Benchmarking for Single Feature Attribution with Microarchitecture Cliffs

Hao Zhen*
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
China

Qingxuan Kang*
National University of Singapore
Singapore

Yungang Bao
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
China

Trevor E. Carlson
National University of Singapore
Singapore

Abstract

Architectural simulators play a critical role in early microarchitectural exploration due to their flexibility and high productivity. However, their effectiveness is often constrained by fidelity: simulators may deviate from the behavior of the final RTL, leading to unreliable performance estimates. Consequently, model calibration, which aligns simulator behavior with the RTL as the ground-truth microarchitecture, becomes essential for achieving accurate performance modeling.

To facilitate model calibration accuracy, we propose *Microarchitecture Cliffs*, a benchmark generation methodology designed to expose mismatches in microarchitectural behavior between the simulator and RTL. After identifying the key architectural components that require calibration, the Cliff methodology enables precise attribution of microarchitectural differences to a single microarchitectural feature through a set of benchmarks. In addition, we develop a set of automated tools to improve the efficiency of the Cliff workflow.

We apply the Cliff methodology to calibrate the XiangShan version of gem5 (XS-GEM5) against the XiangShan open-source CPU (XS-RTL). We reduce the performance error of XS-GEM5 from 59.2% to just 1.4% on the Cliff benchmarks. Meanwhile, the calibration guided by Cliffs effectively reduces the relative error of a representative tightly coupled microarchitectural feature by 48.03%. It also substantially lowers the absolute performance error, with reductions of 15.1% and 21.0% on SPECint2017 and SPECfp2017, respectively.

1 Introduction

Architectural simulators are widely used for performance exploration during chip development due to their flexibility for design modification, fast build times, and efficient execution [7, 48]. In practice, new microarchitectural features are typically implemented and evaluated on simulators first, then validated on the RTL design. This two-stage workflow significantly improves the efficiency of exploration.

However, this workflow faces a fundamental fidelity challenge [8, 15, 47]: simulators may not fully capture the behavior of the final RTL. Such discrepancies can lead to inaccurate performance estimates and undermine the reliability of early-stage design exploration. In particular, because of their higher abstraction level, simulators are more prone to modeling mismatches caused by outdated parameter configurations, idealized or incomplete resource-contention modeling, or misestimated microarchitectural feature importance [24].

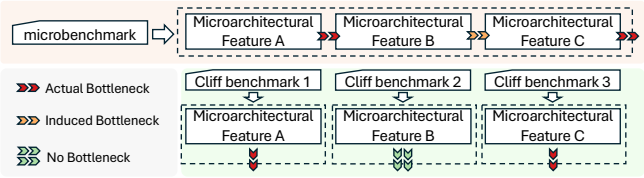


Figure 1: Microbenchmark vs. Cliffs calibration. Microbenchmark confounds the effect of multiple microarchitectural features, while Cliff benchmarks isolate each feature individually.

Model calibration [27, 51, 52], which ensures that the simulator consistently reflects the intended microarchitecture, thus becomes essential. It leverages authoritative design knowledge, such as detailed specifications, feature lists, and RTL source code, to identify modeling gaps, validate assumptions, and refine simulator configurations so that its behavior matches the RTL, the ground-truth microarchitectural intent. A natural first step is *specification-driven calibration*, where key modeling objectives are extracted directly from design specification and RTL. However, as modern architectures grow more complex, manually tuning numerous configuration knobs becomes increasingly impractical and error-prone [53, 67]. Moreover, the performance impact of certain design features is not always obvious. Simulators may simplify them for flexibility [55, 56], only to later find that these details significantly affect performance in specific scenarios.

Given these limitations, *behavior-driven calibration* provides a complementary perspective by examining software-visible performance behaviors (e.g., IPC) to expose mismatches between the simulator and the RTL. A common technique is microbenchmarking, which uses small, synthetic code kernels to probe specific microarchitectural features [66]. However, microbenchmarks often inadvertently exercise multiple interacting microarchitectural components, making it difficult to isolate the contribution of any single microarchitectural feature [45]. For example, a microbenchmark intended to measure DCache latency is found to be confounded by unrelated

*Hao Zhen and Qingxuan Kang contributed equally to this work.

microarchitectural behaviors, such as load-violation checking and recovery penalties from load-use mis-speculation [15].

To facilitate efficient and unambiguous calibration, we propose *Microarchitecture Cliffs (Cliffs)*[1], a systematic testing methodology that attributes discrepancies to a single microarchitectural feature, thereby enabling an effective assessment of microarchitectural-level calibration accuracy. Each Cliff benchmark targets a specific microarchitectural feature, such as a distinct allocation policy (e.g., ROB allocation) or a resource contention constraint (e.g., DCache bank conflicts).

Microbenchmarking and Cliffs differ in granularity. As shown in Fig. 1, a microbenchmark may simultaneously activate several microarchitectural features. When one or more performance bottlenecks are present (*red* arrows), identifying all of them may not be straightforward, as it is difficult to distinguish them from those that are merely *induced* by upstream constraints (*yellow* arrows, feature B). In contrast, Cliffs isolate individual microarchitectural features, surface bottlenecks without interference from other components, and correctly identify true bottlenecks (features A and C).

To realize the Cliff methodology, we address two key challenges. (1) *Given the large state space of modern microarchitectures, how can we identify which components require calibration?* Cliffs cluster performance counters that reveal performance bottlenecks and use them to prioritize microarchitectural components for calibration. (2) *Given the complex dependencies among microarchitectural components, how can we construct Cliff benchmarks that enable single-feature attribution?* Guided by the prioritized components and feature list, the Cliff methodology constructs instruction snippets that progressively stress and isolate a single microarchitectural feature, allowing reliable single-feature attribution.

Using the Cliff methodology, we curate a set of Cliff benchmarks to support the calibration between the XiangShan version of gem5 (XS-GEM5) and XiangShan open-source CPU (XS-RTL) [35, 49, 50, 63]. During calibration, the performance error between XS-GEM5 and XS-RTL across Cliffs benchmarks is reduced from 59.2% to just 1.4%. We further demonstrate that, through detailed microarchitectural calibration, the Cliff methodology significantly improves the accuracy of the measured performance gain of the evaluated feature (*relative error*). More specifically, we use Store Set, a memory dependency predictor, as a representative *tightly coupled microarchitectural feature* because its effectiveness depends on several interacting microarchitectural mechanisms, including *Nuke Replay* and *STA-STD separation* (described in §5.2). By isolating and calibrating these coupled features, Cliffs reduces the relative error of the Store Set predictor's evaluation on *h264ref* by 48.03%. Meanwhile, Cliffs also effectively reduce the performance discrepancy between the simulator and RTL (*absolute performance error*). The impact of this calibration is demonstrated by reduced runtime differences: 7.6% (*int*), 15.1% (*fp*) for SPEC CPU2006 and 15.1% (*int*), 21.0% (*fp*) for SPEC CPU2017.

In summary, we make the following major contributions:

- We find that isolating performance differences to a single feature improves simulator-RTL calibration.
- We propose Microarchitecture Cliffs, a benchmark generation methodology that attributes performance discrepancies between the architectural simulator and RTL to a single microarchitectural feature.
- Guided by Cliffs, we systematically calibrate XS-GEM5 against XS-RTL, reducing the relative error when using the Store Set predictor by 48.03%.
- Guided by Cliffs, we also significantly reduced the absolute performance error between XS-GEM5 and XS-RTL.

[1]We will open source the tools upon acceptance.

2 Background

To accurately project performance trends for a specific microarchitectural design, the architectural simulator should be carefully calibrated to reflect the corresponding design features and constraints. Unlike reverse engineering [37, 38, 44, 54], calibration treats the microarchitecture as a white box. Thus, a systematic calibration process typically involves two complementary approaches: *specification-driven calibration* (§2.1) and *behavior-driven calibration* (§2.2).

2.1 Specification-Driven Calibration

Specification-driven calibration begins with architects identifying performance-critical features from the design specification and examining their possible mismatches in the existing simulation infrastructure. Once such mismatches are identified as likely sources of performance inaccuracy, the modeling team begins implementing the corresponding feature to capture the relevant microarchitectural details. However, as hardware designs grow increasingly complex to support diverse microarchitectural features, achieving comprehensive calibration through first principles becomes highly challenging. Identifying performance-critical insights from a microarchitectural design specification requires enumerating the possible interactions among multiple components and reasoning about the concurrency, contention, and feedback loops that determine the efficiency of ILP and MLP [9, 25, 43]. Accurately modeling these features, including speculative mechanisms and concurrent pipeline events, requires substantial domain expertise and modeling effort. The structural and behavioral complexity of these components makes critical microarchitectural details prone to modeling mismatches [47]. As a result, *behavior-driven calibration* is required to verify whether the modeled behavior meets expectations by observing the resulting performance characteristics.

