The Alberta Workloads for the SPEC CPU 2017 Benchmark Suite

José Nelson Amaral*, Edson Borin†, Dylan Ashley*, Caian Benedicto†, Elliot Colp‡,
João Henrique Stange Hoffmam†, Marcus Karpoff*, Erick Ochoa*, Morgan Redshaw¶, Raphael Ernani Rodrigues§
* Department of Computing Science, University of Alberta, Edmonton, AB, Canada.
† Instituto de Computação, Universidade de Campinas, Campinas, SP, Brazil.
‡ Bioware, Edmonton, AB, Canada.
§ Microsoft, Redmond, WA, USA.
¶ DeepMind, London, UK.

***Abstract*—A proper evaluation of techniques that require multiple training and evaluation executions of a benchmark, such as Feedback-Directed Optimization (FDO), requires multiple workloads that can be used to characterize variations on the behaviour of a program based on the workload. This paper aims to improve the performance evaluation of computer systems — including compilers, computer architecture simulation, and operating-system prototypes — that rely on the industry-standard SPEC CPU benchmark suite. A main concern with the use of this suite in research is that it is distributed with a very small number of workloads. This paper describes the process to create additional workloads for this suite and offers useful insights in many of its benchmarks. The set of additional workloads created, named the *Alberta Workloads for the SPEC CPU 2017 Benchmark Suite*[1] is made freely available with the goal of providing additional data points for the exploration of learning in computing systems. These workloads should also contribute to ameliorate the *hidden learning* problem where a researcher sets parameters to a system during development based on a set of benchmarks and then evaluates the system using the very same set of benchmarks with the very same workloads.**

***Index Terms*—benchmarking; performance evaluation; workloads; Feedback Directed Optimization (FDO); CPU**

I. Introduction

The measurement of performance using the Standard Performance Evaluation Corporation (SPEC) Central Processing Unit (CPU) benchmark suite follows a long-established methodology. The goal that guides the curation of the SPEC CPU benchmark suite is the comparison of commercially available computing systems. Therefore, the SPEC Open Systems Group (OSG) has established a rigorous review process that must be followed in order for results to be published as sanctioned by the SPEC organization. However, that suite is also extensively used to estimate performance variations due to the application of ideas proposed in the areas of compilation, computer architecture, software stack, and others.

One (mis)use of the SPEC CPU benchmark suite, that has been of great concern for many years, is for the evaluation of code transformations that rely on a technique called Feedback-Directed Optimization (FDO). FDO can be static or dynamic and it is used mostly in compilation systems. Static FDO consists of collecting information from instrumented executions of a program. These executions are performed ahead of time and the collected information is then used to produce a new binary for the program. This binary can then be used to execute the same program with many different workloads. Dynamic FDO collects information from a program while the program is executing and uses such information to change the execution of the program. Dynamic FDO can be used for dynamic "just-in-time" (JIT) compilation [14], for selection amongst multiple versions of statically generated code, or for dynamic binary recompilation. While static FDO has not taken hold in the market, dynamic FDO is prevalent in commercial products. Dynamic FDO originally became popular with Java JIT compilers and today it is used for many dynamic programming languages.

An important issue with the use of the SPEC CPU benchmark suite for evaluation of proposed FDO-based ideas is that the SPEC Open Systems Group (OSG) publishes only two workloads for most benchmarks[2]: one workload that is called the *train* workload and another that is called the *reference* workload.[3] Partially because of the availability of only two workloads for each of the benchmarks in this widely used suite — compounded with the naming of one of them as a "train" workload — a methodology became prevalent in the literature: (1) obtain profiling data from a single instrumented run using the train workload; (2) recompile the benchmark using the profiling data with FDO enabled; (3) measure performance using the reference workload. Often the publication of results obtained with this methodology are accompanied with claims that the FDO technique studied is expected to yield the performance changes observed in the experiment.

A critical view of this FDO-evaluation method leads to im-

[1] https://webdocs.cs.ualberta.ca/~amaral/AlbertaWorkloadsForSPECCPU2017/

[2] A *benchmark*, understood as a "mark on a bench" is the combination of a program with a specific workload. However, following common practice, *benchmark* in this paper can also mean the program used to produce the benchmark without a specific workload.

[3] The SPEC CPU suite also includes a *test* input for each benchmark, but this test input yields very short runs that are only meant to check that the system is working rather than for performance measurement.

portant methodology questions. Can a single training-workload execution of each benchmark program capture the expected behaviour of the program when executing with other workloads? The answer to this question is likely to be different for each specific benchmark. The behaviour of the program during a single execution with a given workload can be thought of as a point in a high-dimensional space. Therefore the single-training-workload experiment has attempted to perform machine learning by observing a single point in the space, building a model of the behaviour of the program on that point, and then evaluating this model on a, hopefully distinct, single point of the space. Such a learning strategy is known to be inappropriate because it leads to bias towards the point used for training. If the training point and the evaluation point happen to exhibit similar behaviour, this practice may lead to overfitting. If the behaviour of the program at these two points is very distinct, it will be difficult to learn anything relevant from the learning attempt; in other words, FDO will fail. To make matters worse, in some cases the evaluation point is not distinct from the training point. In the case of the SPEC benchmarks, seldom do researchers examine how the train and reference workloads were selected. Our own investigation and conversations with SPEC CPU 2000 and SPEC CPU 2006 benchmark program authors revealed that, in some cases, the train and the reference workloads were, on purpose, selected to be a subset of each other. Thus the performance of many FDO techniques has, unwittingly, been predicted from an experiment that used essentially the same workload for training and for evaluation.

Even when FDO is not used, often the evaluation of computing systems suffers from an issue that we call *hidden learning* which consists on the researchers or developers tunning the system to select an appropriate set of static parameters and threshold values using a set of benchmarks from a given suite. After this period of ad-hoc tuning, or learning, often the constructed prototypes are evaluated using the same benchmarks and these benchmarks are often executed with the very same workloads that were used for tuning the prototype. An evaluation performed using such a methodology may fail to predict the expected performance of the proposed system "out in the wild" when it has to execute applications that were not used for the tuning or even when executing the same benchmarks with a different set of workloads.

