CLIP: Load Criticality based Data Prefetching for Bandwidth-constrained Many-core Systems

Biswabandan Panda
biswa@cse.iitb.ac.in
Indian Institute of Technology Bombay
Mumbai, India

ABSTRACT

Hardware prefetching is a latency-hiding technique that hides the costly off-chip DRAM accesses. However, state-of-the-art prefetchers fail to deliver performance improvement in the case of many-core systems with constrained DRAM bandwidth. For SPEC CPU2017 homogeneous workloads, the state-of-the-art Berti L1 prefetcher, on a 64-core system with four and eight DRAM channels, incurs performance slowdowns of 24% and 16%, respectively. However, Berti improves performance by 35% if we use an unrealistic configuration of 64 DRAM channels for a 64-core system (one DRAM channel per core).

Prior approaches such as prefetch throttling and critical load prefetching are not effective in the presence of state-of-the-art prefetchers. Existing load criticality predictors fail to detect loads that are critical in the presence of hardware prefetching and the best predictor provides an average critical load prediction accuracy of 41%. Existing prefetch throttling techniques use prefetch accuracy as one of the primary metrics. However, these techniques offer limited benefits for state-of-the-art prefetchers that deliver high prefetch accuracy and use prefetcher-specific throttling and filtering.

We propose CLIP, a novel load criticality predictor for hardware prefetching with constrained DRAM bandwidth. Our load criticality predictor provides an average accuracy of more than 93% and as high as 100%. CLIP also filters out the critical loads that lead to accurate prefetching. For a 64-core system with eight DRAM channels, CLIP improves the effectiveness of state-of-the-art Berti prefetcher by 24% and 9% for 45 and 200 64-core homogeneous and heterogeneous workload mixes, respectively. We show that CLIP is equally effective in the presence of other state-of-the-art L1 and L2 prefetchers. Overall, CLIP incurs a storage overhead of 1.56KB/core.

KEYWORDS

Instruction criticality, Cache, Prefetching, DRAM

ACM Reference Format:
Biswabandan Panda. 2023. CLIP: Load Criticality based Data Prefetching for Bandwidth-constrained Many-core Systems. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28-November 1, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3613424.3614245

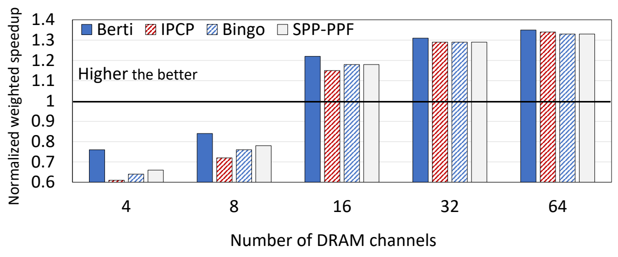

Figure 1: Performance of state-of-the-art prefetchers normalized to no prefetching for 45 64-core SPEC CPU2017 [1] homogeneous workload mixes with DDR4-3200 DRAM channels (peak bandwidth per channel: 25.6GB/sec).

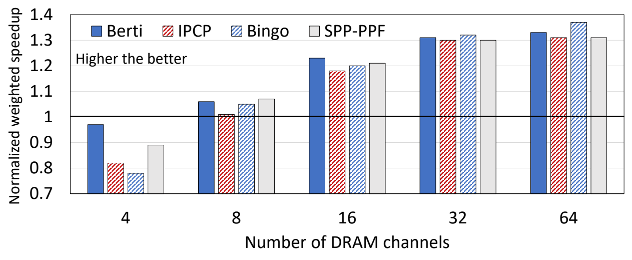

Figure 2: Performance of state-of-the-art Prefetchers normalized to no prefetching for 200 64-core heterogeneous workload mixes created from SPEC CPU2017 [1] and GAP [2] traces with DDR4-3200 DRAM channels (peak bandwidth per channel: 25.6GB/sec).

1 INTRODUCTION

Hardware data prefetching techniques play an important role in hiding the long latency DRAM accesses. Hardware prefetchers learn memory access patterns and fetch data into the cache hierarchy before time so that future memory accesses get cache hits. State-of-the-art L1 and L2 data prefetchers like Berti [3], instruction pointer classifier-based prefetching (IPCP) [4], signature path prefetching with perceptron filtering (SPP-PPF) [5, 6], and Bingo [7] have pushed the performance of L1 and L2 prefetchers. IPCP and Berti train on the demand memory access stream at the L1 data cache and orchestrate prefetch requests across the cache hierarchy. Berti is the state-of-the-art L1 prefetcher that outperforms IPCP and provides high prefetch accuracy (an average of more than 82.9%). Similarly, SPP-PPF, the state-of-the-art L2 prefetcher provides more than 73.4% of prefetch accuracy. However, all these prefetchers fail to deliver performance improvement in the case of many-core systems with limited DRAM bandwidth.

The problem. Commercial many-core systems like 60-core Intel Xeon Platinum [8], 64-core AMD EPYC Rome 7702P [9], and a 64-core AMD Threadripper 3990X [10] support eight DDR4-3200 channels. Figures 1 and 2 show performance improvements for homogeneous and heterogeneous workload mixes with the state-of-the-art L1 and L2 prefetchers on a 64-core simulated system with a different number of DRAM channels. The performance is normalized to a system with no prefetching with respective DRAM channels. All the prefetchers perform well with high DRAM bandwidth (64 DDR4-3200 channels for 64 cores). However, the effectiveness of these prefetchers goes down significantly with low DRAM bandwidth. A highly accurate Berti prefetcher also degrades performance significantly.

Figure 3 shows the normalized increase in average L1, L2, and L3 demand miss latencies with Berti for a different number of DRAM channels. For four and eight DRAM channels, the average L2 and L3 miss latencies increase by more than 1.9X. Note that the prefetcher can still predict future accesses with more than 82.9% accuracy in all cases. However, in the case of four and eight DRAM channels, prefetch lateness jumps to 19% and 13%, respectively, which used to be 1% for 64 DRAM channels. Note that, we consider the late but useful prefetch requests as accurate.

Overall, the low DRAM bandwidth problem manifests into a latency problem causing a slow DRAM response. The problem becomes worse in the presence of prefetching as prefetching accuracy is not 100% and prefetching introduces bursty traffic. As a side effect, the additional delay at the DRAM causes additional delays at the L3, L2, and L1 miss status holding registers (MSHRs), on-chip interconnect, and various queues at the different levels of the cache hierarchy, for both the prefetch and demand requests. As a result, demand and prefetch requests see high miss latency even for L2 and LLC hits.

The question of interest. For many-core systems, with constrained DRAM bandwidth, which load addresses should be prefetched so that we can mitigate the problem of additional latency caused by prefetching?

The ideal solution. Ideally, for many-core systems with constrained DRAM bandwidth, prefetchers should be 100% accurate and should prefetch load addresses that are critical for overall performance (loads that cause retiring stalls at the head of the reorder buffer (ROB)). Note that, not all L3 misses are costly even in the case of low DRAM bandwidth as an aggressive out-of-order processor with more than 500 ROB entries can hide a good fraction of L3 miss latency. Note that for parallel applications, thread criticality also plays an important role along with the load criticality as shown in prior works [11, 12].

Why prior techniques are not good enough? In the case of low DRAM bandwidth, prefetching for loads that are on the critical execution path [13] or loads that contribute to maximum ROB stalls, is a possible solution as it leads to a reduction in prefetch traffic without hampering the performance, significantly. Techniques for finding out critical loads [14–20] depend on the effectiveness of load criticality predictors used that use instruction pointer (IP) as the signature. Keeping an L1 prefetcher in mind, we define a load as a critical load if it stalls the head of the ROB while getting a response from L2, L3, or DRAM. We find the best among the state-of-the-art load criticality predictors provide 41% accuracy in predicting critical load IPs. One of the primary reasons for this low accuracy is that not all the load addresses that are generated by an IP are critical.

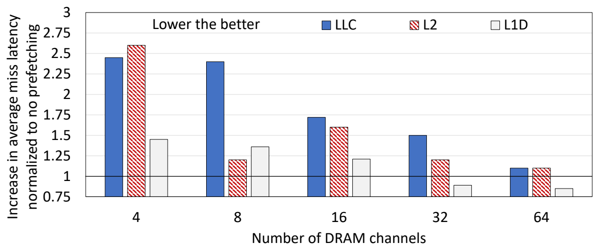

Figure 3: Increase in on-chip L1, L2, and L3 demand miss latencies with Berti normalized to no prefetching averaged across 245 64-core mixes (45 homogeneous and 200 heterogeneous workload mixes).

Another approach is to use prefetcher throttling techniques [21–26] that control the prefetcher's aggressiveness to minimize inaccurate prefetch requests and improve performance. There are software techniques [27] that facilitate bandwidth-conscious prefetching for the hardware prefetchers on the real system. However, prefetch throttling techniques are effective with prefetchers like IP-stride [28] and stream [29] as the prefetch accuracy of these prefetchers is relatively low (an average accuracy of less than 60%). However, state-of-the-art prefetchers deliver high prefetch accuracy. In general, existing throttling techniques are *coarse-grained* (throttling decisions based on the overall performance of the prefetcher) in nature and these techniques cannot identify specific loads that are responsible for performance loss in the case of constrained DRAM bandwidth.

Shared resource management techniques[30–34] are also possible solutions that try to minimize the effect of inaccurate prefetch requests at the shared resources like a last-level cache (LLC), DRAM, and network on chip (NOC). However, these techniques use *prefetch accuracy* again at a *coarse-grained* level at shared resources like DRAM, LLC, and NOC.

