IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 December 2023, accepted 29 December 2023, date of publication 8 January 2024,
date of current version 16 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3350646

RESEARCH ARTICLE

Improving the Representativeness of Simulation Intervals for the Cache Memory System

NICOLAS BUENO[1], FERNANDO CASTRO[1], LUIS PINUEL[1], JOSE I. GOMEZ-PEREZ[1], AND FRANCKY CATTHOOR[2,3]

[1]Department of Computer Architecture and Automation, Complutense University of Madrid, 28040 Madrid, Spain
[2]imec vzw, 3001 Leuven, Belgium
[3]Department of Electrical Engineering (ESAT), Katholieke Universiteit Leuven (KU Leuven), 3000 Leuven, Belgium

Corresponding author: Fernando Castro (fcastror@ucm.es)

This work was supported in part by the Ministerio de Ciencia, Innovación y Universidades (MCIN)/Agencia Estatal de Investigación (AEI)/10.13039/501100011033 under Grant PID2021-123041OB-I00, in part by "European Regional Development Fund (ERDF)—A way of making Europe," and in part by the Comunidad de Madrid (CM) under Grant S2018/TCS-4423.

ABSTRACT Accurate simulation techniques are indispensable to efficiently propose new memory or architectural organizations. As implementing new hardware concepts in real systems is often not feasible, cycle-accurate simulators employed together with certain benchmarks are commonly used. However, detailed simulators may take too much time to execute these programs until completion. Therefore, several techniques aimed at reducing this time are usually employed. These schemes select fragments of the source code considered as representative of the entire application's behaviour–mainly in terms of performance, but not plenty considering the behaviour of cache memory levels–and only these intervals are simulated. Our hypothesis is that the different simulation windows currently employed when evaluating microarchitectural proposals, especially those involving the last level cache (LLC), do not reproduce the overall cache behaviour during the entire execution, potentially leading to wrong conclusions on the real performance of the proposals assessed. In this work, we first demonstrate this hypothesis by evaluating different cache replacement policies using various typical simulation approaches. Consequently, we also propose a simulation strategy, based on the applications' LLC activity, which mimics the overall behaviour of the cache much closer than conventional simulation intervals. Our proposal allows a fairer comparison between cache-related approaches as it reports, on average, a number of changes in the relative order among the policies assessed –with respect to the full simulation–more than 30% lower than that of conventional strategies, maintaining the simulation time largely unchanged and without losing accuracy on performance terms, especially for memory-intensive applications.

INDEX TERMS Cache memory, computer architecture, computer simulation, hardware, memory architecture, microarchitecture.

I. INTRODUCTION

Currently, most researchers in computer architecture employ a real machine or a simulator to evaluate their proposals. However, both approaches exhibit some drawbacks.

On the one hand, native execution can effectively be employed to evaluate new architectural approaches, but at the cost of a large reduction in exploration space. Fortunately, many commercial systems include performance monitoring support to record execution events and obtain different metrics, that can be used for proposal assessment or benchmark characterization.

On the other hand, the entire execution of a benchmark in a cycle-level simulator that models the operation of a complex system (processor, multi-level memory hierarchy, interconnection network, etc.) may require unacceptable long time. Notably, in recent years, processor performance has increased significantly, and also augmented the design complexity

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

of the organizations they currently integrate, mainly multi-core, heterogeneous and specialized-hardware architectures. Moreover, memory hierarchy is one of the principal components because of its significant impact on performance, energy consumption and area occupied, so that an accurate and fast experimental evaluation by means of simulation becomes decisive. Hence, researchers leverage sampling techniques that allow to approximate the behaviour of a full application by using small sections of the program code as simulation intervals [1], [2]. However, in microarchitectural research there is great diversity in the selection of these simulation windows. Thus, many authors [3], [4], [5], [6], [7], [8] employ SimPoint [2], which defines application-specific simulation intervals, whereas other authors [9], [10], [11], [12] choose to perform an initial fast forwarding or warm up of a determined number of instructions followed by a detailed simulation of a fixed number of subsequent instructions (both processes –forwarding and detailed simulation– are not application-specific and imply the same number of instructions for all evaluated benchmarks). This diversity also exists in the simulator employed (gem5 [13] in [4], [5], [7], [11], and [14], Sniper [15] in [9], and [16], or Scarab [17] in [6] and [18], among others), the benchmarks used and the input data these applications receive (e.g., in the case of SPEC CPU suites, *reference* inputs in [3], [4], [7], [18], and [19], *test* inputs in [16] or *train* inputs in [20]). Our motivational hypothesis in this work is that the particular simulation window employed when evaluating microarchitectural proposals related to the last level cache (LLC), such as cache replacement policies, can lead to incorrect conclusions. To demonstrate this, we assess different conventional cache replacement policies using various established simulation intervals, and also simulate the entire benchmarks. The results obtained confirm that the particular simulation window employed significantly affects the relative performance of the policies evaluated. Therefore, we also propose a systematic methodology for selecting simulation intervals aimed at reporting results that reproduce the overall cache behaviour during program execution more accurately than conventional simulation strategies.

To reinforce the demonstration of our hypothesis, in this work we also employ the hardware performance monitoring counters (PMCs) available on a real ARM machine to further study the level of accuracy that SimPoint reports in reproducing the LLC behaviour. The motivation behind this combined analysis is that a key aspect in determining the simulation intervals of SimPoint is the correlation between the performance delivered in the complete execution of the benchmark and that obtained when running the selected portions of the program. Nevertheless, the suitability of these simulation intervals for approximating the LLC activity has not been studied in detail previously [21]. Our experiments reveal that, although SimPoint is appropriate for characterizing the entire application behaviour in terms of performance, it fails to properly characterize the LLC behaviour of applications.

In this work, we make the following contributions: we demonstrate that 1) following our systematic methodology for using the original SimPoint intervals in a different way –considering applications' LLC activity– leads to a fairer comparison among cache-related proposals. Also, in the case of memory-intensive programs and compared to conventional simulation strategies, 2) our approach significantly increases the degree of similarity with the full simulation in terms of both cache activity and performance, without impacting on simulation time.

The rest of the paper is organized as follows: Section II presents some background and related work. Section III details the experimental framework used. Section IV motivates and describes our proposed simulation intervals and Section V presents the results obtained. Finally, Section VI concludes.

II. BACKGROUND AND RELATED WORK

New memory technologies and organizations contribute to significantly augment the complexity in performing an accurate and efficient simulation of the memory hierarchy behaviour. Consequently, many studies have aimed to verify whether current simulation strategies are still valid for new complex memory systems. A widespread strategy consists in characterizing the workloads by selecting a specific subset of benchmarks and then simulating this set using a cycle-accurate simulator [21], [22], [23].

Next, we briefly describe how SimPoint operates and the cache replacement policies employed in this study.

A. SIMPOINT

Although it was proposed almost 20 years ago, it is still the most referenced technique for automatic off-line phase detection. In selecting the specific fragments of a program code to approximate the behaviour of each full application with a significantly reduced execution time, SimPoint first slices the program into chunks with the same number of instructions. Then, for each chunk, its Basic Block Vector (BBV) is determined, which implies to record the times every single basic block is executed inside the region. After dimensionality reduction performed by random projection, SimPoint employs the K-means algorithm to find the optimal clustering of the program regions, where a similar code is executed and consequently similar behaviour in the system (mainly in terms of performance) is expected. Finally, a single region is selected from each cluster as a representative SimPoint. Although several simulation intervals are determined for each application, only the most representative interval is typically employed in many research works.

