmarks that are core-bound will not distinctively illustrate the efficacy of a prefetching algorithm, for example. We classified the SPEC CPU 2000 benchmarks as memory-bound if their L1 D-cache miss rate was greater than 6% for a 32KB, 2-way L1 D-cache miss rate (There was a separation of benchmarks around 6%). We also checked the cache miss rates for several other cache configurations to ensure that we properly classified the benchmarks. The benchmark and input set pairs that were classified as memory-bound are as follows: *gzip-program*, *gzip-source*, *swim*, *mgrid*, *applu*, *gcc-166*, *gcc-integrate*, *galgel*, *art-110*, *art-470*, *mcf*, *equake*, *ammp*, *lucas*, and *twolf*; the other 31 benchmark and input set pairs are core-bound.

To form the subset for each group of benchmark and input set pairs, benchmark and input set pairs were randomly selected from each group (core-bound or memory-bound).

2.2.3 Language/Compiler (C vs. FORTRAN)

For various reasons, including simulation infrastructure and the availability of modern compilers, computer architects often subset the benchmarks based on the language in which the benchmark is written. An excellent example of this approach occurs when computer architects use the PISA ISA from the SimpleScalar tool suite. Since the only compiler that compiles benchmarks into PISA binaries is `gcc` 2.63, the only benchmarks that SimpleScalar-PISA users can compile and run are C benchmarks, without using `f2c`. In SPEC CPU 2000, this restricts the list of benchmarks to *gzip*, *vpr*, *gcc*, *mesa*, *art*, *mcf*, *crafty*, *ammp*, *parser*, *perl*, *gap*, *vortex*, *bzip2*, and *twolf*.

In this paper, we divide the benchmarks into two groups: C-based and FORTRAN-based; we group FORTRAN 77 and FORTRAN 90 benchmarks together. (Although these two groups exclude *eon* as a benchmark candidate because it is written in C++, the omission of a single benchmark should not significantly alter the results.)

2.2.4 Random

Based on the lack of justification as to why architects chose specific benchmarks [1, 17], the most popular subsetting approach is probably random. In this paper, to form the subset for each group of benchmark and input set pairs, we randomly select X benchmark and input set pairs for each subset of size X. In our evaluation, the probability of selecting integer and floating-point benchmarks in a subset is independent.

2.2.5 High-Frequency

Finally, in addition to the seven approaches that were previously described in this section, we also add "high-frequency" to the list of approaches. This "approach" subsets benchmarks based on their frequency-of-use in papers. The frequency-of-use for each benchmark was taken from [1], which tabulated the number of times specific SPEC CPU 2000 benchmarks were used in HPCA, ISCA, and MICRO from 2000 to 2003.

Using this approach, the most popular benchmark, *gzip*, was present in all subsets, while the second most-popular benchmark, *gcc*, was present only in subsets with 2 or more benchmarks, *etc*. Therefore, the maximum number of benchmarks in a subset is 26, which is the number of benchmarks in the SPEC CPU 2000 benchmark suite. Finally, since this listing only tabulates the frequency-of-use by benchmark, and not by benchmark and input set pair, we created two sets of subsets. The first only uses only input set per-benchmark while the second uses all input sets for that benchmark. In the former case, the input set that has a CPI that is closest to the average CPI of input sets and across all processor configurations was chosen to be the input set. Since *art* only has two input sets, we randomly selected one (110). For example, in the former case (1 input set), the *rushmeier* input set was selected as *eon*'s input set for all subsets, while in the latter case, all three input sets (*cook*, *kajiya*, and *rushmeier*) were included.

2.2.6 Other Subsetting Approaches

Machine ranking-based subsetting: Vandierendonck and De Bosschere [14] presented an experimental evaluation of the reliability of four clustering algorithms (k-means, hierarchical, forward PCA and backward PCA) to form a representative subset of programs from the SPEC CPU2000 benchmark suite. They evaluated the subsetting algorithms based on their ability to produce a subset of the SPEC benchmarks that ranks computer systems in the same way as the full benchmark suite. In their follow-on study [13], they also ranked programs in the SPEC CPU2000 benchmark suite using the SPEC peak performance rating. The program ranks were based on their uniqueness, *i.e.*, the programs that exhibit different speedups on most of the machines were given a higher ranking as compared to other programs in the suite. This criterion of subsetting programs is more appropriate for a customer comparing computer systems because this approach looks for the corner cases in the workload space. In addition, this approach may be difficult to use in practice for subsetting purposes because it requires that all benchmarks be measured on a large set of machines (340 machines in their setup). For these reasons, we do not include this technique in our comparison study.

Subsetting based on programming language characteristics: Saavedra and Smith [10] measured similarity between applications using workload characteristics at a program language level. Considering that modern day microprocessors are out-of-order and compilers are becoming more effective in optimizing programming language constructs, it is unlikely that these characteristics will relate well to the performance of the program. Therefore, we did not include this technique in our evaluation.

2.3 Summary of Subsetting Approaches

Table 1 summarizes the subsetting approaches that we evaluated in this paper. The first column lists the approach while the second column lists the maximum subset size, based on the type (integer vs. floating-point, core vs. memory, and C vs. FORTRAN). The third column lists the permutations of each approach and the number of permutations.

3 Simulation Methodology

To gather the program characteristics for PCA, we instrumented the benchmark binaries with ATOM. To gather the profiling data for the P&B design simulations and the results in Section 4, we used SMARTS [15] with sampling and functional warming on, and while simulating 25,000 samples initially. The detailed simulation and warm-up lengths per sampling unit were 1000 and 2000 instructions, respectively. We used a confidence level of 99.7% with +/- 3% confidence interval. All SMARTS simulations were run until the sampling frequency was greater than the recommended frequency. For the results in Section 4.2, we added next-line prefetching as described in [6] to the base version of SMARTS.