Hermes [35] and DSPatch [36] are two related techniques that can solve this problem. Hermes predicts the off-chip loads and expedites off-chip load accesses by bypassing the cache hierarchy. However, we find that Hermes is not an effective approach as we find that not all DRAM responses cause ROB stalls and a majority of the ROB stalls (60%) come even from L2 and LLC hits, thanks to the constrained DRAM bandwidth. DSPatch [36] is a technique that can be applied to any prefetcher for DRAM bandwidth-conscious prefetching. DSPatch improves the effectiveness of prefetchers in case of high DRAM bandwidth. It uses prefetch accuracy-based prefetching if bandwidth is utilized heavily and it goes to prefetching based on prefetch coverage if the bandwidth is underutilized. Note that DSPatch considers per DRAM-controller bandwidth *independently* and not the overall DRAM bandwidth across all the DRAM controllers, and hence provides a myopic picture of bandwidth utilization. For bandwidth-constrained systems, DSPatch uses prefetch coverage-based prefetching, exacerbating the problem.

Our approach. We propose CLIP, a lightweight yet highly accurate critical load predictor that advocates for highly accurate and critical load prefetching. CLIP enhances the effectiveness of state-of-the-art L1 and L2 prefetchers and provides answers to questions like "which load address to prefetch" while keeping constrained DRAM bandwidth in mind. CLIP works in two stages: (i) Stage I: it filters out critical loads (loads that stall the head of the ROB while servicing an L1 load miss) and uses a criticality predictor using a *critical signature* to predict the dynamic behavior of loads. (ii) Stage II: It uses a fine-grained accuracy filter to find out whether the underlying prefetcher will be able to prefetch accurately the predicted critical load addresses.

Our contributions. We provide a detailed overview of existing load criticality predictors, and prefetch throttlers, and show why existing techniques are not effective in the presence of constrained DRAM bandwidth(Section 3). We overcome the limitations of existing critical IP predictors and prefetch throttlers and propose CLIP (Section 4). CLIP uses *fine-grained* features instead of an IP to predict the criticality and usefulness of a given load IP. Experimental results show that CLIP provides accuracy as high as 100% (on average 93%) in predicting critical load addresses that leads to accurate prefetching. For a 64-core system with eight DRAM channels, CLIP improves the performance of state-of-the-art L1 prefetcher by 24% and 9% for 45 and 200 64-core homogeneous and heterogeneous workload mixes, respectively (Section 5). CLIP provides these performance improvements with additional storage of 1.56KB per core.

2 RELATED WORK

2.1 Recent Data Prefetching Mechanisms

Signature path prefetching (SPP) [5] and Perceptron Prefetch Filtering (PPF) [6] uses a lookahead mechanism to predict the future address deltas for a given signature (e.g., a memory region). For each region, SPP stores the history of the past deltas observed in the form of a signature. SPP uses this signature to predict the next delta in the path and generates a prefetch request accordingly. PPF allows SPP to continue the prediction regardless of confidence. It uses a perceptron-based prefetch filter to decide whether to issue prefetch requests or not based on their usefulness.

Bingo [37] argues that correlating memory access patterns to a single event is not enough to perform effective prefetching. A single event stands for the occurrence of one incident e.g. execution of the instruction with IP 'A'. Thus, Bingo fine-tunes its learning with longer event recurrences. Bingo uses the following two events for the same: (i) IP + Offset - the short event when an IP requests the same offset in any region, (ii) IP + Address - the long event when an IP requests the same address. The short event gets its name from the fact that it recurs more frequently than the long event.

Instruction pointer classifier prefetching (IPCP) [4] uses a lightweight multi-level prefetcher. IPCP classifies IPs into three classes and prioritizes prefetch requests by classes. After the classification, IPCP uses a bouquet of prefetchers corresponding to each class to generate the prefetch requests. IPCP is trained at the L1D with the benefits of monitoring the L1D access stream.

Berti [3] is the state-of-the-art local-delta L1 data prefetcher that trains at L1 and orchestrates prefetch requests across L1 to L3. Berti emphasizes the detection of timely local (per IP) deltas along with a precise mechanism to compute the local coverage of the detected timely deltas. Berti uses watermarks based on local coverage of deltas and decides the prefetch fill level. The high coverage timely deltas lead to a highly accurate prefetcher that outperforms IPCP, SPP-PPF, and Bingo.

2.2 Load Criticality Predictors

Focused Value Prediction based Criticality detector [16] uses a confidence mechanism to mark instructions whose execution is in-flight when they are present in the retire width window (distance between the ROB head and the instruction is less than retire width).

Criticality Aware Tiered Cache Hierarchy (CATCH) [15] uses an enumeration of the data dependency graph (DDG) to find out critical IPs. The costliest path in the DDG is the critical path and all the load IPs lying on this path are the critical load IPs. CATCH captures the costliest path incrementally by checking the costliest incoming edge during the insertion of each retiring instruction into the graph. Using the DDG along with a confidence mechanism, CATCH marks the critical IPs.

Focused Prefetching (FP) [17] Focused Prefetching showcases that instructions that stall at the head of the ROB comprise of a few loads; loads Incurring Majority of Commit Stalls (LIMCOS). It also shows that LIMCOS does not entirely overlap with delinquent loads i.e., a few loads that cause the most misses in the cache hierarchy or loads that fall on the critical path of execution.

Commit Block Predictor (CBP)[20] predicts the loads that stall the head of the ROB. It also uses maximum stall time or total stall time to predict a critical load.

ROB occupancy based criticality predictor (ROBO) [18] uses ROB occupancy for critical IP detection. On a ROB stall during retirement, higher ROB occupancy is an indicator of a critical load. It also uses thresholds based on ROB stalls.

Critical Slice Prefetching (CRISP) [19] is a technique that considers loads that get LLC misses and with low memory level parallelism (MLP) as the critical loads. CRISP sets pre-defined thresholds on LLC miss count and MLP for a given set of workloads.

Note that some of the above proposals are proposed for predicting the criticality of all kinds of instructions, and can be used for predicting critical load instructions, only.

3 WHY NOT EXISTING SOLUTIONS?

Load criticality predictors. Existing techniques that predict load criticality [15–20] when used along with hardware prefetching suffer from low prediction accuracy in detecting critical IPs while trying to achieve higher prediction coverage. Figure 4 shows the load criticality prediction accuracy and coverage of some of the

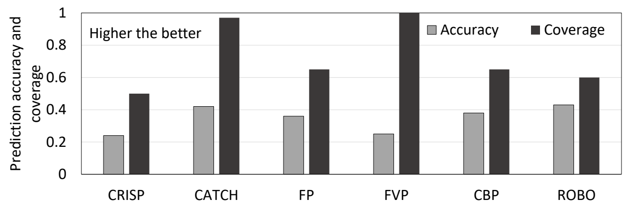

Figure 4: Load criticality prediction accuracy and coverage of state-of-the-art critical load predictors averaged across 245 64-core workload mixes (45 homogeneous and 200 heterogeneous mixes).

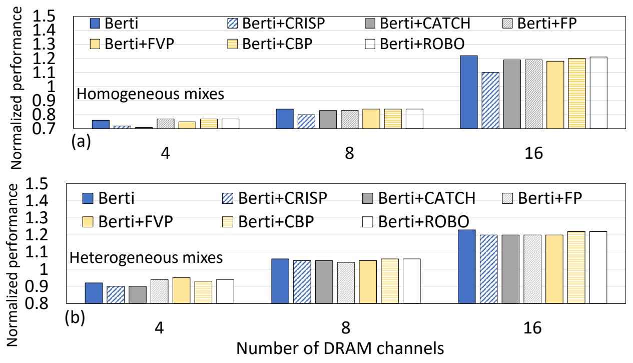

Figure 5: Performance of Berti with critical load predictors normalized to no prefetching for (a) 45 homogeneous and (b) 200 heterogeneous 64-core workload mixes.

state-of-the-art load criticality predictors. We quantify load criticality prediction accuracy as the ratio of correct predictions made by the predictor and load IPs that stall the head of the ROB while servicing an L1 miss. Predictors like CATCH and FVP provide 100% load criticality prediction coverage. However, in terms of load criticality prediction accuracy, these techniques over-predict leading to poor criticality prediction accuracy. One of the primary reasons for this low criticality prediction accuracy is that different instances of the same load IP do not lead to a stall at the head of the ROB. However, most of the existing predictors assume that if a load IP is critical and stalls the head of the ROB, it will be critical all the time (static-critical), which is not the case for all load IPs. For example, conditional branches and branch histories affect the loads and their criticality, resulting in dynamic-critical IPs.

Table 1 summarizes why existing load criticality predictors fail to do a good job in the presence of hardware prefetching. Figure 5 shows the performance of Berti in the presence of critical load predictors. On average, critical load predictors fail to improve the performance of Berti in the presence of constrained DRAM bandwidth.

