FAB : Framework for Analyzing Benchmarks

Varun Gohil*
Indian Institute of Technology Gandhinagar
gohil.varun@iitgn.ac.in

Shreyas Singh*
Indian Institute of Technology Gandhinagar
shreyas.singh@iitgn.ac.in

Manu Awasthi
Ashoka University
manu.awasthi@ashoka.edu.in

ABSTRACT

Performance evaluation is an integral part of computer architecture research. Rigorous performance evaluation is crucial in order to evaluate novel architectures, and is often carried out using benchmark suites. Each suite has a number of workloads with varying behavior and characteristics. Most analysis is done by analyzing the novel architecture across *all* workloads of a *single* benchmark suite. However, computer architects studying optimizations of specific microarchitectural components, require evaluation of their proposals on workloads that stress the component being optimized across *multiple* benchmark suites.

In this paper, we present the design and implementation of FAB - a framework built with Pin and Python based workflow. FAB allows user-driven analysis of benchmarks across multiple axes like instruction distributions, types of instructions etc. through an interactive Python interface to check for desired characteristics, across multiple benchmark suites. FAB aims to provide a toolkit that would allow computer architects to 1) select workloads with desired, user-specified behavior, and 2) create synthetic workloads with desired behavior that have a grounding in real benchmarks.

CCS CONCEPTS

• **General and reference** → *Empirical studies*; • **Human-centered computing** → Visualization.

KEYWORDS

workload analysis, instruction mix, workload selection, workload similarity

ACM Reference Format:
Varun Gohil, Shreyas Singh, and Manu Awasthi. 2019. FAB : Framework for Analyzing Benchmarks. In *Tenth ACM/SPEC International Conference on Performance Engineering Companion (ICPE '19 Companion), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3302541.3313102

1 INTRODUCTION

Performance evaluation is an integral part of computer architecture research, and is typically carried out with the help of benchmark suites. These benchmark suites consist of a number of workloads which are generally representative of a particular application domain. Benchmark suites are used to quantitatively measure the performance of system or a sub-system thereof. Computer architecture research hinges on the ability of architects to select the *right* set of workloads required to demonstrate the benefits of proposed enhancements to existing architectures.

Selecting the right set of workloads for performance evaluation of an architecture is not an easy task due to multiple reasons. First, a benchmark suite comprises of a number of workloads that are representative of the typical characteristics and behaviors of a class of applications. For example, SPEC CPU2017 [2] suite comprises of workloads that are compute-intensive and stress the CPU. Similarly, there are individual benchmarks like STREAM [12] and that stress the memory sub-system. Other, multi-threaded benchmark suites like PARSEC 3.0 [5] which represent emerging application domains, or SPLASH 3.0 [10] which represent complex, parallel applications are also very popular for evaluating CPU architectures. In addition, a number of these workloads could be executed using different input sets. As a result, multiple behaviors could be exhibited by a single workload, depending on provided inputs. Secondly, for evaluating the *same* sub-system across multiple contexts could lead to different benchmark suites being used. As an example, the performance of CPU architectures across most verticals can be evaluated using standard, well-known benchmarks suites like SPEC CPU 2017 [2], PARSEC 3.0 [5], SPLASH 3.0 [10]. If we increase the scope to include the server and high performance computing space as well, this list grows to include suites like TPC [13], LINPACK [7] and NAS parallel benchmarks [4].

Thirdly, a benchmark suite is an aggregation of a large number of disparate applications where each workload can potentially have multiple phases of execution [15]. Typically, in a given phase, a workload stresses one or a small subset of system components. For example, a memory intensive phase could potentially be followed by a compute intensive phase. Not only that, each successive compute intensive phase (of the same workload) might stress different execution units: if the current phase is dominated by integer operations, the next one could potentially be dominated by floating point operations. The behavior of each phase and their order is predominantly defined by the application.

This varied space could become especially difficult to maneuver for architects and designers who are trying to optimize a specific components of the architecture, say the floating point subsystem. Once an enhancement has been designed, the architects would like to study the system level implications of the proposed optimization in terms of various metrics for performance and/or energy consumption. As a result, the architect would like to select the workloads from all possible available suites that show a significant amount of floating point activity.