2.2 Behavior-Driven Calibration

Behavior-driven calibration identifies modeling mismatches by using targeted benchmarks and comparing performance metrics, such as IPC or microarchitectural events, between the simulator and the RTL. This process is analogous to RTL design and verification: implementing RTL from a design specification is a specification-driven calibration process. However, bugs in RTL implementations require systematic verification using test cases designed to probe corner cases [14, 60]. Similarly, in performance modeling and calibration, developing an architectural simulator based on RTL code and feature lists is a specification-driven calibration process, and behavior-driven calibration is crucial for identifying implementation discrepancies between the simulator and the RTL. Therefore, performance convergence using targeted behavior-driven calibration approaches is as essential to architectural simulators as functional verification

is to RTL development. Under the behavior-driven calibration approach, we observe three methods are commonly used: *Microbenchmarking*, *Time-Proportional*, and *Performance Event Analysis*.

Microbenchmarking: Microbenchmarks use instruction snippets to analyze specific operations, enabling fast evaluation of processor performance. For instance, prior work [2, 10, 15, 28, 34] develops microbenchmarks to calibrate architectural simulators against RTL, which can be used to assess key processor modules such as branch predictors and execution units. However, in most cases, microbenchmarks fail to isolate performance differences to a single microarchitectural feature, as they are often affected by multiple interacting components. Ideally, each benchmark should capture only one source of discrepancy in the simulator to facilitate targeted calibration.

Time-proportional Method (e.g., TEA/TIP): Time-proportional analysis [22, 23], widely used in software profilers, attributes performance bottlenecks to individual instructions by ranking them according to the number of cycles they occupy the head of the Reorder Buffer (ROB). However, although effective in pinpointing bottlenecks for specific microarchitectural features, these methods fail to expose microarchitectural discrepancies across different processors, since the resulting bottleneck rankings are not directly comparable. This limitation stems from the fact that Time-proportional methods are primarily designed to analyze performance within a single microarchitecture to guide software optimizations, rather than to compare behaviors across different hardware configurations.

Performance Event Analysis: Performance event analysis attributes performance bottlenecks to microarchitectural components [18, 64]. However, event counters are not necessarily directly correlated with performance impact [23]. Multiple correlated events may exhibit anomalies simultaneously, and domain expertise remains essential to distinguish relevant counters from noise to accurately identify the root cause. Moreover, they have the same incomparability problem as the Time-proportional method, as shifts in bottlenecks do not directly imply the microarchitectural discrepancies between the two implementations.

Overall, existing performance analysis and calibration methods face challenges in single microarchitectural feature attribution, making calibration difficult.

3 Motivation: Single Feature Attribution

Behavior-driven calibration is an indispensable step in calibrating the architectural simulator with the RTL. Although we already have access to the RTL code and feature list, behavior-driven calibration enables us to effectively detect outdated parameter configurations, idealized or incomplete resource-contention modeling, or misestimated microarchitectural feature importance that may remain in specification-driven calibration.

Existing behavior-driven calibration approaches can pinpoint discrepancies between the simulator and RTL. For example, TEA attributes performance differences to individual instructions. However, these discrepancies often stem from interactions among multiple microarchitectural components and fail to provide fine-grained guidance for calibration. We implement TEA [23] on XS-RTL and evaluate it on two configurations: *KMH*, a 6-wide high-performance core, and *MINIMAL*, a simplified in-house design. As shown in Table 1, the bottleneck instructions identified in the two configurations differ significantly. When different instructions are identified as bottlenecks across runs, the absence of a consistent baseline makes it difficult to provide accurate guidance for microarchitectural calibration. Moreover, bottleneck instructions are often associated with multiple microarchitectural features, further complicating the analysis. While instruction-level attribution works well for analyzing a single processor, it faces challenges when applied to cross-architecture calibration.

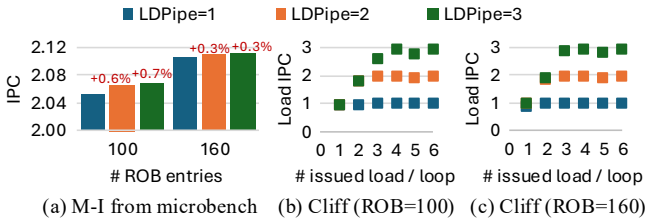
Table 1: Top 10 bottleneck instructions identified by TEA on two hardware configurations. These results show a differing attribution order making root-cause analysis difficult.

KMH-XS-RTL		MINIMAL-XS-RTL	
PC	Top 10 Instructions	PC	Top 10 Instructions
0x83f98	*fadd.d fa0,fa0,ft1*	*0x5aba8*	*ld a5,0(a5)*
0x5ab98	*ld a1,48(s1)*	*0x5ab98*	*ld a1,48(s1)*
0x26c70	*fmadd.d fa4,fa4,fa1,fa3*	*0x417dc*	*ld a5,8(a5)*
0x26cfa	*fneg.d fa5,fa5*	*0x5abe0*	*lw a5,8(s0)*
0x417e0	*snez a0,a0*	*0x5ab94*	*ld s1,16(s1)*
0x226e6	*beqz a5,226bc*	*0x26cfa*	*fneg.d fa5,fa5*
0x226a6	*bnez a5,226c4*	*0x5abb0*	*lw a5,20(s2)*
0x5abe0	*lw a5,8(s0)*	*0x5aba2*	*ld a5,0(s1)*
0x26d06	*beqz a5,26d82*	*0x5abf2*	*ld s0,16(s0)*
0x5a6bc	*lw a5,8(s0)*	0x226c8	*fld fa1,-1944(gp)*

To accurately identify the root causes, behavior-driven calibration approaches (e.g., microbenchmarks) should produce outputs that correlate closely with the targeted microarchitectural feature. Although extensive efforts have been made to design targeted benchmarks for detecting functional and performance bugs or exploring simulator details [1, 5, 29, 39, 57, 66], benchmarks aimed at verifying the consistency between architectural simulators and RTL implementations remain relatively underexplored. We make the insight that *a single benchmark is often insufficient to capture the entire state space of a given microarchitectural feature, and is prone to interference from unrelated factors.*

Based on these observations, we propose *Microarchitecture Cliffs (Cliffs)*, a methodology that systematically constructs targeted *Cliff benchmarks* to expose the behavior of a single microarchitectural feature, enabling effective diagnosis of microarchitectural feature-level inconsistencies. The methodology begins by identifying key architectures that require calibration and generates benchmark sets for each critical microarchitectural feature to localize performance discrepancies between the simulator and RTL. In addition, we develop automated tools to enhance the efficiency of this process.

In particular, Cliff benchmarks use instruction snippets tailored to specific microarchitectural components to enable precise external observation of these features. The outputs of Cliff benchmarks produce a correlated mapping between the degree of stress applied and the microarchitectural response, typically reflected in metrics such as execution cycles or IPC. These performance trends, including non-linear responses identified by inflection points in the state space, provide valuable insights into the modeled behavior of the microarchitectural feature.



(a) M-I from microbench (b) Cliff (ROB=100) (c) Cliff (ROB=160)

Figure 2: M-I microbenchmark vs. Cliffs for L1 DCache bandwidth. M-I shows minimal IPC sensitivity to LDPipe, while Cliffs reveal clear distinctions among pipeline widths.

To illustrate the importance of Cliffs' single-feature attribution, we compare Cliff benchmarks against the M-I microbenchmark for independent memory accesses from a widely used calibration suite [15, 46]. As shown in Fig. 2, both benchmarks aim to measure the effective bandwidth of the L1 DCache (which we call LDPipe). However, M-I's testing effectiveness falls short of expectations: increasing LDPipe from 1 to 3 raises the IPC by less than 1% across different ROB configurations. Consequently, using M-I under the assumption that LDPipe miscalibration will manifest as noticeable IPC discrepancies is ineffective, as the variation induced by LDPipe is even smaller than that caused by unrelated microarchitectural features such as ROB size.

In contrast, Cliffs clearly differentiates the behaviors of LDPipe=1, 2, and 3. As the number of issued loads per loop increases, the load IPC saturates at distinct levels, forming well-separated plateaus corresponding to different LDPipe configurations. This quantitative separation directly reflects the underlying bandwidth constraint and demonstrates Cliff's ability to attribute performance differences to the LDPipe feature alone, thereby enabling reliable feature-aware calibration.

As mentioned in §2.1, most high-level simulators do not implement every microarchitectural detail of the RTL design. The Cliff methodology provides an effective way to assess how accurately a simulator reflects the underlying hardware. By operating at the microarchitectural feature level, it enables us to determine which components are faithfully modeled and which are insufficient, thereby revealing whether the simulator is suitable for evaluating a set of optimization ideas.