Table 2 lists the key parameters for the eight processor configurations that we used for the base processor simulations. For each issue width, the "aggressiveness" of the processor increases from left to right. For this paper, since we wanted to use a set of benchmarks that had a wide range of behavior and represented a wide range of applications, *i.e.*, general-purpose computing, we decided to use the SPEC CPU 2000 benchmark suite. We downloaded pre-compiled Alpha binaries from [12]. Also, to help architects directly use our subsetting results, we evaluated all 26 benchmarks and all benchmark and input set pairs (with the exceptions of *vpr-place* and *perlbmk-perfect*, as they both crash SMARTS).

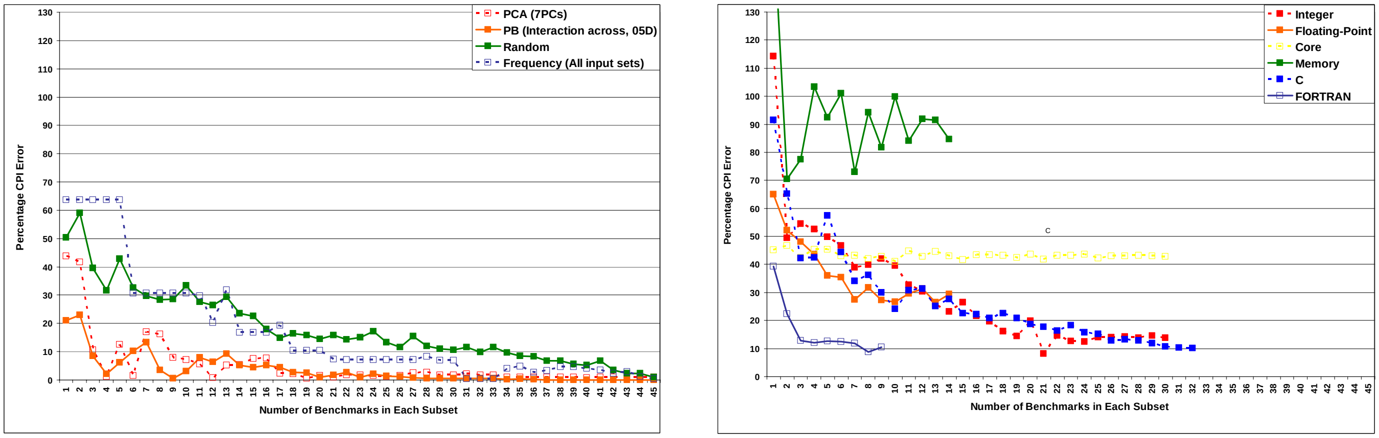
In this paper, with the exception of the high-frequency subsetting approach, we subset the SPEC CPU 2000 based on benchmark and input set pairs, instead of just benchmarks. Our rationale for doing so is based on our observation on how

Table 1. Summary of benchmark subsetting approaches

Approach	Max Benchmark + Input Sets Per Subset	Permutations (Total)
PCA	45	4, 5, 6, 7, and 8 principal components (5)
Performance bottleneck (P&B)	45	Top 3, 5, 7, and all bottlenecks per-benchmark (12) Top 3, 5, 7 and all bottlenecks across-benchmarks (16)
Integer vs. Floating-point	30 (Integer) 14 (Floating-Point)	30 randomly created subsets (30)
Core-bound vs. Memory-bound	30 (Core) 14 (Memory)	30 randomly created subsets (30)
Language	32 (C) 9 (FORTRAN)	30 randomly created subsets (30)
Random	45	30 randomly created subsets (30)
High-frequency	45 / 26	All input sets per benchmark / 1 input set (2)

Table 2. Key parameters for eight base processor configurations

Parameter	Configuration							
	#1	#2	#3	#4	#5	#6	#7	#8
Decode, issue, commit width	4-way				8-way			
Branch predictor, BHT entries	Combined, 2K	Combined, 4K	Combined, 4K	Combined, 8K	Combined, 4K	Combined, 8K	Combined, 16K	Combined, 32K
ROB/LSQ entries	16/8	32/16	32/16	64/32	64/32	128/64	128/64	256/128
Int/FP ALUs, (mult/div units)	2/2, (1/1)	4/4, (4/4)	2/2, (1/1)	4/4, (4/4)	6/6, (4/4)	8/8, (8/8)	6/6, (4/4)	8/8, (8/8)
L1 D-cache size, assoc, latency (cycles)	8KB, 2-way, 1	16KB, 4-way, 1	32KB, 2-way, 1	64KB, 4-way, 1	16KB, 4-way, 1	32 KB, 4-way, 1	128 KB, 2-way, 1	256 KB, 4-way, 1
L2 cache size, assoc, latency (cycles)	128KB, 4-way, 15	256KB, 4-way, 12	256KB, 4-way, 10	512KB, 8-way, 7	1024KB, 4-way, 25	2048KB, 8-way, 20	1024KB, 4-way, 15	2048KB, 8-way, 12
Memory latency (cycles): First, following	150, 10	100, 5	150, 10	100, 5	300, 20	200, 10	300, 20	200, 10

