2022 IEEE International Symposium on Workload Characterization (IISWC) | 978-1-6654-8798-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/IISWC55918.2022.00011

PInTE: **P**robabilistic **In**duction of **T**heft **E**victions

Cesar Gomes † Xuesi Chen § Mark Hempstead †

† *Dept. of ECE, Tufts University* § *Dept of ECE, Carnegie Mellon University*

***Abstract*—Cache contention analysis remains complex without a controlled & lightweight method of inducing contention for shared resources. Prior art commonly leverages a second workload on an adjacent core to cause contention, and the workload is either real or tune-able. Using a secondary workload comes with unique problems in simulation: real workloads aren't controllable and can result in many combinations to measure a broad range of contention; and tune-able workloads provide control but don't guarantee contention without filling all cache sets with contention behavior. Lastly, running multiple workloads increases the run-time of simulation environments by 2.4× on average.**

We introduce Probabilistic Induction of Theft Evictions, or PInTE which allows controllable contention induction via data movement towards eviction in the last level cache replacement policy. PInTE provides configurable contention with 2.6× fewer experiments, 2.2× less average time, and 5.6× less total time for a set of SPEC 17 speed-based traces. Further, PInTE incurs -8.46% average relative error in performance when compared to real contention. Run-time and reuse behavior of workloads under PInTE contention approximate behavior under real contention — information distance is 0.03 bits and 0.84 bits, respectively. Additionally, PInTE enables a first-time contention sensitivity analysis of SPEC and case studies which evaluate the resilience of micro-architectural techniques under growing contention.

I. Introduction

With dozens of cores found on modern multi-cores, there's a high likelihood that multiple workloads execute concurrently. Enabling more concurrency leads to resource contention [1, 9, 33, 37, 50], but contention analysis is complex and time consuming. Indirect analysis is common when studying architectural techniques at higher core counts or in benchmarks that are designed to exercise throughput [12, 36, 41, 51–53]. However, the combinations of workloads curated for this analysis aren't guaranteed to cover the range of contention a system or workload will see in its lifetime. Direct analysis uses tune-able workloads to cause different rates of contention across the shared cache [11, 15, 16, 21, 34], but these methods "blanket" resources with contentious behavior and can waste energy and time. Investigating the mechanics of cache contention reveals that inter-core evictions, or **thefts** are simple contention events which can be induced by the system in a targeted and controlled way.

We present Probabilistic Induction of Theft Evictions, or PInTE which induces Last Level Cache data movement and replacement for fast contention analysis. PInTE takes the notion of cache thefts presented by Gomes et al [18] and allows the system to act as a second workload by inducing thefts on LLC accesses with a tune-able probability. By doing this, PInTE minimizes the increased simulation time and reduces the total number of simulations needed. In addition, because the rate of contention can be swept, PInTE ensures that a workload or system has been studied across the full range of contention, something mixes of workloads cannot do. Real systems have resource contention so evaluating micro-architecture techniques, cache policies or system designs in isolation alone is no longer responsible design or research practice. Thus, we envision that PInTE will be used by computer architects during the design process or by application/system designers performing workload characterization. Our work contributes the following:

- A contention induction methodology that reduces experiments by 2.6×, average simulation time by 2.2×, and total simulation time by 5.6× while incurring -8.46% average relative error in performance;
- A first-time contention sensitivity characterization of SPEC-based traces, showing 57% of benchmarks are insensitive to cache contention assuming a 5% performance loss is tolerable;
- Case studies analysing architectural techniques under contention — replacement & inclusion performance becomes similar, aggressive prefetching has advantages, & branch prediction accuracy is more important.

II. Motivation

Contention analysis is already complex and the standard practice is to simulate multiple workloads concurrently — often called multi-programmed simulations. If a pair of workloads is not representative, then more than two workloads will need to be run concurrently which increases CPU and memory costs. Not only does the cost of a single experiment go up, but the amount of computing required to ensure that all combinations of workloads have been studied can be prohibitive.

TABLE I: Simulation run-times and experiment sizes

Source of Contention	# Sims.	Wall Clock Time in Hours				
		Avg.	Std. Dev.	Max.	Min.	Total
None	95	1.16	0.55	3.56	0.43	110.50
2nd-Trace	4221	2.81	1.86	23.91	0.94	11861.62
PInTE	**1615**	**1.30**	**0.59**	**5.84**	**0.45**	**2105.29**

The major reasons that multi-programmed simulation takes time is the need for a second source of contention and the need to model the full range of contention. Table I demonstrates this cost by comparing the number of experiments and time required for simulating SPEC 17 speed-based simpoint [49] traces run across three contexts of contention: no contention or Isolation; contention from another trace on another core (2nd-Trace); and system-induced contention from our PinTE framework. The table shows adding a second trace increases

978-1-6654-8798-6/22/$31.00 ©2022 IEEE
DOI 10.1109/IISWC55918.2022.00011

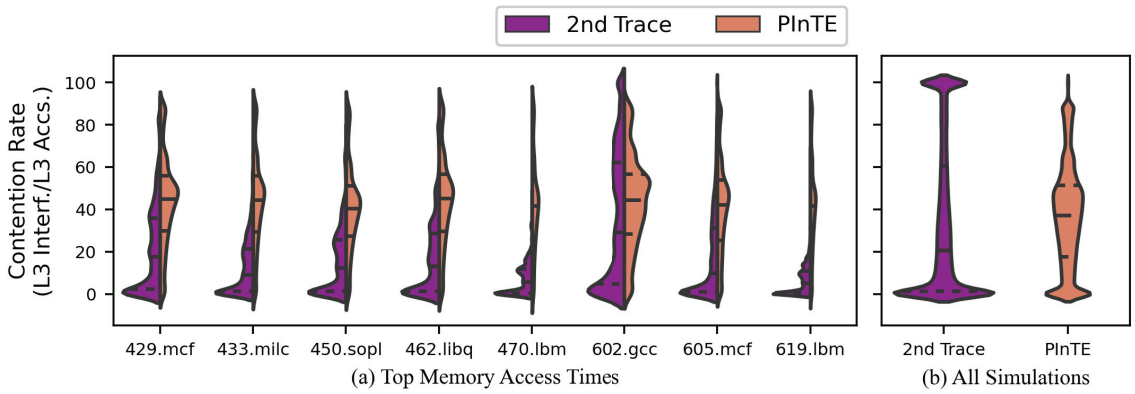


Fig. 1: PInTE Integration Flow Comparison: Contention analysis is difficult to do given difficulties modeling complete set of contention and run-time costs. The y-axis reflects contention rate (number of *thefts* experienced divided by LLC accesses) as a percentage (0-100%); (a) shows contention rate often has an over-representation of low contention (closer to 0 on a scale of 0 to 100%) when workloads share L3, but PInTE can create uniform range of contention rates.

experiments 44×, increases average run time by 2.4×, and total consecutive run time by 107×. PInTE grows experiment size by 17×, sees 12% gain in average run time, and 21× increase in total time.

Simulating all combinations of multi-programmed workloads does not guarantee that the entire range of resource contention has been simulated. As we will show in the case studies, the amount of contention can impact design decisions. Figure 1 shows that using SPEC-based traces as a source of contention only achieves extreme contention behavior. Having said that, we show that controlling contention with PInTE (orange) covers a broad range of contention rates and allows finer-grain contention analysis.

III. Experimental Setup

PInTE aims to simplify the process of contention analysis through system-induced contention. The setup in this section is used to compare PInTE and contention from a 2nd-trace (section IV). We also define the micro-architecture parameters and policies used in the case study described in Section VI.

A. Simulation Environment

The results we report are generated via the ChampSim trace-based simulator [29]. We modify the version of ChampSim used during the 2nd Cache Replacement Competition [20]. It models an Intel Skylake [8] with non-inclusive caches, and has prefetchers available for L1 and L2. For all experiments, LLC is 4MB with 16 way set associativity, and DRAM is configured as 8GB with 2-channels (4GB DIMMs).

B. Benchmark Traces

We analyze 188, 1 Billion (B) instruction simpoint traces [49] made available for the last Data Prefetching Competition [3]. Traces are based on SPEC [51]. For PInTE, we warm cache for 500 million (M) instructions and simulate for 500M instructions. Using default simulators makes it difficult to compare the same portion of a trace. For example, ChampSim runs until all traces have warmed and simulated the same number of instructions, and restarts faster traces until slower traces have finished. Our **2nd-Trace** method runs multi-programmed simulations from start to finish with no warm-up, and we collect data every 10M instructions — we do this for all unique pairs of traces in our set. Segments from instruction 520M to 990M (470M in total) were collected for all results to remove warm-up artifacts from PInTE statistics. Though ChampSim doesn't support multi-threaded workloads, PInTE could be used to induce contention on a multi-threaded workload on another simulator. Multi-threaded workloads naturally cause inter-thread contention which can already be observed, while PInTE causes contention external to the workload.