4 Design

The design of the Cliff methodology faces two key technical challenges. First, how to select the key architectures that require calibration; second, how to construct Cliff benchmarks for these features to enable single feature attribution.

To address these challenges, the Cliff methodology is designed with two main components, as illustrated in Fig. 3. Cliff-SKP (Selecting Key architectural bottleneck Points) identifies performance counters that reflect architectural bottlenecks as *information probes* and clusters these counters to pinpoint bottleneck architectures. Next, after identifying key architectures, Cliff-BACT (Benchmark Assembly for Characterizing Traits) extends these microarchitectural features to construct Cliff benchmarks, enabling single-feature attribution.

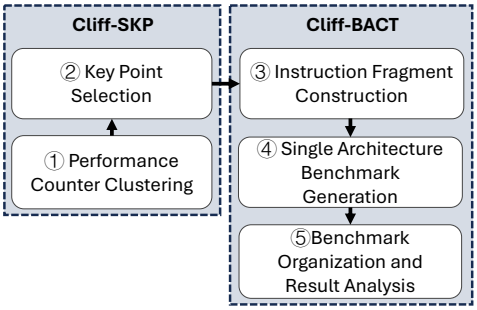


Figure 3: Overview of the Cliff methodology. Cliff-SKP clusters performance-counter probes to identify key architectural bottlenecks. Based on the key architectures, Cliff-BACT constructs Cliff benchmarks that isolate individual microarchitectural traits for single-feature performance attribution

4.1 Cliff-SKP

To prioritize the calibration of architectural components with significant performance impact, Cliff-SKP uses performance counters that reflect architectural bottlenecks as *information probes*. It executes SPEC CPU2006 on XS-GEM5 during the initial calibration phase, followed by clustering on the collected performance counter data. Based on the results, it identifies critical architectural regions as key directions for constructing Cliff benchmarks. The clustering process consists of two rounds. The first round broadly categorizes performance bottlenecks into either frontend-bound or backend-bound, while the second round further breaks down the specific architectural components that contribute most significantly to the relevant category.

The first-round clustering begins with the top-level performance counters defined by Top-Down [64], which organizes performance bottlenecks into four key categories: Frontend Bound, Bad Speculation, Backend Bound, and Retiring. These counters provide a coarse-grained foundation for identifying architectural bottlenecks. We then apply the DBSCAN [17] clustering algorithm to analyze their behavior, and visualize the results with a heatmap. With the exception of a few outliers, most test points are clustered under the Backend Bound category. This indicates that the main performance bottleneck lies in the processor backend, providing a clear direction for the design of Cliff benchmarks.

Building on the initial clustering results, we conduct a finer-grained clustering analysis on the 16 performance counters categorized under backend bottlenecks in the Top-Down methodology, which further breaks down the causes of backend bottlenecks. As shown in Fig. 4, the refined clustering allows us to preliminarily exclude architectural components with relatively minor impact on overall performance, such as the DTLB (DTlbStall), integer bound (IntDQBandwidth), L3 cache (LoadL3Bound), and commit limit (MemCommitLimit). Ultimately, we focus Cliff benchmarks construction on key backend architectures including floating-point bound (FVDQBandwidth), memory instruction (MemDQBound), L1 cache (LoadL1Bound), L2 cache (LoadL2Bound), and system memory (LoadMemBound), which are identified as the most likely to have the greatest performance impact.

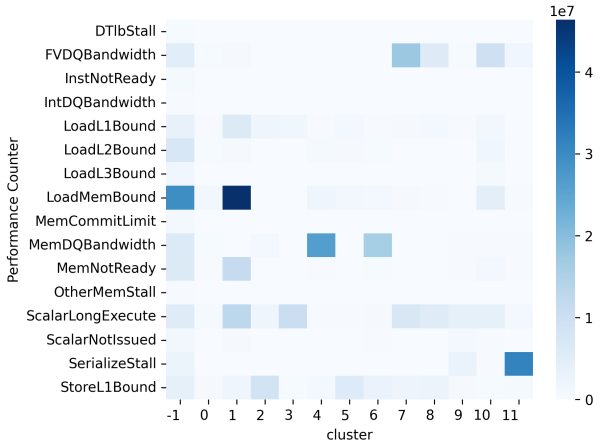


Figure 4: Second-round clustering of backend performance counters identifies floating-point bandwidth, memory instruction bandwidth, L1/L2 caches, and system memory as concrete directions for Cliff benchmark construction.

4.2 Cliff-BACT

Given the identified target architectural components, we first construct Cliff benchmarks based on prioritized microarchitectural features, ensuring that each benchmark reflects a single microarchitectural feature.

Unlike traditional microbenchmarks that rely on a single program to evaluate performance, the Cliff methodology employs carefully designed groups of instruction snippets. By analyzing the IPC or execution time trends of these snippet groups, one can identify differences in the target microarchitectural feature between the simulator and RTL, and assess the degree of calibration achieved. Cliff benchmarks start from parameter-level features and extend known behaviors using instruction snippet groups with performance trends, ensuring both visibility and isolation of the target feature. The Cliff methodology constructs Cliff benchmarks that need to bridge the gap between instruction-level behavior and microarchitectural level, while reducing interference between different microarchitectural features within a single benchmark. For complex structures, accurate evaluation may require combining multiple instruction snippets and incorporating observations from related components.

4.2.1 Bridge the Gap Between the Instruction Level and the Microarchitecture Level. To bridge the gap between the instruction level and the microarchitecture level, we construct instruction snippets that make specific microarchitectural features the performance bottlenecks. Benchmarks are categorized into functional classes, each with a tailored construction method, and microarchitectural pressure is progressively increased within each category. By analyzing performance trends and identifying inflection points, we systematically reveal the features of targeted microarchitectural components.

Latency Testing: By constructing read-after-write (RAW) dependency chains between instructions, the execution latency of the target instruction is the bottleneck on the critical path.

Bandwidth Testing: Independent instruction streams without dependencies are constructed to expose execution bandwidth bottlenecks in target microarchitectural components.

Capacity Test: This benchmark combines long-latency and short-latency instructions to apply structural pressure. A long-latency instruction blocks the release of resources, so the maximum capacity can be determined by measuring the execution time with different numbers of short-latency instructions.

Special Case Testing: Certain microarchitectural features can be evaluated by designing paired benchmarks, where one benchmark triggers the feature and the other does not. This approach enables researchers to determine whether the processor implements a specific optimization.

4.2.2 Reduce Interference Between Different Microarchitectural Features Within a Single Benchmark. To ensure accurate and valid measurement, effective Cliff benchmark design relies on *interference suppression* and *causal observability*. The interference suppression principle requires minimizing the influence of irrelevant microarchitectural features during the execution of instruction snippets. To ensure that each Cliff benchmark reflects only the effect of the targeted microarchitectural feature, interference from other features on IPC should be minimized. Additionally, Cliff benchmarks should warm up the branch predictor through repeated execution and constrain the memory footprint size to ensure that the target cache block resides in the intended cache level prior to measurement. These precautions help produce stable and reliable results.

The causal observability principle is grounded in the pressure gradient construction methodology. Each Cliff benchmark consists of a set of instruction snippets that reflect the characteristics of a single microarchitectural feature. When the set of instruction snippets applies progressively increasing pressure to the target microarchitectural feature, its execution time or IPC should exhibit a corresponding regular variation.

4.2.3 Organizing Cliff Benchmarks by Microarchitectural Feature. Some microarchitectural features span multiple related components. Cliff benchmarks can start by using simple instruction snippets to test relatively independent baseline structures. Once the performance characteristics of these simpler components are well understood, more complex microarchitectural traits can be progressively identified. Subsequently, by systematically organizing and coordinating multiple Cliff benchmarks, the overall behavior of the target microarchitectural feature can be accurately captured.

4.2.4 Identifying Undervalued Microarchitectural Features. After constructing Cliff benchmarks based on prioritized microarchitectural features, we analyze abnormal trends or deviations from expected behavior to reveal microarchitectural features that might initially be deemed unimportant. At the same time, this process serves as a *microarchitectural audit* of the original design and specification-driven calibration flow, helping identify limitations in the calibration process.

Next, we use two case studies to demonstrate (1) how to systematically construct Cliff benchmarks and (2) how we can discover unexpected discrepancies due to undervalued performance features.

4.3 Cliffs Design Case Study 1: ROB

This case study illustrates how to systematically construct Cliff benchmarks to reveal individual microarchitectural features, using our experience in building ROB-related Cliff benchmarks as an example.