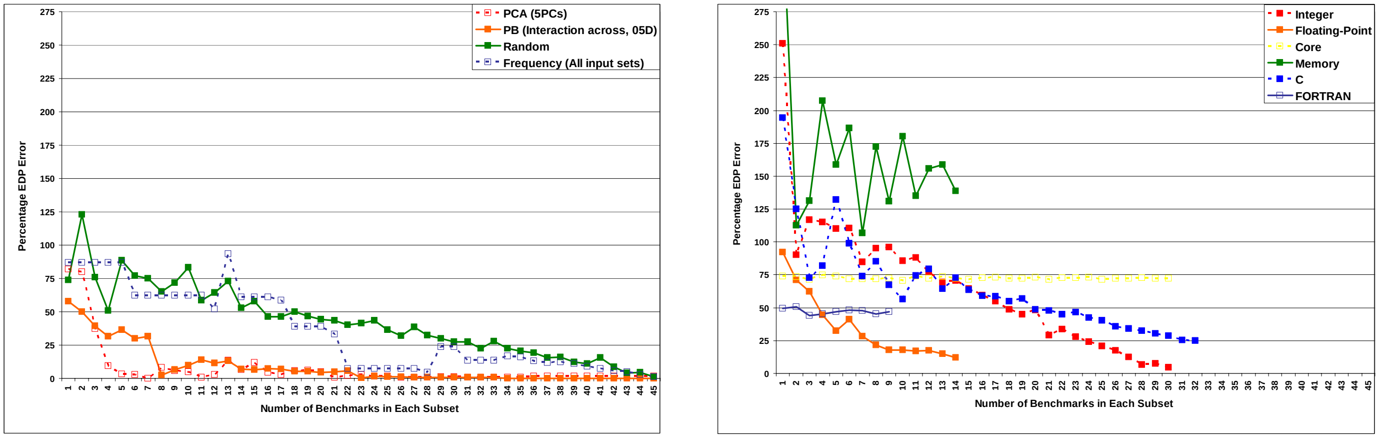


A. PCA, P&B, Random, and High-frequency

B. Int/Float, Core/Memory, and C/FORTRAN

Figure 1. Absolute CPI accuracy of subsetting approaches, configuration #3

computer architects subset benchmark suites. Namely, architecture researchers seem to choose a benchmark and then choose one input set for that benchmark, thereby choosing a specific benchmark and input set pair, as opposed to choosing a benchmark and **all** of its input sets. Therefore, to maximize the utility of the results and conclusion that we present in this paper, we follow the lead of our colleagues and subset SPEC CPU 2000 based on benchmark and input set pairs. Note that,



A. PCA, P&B, Random, and High-frequency

B. Int/Float, Core/Memory, and C/FORTRAN

Figure 2. Absolute EDP accuracy of subsetting approaches, configuration #3

in the remainder of this paper, for brevity, we often use the term "benchmarks" to represent both benchmarks and input sets.

4 Results and Analysis of Prevailing and Emerging Benchmark Subsetting Approaches

The following three subsections present the benchmark subsetting results and our analysis from the perspectives of absolute accuracy, relative accuracy, and representativeness/coverage.

4.1 Absolute Accuracy

Good absolute accuracy is important for processor designers who want to run a small representative subset of the benchmarks for performance estimation (*e.g.*, estimating the SPEC score) and for HDL verification. For others, good absolute accuracy is important for profiling, workload characterization, and performance estimation.

4.1.1 Absolute Accuracy of CPI Results and Analysis

Figure 1 presents the absolute accuracy of the various subsetting approaches for CPI for configuration #3. (The results for the other 7 configurations were extremely similar.) Figure 1A presents the absolute accuracy results for the PCA, P&B, random, and high-frequency subsetting approaches, while Figure 1B presents the results for the other approaches (Integer vs. floating-point, core-bound vs. memory-bound, and C vs. FORTRAN).

The x-axis in Figure 1 shows the number of benchmark and input set pairs in each subset while the y-axis shows the percentage CPI error for each subset size, as compared to the full SPEC CPU 2000 benchmark suite. Since there are a total of 46 benchmark and input set pairs, the maximum number of benchmark and input set pairs that can be in a subset is 45, which is the maximum x-value.

Figure 1A shows several "plateaus" in the percentage CPI error for the high-frequency ("frequency") subsetting approach. Since [1] only lists the benchmarks in order of frequency, *i.e.*, does not list the frequency of the specific benchmark and input set pairs, for the "all input set" permutation of the high-frequency approach, to represent all input sets, we assign the same percentage CPI error to all input sets for that benchmark. For example, *gzip*, which is the most frequently used benchmark, has five input sets. To represent each of its five input sets, we set the percentage CPI error for each input set to be 63.7%, which is the overall percentage CPI error for *gzip*.

The results in Figure 1 show that PCA (with 7 principal components) and P&B (Top 5 bottlenecks by percentage of the total variation in the CPI) are clearly the most accurate – and consistently so – subsetting approaches. In particular, the percentage CPI error is consistently less than 5% for subset sizes larger than 17. In contrast, using any of the other five approaches to subset SPEC CPU 2000 results in subsets that: 1) Are not representative of the entire benchmark suite (*i.e.*, have high percentage CPI error) or 2) Have inconsistent accuracy for larger subset sizes. An example of the former is the random approach where the percentage CPI error is still greater than 15% when half (23) of the benchmark and input sets are in the subset. An example of the latter is the memory-bound subsetting approach since its percentage CPI error alternately increases and decreases with increasing subset size. These two results illustrate the efficacy of the statistically-based subsetting approaches.

From the point-of-view of simulation time vs. accuracy, where the simulation time is proportional to the number of benchmark and input set pairs in the subset, PCA, P&B, and high-frequency are the only three approaches where the percentage CPI error is less than 5% for consecutive subset sizes. (The percentage CPI error never dips below 5% for consecutive subset sizes for the other four subsetting approaches.) To achieve this level of absolute accuracy, PCA and P&B require 17 benchmark and input set pairs only, or, approximately one-third of the entire benchmark suite. By contrast, the high-frequency approach requires 39 benchmark and input set pairs, or, approximately 80% of the entire suite.

4.1.2 Absolute Accuracy of EDP Results and Analysis

In addition to CPI, EDP is another important metric that measures the processor’s efficiency. The formula to compute the EDP is: EDP = CPI * EPI (where EPI is the average energy consumed while executing one instruction). Figures 2A and 2B are the corresponding figures to Figures 1A and 1B for EDP accuracy, again for processor configuration #3. The results for the other configurations were, again, extremely similar.

The conclusions from Figure 2 are exactly the same as those from Figure 1. Namely, PCA and P&B are the most accurate subsetting approaches while the other approaches are generally very inaccurate. This conclusion is not particularly surprising given the CPI results that were presented in Figures 1A and 1B.