C. Micro-Architecture Policies

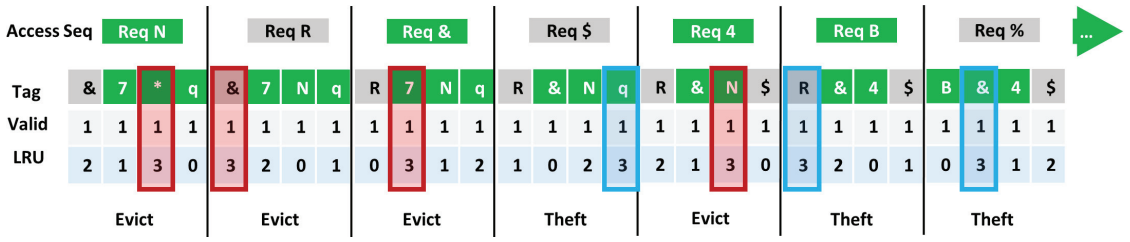
We demonstrate how to evaluate hardware techniques under contention in a case study — see section VI. Here, we highlight the hardware techniques evaluated in that case study.

a) Replacement: We investigate 4 replacement policies: Least Recently Used (LRU) [39]; pseudo Least Recently Used (pLRU) [54]; not Most Recently Used (nMRU) [39]; and Re-Reference Interval Prediction (RRIP) [25].

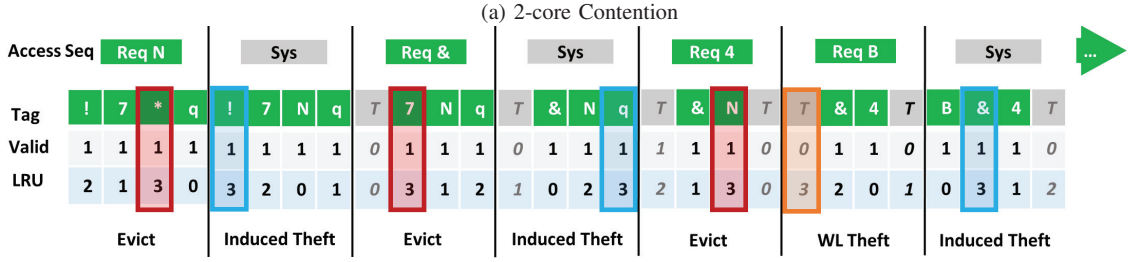
b) Inclusion: Inclusion describes how copies are maintained in cache [4], and we evaluate three: Inclusive (in), Exclusive (ex), and Non-inclusive (no).

c) Prefetching: Prefetchers speculatively fetch data from memory before the workload needs it [13]. Given the availability of next line prefetchers at L1 and L2, as well as an IP stride prefetcher at L2, we simulate traces with four permutations that we represent with a prefetch character string (L1IL1DL2): no prefetching (000); L1 next line (NN0); L1 and L2 next line (NNN); and L1 next line plus L2 IP stride (NNI).

d) Branch Prediction: Cores use branch prediction to speculate, or execute instructions far ahead of what the processor is currently working on, and we evaluate four such predictors: Bimodal [5], G-Share [35], Perceptron [26], and Hashed Perceptron [14].



(a) 2-core Contention



(b) System Induced Contention

Fig. 2: Real vs Induced Block Theft: Thefts [18] induced by another core have mechanics that the system can mimic, and we illustrate this by comparing (a) real contention and (b) induced contention in a 4-way cache set; we see core 1 reflected by a green color scheme and core 2 (adversary) reflected by a gray color scheme, and we distinguish evictions by whether the workload evicts its own data (red), if it evicts the data of another workload (blue), or if the workload mocks a theft (orange); (a) the access pattern results in core 1 experiencing 2 thefts and causing 1 theft; (b) having the system (Sys) represented in grey to reflect a non-workload entity, the workload experiences similar theft evictions while mocking a theft by inserting on a previously invalidated block.

D. Performance Metrics

We leverage metric rates in our analysis to show the impact of cache contention directly: Instructions Per Cycle (IPC); Miss Rate (MR); and Average Memory Access Time (AMAT). Further, we represent performance changes as weighted IPC [12, 41, 45, 53, 55]. We calculate weighted IPC as follows:

$$Weighted\ IPC = \frac{IPC_{contention}}{IPC_{isolation}} \quad (1)$$

such that $IPC_{contention}$ can come from a 2nd-Trace experiment or PInTE, and $IPC_{isolation}$ is from the workload run by itself.

E. Comparing across Similar Contention Rates for Analysis

We evaluate how well PInTE induces a performance response at specific contention rates and compare to the response from the 2nd-Trace method. Due to the size of experiments, we compare workloads across like contention rates — a method we call contention rate grouping (CRG). To do this, we group contention rates in +/-5% sub-ranges (e.g. 0 to 5% contention rate) by rounding the observed contention rates from each experiment to the nearest 10%. The group was chosen because it trades error for experiment coverage, which we demonstrate in section IV-E4. We note when we apply CRG in our analysis.

IV. PInTE: Probabilistic Induction of Theft Evictions

PInTE provides a means for simulation environments to induce last level cache contention in a controlled way. Last level cache contention manifests as inter-core evictions, or **thefts** that have simple mechanics that PInTE emulates.

A. What's a Theft?

A **Theft**, first introduced in CASHT [18], happens when one core evicts valid data on a miss that was originally inserted by another core. The resulting inter-core eviction is illustrated in Figure 2a. This figure shows the interleaving access streams of two workloads that share a 4-way cache set happening from left to right. By the end of the stream the green workload caused one theft and experienced contention, or interference two times. Consequently, the gray workload causes 3 thefts and sees interference once. A similar sequence is shown in figure 2b, but with two key differences: the green workload runs alone; and the system invalidates data in line mimicking the gray workload. By facilitating thefts with block invalidation and promoting the invalid block as if it were a normal access, the cache simulation can be modified to induce thefts!

B. Limitations

Causing contention by evicting cache lines is not a perfect proxy for all contention a workload may experience, and here we talk about PInTE's limitations: only exists in LLC; access patterns aren't exact; and inducing contention in hardware is hard. First, implementing PInTE in the last level cache means it only causes contention in LLC. Though PInTE creates traffic to DRAM due to dirty evictions, this is not a sufficient source of DRAM or bandwidth contention. Second, PInTE achieves contention rates set at the start of the simulation, but does not replicate precise access patterns. Enumerating access patterns adds a new dimension to the design space, which PInTE is not bound to — this keeps our experiment size small. Finally,

PInTE requires changes to the replacement policy and the addition of hardware counters to be properly evaluated in hardware. Currently, secondary workloads are leveraged to cause contention, and there aren't many efforts to motivate contention counter adoption (like this work).

C. Configuration & High-Level Flow

PInTE integrates into the last level cache, uses existing function calls (block update, promotion, eviction), and is designed to yield configurable contention rates by setting a threshold at the start of simulation: Probability of Induction, or P_{Induce}. This is a proxy for the probability that contention occurs; P_{Induce} sits in the range from 0 to 1. PInTE uses P_{Induce} to decide when to cause contention, and Figure 4 illustrates how PInTE operates after each access to LLC. The flow starts with **UPDATE-ACCESS** which updates the currently accessed block before transitioning to **GEN-PROBABILITY**. **GEN-PROBABILITY** computes a contention trigger ratio:

$$Trigger\ Ratio = \frac{Random\ Number}{Max\ Random\ Number} \quad (2)$$

to determine the next state: if the ratio is larger than P_{Induce}, then PInTE exits; otherwise, PInTE transitions to **GEN-EVICT-CNT**. **GEN-EVICT-CNT** generates a new random number called $Block_{evict}$ (bounded between 0 and associativity) and represents the number of contention events to induce. **GEN-EVICT-CNT** further initializes a way counter (w) to 0 before transitioning to **BLOCK-SELECT**. **BLOCK-SELECT** sets a pointer to the current block (blk) and determines the next state: move to PROMOTE if the current block is at the end of the replacement stack; or it moves to increment w by 1 and exits if the set has been exhausted. **PROMOTE** updates the block position in the replacement stack, and transitions to **INVALIDATE** if the current block is valid, but will transition to DECREMENT otherwise. **INVALIDATE** sets the valid bit for *blk* to 0, adds blk to the write-back queue if it's dirty, and then goes to **DECREMENT**. **DECREMENT** reduces $Blocks_{evict}$ by 1, and either exits if $Block_{evict}$ is 0 or returns to **BLOCK-SELECT**.