*Both authors contributed equally to this research

ICPE '19 Companion, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6286-3/19/04…$15.00
https://doi.org/10.1145/3302541.3313102

Figure 1: Framework Flow

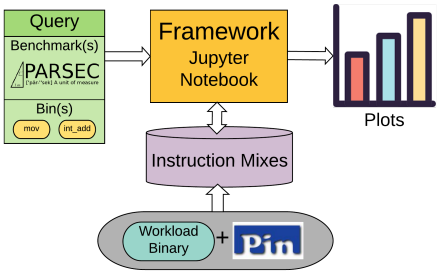
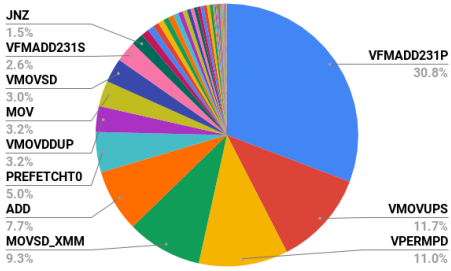


Figure 2: Instruction Mix of dgemm benchmark



Currently, selecting a set of workloads that satisfy the criteria specified by an architect, is a very difficult task, mostly because there is no central repository which can help architects compare workload characteristics across multiple suites in one place. Furthermore, there are no existing mechanisms for an architect to create synthetic workload(s) by putting together interesting phases of multiple workloads, potentially from multiple benchmark suites. Finally, there are very few, well-defined axes using which an architect can specify the requirements of a workload that she is interested in.

In this paper, we present FAB which aims to facilitate the process of workload selection and creation. FAB presents a Pin and Python based workflow which helps an architect select interesting workloads from a wide variety of available workloads, across benchmark suites. FAB does this by making the following important contributions:

(1) FAB provides a mechanism of instruction classification for the x86_64 instruction set. We manually classify the entire instruction set in to multiple bins, with each bin representing a set of instructions with similar characteristics. These bins can then be used by the architects for specifying a number of things ranging from analysis of a single workload to comparison across workloads. Currently, FAB provides 43 such bins that cover all instructions of the x86_64 instruction set. The FAB back-end allows this information to be generated for any new workload compiled for the x86_64 ISA.
(2) FAB provides an interactive Python notebook to visualize the architectural characteristics of multiple workloads, from across benchmark suites. This visualization is user defined and can be along as many axes/bins as she desires. The notebook also provides for visualizing the similarities (or differences) along user specified axes, across various workloads.

2 FRAMEWORK DESCRIPTION

As shown in Figure 1, FAB's workflow currently consists of two parts. The first part, or the backend, deals with a Pin based flow, which takes a workload binary and the associated input files as input, and generates an instruction mix for the workload as output. This has to be done once for every new workload. The instruction mix is then used for various kinds of analyses using the tool's frontend, which is a Jupyter notebook. The frontend also contains the pre-classified instruction bins, which can be referenced directly from within the notebook. The notebook takes the benchmark and instruction bins as input and produces stacked barcharts and dendrograms which assist in analysis. We describe the overall workflow in more detail, next.

2.1 Instruction Mixes

To understand a workload's behavior it is important to analyze different characteristics of the workload. Examples of these characteristics include the instruction mix, instruction level parallelism, branch transition rate and working set size, among others.

Currently, FAB uses the *instruction mixes* as the primary characteristic for facilitating analysis. Instruction mixes include the various instructions executed by a workload and their frequency. Instruction mixes are chosen since they are a microarchitecture independent characteristic; they depend on the ISA, the source code and the compilation method, making it completely independent of the circuit-level implementation. This ensures that instruction mixes can be used for picking workloads as long as the ISA, and compilation flags remain the same. While some prior work has used instruction mixes [6, 14] as a characteristic for studying workload behavior, they do not use an exhaustive instruction mix. Instead, most limit themselves to the study of memory and branch operations obtained using the *perf* utility. FAB, on the other hand provides the capability to study workloads across all possible instruction mixes for the x86_64 ISA.