Condensed versions, sometimes called *kernels*, of benchmarks are sometimes captured to accelerate simulation-based performance prediction. Again, the computer architecture community often uses a single workload — the reference input in the case of the SPEC CPU benchmark suite — to create these kernels [12], [15], [22]. However, for benchmarks whose behaviour changes with the workload, the goal of characterizing program behaviour cannot be attained through a single execution of the program. Using MiBench, a benchmark suite formed by much simpler programs compared with the SPEC CPU 2017 suite, Breughe et al. observed that processor customization was largely insensitive to the workloads used in the program [7]. The study summarized in this paper indicates that this observation may be true for some, but not all, the programs in the SPEC CPU 2017 suite.

Often, when issues concerning variation of program behaviour with workloads are raised during the discussion of a research manuscript under evaluation, there is an argument that the authors of the research are limited by the unavailability of additional workloads for the benchmarks. Studies that do focus on various workloads use simpler programs [11]. To help address this shortcoming, in the past several years the University of Alberta has functioned as a supporting contributor to the development of the SPEC CPU 2017 benchmark suite. During this period we have worked both independently, with the members of the OSG CPU subcommittee, and with benchmark authors, to create additional workloads for programs included in the SPEC CPU 2017 suite. In the process we have made several contributions to the development of the benchmarks themselves, and have also discovered that for some of the programs included in the suite it was possible to create a system to automatically generate new workloads. The availability of such tools will advance performance-based system research by allowing researchers to create as many workloads as is required for their experimental methodology.

This paper describes the effort to develop the Alberta Workloads for the SPEC CPU 2017 suite, an enterprise that took many years and involved many people. Additional contributions, not described here, were made to the development of the SPEC CPU 2017 suite. Our team helped clarify the behaviour of several of the benchmark candidates with the workloads that were under consideration for these candidates. In some cases our contribution lead to changes in the benchmark candidate programs themselves, in others it contributed to the decision of the SPEC OSG CPU subcommittee to abandon some of the candidates, and in others it contributed to important changes in the workloads distributed with the benchmark suite.

We also offer a method to summarize the sensitivity of benchmark behaviour to workload that attempts to characterize the influence of the workloads on the behaviour of the benchmarks and to estimate the influence of different compilers on these results. Section II reviews the use of Feedback-Directed Optimization. Section III presents some main differences between SPEC CPU 2006 and 2017. Section IV presents the Alberta Workloads. Section V summarizes the major variances in benchmark behaviour due to workloads. Section VI discusses related work. Section VII discusses research questions that could be addressed with the Alberta Workloads.

II. Feedback-Directed Optimization

FDO in practice has to overcome two important challenges: (1) the observation interference problem; and (2) the mapping of feedback collected on an old version of code to a newer version. Profiling not only generates overhead, but it may change the very behaviour of the program that it attempts to observe. The classical solutions to overcome this challenge are to either use sampled profiling for online FDO [20] or to collect information with a separate execution. Limited progress

has been made toward mapping of profiling information between different versions of a program [10], [24], [25].

FDO gained mainstream usage and acceptance with the focus on microarchitecture features when studying applications that execute in large data centres. Kanev et al. used continuous profiling, where a small fraction of a large data centre is randomly selected for profiling each day [18]. Profiling data is collected from a brief period of time in each of the sampled executions. They report great diversity in workload behaviour and confirm the importance of cache performance for overall performance of applications. Applying a similar sampling methodology, but using instrumented binaries for profiling, Chen et al. made the deployment of FDO-optimized binaries in google servers much more prominent [8].

III. From SPEC CPU 2006 to SPEC CPU 2017

The set of programs included in the SPEC CPU INT 2017 suite is fairly consistent with the programs included in the SPEC CPU INT 2006 suite. The intent of Table I is to offer a quick visual parallel between the two suites. The Intel Core i7-6700K running at 4.2 GHz was selected to display the times because at the time of writing these were the only official submissions to the SPEC webpage that presented results to the same machine with the same configuration.[4]

A similar comparative table between the SPEC CPU FP 2006 and the SPEC CPU FP 2017 is not included because of space constraints, but a comparative analysis reveals that there were more significative changes in the floating-point applications in the suite with eight new benchmarks introduced in 2017 and eleven of the 2006 benchmarks not making into the newer suite. Areas of applications no longer represented in the suite include quantum chemistry, quantum physics, linear programming, structural mechanics, and speech recognition. New areas that were introduced in the SPEC CPU 2017 suite include optical tomography for biomedical imaging, 3D rendering and animation, atmosphere and ocean modelling, image manipulation, and molecular dynamics.

IV. The Alberta Workloads

The Alberta workloads for the SPEC CPU 2017 benchmark suite were generated by one of the following methods:

- Some benchmarks use workloads contained in files that are publicly available online. For this type of benchmark, publicly available resources were downloaded and are used as new workloads. E.g. for **502.gcc_r**, some of the workloads are from the "Large single compilation-unit C programs" website [21].
- In some cases, publicly available resources can be used to create new workloads but are not suitable to include in the new workloads without modification. For this type of benchmark, multiple online resources were combined and/or modified to produce a workload. E.g., for **523.xalancgmk_r**, unsuitable XML files that led to running time that were too short in comparison with the `refrate` workload were found on XML benchmarks (XSLTMark and XMark). The solution was to combine several of these XML files into a single workload to approximate the execution time of the **refrate** workload.
- For some benchmarks, a script is included to automate the generation of workloads based on resources that can be found online.
- For some types of workloads, it is possible to generate a program that can procedurally generate new workloads. E.g. a significant programming effort went into automating the creation of workloads for **505.mcf_r**.
- Inputs for some benchmarks are not widely available online, but it is possible to manually generate workloads for such programs after reading the documentation.

A. Workloads for the SPEC CPU INT 2017 Suite

The Alberta Workloads provide new workloads to all but one, `500.perlbench_r`, of the benchmarks in the SPEC CPU INT 2017. `500.perlbench_r` is a a stripped-down version of the Perl interpreter. It accepts Perl scripts as inputs and thus a simple way to create new workloads would be to collect open-source Perl applications. However, all applications that we found contain dependencies that require the integrated compilation of C code to work. The `500.perlbench_r` benchmark was not designed to support extending Perl to use C libraries. The search process analyzed the following Perl scripts and frameworks: Perl Defence Blaster, Perl Racer and Perl Arena, BioPerl, Catlyst, and Dancer.

