Instruction Criticality based Energy-efficient Hardware Data Prefetching

Neelu Shivprakash Kalani* and Biswabandan Panda†
*IIT Kanpur(neeluk@cse.iitk.ac.in) †IIT Bombay (biswa@cse.iitb.ac.in)

Abstract—Hardware data prefetching is a latency hiding technique that mitigates the memory wall problem by fetching data blocks into the caches before the processor demands them. For high performing state-of-the-art data prefetchers, this leads to increase in dynamic and static energy in memory hierarchy, due to increase in number of requests. A trivial way to improve the energy efficiency of hardware prefetchers is to prefetch instructions on the critical path of execution. So, we perform energy-efficient hardware data prefetching by prefetching only for load instructions that are critical to application's performance. As criticality-based data prefetching does not degrade performance significantly; this is an ideal approach to solve the energy-efficiency problem without affecting the performance improvement, significantly. Prefetcher throttling is another approach to reduce inaccurate prefetch requests and thus, energy consumption. We discuss the limitations of existing critical instruction detection techniques and prefetcher throttler, and propose a new technique that uses re-order buffer occupancy as a metric to detect critical instructions and performs prefetcher-specific threshold tuning. With our detector, we achieve maximum memory hierarchy energy savings of 12.3% with 1.4% higher performance, for PPF, and average as follows: (i) SPEC CPU 2017 benchmarks: 2.04% lower energy, 0.3% lower performance, for IPCP at L1D, (ii) client/server benchmarks: 4.7% lower energy, 0.15% lower performance, for PPF, (iii) Cloudsuite benchmarks: 2.99% lower energy, 0.36% higher performance, for IPCP at L1D. IPCP and PPF are state-of-the-art hardware data prefetchers.

1 Introduction

State-of-the-art data prefetchers [1], [2], [3], [4] train on the demand memory access stream at private caches and generate prefetch requests into the memory hierarchy to provide performance improvement. However, these techniques do not discuss about the increase in memory hierarchy energy consumption due to (i) prefetcher's training and (ii) inaccurate requests into the memory hierarchy.

The problem: Figure 1 shows an increase of as high as 122% in dynamic and 26% in static energy in the memory hierarchy in the presence of state-of-the-art data prefetchers [1], [2], [3], [4] for 89 single threaded benchmarks. Across 89 benchmarks, dynamic energy increases by more than 10% for 15 benchmarks, and by more than 5% for 34 benchmarks. For 21 benchmarks, static energy is higher than baseline. Overall, more than 50 benchmarks show energy consumption higher than the average case. We use PCACTI [5] to model on-chip memory hierarchy energy for 7 nm FinFET technology (See Table 1). We use high-performance configuration for L1D and energy-efficient configuration for the rest. We use power-gating [6] for L2 and L3 caches, and DRAM [7]. We extend Orion [8] to 7 nm technology to model on-chip interconnect energy.

A trivial solution: Data prefetchers increase the number of accesses in the memory hierarchy, hence, dynamic energy. With lower execution time, static energy reduces. However, with energy optimization techniques like power-gating [6], an increase in requests leads to an increase in static energy due to lower power-gating opportunity. A trivial solution to reduce memory hierarchy energy while retaining performance that prior works [9], [10], [11] propose is to perform prefetching only for critical load instruction pointers (IPs). Critical IPs are instructions on the critical execution path, that contribute to the application's overall execution time. However, data prefetchers treat all demand loads impartially instead of prioritising critical IPs, as performance is the primary citizen. Another approach is to use prefetcher throttling [12] which tunes the prefetcher's aggressiveness to minimize inaccurate requests and improve performance.

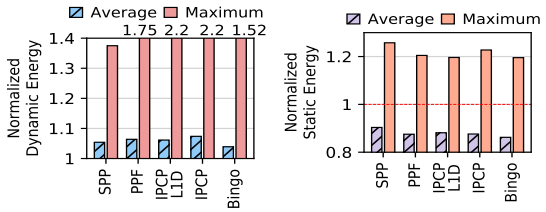

Fig. 1: Memory hierarchy energy consumption for 89 benchmarks (refer Section 5) normalized to no prefetching. Lower the better.