Prefetch throttlers. Existing prefetch throttling techniques like feedback-directed prefetching (FDP), hierarchical prefetcher aggressiveness controller (HPAC), synergistic prefetcher aggressiveness

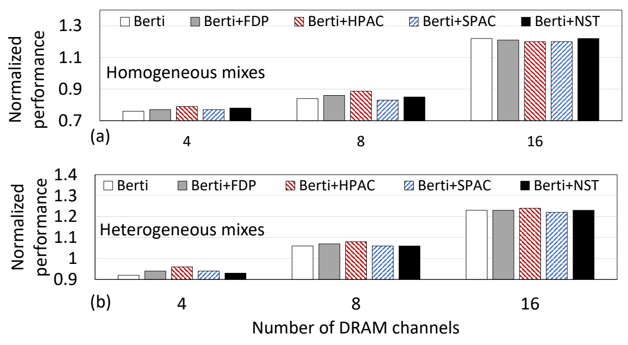

Figure 6: Performance of Berti with prefetch throttlers normalized to no prefetching for (a) 45 homogeneous and (b) 200 heterogeneous 64-core workload mixes.

controller (SPAC), and near side prefetch throttling (NST) [25, 38–40] use prefetch accuracy as one of the primary metrics for controlling prefetch degree and distance. However, state-of-the-art prefetchers are more accurate than IP-stride and stream prefetchers (most of these throttlers are applied to IP-stride and stream prefetchers), and there is a marginal utility in terms of performance improvement when we apply prefetch throttlers on top of prefetchers like Berti. Second, it is not essential to prefetch all the loads even in the case of high prefetch accuracy because it may not lead to any performance improvement, thanks to a large out-of-order instruction window. Third, existing prefetch throttlers operate at a coarse granularity and measure prefetch accuracy and other metrics of interest from a shared memory system point of view, for an epoch of a few kilo instructions or cycles. However, even within an epoch, there are loads that provide high accuracy even if the overall prefetch accuracy for that epoch is low and vice versa.

Figure 6 shows the effectiveness of prefetch throttlers for the Berti prefetcher with constrained DRAM bandwidth. Some of these throttlers are effective in improving performance marginally. However, the performance slowdown is still huge for systems with low DRAM bandwidth.

Prefetch aware shared resource management techniques. With highly accurate prefetchers like Berti, cache pollution at the LLC is not a problem. We corroborate the findings of a recent work [41] that shows that the impact of inaccurate prefetching because of LLC pollution is marginal. Nevertheless, we use the state-of-the-art LLC replacement policy Mockingjay [42] that significantly minimizes the prefetcher-caused negative interference. Our baseline system uses NOC and DRAM controllers that are prefetch aware [32][30]. On average, the utility of these techniques is marginal with an average performance improvement of less than 0.72% when compared to NOC and DRAM controllers that are not prefetch-aware, again thanks to the high prefetch accuracy of Berti. Overall, these techniques are effective for prefetchers with a relatively lower prefetch accuracy.

Table 1: Limitations of existing load criticality predictors in the presence of hardware prefetchers.

CATCH [15]	It uses the dependency graph [13] and tags loads that are in the vicinity of branch predictions as critical even if the loads do not cause stalls. Blind to MLP, low latency loads masked by high latency loads are also flagged as critical.
FP [17]	Relies on the number of stall cycles at the ROB as a metric. It does not try and predict IPs that do not stall significantly. Overall, it marks most of the L3 misses as critical loads.
FVP [16]	Identifies the root of a data dependency chain. So, it ends up identifying all those loads that are likely to delay the execution of other loads/ instructions. The prediction accuracy is low because it ends up tagging excessively (any load that is the producer of any other instruction in its vicinity).
ROBO [18]	Once an IP is flagged critical, throughout the execution, the IP is considered critical. Thus, it is blind to the dynamic nature of an IP's criticality throughout its many recurrences.
CBP [20]	Same as ROBO.
CRISP [19]	Considers only LLC misses and MLP. It does not consider L1 and L2 misses that stall the head of the ROB.

4 CLIP: DESIGN AND IMPLEMENTATION

CLIP advocates for prefetching critical load addresses that have a high chance of getting cache hits (if the prefetcher prefetches the critical load addresses). CLIP considers both *load criticality and fine-grained (per IP) prefetch accuracy.* CLIP drops a prefetch request to an address X if (i) X is not predicted to be critical or (ii) X is predicted to be critical but the prefetcher cannot provide high prefetch accuracy for the trigger IP corresponding to X.

CLIP identifies the IPs that are critical for overall system performance and triggers data prefetching only for those selected IPs provided the prefetch requests generated by these IPs will be accurate. Note that, all the memory requests generated by a given IP are not critical in the presence of high-performing data prefetchers. So, a simple binary classification is not useful for our purpose. Similarly, triggering prefetching only for critical IPs will not help unless the underlying prefetcher is accurate in predicting the future load addresses for all the critical IPs. To mitigate this problem, we propose a two-stage critical and accurate IP predictor that can lead to accurate prefetching for critical loads, only.

Stage I: CLIP shortlists IPs that stall the head of the ROB frequently while waiting for a response from L2, L3, or DRAM. CLIP also uses a signature called critical signature to predict the dynamic nature of the criticality of loads.

Stage II: Next, CLIP selects the load addresses corresponding to critical IPs that can be prefetched accurately by the underlying prefetcher.

Note that a hit at the L1 can also stall the head of the ROB and in fact, it is the case for a majority of the memory accesses as rightly mentioned in one of the recent works [43]. However, an L1 data prefetcher cannot hide the ROB stalls for loads that get L1 hits.

4.1 CLIP: Training

ROB stall and miss-level flags. To find out whether an IP has stalled the retiring process at the head of the ROB, we use a ROB stall flag that is *set* the moment ROB stops retiring instructions. We also use a miss level flag that is appended along with the memory request that goes to the memory hierarchy. The miss level flag is set to one if the load request is serviced by L2, LLC, or DRAM but not by L1. On a load response back to the processor, we check if the ROB stall flag is set and the miss-level flag is non-zero. If it is the case, then we shortlist the corresponding IP by sending it to a structure called the criticality filter. Note that a miss level flag of zero indicates that the load gets a hit either at the L1 or at the LSQ.

Criticality filter and prefetch accuracy tracker. Next, we populate the criticality filter that stores all the IPs that stall the head of the ROB while servicing an L1 miss. We also use a criticality counter to count the number of times an IP stalls the ROB while getting a response from L2, L3, or DRAM. Once an IP crosses a criticality count threshold, we start triggering the underlying prefetching technique and monitor prefetch accuracy at a finer granularity (per IP level) so that we will be able to quantify its *utility* in terms of both, prefetch accuracy and *criticality.* To find out accuracy at a finer granularity, we use two counters: prefetch issue count and prefetch hit count for a given IP. We use a utility buffer that can help us in providing per IP hit count. The utility buffer stores the following information for each prefetch address issued by the prefetcher: the prefetch address X, and the triggering load IP that triggers the prefetch address. In the future, if a demand request gets a match for the prefetch address in the utility buffer that was issued by the prefetcher then we increment the hit count for the corresponding IP in the criticality filter. We measure per IP accuracy at the end of one exploration interval. We define one exploration interval based on the number of L1D misses. At the end of each exploration interval, we check for IPs that deliver a 90% hit rate per IP to initiate prefetching, for the next window. We maintain a bit in the criticality filter that shows whether an IP is critical as per the criticality count and generates highly accurate prefetching. We change this bit after every exploration interval based on the prefetch hit rate and critical count achieved by the IP in that same interval. Note that we use a highly tuned Berti prefetcher that uses the best watermarks (coverage of local deltas) for a 64-core system to achieve the best coverage and accuracy.

4.2 CLIP: Prediction-based Prefetching

Criticality predictor. Once we filter out the IPs that stall the head of the ROB for at least `criticality-count-threshold`[1] number of times and the corresponding IP hit rate is high (90%), then we continue prefetching for the respective IP and use the criticality predictor to predict the criticality of future prefetch addresses that will be generated by the same IP. We need a predictor as the behavior of an IP is dynamic in nature. For example, in `mcf_1554B`, not all prefetch addresses triggered by one IP are critical. We use a new signature called *critical signature* that is created based on a hashed bitwise XOR of an IP, virtual address, global conditional branch history of the last 32 branches, and global criticality history of the last 32 loads. This new signature makes sure that we are predicting the criticality of a given IP for a given load address along with the information about recent control flow and recent criticality history. The critical-signature is inspired by some of the prior works on caching and prefetching [6, 35, 44, 45]. Each entry in the criticality

[1]We use four as the count threshold as it provides the sweet spot in terms of the number of IPs selected, storage, and performance improvement achieved.

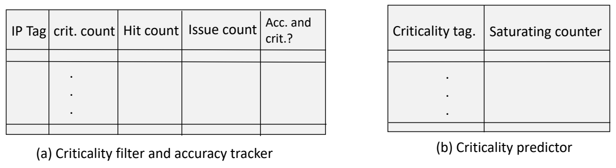
IP Tag
crit. count
Hit count
Issue count
Acc. and crit.?
(a) Criticality filter and accuracy tracker
Criticality tag.
Saturating counter
(b) Criticality predictor

Figure 7: Criticality filter and criticality predictor.

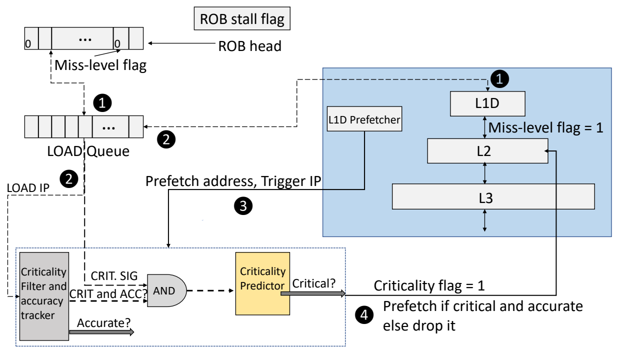

Figure 8: CLIP in action for an L1 prefetcher. The dotted lines show the training phase.

predictor maintains a *criticality tag*. The predictor is indexed by the *critical signature*.