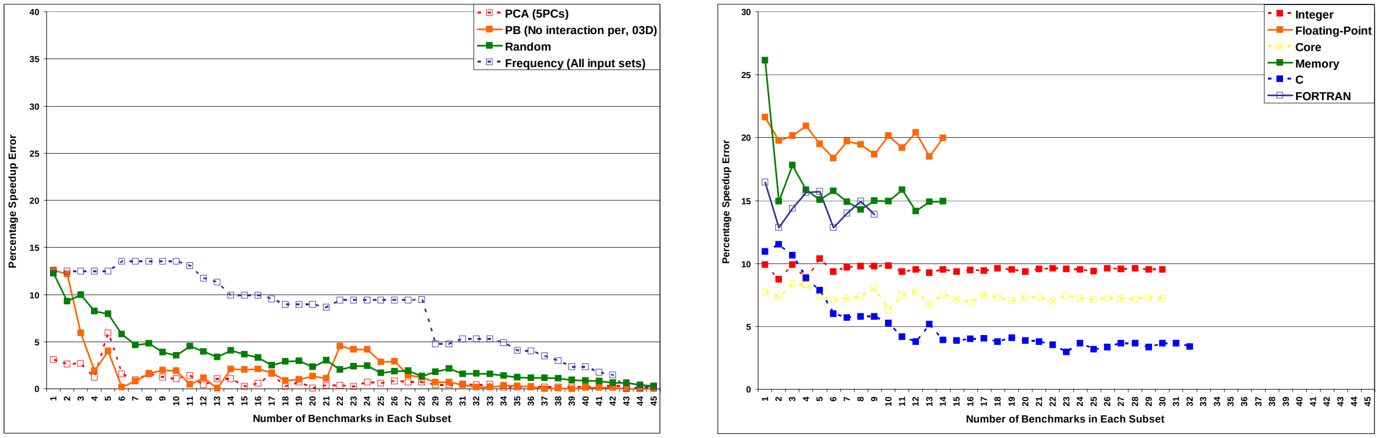
4.1.3 Comparison of Subsets for Principal Component Analysis, Plackett and Burman, and High-Frequency

Table 3 shows benchmark and input set pairs that the PCA, P&B, and high-frequency subsetting approaches select for subset sizes of 10, 15, 20, and 25. Note that the high-frequency approach is the *de facto* approach in computer architecture research. By comparing the performance metrics of each PCA or PB subsets vs. the performance metrics of the high-frequency subset, we can determine why the high-frequency subsetting approach has lower absolute accuracy.

The results of this analysis shows that, for all four subset sizes, the high-frequency subsets are less accurate than the two statistically-based approaches because they include benchmark

Table 3. Comparison of the subsets for PCA, P&B, and High-frequency subsets for 10, 15, 20, and 25 benchmark and input sets. Weights are shown only for the pairs that are in each subset.

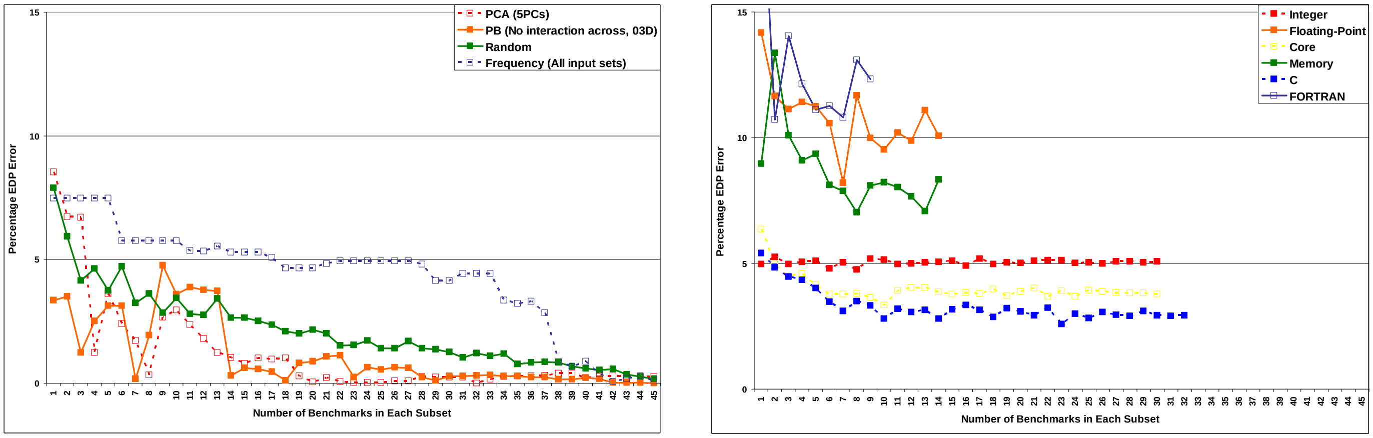
		Principal Components Analysis				Plackett and Burman				High-frequency			
Benchmark	**Input Set**	**10**	**15**	**20**	**25**	**10**	**15**	**20**	**25**	**10**	**15**	**20**	**25**
gzip	***graphic***				0.067			0.087					
	log												
	program				0.044		0.130	0.109		0.100	0.067	0.050	0.040
	random						0.065		0.065				
	source		0.133	0.111					0.043				
wupwise								0.043	0.022				0.040
swim		0.022	0.022	0.022	0.022				0.022			0.050	0.040
mgrid				0.022	0.022	0.087	0.065						0.040
applu				0.022	0.022		0.043	0.043	0.022			0.050	0.040
vpr	***route***		0.022	0.022	0.022			0.022	0.022	0.100	0.067	0.050	0.040
gcc	***166***				0.044								
	200												
	expr	0.111	0.111	0.111	0.067	0.152							
	integrate						0.130	0.043	0.043	0.100	0.067	0.050	0.040
	scilab							0.065	0.065				
mesa						0.239						0.050	0.040
galgel			0.022	0.022	0.022	0.065	0.065	0.043	0.022				0.040
art	***110***	0.044	0.044	0.044	0.044		0.065	0.065			0.067	0.050	0.040
	470					0.087			0.043				
mcf		0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.100	0.067	0.050	0.040
equake		0.067	0.022	0.022	0.022	0.043	0.022	0.022	0.022		0.067	0.050	0.040
crafty					0.022					0.100	0.067	0.050	0.040
facerec				0.089	0.067				0.022				0.040
ammp				0.022	0.022		0.022	0.022	0.022		0.067	0.050	0.040
lucas			0.022	0.022	0.022		0.022	0.022	0.022			0.050	0.040
fma3d			0.111					0.043	0.043				
parser								0.022	0.022	0.100	0.067	0.050	0.040
sixtrack		0.178	0.067	0.022	0.022	0.022	0.022	0.022	0.022				0.040
eon	***cook***												
	kajiya				0.022			0.109	0.087				
	rushmeier	0.089	0.067	0.067	0.044						0.067	0.050	0.040
perlbmk	***diffmail***						0.196						
	makerand			0.022	0.022		0.043	0.043	0.043				
	splitmail 535												
	splitmail 704			0.111	0.111				0.087	0.100	0.067	0.050	0.040
	splitmail 850					0.196							
	splitmail 957	0.156	0.156										
gap											0.067	0.050	0.040
vortex	***1***	0.067	0.067	0.067	0.067					0.100	0.067	0.050	0.040
	2							0.087	0.109				
	3												
bzip2	***graphic***					0.087	0.087	0.065	0.065	0.100	0.067	0.050	0.040
	program		0.111	0.133	0.111								
	source	0.244											
twolf				0.022	0.022				0.022	0.100	0.067	0.050	0.040
apsi					0.022				0.022			0.050	0.040