B. CACHE REPLACEMENT POLICIES

In this work, we employ the gem5 simulator, which includes several out-of-the-box replacement policies [24]. Each one

uses its specific replacement data to determine a replacement victim on evictions [24]. Next, we briefly describe the five cache replacement algorithms that we evaluate:

- Least Recently Used (LRU): The victim is chosen based on a last touch timestamp: the older it is, the more likely its respective entry is to be victimized.
- Tree LRU: LRU variation that uses a binary tree to keep track of the temporal locality of the entries through 1-bit pointers.
- Random: In this straightforward approach, the block to be replaced is always randomly selected.
- Re-Reference Interval Prediction (RRIP): It uses a re-reference prediction value (RRPV) to determine if blocks are going to be re-used in the near future or not. The higher the RRPV value, the more distant the block is from its next access. From the original paper [19], this implementation of RRIP is also called Static RRIP (SRRIP), as it always inserts blocks with the same RRPV.
- Bimodal Re-Reference Interval Prediction (BRRIP): BRRIP [19] modifies the *insertion* of RRIP so that it inserts the majority of cache blocks with a *distant* RRIP and infrequently inserts new blocks with a *long* RRIP.

III. EXPERIMENTAL SETUP

In this section we detail the experimental environments we employed in this work (both a simulator and a real machine) as well as the benchmarks used.

A. EXPERIMENTAL ENVIRONMENTS

We motivate and evaluate our proposal by using the gem5 simulator. Moreover, some experiments were conducted on the 2-socket ARM Huawei Taishan 2280 v2 server, equipped with two 64-bit Kunpeng 920 CPUs (model 4826, 48 cores each) running at 2.6 GHz. Among all the PMCs available on our platform, we selected those able to measure the events closely related to the cache hierarchy (number of cache misses, cycles executed and instructions retired), employing the perf tool [25] to obtain PMC information. When the gem5 simulator was used, we employed the Syscall Emulation mode and the O3CPU model. Taishan configuration is roughly simulated from available specification data, with the per-core main features of the cache hierarchy shown in Table 1. Note that no prefetching technique is applied to any cache level.

B. EXPERIMENTAL WORKLOADS

For both the Taishan platform and its simulated gem5 counterpart, we employed the 20 *speed* benchmarks from SPEC CPU2017 suite [26], compiled with *gcc v6*. We leverage *train* inputs, using only one input data set per benchmark except in the cases of *perlbench* (5), *gcc* (3), *bwaves* (2), *xz* (2) and *nab* (2), where we experiment with different inputs (the particular number is expressed in parentheses right after the name of the benchmark).

TABLE 1. Cache parameters employed in the gem5 simulator.

	Type	Assoc.	Size	Latency	MSHR
DL1/IL1	Private	4	64KB	4 cycles	4
L2	Private	8	512KB	8 cycles	20
L3 (LLC)	Shared	8	1MB	37 cycles	24

IV. MOTIVATION AND PROPOSAL

To motivate our work, we first evaluate with gem5 –using the settings detailed in Table 1– the five aforementioned cache replacement policies by running the *speed* benchmarks from the SPEC CPU2017 suite under four different simulation approaches:

- Fast-forwarding of the first 1000M instructions followed by a detailed simulation of the subsequent 1000M or 2000M instructions (we refer to these simulation strategies as *ff1000* and *ff2000*, respectively).
- Simulation of 100M-instruction windows according to SimPoint (we denote this strategy as *spt*). Note that, for the sake of fairness, we employ **all** the SimPoint simulation intervals, not only that of the highest weight. Table 2 shows the specific number of simulation intervals employed for each application used. It is also worth noting that as we are using windows of enough interval size, no warmup is required, as other works pointed out [27], [28], [29].
- Full simulation. We drop intermediate results every 100 ms so that we can partly reconstruct the temporal behaviour of the application (we refer to this approach as *full*).

A. MOTIVATIONAL ANALYSIS

In this section we aim to validate our hypothesis. Recall that it states that *the chosen simulation intervals may lead to incorrect conclusions when exploring cache-related microarchitectural proposals*.

1) GENERAL BEHAVIOUR

We explore the LLC misses per 1K instructions (MPKI) and the cycles per instruction (CPI) values for the evaluated applications.

Regarding the LLC misses it is worth noting that we measure the total numbers of misses, including both data misses and instruction misses without distinction. This is based on the fact that instruction misses in the LLC are extremely rare. Recall that our simulated configuration roughly models that of the Taishan platform, which features a separated first level cache for instructions (IL1) and data (DL1), whereas the second level cache (L2) and the LLC (L3) are both shared by instructions and data, as typically occurs in commodity systems. According to our experiments, instruction misses in LLC represents less than 0.5% of the total LLC misses for the vast majority of the benchmarks assessed. The average value, using the arithmetic mean and considering all the 20 applications, is around 5.5%. If we omit the contribution of the very few outlier benchmarks exhibiting a high percentage of LLC instruction misses, this number

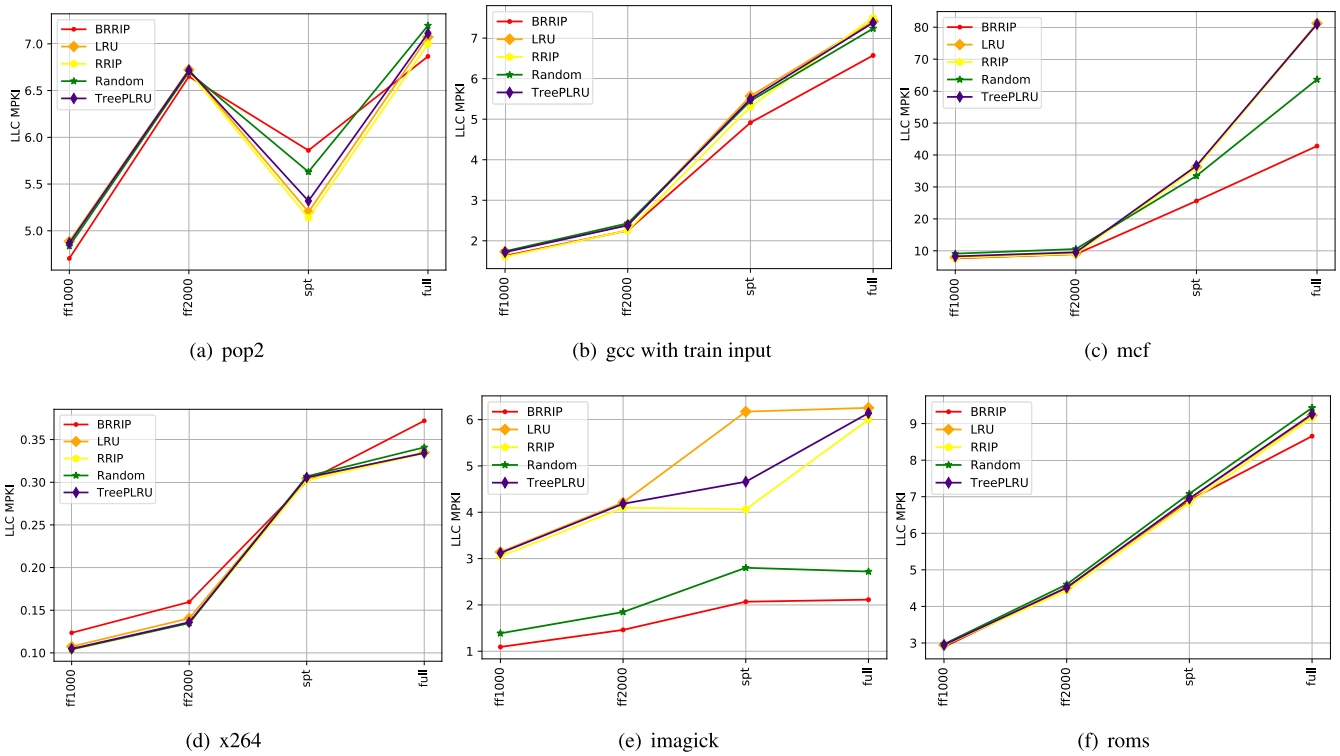


FIGURE 1. LLC MPKI obtained with the four simulation strategies, using gem5, for different benchmarks and cache replacement policies.

is just 0.8%. If we employ the geometric mean, the average value is 0.06% considering all the 20 applications and 0.03% removing the contribution of the outliers.