`502.gcc_r`: The input to this benchmark is a single file that must be preprocessed.[5] Most interesting C programs are distributed as multiple files and multiple directories and are compiled with elaborate make files. Simply concatenating the individual files into a single one and feeding it to the preprocessor is most likely to result in an incorrect workload. There are likely to be name collisions between identifiers used in different files. Moreover, preprocessing logic may produce wrong code when simply concatenated into a single file. The main challenges to automatically generate single-file workloads for the gcc benchmark include tracking all files and external declaration, name-mangling the identifiers to avoid collision, and properly handling preprocessing logic. The Alberta workloads include a tool named `OneFile` that can be used to combine multiple-file C source code into a single compilation unit that is suitable for the gcc benchmark. `OneFile` is provided "as is" and may require human intervention to transform more complex applications. However, it was used to generate workloads for three substantial code bases: `mcf`, `lbm`, and `johnripper`. The new workloads also include existing single-file C codes that are publicly available and these new ones generated with the `OneFile` tool.

[4]The actual results can be found in the SPEC website at `https://www.spec.org/cpu2006/results/res2015q4/cpu2006-20151019-37701.html` and `https://www.spec.org/cpu2017/results/res2017q2/cpu2017-20161026-00029.html`

[5]The generated workloads used gcc version 4.8.4 on Ubuntu 14.04 for preprocessing.

Application Area	**SPEC 2017**	**SPEC 2006**	**SPEC 2017 Time**	SPEC 2006 Time
Perl interpreter	500.perlbench_r	400.perlbench	542	425
Compiler	502.gcc_r	403.gcc	518	346
Route planning	505.mcf_r	429.mcf	633	333
Discrete event simulation	520.omnetpp_r	471.omnetpp	787	483
SML to HTML conversion	523.xalancbmk_r	483.xalancbmk	323	221
Video compression	525.x264_r	464.h264ref	379	575
AI: $\alpha - \beta$ tree search	531.deepsjeng_r	458.sjeng	373	562
AI: Sudoku recursive solution	548.exchange3_r		498	
Data compression	557.xz_r	401.bzip2	532	681
AI: Go game playing	541.leela_r	445.gobmk	586	506
Search Gene Sequence		456.hmmer		202
Physics: Quantum Computing		462.libquantum		65
AI: path finding algorithm		473.astar		461
Arithmetic Average of Times			**517**	**405**

TABLE I: Evolution from SPEC CPU 2006 to SPEC CPU 2017. Times, in seconds, are for official results submitted to the SPEC website for an ASUS Z170MPLUS motherboard with Intel Core i7-6700K running at 4.2 GHz running 8 copies of the benchmark.

505.mcf_r: Extensive development was dedicated to produce suitable new inputs for the this benchmark. This benchmark is based on the commercial program MCF designed to schedule the movement of vehicles transitioning between the end of one route and the start of another route in a transportation system. These transitions are called deadhead routes. The objective of the program is to minimize costs and to keep the vehicle fleet size to a minimum. Löbel describes the minimum cost flow (MCF) problem and explains the data structures and the interface provided by the implementation [19]. A workload consists of specification for the number of routes and deadhead routes as well as start and end time for routes and costs for deadhead routes. Generating consistent data for mcf is rather challenging — our initial effort failed badly and led the benchmark to failed states. The workload generator for this benchmark included with the Alberta Workloads automatically generates a map for a city with various levels of density and connectivity and also uses a circadian cycle to schedule the number of buses running throughout the day. Based on this generated map the generator then creates schedules that are consistent with the constraints expected by the benchmark.

Given that the generation of workloads is completely automatic and based on a random seed and some parameters, researchers can generate as many workloads as they wish. Three new automatically generated workloads are included in the Alberta Workloads. Each workload defines a different single-depot vehicle scheduling problem.

520.omnetpp_r: A workload for this benchmark is a NEtwork Description (.ned) file in the NED language and a configuration file. The existing train and ref inputs distributed with the SPEC CPU INT 2017 suite change the amount of time that the simulation is run, but do not change the network configuration. The Alberta Workloads include seven new workloads that simulate different network topologies: line topology, ring topology, star topology, tree topology, and three random topologies with 9, 18, and 27 edges.

523.xalancbmk_r: This benchmark is a program that transforms XML data. Thus one may naively assume that new workloads can be created simply by gathering XML data files which are plentiful. However, in order to create a valid workload one also needs to provide a .xls file that describes, in a Xalan-specific language, the transformation to be done to the XML file. Few existing XML files have a corresponding .xls file. The Alberta Workloads include five new workloads from XSLT benchmarks (XSLTMark and XMark) that were based on files from the repository for XT-Speedo, an XSLT benchmarking framework.[6] For the XSLTMark benchmark, after examining the format of the XML file, we created a script to produce new random XML files with different sizes but with the same format so that they could be processed with the same .xls file. The XMark benchmark includes twenty short queries, but two of those require XSLT 2.0, which is not supported by the 523.xalancbmk_r benchmark. Thus, to create a suitable workload, we combined the remaining eighteen queries.

525.x264_r: This is a video-encoding program and one could naively think that all we need to do to generate additional workloads is to collect some videos. But reality is a bit more complex. The benchmark requires the video to be in .264 format and, in order to use a validation tool created for the benchmark, the resolution must be 1280x720. The workload must also include a control file and Truevision Graphics Adapter (TGA) versions of the interval of frames to be dumped. These versions can be generated by first running the benchmark with the --dumptga option.

Three programs are executed for each workload. First, the input video is decoded using the ldecod_r program.

[6]https://github.com/Saxonica/XT-Speedo

Next, the video is encoded again using the `x264_r` program. Finally, the output is validated using the `imagevalidate_r` program, which compares frames extracted from the video. A workload also includes several parameters such as the video frame where the encoding should start, the number of frames to be encoded, the interval between frames that are output by the benchmark, and others. The Alberta Workloads include a script to generate new workloads that, given a video input and parameters, does most of the work needed to create a new workload. This script encodes the video and creates a grayscale version of the video both in one and in two passes. The Alberta Workloads also include ten new workloads with high-definition videos that are in the public domain and were encoded to the `x264` video format using the script provided.