We maintain a k-bit saturating counter that is initialized to $\frac{2^k}{2}$, for each entry of the criticality predictor where we increment the counter on an L1 miss that stalls the head of the ROB and decrement the counter if it leads to an L1 hit or an L1 miss that does not stall the head of the ROB. Note that on average, with Berti prefetcher, L1 hit rate is more than 90% for SPEC CPU2017 benchmarks. We *drop* a prefetch request in two cases: (i) the most significant bit (msb) of a saturating counter is zero or (ii) the msb is one but the per IP accuracy is low. We continue prefetching if the msb is one and the per-IP accuracy is high. With the prefetch request, we append a `criticality flag` of that goes through the entire memory hierarchy. Note that in contrast to existing literature [3, 4, 6] that decides which level in the cache the prefetch address will get filled into, we prefetch all the requests to L1 for Berti because we ensure that we are fetching only for critical IPs that miss at L1 and show high utility in terms of per IP prefetch accuracy. Figure 7(a) shows the structure of a criticality filter with the accuracy tracker. Figure 7 (b) shows the structure of the criticality table. Note that the underlying prefetcher continues to learn memory access patterns irrespective of our prediction.

Which IP for the signature? A prefetch request generated by a prefetcher does not have any IP information. So, we use the IP of a load request that triggers a prefetch request as our IP for the prefetch requests. So, when an address is generated by a prefetcher for a given trigger IP, we probe the criticality predictor with the prefetch address and the triggered IP that will create a critical signature. Note that a prefetch request triggered by one IP can get a hit at the utility buffer for a future request generated by another IP. However, we use the trigger IP as the IP of interest for our prediction as our goal is to decide whether to prefetch or not in the future for a given trigger IP and a given prefetch address.

Phase change and count reset. We reset the hit count and issue count to half of the current value (to maintain hysteresis) after one exploration window (k number of misses to L1D, where k is empirically determined to be just greater than the number of cache lines at the L1D. We also reset all the entries of the criticality filter, accuracy tracker, and criticality predictor, and stop prefetching, on an application phase change. We use accesses per cycle (APC) [46] at L1D as a metric to detect phase change. We monitor the APC of the last 16 windows and take the average APC. If the current APC is different from the average APC by more than 15% then we term it an application phase change. We empirically select the 15% threshold based on APCs of SPEC CPU2017 and GAP traces. This method of phase change detection is used in one of the prior works [18].

CLIP for L2 prefetchers. So far, we show the design of CLIP for an IP-based L1 Berti prefetcher. CLIP can be easily extended for non-IP-based L2 prefetchers like SPP-PPF. The criticality detection and prediction happen at the L2 considering L2 misses that stall the head of the ROB as the critical loads. In case the IP information is not available, then the IP hit rate is replaced by the page hit rate.

Load Criticality conscious NOC and DRAM. With CLIP, we pass the criticality flag along with the prefetch request that is selected by CLIP. We use the same flag to prioritize packets at the NOC and DRAM controller so that demand-loads and critical and accurate prefetch requests get the same priority at the NOC and DRAM controller.

The crux. Compared to prior works like CRISP [19] and FP [17] that also use ROB stalls, the primary difference with CLIP is its critical signature driven by branch history and criticality history, and the dynamic prediction using the critical-signature.

CLIP design choices. CLIP uses the global branch history and criticality history of the last 32 branches. We find that short histories of branch and criticality do not help in improving the prediction accuracy and in fact, the accuracy drops compared to a simple IP-based prediction. Branch history beyond 16 and load criticality history beyond eight start showing improvement in prediction accuracy as it helps the critical signature in distinguishing loads based on control flow and criticality history. We find zero utility in maintaining longer histories: more than 32 for branch and criticality outcomes. We use an exploration window, which is a window with an L1D miss count just greater than the size of the L1D (768 cache lines). Our window is defined as a window of 1024 L1D misses. We observe for SPEC CPU2017 benchmarks, 768 misses occur on average every 87,000 cycles. Smaller exploration windows make the training noisy leading to noisy per IP hit rates. The threshold for per-IP hit rate should be high else CLIP loses its effectiveness. We find that even an IP hit rate of more than 90% is the best threshold. However, it should not be 100% because most of the IPs do not reach 100% per IP hit rate. Similarly, it should not be below 80% because after that CLIP loses its effectiveness. Note that, one can argue about the usage of existing per IP local delta coverage of Berti as a proxy for per IP prefetch accuracy. However, the correlation does not hold true for all the benchmarks.

Figure 8 illustrates the events of interest. In step ❶, a load request passes through the LOAD queue with a miss-level flag that is initialized to zero for all the ROB entries. In step ❷, the load request

Table 2: Storage overhead of CLIP.

Structure		**Storage**
Criticality Filter	32-set, 4-way (128-entry). Each entry: 6-bit IP tag, 2-bit criticality count, 6-bit hit count, 6-bit prefetch count, and Is-critical-and-accurate bit	336 Bytes
Criticality predictor	128 sets, 4-way (512-entry) cache. Each entry: 6-bit criticality tag and 3-bit saturating counter, NRU bit	640 Bytes
ROB extension	Miss level flag, 1 bit per entry (512 entries)	64 Bytes
ROB flag	1 bit	1 bit
Utility buffer	64 entries, each entry 6-bit IP tag, 58-bit cache-line aligned prefetch address	512 Bytes
Branch and criticality history	32-bit array for each	8 Bytes
APC	Two 11 bit registers	22 bits
exploration widow	10 bits for reset count	10 bits
Total		**1.56** KB

Table 3: Simulation parameters of the baseline system.

Core	64 cores, Out-of-order, hashed perceptron branch predictor [52], 4 GHz with 6-issue width, 4-retire width, 512-entry ROB
TLBs	L1 ITLB/DTLB: 64 entries, 4-way, 1 cycle, STLB: 2048 entries, 16-way, 8 cycles
L1I	32 KB, 8-way, 4 cycles
L1D	48 KB, 12-way, 5 cycles, Berti [3]
L2	512 KB 8-way associative, 10 cycles, SRRIP [53], non-inclusive
LLC	2 MB/core, 16-way, 20 cycles, Mockingjay [42], non-inclusive
MSHRs	8/16/32 at L1I/L1D/L2, 64/core at the LLC
Network Router	2-stage wormhole, six virtual channels per Port, five flit buffer depth, eight flits per data packet, and one flit per address packet.
Network Topology	8x8 mesh, each node has a router, processor, private L1 cache, L2 cache, and an LLC slice
DRAM controller	DDR4-3200, Eight channels/64-cores, PADC [30], 64-entry RQ and WQ, reads prioritized over writes, write watermark: 7/8th
DRAM chip	4 KB row-buffer per bank, open page, burst length 16, $t_{RP,\ RCD,\ CAS}$: 12.5 ns

goes to the memory hierarchy, and miss-level flag gets updated, and a response comes back from the memory hierarchy. At the same time, the criticality filter and the prefetch accuracy tracker are updated with the IP if the response comes from L2, L3, or DRAM. In step ❸, the prefetch addresses (X) generated by the prefetcher along with the trigger IP go through the criticality filter and the predictor. A hit in the criticality predictor can have two outcomes. A hit with high confidence (as per the saturating counter) results in prefetching if the per-IP accuracy is high, with a criticality flag appended to the prefetch packet. In case of a miss or low confidence, the prefetch request is dropped and not allocated to the MSHR of L1 (step ❹).

4.3 Storage Overhead

CLIP incurs additional storage in the form of criticality filter, criticality predictor, accuracy tracker, and additional bits (miss-level flag) for each entry at the ROB. We use a per-core criticality filter of 128 entries with an IP tag of six bits and a criticality count (crit. count) of two bits. Our replacement policy uses crit. count bits to find a victim IP (least frequently used policy). The filter uses two counters issue count and hit count to keep track of the number of prefetch requests that are triggered by a given IP and the number of cache hits for that corresponding triggered IP. It also uses an is-critical-and-accurate bit per entry.

We use a utility buffer of 64 entries, which is a circular buffer that stores the recent 64 pairs of prefetch addresses and the corresponding triggering IPs in a temporal order. It is implemented as a content addressable memory (CAM) with input as the prefetch address (X) that was prefetched within a window of the last 64 prefetch requests and the output is the IP that triggered the prefetch address X. On a hit at the CAM, the hit count is incremented for the corresponding IP tag at the criticality filter. This helps in tracking the accuracy of prefetch requests per IP. Our criticality predictor has three ports (LOAD width of two for the request path and a port for LOAD response path), 512 entries (128 sets, 4 ways) with a 6-bit criticality tag that uses a 3-bit saturating counter to predict the load criticality. Table 2 shows the storage overhead of 1.56KB per core. Existing criticality predictors take three to five KB per core whereas existing throttlers need four to 10s of KBs per core.

Is 512 entries enough? Our criticality predictor is of just 512 entries that are indexed by the critical signature, thanks to our filter that makes sure we do not predict the criticality of loads for all the IPs. As our criticality filter and prefetch accuracy tracker filter out most of the IPs, it helps in reducing the storage budget of our criticality predictor by 4.75 times on average.

