Invited Paper for the Hot Workloads Special Session

# Hot Regions in SPEC CPU2017

Qinzhe Wu, Steven Flolid, Shuang Song, Junyong Deng[†], Lizy K. John
The University of Texas at Austin
[†] Xi'an University of Posts & Telecommunications
qw2699, stevenflolid, songshuang1990@utexas.edu, djy@xupt.edu.cn, ljohn@ece.utexas.edu

***Abstract*—Simulating applications and benchmarks can take hundreds of hours in full-system cycle-accurate simulators. This problem is exacerbated in many emerging applications, as they execute a large amount of dynamic instructions. For instance, in contrast to SPEC CPU2006 benchmarks, the newly released SPEC CPU2017 programs dramatically increase the total instruction count, which results in a much longer runtime. Therefore, it is of interest to analyze the program behavior to tell whether there are distinct behaviors throughout the executions or the executions are formed by the repetitions of the same behavior. Techniques that identify the repeated program behaviors (a.k.a simulation points) can narrow down the regions of interest. Researchers leverage such techniques to only simulate the regions of interest while maintaining a high simulation accuracy.**

**In this paper, we study the phase behavior of the recent SPEC CPU2017 benchmarks and provide simulation points for them using the SimPoint methodology. We find that the number of simulation points are approximately the same as that for CPU2006, even though CPU2017 has significantly higher execution time. Besides identifying SimPoints, we also study the time-varying behavior of the SPEC CPU2017 benchmarks, and observe a strong correlation between the runtime behavior and the simulation points that are invoked.**

## I. INTRODUCTION

It is widely known that during execution, the behavior of a program varies over time. Distinct program behavior phases have been observed even at large scales [1]. Knowing the phase behavior can help processor designers in multiple ways. On one hand, these phase behaviors usually reveal a program's bottlenecks (e.g., flurry of cache misses, increase of power consumption and thermal peaks, poor CPU utilization in certain phases). On the other hand, only targeting "hot" regions of the benchmark can save a considerable amount of simulation time [2][3]. It is even helpful to know the phase behavior for users who run an entire application from beginning to end. If aware of the phase behavior, they can configure the system and schedule resources accordingly to achieve a higher performance and efficiency [4] [5]. For instance, Dynamic Voltage Frequency Scaling (DVFS) can achieve a better energy efficiency in longer program phases [6] [7].

To keep the pace of emerging applications, one of the most prominent benchmark providers, SPEC (Standard Performance Evaluation Corporation), released its $6^{th}$ generation CPU benchmark suite, CPU2017 [8], in June 2017. The CPU2017 benchmark suite contains applications from many domains including Artificial Intelligence as illustrated in Table I. As the

TABLE I: Classification of SPEC CPU2017 benchmarks based on application domains.

| **App Domains** | **Benchmarks** |
|---|---|
| Compiler | *gcc*, *perlbench* |
| Compression | *x*264, *xz* |
| Artificial Intelligence | *deepsjeng*, *leela*, *exchange*2 |
| Combinational Optimization | *mcf* |
| Discrete Event Simulation | *omnetpp* |
| Document Processing | *xalancbmk* |
| Physics | *cactuBSSN*, *fotonik*3*d* |
| Fluid Dynamics | *lbm*, *bwaves* |
| Molecular Dynamics | *namd*, *nab* |
| Visualization | *povray*, *blender*, *imagick* |
| Bimedical | *parest* |
| Climatology | *wrf*, *cam*4, *pop*2, *roms* |

predecessor of CPU2017, CPU2006 benchmark suite has served as the de facto benchmark suite of single-threaded performance study the semiconductor community for more than a decade. Thus,the new SPEC CPU2017 benchmark suite has immediately received wide attention from both academia [9][10][11] and industry [12][13][14] since release.

Some of those experiments [12][13] have tested CPU2017 on state-of-the-art platforms, and some of those studies [9][10] have compared CPU2017 against CPU2006, and reported high level similarities and differences such as: the overall performance, micro-architecture challenges, redundancy, coverage, and balance of the suite. However, to the best of our knowledge, there is no experiment on the CPU2017 benchmark has been done for large scale phase behaviors.

To capture the phase behavior of the CPU2017 benchmarks, we conduct experiments sampling their performance during execution. We identify the "hot" regions in those workloads with an approach developed by preceding work [15][16]. Apart from an IPC validation, we link the region clustering results with the phase behaviors to expose the correlation between those two, which further justifies the correctness of the approach. We summarize the contributions of this paper as follows:

- We present simulation points for SPEC CPU2017 benchmark suite and share them with the computer architecture community via our github repository [17].
- We observe that even though SPEC CPU2017 has 10× dynamic instructions as CPU2006 [9] there is no significant increase in the number of simulation points.
- Our experiment results reveal that some workloads in SPEC CPU2017 have distinct phase behaviors, while others including 2 of the 3 from the artificial intelligence domain exhibit relatively stable behavior over the execution.

[*]Qinzhe and Steven made equal contributions to this work.

The rest of this paper is organized as follows. Section II provides necessary background knowledge and introduces our experiments. Section III presents the phase behavior, "hot" regions, as well as observations and analysis. We discuss related work in Section IV and conclude in Section V.

## II. Experimental Methodology

### A. Basic block vector

In the computer architecture field, software programs are sometimes split into to smaller components for study, and basic block is one of them. A basic block usually refers to a single-entrance, single-exit block of instructions that is always getting executed sequentially. In a region of dynamic instructions sliced by some analytical tools [15] [18], the same basic block could repeat several times. This is due to the basic blocks often consist of a couple of instructions, while the size of a program region is 100 million instructions for the tools' default setting ($<$ 100 million instructions requests cache-warming to guarantee the accuracy [19][20]). Counting how many times each basic block is executed during a region's execution produces a vector known as Basic Block Vector (BBV).

