


# Instruction Criticality Based Energy-Efficient Hardware Data Prefetching

Neelu Shivprakash Kalani and Biswabandan Panda 

**Abstract**—Hardware data prefetching is a latency hiding technique that mitigates the memory wall problem by fetching data blocks into caches before the processor demands them. For high performing state-of-the-art data prefetchers, this increases dynamic and static energy in memory hierarchy, due to increase in number of requests. A trivial way to improve energy-efficiency of hardware prefetchers is to prefetch instructions on the critical path of execution. As criticality-based data prefetching does not degrade performance significantly; this is an ideal approach to solve the energy-efficiency problem. We discuss limitations of existing critical instruction detection techniques 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 data prefetchers.

**Index Terms**—Cache memory, microarchitecture

## 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.* Fig. 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 PRACTI [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

- Neelu Shivprakash Kalani is with EPFL, 1015 Lausanne, Switzerland. E-mail: neelu.kalani@epfl.ch.
- Biswabandan Panda is with IIT Bombay, Mumbai, Maharashtra 400076, India. E-mail: biswa@cse.iitb.ac.in.

Manuscript received 21 Aug. 2021; revised 20 Sept. 2021; accepted 28 Sept. 2021. Date of publication 1 Oct. 2021; date of current version 21 Oct. 2021.

This work is supported by the SRC under Grant SRC-2922.001.

(Corresponding author: Biswabandan Panda.)

Digital Object Identifier no. 10.1109/LCA.2021.3117005

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. Another approach is to use prefetcher throttling [12] which tunes the prefetcher's aggressiveness to minimize inaccurate requests and improve performance.

*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 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. Bingo [2] fine-tunes its learning with longer event recurrences, i.e., IP and full address or address offset. 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) prefetch accuracy, (ii) lateness, and (iii) cache pollution due to prefetcher.

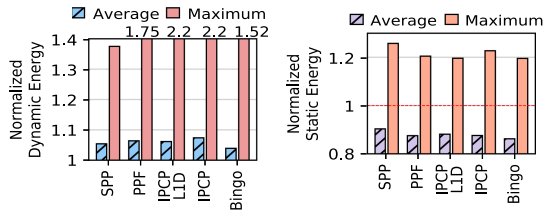


Fig. 1. Memory hierarchy energy consumption for 89 benchmarks (refer Section 5) normalized to no prefetching.

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

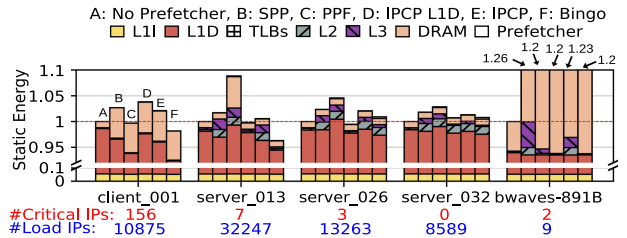


Fig. 3. Static energy consumption in memory hierarchy with data prefetchers normalized to no prefetching.

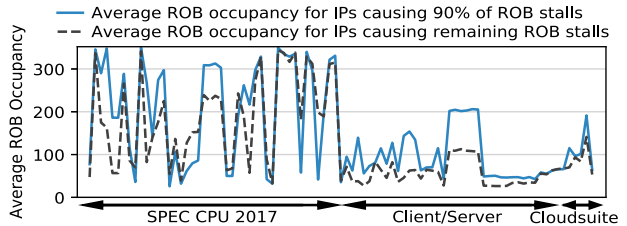


Fig. 4. ROB occupancy for MjS & MnS IP ROB stalls.

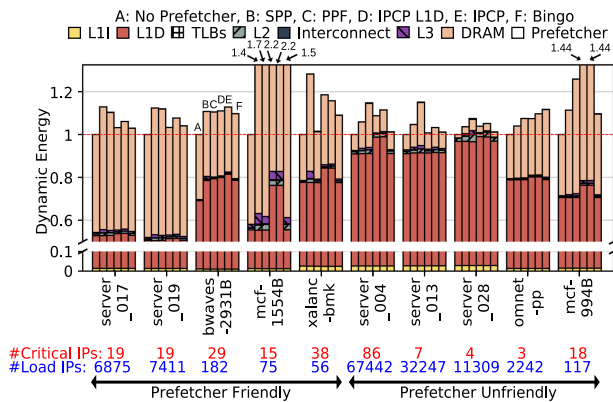


Fig. 2. Dynamic energy consumption in memory hierarchy with data prefetchers normalized to no prefetching.

### 3 MOTIVATION

Figs. 2 and 3 show the memory hierarchy dynamic and static energy consumption, respectively, with no prefetching and state-of-the-art hardware prefetchers.

**Dynamic Energy Consumption.** Fig. 2 shows maximum increase in dynamic energy of 2.2X for mcf-1554B. Fig. 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. Fig. 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.** Fig. 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 with power-gating, (ii) low or nil performance improvement with prefetchers. For example, in client\_001 in Fig. 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 enough to counter static energy increase due to an increase in cache or DRAM activity.