With 512 entries, it is possible that there will be negative interference because of *aliasing*. However, we find that our hash function used as part of the critical signature scatters the concurrent load requests (signatures) to different entries. For load addresses that get mapped to the same entry of the criticality predictor but recur after a long gap, we find that the saturating counter used per entry takes care of the recent behavior of one load, only. We also see a positive correlation, especially for load addresses triggered by one IP within a loop. Note that this observation holds true for SPEC CPU2017 benchmarks [47] only. For client/server [48, 49] and CloudSuite [50] workloads, we need around 2048 entries to mitigate the aliasing and other interference problem. However, for most of the client/server workloads, 512 entries are enough. For example, `server_013` trace [51] has 32 thousand IPs within a window of 30M instructions. However, only nine IPs are critical.

5 EVALUATION

Simulation methodology. We use a modified version of ChampSim [54], a trace-driven simulator used for the 2nd and 3rd Data Prefetching Championships (DPC-2 [55] and DPC-3 [56]). Recent prefetching proposals [3, 4, 6, 7, 57] are also coded and evaluated on ChampSim. The recently modified ChampSim extends the one provided with the DPC-3 with a decoupled front-end [58] and a detailed memory hierarchy support for address translation that further improves the baseline performance. We extend it further by adding a detailed network-on-chip (NOC) with sliced LLCs. We also integrate DRAMSim [59] with ChampSim as it does not model a detailed DRAM with all the DRAM timing constraints. ChampSim provides a knob called `low-bandwidth` to simulate a single-core with low DRAM bandwidth to understand the constrained bandwidth effects. However, we find that the knob is not realistic and we simulate a detailed 64-core system to see the impact of low DRAM bandwidth on a many-core system. Table 3 summarizes our system configuration, mimicking an Intel Sunny Cove microarchitecture [60–62].

Workloads. We use the simpoint [63] traces from SPEC CPU2017 [47], GAP [2, 64], CloudSuite [50], and client and server traces provided as part of Value Prediction Championship (CVP) [48, 49, 51]. We limit our study to memory-intensive SPEC, GAP, and CloudSuite

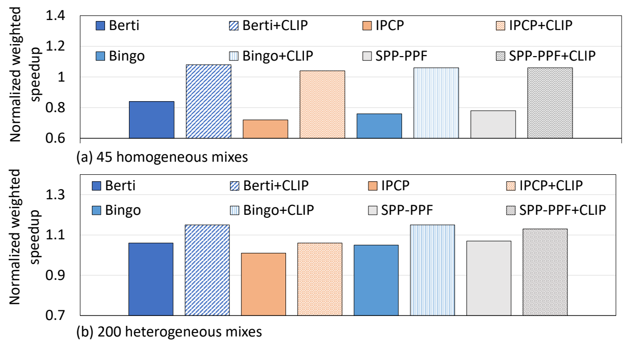

Figure 9: CLIP with state-of-the-art prefetchers for (a) 45 homogeneous and (b) 200 heterogeneous mixes on a 64-core system with eight DRAM channels.

traces, those that showed at least one miss per kilo-instruction (MPKI) with a 2MB LLC/core in our modeled baseline system. All GAP traces and 45 SPEC CPU2017 traces are memory-intensive. We use all the CloudSuite and CVP traces that include client/server traces. We provide a detailed analysis of 64-core homogeneous mixes with eight DRAM channels where all the cores of a many-core system run the same benchmark in the SPEC RATE mode. We also show results for heterogeneous mixes.

We evaluate CLIP with 64-core multi-core simulations. We warm up the caches for 100M instructions per core (6400M instructions for a 64-core system) and collect statistics for the next 200M per-core instructions/core. For CloudSuite and client/server traces, we simulate 30M instructions per core. For our simulations, we use 45 64-core homogeneous mixes and 200 randomly generated heterogeneous mixes based on SPEC CPU2017 and GAP benchmarks. For each mix, when a core finishes its 200M instructions, it gets replayed until all the cores finish their respective 200M instructions. We report performance in terms of weighted speedup [65] with respect to no prefetching as no-prefetching performs better than state-of-the-art prefetchers in the presence of constrained DRAM bandwidth. Weighted speedup is equivalent to system throughput which accounts for the number of programs completed per unit of time. We provide detailed analysis for homogeneous mixes as it is relatively easier to reason about with all the 64 cores running the same trace.

Energy model. We also report the dynamic energy consumption of the memory hierarchy. We obtain the energy consumption of reads and writes to tag and data arrays at each cache level and DRAM with CACTI-P [66] and Micron DRAM power calculator [67]. Then, we compute the total energy expenditure by accounting for the number of accesses of each type across the memory hierarchy. We use the 7 nm process technology.

Evaluated Techniques. We compare the effectiveness of CLIP with high-performing L1D and L2 prefetchers like Berti, IPCP, Bingo and SPP-PPF. However, we focus mostly on Berti as Berti is the high-performing prefetcher among the evaluated prefetchers. As a related comparison, We compare CLIP with Hermes and DSPatch too. For all prefetchers, we use a highly tuned implementation as provided by the authors and tune it again for the parameters mentioned in Table 3.

Key results. Figures 9 shows the normalized performance improvement (weighted speedup) of CLIP with state-of-the-art L1 and L2 prefetchers. CLIP is equally effective across all the prefetchers. For the most accurate prefetcher Berti, CLIP provides an improvement of 24% and 9% for 45 homogeneous and 200 heterogeneous workload mixes, respectively. Note that 200 64-core heterogeneous mixes were created from SPEC CPU2017 and GAP benchmarks, randomly with no bias towards any specific benchmark. Compared to homogeneous mixes, heterogeneous mixes have mixes where there is no significant performance drop even in the case of low DRAM bandwidth. This happens for the mixes where half of the benchmarks are almost cache friendly and their respective LLC MPKI is closer to one. With large LLCs of 128MB, these mixes get most of their demand hits at the LLC.Next, to understand the subtleties better, we show a detailed performance analysis of CLIP with Berti for 45 homogeneous workload mixes.

5.1 Performance analysis

Figure 10 shows the normalized performance improvement of Berti with CLIP, for 45 64-core homogeneous mixes. On average, CLIP enhances Berti's performance by 24% (16% slowdown becomes 8% improvement). With CLIP, out of 45 64-core homogeneous mixes, only three mixes show performance slowdowns whereas, without CLIP, more than 26 mixes show performance slowdowns. Note that the contribution of criticality-conscious NOC and DRAM is just 2.8% out of 24% of enhanced performance. Also, on average, 77.5% of the performance benefit comes from criticality filtering and prediction, and the rest comes from accuracy filtering.

Latency improvement. To understand the primary contributors of performance benefits, Figure 11 shows improvement in average L1 miss latencies with CLIP when compared with Berti without CLIP. On average across 45 64-core mixes, the average L1 miss latency drops from 168 cycles to 132 cycles, with maximum improvements of more than 900 cycles for one of the mixes that contain 64 copies of `lbm`. The improvement in latency also helps in improving prefetch lateness (on average, it improves from 13% to 5.8%). Note that CLIP improves the average miss latency. However, this comes at the cost of the prefetch coverage as CLIP drops prefetch requests that are not critical and accurate.

Miss coverage. Figure 12 shows the average drop in prefetch coverage at L1, L2, and LLC. There is a significant drop at the L1 (7%) whereas, at L2 and LLC, the coverage drops by 2% and 3%, respectively. This trend provides an interesting trade-off in terms of miss rate and miss. latency as CLIP improves performance even if there is a drop in coverage thanks to the average improvement in L1 miss latency (36 cycles).

Load criticality prediction accuracy and coverage. The improvement in average miss latency is caused by high critical load prediction accuracy as compared to the best of critical load predictors(Figure 13). However, a high prediction accuracy does not necessarily result in high load criticality prediction coverage (Figure 14). On average, CLIP provides a load criticality prediction accuracy of 93% that helps in covering 76% of the critical loads that stall the head of ROB while getting responses from L2, LLC, and DRAM.

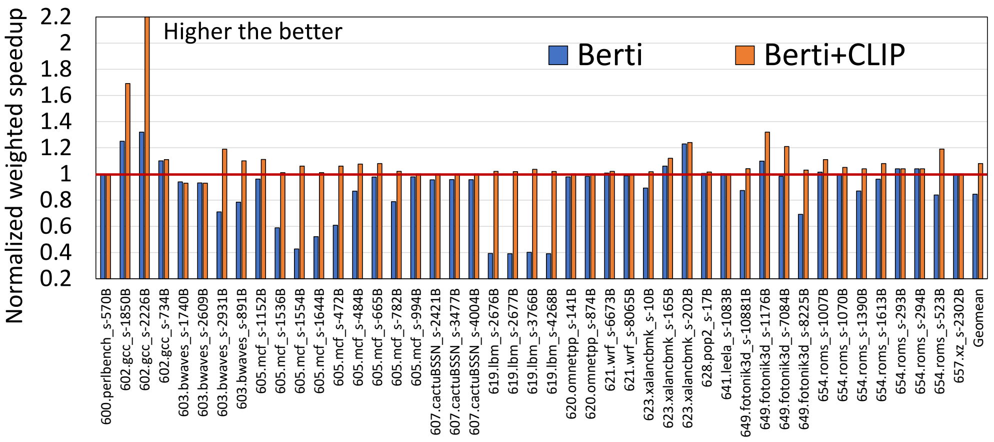

Figure 10: Performance normalized to no prefetching for 45 64-core homogeneous mixes with eight DRAM channels.