As a representative sample of the results obtained, Fig. 1 shows the MPKI values reported in the execution of six benchmarks under the four different simulation strategies described (the y-axis represents the absolute number of MPKI, so the scales in each figure may be different). The following conclusions can be drawn:

First, for all the applications shown, we can observe how the MPKI values vary significantly depending on the specific intervals simulated. MPKI figures obtained with the full simulation are generally substantially higher than those reported by the first two simulation windows. When using *fast-forwarding*, doubling the simulation window slightly improves the results, but they generally remain significantly far from the reference (except for *pop2*). Therefore, the results obtained when using these two first simulation strategies seems to be not representative of the application's overall behaviour, which suggests that no reliable conclusions on the cache-related proposals evaluated under these strategies can be extracted. SimPoint results are closer to the full simulation results (except for *pop2* again), but the differences, in relative terms, are also notable. In the applications shown, these variations range up to 20-55% for some cache replacement policies. In the case of *mcf* –the benchmark with the highest LLC MPKI among the 20 evaluated applications, as shown in Table 2– LRU, Tree LRU and RRIP policies all report an MPKI value with SimPoint approximately 55% lower than when using full simulation. These significant differences also occur in most of the evaluated applications. Actually,

TABLE 2. Number of SimPoint intervals per evaluated benchmark-input pairs and LLC MPKI values obtained with LRU policy.

Benchmark	Input	# SimPoints	MPKI (LRU)
perlbench	diffmail	12	0.31
	perfect	6	0.01
	scrabbl	9	0.16
	splitmail	13	0.80
	suns	15	1.65
gcc	200	19	5.01
	scilab	21	3.55
	train	7	7.25
bwaves	bwaves1	16	9.03
	bwaves2	14	8.52
mcf	train	23	81.14
cactuBSSN	train	20	6.62
lbm	train	27	41.09
omnetpp	train	25	11.43
wrf	train	25	12.52
xalancbmk	train	27	0.08
x264	train	13	0.34
cam4	train	28	4.91
pop2	train	20	7.11
deepsjeng	train	12	0.26
imagick	train	22	6.23
leela	train	16	0.38
nab	aminos	23	$3x10^{-4}$
	gcn4dna	17	0.47
exchange2	train	16	10^{-5}
fotonik3d	train	21	25.11
roms	train	22	9.21
xz	input_combined 40	16	1.91
	IMG_2560 40	16	5.82

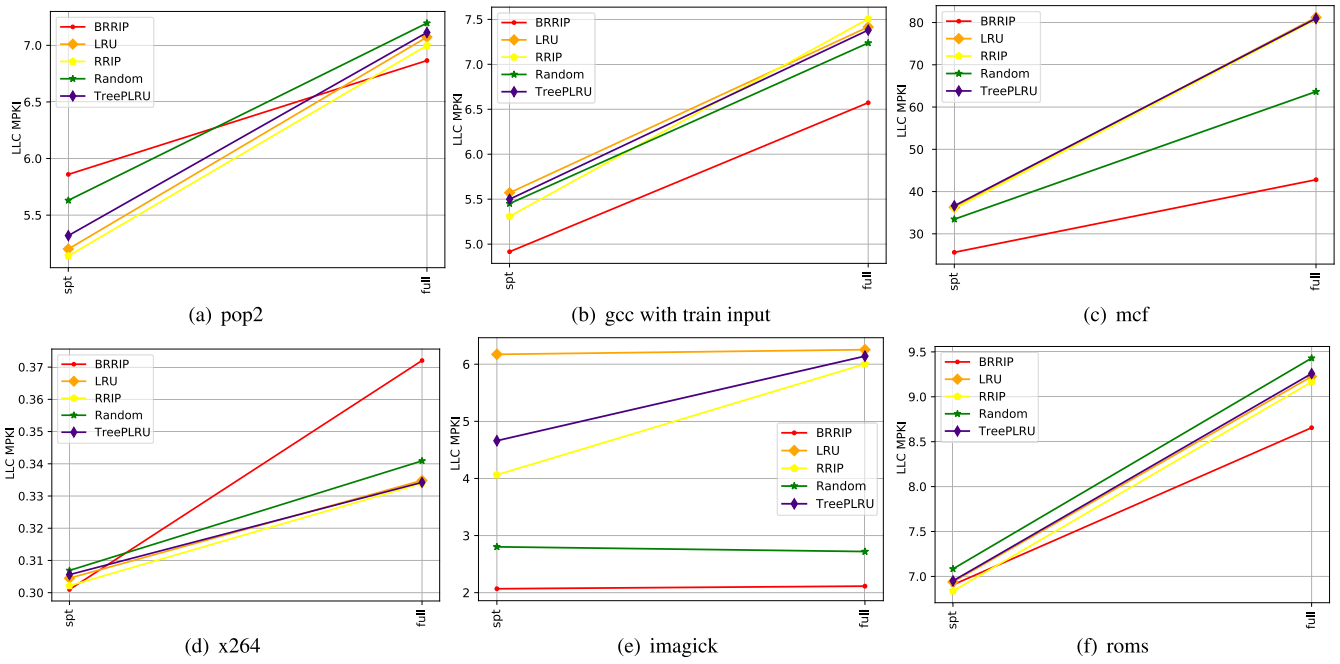


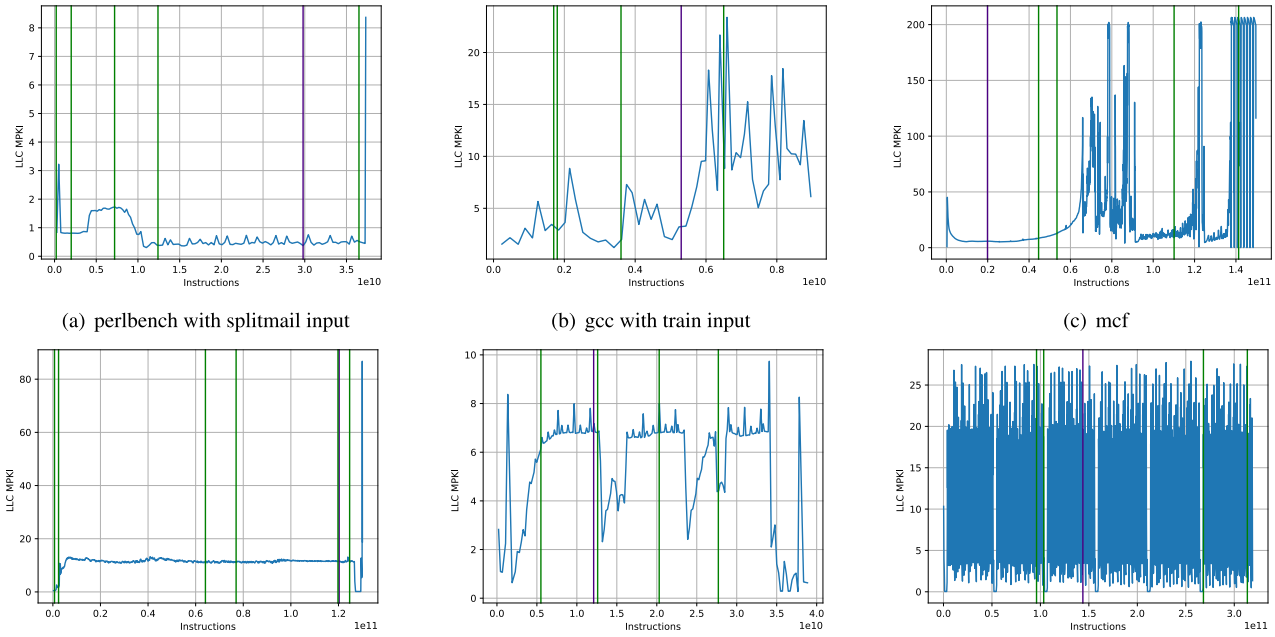
FIGURE 2. Zoom into LLC MPKI obtained with SimPoint and full simulation, using gem5, for different benchmarks and cache replacement policies.

considering the maximum difference (between MPKI values using SimPoint and the full simulation) reported by any of the replacement policies assessed for each benchmark, the average value considering all benchmarks is around 23%. Note that each benchmark contributes only one value to the final mean (we use the average value for benchmarks with more than one input). In addition, we excluded the contribution of *xalancbmk*, *exchange2*, *perlbench* (*perfect* input) and *nab* (*aminos* input), because they all report very low MPKI values (below 0.1 –we choose this threshold as it constitutes the 1% of the average MPKI obtained with the LRU policy considering the 20 benchmarks evaluated–), so that relatively small variations in absolute numbers lead to extraordinarily high variations in percentage, distorting the final result. Note that all the numbers reported in this experiment were obtained with the same simulator toolchain, so comparisons between the different approaches are fair.

