BenchPrime: Accurate Benchmark Subsetting with Optimized Clustering Algorithm Selection*

Qingrui Liu [1] Xiaolong Wu[1] Larry Kittinger[1] Markus Levy[2] Changhee Jung [1]
Virginia Tech[1] EEMBC[2]
{lqingrui,xlwu,larryk85@vt.edu,chjung}@vt.edu markus.levy@eembc.org

ABSTRACT

This paper presents BenchPrime, an automated benchmark analysis toolset that is systematic and extensible to analyze the similarity and diversity of benchmark suites. BenchPrime takes multiple benchmark suites and their evaluation metrics as inputs and generates a hybrid benchmark suite comprising only essential applications. Unlike prior work, BenchPrime uses linear discriminant analysis rather than principal component analysis, as well as selects the best clustering algorithm and the optimized number of clusters in an automated and metric-tailored way, thereby achieving high accuracy. In addition, BenchPrime ranks the benchmark suites in terms of their application set diversity and estimates how unique each benchmark suite is compared to other suites.

As a case study, this work for the first time compares the DenBench with the MediaBench and MiBench using four different metrics to provide a multi-dimensional understanding of the benchmark suites. For each metric, BenchPrime measures to what degree DenBench applications are irreplaceable with those in MediaBench and MiBench. This provides means for identifying an essential subset from the three benchmark suites without compromising the application balance of the full set. The experimental results show that the necessity of including DenBench applications varies across the target metrics and that significant redundancy exists among the three benchmark suites.

1 INTRODUCTION

> *"We need not one but many application benchmarks, as diverse as possible, to rule out bias through some factors hidden in most of the benchmarks." — Jan Vitek in his EMSOFT paper [66] on experimental evaluation.*

To achieve an accurate and fair evaluation of any hardware and software techniques, it is essential to use representative benchmarks [20]. For researchers in both industry and academia, it has always been an important problem whether the proposed techniques in their respective research are indeed beneficial. As such, fair and accurate evaluation is of particular interest due to their impact on the determination of said beneficialness.

There are two conflicting requirements (i.e., diversity and irreplaceability) in building a representative benchmark set to conduct a thorough evaluation of any research proposal. For diversity, researchers often choose multiple benchmark suites based on their knowledge of the research and the benchmark characteristics. In essence, such a hybrid benchmark suite broadens the evaluation spectrum and no critical applications are missing in the resulting benchmark set.

On the one hand, the redundant applications in the benchmark set, that are pretty much the same in terms of their evaluation characteristics, can skew the average towards redundant characteristics overestimating the benefit of any proposed research [64]. This is mainly because one application is replaceable with other applications in the benchmark set. Thus, an ideal benchmark set should be comprised of only irreplaceable applications, and we refer to this property as **irreplaceability** of the benchmark set. On the other hand, the huge evaluation time for each application especially in microarchitecture simulation, causes researchers to use so-called benchmark subsetting for evaluating only a subset of the applications in a hybrid benchmark suite.

It is a challenge to construct a representative benchmark set in a way to avoid destroying the diversity but to achieve the irreplaceability meanwhile. Unfortunately, there does not exist to date a systematic methodology for effective and accurate benchmark analyses. Prior work tries to subset candidate benchmark applications based on some machine learning algorithms that can cluster the applications in different groups; one approach [67] resorts to K-means clustering [7] while another [22] to agglomerative clustering [3]. However, for a given user-specified classification metric (e.g., performance/power), all the previous approaches do not provide a basis for why one clustering algorithm is better than another; they either blindly apply one algorithm for every metric or require user's knowledge to figure out the best number of clusters [55].

In addition, the benchmark training of the prior work requires significant knowledge of machine learning and statistics which practitioners (e.g., programmers or performance engineers) may lack, making it difficult for them to carry out this daunting task in an error-free manner. This practice imposes an inherent limitation on the capability of the prior benchmark analysis frameworks. Therefore, there is a compelling need for a systematic, automated, and transparent benchmark analysis framework.

To fill that need, this paper presents BenchPrime, a framework used to generate a hybrid benchmark suite which is complete but with only a small amount of redundancy (i.e., high irreplaceability). It conducts different machine learning algorithms to select the best-performing clustering strategy for a given classification metric and can calculate the optimized number of clusters for the selected algorithm. BenchPrime customizes standard clustering methods, i.e., using Linear

*Correspondence to Changhee Jung (chjung@vt.edu). Qingrui Liu and Xiaolong Wu contributed equally to this work. The original article was presented in EMSOFT'17 [48]

Discriminant Analysis (LDA) for their input rather than Principle Component Analysis (PCA), to achieve high accuracy. BenchPrime also provides a set of post-analyses and visualization tools to readily process the raw data.

The vision of this work is to deploy BenchPrime as a cloud service in data centers and to maintain a common repository for keeping the analysis results of new benchmarks and the metrics up-to-date and open to the public. These results can then be used by many researchers including performance architects and benchmark creators. For example, users can build the representative benchmark set from all the benchmark suites that have been processed by BenchPrime, for a given classification metric.

In particular, the results of BenchPrime can be used for further analysis of the input benchmark suites. That is, the diversity analysis results can be used to rank each benchmark suite, which would be useful information for those whose budget can afford to buy only one benchmark suite. Likewise, the irreplaceability analysis results can be used to estimate how unique each benchmark suite is compared with other suites. This serves as a basis for judging whether researchers need to integrate a new benchmark suite for the evaluation of any research proposal, e.g., one only needs to purchase those applications that are irreplaceable with existing benchmarks he or she currently owns.

As a case study, this work leverages BenchPrime to compare MediaBench and MiBench to DenBench created by EEMBC. In the embedded systems community, researchers often leverage MediaBench [36] or MiBench [16] to evaluate their research results. Some researchers use both benchmarks while others [8, 31, 39] mix MediaBench with a few applications from MiBench. However, contrary to the two popular benchmark suites, DenBench has received relatively less attention [58]. That is supported by Figure 1 which shows the number of papers published in major embedded system conferences from 2002 to 2016, leveraging each benchmark in the experiments. With the imbalance between benchmark popularities, the lack of existing work on analyzing DenBench compared with the others raises important questions to the embedded systems community. For example, one might be afraid that different (similar) characteristics of DenBench compared with MediaBench and MiBench will cause incomplete (skewed) evaluation.

To analyze the three benchmark suites, BenchPrime intendedly uses multiple different metrics. That is because both the diversity and irreplaceability of benchmark suites vary according to different metrics. In addition to traditional architectural performance numbers (e.g., CPI), this work takes a step forward by adding measurements of their power consumptions and architectural vulnerability factors (AVFs) due to the ever-increasing concerns about energy efficiency [5, 10, 11, 33, 42, 43, 59, 61, 65] and soft error resilience [12–14, 21, 34, 44–47, 50, 51], respectively.

For each metric being used to characterize the benchmark suites, BenchPrime selects only the essential subset of the applications from MediaBench, MiBench, and DenBench, without compromising the *application balance* in the full set.

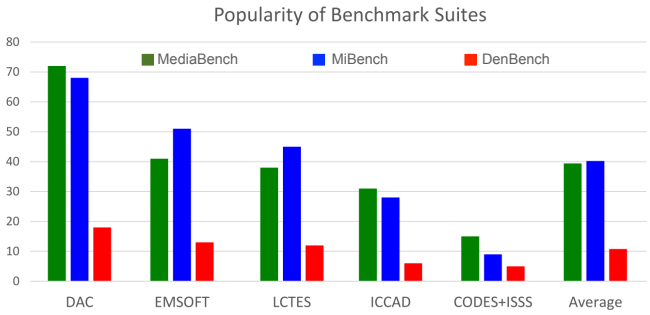


Figure 1: Popularities of three embedded benchmark suites: each bar represents the number of publications that evaluate each benchmark suite

Moreover, this work estimates to what degree DenBench benchmarks are irreplaceable with those in MediaBench and MiBench. The experimental results show that the necessity of including DenBench benchmarks varies across the target metrics, and that significant redundancy exists between these three suites, with which a small subset of carefully chosen benchmarks can effectively cover the evaluation space of the original benchmarks. Overall, this paper makes the following contributions:

- To the best of our knowledge, BenchPrime is the first systematic/automated benchmark analysis framework that can select the best clustering algorithm and the optimized number of clusters.
- The resulting hybrid benchmark of BenchPrime is a lot more accurate (i.e., close to oracle suite) than any unoptimized clustering approaches. The evaluation results demonstrate that BenchPrime always selects the optimized clustering configuration for 4 different benchmark evaluation metric.
- BenchPrime ranks input benchmark suites in terms of their application set diversity and irreplaceability, which can serve as a basis for choosing one benchmark suite over another.
- This work is the first effort in directly comparing MediaBench, MiBench, and DenBench, which is meaningful for the embedded systems community.