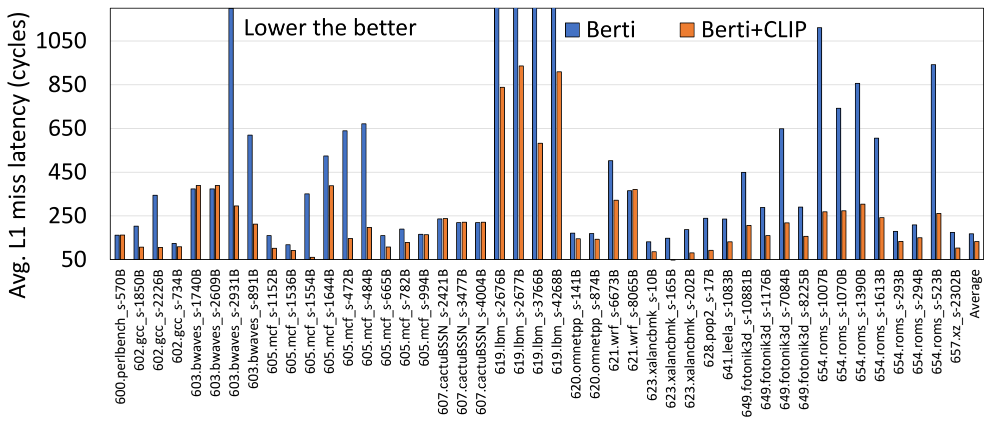

Figure 11: Average L1 miss latency for 45 64-core homogeneous mixes with eight DRAM channels.

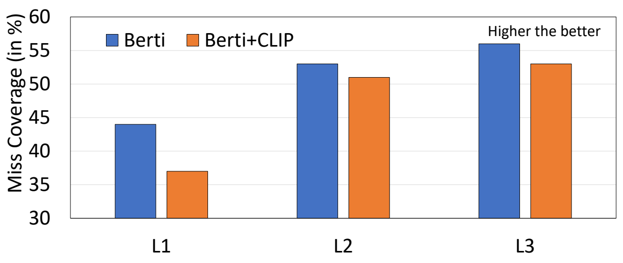

Figure 12: L1, L2, and LLC miss coverage (in %) for Berti running on a 64-core system with eight DRAM channels. L1 and L2 miss coverage is averaged across 64 cores.

This high load criticality prediction accuracy and coverage led to a significant drop in the prefetch requests generated by Berti.

Number of critical IPs. Figure 15 shows the absolute number of critical IPs (divided into static-critical and dynamic-critical ones) as detected by CLIP over a window of 200M simulated instructions. The dynamic IPs are the IPs that sometimes behave like critical IPs and sometimes not. For 20 mixes, the number of critical and accurate IPs selected by CLIP is just 20, whereas the actual number of IPs is in the hundreds. On average, around 50% of the IPs are dynamic-critical IPs. Overall, there are a few IPs that stall the head of the ROB while getting a response from L2, LLC, or DRAM, which helps in reducing the prefetch traffic.

Figure 16 shows the drop in prefetch requests generated by Berti in the presence of CLIP. On average, there is a 50% drop in prefetch requests and as high as 90% (for cactubssn). Note that the improvement in the accuracy of critical and accurate IP

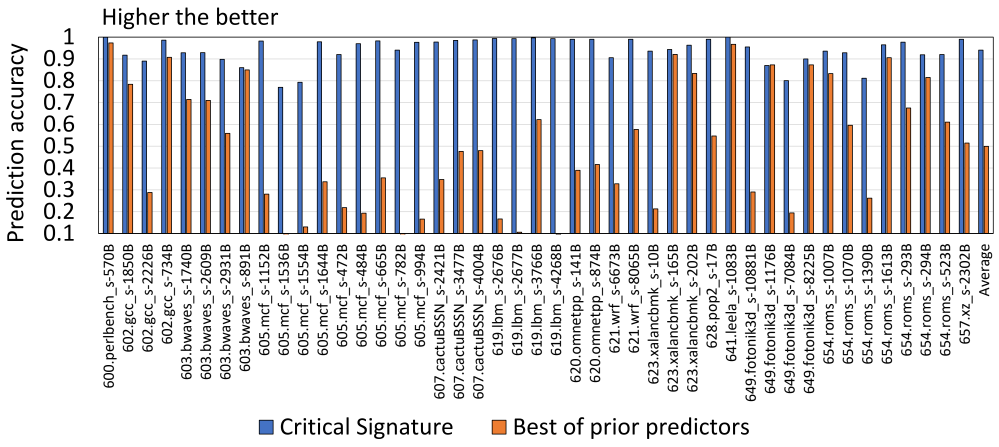

Figure 13: Critical load prediction accuracy of Berti+CLIP averaged across 64 cores for homogeneous mixes with eight DRAM channels.

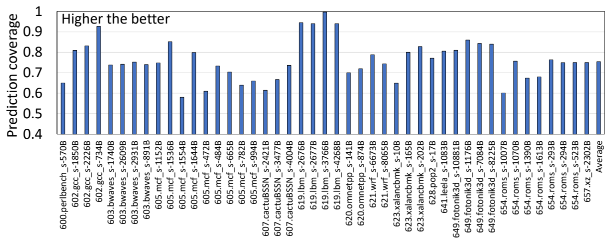

Figure 14: Critical load prediction coverage of Berti+CLIP averaged across 64 cores for homogeneous mixes with eight DRAM channels.

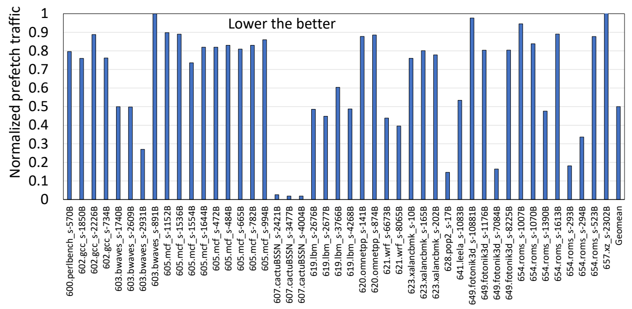

Figure 16: Reduction in prefetch requests with Berti+CLIP normalized to Berti for 45 64-core homogeneous mixes with eight DRAM channels.

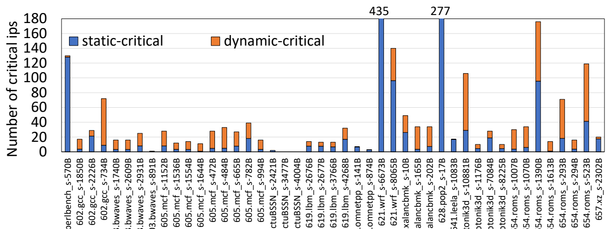

Figure 15: Number of critical and accurate IPs (static-critical and dynamic-critical) per core selected by CLIP for 45 64-core SPEC CPU2017 homogeneous mixes over a window of 200M instructions.

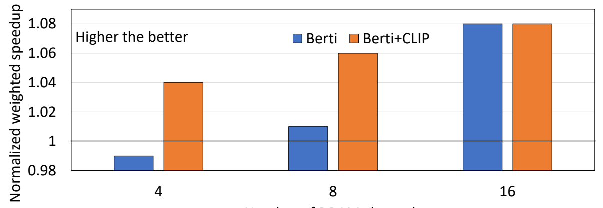

Figure 17: Performance normalized to no prefetching averaged across CloudSuite and CVP 64-core homogeneous workloads.

predictor results in an increase in overall prefetch accuracy, with the average prefetch accuracy of Berti improving from 82.9% to 94.2%. The benchmarks that see the maximum improvements are `mcf, omnetpp, cactubssn`. For `cactubssn_2421B`, prefetch accuracy improved from 12% with Berti to 89.65% with Berti+CLIP. For `mcf_1536B`, CLIP improves Berti's accuracy from 51.1% to 93%.

Dynamic energy. CLIP improves run-time that directly leads to improvement in static energy. For homogeneous mixes, CLIP improves the dynamic energy of the memory hierarchy by 18.21% over Berti, thanks to an average 50% reduction in prefetch traffic. For heterogeneous mixes, the improvement in dynamic energy is just less than 7%. Note that, in our energy calculations, we include

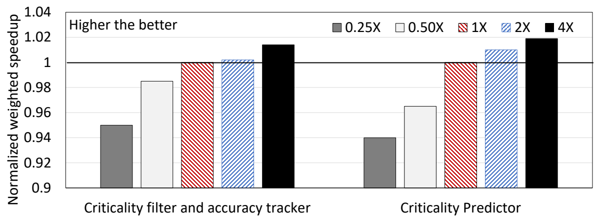

Figure 18: Sensitivity study across 45 homogeneous and 200 heterogeneous mixes: Size of the hardware tables.

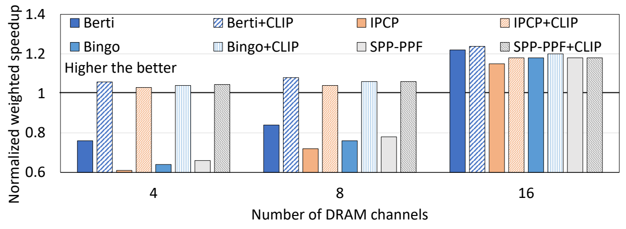

Figure 19: CLIP with state-of-the-art prefetchers with different numbers of DRAM channels for 45 homogeneous mixes.

the dynamic energy consumed by the additional structures used by CLIP.

