A Two Level Neural Approach Combining Off-Chip Prediction with Adaptive Prefetch Filtering

Alexandre Valentin Jamet
alexandre.jamet@bsc.es
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (UPC)

Georgios Vavouliotis
georgios.vavouliotis2@huawei.com
Huawei Zurich Research Center

Daniel A. Jiménez
djimenez@acm.org
Texas A&M University

Lluc Alvarez
lluc.alvarez@bsc.es
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (UPC)

Marc Casas
marc.casas@bsc.es
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (UPC)

2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 979-8-3503-9313-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCA57654.2024.00046

***Abstract*—To alleviate the performance and energy overheads of contemporary applications with large data footprints, we propose the *Two Level Perceptron (TLP)* predictor, a neural mechanism that effectively combines predicting whether an access will be off-chip with adaptive prefetch filtering at the first-level data cache (L1D). TLP is composed of two connected microarchitectural perceptron predictors, named *First Level Predictor (FLP)* and *Second Level Predictor (SLP)*. FLP performs accurate off-chip prediction by using several program features based on virtual addresses and a novel selective delay component. The novelty of SLP relies on leveraging off-chip prediction to drive L1D prefetch filtering by using physical addresses and the FLP prediction as features. TLP constitutes the first hardware proposal targeting both off-chip prediction and prefetch filtering using a multi-level perceptron hardware approach. TLP only requires 7KB of storage.**

To demonstrate the benefits of TLP we compare its performance with state-of-the-art approaches using off-chip prediction and prefetch filtering on a wide range of single-core and multi-core workloads. Our experiments show that TLP reduces the average DRAM transactions by 30.7% and 17.7%, as compared to a baseline using state-of-the-art cache prefetchers but no off-chip prediction mechanism, across the single-core and multi-core workloads, respectively, while recent work significantly increases DRAM transactions. As a result, TLP achieves geometric mean performance speedups of 6.2% and 11.8% across single-core and multi-core workloads, respectively. In addition, our evaluation demonstrates that TLP is effective independently of the L1D prefetching logic.

I. Introduction

Emerging workloads from various domains [10], [17], [18], [19], [24] have large data footprints that are orders of magnitude larger than the capacity of current cache hierarchies [15]. These workloads frequently trigger DRAM accesses, spending a large portion of their execution time waiting for data transfers to and from DRAM to complete with a detrimental effect on performance and energy [6], [8], [9], [12], [51], [57].

Prior work has proposed several techniques to mitigate the performance and energy overheads of these applications. These techniques can be broadly classified into four categories: (i) off-chip prediction schemes that predict whether a memory access will result in a DRAM access or hit in the cache hierarchy [13], [26], [34], [42], [54], (ii) aggressive data prefetching with adaptive filters to ensure that only correct prefetches will be issued [14], (iii) cache bypassing that avoids caching blocks that will not be referenced in the near future [25], [30], [32], [45], [48], [52], and (iv) disruptive cache designs and optimizations for specific workload types [5], [20], [23], [36], [38], [41], [47], [56]. This work focuses on the first two categories and aims at combining their benefits in a cost-effective manner.

Despite their potential for determining the location of requested data in the memory hierarchy, previously proposed off-chip predictors [13], [26] have important drawbacks that undermine their potential for boosting the performance of the memory subsystem while hindering their implementation in real-world designs. For example, the state-of-the-art off-chip predictor [13] triggers two memory accesses, one to DRAM and a second regular request to the cache hierarchy, when it predicts that the corresponding load access will be served from DRAM. While this approach can potentially reduce the latency of a load request that ends up being served from DRAM, it may also significantly increase the number of DRAM transactions. This work shows that, although effective, the state-of-the-art off-chip predictor significantly increases the number of DRAM transactions, which is a critical aspect in bandwidth-constrained scenarios. In addition, our analysis indicates that a large fraction of the inaccurate off-chip predictions is actually served by the first-level data cache (L1D). Therefore, a microarchitectural scheme that selectively delays the off-chip predictions with modest confidence until the L1D lookup is resolved has potential to significantly reduce the number of useless DRAM transactions and deliver higher performance.

Previous approaches have successfully applied prefetch filtering at the lower level caches [14], [43], [44], [58]. However, these approaches are not agile since they are typically optimized on top of specific prefetch engines, incur significant area overheads, and are not exposed to program features that are very valuable to produce accurate predictions (*e.g.*, a complete sequence of accessed virtual addresses). This work argues that the concept of off-chip prediction can be leveraged to form

2378-203X/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCA57654.2024.00046

effective prefetch filters for L1D. Specifically, our analysis demonstrates that the vast majority of the L1D prefetch requests served from DRAM are inaccurate.