`531.deepsjeng_r`: This benchmark is a chess-playing and analysis engine that performs an α-β tree search. An input to `531.deepsjeng_r` is a chess position in the standard Forsyth-Edwards Notation (FEN) with optional Extended Position Description (EPD) tags and the depth to which this position should be analyzed (ply depth). The Alberta Workloads include an script to generate workloads for this benchmark through the use of 946 positions from the Arasan chess program's test suite.[7] The inputs to this script are the number of board positions to include per workload and a range from which the ply depth for each position will be randomly chosen. The Arasan chess positions can easily be replaced with a different set of chess positions by simply replacing the file read by the script. The Alberta Workloads include nine new workloads, each one containing eight chess positions with ply depths varying from 11 to 16.

`541.leela_r`: This benchmark takes as input an incomplete game of Go described as the state of the board and plays the game to the end performing a fixed number of simulations per move. The nine additional workloads provided with the Alberta Workloads are randomly selected sets of Go positions taken from the No-Name Go Server's archive.[8] Each of the new workloads contains exactly six Go positions. Moves are culled from the end of the game to ensure that the algorithm plays the game to its completion. The size of the board and number of moves culled vary between the new workloads. The Alberta Workloads also include a script that randomly selects Smart Game Format (SGF) games from the `sgf` directory and removes moves from the end of the games so that the games are incomplete. There are three board sizes to choose from and the number of moves to be removed can also be specified.

`548.exchange2_r`: This benchmark is a Sudoku puzzle generator. The input to the benchmark is a collection of valid Sudoku puzzles, each puzzle represented by 81 characters. These puzzles are used as seeds for the generation of new puzzles with identical clue patterns. The benchmark is distributed with a collection of 27 puzzles to be used as seeds. For the generation of additional workloads we attempted to use different sets of seed puzzles but that led the benchmark execution to be too short — even when the seed puzzles were rated at the highest level of difficulty by Sudoku generators. Thus the ten additional workloads provided also use the 27 seeds distributed with the benchmark. The Alberta Workloads also include a script that, given the number of puzzles to process per workload, randomly chooses from a file containing seeds. In the distribution the same seeds distributed with the SPEC CPU 2017 benchmarks are used, but a different collection of seeds can be used by replacing a file.

`557.xz_r`: This benchmark is a file compressor that uses the LZMA2 compression algorithm. The input to this benchmark is a file stored in compressed format. The execution of the benchmark encompasses decompressing the file content to memory, compressing, and then decompressing it again. For benchmarking purposes, it is a common practice to repeat the content of a file multiple times to increase the running time of the compression algorithm. The contribution of the Alberta team to this benchmark was significant because the team discovered that a memoization feature in this benchmark has non-trivial effect on the portions of the program code that are executed. The LZMA compression uses a method called "sliding-window compression" that maintains a buffer split into two parts: the dictionary and the look-ahead buffer. The encoding algorithm searches the dictionary for the longest possible match for the data currently in the look-ahead buffer. When a match is found, its length, position relative to the start of the dictionary, and following byte are written to the output stream. Thus, creating a workload by repeating the content of a file that is short enough to be captured as an entry in this dictionary greatly skews the execution from the compression portion of the algorithm to dictionary lookups. This observation led to improvements in the workloads distributed with the SPEC CPU 2017 benchmark suite. The Alberta Workloads include eight new workloads to this benchmark. These workloads were designed to provide workloads with both very compressible files and not very compressible files and also files that are smaller and larger than the dictionary.

B. Workloads for the SPEC CPU FP 2017 Suite

The Alberta Workloads provide new workloads for six out of fourteen benchmarks in the SPEC CPU FP 2017 suite.

`507.cactuBSSN_r`: This benchmark solves Einstein equations in a vacuum using the EinsteinToolkit. The generation of additional workloads consists of changing computational parameters to the solver. These parameters are provided in a file. The seven new workloads were generated following suggestions for parameter setting from the benchmark authors.

`511.povray_r`: This is a ray-tracing program. The seven new workloads aim to exercise different portions of the program by including different rendering methods and parameters. The new workloads can be organized into three categories. The `collection` workloads represent real-world uses of POV-Ray and exercise the rendering of moderately complex geometry made up of simple primitives. The `lumpy`

[7] `http://www.arasanchess.org/tests.html`
[8] `https://github.com/zenon/NNGS_SGF_Archive`

workloads render a single object placed over a checkered plane and illuminated by two spotlights. These workloads tend to put more stress on the processor's floating-point unit. The `primitive` workloads render some geometric primitives that are built into POV-Ray. These workloads emphasize rendering techniques such as reflection, refraction, and camera lens aperture.

`519.lbm_r`: This benchmark simulates incompressible fluids in 3D using the Lattice Boltzmann Method. A workload for this benchmark consists of an ASCII file describing objects that occupy the channel that the fluids flow through and some command-line arguments that specify the number of steps to be simulated and the type of simulation step used. The Alberta Workloads include twenty-four new workloads generated by varying the shape and size of the objects, the object density and the parameter for the simulation.

`521.wrf_r`: This benchmark is a numerical weather prediction system. Each workload is formed by a file generated by the original Weather Research and Forecasting (WRF) program on an input data set. These input data sets are typically captured during major weather disruption events. The workload also needs a file with parameters that specify how the weather prediction system will run. The twelve new workloads included in the Alberta Workloads were generated by running the original version of WRF (version 3.1.1) on two different data sets from the hurricane Katrina and from the typhoon Rusa obtained from the Center for Atmospheric Research website.[9] Workload's parameters — such as the micro physics, the long-wave radiation, the land surface temperature, and the boundary-level scheme — are set through command-line arguments. The Alberta Workloads include a script that assists in the generation of new workloads from a WRF input dataset. This script allows for the easy manipulation of different physics options for the weather prediction.