A. PCA, P&B, Random, and High-frequency

B. Int/Float, Core/Memory, and C/FORTRAN

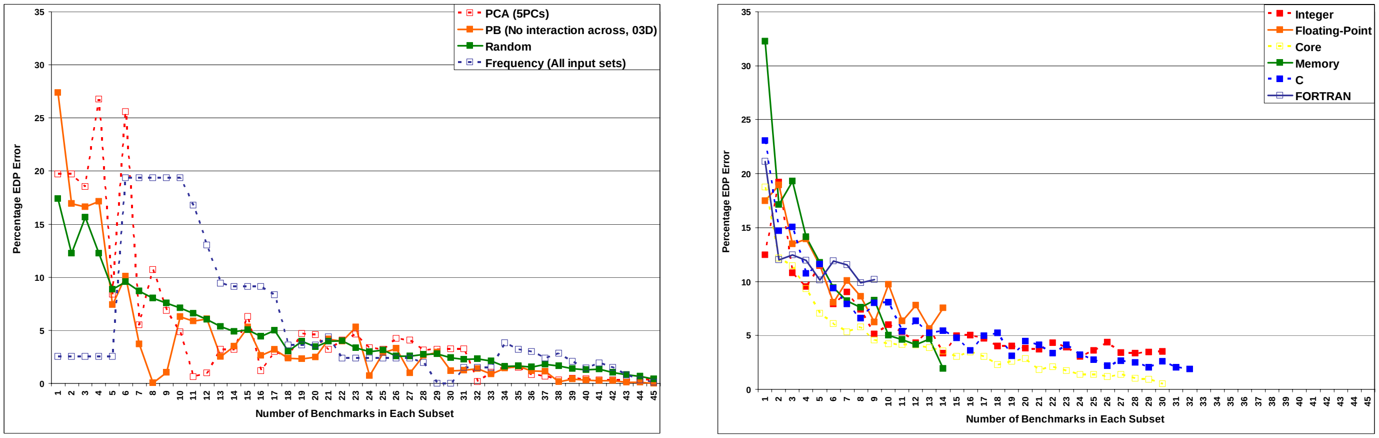
Figure 3. Relative CPI accuracy of subsetting approaches for a larger ROB/LSQ, configuration #1



A. PCA, P&B, Random, and High-frequency

B. Int/Float, Core/Memory, and C/FORTRAN

Figure 4. Relative CPI accuracy of subsetting approaches for next-line prefetching, configuration #1



A. PCA, P&B, Random, and High-frequency

B. Int/Float, Core/Memory, and C/FORTRAN

Figure 5. Relative CPI accuracy of subsetting approaches for larger and more associative caches, configuration #1

and input sets pairs with lower L1 D-cache and L2 hit rates. Furthermore, for some subset sizes, the branch prediction accuracy is also lower. For example, for configuration #1, the L1 D-cache hit rate for the entire suite is 91.1%, but the high-frequency L1 D-Cache hit rates are 89.7%, 88.2%, 88.3% and 88.9% for subset sizes of 10, 15, 20, and 25, respectively. For the same configuration, the overall L2 hit rate is 80.3%, but L1 D-cache hit rates for the high-frequency subsets are 77.5%, 74.7%, 75.2%, and 75.9%, respectively. By contrast, the average L1 D-cache and L2 cache hit rates for PCA and P&B subsets are much closer to the overall hit rates. Consequently, and unsurprisingly, the average CPI for the high-frequency subsets is significantly higher than their PCA and P&B counterparts.

In particular, the high-frequency subsets contain benchmark and input set pairs that have relatively low L1 D-cache hit rates such as: *gzip-program* (87.8%), *swim* (82.7%), *art-110* (59.8%), and *twolf* (88.9%). On the other hand, the benchmark and input sets pairs that lower the L2 cache hit rates of the high-frequency subsets include: *gcc-integrate* (77.4%), *swim* (65.3%), *vpr-route* (67.9%), *art-110* (32.8%), *bzip2-graphic* (59.9%), and *twolf* (76.6%).

Finally, since the PCA and P&B subsets in Table 3 have the best absolute accuracy, it is important to note that computer architects can use those subsets "out-of-the-box" for their own simulations.

4.2 Relative Accuracy

As described in the previous section, good absolute accuracy from a subsetting approach is important to processor designers and for researchers doing profiling and workload characterization research. However, for the computer architects that are researching new processor enhancements, relative accuracy, *e.g.*, speedup, decreased cache miss rates, power consumption reduction, *etc.*, is more important.

To quantify the relative accuracy of each the benchmark subsetting approaches, we quantified the speedup due to the following microarchitectural "enhancements": 1) A four-fold larger reorder buffer and load-store queue, 2) Next-line prefetching as described in [6], and 3) A L1 D-cache, L1 I-cache, and L2 cache with four times the capacity of the original cache and with 8-way associativity, without the corresponding increase in cache access latency.