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?

**Limitations of Existing Critical IP Detection Techniques.** 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 when the pipeline issues one or zero instructions into ROB, it is not full 60% of the time.

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. Fig. 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.

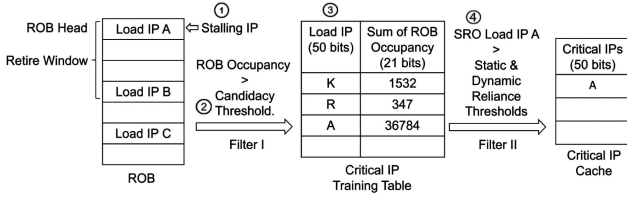


Fig. 5. Critical IP detection process.

TABLE 2  
Parameters of the Simulated System

Processor	4 GHz out-of-order, 6-wide issue, 352-entry ROB
TLBs	64-entry ITLB/DTLB, 1536-entry STLB, LRU
L1L cache	32KB 8-way (4 cycles), 8 MSHRs, LRU, FNL+MMA[16]
L1D cache	48KB 12-way (5 cycles), 16 MSHRs, LRU
L2 cache	512KB 8-way (10 cycles), 32 MSHRs, SRRIP[17]
L3 cache	2MB 16-way (20 cycles), 64 MSHRs, DRRIP[17]
DRAM	6400 MT/s, 1 channel

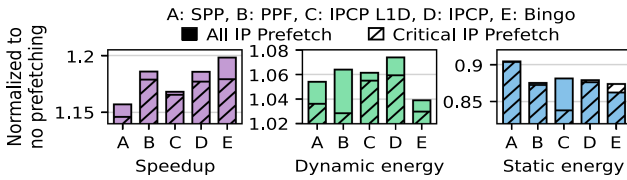


Fig. 6. Normalized speedup and memory hierarchy energy with all IP vs. critical IP data prefetching.

**Critical IP Detection.** We use two hardware filters to classify an IP as critical (Fig. 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.

**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. For resilience against application phase change, we compare the moving average of L1D demand accesses per cycle (APC) of last 16 windows with the APC of the current window [15]. If difference between the average-APC and current-APC crosses a threshold, a phase change is reported, upon which we start the threshold relaxation process again.

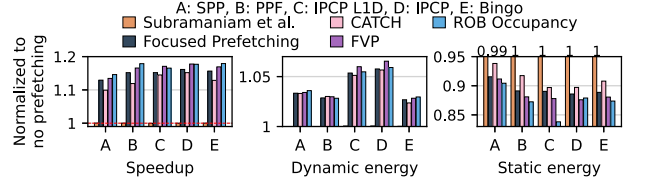


Fig. 7. Normalized speedup and memory hierarchy energy with existing techniques and our critical IP detector.

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

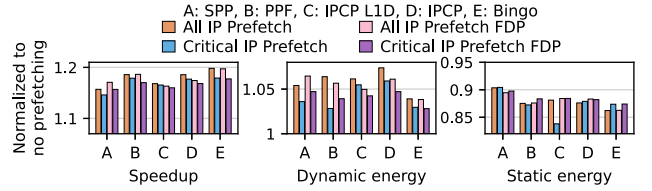


Fig. 8. Normalized speedup and memory hierarchy energy with all and critical IP prefetching, with and without FDP.

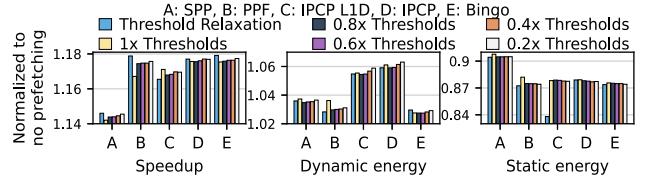


Fig. 9. Normalized speedup and memory hierarchy energy with threshold relaxation and preset thresholds.

## 5 RESULTS

We use ChampSim [18] with an extensive front-end and virtual memory support for evaluation. We use 89 benchmarks: memory-intensive SPEC CPU 2017 [19], client/server [20], and Cloudsuite [21]. 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.

**Effect of Switching Penalty With Power-Gating.** We implement L2/LLC power-gating at a 256KB size bank level. The switching penalty for a 2MB LLC (6T SRAM cell with 0.8V supply) is one to two cycles [5]. With a pipelined L2 and LLC implementation (32 MSHRs at L2 and 64 at LLC) for a 352 entry ROB, we see negligible performance impact (average gap of less than 0.03%). We corroborate our findings with one of the prior works [22]. For DRAM, we model 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). We show 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 (Fig. 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



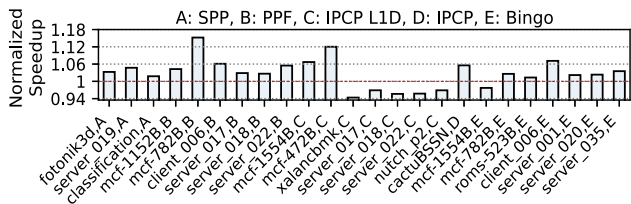


Fig. 10. Speedup using threshold relaxation over preset thresholds for critical IP detection.