2 MOTIVATION

The optimized clustering algorithm selection of BenchPrime is inspired by the fact that there is no one-size-fit-all clustering algorithm. The result of the benchmark subsetting derived by unoptimized clustering would not effectively represent the full set. In fact, the best clustering algorithm varies depending on not only the input benchmark suites but also their classification metric. Figure 2 supports this for the AVF (Architecture Vulnerability Factor) metric; the accuracy of each clustering algorithm is measured by calculating Euclidean distance between the average AVF of the resulting subsetted benchmarks and that of the oracle, i.e., a full set. Overall, each clustering algorithm shows significantly different Euclidean distances. The best algorithm, *pam*, has only 30% of Euclidean distance

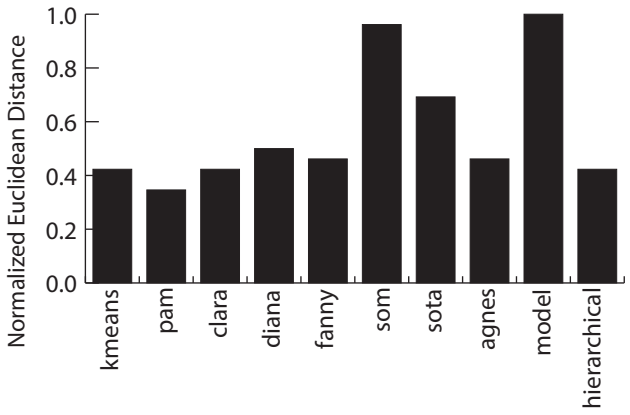


Figure 2: Accuracy of different clustering algorithms for AVF measured by Euclidean distance from the oracle: the lower, the better

achieved by *model*-based clustering algorithm which is the worst. In light of this, BenchPrime takes into account the optimized clustering algorithm selection in its design.

3 BENCHPRIME OVERVIEW

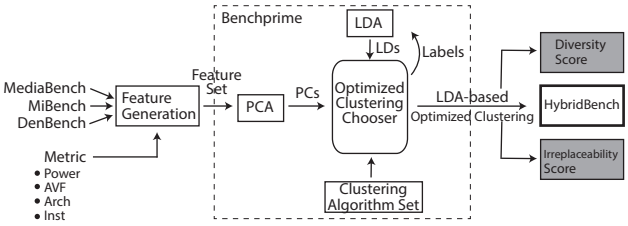


Figure 3: BenchPrime framework

The goal of BenchPrime is providing its users with a systematic and automated benchmark analysis toolset. Figure 3 shows a high-level workflow of the proposed BenchPrime framework and its toolset. For a user-specified classification metric (e.g., Power, AVF), BenchPrime analyzes the input benchmark suites and generates the hybrid benchmark suite along with their characterization results (i.e., diversity and irreplaceability).

BenchPrime takes a set of input features that summarize the characteristics of given benchmark suites based on the metric used to evaluate the benchmark applications. In general, users generate the input features by running compiled executables for a given metric on top of one or multiple architecture platforms. All the feature data are first processed by Principal Component Analysis (PCA) to lower their dimensions with the principle components. The optimized clustering algorithm chooser first generates the initial clusters taking the principle components as inputs. BenchPrime uses the initial cluster information to make labels for each application; they are required to generate the labeled feature data for Linear Discriminant Analysis (LDA). Next, BenchPrime conducts LDA on the labeled feature data, and the resulting LDs (Linear Discriminants) are fed back into the clustering algorithm chooser to identify the best one among 10 candidates from a clustering algorithm set (bottom of Figure 3) as well as the best fitting number of clusters.

This is achieved by evaluating Bayesian Information Criterion (BIC) for any possible clustering configuration with each candidate algorithm and its cluster number. BenchPrime then selects the configuration with the highest BIC value. Finally, by picking a representative benchmark from each cluster of the configuration, BenchPrime can generate the resulting hybrid suite comprised of only essential applications from the input benchmark suites.

In addition, the results of BenchPrime can be used for further analysis of the input benchmark suites. That is, the diversity analysis results can be used to rank the benchmark suites based on the application set diversity while the irreplaceability analysis results to estimate how unique each benchmark suite is compared with other suites. In addition, the characterization results of BenchPrime can enable further analysis of the input benchmark suites to rank them for quantitative comparison, e.g., the diversity and irreplaceability analyses, as shown Section 5.

4 CHOOSING OPTIMIZED CLUSTERING

This section first shows how to effectively process feature data being used as inputs to clustering algorithms without loss of generality and then details the mixed two-phase optimized clustering algorithm selection of BenchPrime.

4.1 Data processing

Rather than taking the absolute numbers generated from feature generation, this work pursues the relative results between these benchmarks. However, it is hard to tell such relations using the raw measurement data. On the one hand, the number of dimensions in the result is too high to present in a human-digestible way. For example, the output of the AVF analysis consists of six dimensions since there are six microarchitectural components being evaluated, thus it is impossible to synthesize a meaningful conclusion by directly inspecting these six aspects. On the other hand, measurements on each of these multiple dimensions are likely to interact with each other in a complex manner, which prevents us from getting precise insight from the data. To overcome such difficulties and provide means for grounded comparative analysis, BenchPrime leverages the following statistical tools.

4.1.1 Principle Component Analysis (PCA). Principle Component Analysis (PCA) aims at reducing the dimensionality of input features and eliminating the correlations between the features. It transforms the original (possibly) correlated multi-dimensional feature data into a new, orthogonal space, in which each new dimension, so-called the principal component, is uncorrelated to each other. Moreover, these principal

components are ordered by the data's covariance, which indirectly reflects the "importance" of these dimensions in describing the features of the data. With this transformed result, one can realize dimension reduction without losing much important information by keeping only the most significant principal components while dropping the rest.

BenchPrime uses the result of PCA, which already eliminates correlation between different dimensions, as the input for further clustering analysis. Here, this work applies Kaiser's rule [52] to retain the PCs with eigenvalues that represent the main feature of these benchmark programs; if they are larger or equal to one, we feed them to BenchPrime's optimized clustering algorithm chooser. To this end, the PCA process reduces the original raw data to lower dimensional data containing main features.

4.1.2 Linear Discriminant Analysis (LDA). Even though PCA is a popular dimension reduction algorithm, it does have some inherent limitations. It is not hard to construct pathological examples where PCA is the worst possible thing in the data analytics. More precisely, PCA is not optimized for some classification problems because it lacks a notion of the class label in the definition of PCA; here keeping the dimensions of largest variance is a good idea, but not always leads to the right decision for the best clustering results [4].

In contrast, LDA (Linear Discriminant Analysis) is most commonly used as a dimension reduction technique in the preprocessing step for pattern-classification and matching/learning applications, thus it is a better fit to the benchmark clustering problem BenchPrime pursues. The goal is to project a dataset onto a lower-dimensional space with good class-separability. The general LDA approach is very similar to a Principal Component Analysis. However, in addition to finding the component that maximizes the variance of data, LDA is particularly interested in the axes that maximize the separation between multiple classes. According to the evaluation results, LDA turns out to be more efficient (higher BIC score) in finding the best clustering configuration, thus the resulting subset of input benchmark suites is much closer to the full set (oracle).

4.2 Two-phase approach for optimized clustering algorithm selection

BenchPrime uses BIC (Bayesian Information Criterion) as the criterion for the evaluation of each possible clustering configuration comprising a candidate algorithm and its cluster number to select the configuration with the highest BIC. BenchPrime takes a two-phase approach to choose the optimized algorithm and the best cluster number leveraging both PCA and LDA processes.

With principle components (PCs) from PCA as inputs, the first phase uses the optimized clustering strategy selection algorithm shown in Section 4.2.3 to get the best PC-based clusters. Based on this cluster information, BenchPrime adds a label to the original feature data for a later LDA process. For example, the benchmark program corresponding to the first cluster will be labeled as "1" , the program corresponding to the second cluster will be set as "2", and so on.

In the second phase, BenchPrime processes the labeled feature data and obtains the generated linear discriminants (LDs) that are fed as inputs to the optimized clustering algorithm chooser again (See Figure 3). To this end, BenchPrime selects the final LD-based optimized clustering method and the best number of clusters to maximize the accuracy of the resulting subsetted benchmarks compared to the full set.