To address our findings and improve the performance or memory-intensive workloads, we propose the *Two Level Perceptron (TLP)* predictor. TLP constitutes the first hardware proposal targeting both off-chip prediction and prefetch filtering using a multi-level perceptron hardware approach. TLP is composed of two connected microarchitectural perceptron predictors: the *First Level Predictor (FLP)* and the *Second Level Predictor (SLP)*. FLP is a perceptron hardware predictor located near the core that employs a novel mechanism to reduce the number of DRAM accesses by selectively delaying off-chip predictions when needed. SLP is a perceptron predictor located alongside the L1D. The novelty of SLP relies on leveraging off-chip prediction to drive L1D prefetch filtering using physical addresses as well as the FLP prediction as features. Our evaluation illustrates that TLP yields significantly higher performance than the state-of-the-art off-chip predictor [13] and prefetch filtering scheme [14] across a large set of single-core and multi-core workloads.

This paper makes the following contributions:

- We design and propose *Two Level Perceptron (TLP)* predictor, a scheme composed of two connected perceptron predictors: FLP and SLP. FLP reduces the pressure on the memory subsystem using a novel selective delay mechanism. SLP leverages off-chip prediction to guide prefetch filtering in the L1 data cache. TLP is the first hardware proposal targeting both off-chip prediction and prefetch filtering. TLP only requires 7KB of storage.
- We compare TLP with the state-of-the-art off-chip predictor, Hermes [13], the state-of-the-art prefetch filter, PPF [14], and a combination of both. Our evaluation considers 55 single-core and 200 multi-core workloads. When considering a system that uses IPCP [39] as L1D prefetcher, TLP reduces the average number of DRAM transactions by 30.7% and 17.7%, as compared to a baseline that uses IPCP as L1D prefetcher but no off-chip prediction mechanism, across the single-core and multi-core workloads, respectively, while state-of-the-art approaches significantly increase DRAM transactions. As a result, TLP achieves geometric mean performance speedups of 6.2% and 11.8% across single-core and multi-core workloads, respectively. When considering a scenario with the Berti [37] L1D prefetcher, TLP also outperforms Hermes, PPF, and a combination of them in both single-core and multi-core contexts since it significantly reduces DRAM accesses.

II. Background

A. Off-Chip Prediction

Emerging workloads spanning various domains [10], [17], [18], [19], [24], have a key property in common: massive working set sizes that do not fit in the existing cache hierarchies [15], making cache management a major performance bottleneck for processor design. Indeed, recent work [6], [8], [9], [12], [51], [57] shows that these workloads spend up to 80% of their total execution time waiting for DRAM.

To address the high-latency load requests of these emerging applications, prior work [13], [26], [34], [42], [54] has introduced the concept of *off-chip prediction*. The core idea behind off-chip prediction is to predict whether a memory access will eventually result in a DRAM access or in a hit in the cache hierarchy (L1D, L2C, LLC). Prior work in the domain can be classified in two categories depending on their prediction strategy: i) predict which cache level (L1D, L2C, LLC) will provide a hit, if any [26], [42], and ii) predict whether the cache hierarchy as a whole will provide a hit or not [13]. A representative work from the first category is Level Prediction (LP) [26], a scheme that dynamically predicts where in the memory hierarchy a demanded memory block is most likely to be found. A representative scheme of the second category is Hermes [13], an adaptive perceptron-based off-chip predictor that routes demand load requests directly to DRAM when it is confident that the load will miss in all cache levels.

Hermes [13] is the state-of-the-art microarchitectural off-chip prediction scheme. At the core of Hermes, there is a perceptron predictor composed of several prediction tables, one per selected program feature, similar to prior work on perceptron-based microarchitectural prediction: from branch prediction [21], [28], [29] to cache replacement policies [30], [48] and other intelligent modules [14].

Hermes is consulted to provide a prediction upon demand load requests. If the prediction is positive (*i.e.*, the demand load request is predicted to go off-chip), the core issues two requests: one regular request to the cache hierarchy, that might go down to DRAM, and another speculative request that fetches the cache line from DRAM in an attempt to hide the latency cost of accessing the caches. When a demand request eventually returns to the core to be consumed, the training logic of Hermes compares the original prediction with the actual outcome and accordingly updates the weights in the prediction tables.

B. Prefetch Filtering

Hardware prefetching is a technique that proactively fetches blocks in the cache hierarchy before they are explicitly requested by a core. Hardware prefetchers need to deal with two metrics that are at odds with one another: miss coverage and prefetching accuracy. Aggressive prefetchers typically have high coverage but low accuracy while conservative prefetchers tend to have low coverage and high accuracy.