Challenges: Existing critical IP detectors and prefetcher throttlers are either costly (operations per cycle) or do not use apt metrics (Section 3). This motivates us to find an apt critical IP detection metric. Since, different prefetcher and benchmark combinations can impact differently with various data prefetching techniques, the challenge is to ensure that the critical IP detector can tune itself specifically for the prefetcher and the benchmark.

Our contributions: We use data prefetching for critical IPs for energy-efficiency in memory hierarchy. We motivate for the usage of re-order buffer (ROB) occupancy as the critical IP detection metric (Section 4) to overcome limitations of existing detectors and throttlers (Section 5). We introduce prefetcher-specific threshold relaxation to tune critical IP detection thresholds at run-time (Section 4, 5). With data prefetching for critical IPs, we show a maximum reduction of 14.94% and 16.02% in dynamic and static energies with 0.5% performance loss and 1.4% performance gain, for PPF [4] and SPP [3], respectively. Our detector incurs a hardware overhead of 1.4 KB. *To the best of our knowledge, this is the first*

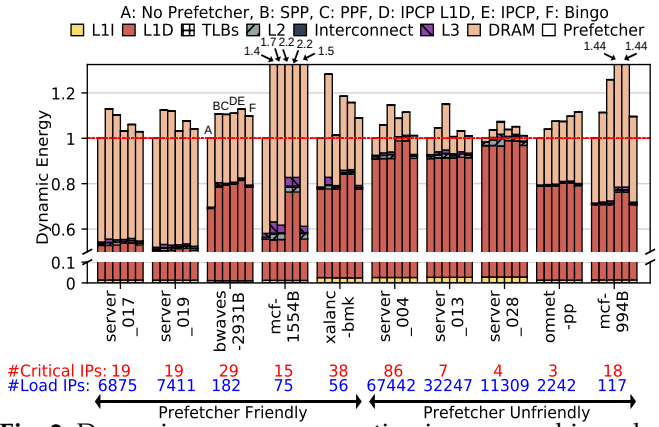

Fig. 2: Dynamic energy consumption in memory hierarchy with data prefetchers normalized to no prefetching.

work that (i) uses ROB occupancy for critical IP detection, (ii) performs prefetcher and benchmark specific threshold relaxation at run-time for critical IP detection, and (iii) shows the impact of prefetching only for critical IPs with state-of-the-art data prefetchers and state-of-the-art critical IP detection techniques.

2 Background & Related Work

Energy consumption: Dynamic energy consumption is due to run-time operations (reads/writes etc.) in memory hierarchy. Static energy is leakage energy in inactive or idle state. Prefetching increases dynamic energy consumption. Ideally, prefetching reduces static energy as it reduces execution time of an application (energy is directly proportional to time) [13]. Power-gating [6] reduces leakage energy by powering off caches in idle state.

State-of-the-art data prefetchers: Instruction Pointer Classifier-based spatial hardware Prefetching (IPCP) [1] uses a light-weight (< 1 KB) multi-level prefetcher for private caches. It performs per IP training and prefetching. Bingo [2] fine-tunes its learning with longer event recurrences, i.e., IP and full address or address offset, and requires a storage overhead of 119 KB. Signature Path Prefetching (SPP) [3] performs lookahead to predict future address deltas for a given signature (e.g., a memory region). The prediction of deltas and prefetching continues until the confidence drops below certain thresholds. Perceptron-based Prefetch Filtering (PPF) [4] augments SPP by allowing it to continue prediction and filters prefetch requests with a perceptron-based prefetch filter.

State-of-the-art critical IP detectors: Focused Value Prediction (FVP) [14] uses ROB stall, confidence-based critical IP detector. Criticality Aware Tiered Cache Hierarchy (CATCH) [10] uses enumeration of data dependency graph (DDG) to detect critical IPs. Focused Prefetching [9] uses the number of cycles for which ROB stall occurs as a critical IP detection metric, and uses preset thresholds. Subramaniam et al. [11] use the number of loads' direct dependents as a metric, and train on low issue rate in the processor pipeline.

Prefetcher throttling: Feedback Directed Prefetching (FDP) [12] controls the aggressiveness of prefetcher by taking into account (i) prefetcher accuracy, (ii) prefetcher lateness, and (iii) cache pollution due to prefetcher.