### B. SimPoint and PinPoint

Based on the basic block analysis, SimPoint [15] helps researchers find representative slices of instruction sequence. SimPoint first profile the program for the BBV of each region (i.e., a slice of 100 million instruction), then conducts a K-mean clustering. The clustering, where distances are calculated according to the BBV, groups similar program slices together, and picks only one slice from a cluster as the representative (a.k.a. simulation point). Although SimPoint does not take all potential factors into account, such as the instruction order and the influence from data side, SimPoint has achieved a great success since it was introduced. This success is due to its high adaptability and accuracy. A variant of SimPoint, PinPoint [18], accelerates the BBV generation by instrumenting a program's execution dynamically using the PIN profiler [21].

### C. Experiments design

The remainder of the paper will classify our experiments into two groups: clustering and and sampling. Clustering experiments profile the micro-architecture independent program information, namely the BBVs. This analysis is used to identify the "hot" regions. Sampling experiments focus on the hardware performance (e.g., IPC and cache misses) during the execution of a benchmark.

In our clustering experiments, we follow the philosophy of SimPoint, but use the Pin profiler from Intel to profile and generate the simulation points. In the sampling experiments, we use Perf [22] from the Linux toolkit to do the sampling on the time scale (i.e., every 100 milliseconds), and Tuning and Analysis Utilities (TAU) [23] to do the sampling on the instruction scale (i.e., every 100 million instructions).

### D. Workloads and platform

Since the formation in 1988, SPEC has been carefully choosing benchmarks from real world applications, and periodically releases benchmark suites for processor design evaluation. CPU2017 [8] is SPEC's latest CPU benchmark suite. Compared to the CPU2006 [24] benchmark suite, CPU2017 changes its organization. For the first time, benchmarks are categorized not only according to the data type: floating point or integer, but also according to the evaluation purpose: speed or rate. Users use the speed version to test a benchmark's execution time, while rate version aims to measure the maximum throughput of the underlying hardware. As reported in prior work [9], the speed version of some applications has much heavier workloads than its rate variation. Since multiple processes/copies of a benchmark (rate version) can easily lead to interference and the maximum throughput is not the focus of our study, this paper only evaluates the speed version of CPU2017 benchmarks.

As mentioned in Section I, there are new benchmarks and application domains introduced in the CPU2017 suite. The most remarkable enhancement is three new Artificial Intelligence (AI) applications. They are *deepsjeng* for $\alpha$-$\beta$ tree search (i.e., chess), $leela$ for Monte-Carlo tree search (i.e., go), and *exchange*2 for recursive solution generator (i.e., Sudoku). Three new benchmarks are related to image processing: *imagick* for image manipulation; *parest* for biomedical imaging; $blender$ for 3D rendering and animation. Three new benchmarks are from the climatology domain: *cam*4 for atmosphere modeling; *pop*2 and *roms* for ocean modeling. Three new benchmark are selected on behalf of scientific computing: *cactuBSSN* for solving Einstein equations, *nab* for building nucleic acid molecules, and $fotonik3d$ for computing Maxwell equations. Two new benchmarks, *x*264 and *xz* are for compression. As those new applications contribute to expanding the workload space coverage [9], they are also expected to increase the diversity of phase behavior.

The experiments are conducted on a state-of-the-art platform, Dell PowerEdge R320 server equipped Xeon E5-2430 v2 processor (codenamed Ivy Bridge), and 64 GB DDR3 memory.

## III. Hot Region and Phase Behavior Analysis

In this section, we first provide an overall discussion on the runtime performance variability as well as the information of "hot" or frequently used regions. A study pairing the phase behavior captured with "hot" regions (i.e., SimPoints) is presented to illustrate the correlation between the two.

**"Hot" regions:** Table II lists detailed information about the simulation regions picked using the SimPoint approach [15]. The Pinballs (a compressed simulation point format of PinPoint [18]) are available for download on our Github repository [17]. Some CPU2017 benchmarks are absent due to failures running with the tools [15][18]. Each region is described by a start instruction count (specifically apply to binaries compiled for the aforementioned platform) and a weight. The total numbers of "hot" regions are in the rightmost column (the left ones are the counts of top 90% "hot" regions). For the nine Speed&INT benchmarks we have investigated, the average unique region count is 19.1, and the average number of regions to cover the top 90% weight is 12.22. In comparison, those numbers for SPEC CPU2006 were 16.4, and 11.4, respectively [16][25]. Such an increase is not significant, especially considering that SPEC CPU2017 has 10$\times$ dynamic

TABLE II: “Hot” Regions of SPEC CPU2017 (100 million instruction regions selected by Multiple SimPoint tool)