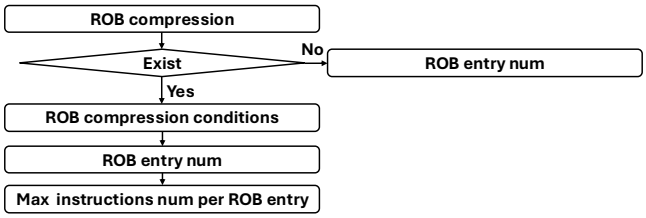


Figure 5: ROB-related Cliff benchmarks.

Complexities and solutions. ROB compression packs eligible instructions into a single ROB entry. This significantly increases the effective capacity of the ROB, thereby enhancing overall processor performance. To evaluate ROB-related features, we construct a series of Cliff benchmarks. When the ROB can hold all instructions, the execution time remains constant due to parallelism; once its capacity is exceeded, the execution time increases. As shown in Fig. 5, the overall evaluation procedure consists of the following three stages: (1) determining whether ROB compression exists; (2) identifying the conditions under which compression occurs and the compression capacity per entry; (3) measuring the maximum number of instructions a single ROB entry can compress.

4.3.1 Determining the Existence of ROB Compression. To verify the performance correctness of ROB compression feature modeling in the simulator, we construct Cliff benchmarks that use different combinations of instruction types occupying the ROB. Therefore, we probe the ROB behavior using various permutations of six instruction types: *beq*, *ld*, *st*, *add*, *fadd*, and *nop*. To bridge the gap between the instruction level and the microarchitectural level, we adopt the approach described in the *capacity test* to evaluate ROB size: constructing a long-latency load dependency chain to stall the head of the ROB, while filling the remaining entries with short-latency instructions selected from the six instruction types discussed above. As the number of short-latency instructions increases, more ROB entries are occupied, leading to higher pressure on ROB. Once the ROB can no longer accommodate these instructions, the execution time of the instruction snippet starts to rise. This marks the inflection point in the test results.

To isolate the ROB as the performance bottleneck, we measure multiple combinations of instruction snippets. A single *ld* instruction is affected by both the load queue and the ROB. By gradually reducing the load queue pressure (from *ld* to *ld+st* and then to *beq+ld+st*), the inflection point eventually becomes determined solely by the ROB, effectively eliminating the impact of the load queue. Similarly, as shown in Table 2, sequences involving *beq* or combinations such as *beq + add/fadd/nop* and *beq + ld/st + nop/add/fadd*, exhibit the same inflection point, suggesting that a standalone *beq* instruction cannot be compressed. Experimental results show that the ROB holds 152 entries under non-compression conditions, with an additional 8 entries required to accommodate long-latency load instructions. This yields a total ROB capacity of 160 entries.

Table 2: Cliff benchmark results on calibrated XS-GEM5: ROB capacity under different instruction snippets. Instr. Num records the number of instructions at which the execution time starts to increase Performance inflection points indicate bottlenecks related to ROB compression capacity. Background colors highlight regions where instruction snippets are influenced by other microarchitectural constraints.

Instr. Fragments	Instr. Num	Instr. Fragments	Instr. Num	Instr. Fragments	Instr. Num
ld	72	ld + nop	144	beq + ld + st	152
st	56	ld + add	144	beq + ld + nop	152
nop	>350	ld + fadd	144	beq + ld + add	152
fadd	160	ld + beq	144	beq + ld + fadd	152
add	182	st + nop	112	beq + st + nop	152
beq	152	st + beq	112	beq + st + add	152
		st + ld	112	beq + st + fadd	152
		nop + add	380	beq + add + add	240
Bottlenecks		add + fadd	324	beq + fadd + fadd	240
load queue size		beq + fadd	152	beq + nop + nop	320
store queue size		beq + nop	152	beq + nop + add	180
int reg num		beq + add	152	beq + nop + fadd	180
fp reg num				beq + fadd + add	180

4.3.2 Determining the Conditions for Compression. To eliminate interference from the Load/Store Queue and register file pressure, we first fill the ROB with 130 *beq* instructions, then issue the target instructions and observe the ROB saturation point. The results show that *ld* and *st* instructions saturate the ROB at around 22 entries, indicating that they are incompressible. In contrast, *nop*, *add*, and *fadd* instructions exceed this limit, confirming they are compressible.

4.3.3 Determining the Capacity for Compression. To eliminate register file pressure, we fill the first 130 ROB entries with *beq* instructions and use the remaining capacity to evaluate compression behavior. Unexpectedly, the results show an average of 5.11 instructions per ROB entry, which is inconsistent with expected ROB behavior. Further analysis of the Cliff benchmarks and RTL source code reveals that ROB compression is only performed on instructions dispatched in the same cycle. To address this, we initially attempt to use correctly predicted conditional branches to control the boundaries of the test instructions. However, since conditional branches typically rely on an *add* instruction to control loop iteration, and this data dependency makes it difficult to ensure that both instructions enter the ROB in the same cycle, preventing precise control. We therefore switch to using unconditional jump instructions (*jal*) combined with *nop* instructions to construct the test snippet. We use one *jal* followed by N *nops*, i.e., $jal + N \times nop$), where N is the number of *nops* in each instruction snippet. Since *jal* is not compressible but *nops* are, each group of *jalr + nops* instructions occupies exactly two ROB entries when N is smaller than the compression capacity.

Results show that when N is less than or equal to 6, the ROB capacity is 64 groups of test snippets. When N exceeds 6, the effective ROB capacity starts to decrease, indicating that the maximum number of compressible instructions per ROB entry is 6. Additionally, by comparing the *jal + nops* structure with snippets using only *nop* instructions, we further confirm that only instructions fetched in the same cycle can be compressed into a single ROB entry, as successful jumps break the fetch continuity across cycles.

4.4 Cliffs Design Case Study 2: DCache Banks

The Cliff methodology focuses on single-feature performance validation. When constructing Cliff benchmarks, we use our domain knowledge of the target architecture (XS-RTL) to obtain a comprehensive list of features to be tested and calibrated. This design is a necessary step, as specification-driven calibration from design specifications may introduce discrepancies that are difficult to capture statically through code reviews. This case study illustrates how analyzing performance deviations can reveal microarchitectural features that have been undervalued for its performance impact initially, thereby motivating the construction of additional Cliff benchmarks. The overall workflow is summarized in Fig. 6.

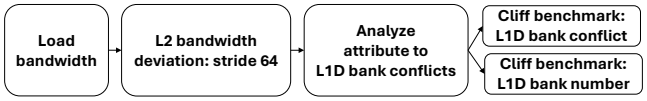


Figure 6: Analyzing performance deviations.

As discussed in §4.1, clustering results from *Cliff-SKP* highlight memory instructions and the L1 cache as primary targets for calibration. Accordingly, we construct Cliff benchmarks starting from the architectural parameters of these components. With the load pipeline bandwidth already characterized, we proceed to evaluate L2 bandwidth under different load stride patterns. Specifically, RTL shows a significant drop in L2 bandwidth at a stride of 64 bytes, while XS-GEM5 does not. Based on the observed performance trends and the feature list, we identified that the discrepancies are primarily caused by L1D bank conflicts, a constraint that was deemed a low priority feature. After confirming the phenomenon, we construct new Cliff benchmarks by varying the number of banks to characterize its impact, enabling fine-grained calibration between the simulator and RTL.

5 Evaluation

We evaluate the Cliff methodology from two major aspects. In the first part, we focus on evaluating Cliffs' effectiveness. We begin by examining the aggregate results of Cliffs (§5.1), then assess its ability to reduce the relative error of feature evaluation (§5.2). We further evaluate its impact on lowering the simulator's overall absolute performance error (§5.3). Finally, we analyze the effect of the clustering process (§5.4) and present detailed results for selected Cliff benchmarks (§5.5). In the second part, we examine whether Cliffs can accurately capture specific microarchitectural features of BOOM [11] (§5.6). We further adapt the methodology to new workloads beyond SPEC to demonstrate that Cliffs remains effective across different workloads (§5.7).

5.1 Cliff Results

By leveraging Cliff benchmarks, the Cliff methodology enables accurate and detailed characterization of microarchitectural features. As shown in Fig. 7, Cliff benchmarks achieve a low overall deviation between the measured values and the design-intended values (measurement error) of just 1.8%, demonstrating their strong evaluation capability.

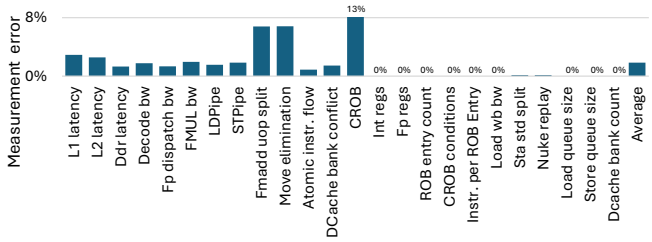


Figure 7: Accuracy measurement of Cliff benchmarks shows an average error of only 1.8% compared to the design values.