To handle the coverage-accuracy trade-off, smart prefetch filters and throttling schemes able to accurately identify useless prefetch requests and discard them have been proposed [14], [43], [44], [58]. An effective prefetch filter would increase the accuracy of a hardware prefetcher without harming its coverage, resulting in higher performance by enabling better cache management.

The state-of-the-art prefetch filter is the Perceptron-based Prefetch Filter (PPF) [14], a perceptron predictor that uses several program features to filter out inaccurate prefetch requests, increasing the accuracy of the underlying prefetcher. Although effective, PPF has two limitations. First, PPF is built and optimized on top of a specific prior prefetcher [33], thus

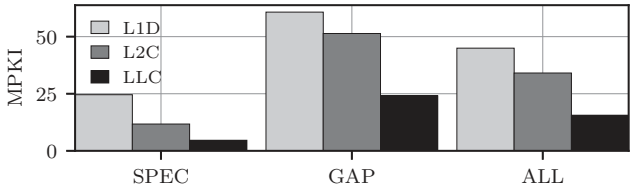


Fig. 1: MPKI of all caches (L1D, L2C, LLC) across the SPEC (SPEC CPU 2006 and SPEC CPU 2017) and GAP workloads.

it requires significant engineering effort as well as feature exploration and tuning to make it filter effectively the requests of other prefetchers. Second, PPF incurs 40KB of storage overhead, which hinders its adoption by commercial designs.

III. Motivation

This section motivates the need for better off-chip predictors and highlights the potential of leveraging the concept of off-chip prediction to apply effective prefetch filtering for the L1D cache. Section III-A characterizes the cache behavior of contemporary applications, showing that a large fraction of the memory accesses that miss in the L1D result in a DRAM access. Section III-B analyzes the behavior of Hermes [13], the state-of-the-art off-chip predictor presented in Section II-A, in both single-core and multi-core contexts. Our analysis indicates that Hermes significantly increases DRAM bandwidth consumption, especially in multi-core contexts. Therefore, performance improvements are possible by reducing the number of additional DRAM transactions triggered by Hermes. Section III-C focuses on L1D cache prefetching and characterizes the inaccurate prefetches issued by two state-of-the-art L1D prefetchers, and reveals that off-chip prediction can drive the design of effective prefetch filters for L1D. Section V presents in detail our simulation infrastructure and all the considered workloads.

A. Cache Behavior of Modern Workloads

Prior work discussed in Section II-A shows that the majority of demand load requests of applications featuring huge data working sets miss in all levels of the cache hierarchy, triggering many DRAM accesses. This section analyzes the cache behavior of all single-core workloads presented in Section V.

Figure 1 shows the average Misses per Kilo Instruction (MPKI) rates of L1D, L2C and LLC. On average the MPKIs of L1D, L2C, and LLC are 45.0, 34.1, and 15.6, respectively. Therefore, 34.7% of L1D misses eventually require a DRAM access. Remarkably, workloads from domains such as graph processing put more pressure on the cache hierarchy, resulting in more frequent DRAM accesses. Indeed, Figure 1 reveals that, on average, the graph-processing (GAP) workloads trigger a DRAM access for 39.7% of the L1D misses.

> ***Finding 1.*** *A large fraction of the demand load requests triggered by applications with large working set sizes miss in all cache levels.*

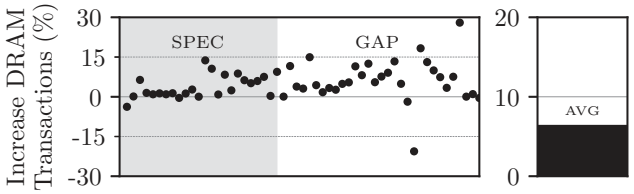


Fig. 2: Increase in DRAM transactions due to Hermes off-chip predictions relative to a baseline without off-chip prediction mechanism. Lower is better.

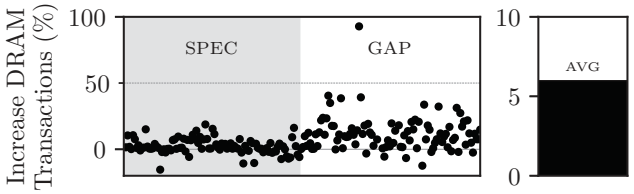


Fig. 3: Increase in DRAM transactions due to Hermes off-chip predictions relative to a baseline without off-chip prediction mechanism in the 4-core context. The x-axis ticks represent 200 different 4-core workload mixes of SPEC and GAP workloads. Lower is better.