3 Motivation

Figure 2 and 3 show the memory hierarchy dynamic and static energy consumption, respectively, with no prefetching

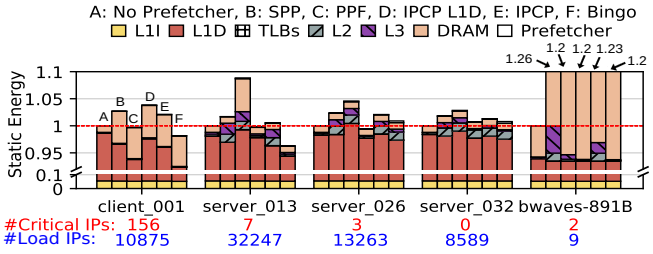

Fig. 3: Static energy consumption in memory hierarchy with data prefetchers normalized to no prefetching.

and following data prefetchers: SPP at L2, PPF at L2, IPCP at L1D, IPCP at L1D+L2, and Bingo at L2.

Dynamic energy consumption: Figure 2 shows maximum increase in dynamic energy of 2.2X for `mcf-1554B`. Figure 2 shows dynamic energy for five prefetcher friendly and unfriendly benchmarks. We name a benchmark friendly and unfriendly if it shows at least 24% and at most 4% performance improvement, respectively. Figure 2 also shows the critical, as per the detector we propose (Section 4), and the total number of load IPs per benchmark. Here, out of thousands of load IPs, only tens are critical to the application's performance. Yet, prefetchers train and prefetch on all load IPs instead of just critical ones. Ideally, a data prefetcher should improve performance significantly, while consuming marginal energy and critical IP based prefetching is one of the ways to achieve this.

Static energy consumption: Figure 3 shows that static energy consumption increases with data prefetchers in some cases (26% for `bwaves-891B`). Two factors contribute to this: (i) increase in cache or DRAM activity in the presence of power-gating, (ii) low or nil performance improvement with prefetchers. For example, in `client_001` in Figure 3, with Bingo, overall static energy is less than baseline, but DRAM static energy is higher. Performance improvement for `client_001` with Bingo (7%) is high enough to reduce overall static energy consumption even with power gating. However, static energy is higher than baseline when performance improvement is not high enough to counter static energy increase due to an increase in cache or DRAM activity.

Summary: Based on these observations, we pose the following questions: (i) can we perform energy-efficient data prefetching to gain similar performance?, and (ii) even if we do not improve performance further, can we reduce static energy consumption?

Limitations of existing critical IP detection techniques and prefetcher throttling: FVP uses a 2-bit confidence counter and aggressively marks IPs with incomplete execution in retire width as critical. CATCH finds the maximum cost incoming edge for each DDG node. So, it performs at least *retire width* number of operations per cycle. Both CATCH and Subramaniam et al.'s technique also use 2-bit confidence counters, thus aggressively mark IPs as critical. In Focused Prefetching, the processor has to increment a global counter in every cycle to compute the number of ROB stall cycles. Further, preset thresholds are ineffective in detecting critical IPs for certain benchmarks. Subramaniam et al.'s technique inaptly trains on a low issue rate. Large code footprint applications can have low issue rate due to the fetch stage (front-end) of the pipeline too. We find that

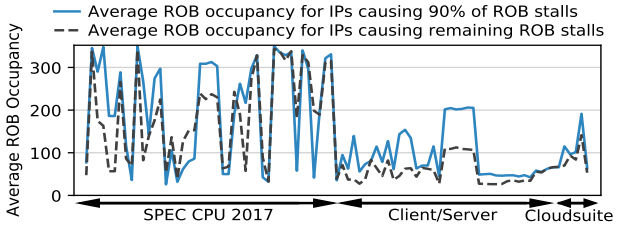

Fig. 4: ROB occupancy for MjS & MnS IP ROB stalls.

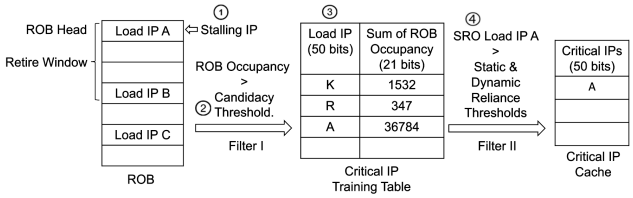

Fig. 5: Critical IP detection process.

when the pipeline issues one or zero instructions into ROB, it is not full 60% of the time. The metrics that FDP uses do not correlate directly with performance improvement.