4.2.1 Clustering Algorithm Candidates. In order to find the best clustering algorithm, BenchPrime considers ten most popular clustering algorithms [9]. These algorithms represent a wide range of clustering methods: "Agnes", "Hierarchical", "Diana", "K-means", "Pam", "Clara", "Fanny", "Model-based", "Som", and "Sota". Diana is a hierarchical clustering algorithm that is divisive rather than agglomerative. Thus, unlike agglomerative hierarchical, all the observations start from one large cluster and in each step, the largest cluster is first identified. Pam (Partitioning around medoids) is a clustering algorithm that works in a very similar way to K-means clustering algorithm; the difference is that the centers are medoids instead of centroids. Clara is an extension of Pam and is a time-efficient algorithm for clustering large datasets. Fanny is a fuzzy clustering algorithm that uses probability to determine the cluster allocation. Rather than having each observation defined to a single cluster, Fanny allows it to have some degree of association with each other. The Model-based algorithm assumes that the input comes from a finite mixture of normal distribution. Som is a clustering algorithm based on biological neural networks. Lastly, Sota is a divisive clustering algorithm that has properties similar to both hierarchical and Som clustering techniques.

4.2.2 Bayesian Information Criterion. While many criteria are used to determine the optimized number of clusters, i.e., the K value, the most popular one is Bayesian Information Criterion (BIC). It is a measure of the "goodness of fit" of any clustering results. The larger a BIC score, the higher probability that the clustering is a good fit to the original data. To figure out which pair of clustering algorithm and the K yields the highest BIC score. This work uses Equation 1 and 2 from [22, 60] for BIC calculation. The BIC contains two parts: the likelihood and the penalty. The likelihood is a measure of how well the clustering models the data.

$$BIC(D, K) = l(D|K) - \frac{p_j}{2} log(R) \quad (1)$$

D is the data set to be clustered, i.e., the output of PCA or LDA; $l(D|K)$ is the likelihood, i.e., the measure of how well the clustering models the data; R is the number of applications to be clustered; p_j is the sum of $(K-1)$ cluster probabilities, i.e., $K + dK$ where d is the dimension of each benchmark, i.e., the number of chosen PCs or LDs; and R is the number of points in the data, i.e., the number of benchmarks to be clustered. To compute $l(D|K)$, this work

uses Equation 2.

$$l(D|K) = \sum_{i=1}^{K} \left(-\frac{R_i}{2} log(2\pi) - \frac{R_i \times d}{2} log(\sigma^2) - \frac{R_i - K}{2} + R_i log R_i - R_i log R \right) \quad (2)$$

where R_i is the number of points in the i^{th} cluster, and σ^2 is the average variance of the Euclidean distance from each point to its cluster center.

4.2.3 Optimized clustering algorithm selection. By comparing the BIC scores of any possible clustering configuration comprising each candidate algorithm and its cluster number, BenchPrime can select the best-performing configuration. Algorithm 1 details how to perform this process taking a PCA or LDA generated data set D for the input benchmark suites and their classification metric. BenchPrime first loops each possible cluster number k to find the local optimized cluster number with the highest BIC score. Here, the maximum cluster number is the sum of the cardinalities of the input benchmark suites, e.g., for a case study in Section 6, since there is a total of 54 applications from DenBench, MediaBench, and MiBench, the loop iterates until the cluster number becomes 54. Similarly, it loops all clustering algorithms to identify the global optimized cluster number, k_{opt}, with the overall highest BIC score. For example, k_{opt} is 20 for a given microarchitecture dependent metric while it is 26 for an instruction mix metric. We show how BenchPrime obtains hybrid benchmark suites with the optimized clustering configuration in Section 6.4.

For constructing the hybrid benchmark suite with high diversity and low redundancy, it is important to pick a representative application from each cluster. Once the optimized clustering configuration is obtained, BenchPrime selects the closest-to-centroid point for each cluster to pick the most representative application of each cluster. For the centroid-directed selection, BenchPrime calculates the mean value for every cluster in the optimized clustering configuration, and then obtains the distance from the mean point for each application belonging to the cluster. BenchPrime selects the benchmark application, whose distance is smallest, as the centroid application. Equation 3 calculates the centroid point for a given k^{th} cluster.

$$c_k = argmin_{x_i \in k^{th}\ cluster} (x_i - \mu_i), \quad (3)$$

where μ_i is the center point, i.e., the mean value of the k^{th} cluster which x_i is assigned to; we use Euclidean distance to calculate the distance between points.

5 COMPARISON OF BENCHMARK SUITES

In addition to the optimized clustering configuration selection, BenchPrime calculates the diversity and redundancy scores of input benchmark suites which can be used to rank them for quantitative comparison.

Algorithm 1 Determine the optimized number of clusters with clustering algorithm.

1: **Input**: D, PCA or LDA generated data set; C, a set of clustering algorithms.
2: **Output**: k_{opt}, the optimized number of clusters; C_{opt}, the corresponding clustering algorithm; $score_{max}$, the overall maximum score.
// Store maximum BIC scores and corresponding number of clusters for all clustering algorithms.
3: Create an $\|C\|$-sized array $S_p = \{(score, k)\}$
4: K: the number of D rows;
5: **for** $i \leftarrow 1, \ldots, \|C\|$ **do**
// Store BIC score for each k-clustering.
6: Create a K-sized temporary array S
7: **for** $k \leftarrow 1, \ldots, K$ **do**
8: $S[k] = C_i(D, k)$; ▷ Use BIC to measure the score
9: **end for**
10: $S_p[i].score = max_{1,\ldots,K}\{S\}$
11: $S_p[i].k = argmax_{1,\ldots,K}\{S\}$
12: **end for**
13: $i_{max} = argmax_{1,\ldots,\|C\|}\{S_p.score\}$
14: $score_{max} = S_p[i_{max}].score$
15: $k_{opt} = S_p[i_{max}].k$
16: $C_{opt} = C[i_{max}]$
17: **Return** $k_{opt}, C_{opt}, score_{max}$.

5.1 Diversity measurement

For a quantitative comparison of each benchmark suite separately, BenchPrime can deduce how many applications have different characteristics within each benchmark suite. For a given classification metric, the applications with the same characteristics form one cluster. Note, the resulting cluster number varies depending on the characteristic distribution. In other words, the more clusters there are, the more different characteristics the benchmark suite has, meaning that it owns more diverse characteristic distribution, i.e., higher diversity. Similarly, fewer clusters represent lower diversity.

Such information provides a basis for picking one benchmark suite over another, e.g., users would prefer one with high diversity. To this end, this work defines the diversity score of each benchmark suite for a given metric as the ratio of the number of clusters grouped by the metric to the number of all applications in the suite.

$$s_D = \frac{The\ number\ of\ clusters}{The\ number\ of\ benchmark\ applications} \quad (4)$$

Thus, the larger the s_D is, the more diverse the benchmark suite is [1]. BenchPrime calculates the s_D of a given benchmark suite following the same steps described in Figure 3 without taking other suites into account. In contrast to the diversity that quantifies a single benchmark suite, the irreplaceability

[1] If a given benchmark suite has only one application, the resulting s_D value is one. This is a rare but misleading delicacy. One way to fix this is to redefine s_D using the number of applications in a benchmark suite as some weight coefficient. However, all of the benchmark suites we evaluated have tens of applications, making the Equation 4 sufficient.

Algorithm 2 Determine the k_{irr} for a given benchmark suite compared to BenchPrime's optimized hybrid suite.

Input: D, feature data; P, a set of all applications for all the input benchmark suites; P_{irr}, a program set from a particular single benchmark suite; C_{opt}, the optimized clustering algorithm from Algorithm 1.
Output: k_{irr}, the minimum number of clusters where at least one program from this benchmark suit must be included.
// Mark each program's cluster number
Create a $\|P\|$-sized array F.
for $k \leftarrow 2, \ldots, \|P\|$ **do**
    $F = C_{opt}(D, k)$; ▷ Use the optimized clustering algorithm with k clusters on the feature data D.
    for $l \leftarrow 1, \ldots, k$ **do**
        if $\|P_{F=l} \cap P_{irr}\| = \|P_{F=l}\|$ **then**
            Return k.
        end if
    end for
end for

of a benchmark suite (Section 5.2) is determined according to the contribution of the suite towards BenchPrime's hybrid benchmark suite taking into account other benchmark suites.

5.2 Irreplaceability measurement

For comparing multiple benchmark suites comprehensively, it is important to figure out whether one suite can be replaced with another. When constructing the hybrid benchmark suite, such information allows users to understand how important one benchmark suite is compared to other suites. For example, if the existing suites owned by the users can cover a new benchmark suite being released, they do not have to purchase it; otherwise, it is rather important and required for building the target hybrid benchmark suite.