B. Impact of Hermes

This section quantifies the impact of Hermes on the number of DRAM transactions processed by the main memory in both single-core and multi-core contexts and identifies features that can potentially increase Hermes' efficiency and performance. This analysis is conducted using the methodology and the set of workloads presented in Section V.

1) DRAM Transactions: Figures 2 and 3 illustrate the impact of Hermes on the number of DRAM transactions in single-core and multi-core contexts, respectively. The x-axis display different SPEC and GAP workloads. Both SPEC and and GAP workloads are separately sorted considering the LLC MPKI. The y-axis displays the increase in terms of DRAM transactions that Hermes incurs over a baseline without any off-chip predictor.

Figures 2 and 3 indicate that Hermes places high pressure on DRAM, especially in the multi-core scenario, since it issues many speculative DRAM requests. Regarding the single-core evaluation, Hermes increases the number of DRAM transactions by 5.2%, 6.6%, and 6.4% over the baseline system that does not use any off-chip predictor for the SPEC, GAP, and all workloads combined, respectively. Figure 3, which presents the impact of Hermes on DRAM transactions in a multi-core context, shows that Hermes significantly increases DRAM transactions. Specifically, Hermes increases the average number of DRAM transactions by 2.2%, 9.6%, and 6.0% over the multi-core baseline for the SPEC mixes, GAP mixes, and all mixes, respectively. Notably, the increase in DRAM transactions for the GAP workloads is significantly higher than the increase for the SPEC workloads; this happens because the GAP suite is made of graph-processing applications that have much larger data working sets than the general-purpose SPEC CPU workloads.

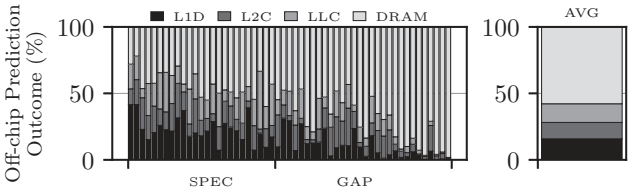


Fig. 4: Location of a block upon a Hermes off-chip prediction.

Finding 2. *Hermes significantly increases the number of DRAM accesses in both single-core and multi-core contexts, especially for graph-processing applications.*

2) Analysis of Hermes Predictions: This section characterizes the off-chip predictions of Hermes (*i.e.*, cases where Hermes triggered a speculative DRAM request), and motivates potential design and functionality enhancements. To do so, we categorize the off-chip predictions of Hermes depending on where the corresponding block is located in the memory hierarchy (L1D, L2C, LLC, DRAM). Specifically, we consider the following categories: (i) block resides in L1D, (ii) block resides in L2C, (iii) block resides in LLC, and (iv) block resides in DRAM. Predictions belonging to categories (i), (ii), and (iii) correspond to inaccurate off-chip predictions since the block is located in the cache hierarchy while category (iv) represents accurate off-chip predictions since the block is not present in the caches. Figure 4 presents this breakdown for both SPEC and GAP single-core workloads, using the methodology that Section V describes. Both SPEC and GAP workloads are separately sorted based on LLC MPKI, similar to Figure 2.

Figure 4 shows that 42.2% of the total off-chip predictions are inaccurate since the corresponding blocks reside in the cache hierarchy (L1D , L2C, or LLC). Notably, a large fraction of the load requests corresponding to an inaccurate off-chip prediction are served by the L1D cache. Specifically, 17.7% of the total off-chip predictions are useless since their corresponding block resides in the L1D. In other words, delaying Hermes to issue an off-chip prediction after the L1D lookup completion would significantly reduce DRAM transactions. However, constantly delaying the off-chip predictions of Hermes until the L1D lookup is completed would result in suboptimal performance gains since more than 50% (57.8% on average in Figure 4) of the Hermes off-chip predictions are accurate. In these cases, issuing the DRAM access before the L1D access is resolved provides latency benefits. Thus, a mechanism to decide whether or not an off-chip prediction of Hermes should be issued before or after the L1D access completion has the potential to significantly reduce the number of useless DRAM accesses triggered by Hermes.