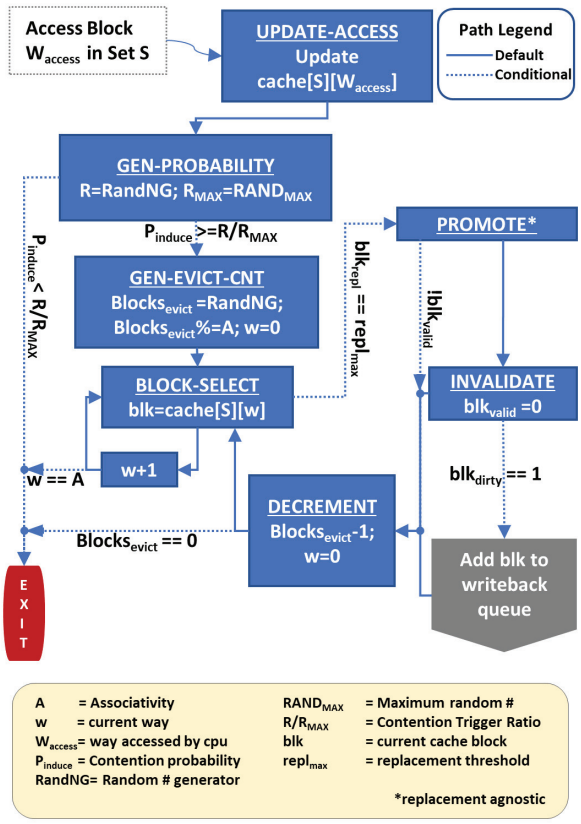


Fig. 4: PInTE Flow Diagram

D. Stability Analysis

PInTE is triggered randomly on any given access to LLC which means re-runs of the same experiment won't have the same contention events. Figure 3 shows how high-level metrics vary between PInTE iterations — we run 12 PInTE experiments per SPEC trace 25 times each. Figure data is the normalized standard deviation which we calculate as follows:

$$Normalized\ Standard\ Deviation = \frac{Standard\ Deviation}{Mean} \quad (3)$$

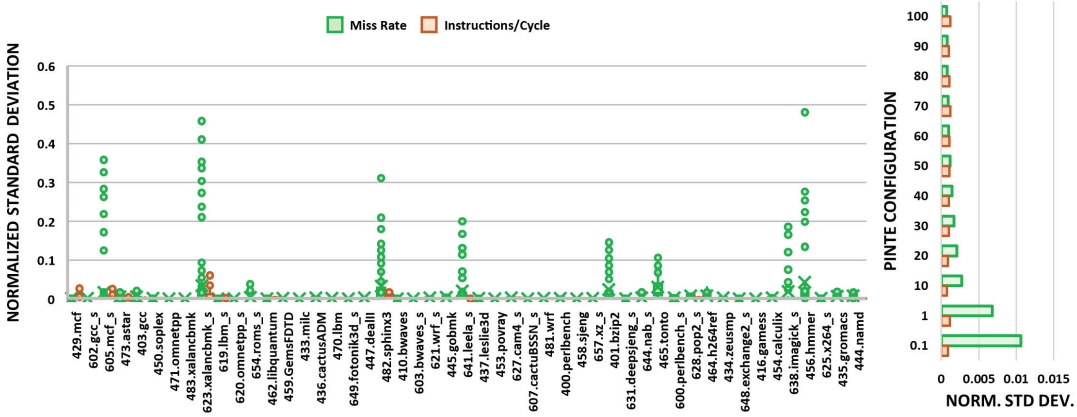


Fig. 3: PInTE Stability Analysis: PInTE is stable across 25 runs of 12 P_{Induce} configurations; Miss are (green) and IPC (orange) are represented; (left) x lists benchmarks, y shows normalized standard deviation (norm. to mean); (right) x shows norm standard deviation, y shows P_{Induce} configurations.

Additionally, the right of figure 3 shows the same metric but as a function of P_{Induce} configurations. On the left, the median standard deviation for miss rate and instructions per cycle are near 0 for all workloads, and the proximity of the whiskers suggest low variation (less than 0.01). The right plot shows PInTE is stable as induced theft evictions increase (<0.00125 and <0.011 for miss rate and IPC respectively). Clearly, because of the large numbers involved, PInTE does not have high variation between re-runs of experiments and can be trusted enough after one simulation.

E. Approximation Validation

PInTE generalizes contention from a *2nd workload* as cache evictions that occur at a given rate, and this section validates that approach. A reminder, there are limitations of the 2nd workload approach that PInTE overcomes such as simulation time, number of simulations and covering a range of contention.

1) Approximation Measurements: Our validation uses two methods to evaluate the PInTE approximation: relative error and distribution distance. To evaluate high-level metrics, we compute relative error as follows:

$$Relative\ Error_m = 100.0 \times \frac{m_{2nd-Trace} - m_{PInTE}}{m_{PInTE}} \quad (4)$$

where m is a high-level metric like IPC. Relative error shows discrepancies in performance outcomes. For fine-grained behavior like memory reuse and run-time samples, we leverage Kullback–Leibler (KL) divergence to measure the statistical distance between probability distributions [31]. KL Divergence is computed as follows:

$$D_{KL}(p(x)||q(x)) = \sum_{x \in X} p(x) \times log_2(\frac{p(x)}{q(x)}) \quad (5)$$

where x represents distribution buckets, $p(x)$ is an observed probability distribution, and $q(x)$ is a reference probability distribution. We use log-base-2 KL divergence to measure the distance between PInTE simulation and 2nd-Trace contention in bits. Measuring distance in bits allows us to reason about information entropy in terms of missing state between two models. A KL divergence of 0 means the two probability distributions are identical while a result of 2 means $q(x)$ needs 2 more bits to correctly encode $p(x)$.

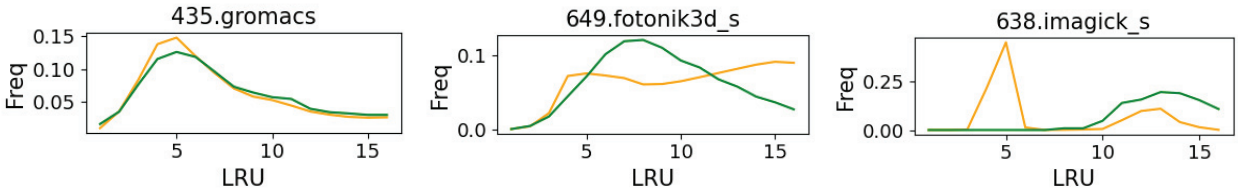
2) Relative Error in High-Level Metrics: In this section, we compare the workload performance response to PInTE and 2nd-Trace contention using equation 4. Table II shows average relative error for the benchmarks represented by our traces, and we divide the table into SPEC 2006, SPEC 2017 speed, and average. Positive error means PInTE underestimates, while a negative error means PInTE overestimates. We grade relative error as significant if the value exceeds +/-10% of the 2nd-Trace results. That said, we leave it to designers to employ their own criteria.

a) Error Analysis: PInTE incurs 1.43%, 1.29%, and –8.46% average relative error respective of AMAT, MR, and IPC. High AMAT and IPC relative error indicates workloads with high DRAM dependency beyond LLC — we underline such benchmarks in Table II. High MR error alone suggests LLC access is infrequent, which implies the workload is core-bound — we annotate such workloads with an asterisk (*). Workloads with high relative error in IPC alone are LLC-bound workloads (AMAT for these are between L2 and LLC latency). LLC-bound workloads become DRAM bound under increased contention — annotated with a plus (+). Despite outliers, PInTE yields average relative error comparable to modern simulators [6].

TABLE II: Average Relative Error in High-Level Performance Metrics: We compare the Average Memory Access Time (AMAT), Miss Rate (MR), and Instructions per Cycle (IPC) collected from 2nd-Trace simulations and our PInTE engine and methodology. The data in the table reflect the average relative error between PInTE and 2nd-Trace results. *KEY*: AMAT & IPC $\geq$ 10%, *MR error $\geq$ 10%, +IPC Error $\geq$ 10%.