Table 3: Evaluation of microarchitectural features using Cliff benchmarks shows that calibration reduces the error on Cliff benchmarks by 57.8%.

Test	RTL	Before		After	
L1 latency	4.07	4.14	(+1.72 %)	4.14	(+1.72 %)
L2 latency	16.07	27.08	(+68.51 %)	15.08	(-6.16 %)
Ddr latency	226.49	248.02	(+9.51 %)	218.48	(-3.54 %)
Decode bw	5.92	5.88	(-0.68 %)	5.88	(-0.68 %)
LDPipe	2.96	2.95	(-0.34 %)	2.96	(+0.00 %)
STPipe	1.95	1.97	(+1.03 %)	1.97	(+1.03 %)
Int regs	224	224	(+0.00 %)	224	(+0.00 %)
Fg regs	192	192	(+0.00 %)	192	(+0.00 %)
ROB entry count	160	320	(+100 %)	160	(+0.00 %)
Load queue size	72	128	(+77.78 %)	72	(+0.00 %)
Store queue size	56	96	(+71.43 %)	56	(+0.00 %)
Fmadd uop split	4.10	2.18	(-46.78 %)	4.16	(+1.61 %)
Move elimination	5.65	5.56	(-1.60 %)	5.52	(-2.30 %)
Atomic instr. flow	33.19	13.16	(-60.35 %)	32.71	(-1.45 %)
Fp dispatch bw	2.97	3.94	(+32.66 %)	2.85	(-4.04 %)
FMUL bw	2.96	3.91	(+32.09 %)	2.95	(-0.34 %)
DCache bank conflict	0.99	2.95	(+198 %)	0.93	(-6.06 %)
CROB	4.60	1.00	(-78.26 %)	5.11	(+11.09 %)
CROB conditions	4	0	(-99.99 %)	4	(+0.00 %)
Instr. per ROB Entry	6	1	(-83.33 %)	6	(+0.00 %)
Load wb bw	3	8	(+166 %)	3	(+0.00 %)
Sta std split	0	501	(+99.99 %)	0	(+0.00 %)
Nuke replay	0	501	(+99.99 %)	0	(+0.00 %)
DCache bank count	8	0	(-99.99 %)	8	(+0.00 %)

Table 3 summarizes the results of Cliff benchmarks across various scenarios and reports their relative deviations from RTL. Using Cliffs, we calibrate a set of microarchitectural features that are identified during specification-driven calibration. Overall, the performance discrepancy between XS-GEM5 and XS-RTL on Cliff

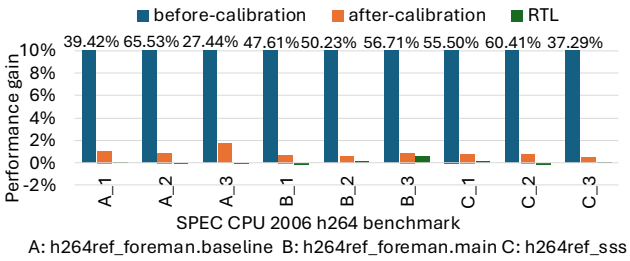


Figure 8: Comparison of Store Set evaluation on *h264ref* before and after calibration. The sensitivity is significantly larger on the uncalibrated simulator, illustrating a false trend.

benchmarks is 59.2% before calibration, and is reduced to 1.4% after calibration. It is worth noting that we also evaluate the calibration of several detailed microarchitectural features, such as Nuke Replay, sta std split, and DCache bank conflict. After calibration, XS-GEM5 accurately models the corresponding pipeline behaviors observed in RTL.

5.2 Relative Error of Store Set Evaluation

To better evaluate Cliffs' ability to guide simulator calibration and reduce relative error, we compare the Store Set evaluation on XS-RTL and XS-GEM5 before and after calibration. It is important to note that simulator mismatches do not only originate from incorrect parameter settings; they often stem from missing or incomplete microarchitectural features.

Before calibration, we observed significant discrepancies between XS-GEM5 and XS-RTL in the Store Set performance gains on *h264ref*, with a relative error of 48.86%. However, solely observing the uncalibrated XS-GEM5 and XS-RTL behaviors on *h264ref* makes it difficult to determine which microarchitectural features are responsible for these discrepancies. As shown in Table 3, the Cliff methodology reveals that two microarchitectural features related to memory-ordering dependencies (Nuke Replay and STA-STD separation) differ between the uncalibrated XS-GEM5 and XS-RTL.

With these microarchitectural features fixed, the simulator's relative error in evaluating Store Set is significantly reduced to 0.83%. As shown in Fig. 8, before calibration, the simulator reports an average performance improvement of 48.90% with Store Set on *h264ref*, with the highest gain reaching 65.53% for certain slices. As a result, the average performance benefit of Store Set drops to 0.87%, and previously high-gain checkpoints show only 0.89% improvement, which closely matches with the RTL behavior (0.04% error).

These results demonstrate that, with Cliffs-assisted calibration, the architectural simulator can more accurately evaluate the true performance benefits of optimization techniques, effectively reducing the relative error. Meanwhile, Cliffs help clarify which components of the simulator are faithfully modeled and which are insufficient, thereby enabling us to determine whether the simulator is suitable for evaluating this class of microarchitectural features. In the Store Set evaluation, uncalibrated XS-GEM5 misrepresents strongly related features, resulting in high relative error, whereas calibrated XS-GEM5 correctly models these features and enables accurate evaluation of Store Set gains.

5.3 Absolute Error

We use absolute performance error to evaluate the overall calibration between the simulator and RTL. To precisely assess the performance impact of Cliffs calibration on both parameter-level and microarchitecture-level features, we compare the performance errors before calibration, after parameter calibration, and after microarchitectural feature calibration.

As shown in Fig. 9, for SPECint2006, the absolute performance error between XS-GEM5 and XS-RTL decreases from 18.1% before calibration to 15.6% after parameter calibration, representing a 2.5% reduction and indicating a substantial improvement in accuracy. After calibrating the microarchitectural features, the absolute performance error further drops to 10.4%, corresponding to a total reduction of 7.6% compared with the before-calibration result and an additional 5.2% improvement over the parameter-calibrated stage. Similarly, for SPECfp2006, the absolute performance error is 23.1% before calibration. After parameter calibration, the error decreases to 18.0%, representing a reduction of 5.1%. With further microarchitectural-feature calibration, the error drops to 8.0%, corresponding to a total reduction of 15.1% compared with the before-calibration result and an additional 10.1% improvement over the parameter-calibrated stage. For SPECint2017, parameter calibration reduces the absolute performance error from 19.3% to 13.5%, achieving a 5.8% improvement. With further microarchitectural-feature calibration, the error decreases to 4.2%, representing a total reduction of 15.1% compared with the before-calibration result and an additional 9.3% improvement over the parameter-calibrated stage. For SPECfp2017, the parameter calibration gap narrows from 28.5% to 8.2%, yielding a 20.3% improvement. With microarchitectural-feature calibration, the error decreases to 7.5%, representing a total reduction of 21.0% compared with the before-calibration result and an additional 0.7% improvement over the parameter-calibrated stage.

Performance differences across some benchmarks exhibit local fluctuations, which is expected, as calibration does not reduce the gap in a strictly monotonic fashion. Nevertheless, the overall trend is clearly moving downward [8]. The Cliff methodology effectively narrows the performance gap between XS-GEM5 and XS-RTL. Before calibration, opposing errors cancel out, concealing the true extent of model inaccuracies. Partial calibration breaks this balance, revealing hidden inaccuracies and occasionally creating the illusion of reduced accuracy, while actually improving model fidelity. For instance, although *h264ref* appears less accurate after calibration, (§5.2) shows that the post-calibration trend better reflects RTL performance, enabling more faithful evaluation of Store Set. These findings confirm that Cliffs enhance calibration accuracy.