`526.blender_r`: This benchmark creates 3D images through rendering, modelling, and simulating of material properties. A workload consists of a `.blend` file that describes a scene to be rendered. However, not all `.blend` files are suitable workloads for the benchmark because a subset of features is supported. Also, some `.blend` files are simply resources for other `.blend` files and are not meant to be rendered. The Alberta Workloads supply a script that can be used to identify `.blend` files that work with the benchmark and a second script that can randomly select `.blend` files for use in a workload. Thirteen new workloads, from the Crazy Glue[10] and Elephants Dream[11] `.blend` files, are supplied with the Alberta Workloads. They use `.blend` files that have different maximum runtime memory, start rendering at different frames, and also vary the number of frames rendered.

`544.nab_r`: This benchmark, named the Nucleic Acid Builder, simulates forces at the molecular level. Each input to this benchmark consists of a protein-data-bank file (`pdb`), and a parameter file (`prm`) The seven new workloads model forces in seven distinct proteins. The `pdb` files, which describe the protein structure, were downloaded from the Brookhaven Protein Data Bank.[12]

[9] http://www2.mmm.ucar.edu/wrf/users/download/free_data.html

[10] `http://www.weybec.com/portfolio-item/crazyglue`

[11] `https://orange.blender.org.`

V. Characterizing Benchmark Behaviour

An important question is whether changing the workload for a benchmark will change the behaviour of the benchmark. This section presents results from three types of measurements. The first is simply the execution time of the benchmarks with each workload; the second one is the percentage of the execution time that a benchmark spends in each of its subroutines; and the third one uses Intel's top down methodology to estimate how the application is utilizing available hardware resources. The Alberta Workloads are distributed with an extensive amount of data and analysis from these measurements and also includes a study of the variation in branch prediction, cache/TLB performance, and execution time when different compilers, with different levels of optimization, are used. Only a sample of such data is included in this paper along with a summarization of the data for comparison.

For all the results reported in this Section, the benchmarks were compiled with GCC 4.8.4 at optimization level `-O3`, and the execution times were measured for each workload on machines with Intel Core i7-2600 processors at 3.4 GHz and 8 GiB of memory running Ubuntu 14.04 LTS on Linux kernel 3.16.0-76-generic. The data reported in this section was generated with a development version of the SPEC CPU benchmark suite labeled "kit 93". With the exception of `526_blender`, the benchmarks were compiled using GCC 4.8.2. at optimization level O3. The `526_blender` benchmark was compiled using llvm with optimization level O3. Details on versions of compiler and configuration machines are available in each of the individual benchmark reports distributed with the Alberta Workloads. The times reported are the mean of three executions of the benchmark with the same configuration. Three runs were selected because of the total time required to run all the variations of each benchmark for the entirety of the results reported with the Alberta Workloads. The variance observed across these three executions was minimal and do not affect the summarization results presented in this paper. The raw execution times for each run for these specific measurements are made available with the Alberta Workloads.

A. Execution Time

The reports distributed with the Alberta Workloads contain bar plots representing the mean and variance of the execution time of each workload. Due to space constraints, this paper only includes the mean of the time for three executions of each benchmark when executing with the refrate workload in the last column of Table II.

[12] `http://www.rcsb.org/pdb/home/home.do`

Benchmark	# workloads	f		b		s		r		$\mu_g(V)$	$\mu_g(M)$	refrate time (s)
		μ_g	σ_g	μ_g	σ_g	μ_g	σ_g	μ_g	σ_g			
502.gcc_r	19	23.4	1.2	33.6	1.2	11.9	1.2	29.5	1.1	5.1	25	281
505.mcf_r	7	14.1	1.8	44.9	1.3	15.3	1.6	19.8	1.2	6.9	1	324
507.cactuBSSN_r	11	20.4	1.7	42.8	1.4	0.2	1.3	31.0	1.1	17.1	1	355
510.parest_r	8	12.4	1.1	26.0	1.2	6.9	1.3	53.7	1.1	6.2	5	449
511.povray_r	10	9.4	1.7	39.7	1.5	8.8	2.2	32.7	1.4	9.2	66	535
519.lbm_r	30	1.9	1.8	61.2	1.1	0.4	3.3	34.1	1.3	27.4	59	260
520.omnetpp_r	10	9.1	1.2	64.7	1.1	8.1	1.1	17.4	1.2	6.8	17	577
521.wrf_r	16	7.1	1.4	54.9	1.1	4.3	1.3	32.2	1.0	7.8	4	904
523.xalancbmk_r	8	13.4	1.8	42.7	1.4	2.3	2.4	33.7	1.4	11.8	108	263
526.blender_r	16	17.1	1.6	25.9	1.4	11.3	1.8	41.1	1.1	6.7	44	162
531.deepsjeng_r	12	19.1	1.1	27.4	1.2	11.5	1.1	41.2	1.1	5.0	1	316
541.leela_r	12	16.9	1.1	23.0	1.1	27.6	1.1	32.2	1.0	4.3	1	484
544.nab_r	11	3.6	1.4	55.3	1.1	7.5	1.3	33.0	1.0	7.9	2	476
548.exchange2_r	13	13.9	1.0	22.4	1.0	5.1	1.1	58.6	1.0	5.9	1	920
557.xz_r	12	11.7	1.1	42.8	1.2	16.5	1.3	27.2	1.2	5.5	23	352

TABLE II: Number of workloads, geometric mean (expressed as a percentage) and geometric standard deviation of the fraction of cycles attributed to front-end bound (f), back-end bound (r), bad speculation (s), and retiring instructions (r), geometric mean of the proportional variation, and arithmetic mean of the execution time from three executions of the benchmark with the refrate workload.

B. Intel's Top Down Methodology

The Intel's Top-Down Methodology aims to show how well a microprocessor's pipeline resources are used when executing the application[16]. To this end, the methodology monitors the allocation of micro-ops and classify each cycle as:

- **Front-end Bound:** micro-ops could not be allocated because the microprocessor front-end could not supply enough micro-operations;
- **Back-end Bound:** micro-ops could not be allocated because there are not enough back-end resources available to process the pending micro-operations;
- **Bad Speculation:** micro-ops were allocated, however, they did not retire. This is mainly caused by bad speculation;
- **Retiring:** micro-ops were allocated and retired.

This information is used to guide the developer to identify the main sources of overhead in the program.