Benchmark	AMAT	MR	IPC	Benchmark	AMAT	MR	IPC
400.perlbench	-0.14	-1.13	-0.27	600.perlbench	-0.04	1.94	-0.37
401.bzip2	-1.17	-0.24	-2.35	602.gcc	31.77	0.5	-42.65
403.gcc	-0.91	-1.54	-6.16	603.bwaves	5.82	-3.04	-4.84
410.bwaves	3.91	1.58	-6.17	+605.mcf	6.69	1.46	-33.79
416.gamess	-0.02	0.31	0.25	607.cactuBSSN	1.89	-0.67	-2.71
+429.mcf	0.06	-0.47	-71.53	619.lbm	-1.22	-5.3	-7.57
+433.milc	2.52	-0.72	-13.44	620.omnetpp	-2.38	-0.47	-3.29
434.zeusmp	-0.08	-0.78	-6.29	621.wrf	-0.15	-4.14	-5.95
435.gromacs	-0.34	-1.46	-1.9	623.xalancbmk	6.11	-0.21	-7.92
436.cactusADM	-0.75	0.55	-1.8	625.x264	-0.75	-4.8	-0.71
437.leslie3d	-1.92	-1.08	-6.19	627.cam4	-1.12	-1.68	-5.1
444.namd	0.08	6.85	-0.16	628.pop2	-0.45	-3.29	-4.79
445.gobmk	-0.06	1.48	-0.07	631.deepsjeng	-0.04	0.32	0
447.dealII	-0.89	-1.68	-0.96	*638,imagick	0.07	21.22	-0.37
+450.soplex	-1.15	-2.16	-17.07	*641.leela	-0.06	19.19	-0.14
453.povray	-0.01	0.1	-0.03	644.nab	0.6	0.09	-2.01
454.calculix	0.49	-2.08	-2.73	648.exchange2	0	0	0.01
*456.hmmer	0.16	12.67	-4.6	649.fotonik3d	0.19	-0.42	-4.59
458.sjeng	-0.12	1.44	-0.04	654.roms	-0.17	3.95	-7.19
459.GemsFDTD	-2.03	-2.08	-5.38	657.xz	1.17	2.22	-1.22
462.libquantum	12.24	-4.29	-25.94				
464.h264ref	-0.22	5.26	-0.92	**2006**	0.76	1.26	-9.63
*465.tonto	0.01	30.13	0.01	**2017**	2.4	1.34	-6.76
470.lbm	-1.34	-4.28	-9.45	**All**	1.43	1.29	-8.46
+471.omnetpp	-3.8	-1.99	-14.55				
+473.astar	-4.75	-6.8	-48.23				
481.wrf	-1.58	-2.42	-6.93				
482.sphinx3	16.03	11.04	-15.95				
+483.xalancbmk	7.73	0.28	-10.45				

b) Limitation Contributions to Error: The errors we see in table II are a consequence of PInTE's limitations that we discuss in section IV-B. The 6% of benchmarks with high AMAT & IPC error experience this because PInTE is limited to LLC. Real workloads cause contention at LLC and beyond (for example, at DRAM), so PInTE can be an insufficient source of contention — increasing DRAM access costs could complement this. The 8% of our benchmarks with high MR error also have AMAT below core-cache latency (figure 9), so access to LLC that trigger PInTE are infrequent — an independent PInTE module could avoid this. Lastly, the 12% with high IPC error are looking for precise behaviors that PInTE eschews for contention rates — mocking specific access behavior could address this. We leave extending PInTE beyond the LLC to future work.



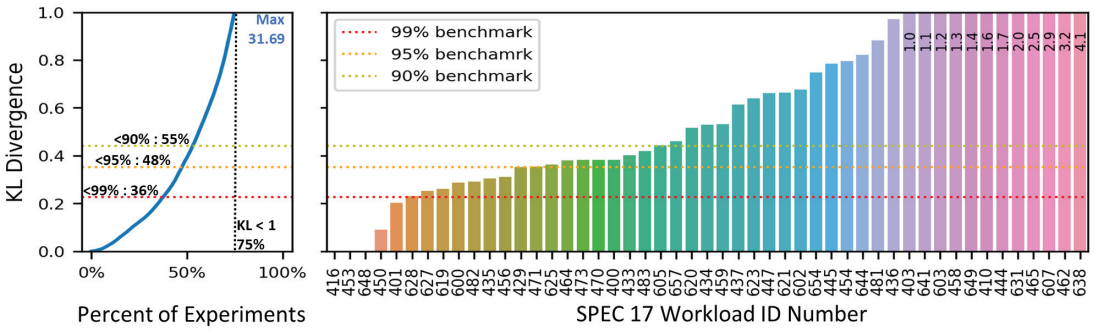
(a) Contention rate: 14.6%
KL divergence:0.013

(b) Contention rate: 62.35%
KL divergence:0.202

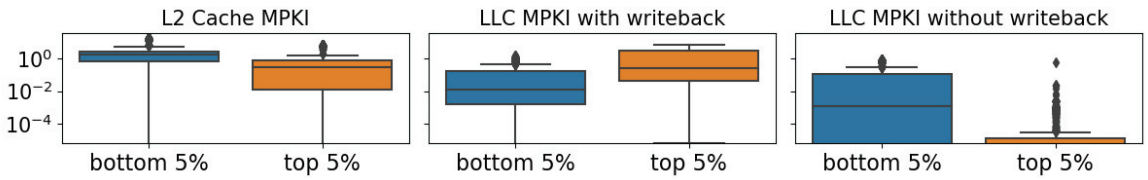
(c) Contention rate: 1.26%
KL divergence:2.332

Fig. 5: Reuse Behavior under PInTE vs 2nd-Trace contention: We visually compare reuse histograms under different sources of contention: PInTE (yellow) and 2nd-Trace (green); (a) gromacs yields a near match (good alignment), and the lowest KL divergence of the three; (b) fotonik3d_s shows some overlapping behavior with imperfections (medium alignment), and the KL divergence is 20× the result of the good alignment case; (c) imagick_s shows clear differences (worst alignment), and KL divergence $>$ 200× the good case to match this misalignment. The figure shows that even when KL divergence is high, PInTE is still able to capture a general or a partial reuse trend.

3) Approximating In-cache Behavior Using Reuse Distance Histogram Comparison Quantified with KL Divergence: We compare reuse histograms for workloads under PInTE and the 2nd-Trace method, and validate the induced contention behavior using equation 5. Whenever a reuse of a cache block occurs in the LLC, we increment the counter of the corresponding LRU hit position in the histogram. We performed this experiment on all benchmarks and contention pairs. The reuse hit histogram is sampled every 10M instructions in the region of interest from 500M to 1B instructions. Given that the run-time histogram data is relatively stable, we choose to use an average reuse histogram for plotting and comparison purposes. Figure 5 shows three plots that compare reuse histograms under PInTE and 2nd-Trace contention. Leveraging equation 5, we compute KL divergence between the PInTE hit histogram distribution ($q(x)$) and real contention hit histogram distribution ($p(x)$) where x is the associated hit positions. Figure 6a shows the average KL divergence for each benchmark. In summary, the PInTE simulated LLC reuse hit histogram distribution is 0.84 bits of information away from the reuse hit histogram generated by real contention. To calibrate and benchmark the KL divergence metrics, we established benchmarks based on randomly-generated distributions. For example, we find that 99% of a randomly-generated distribution has KL divergence greater than 0.26 when comparing to the real contention reuse histogram. Therefore, we denote 0.23, 0.35, and 0.44 (horizontal lines in figure 6a) as the 99%, 95%, and 90% benchmarks, respectively. Our experiments shows that 36%, 48%, and 55% of workloads are within the 99, 95, and 90% benchmarks. To explain why some workloads have relative high KL divergence, we plot the L2 and LLC MPKI for the workloads that have the highest or lowest KL divergence in

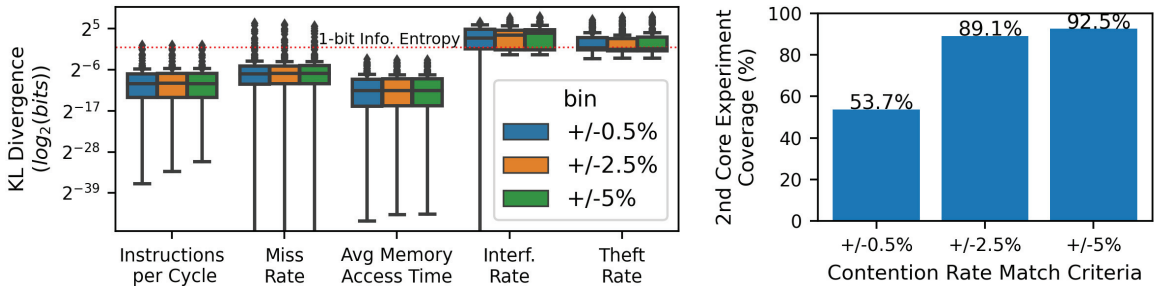


(a) Average KL divergence per benchmark



(b) MPKI of workloads with High vs Low KL divergence

Fig. 6: Benchmark Reuse KL-Divergence & Worst-case Root-cause Analysis: (a) shows KL Divergence (KLDiv) between reuse histograms under PInTE- and 2nd-Trace-contention; y-axis is KLDiv in bits; x-axes show the percent of experiments (left) and distinct workloads (right); horizontals show N% benchmark bounds that note the percent of experiments covered; (b) compares cache results for high & low KLDiv, showing high KLDiv corresponds to L3 writeback traffic (i.e. L2 activity spilling).



(a) KL Divergence Yields <1bit Information Entropy for Key Metrics (b) Varying the Contention Rate Match Granularity

Fig. 7: Entropy between 2nd-Trace- and PInTE-influenced Run-time Behavior is low: 5 metrics see sub 1-bit information entropy; the x-axis shows the metrics and information entropy in bits on the y-axis, and data is represented as box-plots — the three box-plots correspond to our CRG criteria; (a) KL divergence between PInTE- and 2nd-Trace-induced run-time metrics is shown to be <1 bit information entropy; (b) shows the number of experiments covered by different CRG criteria.