5.4 Performance Impact of Key Microarchitectural Bottlenecks

Based on two rounds of performance counter clustering, we first identify the backend as the primary focus for constructing Cliff

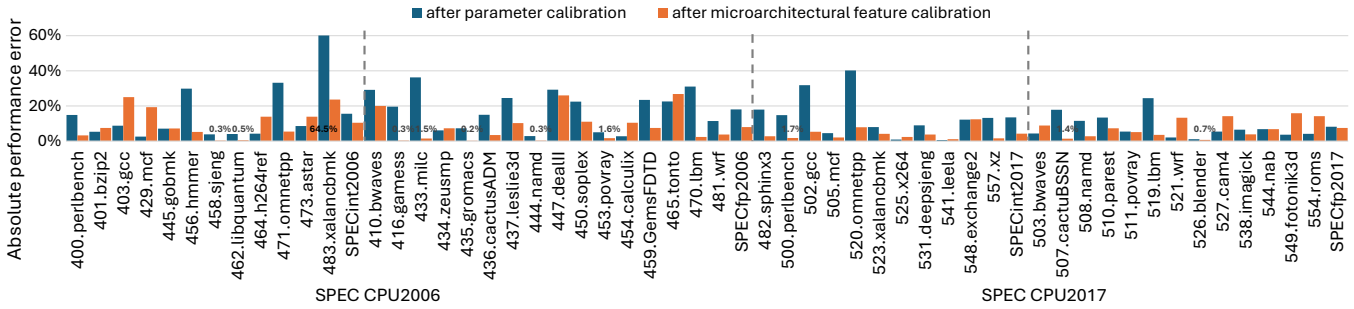


Figure 9: Performance errors between XS-RTL and XS-GEM5 after parameter calibration and after microarchitectural feature calibration. SPECint or SPECfp errors are reported as average absolute errors across benchmarks.

Table 4: Performance counter metrics from Cluster 2-0.

cluster	key point	checkpoint
2-0	LoadMemBound ScalarLongExecute StoreL1Bound	calculix_1, h264ref_sss_1, xalancbmk_1, wrf_1

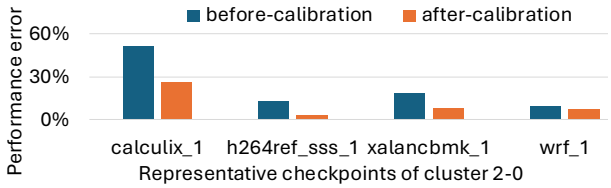


Figure 10: Performance errors before and after calibration (Cluster2-0). Calibrating the critical performance factors correlates with reduction in performance gap on benchmarks belonging to the same cluster.

benchmarks. We then further refine the construction directions within the backend domain.

Table 4 presents the details of Cluster 2-0. To further evaluate the impact of the clustered key factors on code segments, we analyze the effect of calibrating the critical performance features identified in Cluster 2-0. As shown in Fig. 10, after calibration, the performance gap between XS-GEM5 and XS-RTL on the key slices was significantly reduced, with the maximum reduction reaching 25.5%, bringing the absolute error down to 1.2%.

5.5 Analysis of Cliff Benchmarks' Results

As presented in (§5.3), we demonstrate the overall evaluation effectiveness of the Cliff benchmarks. To further present their evaluation process in a more structured and detailed manner, we categorize the results into three types: (1) trendline-based analyses for aiding result interpretation; (2) microarchitectural feature extraction through pairwise comparisons; (3) critical analysis based on identifying inflection points arising from variations in input parameters.

5.5.1 Trend Line Guided Result Evaluation. We employ trendline analysis to assist in evaluating the bandwidth of the floating-point multiplication (FMUL) unit. The benchmark results are shown in

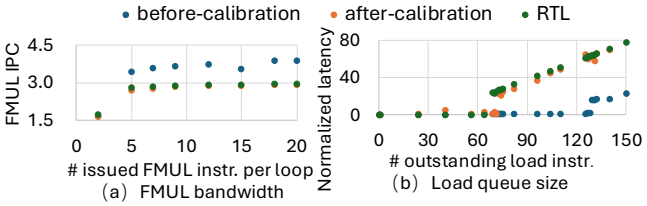


Figure 11: Cliff benchmarks for FMUL bandwidth and load queue size. The performance profile of the calibrated model aligns with RTL.

Fig. 11a. In this benchmark, we gradually increase the number of FMUL instructions that can be issued per cycle to incrementally stress the FMUL unit, while monitoring the corresponding changes in the instructions per cycle (IPC) for these operations. The results show that as the number of issued instructions increases, the IPC for FMUL instructions increases monotonically at first and then plateaus after reaching a peak value. This saturation point indicates the maximum bandwidth of the FMUL unit.

5.5.2 Change Point Detection. We evaluate the capacity of the load queue using a change point analysis approach. In our experiment, we construct an instruction snippet consisting of three dependent high-latency load operations, which serve to occupy and block several entries in the load queue. Concurrently, we gradually increase the number of parallel short-latency loads (i.e., outstanding loads) to continuously apply pressure on the load queue. As shown in Fig. 11b, by measuring the execution latency of the entire instruction segment, we observe a distinct latency jump when the number of outstanding loads reaches a certain threshold. This sudden increase in latency indicates the saturation point of the load queue, from which we can infer its upper-bound capacity.

5.5.3 Architectural Characterization via Pairwise Comparison. To analyze the bank structure of the DCache, we focus on comparing the performance difference of load instructions when accessing the same byte versus different bytes. By constructing instruction sequences with varying memory offsets, we measure the IPC under both access patterns, as shown in Fig. 12. The results indicate that accessing the same word yields significantly lower IPC compared to accessing different words, suggesting the presence of bank conflicts

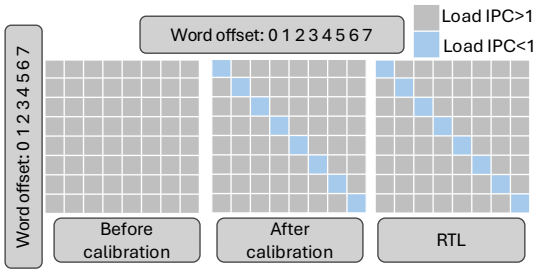


Figure 12: Cliff benchmark for DCache banks. After calibration, the Cliff benchmark for DCache banks achieves performance behavior that closely matches the RTL results.

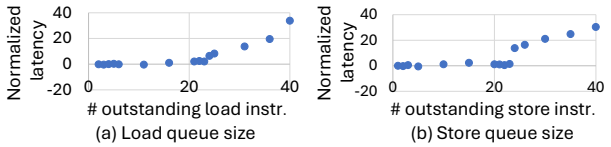


Figure 13: Cliff benchmark for load/store queue size on BOOM.

that limit concurrent execution. This notable contrast implies that each word may map to a separate bank. Based on this observation, we infer that the DCache consists of 8 independent banks.

All three cases can accurately capture the corresponding microarchitectural features. After calibration, they can measure performance changes, accurately reflect architectural details, and quantitatively represent a single microarchitectural feature.

5.6 Evaluating New Processors: BOOM

To evaluate whether the Cliff methodology can effectively capture architectural characteristics across different processors, we apply it to BOOM [11] by constructing Cliff benchmarks targeting load queue size and store queue size. Fig. 13 illustrates the performance trend lines for these two benchmarks. A clear performance inflection point is observed when the number of outstanding memory instructions reaches 24, resulting in a sharp increase in execution latency. This behavior precisely corresponds to BOOM's microarchitectural constraint, where both the load and store queues are 24 entries in size.

These results demonstrate the generality and effectiveness of the Microarchitecture Cliffs methodology. Not only can it capture fine-grained differences in XS-GEM5 before and after calibration, as well as microarchitectural features of XS-RTL, but it also successfully reflects architectural characteristics in a significantly different microarchitecture like BOOM.

5.7 Evaluating New Workloads: Verilator

To better demonstrate the generality of the Cliff methodology across different scenarios, we use the *Verilator* workload outside of the SPEC benchmark suite to evaluate the calibration effectiveness. From the Cliff-SKP analysis, the clustering results indicate that the primary performance bottleneck of the Verilator workload lies in the frontend. To address this bottleneck, we construct a Cliff benchmark to probe the maximum length of the Path History Register

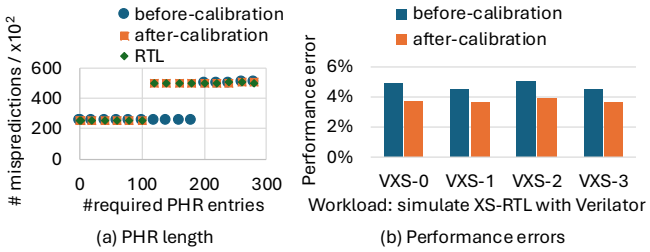


Figure 14: Cliff benchmark for PHR length and performance error running simulate XS-RTL with Verilator as a workload (VXS). A spike in mispredictions indicates that the required PHR entries exceed the current length. Calibrating the PHR length effectively reduces frontend performance error.

(PHR). Specifically, we build two correlated conditional branches and progressively vary the number of intermediate branch instructions that occupy PHR bits between the correlated branches. When the PHR length becomes insufficient, the history of the first branch will be pushed out of the PHR, and the second branch exhibits a 50% misprediction probability, leading to a sudden rise in the number of mispredicted branches.