> Even when considering simulator inaccuracies, we believe that these large relative differences are representative, and both *fast-forwarding* and SimPoint consistently underestimate the MPKI in the LLC, worsening the representativeness of the simulation intervals for the cache system with respect to the entire application's behaviour.

Second, and more importantly, traditional simulation techniques may lead to incorrect conclusions regarding the relative efficiency of the different replacement policies. For clarity, Fig. 2 zooms into the MPKI results reported by the corresponding SimPoint intervals and the full simulation of the same six benchmarks shown in Fig. 1. According to these data, the relative order between different policies is changed in four (*pop2*, *gcc*, *x264* and *roms*) of these six cases. Thus, for example, if we were to compare RRIP with LRU policies in *gcc*, the selected SimPoint intervals would benefit the former, whereas LRU performs better when considering the whole benchmark. This benchmark and *x264* may not be statistically significant, because the absolute values of the differences are quite small. *pop2* shows a more clear behaviour in that concern, with BRRIP clearly penalized when using SimPoint, behaviour also observed in *roms*. In the case of *mcf* and *imagick* (and other applications not shown), although no changes in the relative order between the different replacement policies is observed, we can also note significant relative variations. For example, in *mcf*, whereas using SimPoint the BRRIP approach reports an MPKI value around 27% lower than those of LRU, TreeLRU and RRIP policies, this percentage rises to 47% in the full simulation. Regarding the rest of applications evaluated, we also obtain significant changes in the relative order between the replacement policies employed depending on the simulation strategy used (in particular when comparing SimPoint and the full simulation) in all the benchmarks evaluated except *bwaves*, *cactuBSSN*, *wrf*, *deepsjeng*, *nab*, *exchange2*, *fotonik*, *xz* and *lbm*, so we can conclude that these changes occur in roughly a half of the evaluated benchmarks.

As for the CPI values, the figures obtained exhibit the same trends as the MPKI values, with significant



(a) perlbench with splitmail input (b) gcc with train input (c) mcf

(d) omnetpp (e) xz with IMG 2560 input (f) wrf

FIGURE 3. LLC MPKI values obtained with the full simulation, using gem5, for different benchmarks as instructions are executed using LRU policy.

differences (and also changes in the relative order between policies) depending on the simulated instruction window. Indeed, the average maximum difference (derived from any of the replacement policies evaluated) between the CPI values using SimPoint and the full simulation is close to 15%. In applications with high LLC activity, such as *mcf*, in which we are especially interested (MPKI variations can lead to high impact on performance), this variation is up to 40%.

> Overall, we can conclude that, in many applications, conventional simulation techniques may affect the insights derived from the evaluation of cache approaches, either because the ordering in the relative performance is changed with respect to that obtained with the full simulation, or because the relative differences among the approaches assessed are significantly far from those obtained when the entire simulation is performed.

Moreover, if for a specific metric conventional simulation strategies report values significantly far from those of the full simulation, this can also impact the accuracy on other metrics which depend on the first one, such as energy consumption on the memory hierarchy and the processor, memory endurance or CPI values, which all depend on the LLC MPKI values.

2) LLC VALUES FOR THE WHOLE EXECUTION

Previous results suggest that conventional simulation intervals do not correctly capture the cache behaviour along the entire execution of applications. To further confirm this, we measure how the MPKI value varies during the whole simulation as instructions are executed. This way, we are able to check if the simulation intervals defined by SimPoint are located in code regions with relevant LLC activity (we repeated this experiment with data from PMCs on real execution, obtaining analogous results).

Fig. 3 shows these data for six of the evaluated benchmarks, where the vertical bars show the starting points of the five most-weighted SimPoint intervals for each application (the highest one is highlighted in purple; the other four in green). For the sake of completeness, we now show the results for two of the previously evaluated benchmarks (*gcc* and *mcf*), two applications exhibiting changes in the relative order between cache replacement policies in terms of both MPKI and CPI but not previously shown (*perlbench* and *omnetpp*) and two applications that do not experiment these changes in terms of MPKI nor CPI (*xz* and *wrf*).

As illustrated, for *perlbench*, *gcc mcf* and *omnetpp* –which all exhibit irregular patterns in LLC activity–, SimPoint intervals do not capture the zones of high LLC activity. Notably, in these benchmarks, the main simulation interval (the only one employed by many authors when they apply the SimPoint technique) does not cover –with a typical simulation window of 100M instructions– any zone of the code with high LLC activity. Moreover, almost all these SimPoint intervals are located in zones with low LLC activity. This is because SimPoint relies on a static criteria based on the code's similarity for selecting the simulation intervals, and this does not necessarily imply a direct correspondence with the LLC behaviour, which is more directly related to the

phases of the code. Thus, for *xz* and *wrf*, both of which exhibit a regular LLC activity pattern with highly defined phases, SimPoint intervals do capture the parts of the code with high LLC activity. In fact, with such a regular pattern, regular sampling is likely to report satisfactory results in identifying representative zones of the code in terms of LLC activity.

B. PROPOSAL

> We just verified that the simulation intervals defined by SimPoint do not capture the zones of high LLC activity, where the replacement policy plays a more important role, so that the various approaches cannot be properly compared under SimPoint or other conventional simulation schemes commonly employed.

As a result, we now propose a systematic simulation methodology oriented to employ code fragments that are more representative of the applications' LLC activity and, therefore, to allow a more correct comparison among cache-related proposals. However, it is also needed to maintain a high level of representativeness in terms of performance. To balance both goals and not increase the simulation time required, we suggest employing the same simulation intervals defined by SimPoint, but redefining the associated weights. Essentially, we suggest sorting the intervals based on different criteria related to the LLC activity. Thus, the SimPoint interval exhibiting the highest value according to this criterion becomes the interval with the highest weight in our approach.

Consequently, aimed to derive from the evaluation of LLC-related proposals, such as cache replacement policies, the same conclusions on the relative performance among them than those of the full simulation, we have experimented with different criteria that assign more representativeness of the overall LLC behaviour to those intervals of the program execution where the LLC suffers a high level of pressure due to significant numbers of MPKI. Accordingly to this goal, we explored different criteria when sorting the simulation intervals for each benchmark, but we focused on the following two approaches as they report the most satisfactory results:

1) *mpkilru*: The average MPKI obtained in each interval when the LRU policy is employed. The weight of each interval is proportional to its LLC MPKI, so the weight for a specific simulation interval *s* within an application is calculated as follows:

$$weight_s = \frac{MPKI_{LRU,s}}{\sum_{i=1}^{n} MPKI_{LRU,i}} \quad (1)$$

where n denotes the number of simulations intervals defined by SimPoint for a particular benchmark.

2) *mpkimax*: The maximum LLC MPKI value obtained among all assessed cache replacement policies. Analogously, the weight of each simulation interval within an application is computed as follows:

$$weight_s = \frac{MPKImax_s}{\sum_{i=1}^{n} MPKImax_i} \quad (2)$$

The different steps of the described simulation methodology are recapped in Algorithm 1 for the *mpkilru* approach and in Algorithm 2 for the *mpkimax* strategy.