Figure 6b. The result shows that high KL divergence workloads tend to be core-bound given that most activity occurs in private caches and most LLC misses are caused by write-back (i.e. L2 spills). This correlation makes sense: if the misses caused by contention are low for a workload, then a workload's in-cache behavior is not dominated by contention, therefore making it hard for PInTE to simulate. However, in the above cases, the high-level performance metrics are often well-modeled given that its IPC, MR and AMAT do not heavily rely on the LLC activities. For example, in the case of 638.imagick_s, having a high KL divergence of 4.1 suggests a poor modeling of the in-cache reuse distance behavior, but its IPC is within -0.37% of the real contention result.

4) Approximating Run-time Metric Response Similarity with KL Divergence: We use sequential run-time data as inputs to equation 5 to determine how closely PInTE and 2nd-Trace contention yields similar, dynamic behavior responses: sequential 10M instruction samples are x; metrics collected at each sample under real contention (2nd-Trace results) are $p(x)$; and metrics sampled under PInTE are $q(x)$. Figure 7a shows the information distance between run-time results for high-level metrics is low ($<< 1$) which implies the dynamic response to PInTE contention is similar to using a 2nd-Trace. PInTE incurs high interference and theft rates (interference and thefts per LLC access) because it is not designed to achieve an accurate modeling of 2nd-Trace access patterns. Figure 7b complements the distance results by showing how much of our 2nd-Trace results we found a match for in PInTE. PInTE is able to cover ~92% of 2nd-Trace results with contention rates within 5% at 7.79× fewer experiments (2nd-Trace = $188 \times \frac{187}{2} = 17578$ mixes vs PInTE = 12 PInTE configurations ×188 traces = 2256 experiments).

V. Characterizing Cache Contention Sensitivity

PInTE speeds up and simplifies contention sensitivity analysis. This section, for the first-time, presents a contention sensitivity characterization of SPEC benchmarks.

A. Experimental Design

The following definitions were used to generate data in and analysis of contention sensitivity shown in figure 8:

- Running Context - distinguishes whether workloads cache data in the absence (*isolation*) or presence of *contention*;
- Tolerable Performance Loss, TPL - threshold for changes in IPC between the isolation and contention contexts to determine sensitivity — workloads with service level agreements require minimum performance guarantees; though not shown, we evaluated 1%, 5%, and 10%; 5% yields reasonable sensitivity classification.
- Contention Curve - average weighted IPC as a function of contention rate groups.
- Capacity Curve Analysis & Feature Extraction, C^2AFE [19] - a tool that summarizes curves into 3 features (knee, trend, and sensitivity) — we use a method in C^2AFE to characterize contention sensitivity.
- Sensitive-Curve Population, SCP - the percentage of a workload's contention curves that are sensitive.

We complement the contention curves with Average Memory Access Time (AMAT) in Figure 9. AMAT results show PInTE also induces memory latency similar to 2nd-Trace latency.

B. Contention Sensitivity Characterization

We classify the cache contention sensitivity of each SPEC benchmark into one of three groups: high, low, and mixed assuming a 5% TPL. Benchmarks are **high sensitivity** (red border) when at least 75% of instruction samples see 5% or more difference from isolation IPC. Workloads in this class include 450.soplex, 456.hmmer, 470.lbm, 471.omnetpp, 482.sphinx3, and 619.lbm which represent 12% of our benchmarks. Benchmarks with **low sensitivity** (gray plot area) see no more than 25% of samples with changes in IPC that exceed the 5% TPL — 57% of our benchmarks have low sensitivity. Such benchmarks have very little to no change in IPC and can be considered largely insensitive to LLC contention. Workloads that have a **mixed sensitivity** (white plot area) fall between our extreme characterization criteria. Workloads in this group include 401.bzip2, 403.gcc, 459.GemsFDTD, 464.6264ref, 605.mcf, 621.wrf, 623.xalancbmk, 627.cam4, and 623.pop2 which make up 16% of our benchmarks. The dip in performance at middle contention rates and higher performance at extremes suggests such workloads may prefer to bypass the LLC.

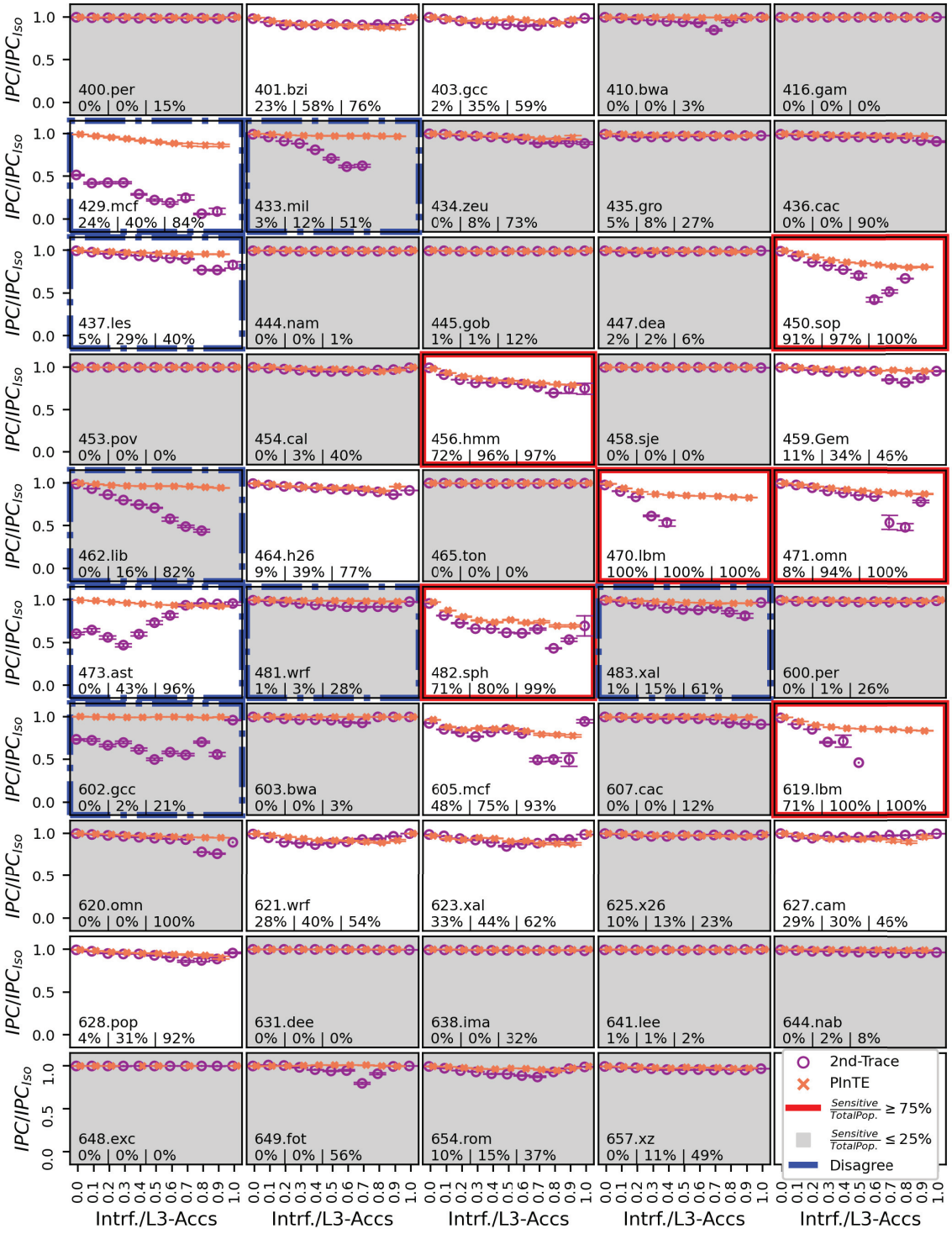


Fig. 8: Contention Sensitivity Curves: Comparison and Classification of PInTE (orange, x) and 2nd-Trace (purple, dot) contention sensitivity curves; x-axis shows interference rate while y-axis shows weighted IPC; Subplots have benchmark names and Sensitivity-Curve Population data embedded within the plot; benchmark characterization includes high sensitivity (red border), low sensitivity (gray), or mixed (white) according to a 5% Tolerable Performance Loss; blue dotted borders indicate benchmarks with empirical disagreements between 2nd-Trace and PInTE classification; PInTE approximates clear insensitive and sensitive behavior, but has trouble inducing comparable behaviors for noted core-bound and DRAM-bound benchmarks.