Typical graphs depicting the outcome of this methodology are shown in Figure 1. A visual inspection of these graphs indicates that changing the workloads has much more effect on the proportion of micro-ops in each category for the `523.xalancbmk_r` benchmark than for the `557.xz_r` benchmark. However, displaying and inspecting this type of graph for all benchmarks with new workloads would not be possible in this paper. Instead it is desirable to compute a single number that gives a sense of the sensibility of a benchmark program behaviour to changes in workload. For each workload i this methodology yields four ratios: f_i is the proportion of cycles that are front-end bound; b_i is the proportion of cycles that are back-end bound, s_i is the proportion of cycles due to bad speculation, and r_i is the proportion of retired cycles. To summarize the results into a single number for a given benchmark we first compute the geometric mean for each of the ratios over all the workloads for the benchmark. For instance:

$$\mu_g(f) = \sqrt[n]{\prod_{i=1}^{n} f_i} \tag{1}$$

Then we compute the geometric standard deviation for each rate over all the workloads. For example:

$$\sigma_g(f) = \exp\left(\sqrt{\frac{\sum_{i=1}^{n}\left(\ln\frac{f_i}{\mu_g(f)}\right)^2}{n}}\right) \tag{2}$$

The goal now is to capture the *proportional variation* — let's call it $V(f)$ for front-end bound — for each one of the ratios. Instead of using the coefficient of variation, which is the ratio between the standard deviation and the mean, we now compute the ratio between the geometric standard deviation and the geometric mean for each of the ratios, as illustrated in Equation 3. This choice is made because the original values are already ratios.

$$V(f) = \frac{\sigma_g(f)}{\mu_g(f)} \tag{3}$$

Finally, we compute the geometric mean of the proportional variations to obtain a single variation number for the benchmark:

$$\mu_g(V) = \sqrt[4]{V(f) \times V(b) \times V(s) \times V(r)} \tag{4}$$

The methodology described above is only intended as a means to allow a summarization for a quick comparison of overall variability of the behaviour of the benchmarks when the workload is changed. This methodology is not recommended to be used to make overarching claims about such variability. The old maxim *look into the data* still applies. Only a detail analysis of much more data than can be included

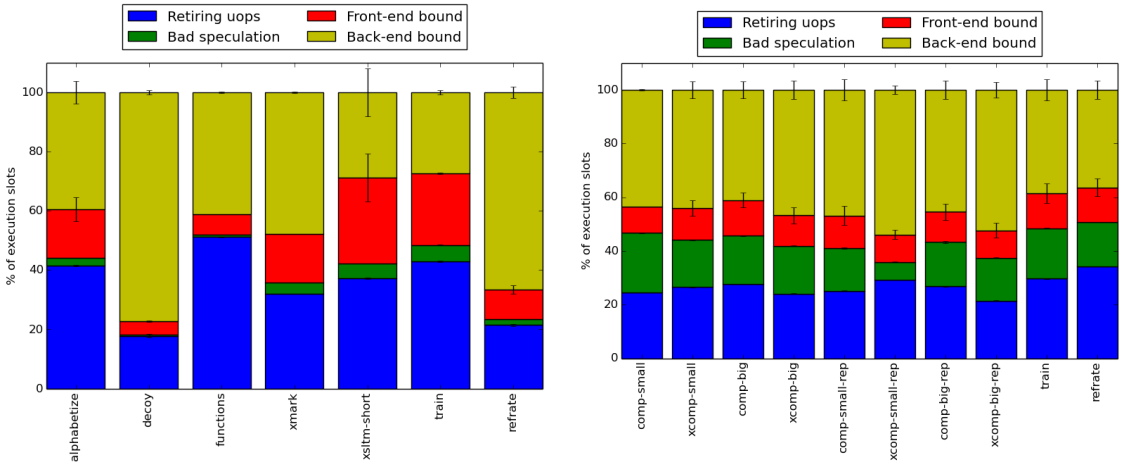

Fig. 1: Plotting of results from Intel Top-Down Methodology for the `523.xalancbmk_r` (left) and the `557.xz_r` (right) benchmarks.

in this paper can support more general claims about such variability.

Table II reports the geometric mean (μ_g) and the geometric standard deviation (σ_g) for each of the four categories of micro-ops in the Intel top-down methodology: front-end bound (f), back-end bound (b), bad speculation (s), and retiring (r). The table also reports the geometric mean of the proportional variation ($\mu_g(V)$) for each benchmark. An examination of the values of $\mu_g(V)$ for the `523.xalancbmk_r` and for the `557.xz_r` and of the plots in Figure 1 provides anecdotal indication that the value of $\mu_g(V)$ could be used as a proxy for the amount of behaviour variation between benchmarks. However, a closer look at the data also reveals shortcomings of this summarization method. A case in point is the data for `519.lbm_r`. The issue here is that this benchmark has very few micro-ops that are attributed to miss-speculation resulting in a geometric mean of only 0.4%, or 0.004. Because of these small numbers, and potentially also due to the sampling of the counters used to collect the data for the Intel top-down methodology, the geometric standard deviation for the speculative execution is quite high (3.3). This combination of low geometric mean and high geometric standard deviation for a given category leads to a high value of $\mu_g(V)$ that does not appear to reflect, in comparison with other benchmarks, the variability in the behaviour. Similar issues are observed for `507.cactuBSSN_r`.

C. Method Coverage

Another aspect of behaviour variation in a program is the amount of time spent during the execution in each method. We define *method coverage* as the percentage of execution time spent in each method. The methodology described in Section V-B is used to summarize method coverage simply by replacing the methods executed in the program for f_i, s_i, b_i and r_i. Thus, let m_j represent each method in the program and let $m_{i,j}$ stand for the fraction of time spent in method j when a benchmark is executed with workload i. Using similar relations to the ones shown in Section V-B, we can compute, for each method m_j, the values of $\mu_g(m_j)$, $\sigma_g(m_j)$, and $V(m_j)$. Then, the single number that represents the variation of behaviour in a given benchmark due to changes in the fraction of time spent in each method because of changes to workloads is given by the following geometric mean:

$$\mu_g(M) = \sqrt[n]{\prod_{j=1}^{k} V(m_j)} \tag{5}$$

where k is the number of methods executed by the benchmark. For values for $\mu_g(M)$ shown in Table II functions that account for less than 0.05% of the time for all workloads are grouped into an "others" category. To enable the computation of the geometric mean the value 0.01 is added to the time fraction of all functions that appear in the computation. Correlating the values of $\mu_g(M)$ in Table II with the bar graphs in Figure 2 provides anecdotal evidence that there is a correlation between $\mu_g(M)$ and the variance in the time spent in different methods when the workload changes. However the "*look into your data*" maxim still stands: the high value of $\mu_g(M)$ for `519.lbm_r` appears to be due to a distinct time usage in the SPEC test input in an otherwise fairly homogeneous distribution of times for this benchmark.