To measure such an importance of a given benchmark suite, this work searches for the cluster configuration for which the suite is required, i.e., finding the cluster comprised of only the applications belonging to the suite; omitting them ends up losing the cluster, thus it might not be possible to cover the full set due to the lost cluster. Once such a clustering configuration is found, this work compares it to BenchPrime's optimized hybrid benchmark suite. Basically, their difference can determine the degree of the importance (i.e., irreplaceability score) of the given benchmark suite.

In light of this, this work defines an irreplaceability score of a given benchmark suite by taking the difference between k_{opt} of BenchPrime's optimized hybrid suite and k_{irr}, i.e., the minimum k whose resulting hybrid suite includes at least one cluster comprising only the programs of the given suite. This work refers to k_{irr} as an irreplaceable cluster number.

$$s_{irr} = \frac{k_{opt} - k_{irr}}{k_{opt}}. \quad (5)$$

The higher irreplaceability score, s_{irr}, means that the benchmark suite plays more important role in covering the given metric to classify the input benchmark suites. Algorithm 2 presents a method to obtain the k_{irr} number.

Type	Metric	Data being measured
Microarchitecture-independent	Instruction mix (percentage of committed instructions)	int
		float
		SIMD
		memory read
		memory write
		others
	Number of instructions	total number
Microarchitecture-dependent	Cache behavior	iCache MPKI
		dCache MPKI
		L2 inst MPKI
		L2 data MPKI
	Control-flow behavior	branch predict MPKI
		BTB hit percentage
	CPI	instructions per cycle

Table 1: Traditional Architectural Characteristics

6 EVALUATION: A CASE STUDY

As a case study, this work leverages BenchPrime to analyze 3 embedded benchmark suites, i.e., DenBench, MediaBench, and MiBench. This work is the first effort on comparing the DenBench, which has received little interest from embedded system communities (See Figure 1), to MediaBench and MiBench. This comparison involves four different metrics to provide a multi-dimensional understanding of the benchmark suites. We believe that the findings this section delivers will be useful to the embedded systems community.

6.1 Metrics

On top of traditional architectural performance numbers (such as CPI and cache behavior), this study takes a step forward by adding the measurements of the power consumptions and architectural vulnerability factors (AVFs) due to the ever-increasing concerns about energy efficiency [5, 10, 33, 42, 59, 61, 65] and soft error resilience [21, 44, 50, 51, 62], respectively.

6.1.1 Traditional Architectural Characteristics. Various architectural characteristics have been proposed for analyzing benchmark diversity and subsetting benchmark applications. They are often divided into two categories, namely microarchitecture dependent and microarchitecture independent characteristics. This work adopts an essential part of these commonly used characteristics and they are listed in Table 1. In addition, Figure 4 shows the dynamic instruction count, i.e., the number of instructions commited, for all the benchmarks from 3 embedded benchmark suites. Overall, DenBench has a significantly larger dynamic instruction count than the other two, while MediaBench has the smallest number of instructions commmited.

For cache and branch prediction related metrics, we use *misses per kilo instructions* (MPKI) to get precise insight for the architectural behaviors. Branch Target Buffer (BTB) hit percentage is also included to facilitate better understanding of control-flow related characteristics. Instruction mix is based on the percentage of committed instructions from different function units (FU), namely float, integer, SIMD

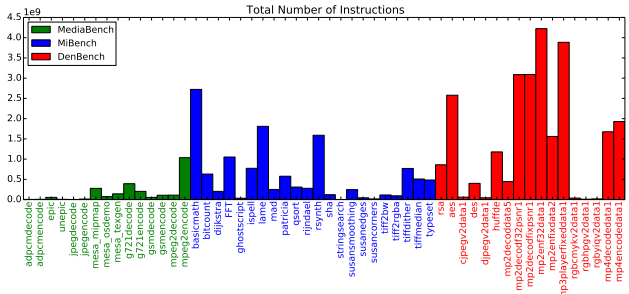


Figure 4: Total Number of Instructions

and memory read/write. With these representative and traditional measurements, we intend to lay the ground work for more unique perspectives.

6.1.2 Architectural Vulnerability Factor. Architectural Vulnerability Factor (AVF) analysis evaluates the reliability of a computer system. It is defined as the probability that a fault in a microarchitecture will result in a visible error [51, 63]. This paper applies the AVF analysis to the execution of the target benchmarks, aiming to evaluate their (di)similarity in exposing soft-error related characteristics of the underlying system.

AVF analysis is built upon the observation that not all faults in a microarchitecture will produce an actual error in the final output. For instance, a single bit flip in the branch predictor may affect the performance due to misprediction, but it will not alter the result of any committed instructions. These bits are resilient to a single-event upset (SEU) due to a particle strike, while the other kinds of processor state bits are architecturally correct execution (ACE) bits which are required for architecturally correct execution. Following [51, 63], the AVF of a hardware structure is calculated with Equation 6:

$$AVF_{hw} = \frac{\sum ACE\ bit\_cycle}{total\ \#\ of\ bits\ in\ a\ structure\ \times\ total\ cycles} \quad (6)$$

where ACE bit_cycle is the live cycles of a ACE bit. The live cycles of a register ACE bit is the number of cycles between the definition and subsequent use of the register. In short, this experiment traces the benchmark execution and calculates the AVF results on various microarchitectural components. Different from the above performance-driven architectural characteristics, AVF analysis measures the diversity and similarity of the benchmarks in evaluating the system reliability. We evaluate instruction queue (IQ), load/store queue (LQ/SQ), integer/float register file (RF) and re-order buffer (ROB) for AVF analysis.

6.1.3 Power Factor. With the rising concerns about energy efficiency for embedded systems [42, 59, 61, 65], this paper adds the power consumption as another dimension of the benchmarks' characteristics. For this group of metrics, the experiment measures the power consumption on the same architectural components as AVF analysis. We leverage HP Labs' *McPAT* [41] to generate the power estimation of the architectural components by feeding Gem5's simulation [6] statistics (e.g. execution cycles and cache sizes etc.) to McPAT.

6.1.4 Limitation and Discussion. Although in this paper we only consider the above metrics as a case study, BenchPrime is not limited to these metrics. For example, if the users would like to select a subset of benchmarks for performance evaluation, they can feed BenchPrime with some static and dynamic metrics like code size and memory usage to construct a smaller set of benchmarks without loss of coverage for the whole benchmark set.

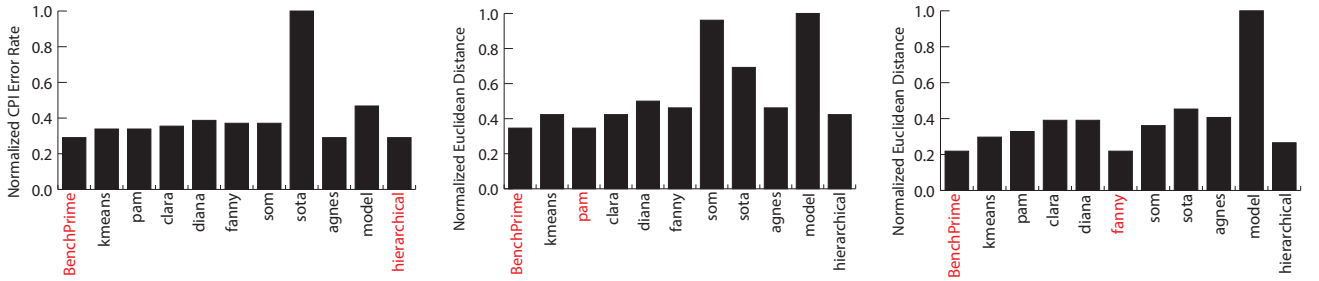
6.2 Experimental Setup

We leverage LLVM [35] to compile our the applications with default "-O3" optimization level. We conduct our simulations on top of the Gem5 simulator [6] with the ARMv7 ISA, modeling a modern two-issue out-of-order 2 GHz processor with L1-I/D (32KB, 2-way, 2-cycle latency, LRU), and L2 (2MB, 8-way, 20-cycle latency, LRU) caches. The pipeline width is two; the ROB has 40 entries; the integer register file has 128 entries; the float register file has 192 entries; and the instruction queue, load queue, and store queue have 32, 16, and 16 entries, respectively. Modifications to Gem5 were implemented to obtain our AVF statistics from the simulated microarchitectural structures. To obtain the power consumption for the previous mentioned architectural structures, HP Labs' *McPAT* [41] was utilized as discussed in Section 6.1.3. BenchPrime takes as input the configuration script and generated statistical files produced by Gem5 and constructs a valid McPAT input script.