To calculate the relative accuracy, we computed the speedup for each enhancement for all possible subsets. More specifically, we compute the CPI speedup for the benchmark and input pairs for that subset. Then to determine the relative accuracy, we compare calculate the speedup error, *i.e.*, the difference in speedups when subsetting is not used and when subsetting is used. The percentage difference between those speedups is the speedup error, and our measure of relative accuracy. In other words, we calculate the relative error as follows:

$$\text{Relative error} = (\text{Speedup}_{\text{with\_Subsetting}} - \text{Speedup}_{\text{without\_Subsetting}}) \,/\, \text{Speedup}_{\text{without\_Subsetting}} * 100$$

Figures 3, 4, and 5 show the relative accuracy for each of the three enhancements for configuration #1. For this configuration, the average (mean) speedup across all 46 benchmarks and input set pairs is 25.7% for the ROB/LSQ enhancement, 7.5% for next-line prefetching, and 56.5% for the larger and more highly-associative caches.

The results in Figures 3, 4, and 5 are generally similar to the results in Figures 1 and 2. Namely, PCA and P&B are the most accurate subsetting approaches and the remaining five approaches are less accurate. However, the key difference between these results and those in Figures 1 and 2 is that the relative error is much lower than the absolute error. For example, the results in Figures 3, 4, and 5 show that the relative error is less than 20% for most approaches and for most subset sizes. On the other hand, the reverse is true for absolute accuracy. With the exception of P&B and PCA, the absolute accuracy of the remaining five approaches is greater than 20% for most subset sizes. The key conclusion from these results is that most subsetting approaches are accurate enough to be used for comparative analysis, *e.g.*, speedup, for a much wider range of subset sizes.

The reason that the relative error is significantly lower than the absolute error is because there is less variation in the CPI between different architectures, *i.e.*, base configuration vs. enhancement, for a subset with the same benchmark and input set pairs – which is the case for relative accuracy – than in the CPI variability for the same architecture for different benchmark and input set pairs – which is the case for absolute accuracy. This is the same reason that allows computer architects to use matched-pair comparisons [5] to reduce simulation time.

Finally, the results for the other configurations and for EDP were very similar.

4.3 Subset Representativeness and Coverage

In the previous two subsections, we analyzed the absolute and relative accuracy of the subsetting approaches in terms of their CPI and EDP accuracy. However, the results in those two subsections do not measure the representativeness of the subsetting approaches across a wide range of metrics nor do they examine how well the benchmark and input set pairs of each subset are distributed across the entire space of metrics that the benchmark suite covers. Ideally, the benchmark and input set pairs in a subset have good absolute CPI and EDP accuracy while simultaneously covering the entire space of metrics, as opposed to being clustered around the average CPI, EDP, and value of each metric.

To quantify the representativeness and coverage of each subsetting approach, we:

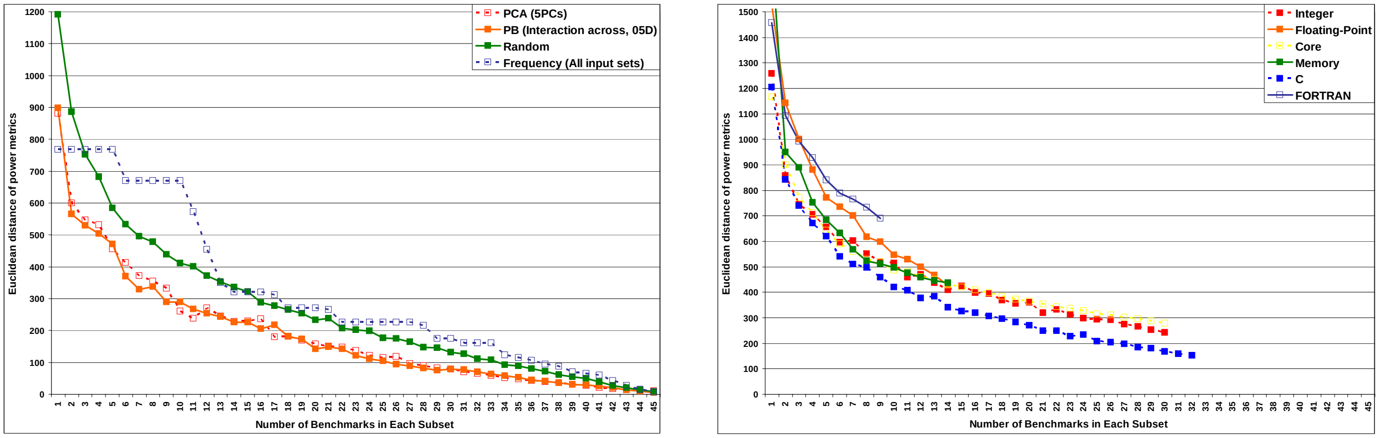
1. Vectorize the performance metrics and power metrics for each benchmark and input set pair. The performance metrics include: IPC; branch prediction accuracy; L1 D-cache, L1 I-cache, and L2 cache hit rates; and the D-TLB and I-TLB hit rates, while the power metrics include the power for the rename logic, branch predictor, reorder buffer, load-store queue, register file, L1 D-cache, L1 I-cache, L2 cache, functional units, result bus, and clock network.
2. Normalize the performance metrics to the maximum possible value of each (the maximum IPC was set to

the issue width for each processor configuration) and scaled to 100, while the power metrics normalized to their percentage of the total power consumption.

3. Compute the Euclidean distance between each benchmark input set *not* in the subset to each benchmark and input set pair in the subset.
4. Assign the minimum Euclidean distance as the distance for the benchmark and input set pair not in the subset.
5. Sum the Euclidean distances for all benchmark and input set pairs not in the subset and assign that number as the total minimum Euclidean distance for that subset size.

Intuitively, the total minimum Euclidean distance for each subset represents how well the benchmark and input set pairs in the subset are spread throughout the entire space of benchmark pairs. A smaller total minimum Euclidean distance means that benchmark and input set pairs that are not in the subset are very close to a benchmark and input set pair that is in subset. Then, from the viewpoint of representativeness, the benchmark and input set pairs not in the subset are accurately represented by a benchmark and input set pair in the subset, and from the viewpoint of coverage, the benchmark and input set pairs in the subset effectively cover the benchmark suite.