As shown in Fig. 14, the maximum PHR length that sustains a low branch misprediction rate is approximately 200 in XS-GEM5, but significantly shorter, around 120, in XS-RTL. This difference reveals a mismatch in the maximum PHR length between the two platforms. After calibrating the relevant architectural parameters, the performance trends of XS-GEM5 and XS-RTL converge, with both exhibiting a sharp rise of mispredictions at a history length of approximately 120.

To assess the impact of calibration, we compare the frontend performance error between XS-GEM5 and XS-RTL before and after calibration using the *Verilator* workload. The results show that after calibrating the PHR length, the frontend performance error of XS-GEM5 decreases significantly, dropping from 4.72% to 3.73%.

6 Related Work

Microbenchmarks for the Memory Subsystem: These benchmarks primarily target the memory subsystem, highlighting critical metrics such as latency and bandwidth, with some extending to GPU memory [19, 42]. Nonetheless, attributing performance differences of instruction snippets to a single architectural factor remains a significant challenge [3, 16, 20, 21, 26, 40, 41, 58, 61, 62].

GPU Performance Calibration and Analysis Methodologies: Current GPU architecture analysis approaches mainly rely on the development and implementation of architecture-customized microbenchmarks. These testing tools primarily serve dual objectives: (1) evaluating the optimizability of GPU code through architecture-driven tests (such as analyzing memory hierarchy access patterns and stress testing computation unit throughput) to maximize computational throughput [19, 65]; (2) designing workload-specific tests based on workload characteristics or hardware features to identify performance bottlenecks during software execution [4, 12, 13, 36, 59]. However, the analysis results are still confined

to software execution metrics, without establishing a quantifiable architecture-performance relationship.

Automatic Benchmark Generation: Automatic benchmark generation can create benchmarks based on the workload characteristics of applications, significantly reducing execution time while accurately reflecting the application's performance. This approach aims to minimize simulation time while accurately capturing the characteristics of the benchmarks. However, it still primarily focuses on software performance and results, with relatively little attention paid to hardware architecture features [6, 30–33].

Compared with memory subsystem microbenchmarks, GPU performance analysis methods, and automatic benchmark-generation, the Cliffs attributes performance differences to single microarchitectural features and provides a more targeted link between microarchitecture and performance.

7 Conclusion

In this work, we introduce Microarchitecture Cliffs, a methodology for constructing benchmarks that isolate and attribute performance differences to single microarchitectural features. Comprehensive evaluation shows that Cliff benchmarking narrows the performance gap between XS-GEM5 and XS-RTL, reducing the absolute performance error by 15.1% on SPECint 2017 and 21.0% on SPECfp 2017. Moreover, in evaluating the representative Store Set algorithm, the calibrated simulator more faithfully reproduced RTL behavior and substantially reduced the relative error of the feature evaluation, further validating the effectiveness of Microarchitecture Cliffs in architectural calibration.

References

[1] 2022. AMD's Zen 4 Part 1: Frontend and Execution Engine. https://chipsandcheese.com/p/amds-zen-4-part-1-frontend-and-execution-engine.

[2] 2025. Agner Fog. https://agner.org/optimize/.

[3] Alif Ahmed and Kevin Skadron. 2019. Hopscotch: a micro-benchmark suite for memory performance evaluation. In *Proceedings of the International Symposium on Memory Systems*. 167–172.

[4] Cédric Augonnet, Samuel Thibault, and Raymond Namyst. 2010. Automatic calibration of performance models on heterogeneous multicore architectures. In *Euro-Par 2009–Parallel Processing Workshops: HPPC, HeteroPar, PROPER, ROIA, UNICORE, VHPC, Delft, The Netherlands, August 25-28, 2009, Revised Selected Papers 15*. Springer, 56–65.

[5] Erick Carvajal Barboza, Sara Jacob, Mahesh Ketkar, Michael Kishinevsky, Paul Gratz, and Jiang Hu. 2021. Automatic Microprocessor Performance Bug Detection. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021*. IEEE, 545–556. doi:10.1109/HPCA51647.2021.00053

[6] Robert H Bell Jr and Lizy K John. 2005. Improved automatic testcase synthesis for performance model validation. In *Proceedings of the 19th annual international conference on Supercomputing*. 111–120.

[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.

[8] Bryan Black and John Paul Shen. 1998. Calibration of microprocessor performance models. *Computer* 31, 5 (1998), 59–65.

[9] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[10] Ccelio. 2014. ccbench. https://github.com/ucb-bar/ccbench.

[11] Christopher Celio, David A Patterson, and Krste Asanovic. 2015. The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167* (2015).

[12] Miłosz Ciżnicki, Michał Kierzynka, Piotr Kopta, Krzysztof Kurowski, and Paweł Gepner. 2012. Benchmarking data and compute intensive applications on modern CPU and GPU architectures. *Procedia Computer Science* 9 (2012), 1900–1909.

[13] Taina Coleman, Henri Casanova, Ketan Maheshwari, Loïc Pottier, Sean R Wilkinson, Justin Wozniak, Frédéric Suter, Mallikarjun Shankar, and Rafael Ferreira Da Silva. 2022. Wfbench: Automated generation of scientific workflow benchmarks. In *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 100–111.

[14] Kypros Constantinides, Onur Mutlu, and Todd Austin. 2008. Online design bug detection: RTL analysis, flexible mechanisms, and evaluation. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, 282–293.

[15] Rajagopalan Desikan, Doug Burger, and Stephen W Keckler. 2001. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th annual international symposium on Computer architecture*. 266–277.

[16] Pouya Esmaili-Dokht, Francesco Sgherzi, Valéria Soldera Girelli, Isaac Boixaderas, Mariana Carmin, Alireza Monemi, Adria Armejach, Estanislao Mercadal, Germán Llort, Petar Radojković, et al. 2024. A mess of memory system benchmarking, simulation and application profiling. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 136–152.

[17] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, Vol. 96. 226–231.

[18] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E Smith. 2006. A performance counter architecture for computing accurate CPI components. *ACM SIGPLAN Notices* 41, 11 (2006), 175–184.

[19] Minquan Fang, Jianbin Fang, Weimin Zhang, Haifang Zhou, Jianxing Liao, and Yuangang Wang. 2018. Benchmarking the GPU memory at the warp level. *Parallel Comput.* 71 (2018), 23–41.

[20] Zhenman Fang, Sanyam Mehta, Pen-Chung Yew, Antonia Zhai, James Greensky, Gautham Beeraka, and Binyu Zang. 2015. Measuring microarchitectural details of multi-and many-core memory systems through microbenchmarking. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 4 (2015), 1–26.

[21] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F Oliveira, and Onur Mutlu. 2021. Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware. In *2021 12th International Green and Sustainable Computing Conference (IGSC)*. IEEE, 1–7.

[22] Björn Gottschall, Lieven Eeckhout, and Magnus Jahre. 2021. Tip: Time-proportional instruction profiling. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 15–27.

[23] Björn Gottschall, Lieven Eeckhout, and Magnus Jahre. 2023. TEA: Time-proportional event analysis. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[24] Anthony Gutierrez, Joseph Pusdesris, Ronald G Dreslinski, Trevor Mudge, Chander Sudanthi, Christopher D Emmons, Mitchell Hayenga, and Nigel Paver. 2014. Sources of error in full-system simulation. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 13–22.

[25] Lance Hammond, Vicky Wong, Michael K. Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg, Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun. 2004. Transactional Memory Coherence and Consistency. In *31st International Symposium on Computer Architecture (ISCA 2004), 19-23 June 2004, Munich, Germany*. IEEE Computer Society, 102–113. doi:10.1109/ISCA.2004.1310767

[26] Cristina Hristea, Daniel Lenoski, and John Keen. 1997. Measuring memory hierarchy performance of cache-coherent multiprocessors using micro benchmarks. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing*. 1–12.

[27] Quentin Huppert, Timon Evenblij, Manu Perumkunnil, Francky Catthoor, Lionel Torres, and David Novo. 2021. Memory hierarchy calibration based on real hardware in-order cores for accurate simulation. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 707–710.

[28] Martinus Intelfx. 2024. nanobench. https://github.com/martinus/nanobench.

[29] Jae-Eon Jo, Gyu-Hyeon Lee, Hanhwi Jang, Jaewon Lee, Mohammadamin Ajdari, and Jangwoo Kim. 2018. DiagSim: Systematically diagnosing simulators for healthy simulations. *ACM Transactions on Architecture and Code Optimization (TACO)* 15, 1 (2018), 1–27.

[30] Ajay Joshi, Lieven Eeckhout, Robert H Bell Jr, and Lizy K John. 2008. Distilling the essence of proprietary workloads into miniature benchmarks. *ACM Transactions on Architecture and Code Optimization (TACO)* 5, 2 (2008), 1–33.