Algorithm 1 Configuration of Simulation Intervals (mpkilru)

```
Establish simulation intervals using SimPoint
Weight redefinition according to MPKI LRU
Require: LLC MPKI values for each SimPoint interval with LRU
Ensure: New weights for the intervals defined by SimPoint
  for All intervals do
    Determine the LLC MPKI value with LRU policy and accumulate it in sum
  end for
  for Every single interval s do
    Divide its LLC MPKI value obtained with LRU policy by sum
    Assign the previous result as new weight
  end for
```

Algorithm 2 Configuration of Simulation Intervals (mpkimax)

```
Establish simulation intervals using SimPoint
Weight redefinition according to MPKI max
Require: LLC MPKI values for each SimPoint interval for all evaluated cache replacement policies
Ensure: New weights for the intervals defined by SimPoint
  for All intervals do
    Determine the maximum LLC MPKI value among all evaluated cache replacement policies and add it in sum
  end for
  for Every single interval s do
    Divide its maximum LLC MPKI value among all policies by sum
    Assign the previous result as new weight
  end for
```

It is worth noting that our *mpkilru* and *mpkimax* approaches do not require any extra simulation time with respect to conventional SimPoint. In the case of original SimPoint, the final value of a particular metric (such as LLC MPKI) is calculated by weighting the metric values obtained in each of the simulation intervals employing the original weights defined by this simulation technique. In our proposals, we employ the same simulation intervals as conventional SimPoint, but changing the associated weights as described in (1) and (2) for *mpkilru* and *mpkimax*, respectively. In the case of *mpkilru* we need to perform the simulation with the LRU policy first, in order to compute then the final LLC MPKI values obtained with the other cache replacement policies by using for the different simulation intervals the

weights obtained from the LRU simulation. When *mpkimax* is employed, the only one restriction is that we first need to perform the simulation of all cache replacement policies evaluated, in order to obtain for each interval the maximum LLC MPKI value among all policies (new weights), and therefore to be able to compute the final LLC MPKI values for all the replacement policies. It is also noticeable that when following the original SimPoint simulation strategy using a simulator like gem5, a checkpoint of every region to be simulated is computed first. This allows to replay these regions much faster when performing an architectural exploration. According to our experiments, this time is negligible compared to the checkpoint generation itself (approximately 69 times shorter on average).

Moreover, it is also important to highlight that our approaches, as original SimPoint, entail the simulation of a number of instructions much lower than that of the full simulation. As previously illustrated in Table 2, the number of 100M-instruction simulation intervals we employ ranges between 6 and 28 depending on the benchmark evaluated, so the number of instructions we simulate varies between 600M and 2800M instructions. However, when the applications are entirely executed, the number of instructions simulated is generally significantly higher. Table 3 shows, for all the evaluated benchmark-input pairs, the total number of instructions that the full simulation entails (referred to as *full size* in the table) and the percentage that the instructions simulated with our approaches represent over the *full size* (denoted in the table as *spt vs full*). According to Table 3, considering all benchmarks we are simulating, on average, the 2.14% of the total instructions (again, each benchmark contributes only one value to the final mean –we use the average value for benchmarks with more than one input–), so we can also infer that the simulation time of our approaches is significantly lower than that of the entire simulation of the applications.

Finally, please also note that it is expected that analogous criteria to the *mpkilru* and *mpkimax* approaches proposed for cache replacement policies could be applied when other type of cache-related proposals are evaluated.

TABLE 3. Size of the full workloads in number of instructions per evaluated benchmark-input pairs and relative portion (%) of instructions selected by SimPoint and our approaches.

Benchmark	Input	Full size	spt vs full
perlbench	diffmail	$7{,}66\text{x}10^{10}$	1,57
	perfect	$1{,}24\text{x}10^{10}$	4,83
	scrabbl	$2{,}94\text{x}10^{10}$	3,06
	splitmail	$3{,}73\text{x}10^{10}$	3,48
	suns	$1{,}50\text{x}10^{9}$	99,71
gcc	200	$1{,}06\text{x}10^{11}$	1,80
	scilab	$8{,}10\text{x}10^{10}$	2,59
	train	$8{,}96\text{x}10^{9}$	7,81
bwaves	bwaves1	$7{,}36\text{x}10^{11}$	0,20
	bwaves2	$7{,}68\text{x}10^{11}$	0,18
mcf	train	$1{,}50\text{x}10^{11}$	1,54
cactuBSSN	train	$1{,}65\text{x}10^{11}$	1,21
lbm	train	$8{,}16\text{x}10^{11}$	0,33
omnetpp	train	$1{,}30\text{x}10^{11}$	1,92
wrf	train	$3{,}19\text{x}10^{11}$	0,78
xalancbmk	train	$2{,}93\text{x}10^{11}$	0,92
x264	train	$2{,}51\text{x}10^{11}$	0,52
cam4	train	$7{,}47\text{x}10^{11}$	0,38
pop2	train	$5{,}68\text{x}10^{11}$	0,35
deepsjeng	train	$3{,}51\text{x}10^{11}$	0,34
imagick	train	$2{,}94\text{x}10^{11}$	0,75
leela	train	$3{,}70\text{x}10^{11}$	0,43
nab	aminos	$6{,}80\text{x}10^{10}$	3,38
	gcn4dna	$6{,}54\text{x}10^{11}$	0,26
exchange2	train	$3{,}64\text{x}10^{11}$	0,44
fotonik3d	train	$2{,}00\text{x}10^{11}$	1,05
roms	train	$1{,}07\text{x}10^{12}$	0,21
xz	input_combined 40	$8{,}86\text{x}10^{10}$	1,81
	IMG_2560 40	$3{,}89\text{x}10^{10}$	4,12

V. EVALUATION

In this section we assess our proposals by employing two metrics: *order* and *closeness*, which are discussed next.

A. RELATIVE ORDERING OF REPLACEMENT POLICIES FROM LLC MPKI RESULTS

Our goal is to obtain the same relative order among the replacement policies used (in terms of LLC MPKI) from the simulation intervals as that observed in the full simulation. Thus, we measure how close of this goal we are by comparing the relative order experienced –in our proposals vs. the entire simulation– by each pair of employed cache replacement policies. Although we evaluate five approaches in our experiments, we do not take into account the *random* results due to their unpredictable behaviour, so we work with six pairs (combinations 1-2, 1-3, 1-4, 2-3, 2-4 and 3-4; each number identifies one of the four cache policies used).

Our proposed per-benchmark metric named *order* (initially set to zero) ranges between 0 and 6 and is computed as follows: if a specific pair maintains the same relative order under our simulation intervals as when simulating the entire benchmark, the metric remains unchanged; otherwise, the metric is incremented by one. Thus, an application that under our approach exactly matches the same relative order between the four cache replacement policies derives an *order* value of 0, and if it provides different orders between the six evaluated pairs it reports an *order* of 6. Thus, numbers close to zero indicate a high level of similarity with the full simulation.

Table 4 recaps the average *order* obtained using just the original most-weighted SimPoint (denoted as *weight*), using **all** SimPoint intervals with original weights (*spt* approach), and also using our *mpkilru* and *mpkimax* proposals, where we employ **all** the SimPoint intervals but ordered and weighted as stated in Section IV-B. We show *order* values in four different scenarios:

- Considering all the 20 evaluated benchmarks (*Avg*).
- Excluding the applications with MPKI values –under LRU policy– below 0.1 (*Avg w/o low*).
- Considering only the seven applications with the highest MPKIs (*Avg-high*). Note that we have chosen the number of benchmarks needed to obtain an accumulated

TABLE 4. Order metric for different proposals.

	weight	spt	mpkilru	mpkimax
Avg	2.57	1.14	0.79	0.86
Avg w/o low	2.47	1.11	0.73	0.81
Avg-high	3.19	1.79	1.31	1.26
Avg-changes	3.45	1.89	1.31	1.39

MPKI value –under the LRU replacement scheme– which exceeds the 80% of the total accumulated MPKI number considering all the 20 evaluated benchmarks.