| Benchmarks | Start Instructions(100 million)/Weight(%) Note: in the order of weight, and the **bold** regions count to top 90% | #90/100% |
|---|---|---|
| 600.perlbench_s | **(214681/28.85), (924424/12.60), (1277200/8.97), (1327590/7.96), (58779/7.77), (54633/5.43), (521547/5.39), (1233410/3.88), (752064/2.42), (701679/2.40), (789265/2.19), (1258340/2.06), (236975/1.78),** (48882/1.71), (482810/1.44), (467463/0.77), (478321/0.68), (29698/0.68), (199580/0.61), (607466/0.55), (236842/0.54), (1339730/0.53), (1044350/0.30), (790578/0.30), (1141890/0.23) | 13/25 |
| 602.gcc_s | **(23077/48.65), (35817/23.62), (21069/12.00), (2812/5.28), (29282/3.60),** (1353/1.36), (35911/1.21), (34118/0.87), (24369/0.84), (33794/0.82), (33733/0.69), (36933/0.44), (27933/0.36), (27185/0.17), (910/0.09) | 5/15 |
| 605.mcf_s | **(5943/21.24), (16201/12.54), (6315/10.82), (14440/10.75), (587/7.69), (6563/6.97), (13680/6.04), (7430/5.14), (14107/3.40), (12030/3.36), (12099/2.41),** (9255/2.25), (7606/2.12), (14669/2.02), (7843/0.93), (15653/0.89), (12189/0.50), (17489/0.42), (9375/0.27), (15180/0.19), (1765/0.07) | 11/21 |
| 620.omnetpp_s | **(6053/32.56), (6737/30.32), (2966/15.25), (623/9.00), (1387/6.55),** (9548/5.76), (10990/0.51), (10994/0.06) | 5/8 |
| 625.x264_s | **(49503/16.54), (22073/14.94), (3910/12.74), (42644/12.32), (10435/12.03), (46339/10.83), (26034/5.69), (41393/4.04), (11557/4.00),** (24352/3.17), (4650/1.17), (11998/1.08), (53156/0.96), (31753/0.47) | 9/14 |
| 631.deepsjeng_s | **(17909/8.84), (21851/7.87), (84/7.02), (9700/7.00), (20151/6.98), (6204/6.37), (637/5.85), (7853/5.74), (20454/5.69), (15965/5.60), (14515/5.28), (9851/5.05), (3602/4.01), (12151/3.94), (5259/3.73), (11812/2.92),** (11347/2.76), (3900/2.53), (15357/2.39), (0/0.46) | 16/20 |
| 641.leela_s | **(13198/8.63), (10150/7.15), (9272/6.85), (9870/6.75), (10635/6.57), (6871/5.64), (12252/5.49), (19208/4.29), (7341/4.17), (1830/3.90), (17685/3.67), (6004/3.59), (4934/3.45), (17328/3.37), (4952/3.10), (923/2.66), (21528/2.52), (1463/2.50), (22193/2.40), (1105/2.03), (820/1.91),** (12296/1.84), (13929/1.60), (15764/1.47), (1193/1.36), (6041/1.15), (21296/0.98), (3514/0.96), (4/0.04) | 21/29 |
| 648.exchange2_s | **(41189/9.81), (46142/9.62), (28949/9.02), (32593/7.77), (46603/7.71), (11758/7.30), (50970/7.10), (27060/6.76), (3146/6.75), (66198/4.31), (38948/3.81), (17591/3.70), (31131/3.13), (20553/3.04), (9923/2.98),** (56692/2.95), (424/1.68), (5824/1.63), (10190/0.95) | 15/19 |
| 657.xz_s | **(36567/13.00), (16895/9.39), (1400/8.92), (39442/8.76), (38731/6.97), (31742/6.59), (35941/6.33), (6913/5.80), (25357/5.25), (28962/4.09), (35764/3.86), (22587/3.21), (10870/3.02), (45973/2.94), (29995/2.88),** (39598/2.72), (45675/2.26), (40484/2.04), (38264/1.88), (18/0.07), (44672/0.02) | 15/21 |
| 603.bwaves_s | **(59291/71.46), (248007/14.61), (287430/5.89),** (313019/3.23), (315053/2.21), (95707/1.67), (318138/0.34), (263763/0.31), (303136/0.28) | 3/9 |
| 619.lbm_s | **(7134/69.69), (10742/14.92), (43825/4.43), (11740/4.36),** (6240/3.65), (9400/2.38),(8491/0.56), (0/0.01) | 4/8 |
| 638.imagick_s | **(551856/43.83), (394592/27.24), (467091/8.12), (28460/4.97), (91452/4.24), (4209/2.76),** (41027/2.50), (515828/2.45), (72623/1.27), (656007/1.21), (651005/0.78), (10504/0.54), (102890/0.04), (272722/0.04), (567401/0.04) | 6/15 |
| 644.nab_s | **(9237/34.24), (52046/22.75), (56855/17.91), (58776/9.76), (122599/7.45),** (133757/6.91),(48950/0.26), (120710/0.25), (130263/0.16), (104620/0.14), (76009/0.09), (84205/0.08) | 5/12 |
| 649.fotonik3d_s | **(19088/43.91), (2468/29.02), (56395/6.90), (1608/4.72), (11626/2.65), (53620/2.21), (17985/2.12),** (33097/1.95), (35644/1.20), (38568/0.91), (31846/0.83), (44869/0.83), (3563/0.80), (46262/0.64), (33943/0.49), (43842/0.41), (36949/0.41) | 7/17 |

TABLE III: IPC prediction error sensitivity to simulation points subsetting

| | Length (100M instructions) | | | | | Error (%) | | | |
|---|---|---|---|---|---|---|---|---|---|
| Benchmarks | Original | 100% | 90% | Top10 | Top5 | 100% | 90% | Top10 | Top5 |
| 602.gcc_s | 17817 | 14 | 5 | 10 | 5 | 0.31 | 0.30 | 0.17 | 0.30 |
| 605.mcf_s | 22460 | 20 | 11 | 10 | 5 | 1.56 | 0.77 | 8.50 | 28.37 |
| 641.leela_s | 49459 | 28 | 21 | 10 | 5 | 0.14 | 0.45 | 1.23 | 2.15 |
| 648.exchange2_s | 66589 | 18 | 15 | 10 | 5 | 0.23 | 0.02 | 0.03 | 0.17 |
| 657.xz_s | 45718 | 20 | 17 | 10 | 5 | 0.31 | 0.30 | 0.17 | 0.30 |

instruction counts as SPEC CPU2006. The average “hot” region count for the five Speed&FP benchmarks is 12.2, and 90 percentile regions count average is 5. It is worth mentioning that the “hottest” regions of the three new AI benchmarks ($631.deepsjeng\_s$, $641.leela\_s$, and $648.exchange2\_s$) take up weights no more than 10%. As opposed to programs such as $602.gcc\_s$, $619.lbm\_s$ and $638.imagick\_s$ and $649.fotonik3d\_s$ that have individual regions that represent almost half of their program’s entire execution.

As a validation, we calculate the IPC prediction errors and report them in Table III. The predicted IPC is derived weighted average of clusters’ representative IPCs. To compare with the measured IPC of the whole program, we scale the average according to the coverage percentage when a subset of simulation points is used. The absolute difference between the predicted IPC and measured IPC is then divided by the measured IPC to get the error rate. The formula is shown in Equation 1, where $SP_i$ refers to the IPC of simulation point, $W_i$ refers to the corresponding weight, and $M$ is the measured IPC.

$$Error\% = \frac{|\sum(SP_i \times W_i) - M\sum W_i|}{M\sum W_i} \tag{1}$$

Because subsetting is a common approach to reduce redundancy, it is also applied to simulation points to further shorten the simulation length. We recreate the predicted IPC from all the simulation points (columns marked with “100%”) as well as from three different subsets of simulation points: columns “90%” means simulation points are used that cover 90% of the program’s execution; columns “Top10” (“Top5”) means the 10 (5) simulation points having the most weights are used for the prediction. From the table, we can see, the IPC error rates are so low that most of the numbers are less then 2%. This means the simulation points offer an accurate estimation on performance even with hundreds of times miniaturization (as shown in the left half of Table III). Two surprising facts are seen here: 1) it is not necessary for the full set of simulation points to achieve the lowest error rate, because clusters with least weights usually have a dispersed spread and their simulation points is not that representative; 2) one more simulation point for $605.mcf\_s$ can cause the error rate to drop from 8.5% to 0.77%. It may also be observed that reasonable simulation experiments can be devised with approximately one billion instruction simulation budget.