In summary, (i) FVP, CATCH, and Subramaniam et al.'s technique make overpredictions, (ii) CATCH and Focused Prefetching are costly in terms of operations per cycle, (iii) using preset thresholds is not optimal in Focused Prefetching, (iv) Subramaniam et al. use inapt metrics, and (v) FDP may reduce energy consumption by reducing prefetcher's aggressiveness, but it can lead to performance loss as well.

Thus, we propose the usage of ROB occupancy for critical IP detection (as a ROB stall affects all instructions in the ROB), with run-time threshold relaxation. It requires one subtract operation on a ROB stall i.e., between the ROB's head and the tail.

4 Data Prefetching for Critical IPs

ROB occupancy: We term IPs causing more than 90% of ROB stalls as MjS IPs (Major Stalling IPs) and IPs causing rest ROB stalls as MnS IPs (Minor Stalling IPs). We capture IPs causing the most stalls as MjS, until we cover 90% of ROB stalls. Figure 4 shows that, in most cases, average ROB occupancy is higher for ROB stalls that MjS IPs cause. Average ROB stalls per kilo cycles (SPKC) are also higher for MjS IPs (27.7) than MnS IPs (3). So, we check both ROB occupancy and stall frequency, to determine critical IPs.

Critical IP detection: We use two hardware filters to classify an IP as critical (Figure 5). IPs that pass through filter I are present in critical IP training table (critical IP candidates). IPs that pass through both filter I and II, are present in critical IP cache. When a ROB-stalling load IP 'A' commits (①), we check if the ROB occupancy is greater than candidacy threshold (②) (filter I). If so, we add or update an entry in the training table. This table stores the sum of ROB occupancy for all ROB stalls (through filter I) by IP 'A' (③). After updating the entry, we check if sum of ROB occupancy is greater than reliance thresholds (filter II) (④). We use a preset static reliance threshold, and a dynamic one i.e., average of sum of ROB occupancy at all ROB stalls in current one million instructions window. We pass a critical IP bit with every load request to the memory hierarchy. Based on empirical results, we use candidacy threshold of 50 and static reliance threshold of 20,000. We use 64 (2 ways) and 128 (4 ways) entries for critical IP training table and critical IP cache, respectively.

TABLE 1: Energy consumption with 7 nm FinFET [5].

	Dynamic read energy (pJ)	Static power (mW)
L1I Cache	33.6	2.935
L1D Cache	1100	49.3
L2 TLB	1.1	0.6
L2 Cache	45.5	17.81
L3 Cache	101.1	64.57

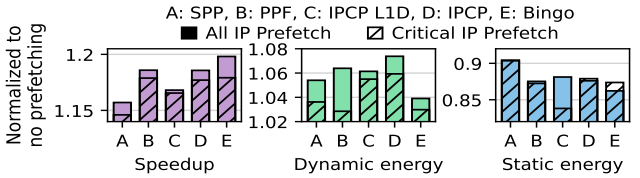

Fig. 6: Normalized speedup and memory hierarchy energy with all IP vs. critical IP data prefetching.

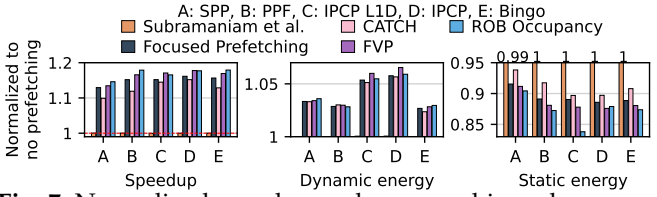

Fig. 7: Normalized speedup and memory hierarchy energy with existing techniques and our critical IP detector.

Prefetcher-specific threshold relaxation: Our goal is to gain the maximum performance per benchmark from each prefetcher. So, our critical IP detector tunes itself, without needing prefetcher-specific information. We observe critical IP detection during windows of one million instructions, and update the thresholds (candidacy and static reliance) as follows: Initially, the prefetcher trains and prefetches for all IPs for a one million instructions window. Then we note the instructions per cycle (IPC) of the application using the processor's performance monitoring counters. In the next window, the prefetcher trains and prefetches only for critical IPs. If the IPC is higher by 'd'% in the first window, we reduce the thresholds by 'x'%. We repeat this process until the IPC difference falls below 'd'%. Based on empirical analysis, we use 'x' as 20 and 'd' as 4.