6.3 Accuracy Analysis

BenchPrime's hybrid suites subsetted from input benchmark suites must reflect the representative behavior of the original full set, lest any experiments to be conducted using the subsetted benchmarks are misleading. We evaluate BenchPrime's accuracy by measuring the difference of two metrics, i.e., CPI and Euclidean distance of the subsetted hybrid benchmark suite from the oracle suite based on the full set. Figure 5 (a) highlights BenchPrime's optimized clustering configuration from other clustering algorithms in terms of CPI accuracy; it is obtained by calculating the mean value error rate of CPI of a given subsetted benchmark suite compared to that of oracle suite. BenchPrime successfully selects a hierarchical clustering algorithm, which has the lowest CPI error rate, as the optimized clustering algorithm. Here, we also use BenchPrime to select the best number of clusters for other 9 algorithms; without the help of BenchPrime, they show lower accuracy, thus their reported accuracy in Figure 5 is overestimated.

Figure 5 (b) and (c) present the accuracy of BenchPrime for AVF and Power separately by calculating the Euclidean distance difference between a given subsetted benchmark suite and the oracle suite. Due to page limitation, we do not show the graph of architectural characteristics where BenchPrime



(a) CPI accuracy of BenchPrime (b) AVF accuracy of BenchPrime (c) Power accuracy of BenchPrime

Figure 5: BenchPrime accuracy for CPI, AVF, and Power measured by Euclidean distance from the oracle: the lower, the better

Target Metric	Optimized Algorithm	BenchPrime's Hybrid Benchmark Suite of MideaBench, MiBench, and DenBench
uArch	Hierachical	tiff2rgba, mp4decodedata1, patricia, typeset, ispell, rgbyiqv2data1, g721encode, mp2enfixdata2, mpeg2decode, mp4encodedata1, qsort, sha, tiffmedian, mp2decoddata5, unepic, stringsearch, huffde, rgbhpgv2data1, ghostscript, mesa_mipmap
Inst	Hierachical	rsynth, epic, lame, jpegencode, gsmencode, mesa_mipmap, mesa_osdemo, gsmdecode, adpcmdecode, rgbhpgv2data1, susansmoothing, basicmath, mpeg2encode, mp2enfixdata2, unepic, mp2enf32data1, mp2decodfixpsnr1, tiff2bw, mp3playerfixeddata1, mp2decoddata5, rijndael, tiff2rgba, susanedges, susancorners, mp4decodedata1, typeset
AVF	Pam	mesa_mipmap, tiff2bw, mp2enf32data1, lame, tiffmedian, djpegv2data1, rgbcmykv2data1, rgbyiqv2data1, rgbhpgv2data1, tiffdither, adpcmencode, g721encode, FFT, tiff2rgba, mp4decodedata1, gsmdecode, susanedges, susancorners, des, adpcmdecode
Power	Fanny	rgbyiqv2data1, gsmencode, sha, basicmath, mesa_texgen, huffde, mpeg2decode, rsa, mp4decodedata1, typeset, rsynth, g721decode, epic, mp2decodfixpsnr1, des, gsmdecode, ghostscript, mesa_mipmap, mad, patricia, rijndael, susancorners, tiffmedian, lame, mp2enf32data1, tiff2bw

Table 2: Hybrid benchmark for the four metrics.

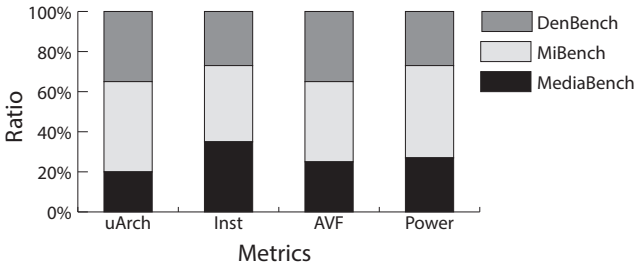


Figure 6: The contribution of each benchmark suite.

performs the best. All the results show that BenchPrime always selects the optimized clustering algorithm with the smallest Euclidean distance from the oracle suite (full set). In contrast, using any of other 9 clustering algorithms leads to a lot higher Euclidean distance, especially for model based clustering algorithm; it has the worst Euclidean distance on average, thus failing to reflect the original AVF/power behaviors of the oracle suite. Again, we use BenchPrime to select the best number of clusters for the 9 other algorithms, overestimating their accuracy. All the experiments illustrate the efficacy of the BenchPrime approach.

6.4 Hybrid Benchmark

Table 2 shows the hybrid benchmark chosen by BenchPrime and their corresponding optimized clustering algorithm for the four metrics, i.e., Fanny clustering for power, Pam clustering for AVF, hierarchical clustering for both microarchitecture-dependent and instruction mix metrics. We use different colors to represent programs belonging to each benchmark suite; green for MediaBench, blue for MiBench, and red for DenBench. From Table 2, each generated hybrid benchmark suite consists of programs from all the three suites. Across different metrics, the hybrid benchmark suite changes both its size and the program distribution of each benchmark suite.

Figure 6 shows the contribution of each benchmark suite to the hybrid benchmark for all metrics; the contributions of all three benchmark suites sum to 100%. In the hybrid benchmark suite, MiBench's contribution is 38%-46% while DenBench's is 27%-35%. Since MiBench size is the largest among the three benchmark suites, i.e., 22 programs, its high contribution is not surprising. On the other hand, it is interesting to observe that DenBench consists of only 17 programs but contributes better than MediaBench. Among the three suites, MediaBench's contribution is the least, i.e., 20%-35%. Note, the hybrid benchmark suite is just a recommendation based on the centroid strategy (Equation 3) to pick the representative application for each cluster, so some applications in the suite might not be irreplaceable.

Observation 1: *The contribution of each input benchmark suite to the* BenchPrime*'s hybrid benchmark suite is not directly proportional to the size of the input benchmark suite.*

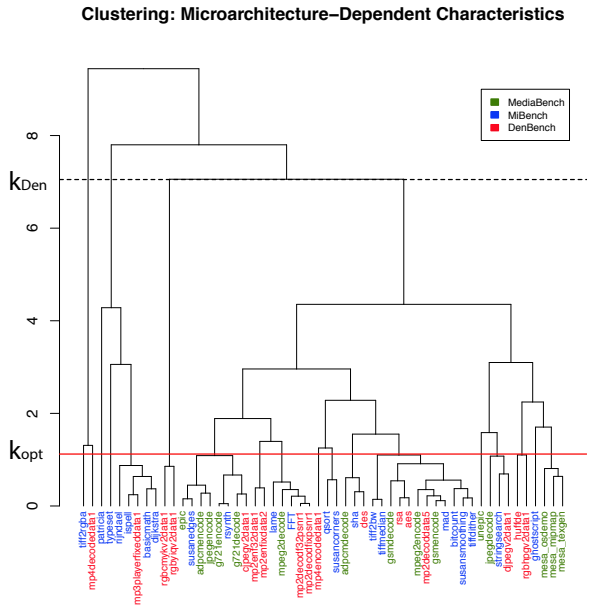


Figure 7: Agglomerative clustering for architecture.

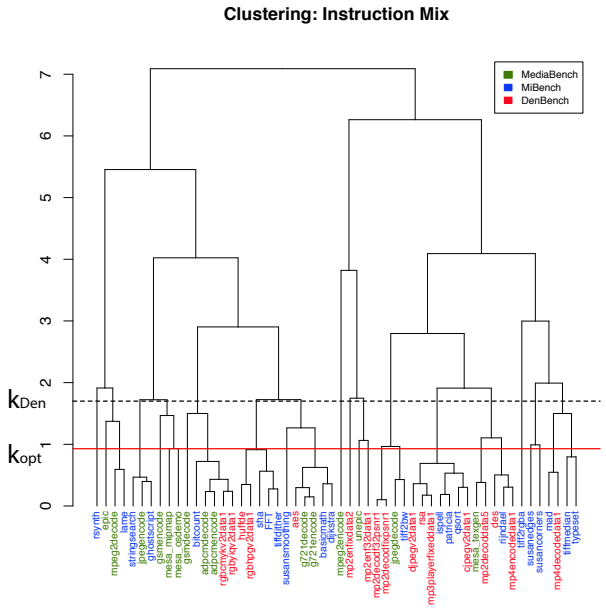


Figure 8: Agglomerative clustering for instruction mix.