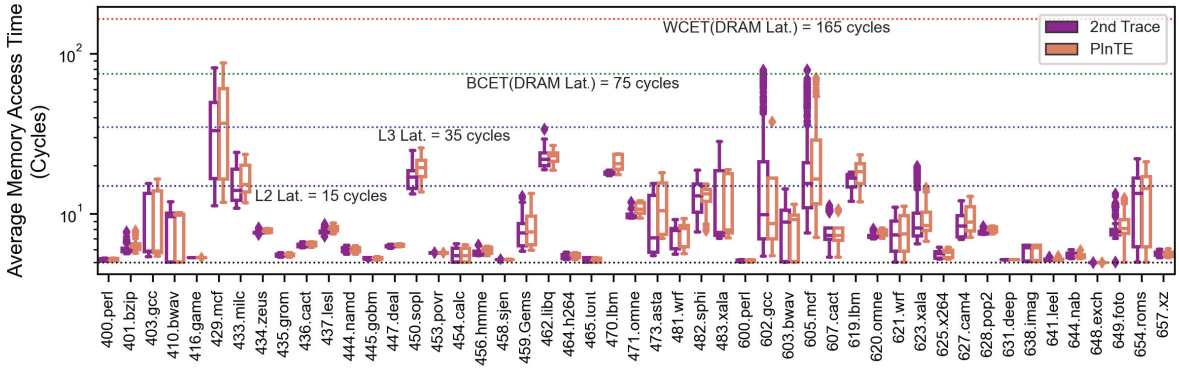


Fig. 9: Average Memory Access Time under Contention: Comparison of 2nd-Trace (purple) and PInTE (orange) AMAT results; x-axis represents benchmarks, and y-axis shows average memory access time; we represent AMAT per 10M instruction sample as boxplots, indicating median, upper and lower quartile (top and bottom of the box, respectively), max and min (high and low whiskers, respectively), and outliers beyond the normal distribution of data (diamonds); PInTE is shown to induce AMAT similar to when traces share LLC and beyond, save for a few understandable results where workloads appear DRAM bound (429.mcf, 602.gcc for example).

C. Disagreement Cases

There are workloads where PInTE does not match the expected behavior demonstrated by the 2nd-Trace contention curves (blue, segmented border). These disagreements are largely due to the memory (DRAM) bounded nature of these benchmarks: 429.mcf, 433.milc, 437.leslie3d, 462.libquantum, 473.astar, 481.wrf, 483.xalancbmk, and 602.gcc. Two of these workloads (429.mcf, 602.gcc) are in fact likely to be DRAM bound given the AMAT shown in figure 9 which indicates they approach DRAM latency. The remainder appear to have some AMAT data that exceeds L2 latency which could indicate some requests find delays in LLC and DRAM. Inconsistencies suggest key cached data blocks are regularly forced out or delayed closer to DRAM by a second workload, while PInTE only causes contention in the L3.

D. Contention in a Real System

We discuss how PInTE compares to real system contention in this section. We cannot do a direct comparison to a real system due to the lack of contention counters, but we can calculate what we call *change in occupancy* as our proxy:

$$\textit{Change In Occupancy} = 100 \times \left(\frac{\textit{Current LLC Occupancy}_w}{\textit{Maximum LLC Allocation}} - 1\right) \quad (6)$$

where w is the workload. Measuring contention in this way represents loss from expected capacity and is like coarse-grained thefts. Figure 10 compares results of a subset of SPEC 17 Rate benchmarks (a) to equivalent simpoint traces (b) [49] — prior work reports similarity between different variants of SPEC benchmarks to allow comparison [38]. Benchmarks are run on an Intel Xeon Silver 4110 @ 2.1GHz with 11MB LLC and 65GB DRAM in pairs to induce contention. We set cache allocation size to 10MB for our experiments and 1MB for the remaining system processes via Intel RDT [27]. Cache, prefetching, and DRAM are modeled in ChampSim to approximate our system, and we halve key DRAM features

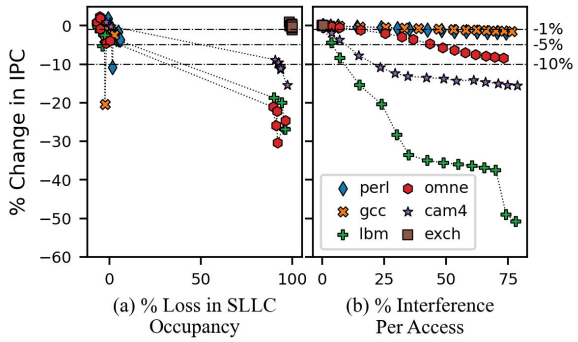


Fig. 10: Real vs PInTE Contention: We evaluate 6 SPEC 17 benchmarks on an Intel Xeon Silver 4110-based Server; y-axis shows the percent change in IPC; dotted horizontal lines denote 1, 5, & 10% differences from the lowest contention case; (a) we run each benchmark concurrently with all other SPEC 17 benchmarks, and sample metrics every second until they complete once; equation 6 is our x-axis; (b) we modify ChampSim to model the server and evaluate related simpoint traces that we apply respective weights to according to the simpoint method; Interference Rate is our x-axis.

(ranks, banks, columns, transfer rate) to facilitate contention off-chip that PInTE does not model. Performance losses for lbm and cam4 are larger than in the real system — controlled contention and higher DRAM costs allow this. Perlbench and gcc show performance within a few percent save for a few outliers. Omnetpp sees comparable trends between PInTE and real results but the impact to performance differs — PInTE can benefit from precise LLC and DRAM contention modeling here. Exchange results suggest it is insensitive, but contention measures are on opposite ends — this is due to exchange having low LLC occupancy and utilization. In summary, PInTE is a valuable tool for controlled and lightweight contention analysis that yields results that are comparable to a real system.

VI. Case Study: Analyzing Prevailing Architecture Design Choices under Contention

Designers choose and tune architecture techniques based on rigorous simulation studies, but to our knowledge there are no studies which test these decisions under increasing cache contention. We explore how resilient the techniques discussed in section III-C are under PInTE contention. We find that the performance advantages measured in isolation are difficult to maintain as contention increases.

a) Figure Setup: Figure 11 shows changes to a technique's advantage as contention increases. Rows reflect architectural logic: replacement, inclusion, prefetching, and branching. Each sample is a different workload trace. Columns reflect the following: Win percentage, or the percent of times a technique was the best option; a primary metric of comparison between techniques (L3 Miss rate, Interference Rate, Prefetcher Miss Rate, and Branch Prediction Accuracy); a secondary metric of comparison; and tie percentage (all are within 1% of each other in black, or there's more than one good solution in red).

b) Replacement: For replacement (column 1, row 1), we see that pLRU (blue) has an advantage at low contention but loses to RRIP up to P_{Induce} configuration 7.5. Beyond 7.5, nMRU dominates until RRIP increases its share up to configuration 70, after which we see LRU rise to the top. The changes are surprising, but inspecting the tie percentage (column 4, row 1) shows at least 50% of results are statistical