[31] Ajay Joshi, Lieven Eeckhout, and Lizy John. 2008. The return of synthetic benchmarks. In *2008 SPEC Benchmark Workshop*. 1–11.

[32] Ajay Manohar Joshi. 2007. *Constructing adaptable and scalable synthetic benchmarks for microprocessor performance evaluation*. The University of Texas at Austin.

[33] Ajay M Joshi, Lieven Eeckhout, Lizy K John, and Ciji Isen. 2008. Automated microprocessor stressmark generation. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. IEEE, 229–239.

[34] zhaopengoog justjohn12345. 2023. googlebmgithub. https://github.com/google/platform_benchmarks.

[35] Wang Kaifan, Xu Yinan, Yu Zihao, Tang Dan, Chen Guokai, Chen Xi, Gou Lingrui, Hu Xuan, Jin Yue, Li Qianruo, Li Xin, Lin Jiawei, Liu Tong, Liu Zhigang, Wang Huaqiang, Wang Huizhe, Zhang Chuanqi, Zhang Fawang, Zhang Linjuan,

Zhang Zifei, Zhang Ziyue, Zhao Yangyang, Zhou Yaoyang, Zou Jiangrui, Cai Ye, Huan Dandan, Li Zusong, Zhao Jiye, He Wei, Sun Ninghui, and Bao Yungang. 2023. XiangShan Open-Source High Performance RISC-V Processor Design and Implementation. *Journal of Computer Research and Development* 60, 3 (2023), 476–493. doi:10.7544/issn1000-1239.202221036

[36] Mahmoud Khairy, Zhesheng Shen, Tor M Aamodt, and Timothy G Rogers. 2020. Accel-sim: An extensible simulation framework for validated gpu modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 473–486.

[37] Chang Liu, Shuaihu Feng, Yuan Li, Dongsheng Wang, Wenjian He, Yongqiang Lyu, and Trevor E Carlson. 2025. MDPeek: Breaking Balanced Branches in SGX with Memory Disambiguation Unit Side Channels. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 622–638.

[38] Chang Liu, Dongsheng Wang, Yongqiang Lyu, Pengfei Qiu, Yu Jin, Zhuoyuan Lu, Yinqian Zhang, and Gang Qu. 2024. Uncovering and exploiting amd speculative memory access predictors for fun and profit. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 31–45.

[39] Biruk Mammo, Milind Furia, Valeria Bertacco, Scott Mahlke, and Daya S Khudia. 2016. BugMD: Automatic mismatch diagnosis for bug triaging. In *Proceedings of the 35th International Conference on Computer-Aided Design*. 1–7.

[40] John McCalpin. 2006. STREAM: Sustainable memory bandwidth in high performance computers. *http://www. cs. virginia. edu/stream/* (2006).

[41] John D McCalpin et al. 1995. Memory bandwidth and machine balance in current high performance computers. *IEEE computer society technical committee on computer architecture (TCCA) newsletter* 2, 19-25 (1995).

[42] Xinxin Mei and Xiaowen Chu. 2016. Dissecting GPU memory hierarchy through microbenchmarking. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (2016), 72–86.

[43] Kevin E Moore, Jayaram Bobba, Michelle J Moravan, Mark D Hill, and David A Wood. 2006. LogTM: Log-based transactional memory. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.* IEEE, 254–265.

[44] Hwayong Nam, Seungmin Baek, Minbok Wi, Michael Jaemin Kim, Jaehyun Park, Chihun Song, Nam Sung Kim, and Jung Ho Ahn. 2024. Dramscope: Uncovering dram microarchitecture and characteristics by issuing memory commands. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1097–1111.

[45] Arash Nasr-Esfahany, Mohammad Alizadeh, Victor Lee, Hanna Alam, Brett W Coon, David Culler, Vidushi Dadu, Martin Dixon, Henry M Levy, Santosh Pandey, et al. 2025. Concorde: Fast and Accurate CPU Performance Modeling with Compositional Analytical-ML Fusion. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 1480–1494.

[46] Tony Nowatzki. 2018. microbench. https://github.com/VerticalResearchGroup/microbench.

[47] Tony Nowatzki, Jaikrishnan Menon, Chen-Han Ho, and Karthikeyan Sankaralingam. 2015. Architectural simulators considered harmful. *IEEE Micro* 35, 6 (2015), 4–12.

[48] Surim Oh, Mingsheng Xu, Tanvir Ahmed Khan, Baris Kasikci, and Heiner Litz. 2024. UDP: Utility-Driven Fetch Directed Instruction Prefetching. In *Proceedings of the 51st International Symposium on Computer Architecture (ISCA) (ISCA 2024)*.

[49] OpenXiangShan. 2025. XiangShan. https://github.com/OpenXiangShan/XiangShan.

[50] OpenXiangShan. 2025. xiangshan-GEM5. https://github.com/OpenXiangShan/GEM5.

[51] Karan Pathak, Joshua Klein, Giovanni Ansaloni, Said Hamdioui, Georgi Gaydadjiev, Marina Zapater, and David Atienza. 2025. Towards accurate RISC-V full system simulation via component-level calibration. *ACM Transactions on Embedded Computing Systems* (2025).

[52] Andy D Pimentel, Mark Thompson, Simon Polstra, and Cagkan Erbas. 2008. Calibration of abstract performance models for system-level design space exploration. *Journal of Signal Processing Systems* 50, 2 (2008), 99–114.

[53] Yudi Qiu, Tao Huang, Yuxin Tang, Yanwei Liu, Yang Kong, Xulin Yu, Xiaoyang Zeng, and Yibo Fan. 2023. Gem5tune: a parameter auto-tuning framework for gem5 simulator to reduce errors. *IEEE Trans. Comput.* 73, 3 (2023), 902–914.

[54] Xida Ren, Logan Moody, Mohammadkazem Taram, Matthew Jordan, Dean M Tullsen, and Ashish Venkat. 2021. I see dead μops: Leaking secrets via intel/amd micro-op caches. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 361–374.

[55] Alex Renda, Yishen Chen, Charith Mendis, and Michael Carbin. 2020. Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 442–455.

[56] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Computer architecture news* 41, 3 (2013), 475–486.

[57] Ronak Singhal, KS Venkatraman, Evan R Cohn, John G Holm, David A Koufaty, Mahesh J Madhav, Markus Mattwandel, Nidhi Nidhi, Jonathan D Pearce, Madhusudanan Seshadri, et al. 2004. Performance Analysis and Validation of the Intel® Pentium® 4 Processor on 90nm Technology. *Intel Technology Journal* 8, 1 (2004).

[58] Carl Staelin. 2005. lmbench: an extensible micro-benchmark suite. *Software: Practice and Experience* 35, 11 (2005), 1079–1105.

[59] Ryan Taylor and Xiaoming Li. 2010. A micro-benchmark suite for AMD GPUs. In *2010 39th International Conference on Parallel Processing Workshops*. IEEE, 387–396.

[60] Shobha Vasudevan, Wenjie Joe Jiang, David Bieber, Rishabh Singh, C Richard Ho, Charles Sutton, et al. 2021. Learning semantic representations to verify hardware designs. *Advances in Neural Information Processing Systems* 34 (2021), 23491–23504.

[61] Rommel Sánchez Verdejo, Kazi Asifuzzaman, Milan Radulovic, Petar Radojković, Eduard Ayguadé, and Bruce Jacob. 2018. Main memory latency simulation: the missing link. In *Proceedings of the International Symposium on Memory Systems*. 107–116.

[62] Rommel Sánchez Verdejo and Petar Radojkovic. 2017. Microbenchmarks for detailed validation and tuning of hardware simulators. In *2017 International Conference on High-Performance Computing & Simulation (HPCS)*. IEEE Computer Society, 881–883.

[63] Yinan Xu, Zihao Yu, Dan Tang, Guokai Chen, Lu Chen, Lingrui Gou, Yue Jin, Qianruo Li, Xin Li, Zuojun Li, et al. 2022. Towards developing high performance RISC-V processors using agile methodology. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1178–1199.

[64] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 35–44.

[65] Xinyao Yi, David Stokes, Yonghong Yan, and Chunhua Liao. 2021. CUDAMicroBench: Microbenchmarks to assist CUDA performance programming. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 397–406.

[66] Kamen Yotov, Keshav Pingali, and Paul Stodghill. 2005. X-ray: A tool for automatic measurement of hardware parameters. In *Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*. IEEE, 168–177.

[67] Nathan Zhang, Rubens Lacouture, Gina Sohn, Paul Mure, Qizheng Zhang, Fredrik Kjolstad, and Kunle Olukotun. 2024. The Dataflow Abstract Machine Simulator Framework. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 532–547.