In the following, we analyze the two hybrid benchmark suites generated from hierarchical clustering in a more straightforward way. Figure 7 and 8 are the dendrograms for the microarchitecture-dependent and instruction mix metrics, respectively. Each figure illustrates how each cluster is composed by drawing a ⊓-shaped link between a non-singleton cluster and its children. Link length represents the distance between single programs or clusters. The shorter the distance between clusters, the more similarity between the child clusters of a ⊓-link.

For the best number of clusters, k_{opt} chosen from Algorithm 1, we draw a horizontal line, i.e., k_{opt} in Figure 7. Based on the k_{opt} line, many programs from different benchmark suites are integrated together to clusters, each of which has multiple programs. However, there are 7 clusters comprising only a single program for microarchitecture-dependent metric in Figure 7; "tiff2rgba", "mp4decodedata1", "patricda","typeset", "mp4encodedata1", "unepic", and "jpegdecode" for their own singleton cluster.

These 7 programs must be included in the output of BenchPrime. Similarly, 8 clusters consist only of a single program for instruction mix in Figure 8, which has to be included in the final hybrid benchmark suite.

According to Algorithm 1 and 2, two horizontal lines are drawn on the dendrograms. The dashed line is k_{Den}, that is the k_{irr} of DenBench, and the red solid line is drew at the height corresponding to k_{opt} for Hybrid Benchmark. We define the dashed line as the Denbench line because at the level under the DenBench line, the subsetting have to include at least one program from the DenBench. On the contrary, if the Hybrid Benchmark line is above the DenBench line, it is not necessary to include the applications from the DenBench. In Figure 7 and 8, DenBench lines are both above the Hybrid Benchmark line.

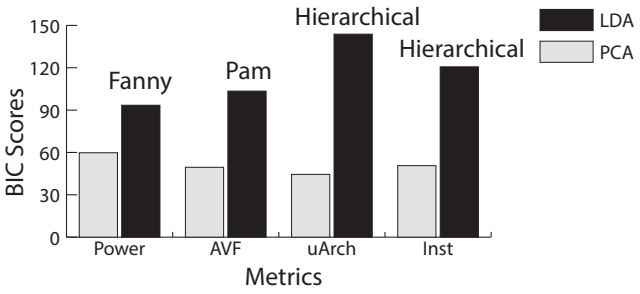


Figure 9: Optimized BIC scores: LDA .vs. PCA

Observation 2: *DenBench easily becomes necessary to build a diverse benchmark suite.*

6.5 PCA and LDA Comparison

Linear discriminant analysis (LDA) is a classification method to find a linear combination of feature that characterizes or separates two or more classes of objects or events. Compared with PCA, it tries to find the line that best separates the classes. Here, we compare the BIC score generated by the optimized clustering chooser with PCA and LDA preprocessing as input separately. There are some interesting findings by comparing them. Figure 9 shows the results of PCA and

	Benchmark suite	uArch	Inst	AVF	Power
Diversity score	MediaBench	0.41	0.43	0.40	0.43
	MiBench	0.42	0.44	0.44	0.45
	DenBench	0.47	0.50	0.47	0.51
Irreplaceability score	MediaBench	0.25	0.73	-0.5	0.54
	MiBench	0.70	0.69	0.42	0.50
	DenBench	0.80	0.38	0.70	0.73

Table 3: Diversity and Irreplaceability scores of the three benchmark suites for each metric

LDA for three metrics. The left half is for LDA and the right part is for PCA.

Observation 3: *LDA is more accurate than PCA as data preprocessing method.*

According to the figure, the optimized BIC score corresponding to LDA is more than 90 for all of the four metrics. Among them, the optimized BIC score 143.7694 for a microarchitecture-dependent metric is the highest one. On average, the BIC score for LDA nearly doubles the score for PCA. This phenomenon is due to the fact that the labeled data provides a more precise guide to the clustering chooser to make the best partition. The more reasonable clustering improves the probability of "goodness of fit", which actually is what the BIC score measures.

6.6 Benchmark Ranking

An algorithmic approach is leveraged to ground the discussion of diversity and irreplaceability of each benchmark suite compared to BenchPrime's hybrid benchmark suite.

6.6.1 Diversity Score. To analyze the diversity of a special benchmark suite, this work counts the proportion of representative applications to the original full set. The top of Table 3 shows the diversity scores for MediaBench, MiBench, and DenBench on the four metrics. Every diversity score is calculated from Equation 4. Interestingly, MediaBench and MiBench do not have much difference in terms of their diversity, even though their benchmark suite sizes are quite different. In contrast, DenBench has a higher diversity score than those of MediaBench and MiBench.

Observation 4: *DenBench owns more dispersed features, and the program distribution within it is more diverse.*

6.6.2 Irreplaceability Score. To measure to what degree one benchmark suite is irreplaceable with other input suites, we compare BenchPrime's optimized clustering with the minimum clustering (with the least number of clusters) where the lack of the one benchmark suite causes the loss of a cluster. The bottom of Table 3 shows the irreplaceability scores for MediaBench, MiBench, and DenBench on the four metrics separately, calculated from Equation 5. The higher the irreplaceability score is, the more critical role the benchmark suite plays to build a balanced hybrid benchmark suite. There is a negative score for MediaBench under AVF metric, i.e., the k_{irr} of MediaBench is bigger than the k_{opt}. Note, this is not contradictory to the data shown in Figure 6 because it picks the closest-to-centroid point as a representative application for the cluster which does not imply the necessity of including Mediabench. It is possible to build a hybrid benchmark suite for the AVF metric from the k_{opt} clusters without MediaBench at all by picking DenBench or MiBench program in every cluster to which MediaBench program belongs. However, MediaBench has the highest score for an instruction mix metric; this demonstrates that the benchmark suite with low contribution to the hybrid benchmark suite of one is still able to dominate other metric's hybrid benchmark suite. DenBench plays an important role on the final hybrid benchmarks suite, especially for microarchitecture-dependent, AVF, and power metrics. DenBench contributes significantly to the hybrid benchmark suite; the irreplaceability scores are $0.7 - 0.8$ and quite close to the perfect score 1.0. Even for AVF whose score is 0.7, the resulting hybrid benchmark needs DenBench for high diversity.

Observation 5: *DenBench is more necessary than MediaBench and MiBench to construct an application-balanced hybrid benchmark suite.*

This phenomenon is consistent with Figure 7, excluding DenBench is possible until the dotted line reaches 4 clusters. Because at this point, at least one of "rgbcmykv2data1" and "rgbyiqv2data1" from DenBench forms a single cluster which has to be included for BenchPrime to build a diverse hybrid benchmark suite. While in Figure 8, the dotted line k_{Den} is close to k_{opt}, meaning that DenBench is negligible until 16 clusters.

6.7 Analysis Time

BenchPrime's analysis times under uArch/Inst/AVF/Power metrics are 37.4/51.2/49.2/48.5 seconds, respectively. Across different metrics, the analysis of an instruction mix metric spends the longest time while that of a microarchitecture-dependent metric spends the shortest time, even if both of them reflect the architecture characteristics. According to our analysis, Fanny and Som among the ten clustering algorithm candidates (shown in Section 4.2.1) take relatively longer than others. Overall, for all four metrics, BenchPrime spends less than 60 seconds, which illustrates that it is practical for real use. Furthermore, it would be possible to accelerate the clustering algorithms by improving the data structure usage [24–26, 30] and leveraging dynamic parallelism adaptation [28, 29, 32, 37, 38].

7 RELATED WORK

Although this is the first effort to do a direct comparison of MediaBench, MiBench with DenBench created by EEMBC to the best of our knowledge, there exists a large body of work in characterizing and subsetting benchmarks. Because of the

high-dimensional and correlated nature of these characteristics, most of the previous work leverages similar statistical tools such as PCA and clustering algorithm to assist the discussion [18, 19, 23]. For GPGPU benchmark characterization, Adhinrayanan et al. provide a thorough taxonomy in documenting such previous work [1]. Eeckhout et al. leverage PCA and hierarchic clustering based dendrograms to analyze a range of architectural characteristics such as cache behavior and instruction level parallelism, with input data also put into consideration alongside the programs themselves [15]. The full spectrum of characteristics are put into one single PCA without dividing them into finer scales. Hoste et al. point out the potential pitfalls in using microarchitecture-dependent measurements, while proposing a range of key microarchitecture-independent characteristics for benchmark comparison [17]. Phansalka et al. follow a similar direction in providing analysis for program inherent characteristics without compromising fairness on a particular microarchitecture [56]. Yi et al. evaluate the various benchmark subsetting methods, in which PCA is identified as a precise approach in producing representative subsets[67]. However, all these methods do not provide a basis for why this clustering algorithm is better than others.