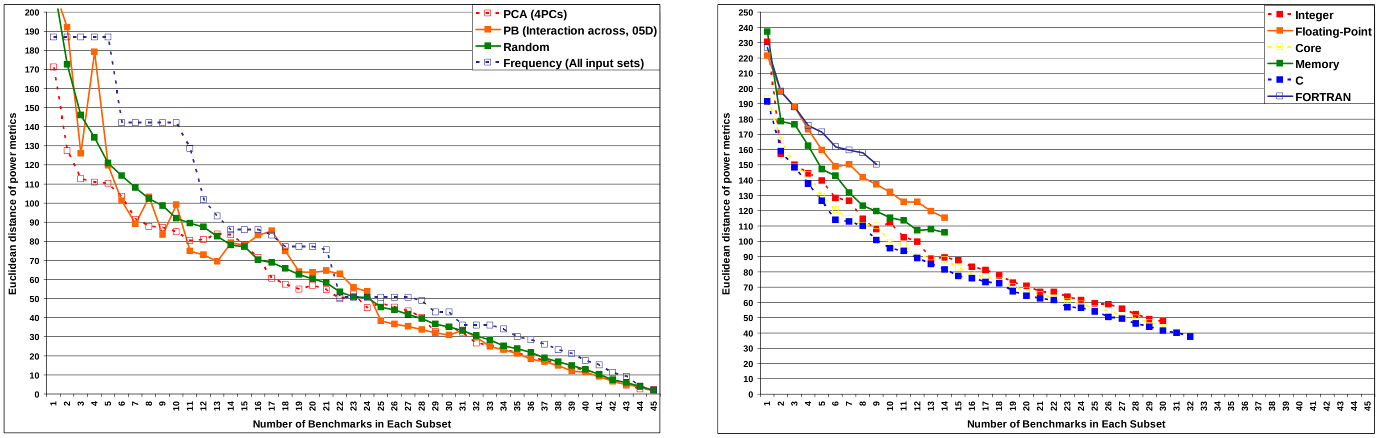
To understand the correlation between total minimum Euclidean distance and the difference between two vectors of performance metrics, consider the following example. Suppose that the Euclidean distance between two vectors is due solely to a single metric, *e.g.*, the L1 D-Cache hit rate, and that the minimum difference in that metric for a benchmark and input set pair not in the subset to a benchmark and input set pair in the subset is the same for all benchmark and input set pairs not in the subset. Assuming that there are 11 benchmark and input set pairs in the subset, *i.e.*, there are 35 benchmark and input set pairs not in the subset, and that the minimum difference between the L1 D-cache hit rates is 3%, since the L1 D-cache hit rate is the only metric that differs, the minimum Euclidean distance for any benchmark and input set pair not in the subset to any benchmark and input set pair in the subset is 3 (square root of 3^2), while the total minimum Euclidean distance for all benchmark and input set pairs not in the subset is 105 (3*35).



A. PCA, P&B, Random, and High-frequency

B. Int/Float, Core/Memory, and C/FORTRAN

Figure 6. Euclidean distance of performance metrics, configuration #5



A. PCA, P&B, Random, and High-frequency

B. Int/Float, Core/Memory, and C/FORTRAN

Figure 7. Euclidean distance of power metrics, configuration #5

If the difference between metrics is 5% and if there are only 6 benchmark and input set pairs in the subset (*i.e.*, 40 benchmark and input set pairs not in the subset), then the total minimum Euclidean distance is 200 (5*40). Finally, for a subset size of 31, if one metric differs by a minimum 3% and another by a minimum 4%, the minimum Euclidean distance is 5 (square root of $3^2 + 4^2$) and the total minimum Euclidean distance is 75 (5*15).

Figures 6 and 7 show the total minimum Euclidean distance for performance and power metrics, respectively, for configuration #5. The results in these two figures show that PCA and P&B consistently have the smallest total minimum Euclidean distances of all subsetting approaches. The difference in the Euclidean distances of the PCA or P&B approaches to the other approaches is typically over 100 for the performance metrics and over 15 for the power metrics for all subset sizes. There are two reasons why there is a smaller difference in the total minimum Euclidean distances for the power metrics as compared to the performance metrics. First, since the power metrics are a percentage of the total power, the maximum value for any single power metric is significantly less than 100, which is the maximum value for a performance metric. As a result, a smaller maximum means that there are smaller differences in the values of the various power metrics, which results in a smaller Euclidean distance. Second, due to static power consumption, there is less variability in the power results to begin with.

The conclusion from this subsection is that the PCA and P&B based subsetting approaches produce benchmark subsets that are significantly more representative than the subsets from the other five subsetting approaches. In addition, PCA and P&B are more effective at covering the entire space of the benchmark suite as compared to the other five subsetting approaches. This further explains and reinforces the conclusions reached in the previous subsections.

5 Discussion of Accuracy vs. Profiling Cost

The results in the previous section showed that the two statistically-based subsetting approaches, PCA and P&B, had the highest absolute and relative CPI and EDP accuracy, produced the most representative subsets, and most effectively covered the benchmark space. However, if the profiling data needed to generate these subsets is prohibitively high or sufficiently intrusive and/or if the time needed to process the profiling data and generate the subsets is too onerous, then computer architects may avoid using these techniques and opt for less complex, albeit less accurate, subsetting approaches. On the other hand, the time needed to generate subsets using any of the other five approaches is trivial. In the remainder of this section, we discuss the profiling cost, and the time/effort needed to generate the subsets.

The profiling cost for P&B is significant. Since the input parameters, *i.e.*, bottlenecks, need to be low and high values that are outside their normal range of values, the processor configurations that the P&B design uses represent configurations at the corners of the design space, whose performance can only be measured via simulation. The simulation time needed to gather the P&B profiling results is proportional to the number of input parameters – which in this case required several months – and the time to process the input parameters is only a few seconds.

Collecting the program characteristics for the PCA approach requires a specialized functional simulator or instrumentation to be run each benchmark and input pair. Since instrumentation using binary instrumentation tools such as ATOM or PIN is faster than detailed processor simulation, collecting the program characteristics for the PCA method is faster than collecting the data set for P&B – the data set for PCA can be gathered in a single instrumentation run whereas P&B requires multiple detailed processor simulation runs. After capturing the program characteristics, computing the principal components is done in the order of seconds.