Currently, FAB contains instruction mixes of **SPEC CPU 2017** [2], **PARSEC 3.0** [5], **SPLASH 3** [10] and **OpenBLAS** [1] benchmark suites. These mixes were generated using the *opcodemix* tool of Pin [11] for x86_64 ISA. Figure 2 provides an example of the dynamic instruction profile of *dgemm*, a workload from the OpenBLAS suite. As mentioned previously, this exercise for generating instruction mixes needs to be done once for every workload. For the aforementioned workloads, we carry out instruction mix generation and provide these mixes as part of the Jupyter notebook. This information is then used for a variety of analyses including inter- and intra- workload comparisons.

2.2 Instruction Binning

A focus on the just the execution frequency of instructions does not provide adequate insight to workload behavior. Most researchers prefer analyzing a specific function or aspect of the system (much like the fact that [14] and [6] look at the memory and branch operations executed). However, more interesting analyses can be carried out, if we can classify the instructions into multiple *bins*, with

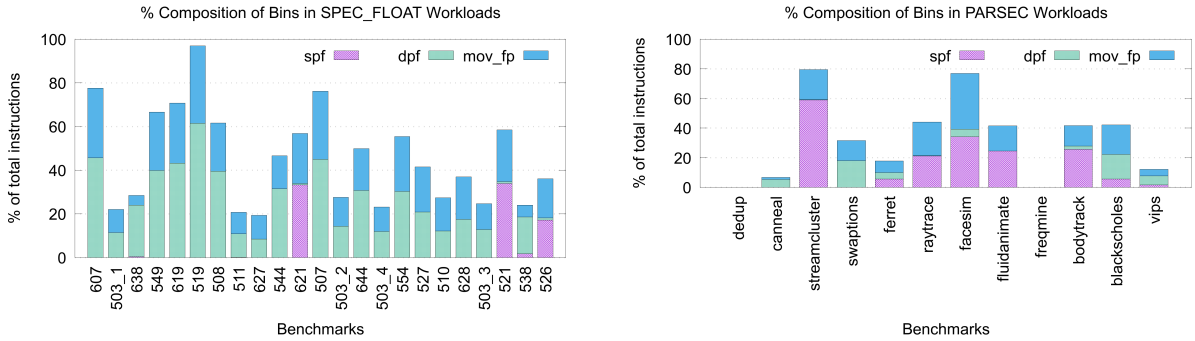


Figure 3: Floating point instructions in SPEC CPU 2017 and PARSEC 3.0

each bin representing a specific type of instructions. For example, branches and integer instructions could be a bin each, bitwise logical operations could be another.

However, care has to be taken regarding the granularity of the bins. Too fine a granularity would result in a large number of bins. As a result, the analyses generated using those would be hard to reason about. On the other hand, coarser granularity will result in a small number of bins, which would potentially not provide enough information. In this work, we group all the x86_64 instructions [3] into a set of **43** pre-made "bins". These include regular instructions, as well as ISA extensions, including SSE and AVX. The binning of instructions was done by manually going through the entire instruction set, and classifying them accordingly.

Table 1 lists the existing bins and the types of instructions contained in each. These bins serve as well defined axes which can help architects analyze workloads along specified axis and choose the workloads as per one's requirements. For example if one wants to analyze the double precision floating point instructions executed by the workloads one can directly use the *dpf* bin provided in the framework. To give more control to the user, the framework also supports creation and usage of custom user-defined bins, as well as grouping pre-existing bins into larger ones.

Bin Type	Bin Names
Logic (fine)	logic_and, logic_and_not, logic_not, logic_or, logic_xnor, logic_xor,
Logic (coarse)	cmp (compare), logic, shift
Arithmetic (fine)	all_add, all_sub, all_mul, all_div, all_spc, int_add, int_sub, int_mul, int_div, int_spc, spf_add, spf_sub, spf_mul, spf_div, spf_spc, dpf_add, dpf_sub, dpf_mul, dpf_div, dpf_spc, int_add_sub, int_add_mul, dpf_add_sub_mul, spf_add_sub_mul
Arithmetic (coarse)	all, int, spf, dpf
Branch	brn
Move	mov, mov_fp
Others	avxsse, con (convert), spc (special)

Table 1: Bin Types

3 USE CASES

In this section, we present details on two present use cases of FAB, namely workload selection and workload similarity analysis.