CLIP with CloudSuite and CVP workloads. We evaluate CLIP with CloudSuite [50, 68] and CVP traces [48, 49] in the form of 64-core homogeneous mixes. Note that the evaluated prefetchers improve performance by less than 10% even in the case of 64 DRAM channels for a 64-core system. So, compared to SPEC CPU2017 homogeneous mixes, the problem of constrained DRAM bandwidth is not significant as the prefetchers find it hard to predict the future addresses for most of the CloudSuite and CVP benchmarks. Figure 17 shows the effectiveness of CLIP for CloudSuite and CVP benchmarks with different numbers of DRAM channels.

5.2 Sensitivity Studies

CLIP table size. To understand the sensitivity of hardware tables used by CLIP, we sweep both tables with the following table sizes: 0.25X, 0.5X, 2X, and 4X of the proposed table size. Note that we sweep through these sizes keeping the other table fixed to the baseline size. Increasing the table sizes to 2X and 4X provides marginal performance improvement with few outliers like `621.wrf` and `628.pop2` that show an additional 3.23% performance improvement. However, when we reduce the table size to 0.5X and 0.25X, we see a performance drop of more than 7% compared to the proposed Berti+CLIP. Figure 18 shows the trend.

CLIP and the number of DRAM channels. Figures 19 and 20 show the performance of CLIP for L1 and L2 prefetchers with four, eight, and 16 DRAM channels for 45 and 200 64-core homogeneous and heterogeneous mixes, respectively. For four and eight DRAM channels, CLIP is highly effective and as expected for 16 DRAM channels the effectiveness is marginal in terms of performance

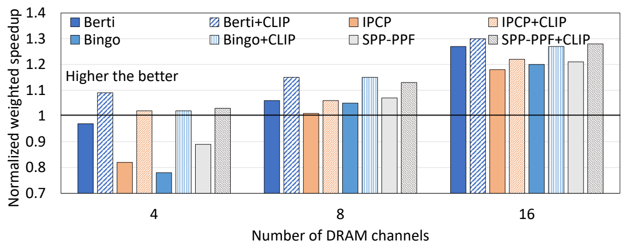

Figure 20: CLIP with state-of-the-art prefetchers with different numbers of DRAM channels for 200 heterogeneous mixes.

improvement. This shows that CLIP is extremely effective for many-core systems with constrained DRAM bandwidth.

CLIP and the number of cores. As a sensitivity study, we evaluate CLIP with 8, 16, 32, 64, and 128 cores with four, eight, 16, 32, and 64 DRAM channels. The effectiveness of CLIP remains similar for all these configurations. Based on our experiments, we find that the effectiveness of CLIP is not significant if we have at least one DRAM channel for two to four cores.

CLIP with different LLC sizes. We perform a sensitivity study based on the LLC capacity as the LLC capacity plays a big role in terms of requests that go to the DRAM. In our baseline, we use 2MB LLC/core. Next, we sweep the size of the LLC from 512KB/core to 4MB/core keeping the number of DRAM channels at eight. For homogeneous workloads, as expected, the performance of Berti for a 4MB LLC/core improves with an average slowdown of 9% as compared to a 16% slowdown with 2MB LLC/core. Similarly, Berti shows a performance slowdown of 29% with 512KB LLC/core. The effectiveness of CLIP improves with smaller LLC/core and in all cases, CLIP makes sure, we get performance improvement with hardware prefetching.

5.3 CLIP vs. Hermes and DSPatch

Hermes[35] with Berti does a relatively better job than Berti alone, thanks to its high prediction accuracy that accurately predicts the loads that will go to DRAM. However, Berti+CLIP outperforms Berti+Hermes for four and eight DRAM channels. There are two primary reasons for this improvement: (i) Not all the loads that go to DRAM are critical for overall system performance. Hermes does not pay attention to L2 and LLC hit that stall the head of the ROB for a significant amount of time. (ii) Hermes does not reduce the number of requests that go to the DRAM, significantly (less than 1%) whereas CLIP reduces the DRAM traffic significantly thanks to high prediction accuracy in identifying critical loads that can lead to accurate prefetching. Figure 21 shows the effectiveness of Hermes and CLIP with Berti. For low DRAM bandwidth (four and eight DRAM channels), CLIP outperforms Hermes. However, for a system with 16 DRAM channels Hermes beats CLIP, which was expected and we corroborate the findings of Hermes for many-core systems with ample DRAM bandwidth. DSPatch [36] as expected performs poorly as compared to CLIP because, for most of the

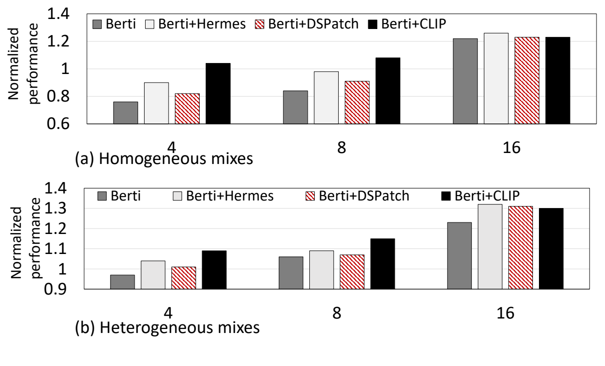

Figure 21: Hermes, DSPatch, and CLIP with Berti for 45 homogeneous and 200 heterogeneous 64-core mixes.

benchmarks, the DRAM bandwidth utilization is low in the constrained DRAM bandwidth scenarios. The low DRAM bandwidth utilization forces DSPatch to optimize for prefetch coverage compromising prefetch accuracy, which increases the average L1 miss latency with a marginal improvement in the prefetch coverage. Overall, the tradeoff between miss latency and miss coverage does not help and Berti+DSPatch performs poorly as compared to CLIP for a 64-core system with four and eight DRAM channels (Figure 21).

Dynamic CLIP. CLIP is a technique that improves the performance of hardware prefetchers when the available DRAM bandwidth is low. However, it is not a useful technique for systems with high per-core DRAM bandwidth (e.g., only a few cores out of 64 cores are active and utilizing the eight DRAM channels). As a future work, similar to DSPatch, a dynamic version of CLIP can be explored that can turn off CLIP in the case of systems with high per-core DRAM bandwidth.

6 CONCLUSION

Hardware prefetchers lose their effectiveness in the case of many-core systems with constrained DRAM bandwidth. We showed that prior works on prefetcher throttling, prefetch-aware resource management techniques, and load criticality-based prefetching are not effective in mitigating this problem. We proposed CLIP, a highly accurate fine-grained critical load predictor that can detect critical loads that stall the head of the ROB while getting a response from L2, LLC, or DRAM. CLIP detects the critical loads and filters out the loads that will lead to accurate prefetching, making sure that the prefetch requests generated by a prefetcher are actually for loads that contribute to the overall performance improvement. CLIP enhances the effectiveness of prefetchers like Berti by 24% and 9% for homogeneous and heterogeneous mixes on a 64-core system with eight DRAM channels, respectively. CLIP provides this performance improvement with a storage overhead of 1.56KB per core.

7 ACKNOWLEDGEMENT

We would like to thank all the anonymous reviewers for their insightful comments and suggestions. Special thanks to Neelu Kalani, Ramya Prabhu, and Sheetal for the discussions during the early phase of the this work. We would also like to thank members of the CASPER group especially Sumon, Shubham, Anubhav, and Kritish for their feedback on the initial draft. Finally, thanks to CAPS research group members especially Alberto Ros and Sawan Singh for their feedback on the draft. This work is supported by the Qualcomm faculty award 2022 and Google India research award 2022.

REFERENCES

[1] "SPEC CPU 2017 traces for champsim." https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/, Feb. 2019.

[2] "GAP traces for champsim." https://utexas.app.box.com/s/2k54kp8zvrqdfaa8cdhfquvcxwh7yn85/folder/132804598561, Mar. 2021.

[3] A. Navarro-Torres, B. Panda, J. Alastruey-Benedé, P. Ibáñez, V. Viñals-Yúfera, and A. Ros, "Berti: an accurate local-delta data prefetcher," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 975–991, 2022.

[4] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *47th Int'l Symp. on Computer Architecture (ISCA)*, pp. 118–131, June 2020.

[5] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *49th Int'l Symp. on Microarchitecture (MICRO)*, pp. 60:1–60:12, Oct. 2016.

[6] E. Bhatia, G. Chacon, S. H. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *46th Int'l Symp. on Computer Architecture (ISCA)*, pp. 1–13, June 2019.

[7] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *25th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pp. 399–411, Feb. 2019.

[8] I. Xeon, "Xeon platinum," 2023.

[9] A. EPYC, "Amd epyc 7702." https://www.amd.com/en/products/cpu/amd-epyc-7702p, May 2019.

[10] A. Ryzen, "Amd ryzen threadripper," 2019.

[11] B. Panda and S. Balachandran, "TCPT - thread criticality-driven prefetcher throttling," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, Edinburgh, United Kingdom, September 7-11, 2013* (C. Fensch, M. F. P. O'Boyle, A. Seznec, and F. Bodin, eds.), p. 399, IEEE Computer Society, 2013.

[12] B. Panda and S. Balachandran, "Introducing thread criticality awareness in prefetcher aggressiveness control," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014* (G. P. Fettweis and W. Nebel, eds.), pp. 1–6, European Design and Automation Association, 2014.