6 Summary

For several reasons, including minimizing the simulation time or due to limitations in the simulation/measurement infrastructure, computer architects often use only a subset of the benchmarks in a benchmark suite. Despite the fact that the virtually all computer architects – based on the results in published papers – subset benchmark suites such as SPEC, the accuracy and characteristics of the most popular and/or proposed approaches is unknown. Using a subset that is not representative of the entire benchmark suite can, in the worst case, result in misleading conclusions.

To address this problem, in this paper, we evaluate the accuracy and representativeness/coverage for the most promising and popular benchmark subsetting approaches. In particular, we evaluate the following seven approaches: 1) Principal components analysis (PCA), 2) Performance bottlenecks using a Plackett and Burman (P&B) design, 3) Integer vs. floating-point, 4) Core-bound vs. memory-bound, 5) Language, 6) Random, and 7) By frequency of use, in terms of their absolute CPI and energy-delay product (EDP) accuracy, relative accuracy for three processor enhancements, and representativeness/coverage for 46 benchmark and input set pairs of the SPEC CPU 2000 benchmark suite. For the latter five approaches, we evaluate 30 random combinations for each subset size. For all approaches, we evaluate the entire range of possible subset sizes.

Our results show that the two statistically-based approaches, PCA and P&B, have the best absolute CPI and EDP accuracy. In particular, their CPI and EDP error drops below 5% for around 20 benchmark and input set pairs. In contrast, the CPI and EDP error for the other five subsetting approaches either never drops below 5% or only drops below 5% around 35 benchmark and input set pairs.

To help computer architects use representative subsets, we give the subsets for PCA and P&B, which are the most accurate and representative subsets, for several subsets sizes.

For the relative accuracy, we compare the speedup for each subset against the speedup for the entire suite for the following enhancements: 1) A larger reorder buffer and load-store queue, 2) Next-line prefetching, and 3) Larger L1 and L2 caches. The conclusions for relative accuracy are the same as the conclusions for absolute accuracy, namely, that PCA and P&B are the most accurate approaches. However, for all approaches and for corresponding subset sizes, the relative error is lower than the absolute error due to smaller variations in the CPI across architectures than across different subsets.

The representative/coverage analysis measures how even distributed the benchmarks of the subset are distributed across the entire space of benchmarks for several performance and power metrics. The conclusion from this analysis is the same as the conclusion from the absolute and relative accuracy analysis in that PCA and P&B are the most accurate subsetting approaches.

Finally, although PCA and P&B are the most accurate subsetting approaches, the key difference between the two is in their profiling cost. PCA uses microarchitectural independent metrics that can be gathered using binary instrumentation, *e.g.*, ATOM, while P&B relies on a simulator and needs to the results of several dozen simulation runs to gather the necessary profiling information. Therefore, given this result, PCA is the subsetting approach that has the best combination of accuracy versus profiling cost.

Acknowledgments

This research is supported in part by US National Science Foundation grants CCF-0541162 and 0429806, the University of Minnesota Digital Technology Center, the University of Minnesota Supercomputing Institute, the European HiPEAC network of excellence, IBM CAS Program, and Intel. Lieven Eeckhout is a Postdoctoral Fellow of the Fund for Scientific Research – Flanders (Belgium) (F.W.O Vlaanderen) and is also supported by Ghent University, IW, the HiPEAC Network of Excellence, and the European SARC project No. 27648.

References

[1] D. Citron, “MisSPECulation: Partial and Misleading Use of SPEC CPU2000 in Computer Architecture Conferences,” Panel Discussion in International Symposium on Computer Architecture 2003.

[2] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, “Workload Design: Selecting Representative Program-Input Pairs,” International Conference on Parallel Architectures and Compilation Techniques, 2002.

[3] L. Eeckhout, J. Sampson, and B. Calder, “Exploiting Program Microarchitecture Independent Characteristics and Phase Behavior for Reduced Benchmark Suite Simulation,” International Symposium on Workload Characterization, 2005.

[4] L. Eeckhout, R. Sundareswara, J. Yi, D. Lilja, and P. Schrater, “Accurate Statistical Approaches for Generating Representative Workload Compositions,” International Symposium on Workload Characterization, 2005.

[5] M. Ekman and P. Stenström, “Enhancing Multiprocessor Architecture Simulation Speed using Matched-Pair Comparison,” International Symposium on Performance Analysis of Systems and Software, 2005.

[6] N. Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-associative Cache and Prefetch Buffers,” International Symposium on Computer Architecture, 1990.

[7] A. KleinOsowski and D. Lilja, “MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research,” Vol. 1, June 2002.

[8] D. Montgomery, “Design and Analysis of Experiments”, Third Edition, Wiley 1991.

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

[10] R. Saavedra and A. Smith, “Analysis of benchmark characteristics and benchmark performance prediction,” ACM Transactions on Computer Systems, Vol. 14, No.4, pp. 344-384, 1996.

[11] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, “Automatically Characterizing Large Scale Program Behavior,” International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.

[12] http://www.eecs.umich.edu/~chriswea/benchmarks/SPEC2000.html.

[13] H. Vandierendonck and K. De Bosschere, “Many Benchmarks Stress the Same Bottlenecks,” Workshop on Computer Architecture Evaluation using Commerical Workloads, 2004.

[14] H. Vandierendonck and K. De Bosschere, “Experiments with Subsetting Benchmark Suites,” Workshop on Workload Characterization, 2004.

[15] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, “SMARTS: Accelerating Microarchitectural Simulation via Rigorous Statistical Sampling,” International Symposium on Computer Architecture, 2003.

[16] J. Yi, D. Lilja, and D. Hawkins, “A Statistically-Rigorous Approach for Improving Simulation Methodology,” International Symposium on High-Performance Computer Architecture, 2003.

[17] J. Yi. S. Kodakara, R. Sendag, D. Lilja, and D. Hawkins, “Characterizing and Comparing Prevailing Simulation Techniques,” International Symposium on High-Performance Computer Architecture, 2005.