Apart from the ones that aim at proposing new methodologies, there exist numerous efforts in analyzing a specifically targeted benchmark set. Closely related to this paper, Poovey et al. [58] present a thorough study of the EEMBC benchmark suite (DenBench), in which a broad range of architectural behaviors are evaluated within the benchmark suite. Phansalka et al. [54, 57] give an in-depth study for the balance and redundancy in the SPEC 2006 benchmarks, while providing subsetting suggestions for both the applications and the input data set. A similar set of architectural characterization is applied to JavaScript benchmarks by Tivari et al. [65], in which both PCA visualization and agglomerative clustering are leveraged as main analyzing tools. Representativeness of embedded Java benchmarks is discussed by Isen et al. in [19], and they also provide a comparison between desktop and embedded versions. Jia et al. [67] and Zhen et al. [22] carry out both characterization and subsetting specifically targeting big data workloads. Recently, Peters at al. [53] characterize a power behavior of web browser workloads on top of heterogeneous multi-processing platforms and propose power management strategies based on the characterization. Finally, Adolf et al. [2] analyze modern deep learning workloads and report their behavior in inference and training.

8 CONCLUSIONS AND FUTURE WORK

This work presents BenchPrime, a systematic benchmark analysis framework that can automatically analyze the similarity and diversity of multiple benchmark suites and generate the hybrid benchmark suite. Unlike prior work, BenchPrime can select the best-performing clustering algorithm which often varies depending on not only the input benchmark suites but also their comparison metrics. This is particularly important because unoptimized clustering algorithms do not effectively represent the original benchmark suites. BenchPrime's optimized clustering algorithm selection enables users to build the hybrid benchmark suite with high accuracy from multiple suites in an automated and metric-tailored manner.

We believe that BenchPrime's findings on the comparison of MediaBench and MiBench with DenBench will be useful to the embedded systems community. Although we only use the three benchmark suites in the current evaluation, BenchPrime is generally applicable to others suites. Our future work will leverage BenchPrime for characterizing and subsetting parallel benchmark suites. It would be another interesting future work to apply BenchPrime to program bug benchmark suites [49, 68] used for data race detection [69, 70] and memory leak detection [27, 40].

Finally, we envision that BenchPrime can not only be a useful tool for subsetting a benchmark set but also serve as a communication channel between the benchmark companies and the researchers who are potential customers for the benchmark products. The companies will be able to offer an evaluation methodology of their new benchmark suite open to the public without releasing the source code. Meanwhile, the researchers will be able to decide whether to purchase the new benchmark suite by comparing it against their existing suites with the help of BenchPrime.

REFERENCES

[1] Vignesh Adhinarayanan and Wu-chun Feng. 2015. An Automated Framework for Characterizing and Subsetting GPGPU Workloads. (2015).

[2] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2016. Fathom: reference workloads for modern deep learning methods. In *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 1–10.

[3] Charu C. Aggarwal and Chandan K. Reddy (Eds.). 2014. *Data Clustering: Algorithms and Applications*. CRC Press.

[4] Avinash C. Kak. Aleix M. Martinez. 2001. PCA versus LDA. In *IEEE Transaction on Pattern Analysis and Machine Intelligence*. IEEE Computer Society, 228–233.

[5] L. Barroso and U. Holzle. 2007. The Case for Energy-Proportional Computing. 40, 12 (2007), 33–37.

[6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (2011), 1–7.

[7] C.M. Bishop et al. 2006. *Pattern recognition and machine learning*. Springer New York:.

[8] Garo Bournoutian and Alex Orailoglu. 2009. Reducing Impact of Cache Miss Stalls in Embedded Systems by Extracting Guaranteed Independent Instructions. In *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '09)*. ACM, New York, NY, USA, 117–126. https://doi.org/10.1145/1629395.1629413

[9] Guy Brock, Vasyl Pihur, Susmita Datta, and Somnath Datta. 2008. clValid: An R Package for Cluster Validation. *Journal of Statistical Software* 25, 1 (2008), 1–22. https://doi.org/10.18637/jss.v025.i04

[10] D. Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. A framework for architectural-level power analysis and optimizations. 83–94.

[11] Songah Chae, Doo-Hyun Kim, Changhee Jung, Duk-Kyun Woo, and Chaedeok Lim. 2007. Experimental Analysis on Time-Triggered Power Consumption Measurement with DVS-Enabled Multiple Power Domain Platform. In *Software Technologies for Embedded and Ubiquitous Systems, 5th IFIP WG 10.2 International Workshop, SEUS 2007, Santorini Island, Greece, May*

2007. Revised Papers. 149–158.

[12] Moslem Didehban, Dheeraj Lokam, and Aviral Shrivastava. 2017. An Integrated Safe and Fast Recovery Scheme from Soft Errors. In *Proceedings of The 54th Annual Design Automation Conference (DAC)*.

[13] Moslem Didehban and Aviral Shrivastava. 2016. NZDC: A Compiler technique for near Zero Silent data Corruption. In *Proceedings of The 53rd Annual Design Automation Conference (DAC)*.

[14] Moslem Didehban, Aviral Shrivastava, and Dheeraj Lokam. 2017. NEMESIS: A Software Approach for Computing in Presence of Soft Errors. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[15] Lieven Eeckhout, Hans Vandierendonck, and Koen Bosschere. 2002. Workload design: Selecting representative program-input pairs. In *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on*. IEEE.

[16] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. 3–14.

[17] Kenneth Hoste and Lieven Eeckhout. 2006. Comparing benchmarks using key microarchitecture-independent characteristics. In *Workload Characterization, 2006 IEEE International Symposium on*. IEEE, 83–92.

[18] Kenneth Hoste, Aashish Phansalkar, Lieven Eeckhout, Andy Georges, Lizy K John, and Koen De Bosschere. 2006. Performance prediction based on inherent program similarity. In *Parallel Architectures and Compilation Techniques (PACT), 2006 International Conference on*. IEEE, 114–122.

[19] Ciji Isen, Lizy John, Jung Pil Choi, and Hyo Jung Song. 2008. On the representativeness of embedded Java benchmarks. In *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*. IEEE, 153–162.

[20] Adam N Jacobvitz, Andrew D Hilton, and Daniel J Sorin. 2015. Multi-program benchmark definition. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*. IEEE, 72–82.

[21] Reiley Jeyapaul, Abhishek Risheekesan, Aviral Shrivastava, and Kyoungwoo Lee. 2014. UnSync-CMP: Multicore CMP Architecture for Energy Efficient Soft Error Reliability. *Transactions on Parallel and Distributed Systems* 25, 1 (January 2014), 254–263.

[22] Zhen Jia, Jianfeng Zhan, Lei Wang, Rui Han, Sally A. McKee, Qiang Yang, Chunjie Luo, and Jingwei Li. 2014. Characterizing and Subsetting Big Data Workloads. *CoRR* abs/1409.0792 (2014). http://arxiv.org/abs/1409.0792

[23] Ajay Joshi, Aashish Phansalkar, Lieven Eeckhout, and Lizy Kurian John. 2006. Measuring benchmark similarity using inherent program characteristics. *IEEE Trans. Comput.* 55, 6 (2006), 769–782.

[24] Changhee Jung. 2013. *Effective techniques for understanding and improving data structure usage*. Ph.D. Dissertation. Georgia Institute of Technology, Atlanta, GA, USA. http://hdl.handle.net/1853/49101

[25] Changhee Jung and Nathan Clark. 2009. DDT: design and evaluation of a dynamic program analysis for optimizing data structure usage. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42nd)*.

[26] Changhee Jung and Nathan Clark. 2009. Toward automatic data structure replacement for effective parallelization. In *Proc. of the Workshop on Parallel Execution of Sequential Programs on Multicore Architectures*. 2–11.

[27] Changhee Jung, Sangho Lee, Easwaran Raman, and Santosh Pande. 2014. Automated Memory Leak Detection for Production Use. In *Proceedings of the 36th International Conference on Software Engineering*.

[28] Changhee Jung, Daeseob Lim, Jaejin Lee, and SangYong Han. 2005. Adaptive execution techniques for SMT multiprocessor architectures. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 236–246.

[29] Changhee Jung, Daeseob Lim, Jaejin Lee, and Yan Solihin. 2006. Helper thread prefetching for loosely-coupled multiprocessor systems. In *Proceedings of the 20th international conference on Parallel and distributed processing (IPDPS'06)*. IEEE Computer Society, Washington, DC, USA, 140–140. http://dl.acm.org/citation.cfm?id=1898953.1899071

[30] Changhee Jung, Silvius Rus, Brian P. Railing, Nathan Clark, and Santosh Pande. 2011. Brainy: effective selection of data structures. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation (PLDI '11)*. ACM, New York, NY, USA, 86–97.