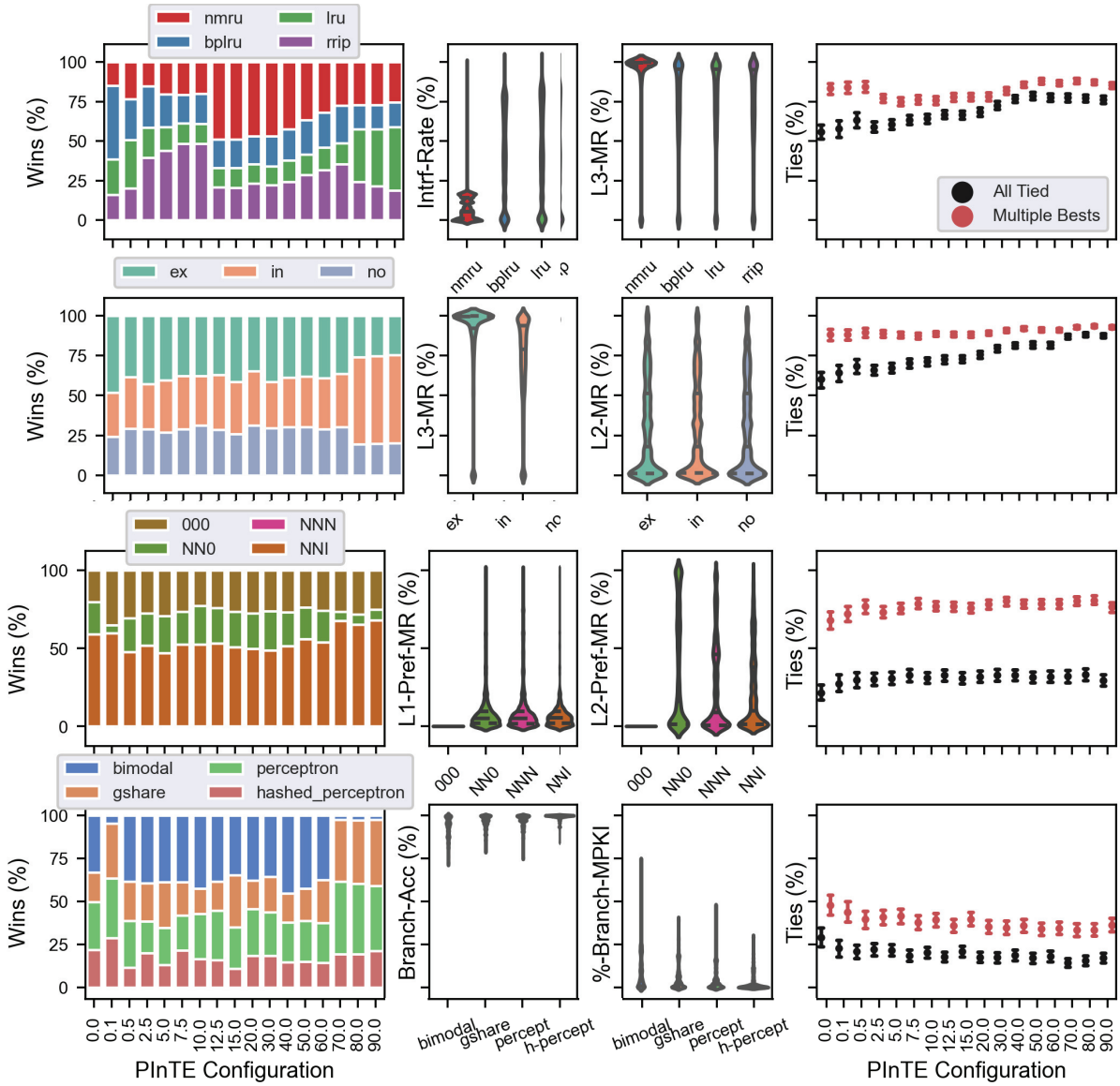


Fig. 11: The best design choice varies with contention: We study what design choices the performance results say are the best as contention increases; column 1 shows the percentage distribution of which policies have the max performance, or win; column 2 shows primary statistics used to evaluate respective policy efficacy; column 3 shows secondary statistics that complement column 2; and column 4 shows whether the "win" is exclusive.

ties (within 1% of each other). The increase in ties with contention suggests a specific micro-architecture technique's advantages are absorbed in a highly utilized and shared LLC. Further, interference and miss rates imply that differences in policy priorities yield different effects on cache stats. For example, reuse stack policies (LRU, pLRU, RRIP) are subject to contention-induced data movement while recency policies (nMRU) are subject to contention frequency.

c) Inclusion: Inclusion properties have a smoother transition, with exclusive caches demonstrating an advantage at lower contention rate while inclusive caches have an advantage at higher contention rates. The percentage of ties varies in a similar way to that of replacement policies, with advantages being diminished as contention increases. The miss rates for L2 and LLC (columns 2 and 3) highlight that an exclusive cache has an high representation of high LLC miss rates vs inclusion, but L2 miss rate sees no difference.

d) Speculation - Prefetching and Branch Prediction: Speculative techniques like prefetching and branch prediction are pretty stable as contention increases. For prefetching (row 3), NNI is the favorite. L1 miss rate is similar across different configurations, and higher L2 prefetch miss rate for NNO suggests L2 cannot keep up with prefetch pressure from L1 without its own prefetcher. Trends are relatively flat despite a mild increase, indicating prefetcher advantages are consistent despite contention. For branch prediction (row 4), perceptron maintains a consistent portion of wins until 70% contention rate where its representation increases. Though hashed perceptron is the most accurate, that does not translate to a performance advantage as contention increases. Further, ties decrease with contention which suggests effective predictors are important as miss criticality grows under contention.

VII. Related Work

In this section, we discuss prior work on other sources of contention, alternate contention measurement methods, contention induction for controlled study, contention-aware architecture, and faster contention analysis.

a) Contention Analysis: Numerous contention analysis methods exist that target shared components of the architecture: Bandwidth [11, 15, 16, 57]; DRAM [22, 23]; Memory Controller [43]; Queues [2, 17]; Multi-threaded [1, 10, 59]. PInTE targets last level cache, and enables the system to induce inter-core cache evictions external to a workload's behavior.

b) Inducing Contention: There exist techniques to induce contention in last level cache: use a second workload to cause contention [34]; estimate individual workload needs in isolation and approximate the shared behavior [21, 56]; and create artificial scarcity [21, 42]. Secondary workloads require a second core, estimations require individual profiles be built prior to contention estimation, and artificial scarcity is not the same as stochastic events from a secondary source.

c) Measuring & Estimating Contention in Last Level Cache: Proposed methods and metrics leveraged to observe contention include: changes in LLC occupancy correlated to MPKI or CPI, which is indirect [48]; applying linear regression to reuse distance histograms for workloads in contention and isolation [56]; using data movement towards eviction as a miss rate proxy, and reports that approximate miss rate curves can sum to approximate a shared curve for contention analysis [12, 36]. We use inter-core evictions, or *thefts* [18] which are a direct consequence of contention as a metric.

d) Contention in Design: Contention drives designers to figure out how to avoid it or weigh how much impact contention has on performance. Thread awareness includes distinct profiling logic per workload [25, 40, 44, 58], with the intent to avoid individual thread decisions being influenced by others. Physical partitioning splits cache into segments per workload (or group) [12, 24, 30, 41] to guarantee capacity. Alternatively, pseudo partitioning maps partition size to reuse distance or time in cache to enforce the partition limit [45, 53, 55]. Lastly, recent work uses thefts to partition LLC, and is comparable to UCP but at a fraction of the cost [18]. PInTE is not an architecture technique but a methodology that enables designers to analyze and design contention-aware architectures and applications.

e) Multi-core Simulation & Analysis: Multi-core evaluation is speeding up and this is mostly due to exploiting parallelism on the system that runs the simulation [7, 32, 46]. Exploiting parallelism assumes multiple cores are available, which is only a certainty in private systems or in larger compute clusters where researchers have resource guarantees. Alternative methods complement traditional models with predictive estimations [47] or leverage FPGAs to speed up evaluation of more complex cores [28]. PInTE models contention events in a single workload simulation rather than modeling additional cores for multi-programmed simulation. By inducing contention at a configured rate, PInTE allows characterization and architectural analysis under contention for the cost of simulating a single core. Further, PInTE resides in LLC and can be implemented in the shared cache of multi-core simulators for multi-threaded workload evaluation.

VIII. Conclusion

Resource contention in modern systems is prevalent. We believe the impact of cache contention should be modeled during any design process or research study. We present Probabilistic Induction of Theft Evictions, or PInTE, a method to allow designers to cover a wide range of contention at 2.6× less run time and 5.6× fewer experiments for SPEC 17-Speed-based traces — 7.79× fewer experiments when conducting an exhaustive multi-core experiment for 188 SPEC-based traces. PInTE incurs -8.46% error in IPC in comparison to performance observed under contention from a 2nd-Trace. Further, low information distance (0.03 bit distance in IPC behavior) shows that PInTE contention induces dynamic and reuse behavior comparable to behavior seen under real contention. Lastly, case studies of micro-architectural techniques run with PInTE reveal a significant impact on the presumed benefits of a technique under contention. LLC-specific techniques see performance wane and look similar, while speculative techniques maintain or increase advantage due to heightened miss criticality.

REFERENCES

[1] U. A. Acar, N. Ben-David, and M. Rainey, "Contention in structured concurrency: Provably efficient dynamic non-zero indicators for nested parallelism," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 75–88. [Online]. Available: https://doi.org/10.1145/3018743.3018762

[2] ——, "Contention in structured concurrency: Provably efficient dynamic non-zero indicators for nested parallelism," *SIGPLAN Not.*, vol. 52, no. 8, p. 75–88, jan 2017. [Online]. Available: https://doi.org/10.1145/3155284.3018762

[3] A. Alameldeen, S. Pugsley, M. Ferdman, M. Dinani, Z. Chisti, P. Gratz, M. Huang, A. Jain, N. Jerger, A. Jaleel, P. Michaud, A. Nori, S. Somogyi, C.-J. Wu, and H. Zhou, "Data prefetching competition 3," https://dpc3.compas.cs.stonybrook.edu/, 2019.

[4] L. Backes and D. A. Jiménez, "The impact of cache inclusion policies on cache management techniques," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 428–438. [Online]. Available: https://doi.org/10.1145/3357526.3357547

[5] I. Bate and R. Reutemann, "Efficient integration of bimodal branch prediction and pipeline analysis," in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, 2005, pp. 39–44.

[6] T. E. Carlson, W. Heirman, and L. Eeckhout, "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*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 52:1–52:12. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063454

[7] D. Chiou, H. Angepat, N. Patil, and D. Sunwoo, "Accurate functional-first multicore simulators," *IEEE Comput. Archit. Lett.*, vol. 8, no. 2, p. 64–67, jul 2009. [Online]. Available: https://doi.org/10.1109/L-CA.2009.44

[8] J. Doweck, W. F. Kao, A. K. y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation intel core: New microarchitecture code-named skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, 03 2017.