**Variability:** The statistics in Table IV are calculated based on the measurements of executions on the aforementioned platform, using the Perf [22] tool with sampling at 100 ms intervals. Apart from IPC, level 1 data cache Mises Per Kilo-Instructions (MPKI) is studied as well, because it is said to be one of the most sensitive metrics [9]. Along with the means of

TABLE IV: Variability in SPEC CPU2017 speed benchmarks

| | IPC | | | L1 D$ MPKI | | |
|---|---|---|---|---|---|---|
| Benchmarks | mean | stdev | COV | mean | stdev | COV |
| 600.perlbench_s | 2.26 | 0.12 | 0.05 | 1.95 | 1.36 | 0.70 |
| 602.gcc_s | 1.58 | 0.26 | 0.16 | 12.91 | 7.46 | 0.58 |
| 605.mcf_s | 0.76 | 0.17 | 0.22 | 91.99 | 63.90 | 0.69 |
| 620.omnetpp_s | 0.74 | 0.06 | 0.08 | 40.83 | 2.45 | 0.06 |
| 623.xalancbmk_s | 1.14 | 0.35 | 0.31 | 55.66 | 11.79 | 0.21 |
| 625.x264_s | 2.31 | 0.11 | 0.05 | 1.18 | 0.36 | 0.31 |
| 631.deepsjeng_s | 1.60 | 0.30 | 0.19 | 9.04 | 38.84 | 4.30 |
| 641.leela_s | 1.23 | 0.05 | 0.04 | 5.26 | 1.80 | 0.34 |
| 648.exchange2_s | 2.04 | 0.02 | 0.01 | 0.01 | 0.00 | 0.00 |
| 657.xz_s | 1.46 | 0.40 | 0.27 | 17.20 | 43.36 | 2.52 |
| **Average of INT** | 1.51 | 0.18 | 0.14 | 23.60 | 17.13 | 0.97 |
| 603.bwaves_s | 2.32 | 0.59 | 0.25 | 12.89 | 21.44 | 1.66 |
| 607.cactuBSSN_s | 0.98 | 0.23 | 0.23 | 127.90 | 96.84 | 0.76 |
| 619.lbm_s | 0.90 | 0.21 | 0.23 | 150.90 | 27.39 | 0.18 |
| 621.wrf_s | 1.14 | 0.13 | 0.11 | 18.00 | 16.12 | 0.90 |
| 627.cam4_s | 1.36 | 0.08 | 0.06 | 19.01 | 4.22 | 0.22 |
| 628.pop2_s | 1.54 | 0.08 | 0.05 | 36.41 | 10.91 | 0.30 |
| 638.imagick_s | 0.87 | 0.18 | 0.21 | 10.67 | 3.89 | 0.36 |
| 644.nab_s | 1.22 | 0.09 | 0.07 | 11.73 | 4.26 | 0.36 |
| 649.fotonik3d_s | 1.46 | 0.66 | 0.45 | 59.83 | 31.64 | 0.53 |
| 654.roms_s | 1.50 | 0.45 | 0.30 | 41.27 | 27.53 | 0.67 |
| **Average of FP** | 1.33 | 0.27 | 0.20 | 48.86 | 24.42 | 0.59 |

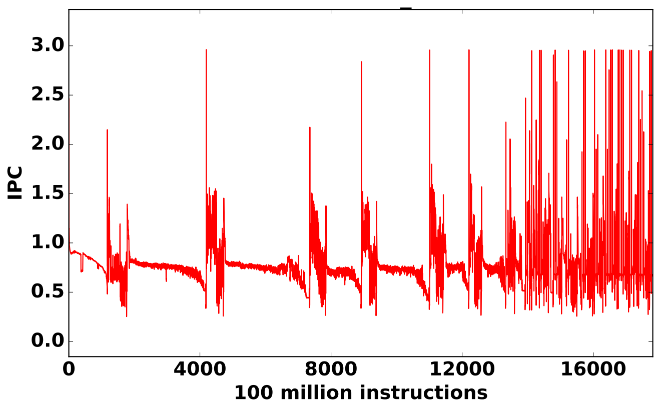
TABLE V: Level of Phase Variations in the SPEC CPU 2017 Integer and FP Programs (based on IPC). The omitted ones have medium level of variability.

| | |
|---|---|
| High (INT) | 605.mcf_s, 623.xalancbmk_s, 657.xz_s |
| Low (INT) | 600.perlbench_s, 641.leela_s, 648.exchange2_s |
| High (FP) | 603.bwaves_s, 607.cactusBSSN_s, 649.fotonik3d_s, 654.roms_s |
| Low (FP) | 627.cam4_s, 628.pop2_s |

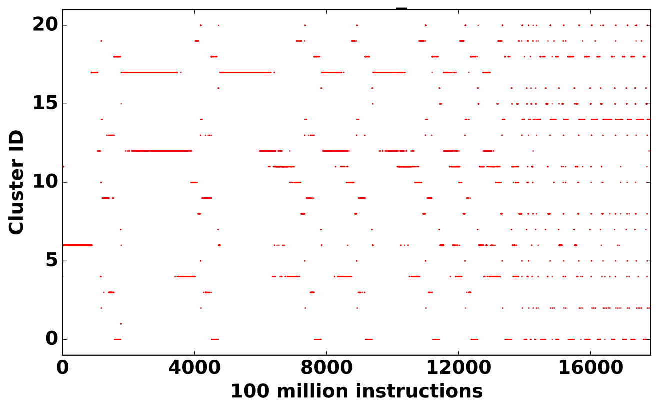
IPC and MPKI, the standard deviation (stdev) and Coefficient Of Variation (COV) are also presented. From Table IV, we observe that some benchmarks, like 605.$mcf\_s$, show great variability on both IPC and L1 data cache MPKI, while some of the new applications, for example 641.$leela\_s$, have relatively stable performance. Taking such difference into consideration, we list the benchmarks having high level variability and low level variability in Table V. It is interesting that diversity is observable among the new applications from the emerging AI domain. Two out of the three AI programs (641.$leela\_s$ and 648.$exchange2\_s$) exhibit very low variations in IPCs, while the 631.$deepsjeng\_s$ benchmark exhibits wide variations in terms of IPC and L1 data cache misses.