- Considering only those benchmarks where the relative order among cache policies in the *spt* approach does not match the order of the full simulation (*Avg-changes*). Note that for all programs where the relative order between policies obtained with *spt* matches that of the full simulation, our two proposals manage to report the same order as well.

As illustrated, our proposals report the lowest *order* value in all the four scenarios assessed.

> Overall, *mpkilru* reduces the number of changes in the relative order among the policies by more than 30% with respect to conventional SimPoint (*spt*).

The same reduction is also achieved when considering only the benchmarks exhibiting changes in *spt* with respect to the full simulation order. In the case of memory-intensive applications, our *mpkimax* and *mpkilru* schemes report *order* values 30% and 27% lower than that of *spt*, respectively. Furthermore, we do demonstrate that using only one SimPoint with its original weight (an approach used by many authors), significantly increases the number of changes in the relative order among the policies, leading to incorrect comparisons between cache replacement schemes as previously explained in Section IV-A.

If we focus on individual applications, we may highlight benchmarks such as *x264* and *pop2*, which with *spt* exhibit *order* values of 4 and 3, respectively (therefore a relative order among the replacement policies assessed quite far from that of the full simulation), but that when using our proposals they derive *order* values of 1 and zero, respectively, so that they practically match the same behaviour as when the entire simulation is performed.

B. CLOSENESS TO ABSOLUTE FULL SIMULATION MPKI NUMBERS

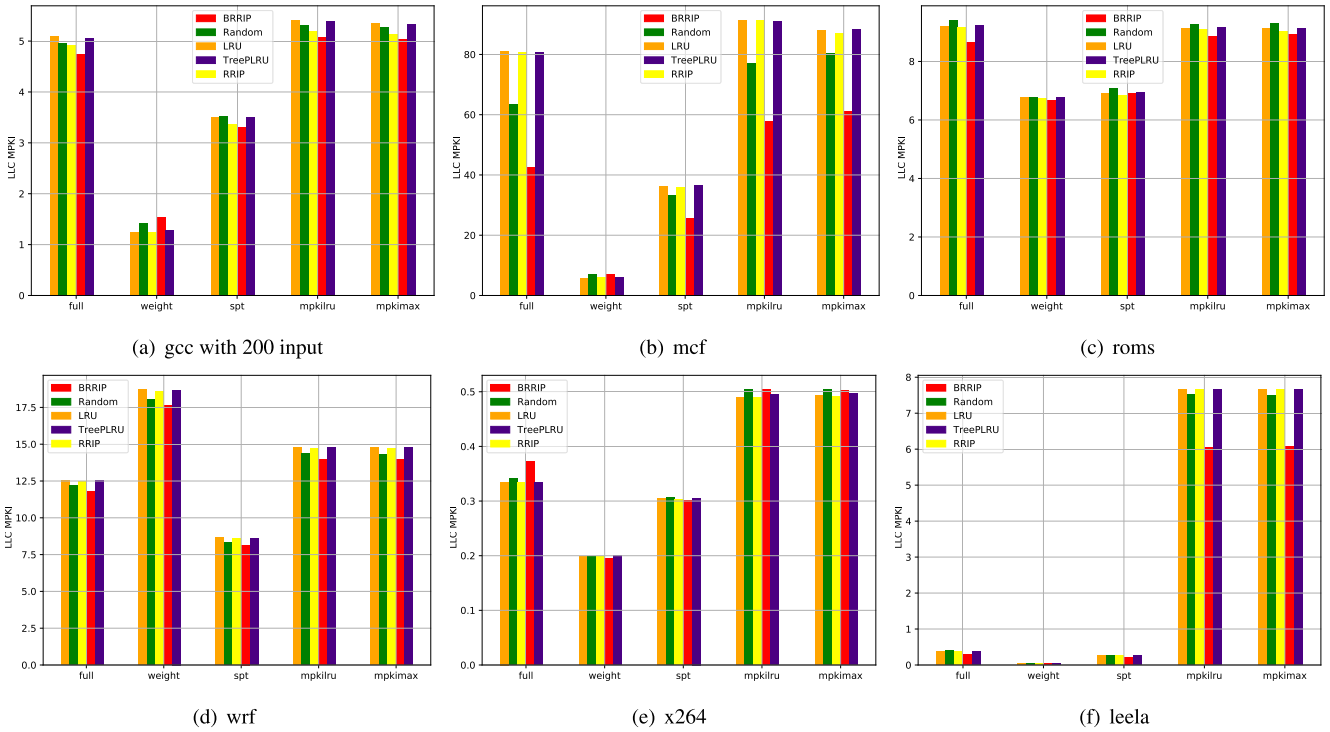
Although our proposals have been demonstrated to provide a higher level of similarity with the full simulation order than that of conventional SimPoint, we also pursue the goal of reporting MPKI values close to those of the full simulation because, as previously stated, these numbers are usually underestimated in conventional simulation schemes.

Fig. 4 illustrates –for six benchmarks as a representative sample of the results obtained– the LLC MPKI values derived under the *spt*, *weight*, *mpkilru* and *mpkimax* approaches as well as by the full simulation. We report the results for four applications also shown in the motivational study (*gcc*, *mcf*, *roms* and *x264*) and two other benchmarks (*wrf* and *leela*). For the three applications in the upper part of Fig. 4, we observe that both of our proposals report MPKI numbers much closer to those of the full simulation than the original SimPoint (both *weight* and *spt* alternatives). In the case of *wrf*, our proposals also obtain LLC MPKI values closer to those of the entire simulation than conventional SimPoint, overestimating the values of the full simulation moderately less than the *spt* underestimate them. In the *x264* benchmark, *spt* can report MPKI values closer to those of the full simulation than our proposals, but it does not capture the high MPKI value of BRRIP (compared to the other policies), which do capture both of our proposals. Finally, for the *leela* application (and also in the case of *cam4*, not shown in the graph), our proposals significantly overestimate the MPKI values, leading to numbers notably far from those of the full simulation and *spt*. To quantify the closeness of the MPKI values reported by the various simulation approaches from the values derived from the full simulation, we introduce the lower-is-better *closeness* metric. For each specific simulation strategy, this metric accumulates the percentage deviation of the LLC MPKI values obtained for all evaluated replacement policies (except *random*) from those obtained with the entire simulation. Hence, low values of *closeness* under a certain simulation approach imply that it is more accurate to reproduce MPKIs derived from the full simulation, and therefore it is also more likely to obtain the same conclusions from the evaluation of LLC-related proposals employing the specific simulation intervals as when performing the entire simulation. Accordingly, we define the metric as follows:

$$closeness(MPKI) = \sum_{i=1}^{4} \left| \frac{MPKI_{i,full} - MPKI_{i,proposal}}{MPKI_{i,full}} \right| \quad (3)$$

Table 5 shows the arithmetic and geometric means of MPKI *closeness* obtained in the *Avg w/o low* and *Avg-high* scenarios already considered for the *order* metric, as well as when considering all applications except those with MPKI values below 0.1 and the outliers *leela* and *cam4* (*Avg w/o low +2)*. We do not show results when considering all applications because the *closeness* metric in applications with very low MPKI, such as *exchange2* (on the order of 10^{-5}), reaches extraordinarily high and distorting values (higher than 5000 for all simulation strategies). This is why we also shown geometric mean values in order to mitigate the effect of the extraordinary contribution to the final arithmetic mean value of a few applications with low values of LLC MPKI. In addition, we do not report the results for the (*Avg-changes*) scenario because it only makes sense in the context of the *order* metric.

As shown, our proposals report *closeness* values significantly higher than those of original SimPoint when considering all applications except those with MPKI values



(a) gcc with 200 input (b) mcf (c) roms

(d) wrf (e) x264 (f) leela

FIGURE 4. LLC MPKI obtained with full simulation, *weight*, *spt*, *mpkilru* and *mpkimax* simulation strategies, using gem5, for different benchmarks and cache replacement policies.