5 Results

We use ChampSim [17] with an extensive front-end and virtual memory support for evaluation. We use 89 benchmarks: memory-intensive SPEC CPU 2017 [18], client/server [19], and Cloudsuite [20]. We report results for 50 million instructions for client/server and Cloudsuite, 100 million instructions for SPEC CPU 2017, after warmup of 50 million instructions. Table 2 shows the system parameters [21].

Effect of power-gating penalty: We model DRAM energy [7] with and without the power-gating penalty. With a power-gating penalty in power-down mode, we observe an average performance slowdown of 2.9% (maximum across all prefetchers). In this paper, we show the results without power-gating penalty, as conclusions do not change.

Performance and energy: Overall, there is lower energy consumption and similar performance improvement with data prefetching for critical IPs (Figure 6). Maximum average performance loss is 1.9% with Bingo. We achieve average dynamic energy savings of 3.5% with PPF: performance improves by 5.8% for `server_022`, and prefetch requests issued reduce by 53.3%. Static energy increases with Bingo as performance degrades. However, it reduces with PPF and IPCP at L1D with more opportunity for power-gating due

TABLE 2: Parameters of the simulated system.

Processor	4 GHz out-of-order, 6-wide issue, 352-entry 4-wide ROB, Hashed Perceptron
TLBs	64-entry ITLB/DTLB, 1536-entry STLB, LRU
L1I cache	32KB 8-way (4 cycles), LRU, FNL+MMA [15]
L1D cache	48KB 12-way (5 cycles), LRU
L2 cache	512KB 8-way (10 cycles), SRRIP [16]
L3 cache	2MB 16-way (20 cycles), DRRIP [16]
DRAM	6400 MT/s, 1 channel

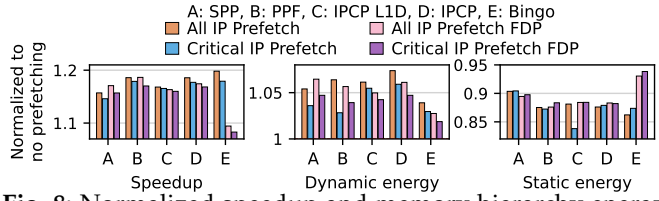

Fig. 8: Normalized speedup and memory hierarchy energy with all and critical IP prefetching, with and without FDP.

TABLE 3: Comparison of existing critical IP detectors.

	Subramaniam et al. [11]	Focused Pref. [9]	CATCH [10]	FVP [14]	ROB occu.
Num. critical IPs	3192	344	2012	4272	84
Operations/cycle	0.03	1.07	4.24	0.03	0.07
Coverage (%)	68.5	63.1	97.7	100	35.5
Accuracy (%)	0.41	51.6	0.68	0.82	68

to less requests in memory heirarchy. We achieve maximum energy savings (including dynamic and static) of 12.3% for `server_013` with PPF. For 53 benchmarks, we detect less than 32 critical IPs; for three benchmarks, zero. This eliminates prefetch requests that do not improve performance and reduces energy consumption.

Comparing existing critical IP detectors: As Figure 7 shows, FVP performs better with IPCP and IPCP at L1D but consumes higher dynamic energy (than our technique). CATCH consumes similar dynamic energy with lower performance. Table 3 shows that our technique is most conservative in detecting critical IPs. It performs lower operations per cycle than CATCH and Focused Prefetching to identify critical IP candidates. Focused Prefetching provides higher coverage but lower accuracy and performance improvement. Subramaniam et al.'s technique [11] is ineffective, as 94.9% of the time, it detects a low issue rate. We compute coverage and accuracy based on MjS IPs. FVP, CATCH, and Subramaniam et al.'s technique provide low accuracy due to overpredictions, and as CATCH and Subramaniam et al. do not use ROB stalls to detect critical IPs.

Comparing with prefetcher throttler: Figure 8 shows that FDP [12] with critical IP based prefetching degrades performance except for SPP. Performance improves with SPP due to aggressive prefetching; dynamic energy consumption increases too. There is a 10.4% performance loss with Bingo, due to reduction in aggressiveness, and as all prefetchers except Bingo have internal aggressiveness tuning. Overall, FDP is unable to retain performance while reducing energy consumption. Ideally, a prefetcher throttler can complement critical IP based prefetching to reduce inaccurate prefetch requests.