**Paired-analysis:** Figure 1a and 2a show the samples of IPC taken every region (i.e., 100 million instructions) during the execution of 605.$mcf\_s$ and 641.$leela\_s$, respectively. The sample points are connected in chronological order.

The curve in Figure 1a shows that the regional IPC fluctuates between 0.2 and 3.0, which is a wide range (the ratio for the maximum and minimum throughput is as high as 15×). 605.$mcf\_s$ possesses two very different patterns: The curve first features a relatively stable descent, but is then followed by an increasingly intense fluctuation. At the beginning, the stable descent dominates, interleaved by the wild fluctuation from time to time. Gradually, the ferocious fluctuation overwhelms the stable descent. The region cluster distribution shown in Figure 1b has a similar phenomenon. From Table II, we know that 605.$mcf\_s$ has 21 clusters, so the vertical axis is ranged



(a) IPC time varying graph of 605.$mcf\_s$



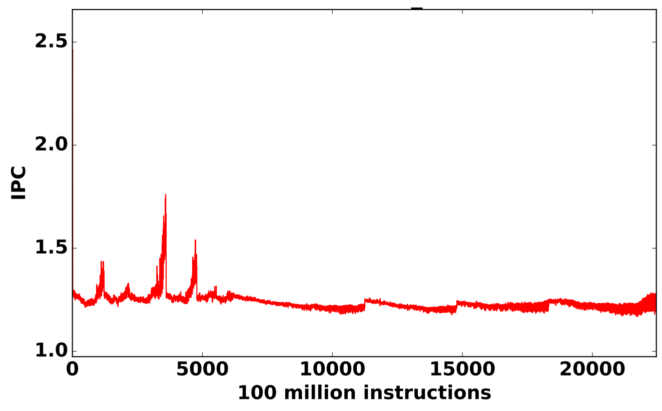
(b) Region cluster distribution of 605.$mcf\_s$

Fig. 1: Time varying graphs and region cluster distributions of 605.$mcf\_s$

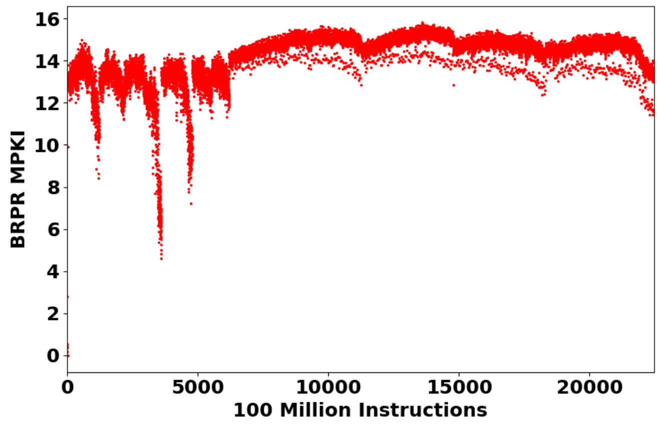
from 0 to 20 for each cluster. The regions' sample points are distributed in the Y scale according to which cluster they belong to. As the same as Figure 1a, the horizontal axis represents the execution order of those regions. For the regions during the stable descent phase, most of them are categorized into cluster 17, 12, 11 and 4. For the regions of the ferocious fluctuation phase, they spread over several clusters. Synchronized with the IPC time varying graph: at the beginning, a great percentage of regions are from cluster 17, 12, 11 or 4; when it goes to the end, regions come from many other clusters.

Similarly, 641.$leela\_s$'s time varying IPC plot, Figure 2a, and region cluster distribution chart, Figure 2c, are aligned to each other. It is worth mentioning that there are two large scale phases in the execution of 641.$leela\_s$. The first quarter of execution is characterized by several spikes in IPC, while the second phase contains slowly decreasing trends of IPC (but fairly stable). We can find a distinct change in behavior after the first quarter of execution in Figure 2c (around the instruction count 620 billion). Before this instruction count, for example, it has no region mapped to cluster 3, while cluster 3 gets executed many times after the 620 billion point.

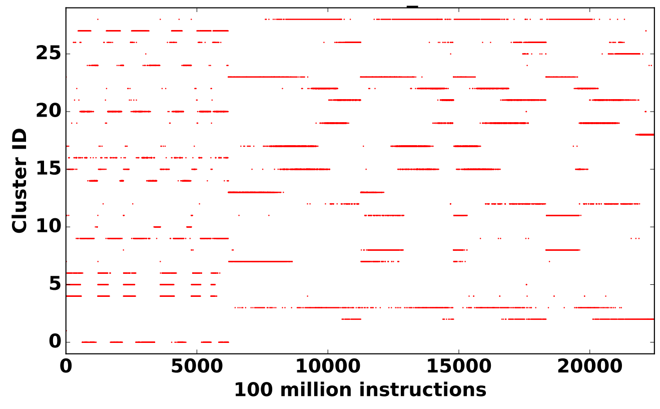
Unlike the previous two IPC plots, the IPC sample points for 649.$fotonik3d\_s$ in Figure 3a are not connected, because 649.$fotonik3d\_s$ frequently changes its behavior. If we were to add lines between continuous sample points, the plot would