3.1 Workload Selection

Using the framework a computer architect can not only select a set of workloads from within a benchmark suite but also across multiple benchmarks. Suppose a computer architect modifies the Floating Point Unit (FPU) to optimize execution of single precision floating point instructions. For evaluating such optimization, the selected workloads should have significant number of single precision floating point instructions, which might not be the case for all the workloads in a given suite – the architect might want to choose workloads that best suit her needs from across multiple suites.

As an example, we use FAB to select a suitable set of workloads from PARSEC 3.0 and SPEC CPU2017 suites. For SPEC CPU 2017, we only consider workloads in the *float* sub-suite. On specifying spf (single precision floating pont), dpf (double precision floating point) and fp_mov (data movement floating point) bins as those of interest, FAB generates plots in Figure 3. Figure 3 shows the percentage of single precision floating point instructions in workloads of SPEC and PARSEC. Only 5 out of 12 PARSEC workloads have greater than 20% single precision floating point instructions, while only 2 out of 23 SPEC FP 2017 benchmarks show a similar statistic. Hence, a set of workloads for analyzing the benefits of the FPU optimization would contain streamcluster, facesim workloads from PARSEC and 521, 621 workloads from SPEC CPU 2017. Each of these workloads has more than 30% single precision floating point instructions. Similarly, the architect can also specify *multiple* such criteria when selecting workloads that match her preferences.

3.2 Workload Similarity Analysis

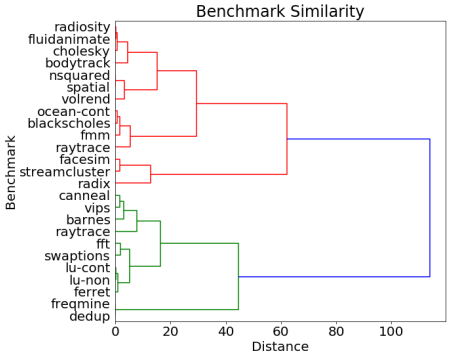
Similarity analysis within a suite or across multiple suites can help architects understand similarities, if any that might exist between different workloads of the suites. This is useful for creating a reduced workload set by removing workloads with mostly similar behavior. For finding similarity between workloads, the framework uses agglomerative hierarchical clustering [8]. This requires each workload to be represented as an n-dimensional vector, which we refer to as the *characteristic vector*. Currently, the elements of the characteristic vector are the percentage frequency of bins as specified by the user.

The workloads are iteratively grouped into clusters based on the distance between their characteristic vectors. Each cluster contains workloads which exhibit similar behavior with regards to specified bins. This process can be represented as a dendrogram. A dendrogram showing the similarity between PARSEC 3.0 and SPLASH 3 with *spf, dpf, fp_mov* bins (floating point instructions) as the characteristic vector is shown in Figure 4. From Figure 4 one can easily divide the workloads into two clusters - one cluster contains all workloads joined with red line and the other contains all the workloads joined with green line. Thus workloads within the same

cluster show similar behavior in terms of floating point instructions executed. Using this mechanism one can get a workload set with each workload having similar behavior for the specified bins.

The generated dendrograms can also be used for finding a reduced workload set from a benchmark suite which has a wide coverage of workload behaviors across the entire suite. Instead of selecting workloads having similar behavior, we can select workloads with diverse behaviors to increase coverage. The methodology for finding a reduced set from a dendrogram is described in [14].

Figure 4: Dendrogram between PARSEC and SPLASH



4 FUTURE WORK

4.1 Extending FAB

FAB is easily extensible owing to pre-defined hierarchy of the underlying database of workload characteristics. In the future, we plan on adding more analytical and visualization features to the front-end, as well as more information to the back-end to enable it to carry out multiple types of analyses. Some of the proposed future work envisioned for FAB is enumerated below.