VI. Related Benchmark Studies

The evaluation of FDO techniques using the SPEC CPU suite that executed a single training run with the SPEC train input and a single evaluation run with the SPEC ref input led to two questions: (1) how much the reported performance depended on that particular choice of workloads for training and testing; and (2) how sensitive different benchmark programs could be to variations in workload. Berube eventually demonstrated that for some benchmarks there is a significant-enough sensitivity that published FDO performance evaluations that do not take this sensitivity into account are

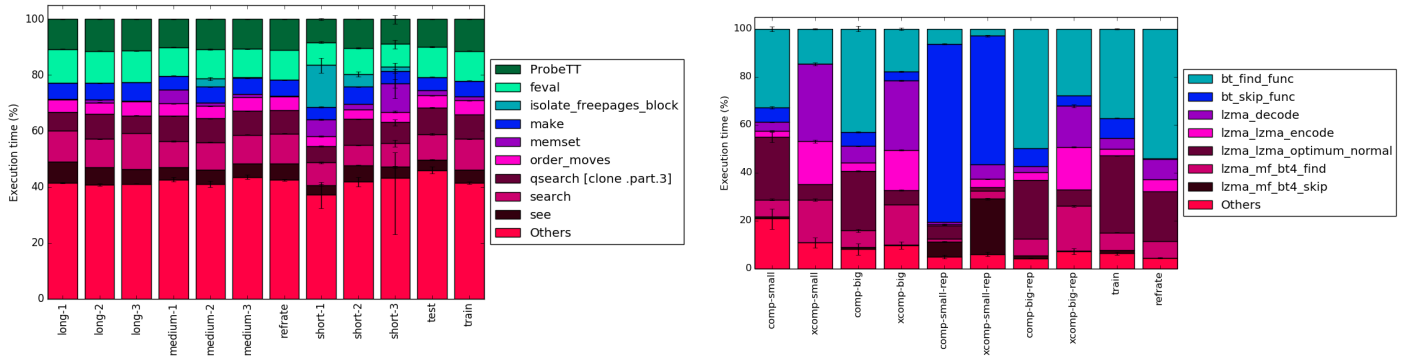

Fig. 2: Variation in function coverage with workload for the `531.deepsjeng_r` (left) and the `557.xz_r` (right) benchmarks.

meaningless [1], [3]. Berube and Amaral then proposed a methodology for more appropriate evaluation of FDO using benchmark suites [4]. That methodology requires that multiple workloads be available for each benchmark used in the evaluation — a requirement fulfilled by the Alberta Workloads. They also developed a methodology to cluster workloads for sampling in a situation where a development group has too many workloads [6]. Berube also contributed a methodology to combine the information obtained from multiple profiling executions to provide feedback to a compiler [2], [5].

Chen et al. introduce KDataSets, a suite with 1000 data sets for 32 programs, most of them from the MiBench benchmark suite [9]. The main contrast with the work presented here is that MiBench is an embedded benchmark suite with much simpler benchmark programs than the SPEC CPU 2017 suite. Using Eeckhout's Principal-Component-Analysis-based methodology [13], Chen et al. conclude that it is possible to find a near-optimal combination of compiler optimizations across the data sets. The Alberta Workloads create the possibility for further analysis to determine if that conclusion extends to the more complex programs in this suite. Another analysis that could be extended is the correlation of program-level behaviour by Jiang et al [17].

Phansalkar et al. presented a methodology to use microarchitecture-independent characteristics to measure program similarities and applied it to the SPEC CPU 2000 benchmark suite [23]. They found that the evolution of the SPEC CPU benchmarks up to SPEC CPU 2000 have increased the dynamic count of the instructions executed but that the static count remained about the same, indicating that benchmarks may execute more iterations through the same instructions. They characterize the benchmarks by their instruction and data locality, branch characteristics and instruction-level parallelism.

VII. Would-Be Nices

The availability of the Alberta Workloads, and scripts to generate new workloads, for many of the benchmarks in the SPEC CPU 2017 suite leads to many lines of questions and possible research work. These include a complete characterization of the behaviour of the benchmarks with the variation of workloads using either existing known techniques or new ones that would be suitable for this set of benchmarks and workload. It would be important to study how the many FDO techniques described in the literature perform when they are evaluated properly with a cross-validation methodology using the multiple workloads now available. Similarly, it would be nice to know if kernels created from SPEC benchmark suites to allow faster simulation studies in computer architecture actually represent the range of behaviours of the benchmarks when they are executed with multiple workloads. The likely answers will be that for some benchmarks the profiles collected by FDO would display significant variation and thus there will be much more variation in performance due to FDO than what has been reported in published papers. Similarly for the kernels used to back up claims in computer architecture. But for other types of programs, there are much smaller variations in behaviour due to changes in workloads. It would be nice to know which ones are which. It would also be nice to collect further workloads for some of the benchmarks in the suite, as well as to work on the ones from the SPEC CPU FP 2017 benchmark suite that were not included in the Alberta Workloads.