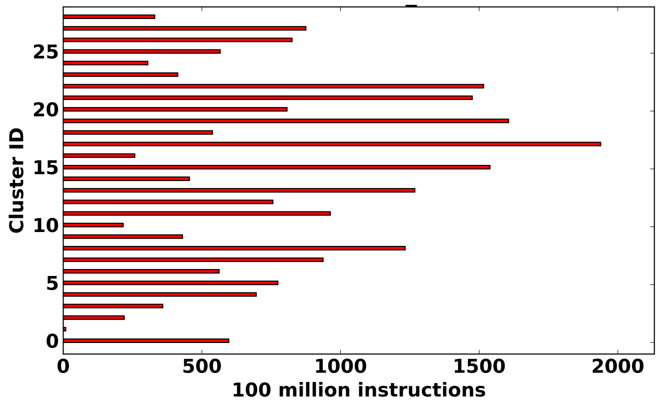
(a) IPC time varying graph of 641.*leela_s*



(b) Branch prediction MPKI time varying graph of 641.*leela_s*

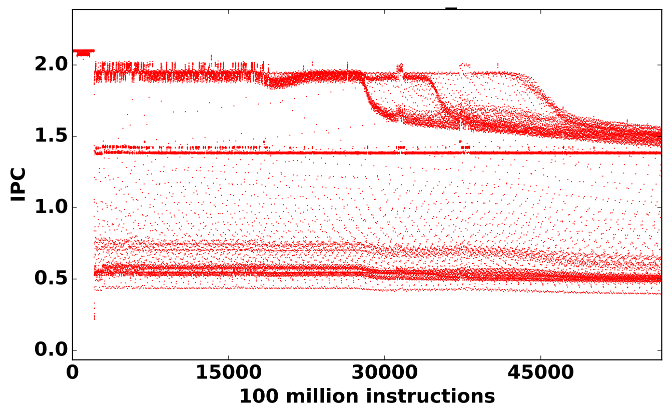


(c) Region cluster distribution of 641.*leela_s*

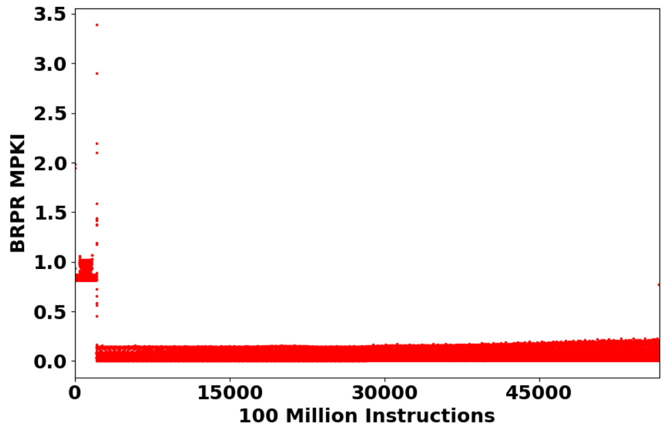


(d) Region cluster usage absolute frequency of 641.leela_s

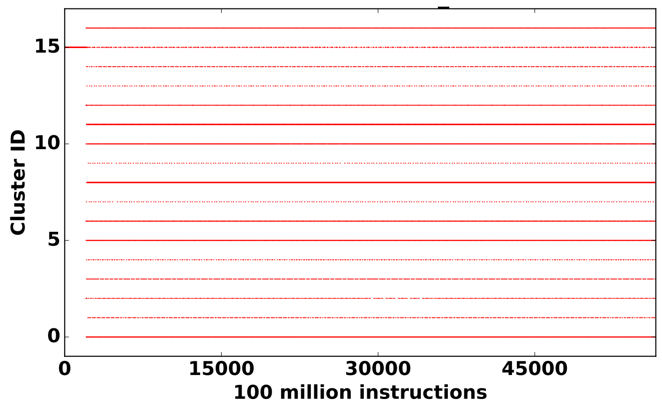
Fig. 2: Time varying graphs and region cluster distributions of 641.*leela_s*



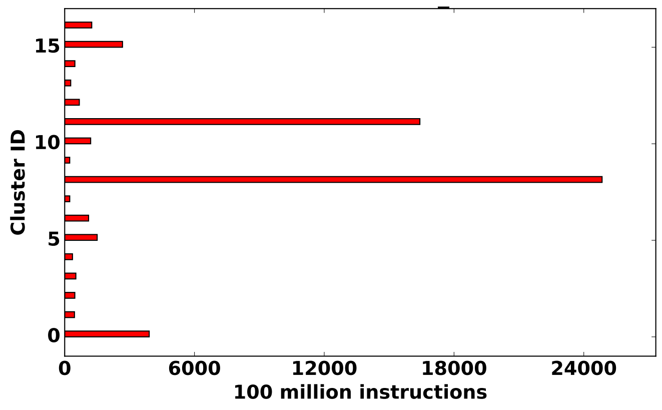
(a) IPC time varying graph of 649.*fotonik3d_s*