Threshold relaxation: Figure 9 shows threshold relaxation provides highest performance improvement except for IPCP at L1D, and consumes lowest energy in most cases. No specific preset threshold performs the best for all prefetchers. Figure 10 shows the utility of prefetcher-specific threshold relaxation for critical IP detection for sensitive benchmarks. We also see cases where different prefetchers

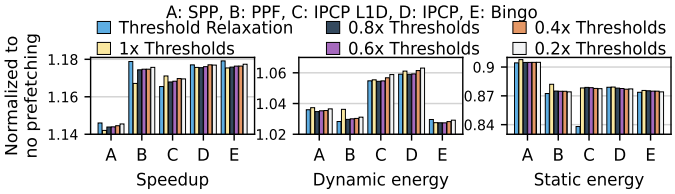

Fig. 9: Normalized speedup and memory hierarchy energy with threshold relaxation and preset thresholds.

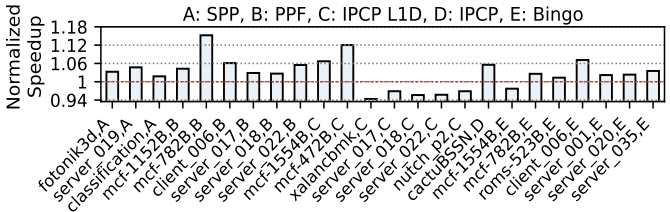

Fig. 10: Speedup using threshold relaxation over preset thresholds for critical IP detection.

use different thresholds for a single benchmark, making it prefetcher and benchmark specific.

6 Conclusion

In this paper, we showed increase in dynamic and static energy consumption in memory hierarchy by data prefetchers. We proposed a ROB occupancy based critical IP detector, with prefetcher-specific threshold relaxation. We showed that our proposal performs better than existing critical IP detectors and prefetcher throttlers. We showed the utility of threshold relaxation. We showed similar performance gain with lower memory hierarchy energy consumption (maximum savings of 12.3%) with critical IP based energy-efficient hardware data prefetching.

References

[1] Pakalapati and Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *ISCA*, 2020.
[2] Bakhshalipour *et al.*, "Bingo spatial data prefetcher," in *HPCA*, 2019.
[3] Kim *et al.*, "Path confidence based lookahead prefetching," in *MICRO*, 2016.
[4] Bhatia *et al.*, "Perceptron-based prefetch filtering," in *ISCA*, 2019.
[5] "Pcacti tool," https://sportlab.usc.edu/downloads/download/.
[6] Chiou *et al.*, "Timing driven power gating," in *DAC*, 2006.
[7] Lee *et al.*, "Leveraging power-performance relationship of energy-efficient modern dram devices," in *IEEE*, 2018.
[8] Kahng *et al.*, "Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration," in *DATE*, 2009.
[9] Manikantan and Govindarajan, "Performance oriented prefetching enhancements using commit stalls," in *ICS*, 2008.
[10] Nori *et al.*, "Criticality aware tiered cache hierarchy:a fundamental relook at multi-level cache hierarchies," in *ISCA*, 2018.
[11] Subramaniam *et al.*, "Criticality-based optimizations for efficient load processing," in *HPCA*, 2009.
[12] Srinath *et al.*, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers.," in *HPCA*, 2007.
[13] Guo *et al.*, "Energy-efficient hardware data prefetching," in *TVLSI*, 2009.
[14] Bandishte *et al.*, "Focused value prediction," in *ISCA*, 2020.
[15] A. Seznec, "The fnl+mma instruction cache prefetcher," in *IPC-1, ISCA*, 2020.
[16] Aamer *et al.*, "High performance cache replacement using re-reference interval prediction (rrip)," in *ISCA*, 2010.
[17] "Champsim simulator," https://github.com/ChampSim/ChampSim.
[18] "Spec cpu 2017," https://www.spec.org/cpu2017/.
[19] "Client/server traces," https://research.ece.ncsu.edu/ipc/.
[20] "Cloudsuite traces," https://crc2.ece.tamu.edu/.
[21] Cutress, "Examining intel's ice lake processors taking a bite of the sunny cove microarchitecture," in *Anandtech*, 2019.