[13] B. Fields, S. Rubin, and R. Bodík, "Focusing processor policies via critical-path prediction," in *28th Int'l Symp. on Computer Architecture (ISCA)*, pp. 74–85, June 2001.

[14] S. Subramaniam, A. Bracy, H. Wang, and G. H. Loh, "Criticality-based optimizations for efficient load processing," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 419–430, 2009.

[15] 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 *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 96–109, 2018.

[16] S. Bandishte, J. Gaur, Z. Sperber, L. Rappoport, A. Yoaz, and S. Subramoney, "Focused value prediction: Concepts, techniques and implementations presented in this paper are subject matter of pending patent applications, which have been filed by intel corporation," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 79–91, 2020.

[17] R. Manikantan and R. Govindarajan, "Focused prefetching: Performance oriented prefetching based on commit stalls," in *Proceedings of the 22nd Annual International Conference on Supercomputing*, ICS '08, (New York, NY, USA), p. 339–348, Association for Computing Machinery, 2008.

[18] N. S. Kalani and B. Panda, "Instruction criticality based energy-efficient hardware data prefetching," *IEEE Computer Architecture Letters*, vol. 20, no. 2, pp. 146–149, 2021.

[19] H. Litz, G. Ayers, and P. Ranganathan, "Crisp: Critical slice prefetching," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, (New York, NY, USA), p. 300–313, Association for Computing Machinery, 2022.

[20] S. Ghose, H. Lee, and J. F. Martínez, "Improving memory scheduling via processor-side load criticality information," *SIGARCH Comput. Archit. News*, vol. 41, p. 84–95, jun 2013.

[21] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *13th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pp. 63–74, Feb. 2007.

[22] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated control of multiple prefetchers in multi-core systems," in *42nd Int'l Symp. on Microarchitecture (MICRO)*, pp. 316–326, Dec. 2009.

[23] B. Panda and S. Balachandran, "CAFFEINE: A utility-driven prefetcher aggressiveness engine for multicores," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 12, pp. 30:1–30:25, Aug. 2015.

[24] B. Panda and S. Balachandran, "Expert prefetch prediction: An expert predicting the usefulness of hardware prefetchers," *IEEE Computer Architecture Letters*, vol. 15, pp. 13–16, Jan. 2016.

[25] B. Panda, "Spac: A synergistic prefetcher aggressiveness controller for multi-core systems," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3740–3753, 2016.

[26] W. Heirman, K. D. Bois, Y. Vandriessche, S. Eyerman, and I. Hur, "Near-side prefetch throttling: Adaptive prefetching for high-performance many-core processors," in *27th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pp. 28:1–28:11, Nov. 2018.

[27] V. Jimenez, A. Buyuktosunoglu, P. Bose, F. P. O'Connell, F. Cazorla, and M. Valero, "Increasing multicore system efficiency through intelligent bandwidth shifting," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 39–50, 2015.

[28] J. W. C. Fu, J. H. Patel, and B. L. Janssens, "Stride directed prefetching in scalar processors," in *Proceedings of the 25th Annual International Symposium on Microarchitecture, Portland, Oregon, USA, November 1992* (W. W. Hwu, ed.), pp. 102–110, ACM / IEEE Computer Society, 1992.

[29] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy, "Power4 system microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–25, 2002.

[30] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, "Prefetch-aware dram controllers," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, pp. 200–209, 2008.

[31] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt, "Improving memory bank-level parallelism in the presence of prefetching," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, (New York, NY, USA), p. 327–336, Association for Computing Machinery, 2009.

[32] N. C. Nachiappan, A. K. Mishra, M. T. Kandemir, A. Sivasubramaniam, O. Mutlu, and C. R. Das, "Application-aware prefetch prioritization in on-chip networks," in *International Conference on Parallel Architectures and Compilation Techniques, PACT '12, Minneapolis, MN, USA - September 19 - 23, 2012* (P. Yew, S. Cho, L. DeRose, and D. J. Lilja, eds.), pp. 441–442, ACM, 2012.

[33] V. Seshadri, S. Yedkar, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Mitigating prefetcher-caused pollution using informed caching policies for prefetched blocks," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 51:1–51:22, 2014.

[34] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware shared resource management for multi-core systems," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, (New York, NY, USA), p. 141–152, Association for Computing Machinery, 2011.

[35] R. Bera, K. Kanellopoulos, S. Balachandran, D. Novo, A. Olgun, M. Sadrosadati, and O. Mutlu, "Hermes: Accelerating long-latency load requests via perceptron-based off-chip load prediction," in *55th IEEE/ACM International Symposium on Microarchitecture, MICRO 2022, Chicago, IL, USA, October 1-5, 2022*, pp. 1–18, IEEE, 2022.

[36] R. Bera, A. V. Nori, O. Mutlu, and S. Subramoney, "Dspatch: Dual spatial pattern prefetcher," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*, pp. 531–544, ACM, 2019.

[37] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Domino temporal data prefetcher," in *24th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pp. 131–142, Feb. 2018.

[38] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pp. 63–74, 2007.

[39] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated control of multiple prefetchers in multi-core systems," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 316–326, 2009.

[40] W. Heirman, K. D. Bois, Y. Vandriessche, S. Eyerman, and I. Hur, "Near-side prefetch throttling: Adaptive prefetching for high-performance many-core processors," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, PACT '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[41] R. Bera, A. V. Nori, , O. Mutlu, and S. Subramoney, "Dspatch: Dual spatial pattern prefetcher," in *52nd Int'l Symp. on Microarchitecture (MICRO)*, pp. 531–544, Oct. 2019.

[42] I. Shah, A. Jain, and C. Lin, "Effective mimicry of belady's min policy," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 558–572, 2022.

[43] S. Shukla, S. Bandishte, J. Gaur, and S. Subramoney, "Register file prefetching," in *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022* (V. Salapura, M. Zahran, F. Chong, and L. Tang, eds.), pp. 410–423, ACM, 2022.

[44] A. Lai, C. Fide, and B. Falsafi, "Dead-block prediction & dead-block correlating prefetchers," in *Proceedings of the 28th Annual International Symposium on Computer Architecture, ISCA 2001, Göteborg, Sweden, June 30-July 4, 2001* (P. Stenström, ed.), pp. 144–154, ACM, 2001.

[45] D. A. Jiménez and E. Teran, "Multiperspective reuse prediction," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2017, Cambridge, MA, USA, October 14-18, 2017* (H. C. Hunter, J. Moreno, J. S. Emer, and D. Sánchez, eds.), pp. 436–448, ACM, 2017.

[46] D. Wang and X.-H. Sun, "Apc: A novel memory metric and measurement methodology for modern memory systems," *IEEE Transactions on Computers*, vol. 63, no. 7, pp. 1626–1639, 2014.

[47] Standard Performance Evaluation Corporation, "SPEC CPU2017," 2017.

[48] "CVP-1 public traces." https://perscido.univ-grenoble-alpes.fr/datasets/DS382, 2018.

[49] "CVP-1 private traces." https://perscido.univ-grenoble-alpes.fr/datasets/DS384, 2018.

[50] "Cloudsuite traces for champsim." https://www.dropbox.com/sh/pgmnzfr3hurlutq/AACciuebRwSAOzhJkmj5SEXBa/CRC2_trace?dl=0&subfolder_nav_tracking=1, Nov. 2017.

[51] "The First Championship Value Prediction." https://www.microarch.org/cvp1/cvp1/index.htm, June 2018.

[52] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *7th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pp. 197–206, Jan. 2001.

[53] A. Jaleel, K. B. Theobald, S. C. S. Jr., and J. S. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *37th Int'l Symp. on Computer Architecture (ISCA)*, pp. 60–71, June 2010.

[54] "ChampSim simulator." http://github.com/ChampSim/ChampSim, May 2020.

[55] "The 2nd data prefetching championship (dpc-2)," June 2015.

[56] "The 3rd data prefetching championship (dpc-3)," June 2019.

[57] M. Shakerinava, M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Multi-lookahead offset prefetching," in *The 3rd Data Prefetching Championship*, June 2019.

[58] G. Reinman, B. Calder, and T. Austin, "Fetch directed instruction prefetching," in *32nd Int'l Symp. on Microarchitecture (MICRO)*, pp. 16–27, Dec. 1999.

[59] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. L. Jacob, "Dramsim3: A cycle-accurate, thermal-capable DRAM simulator," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 110–113, 2020.

[60] A. Fog, "The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers," 2020. Available at https://www.agner.org/optimize/microarchitecture.pdf.

[61] "SunnyCove microarhcitecture." https://en.wikichip.org/wiki/intel/microarchitectures/sunny_cove, May 2018.

[62] "SunnyCove microarhcitecture latency." https://www.7-cpu.com/cpu/Ice_Lake.html, May 2018.

[63] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *10th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pp. 3–14, Sept. 2001.

[64] S. Beamer, K. Asanović, and D. A. Patterson, "The GAP benchmark suite," *CoRR*, vol. abs/1508.03619, Aug. 2015.

[65] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreading processor," in *ASPLOS-IX Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 12-15, 2000* (L. Rudolph and A. Gupta, eds.), pp. 234–244, ACM Press, 2000.

[66] 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 *2011 Int'l Conf. on Computer-Aided Design (ICCAD)*, pp. 694–701, Nov. 2011.

[67] "Micron dram power calculator." https://www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf, Dec. 2015.

[68] M. Ferdman, A. Adileh, Y. O. Koçberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *17th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pp. 37–48, Mar. 2012.