(b) Branch prediction MPKI time varying graph of 649.*fotonik3d_s*

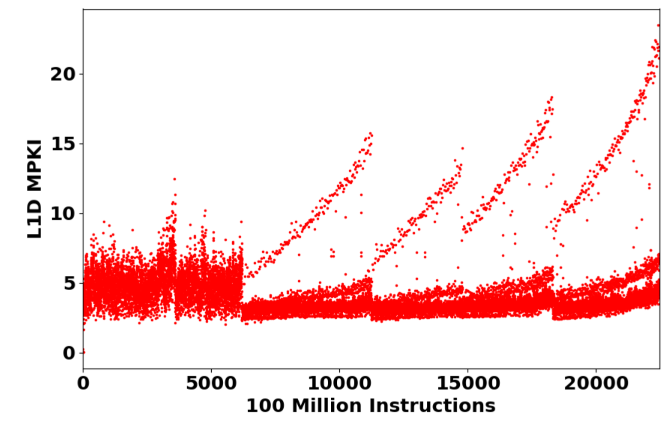


(c) Region cluster distribution of 649.*fotonik3d_s*

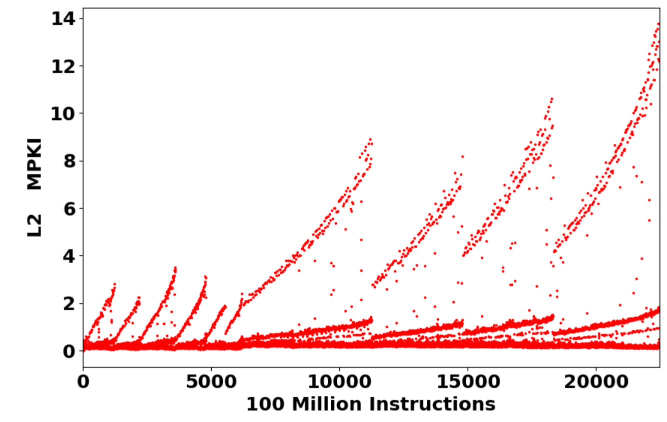


(d) Region cluster usage absolute frequency of 649.*fotonik3d_s*

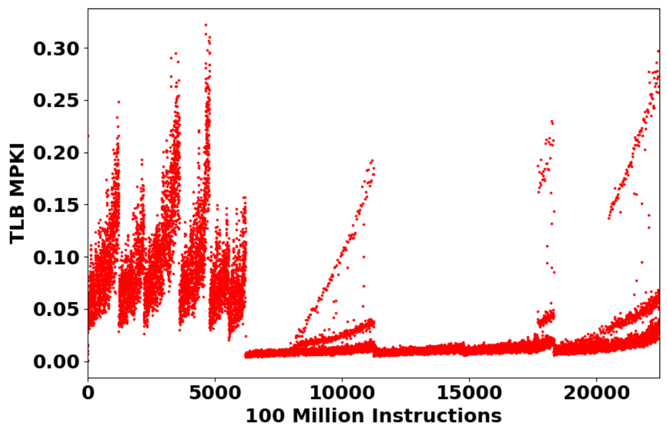
Fig. 3: Time varying graphs and region cluster distributions of 649.fotonik3d_s



(a) L1D MPKI time varying graph of 641.*leela_s*

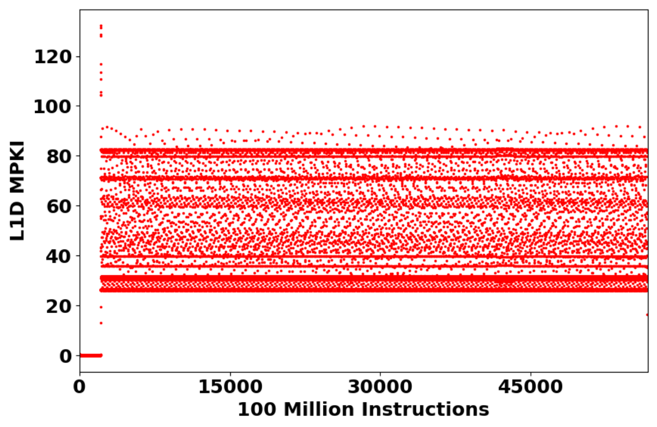


(b) L2 MPKI time varying graph of 641.*leela_s*

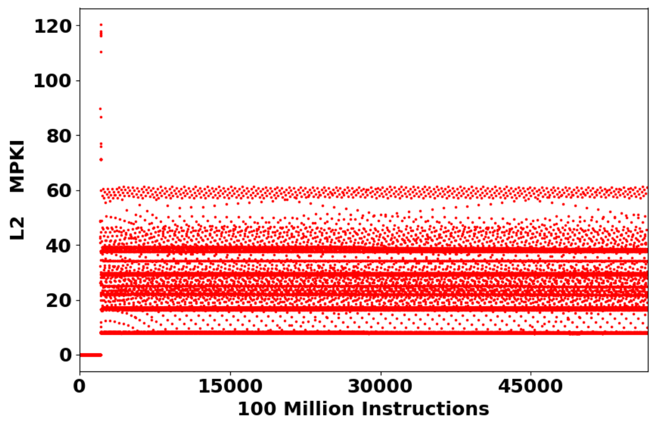


(c) TLB MPKI time varying graph of 641.*leela_s*

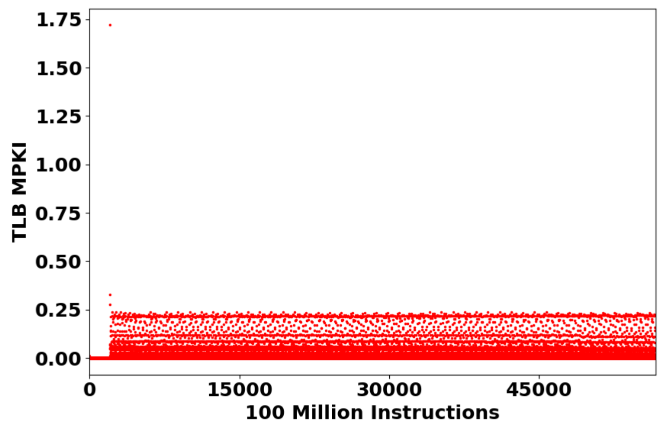
Fig. 4: Cache and TLB performance of 641.*leela_s*



(a) L1D MPKI time varying graph of 649.*fotonik3d_s*



(b) L2 MPKI time varying graph of 649.*fotonik3d_s*



(c) TLB MPKI time varying graph of 649.*fotonik3d_s*

Fig. 5: Cache and TLB performance of 649.fotonik3d_s

show nothing but a large red block. Nevertheless, the discrete points offer interesting insight about the phase behavior. As we can see, after the first 200 billion instructions (process initialization), numerous points gather around the IPC levels of 1.4, 1.95, and 0.51, creating three visible lines. This indicates that even though 649.*fotonik3d_s* changes its behavior quickly, but it does not change its behavior randomly. 649.*fotonik3d_s* mainly switches between limited "hot" regions, which give rise to the three levels of IPCs. The idea is validated by Figure 3d, where four clusters (11, 8, 0, 15) are more active than others. Clusters 8 and 11 are used with very high frequency of usage covering more than 70% of the execution. Figures 4 and 5 illustrate the L1 data cache misses, L2 cache misses and TLB misses of 641.*leela_s* and 649.*fotonik3d_s*.