- Enable FAB analyses to support a richer set of metrics like working set size and branch transition rates, in addition to instruction mixes.
- Enable FAB to analyse a broader set of benchmark suites by adding characterization data from suites like BioPerf (bioinformatics), MediaBench (media) and Moby (mobile applications).
- Enable FAB to carry out cross-ISA analysis of workloads by adding characterization data for suites across multiple ISAs like ARM and RISC-V.

4.2 Workload Creation

We also aim to augment the functionality of FAB to support workload creation by generating synthetic workloads with user specified characteristics. Previous work has shown that statistical profiles of workloads can be used for generating synthetic workloads [9]. These workloads are typically shorter than the actual workloads, leading to reduced simulation time. By using these techniques, the performance characteristics can quickly converge to a steady state which makes such workloads suitable for accelerated design space exploration[8]. Enhancements to specific components of an architecture can be tested by generating workloads that heavily utilize that component.

In future, we aim to use instruction mixes to train statistical machine learning models like generative adversarial networks to generate synthetic workloads using the framework. We envision that the computer architect will be able to define not only the composition of the synthetic workload using bins but also specify different phases that she is interested in observing. Not only that, an architect might be interested in observing the behavior of a proposed architectures under a sequence of pre-specified program phases, which might be harder to find in existing applications. For example, the architect might want to observe the proposed architecture under a integer compute-intensive phase, followed by a memory-intensive phase, followed by a floating point compute-intensive phase. Allowing for the creating of such user-specified workloads, which still have a basis in "real" workloads will help reason about system design and help architects explore what-if scenarios.

5 CONCLUSIONS

In this paper, we introduce FAB - an extensible and flexible framework that allows computer architects to analyze and compare characteristics of workloads within one, and across multiple benchmark suites.

The framework currently can be used for analyzing and selecting workloads that fulfill computer architect specified criteria. By providing visual aids like dendograms, the framework allows for selecting workload sets across multiple benchmark suites by allowing for removal of workloads with similar behavior. We aim to extend FAB to support multiple ISAs and provide richer set of options to the user for specifying workload characteristics.

ACKNOWLEDGMENTS

This work was supported in part by SERB grant ECR/2017/000887, IIT Gandhinagar and Ashoka University.

REFERENCES

[1] [n. d.]. OpenBLAS : An optimized BLAS library. http://www.openblas.net/
[2] [n. d.]. SPEC CPU 2017 Documentation. https://www.spec.org/cpu2017/
[3] 2016. Intel 64 and IA-32 Architectures Software Developers Manual Volume 2A: Instruction Set Reference, A-L.
[4] D. H. Bailey et al. 1991. The NAS Parallel Benchmarks&Mdash;Summary and Preliminary Results. In *Proceedings of Supercomputing*.
[5] Christian Bienia et al. 2008. *The PARSEC Benchmark Suite: Characterization and Architectural Implications.* Technical Report TR-811-08. Princeton University.
[6] C. Bienia et al. 2008. PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip-Multiprocessors. In *Proceedings of IISWC*.
[7] Jack J. Dongarra et al. 2002. The LINPACK benchmark: Past, present, and future.
[8] Lieven Eeckhout. 2010. *Computer Architecture Performance Evaluation Methods* (1st ed.). Morgan & Claypool Publishers.
[9] L. Eeckhout et al. 2000. Performance analysis through synthetic trace generation. In *Proceedings of ISPASS*.
[10] C. Sakalis et al. 2016. Splash-3: A properly synchronized benchmark suite for contemporary research.
[11] Chi-Keung Luk et al. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings PLDI*.
[12] John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25.
[13] D. A. Menasce. 2002. TPC-W: a benchmark for e-commerce. *IEEE Internet Computing* 6, 3 (2002).
[14] R. Panda et al. 2018. Wait of a Decade: Did SPEC CPU 2017 Broaden the Performance Horizon?. In *Proceedings of HPCA*.
[15] Timothy Sherwood et al. 2002. Automatically Characterizing Large Scale Program Behavior. *SIGOPS Oper. Syst. Rev.* 36, 5 (2002).