TABLE 5. MPKI closeness metric (arithmetic and geometric mean) for different simulation proposals.

	Arithmetic mean				Geometric mean			
	weight	spt	mpkilru	mpkimax	weight	spt	mpkilru	mpkimax
Avg w/o low	1.85	0.91	6.65	6.62	1.18	0.58	1.28	1.28
Avg w/o low +2	1.67	0.94	1.54	1.51	1.04	0.59	0.84	0.84
Avg-high	1.36	0.87	0.68	0.66	0.59	0.51	0.32	0.33

below 0.1 (*Avg w/o low* scenario) and we employ the arithmetic mean of the values reported by the different benchmarks (this difference significantly decreases when the geometric mean is used). This is related to applications exhibiting moderately low MPKI values (much less relevant when comparing cache-related approaches such as cache replacement policies), since small variations in absolute MPKI numbers may still lead to high *closeness* numbers. This is the case of *x264* and *leela* applications shown in Fig. 4, which according to Table 2, exhibit LLC MPKI numbers with the LRU policy of just 0.34 and 0.38, respectively. Note also that for the *cam4* application, not being a benchmark exhibiting low LLC MPKI values, our approaches significantly overestimate this metric. In this case it has to do with the fact that the new weights that our strategies assign to the 28 SimPoint intervals of this program present a significant imbalance (due to the high disparity in LLC MPKI values across SimPoint intervals), much greater than in most of the remaining applications. This application has the highest number of simulation intervals of all evaluated programs, increasing the probability that some intervals capture zones of high LLC activity. Although original most-weighted simulation intervals in *cam4* do not fall in this kind of zones, there are other original low-weighted intervals that do capture these zones. Notably, the simulation interval exhibiting the second lowest weight according to original weights (lower than 0.1%) becomes the most-weighted interval in our approaches, contributing to the final LLC MPKI value with more than 30%, significantly more than any of the other intervals, reporting an LLC MPKI value just for this interval around 4X the mean value obtained with the full simulation. If we focus on the four most-weighted intervals in our *mpkilru* approach, they contribute with more than 56% to the final LLC MPKI value, while the same four intervals using their original weights contribute to the final value with just 2.2%. It is also important to recall that as our *mpkilru* and *mpkimax* approaches are assigning higher weights to those simulation intervals with high LLC activity (high LLC MPKI numbers), it was expected that our strategies overestimate MPKI numbers in some cases. However, the differences with respect to the full simulation numbers clearly decrease when we also exclude the outliers *cam4* and *leela* applications. Moreover, when we only consider the seven most memory-intensive programs, our *mpkilru* and *mpkimax* proposals manage to outperform all the other simulation strategies, reporting MPKI *closeness* values when we employ

TABLE 6. CPI closeness metric (arithmetic and geometric mean) for different simulation proposals.

	Arithmetic mean				Geometric mean			
	weight	spt	mpkilru	mpkimax	weight	spt	mpkilru	mpkimax
Avg	0.86	0.56	1.44	1.43	0.34	0.16	0.37	0.39
Avg w/o low	0.95	0.62	1.39	1.37	0.47	0.20	0.36	0.37
Avg w/o low +2	1.00	0.69	0.86	0.86	0.51	0.25	0.26	0.27
Avg-high	0.79	0.53	0.46	0.45	0.36	0.34	0.13	0.15

the arithmetic mean approximately 22% and 24% lower than that of *spt*, respectively, and around 37% and 35% respectively when the geometric mean is used. We consider this as significantly relevant, as our approaches are especially targeted to evaluate microarchitectural proposals involving the cache system, such as LLC replacement policies, which play a more important role in applications with high LLC MPKI numbers.

> Hence, for memory-intensive applications and compared to SimPoint, we significantly improve the representativeness of our simulation intervals –in terms of LLC MPKI numbers– with respect to the full simulation,

as illustrated in Fig. 4 for *mcf*, *roms*, *gcc* and also *wrf*, all of them applications with high LLC MPKI numbers.

C. CLOSENESS TO ABSOLUTE FULL SIMULATION CPI NUMBERS

Although we have demonstrated that our proposed simulation strategy outperforms the conventional SimPoint in terms of the *order* metric in all scenarios evaluated, as well as in terms of MPKI *closeness* in the case of memory-intensive programs, we must also explore how our proposals work in terms of performance if these approaches aim to postulate as an alternative to conventional simulation schemes. For this purpose, we also introduce the CPI *closeness* metric, which is defined –analogously to the case of MPKI– as follows:

$$closeness(CPI) = \sum_{i=1}^{4} \left| \frac{CPI_{i,full} - CPI_{i,proposal}}{CPI_{i,full}} \right| \quad (4)$$

Table 6 recaps the geometric and arithmetic means of the *closeness* obtained for CPI values in the same scenarios as in the case of the LLC MPKI *closeness*, and also when considering all the benchmarks evaluated. As expected, *spt* reports the CPI values closest to those of the full simulation when considering all applications. In the same scenario, when we employ the geometric mean, our *mpkilru* and *mpkimax* approaches are able to report CPI values significantly close to those of the *weight* approach, although still moderately far from *spt*. This could be considered as expectable, as we are mainly focusing on LLC activity to assign weights to SimPoint intervals and it is important to note that original SimPoint defines simulation intervals and the corresponding weights aimed to reproduce the overall behaviour of applications mainly in terms of performance. However, the differences between *spt* and our proposals decrease when we do not take into account the programs with low LLC activity, until the point where our proposals practically match (especially when considering the geometric mean) the performance numbers reported by *spt* when we exclude the applications with LLC MPKI values below 0.1 and also the outliers *leela* and *cam4* (*Avg w/o low +2)* scenario. More importantly, even when we just consider the most memory-intensive applications, our proposals manage to report performance *closeness* values around 13-15% lower than that of SimPoint when the arithmetic mean is used and around a significant 56-62% when we employ the geometric mean, with CPI numbers significantly close to those of the full simulation. In this way, with our redefinition of the weights associated to the simulation intervals defined by original SimPoint, we effectively achieve a satisfactory trade-off in reproducing the overall behaviour of applications in terms of LLC activity and also performance. As a result, for programs with high LLC MPKI numbers we significantly outperform *spt* in terms of MPKI *closeness* (as expected and previously shown) but also in terms of CPI *closeness*, despite of being original SimPoint a technique mainly conceived to match performance numbers of full execution. This also reveals that the impact of an accurate determination of LLC MPKI on other metrics such as CPI is significantly relevant for memory-intensive programs, where the LLC activity is high, so that original SimPoint is, generally, increasingly less accurate on CPI values as we progressively consider only more memory-intensive applications (see *spt* column from top to bottom in Table 6 in the case of the geometric mean) whereas our approaches follow exactly the opposite trend.

> We can conclude that for memory-intensive benchmarks, our simulation intervals obtain, also in terms of performance numbers, a higher level of representativeness of the entire simulation than original SimPoint.

VI. CONCLUSION

In this paper, we first demonstrated our hypothesis regarding the evaluation of microarchitectural cache-related proposals: the particular simulation window employed can lead to incorrect conclusions. As a motivational case study, we explored the impact of different commonly used simulation windows on the performance of various replacement policies implemented in the LLC. This analysis made it possible to infer that current simulation strategies do not fully capture the behaviour of the LLC; therefore the specific simulation window employed may entail wrongful comparisons.

Consequently, we also proposed a different simulation strategy oriented to maintain a proper trade-off in reproducing the overall behaviour of applications in terms of both LLC activity and performance, without affecting the simulation time. For this purpose, we suggested employing the same simulation intervals as SimPoint, but ordered and weighted according to different metrics that take into account the number of LLC misses, aimed to improve the representativeness of the simulation windows for the cache system.