[9] C. Dwork, M. Herlihy, and O. Waarts, "Contention in shared memory algorithms," *J. ACM*, vol. 44, no. 6, p. 779–805, nov 1997. [Online]. Available: https://doi.org/10.1145/268999.269000

[10] A. Eizenberg, S. Hu, G. Pokam, and J. Devietti, "Remix: Online detection and repair of cache contention for the jvm," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 251–265. [Online]. Available: https://doi.org/10.1145/2908080.2908090

[11] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten, "Bandwidth Bandit: Quantitative characterization of memory contention," *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 1–10, 2013.

[12] N. El-Sayed, A. Mukkara, P. Tsai, H. Kasture, X. Ma, and D. Sanchez, "KPart: A hybrid cache partitioning-sharing technique for commodity multicores," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 02 2018, pp. 104–117.

[13] B. Falsafi and T. F. Wenisch, "A primer on hardware prefetching," *Synthesis Lectures on Computer Architecture*, vol. 9, no. 1, pp. 1–67, 2014. [Online]. Available: https://doi.org/10.2200/S00581ED1V01Y201405CAC028

[14] A. S. Fong and C. Ho, "Global/local hashed perceptron branch prediction," in *Fifth International Conference on Information Technology: New Generations (itng 2008)*, 2008, pp. 247–252.

[15] C. Foyer and B. Goglin, "Using Bandwidth Throttling to Quantify Application Sensitivity to Heterogeneous Memory," *2021 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, pp. 9–16, 2021.

[16] D. Genbrugge and L. Eeckhout, "Statistical simulation of chip multiprocessors running multi-program workloads," in *2007 25th International Conference on Computer Design*, 2007, pp. 464–471.

[17] P. B. Gibbons, Y. Matias, and V. Ramachandran, "The queue-read queue-write pram model: Accounting for contention in parallel algorithms," *SIAM J. Comput.*, vol. 28, no. 2, p. 733–769, feb 1999. [Online]. Available: https://doi.org/10.1137/S009753979427491

[18] C. Gomes, M. Amiraski, and M. Hempstead, "CASHT: Contention Analysis in Shared Hierarchies with Thefts," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 1, 03 2022. [Online]. Available: https://doi.org/10.1145/3494538

[19] C. Gomes and M. Hempstead, "C^2AFE: Capacity curve annotation and feature extraction for shared cache analysis," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 113–115.

[20] P. Gratz, J. Kim, A. Alameldeen, A. Jaleel, S. Pugsley, C. Wilkerson, M. Ferdman, D. Jimenez, M. Qureshi, E. Rotenbur, C.-J. Wu, A. Jain, and G. Chacon, "Cache replacement championship 2," http://crc2.ece.tamu.edu/, 2017.

[21] F. Hameed, L. Bauer, and J. Henkel, "Reducing inter-core cache contention with an adaptive bank mapping policy in dram cache," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '13. IEEE Press, 2013.

[22] ——, "Reducing inter-core cache contention with an adaptive bank mapping policy in dram cache," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '13. IEEE Press, 2013.

[23] C. Helm and K. Taura, "Automatic identification and precise attribution of dram bandwidth contention," in *49th International Conference on Parallel Processing - ICPP*, ser. ICPP '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3404397.3404422

[24] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer, "Cache QoS: From concept to reality in the intel® xeon® processor e5-2600 v3 product family," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 657–668.

[25] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 60–71. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815971

[26] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 197–206.

[27] T. Kantecki, *Intel Resource Director Technology*, v2.1.0, 2015 [Online]. [Online]. Available: https://github.com/intel/intel-cmt-cat

[28] A. Khan, M. Vijayaraghavan, S. Boyd-Wickizer, and Arvind, "Fast and cycle-accurate modeling of a multicore processor," in *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software*, ser. ISPASS '12. USA: IEEE Computer Society, 2012, p. 178–187. [Online]. Available: https://doi.org/10.1109/ISPASS.2012.6189224

[29] J. Kim, "Champsim," https://github.com/ChampSim/ChampSim, 2017.

[30] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A defense against cache timing attacks in speculative execution processors," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-51. IEEE Press, 2018, p. 974–987. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00083

[31] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79 – 86, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[32] G. Malhotra, R. Kalayappan, S. Goel, P. Aggarwal, A. Sagar, and S. R. Sarangi, "Partejas: A parallel simulator for multicore processors," *ACM Trans. Model. Comput. Simul.*, vol. 27, no. 3, aug 2017. [Online]. Available: https://doi.org/10.1145/3077582

[33] J. Mars and M. L. Soffa, "Synthesizing contention," in *Proceedings of the Workshop on Binary Instrumentation and Applications*, ser. WBIA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 17–25. [Online]. Available: https://doi.org/10.1145/1791194.1791197

[34] J. Mars, L. Tang, and M. L. Soffa, "Directly characterizing cross core interference through contention synthesis," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, ser. HiPEAC '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 167–176. [Online]. Available: https://doi.org/10.1145/1944862.1944887

[35] T. Mudge, C.-C. Lee, and S. Sechrest, "Correlation and aliasing in dynamic branch predictors," in *23rd Annual International Symposium on Computer Architecture (ISCA'96)*, 1996, pp. 22–22.

[36] A. Mukkara, N. Beckmann, and D. Sanchez, "Whirlpool: Improving dynamic cache management with static data classification," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 113–127. [Online]. Available: https://doi.org/10.1145/2872362.2872363

[37] Y. Oltchik and O. Schwartz, *Network Partitioning and Avoidable Contention*. New York, NY, USA: Association for Computing Machinery, 2020, p. 563–565. [Online]. Available: https://doi.org/10.1145/3350755.3400242

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

[39] T. R. Puzak, "Analysis of cache replacement-algorithms," 1985.

[40] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 381–391. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250709

[41] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 39. Washington, DC, USA: IEEE Computer Society, 2006, pp. 423–432.

[42] S. A. Rashid, G. Nelissen, and E. Tovar, "Trading Between Intra- and Inter-Task Cache Interference to Improve Schedulability," 10 2018, pp. 125–136.

[43] ——, "Cache persistence-aware memory bus contention analysis for multicore systems," in *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, ser. DATE '20. San Jose, CA, USA: EDA Consortium, 2020, p. 442–447.

[44] D. Rolán, D. Andrade, B. B. Fraguela, and R. Doallo, "A fine-grained thread-aware management policy for shared caches," *Concurrency and computation*, vol. 26, no. 6, pp. 1355–1374, 2014.

[45] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and efficient fine-grain cache partitioning," *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 57–68, Jun. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024723.2000073

[46] ——, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, p. 475–486, jun 2013. [Online]. Available: https://doi.org/10.1145/2508148.2485963

[53] R. Wang and L. Chen, "Futility scaling: High-associativity cache partitioning," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014, pp. 356–367.

[47] M. Shantharam, P. Raghavan, and M. Kandemir, "Hybrid techniques for fast multicore simulation," in *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, ser. Euro-Par '09. Berlin, Heidelberg: Springer-Verlag, 2009, p. 122–134. [Online]. Available: https://doi.org/10.1007/978-3-642-03869-3_15

[48] H. Shen and C. Li, "Detecting last-level cache contention in workload colocation with meta learning," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 14–23.

[49] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ACM SIGARCH Computer Architecture News*, vol. 30, no. 5. ACM, 2002, pp. 45–57.

[50] J. Shun, G. E. Blelloch, J. T. Fineman, and P. B. Gibbons, "Reducing contention through priority updates," in *Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 152–163. [Online]. Available: https://doi.org/10.1145/2486159.2486189

[51] Standard Performance Evaluation Corporation, "SPEC Benchmark Suite," http://www.spec.org.

[52] P. Tsai, N. Beckmann, and D. Sanchez, "Jenga: Software-defined cache hierarchies," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 06 2017, pp. 652–665.

[54] P. P. L. Wen-Tzer Thomas Chen and K. C. Stelzer, "Implementation of a pseudo-LRU algorithm in a partitioned cache," in *U. S. Patent Office, June 2006. Patent number 7,069,390*, June 2006.

[55] Y. Xie and G. H. Loh, "PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 174–183. [Online]. Available: https://doi.org/10.1145/1555754.1555778

[56] C. Xu, x. Chen, R. Dick, and Z. Mao, "Cache contention and application performance prediction for multi-core systems," 04 2010, pp. 76 – 86.

[57] H. Xu, S. Wen, A. Giménez, T. Gamblin, and X. Liu, "DR-BW: Identifying Bandwidth Contention in NUMA Architectures with Supervised Learning," *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 367–376, 2017.

[58] D. Zhan, H. Jiang, and S. C. Seth, "Clu: Co-optimizing locality and utility in thread-aware capacity management for shared last level caches," *IEEE transactions on computers*, vol. 63, no. 7, pp. 1656–1667, 2014.

[59] Q. Zhao, D. Koh, S. Raza, D. Bruening, W.-F. Wong, and S. Amarasinghe, "Dynamic cache contention detection in multi-threaded applications," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 27–38. [Online]. Available: https://doi.org/10.1145/1952682.1952688