Based on the frequency of usage of the SimPoint clusters, we divide CPU2017 benchmarks into three categories in Table VI. "Uneven" means those benchmarks that have a few high frequency clusters and then a few low frequency clusters, (i.e., their distribution has a peaky nature). "Even" means those benchmarks that need plenty of simulation regions, but

TABLE VI: Classification of SPEC CPU2017 benchmarks in terms of the distribution of SimPoint cluster weights.

| | |
|---|---|
| Uneven | *fotonik3d*, *gcc*, *bwaves*, *lbm*, *imagick* |
| Medium | *mcf*, *perlbench*, *omnetpp*, *nab*, *x*264 |
| Even | *leela*, *deepsjeng*, *exchange*2, *xz* |

the frequency of usage (weights) is split somewhat evenly, (i.e., their distribution is fairly flat). Other benchmarks with a distribution in between the peaky and the flat behaviors go to the "Medium" category. One observation is that all three Artificial Intelligence programs have a flat distribution.

## IV. Related Work

Sherwood et al. [1] [15] developed the SimPoint methodology to find "hot" regions based on BBV analysis of large program executions. They validated the SimPoint tool with the Alpha binaries of SPEC CPU2000 [26]. Taking advantage of the Pin tool [21] developed by Intel, and following the approach of SimPoint, PinPoint [18] no longer relied on a simulator to extract the basic block vector information, which could be really time-consuming. Finally, Carlson et al. developed a Pin-based simulator called Sniper [27], where they simulated the checkpoints for the "hot" regions of SPEC CPU2006 [24], further validating the representative nature of the "hot" regions. Nair et al. [16] compared SPEC CPU2000 and SPEC CPU2006 with focus on the simulation points.

## V. Conclusion

In this paper, we study the phase behavior of the SPEC CPU2017 benchmarks, using SimPoint approach and performance counter based sampling. The results of our experiments indicate that CPU2017 benchmarks have distinct phases with distinguishing behaviors, and multiple simulation regions are needed to represent the different behaviors of a benchmark. Our analysis pairing the IPC time varying curve with the region clustering time-line shows that those two are correlated. Therefore, we can safely reach a conclusion that SimPoint is still a useful approach to identify the representative phase behaviors of emerging benchmarks.

## VI. Acknowledgement

This research was supported in part by National Science Foundation (NSF-1745813 and NSF-1725743), and by Intel Corporation, as well as by NSF of China (61602377), and by International Collaborative Research of Shaanxi China under grant 2018KW-006. Authors would also like to acknowledge computational resources from Texas Advanced Computing Center (TACC). Any opinions, findings, conclusions or recommendations are those of the authors and not of the National Science Foundation or other sponsors.

## References

[1] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, (New York, NY, USA), pp. 45–57, ACM, 2002.

[2] M. V. Biesbrouck, B. Calder, and L. Eeckhout, "Efficient sampling startup for simpoint," *IEEE Micro*, vol. 26, pp. 32–42, July 2006.

[3] Y. Luo, *Improving Sampled Microprocessor Simulation*. PhD thesis, Univ. of Texas at Austin, Austin, Aug. 2005.

[4] J. Chen and L. K. John, "Predictive coordination of multiple on-chip resources for chip multiprocessors," in *Proceedings of the International Conference on Supercomputing*, ICS '11, (New York, NY, USA), pp. 192–201, ACM, 2011.

[5] J. Chen, L. K. John, and D. Kaseridis, "Modeling program resource demand using inherent program characteristics," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, pp. 1–12, June 2011.

[6] W. L. Bircher, M. Valluri, J. Law, and L. K. John, "Runtime identification of microprocessor energy saving opportunities," in *ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.*, pp. 275–280, Aug 2005.

[7] W. L. Bircher and L. K. John, "Analysis of dynamic power management on multi-core processors," in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ICS '08, (New York, NY, USA), pp. 327–338, ACM, 2008.

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

[9] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance horizon?," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 271–282, Feb 2018.

[10] A. Limaye and T. Adegbija, "A workload characterization of the spec cpu2017 benchmark suite," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 149–158, April 2018.

[11] J. N. Amaral, E. Borin, D. R. Ashley, C. Benedicto, E. Colp, J. H. S. Hoffmam, M. Karpoff, E. Ochoa, M. Redshaw, and R. E. Rodrigues, "The alberta workloads for the spec cpu 2017 benchmark suite," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 159–168, April 2018.

[12] "AMD EPYC$^{TM}$ SoC Consistently Sets World Records on SPEC CPU 2017 Floating Point Benchmarks." https://www.amd.com/system/files/2018-03/AMD-SoC-Sets-World-Records-SPEC-CPU-2017.pdf.

[13] "Three World Record SPEC CPU2017 Benchmark Results for 8-Processor ThinkSystem SR950." https://lenovopress.com/lp0809.pdf.

[14] "All SPEC CPU2017 Results Published by SPEC." http://spec.org/cpu2017/results/cpu2017.html.

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

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

[17] "LCA github repository for Experiments with SPEC CPU 2017: Similarity, Balance, Phase Behavior and SimPoints." https://github.com/UT-LCA/Scalability-Phase-Simpoint-of-SPEC-CPU2017/releases/tag/v1.0.

[18] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi, "Pinpointing representative portions of large intel® itanium ® programs with dynamic instrumentation," in *37th International Symposium on Microarchitecture (MICRO-37'04)*, pp. 81–92, Dec 2004.

[19] E. Perelman, G. Hamerly, and B. Calder, "Picking statistically valid and early simulation points," in *2003 12th International Conference on Parallel Architectures and Compilation Techniques*, pp. 244–255, Sept 2003.

[20] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, (New York, NY, USA), pp. 84–97, ACM, 2003.

[21] "Pin - a dynamic binary instrumentation tool." https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.

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

[23] "Tuning and analysis utilities." https://www.cs.uoregon.edu/research/tau/home.php.

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

[25] S. Song, Q. Wu, S. Flolid, J. Dean, R. Panda, J. Deng, and L. K. John, "Experiments with spec cpu 2017: Similarity, balance, phase behavior and simpoints," Tech. Rep. TR-180515-01, LCA Group, Department of Electrical and Computer Engineering, The University of Texas at Austin, May 2018.

[26] "SPEC CPU2000." https://www.spec.org/cpu2000.

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