[31] Changhee Jung, Duk-Kyun Woo, Kanghee Kim, and Sung-Soo Lim. 2007. Performance characterization of prelinking and preloading for embedded systems. In *Proc. of the 7th ACM & IEEE EMSOFT*. New York, NY, USA.

[32] Chang Hee Jung, Dae Seob Lim, Jae Jin Lee, and Sang Yong Han. 2009. Adaptive execution method for multithreaded processor-based parallel system. (April 28 2009). US Patent 7,526,637.

[33] Soontae Kim, N. Vijaykrishnan, Mahmut Kandemir, Anand Sivasubrmaniam, and M.J Irwin. 2003. Partitioned instruction cache architecture for energy efficiency. 2, 2 (March 2003), 163–165.

[34] Yohan Ko, Reiley Jeyapaul, Youngbin Kim, Kyoungwoo Lee, and Aviral Shrivastava. 2015. Guidelines to Design Parity Protected Write-back L1 Data Cache. In *Proceedings of The 52nd Annual Design Automation Conference (DAC)*.

[35] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization (CGO '04)*. IEEE Computer Society, Washington, DC, USA, 75–. http://dl.acm.org/citation.cfm?id=977395.977673

[36] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. 1997. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO 30)*. IEEE Computer Society, Washington, DC, USA, 330–335. http://dl.acm.org/citation.cfm?id=266800.266832

[37] Jaejin Lee, Changhee Jung, Daeseob Lim, and Yan Solihin. 2009. Prefetching with Helper Threads for Loosely Coupled Multiprocessor Systems. *IEEE Transactions on Parallel and Distributed Systems* 20, 9 (2009), 1309–1324. https://doi.org/10.1109/TPDS.2008.224

[38] Jaejin Lee, Jung-Ho Park, Honggyu Kim, Changhee Jung, Daeseob Lim, and SangYong Han. 2010. Adaptive execution techniques of parallel programs for multiprocessors. *J. Parallel Distrib. Comput.* 70, 5 (May 2010), 467–480.

[39] Kyoungwoo Lee, Aviral Shrivastava, Ilya Issenin, Nikil Dutt, and Nalini Venkatasubramanian. 2006. Mitigating Soft Error Failures for Multimedia Applications by Selective Data Protection. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06)*. ACM, New York, NY, USA, 411–420. https://doi.org/10.1145/1176760.1176810

[40] Sangho Lee, Changhee Jung, and Santosh Pande. 2014. Detecting Memory Leaks Through Introspective Dynamic Behavior Modelling Using Machine Learning. In *Proceedings of the 36th International Conference on Software Engineering*.

[41] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42)*. ACM, New York, NY, USA, 469–480. https://doi.org/10.1145/1669112.1669172

[42] Yuan Lin et al. 2006. SODA: A Low-power Architecture for Software Radio.

[43] Qingrui Liu and Changhee Jung. 2016. Lightweight Hardware Support for Transparent Consistency-Aware Checkpointing in Intermittent Energy-Harvesting systems. In *Proceedings of the IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*.

[44] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2015. Clover: Compiler Directed Lightweight Soft Error Resilience. In *Proceedings of the 16th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems 2015 CD-ROM (LCTES'15)*. ACM, New York, NY, USA, Article 2, 10 pages. https://doi.org/10.1145/2670529.2754959

[45] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler-Directed Lightweight Checkpointing for Fine-Grained Guaranteed Soft Error Recovery. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*.

[46] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler Directed Soft Error Detection and Recovery to

Avoid DUE and SDC via Tail-DMR. *ACM Transactions on Embedded Computing Systems (TECS)* XX, X (2016).

[47] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Low-Cost Soft Error Resilience with Unified Data Verification and Fine-Grained Recovery. In *Proceedings of the 49th International Symposium on Microarchitecture (MICRO)*.

[48] Qingrui Liu, Xiaolong Wu, Larry Kittinger, Markus Levy, and Changhee Jung. 2017. BenchPrime: Effective Building of a Hybrid Benchmark Suite. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 5s (2017), 179.

[49] Shan Lu, Zhenmin Li, Feng Qin, Lin Tan, Pin Zhou, and Yuanyuan Zhou. 2005. Bugbench: Benchmarks for evaluating bug detection tools. In *In Workshop on the Evaluation of Software Defect Detection Tools*.

[50] Shubhendu S. Mukherjee, Joel Emer, and Steven K. Reinhardt. 2005. The Soft Error Problem: An Architectural Perspective. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA '05)*. 243–247.

[51] S. S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. 2003. A Systematic Methodology to Compute the ARchitectural Vulnerability Factors for a High Performance Microprocessor. 29–42.

[52] J.C. Nunnally. 1978. *Psychometric theory*. McGraw-Hill. https://books.google.com/books?id=WE59AAAAMAAJ

[53] Nadja Peters, Sangyoung Park, Samarjit Chakraborty, Benedikt Meurer, Hannes Payer, and Daniel Clifford. 2016. Web browser workload characterization for power management on HMP platforms. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 26.

[54] Aashish Phansalkar, Ajay Joshi, Lieven Eeckhout, and Lizy Kurian John. 2005. Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2005, March 20-22, 2005, Austin, Texas, USA, Proceedings*. 10–20.

[55] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. 2007. Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite. *SIGARCH Comput. Archit. News* 35, 2 (June 2007), 412–423. https://doi.org/10.1145/1273440.1250713

[56] Aashish Phansalkar, Ajay Joshi, and Lizy Kurian John. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite.. In *ISCA*. ACM.

[57] Aashish Phansalkar, Ajay Joshi, and Lizy K John. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *ACM SIGARCH Computer Architecture News*, Vol. 35. ACM, 412–423.

[58] Jason A. Poovey, Thomas M. Conte, Markus Levy, and Shay Gal-On. 2009. A Benchmark Characterization of the EEMBC Benchmark Suite. *IEEE Micro* 29, 5 (2009), 18–29.

[59] Peter Sassone, D. Scott Wills, and Gabriel Loh. 2005. Static strands: safely collapsing dependence chains for increasing embedded power efficiency. 127–136.

[60] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*. ACM, New York, NY, USA, 45–57. https://doi.org/10.1145/605397.605403

[61] Aviral Shrivastava, Illya Issenin, and Nikil Dutt. 2005. Compilation techniques for energy reduction in horizontally partitioned cache architectures.

[62] Ashish Shrivastava, Abhishek Rhisheekesan, Reiley Jeyapaul, and Carole-Jean Wu. 2014. Quantitative analysis of control flow checking mechanisms for soft errors. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*. IEEE, 1–6.

[63] Vilas Sridharan and David R Kaeli. 2010. Using hardware vulnerability factors to enhance AVF analysis. In *ACM SIGARCH Computer Architecture News*.

[64] Devesh Tiwari and Yan Solihin. 2012. Architectural characterization and similarity analysis of sunspider and Google's V8 Javascript benchmarks.. In *ISPASS*. IEEE Computer Society, 221–232.

[65] V. Tiwari, S. Malik, and A. Wolfe. 1994. Power analysis of embedded software: A first step towards software power minimization. 2, 4 (1994), 437–445.

[66] Jan Vitek and Tomas Kalibera. 2011. Repeatability, reproducibility, and rigor in systems research. In *Proceedings of the ninth ACM international conference on Embedded software*. ACM, 33–38.

[67] Joshua J Yi, Resit Sendag, Lieven Eeckhout, Ajay Joshi, David J Lilja, and Lizy K John. 2006. Evaluating benchmark subsetting approaches. In *Workload Characterization, 2006 IEEE International Symposium on*. IEEE, 93–104.

[68] Jie Yu, Satish Narayanasamy, Cristiano Pereira, and Gilles Pokam. 2012. Maple: A Coverage-driven Testing Tool for Multithreaded Programs. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '12)*. ACM, New York, NY, USA, 485–502. https://doi.org/10.1145/2384616.2384651

[69] Tong Zhang, Changhee Jung, and Dongyoon Lee. 2017. ProRace: Practical Data Race Detection for Production Use. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 149–162.

[70] Tong Zhang, Dongyoon Lee, and Changhee Jung. 2016. Txrace: Efficient data race detection using commodity hardware transactional memory. In *ACM SIGOPS Operating Systems Review*, Vol. 50. ACM, 159–173.