VIII. Concluding Remarks

The main goal of this publication is to report on the availability of the Alberta Workloads for the SPEC CPU 2017 Benchmark Suite in the hope that researchers start using this resource to improve the performance evaluation of techniques that rely on any type of learning. This includes formal Feedback-Directed Optimization (FDO), which should be evaluated using techniques such as cross validation, and informal parameter settings in techniques where learning is not formally described. There shall be interest also in industries that are interested in stressing their software tool stack. The original motivation for this work, many years ago, was a frustration with discussions in program committees of very high profile conferences where the argument was made that FDO studies that used a single workload for training and a single workload for testing — and for a while these two were often the same — could be excused because the researchers

did not have access to multiple workloads for the published benchmark programs. The Alberta Workloads are currently available in a University of Alberta's website[13] and have also been submitted to the SPEC Research Group so that they can be archived directly in the `spec.org` website as one of the SPEC Research Group tools.[14]

IX. Acknowledgements

The Alberta Workloads for the SPEC CPU 2017 would not exist without the strong support from, and active collaboration with, the SPEC Open Systems Group CPU working group. Many individual members contributed and supported this effort, including Cloyce Spradling, Jeff Reilly, John Henning, James Bucek, Matthew Colgrove, Alan MacKay, Michael Carroll. We also thank the SPEC Board of Directors from the approval of the supporting status to the SPEC OSG CPU group for the University of Alberta. Students that worked in this project were supported by fellowships from the Natural Sciences and Engineering Research Council (NSERC) of Canada and by the Brazilian Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) through the Science Without Borders program.

References

[1] P. Berube. Aestimo a feedback-directed optimization evaluation tool. Master's thesis, University of Alberta, Edmonton, AB, Canada, September 2005.

[2] P. Berube. *Methodologies for Many-Input Feedback-Directed Optimization*. PhD thesis, University of Alberta, Edmonton, AB, Canada, September 2012.

[3] P. Berube and J. N. Amaral. *Aestimo*: A feedback-directed optimization evaluation tool. In *Intern. Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 251–260, Austin, TX, March 2006. NSERC, (iCore), IBM.

[4] P. Berube and J. N. Amaral. Benchmark design for robust profile-directed optimization. In *Standard Performance Evaluation Corporation Benchmark Workshop*, Austin, January 2007. Received Kaivalya Dixit award, NSERC, (iCore).

[5] P. Berube and J. N. Amaral. Combined profiling: A methodology to capture varied program behavior across multiple inputs. In *Intern. Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 210–220, New Brunswick, NJ, USA, April 2012.

[6] P. Berube, J. N. Amaral, R. Ho, and R. Silvera. Workload reduction for multi-input profile-directed optimization. In *Intern. Symp. on Code Generation and Optimization (CGO)*, pages 59–69, Seattle, WA, USA, March 2009. (NSERC), AIF, iCore.

[7] M. Breughe, Z. Li, Y. Chen, S. Eyerman, O. Temam, C. Wu, and L. Eeckhout. How sensitive is processor customization to the workload's input datasets? In *Symposium on Application Specific Processors (SASP)*, pages 1–7, June 2011.

[8] D. Chen, D. X. Li, and T. Moseley. AutoFDO: Automatic feedback-directed optimization for warehouse-scale applications. In *Intern. Symp. on Code Generation and Optimization (CGO)*, pages 12–23, Barcelona, Spain, 2016.

[9] Y. Chen, Y. Huang, L. Eeckhout, G. Fursin, L. Peng, O. Temam, and C. Wu. Evaluating iterative optimization across 1000 datasets. In *Conference on Programming Language Design and Implementation (PLDI)*, pages 448–459, Toronto, ON, Canada, 2010.

[10] R. Chouhan, S. Roy, and S. Baswana. Pertinent path profiling: Tracking interactions among relevant statements. In *Intern. Symp. on Code Generation and Optimization (CGO)*, pages 1–12, Shenzen, China, 2013.

[11] Y. Ding, J. Ansel, K. Veeramachaneni, X. Shen, U.-M. O'Reilly, and S. Amarasinghe. Autotuning algorithmic choice for input sensitivity. In *Conference on Programming Language Design and Implementation (PLDI)*, pages 379–390, Portland, OR, USA, 2015.

[12] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere. Designing computer architecture research workloads. *IEEE Computer*, 36(2):65–71, February 2003.

[13] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction-Level Parallelism (JILP)*, 1(33):1–33, April 2003.

[14] I. Gartley, V. Sundaresan, M. Pirvu, and N. Grcevski. Experiences in designing a robust and scalable interpreter profiling framework. In *Intern. Symp. on Code Generation and Optimization (CGO)*, pages 1–10, Shenzen, China, 2013.

[15] Q. Guo, T. Chen, Y. Chen, and F. Franchetti. Accelerating architectural simulation via statistical techniques: A survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(3):433–446, March 2016.

[16] Intel. Tuning applications using a top-down microarchitecture analysis method. https://software.intel.com/en-us/vtune-amplifier-help-tuning-applications-using-a-top-down-microarchitecture-analysis-method. accessed October 2017.

[17] Y. Jiang, E. Zhang, K. Tian, F. Mao, M. Gethers, X. Shen, and Y. Gao. Exploiting statistical correlations for proactive prediction of program behaviors. In *Intern. Symp. on Code Generation and Optimization (CGO)*, pages 248–256, 2010.

[18] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G. Y. Wei, and D. Brooks. Profiling a warehouse-scale computer. In *International Symposium on Computer Architecture (ISCA)*, pages 158–169, June 2015.

[19] A. Löbel. MCF: a network simplex implementation, January 2000. http://www.zib.de/opt-long_projects/Software/Mcf/latest/mcf-1.2.pdf.

[20] S. Mahlke, T. Moseley, R. Hank, D. Bruening, and H. K. Cho. Instant profiling: Instrumentation sampling for profiling datacenter applications. In *Intern. Symp. on Code Generation and Optimization (CGO)*, pages 1–10, Shenzen, China, 2013.

[21] S. McCamant. Large single compilation-unit c programs. http://people.csail.mit.edu/smcc/projects/single-file-programs/.

[22] A. K. Osowski and D. J. Lilja. MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research. *Computer Architecture Letters*, 1, June 2002.

[23] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John. Measuring program similarity: Experiments with spec cpu benchmark suites. In *Intern. Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 10–20, March 2005.

[24] Z. Wang, K. Pierce, and S. McFarling. BMAT – a binary matching tool for stale profile propagation. *The Journal of Instruction-Level Parallelism*, 2, May 2000.

[25] M. Zhou, B. Wu, Y. Ding, and X. Shen. Profmig: A framework for flexible migration of program profiles across software versions. In *Intern. Symp. on Code Generation and Optimization (CGO)*, pages 1–12, Shenzen, China, 2013.

[13] `https://webdocs.cs.ualberta.ca/~amaral/AlbertaWorkloadsForSPECCPU2017/`

[14] `https://research.spec.org/tools`