L1D with more opportunity for power-gating due to less requests in memory hierarchy. 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.

**Comparison With the Critical IP Detectors.** Fig. 7 and Table 3 compare our critical IP detector with existing techniques. Our technique is the most conservative to detect critical IPs, with lower operations per cycle than CATCH and Focused Prefetching to identify critical IP candidates. 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. For 2/3rd of benchmarks that we study, our technique shows higher energy savings than FVP and Focused Prefetching (maximum gap of 10.36%). Our technique shows performance loss higher than 5% for five benchmarks, whereas we observe the same with FVP and Focused Prefetching for 1/4th of benchmarks. Our technique shows a maximum performance loss of 13%, whereas the same for FVP and Focused Prefetching is 24% and 37%, respectively.

**Comparison With the Prefetcher Throttler.** Fig. 8 shows that FDP [12] degrades performance except for SPP. Performance improves with SPP due to aggressive prefetching; dynamic energy consumption increases too. Critical IP based prefetching shows higher reduction in dynamic energy consumption than FDP except in IPCP at L1D. Further, FDP is unable to complement critical IP-based prefetching. Ideally, a prefetcher throttler can complement critical IP based prefetching to reduce inaccurate prefetch requests.

**Threshold Relaxation:** Fig. 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. Fig. 10 shows the utility of prefetcher-specific threshold relaxation for critical IP detection for sensitive benchmarks. We also see cases where different prefetchers 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 our proposal can provide competitive performance with lower memory hierarchy energy consumption (maximum savings of 12.3%) with critical IP based energy-efficient hardware data prefetching.

## ACKNOWLEDGMENTS

The authors would like to thank the CARS group, Anant Nori, and Shankar Balachandran for their valuable feedback.

## REFERENCES

- [1] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 118–131.
- [2] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 399–411.
- [3] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, pp. 1–12.
- [4] E. Bhatia *et al.*, "Perceptron-based prefetch filtering," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 1–13.
- [5] "Pcacti tool," 2011. [Online]. Available: <https://sportlab.usc.edu/downloads/download/>
- [6] D.-S. Chiou, S.-H. Chen, S.-C. Chang, and C. Yeh, "Timing driven power gating," in *Proc. 43rd ACM/IEEE Des. Autom. Conf.*, 2000, pp. 121–124.
- [7] S. Lee, H. Cho, Y. H. Son, Y. Ro, N. S. Kim, and J. H. Ahn, "Leveraging power-performance relationship of energy-efficient modern dram devices," *Proc. IEEE*, vol. 6, pp. 31387–31398, 2018.
- [8] A. B. Kahng, Bin Li, L. Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proc. Des. Autom. Test Europe Conf. Exhibit.*, 2009, pp. 423–428.
- [9] R. Manikantan and R. Govindarajan, "Performance oriented prefetching enhancements using commit stalls," in *Proc. Int. Conf. Supercomput.*, 2008, pp. 1–10.
- [10] A. V. Nori, J. Gaur, S. Rai, S. Subramoney, and H. Wang, "Criticality aware tiered cache hierarchy: a fundamental relook at multi-level cache hierarchies," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 96–109.
- [11] S. Subramaniam, A. Bracy, H. Wang, and G. H. Loh, "Criticality-based optimizations for efficient load processing," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, 2009, pp. 419–430.
- [12] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching," in *Proc. IEEE 13th Int. Symp. High Perform. Comput. Archit.*, 2007, pp. 63–74.
- [13] Y. Guo, P. Narayanan, M. A. Bennis, S. Chheda, and C. A. Moritz, "Energy-efficient hardware data prefetching," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 2, pp. 250–263, Feb. 2011.
- [14] S. Bandishte, J. Gaur, Z. Sperber, L. Rappoport, A. Yoaz, and S. Subramoney, "Focused value prediction," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 79–91.
- [15] B. Panda, "SPAC: A synergistic prefetcher aggressiveness controller for multi-core systems," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3740–3753, Dec. 2016.
- [16] A. Seznec, "The FNL+MMA instruction cache prefetcher," in *1st Instruction Prefetching Championship Workshop*, 2020, pp. 1–4.
- [17] A. Jaleel *et al.*, "High performance cache replacement using re-reference interval prediction (RRIP)," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 60–71.
- [18] "Champsim simulator," 2015. [Online]. Available: <https://github.com/ChampSim/ChampSim>
- [19] "Spec CPU 2017," [Online]. Available: <https://www.spec.org/cpu2017/>
- [20] "Client/server traces," [Online]. Available: <https://research.ece.ncsu.edu/ipc/>
- [21] "Cloudsuite traces," [Online]. Available: <https://crc2.ece.tamu.edu/>
- [22] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2011, pp. 694–701.

▷ For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/csdl](http://www.computer.org/csdl).