Our experimental evaluation demonstrated that our approaches outperform conventional SimPoint in terms of the *order* metric (up to 30%) in all scenarios evaluated, and, in the case of memory-intensive programs, also in terms of MPKI and CPI *closeness* (up to 24 and 15%, respectively). Overall, we can conclude that our simulation strategies report a satisfactory trade-off in reproducing the overall behaviour of the applications in terms of both LLC activity and performance, particularly in the case of memory-intensive benchmarks, which also makes it possible a more accurate simulation in terms of other features at the whole processor level which depend on the mentioned metrics, such as energy consumption or memory endurance.

REFERENCES

[1] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "SimFlex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, Jul. 2006.

[2] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 318–319, Jun. 2003.

[3] I. Shah, A. Jain, and C. Lin, "Effective mimicry of Belady's MIN policy," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 558–572.

[4] M. Jalili and M. Erez, "Reducing load latency with cache level prediction," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 648–661.

[5] V. Baoni, A. Mittal, and G. S. Sohi, "Fat loads: Exploiting locality amongst contemporaneous load operations to optimize cache accesses," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 366–379.

[6] A. Deshmukh and Y. N. Patt, "Criticality driven fetch," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 380–391.

[7] A. Perais, "Leveraging targeted value prediction to unlock new hardware strength reduction potential," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 792–803.

[8] M. Chaudhuri, "Zero inclusion victim: Isolating core caches from inclusive last-level cache evictions," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 71–84.

[9] L. Pentecost, A. Hankin, M. Donato, M. Hempstead, G.-Y. Wei, and D. Brooks, "NVMExplorer: A framework for cross-stack comparisons of embedded non-volatile memories," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 938–956.

[10] A. Ros and S. Kaxiras, "Speculative enforcement of store atomicity," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 555–567.

[11] J. M. Cebrian, S. Kaxiras, and A. Ros, "Boosting store buffer efficiency with store-prefetch bursts," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 568–580.

[12] S. Ainsworth and T. M. Jones, "MuonTrap: Preventing cross-domain spectre-like attacks by capturing speculative state," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 132–144.

[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, and J. Hestness, "The gem5 simulator," *ACM Comput. Archit. News*, vol. 39, no. 2, p. 1, 2011.

[14] S. Yadalam, N. Shah, X. Yu, and M. Swift, "ASAP: A speculative approach to persistence," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 892–907.

[15] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 3, pp. 1–25, Aug. 2014.

[16] Y. Xu, C. Ye, X. Shen, and Y. Solihin, "Temporal exposure reduction protection for persistent memory," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 908–924.

[17] *Scarab Simulator*. Accessed: Dec. 16, 2023. [Online]. Available: https://github.com/hpsresearchgroup/scarab

[18] S. Pruett and Y. Patt, "Branch runahead: An alternative to branch prediction for impossible to predict branches," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 804–815.

[19] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, Jun. 2010, pp. 60–71.

[20] R. Rodríguez-Rodríguez, F. Castro, D. Chaver, R. Gonzalez-Alberquilla, L. Piñuel, and F. Tirado, "Write-aware replacement policies for PCM-based systems," *Comput. J.*, vol. 58, no. 9, pp. 2000–2025, Sep. 2015.

[21] A. Navarro-Torres, J. Alastruey-Benedé, P. Ibáñez-Marín, and V. Viñals-Yúfera, "Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP," *PLoS ONE*, vol. 14, no. 8, Aug. 2019, Art. no. e0220135.

[22] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did SPEC CPU 2017 broaden the performance horizon?" in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 271–282.

[23] A. Limaye and T. Adegbija, "A workload characterization of the SPEC CPU2017 benchmark suite," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2018, pp. 149–158.

[24] *Cache Replacement Policies in Gem5*. Accessed: Dec. 16, 2023. [Online]. Available: https://www.gem5.org/documentation/general_docs/memory_system/replacement_policies/

[25] *Perf Wiki Tutorial on Perf*. Accessed: Dec. 16, 2023. [Online]. Available: https://perf.wiki.kernel.org/index.php

[26] *SPEC CPU 2017*. Accessed: Dec. 16, 2023. [Online]. Available: https://www.spec.org/cpu2017/

[27] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0: Faster and more flexible program phase analysis," *J. Instruct. Level Parallelism*, vol. 7, no. 4, pp. 1–28, 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:11937761

[28] 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 *Proc. 37th Int. Symp. Microarchitecture*, 2004, pp. 81–92.

[29] G. Hamerly, E. Perelman, and B. Calder, "How to use SimPoint to pick simulation points," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, pp. 25–30, Mar. 2004, doi: 10.1145/1054907.1054913.

NICOLAS BUENO received the degree in computer engineering and the M.Sc. degree in Internet of Things from the Complutense University of Madrid, in 2017, where he is currently pursuing the Ph.D. degree in computer engineering.

His research interest includes accurate and fast memory system simulation.

FERNANDO CASTRO received the M.S. degree in physics from the University of Santiago de Compostela, Spain, in 2000, and the M.S. degree in electrical engineering and the Ph.D. degree in computer science from the Complutense University of Madrid (UCM), Spain, in 2004 and 2008, respectively.

He is currently an Associate Professor with the Department of Computer Architecture and Automation, UCM. Since 2003, he continuously participated in competitive projects related to the computer architecture field. He is the author of more than 35 international articles, including publications in international journals with JCR impact factor, such as IEEE TRANSACTIONS ON COMPUTERS or *Journal of Parallel and Distributed Computing*, and the proceedings of very recognized prestige conferences, such as IEEE/ACM MICRO or ACM/IEEE ISLPED. His research interests include energy-aware processor design, efficient memory management (including emerging non-volatile memory technologies), and OS scheduling on heterogeneous multiprocessors.

LUIS PINUEL received the M.Sc. and Ph.D. degrees in computer science from the Complutense University of Madrid (UCM), Spain, in 1996 and 2003, respectively.

From June 2010 to April 2015, he was an Academic Secretary with the Faculty of Physics, UCM. Previously, he was a Research Assistant with the Acoustic Institute, Spanish National Research Council (CSIC). He is currently an Associate Professor with the Department of Computer Architecture and Systems Engineering, UCM. His research interests include computer architecture, high-performance computing, low-power microarchitectures, embedded systems, and resource management for emerging computing systems. In these fields, he is the coauthor of more than 70 publications in prestigious journals and international conferences, and several book chapters. He has advised or co-advised five Ph.D. dissertations. He is a member of the technical program and organization committee of some relevant conferences, such as HPCA. Worried about improving knowledge transfer between research institutions and industry, he has directed more than 15 research contracts with different companies, such as Texas Instruments, Indra, and Satlink.

JOSE I. GOMEZ-PEREZ received the M.Sc. degree in computer science and the Ph.D. degree from the Complutense University of Madrid (UCM), Spain, in 2001 and 2007, respectively.

During the Ph.D. degree, he was a Visiting Researcher with imec, Leuven, working on optimizing low-power embedded systems, at both the compiler and system levels. After the Ph.D. degree, he moved to GPGPU computing, still at the compiler level. He is currently an Associate Professor with the Department of Computer Architecture and Automation, UCM, within the ArTeCS Group. His current research interest includes low-power embedded systems, focusing on the architectural impact of new resistive memory technologies, in the IoT ecosystems.

FRANCKY CATTHOOR received the Ph.D. degree in EE from Katholieke Universiteit Leuven (KU Leuven), Belgium, in 1987.

From 1987 to 2000, he headed several research domains in the area of synthesis techniques and architectural methodologies. Since 2000, he has been strongly involved in other activities with imec, Leuven, Belgium, including co-exploration of applications, computer architecture and deep submicron technology aspects, biomedical systems and IoT sensor nodes, and photo-voltaic modules combined with renewable energy systems. He is currently a Senior Fellow with imec. He is also a part-time Full Professor with the Department of Electrical Engineering (ESAT), KU Leuven.

Dr. Catthoor has been an associate editor of several IEEE and ACM journals.