Finding 3. *Selectively delaying Hermes off-chip predictions until the L1D lookup is resolved has the potential to significantly reduce the number of useless DRAM transactions and deliver higher performance.*

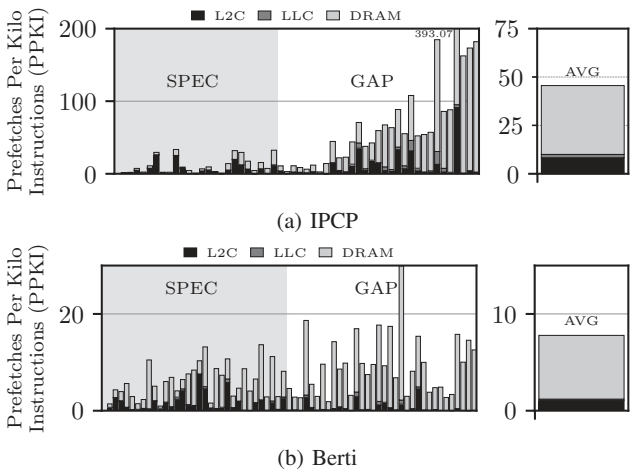


Fig. 5: Location where the inaccurate L1D prefetch requests are served across two state-of-the-art L1D prefetchers. Both SPEC and GAP workloads are separately sorted based on LLC MPKI, similar to Figure 2.

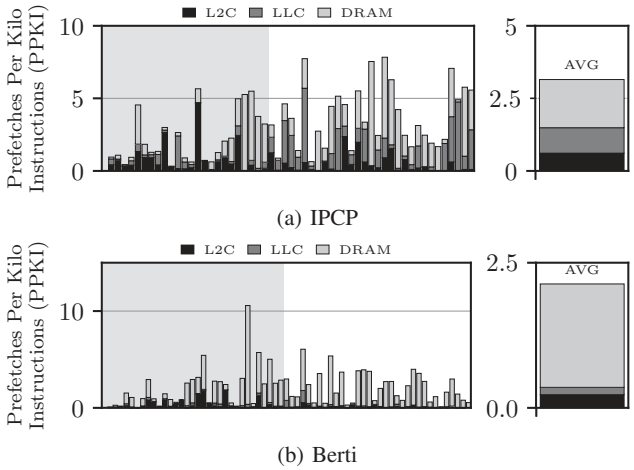


Fig. 6: Location where the accurate L1D prefetch requests are served across two state-of-the-art L1D prefetchers. Both SPEC and GAP workloads are separately sorted based on LLC MPKI, similar to Figure 2.

C. Off-Chip Prediction for L1D Prefetch Filtering

This section characterizes the inaccurate prefetches issued by L1D prefetchers across the considered single-core SPEC and GAP workloads. To do so, we consider two state-of-the-art L1D prefetchers: (i) the Instruction Pointer Classification Prefetcher (IPCP) [39], and (ii) the Berti prefetcher [37].

Figure 5a presents the breakdown of the inaccurate L1D prefetches issued by IPCP depending on where in the memory hierarchy (L2C, LLC, DRAM) the corresponding prefetch request is served. To do so, we use the Prefetches Per Kilo Instruction (PPKI) metric. Overall, 18.2%, 3.8%, and 78% of the total inaccurate prefetch requests are served by L2C, LLC, and DRAM, respectively. We observe that the majority of the

inaccurate prefetch requests are the ones that were served from DRAM. This behavior is more prevalent for the GAP workloads since these workloads have more complex patterns than SPEC. In addition, we compare the accurate L1D prefetches of IPCP that were served from DRAM with the inaccurate ones. Our analysis indicates that, on average, 95.2% of the prefetches that were served from DRAM are inaccurate (the rest 4.8% is accurate prefetches) for our set of workloads; the ratio is higher for the GAP benchmarks (96.7%) than for the SPEC (82%) since the former exhibit more complex memory access patterns. Figure 5b presents the breakdown of Berti's inaccurate prefetches depending on where in the memory hierarchy the corresponding request is served, similar to Figure 5a. Overall, we observe the same behavior. The vast majority of Berti´s useless prefetch requests are served from DRAM and there is high probability for a prefetch that goes all the way to DRAM to fetch a block to be inaccurate, making a strong case for exploiting the off-chip prediction technique to design an L1D prefetch filter. Figure 6 indicates how the overall number of accurate prefetchers served from DRAM (1.7 and 0.5 PPKI for IPCP and Berti) is much smaller than the inaccurate prefetchers served from DRAM (35.6 and 6.6 for IPCP and Berti).

Thus, we conclude that accurately predicting whether a prefetch will be served from DRAM can provide a useful hint regarding the usefulness of the corresponding prefetch. Consequently, an accurate off-chip predictor can be leveraged as an L1D prefetch filter. Finally, we observe similar behavior in the multi-core context.

Finding 4. *Off-chip prediction can be leveraged to design an effective prefetch filtering scheme for L1D.*

These four findings demonstrate that the state-of-the-art approach for off-chip prediction incurs a significant overhead in terms of additional DRAM transactions, and that there are opportunities to eliminate this overhead and boost performance by unifying off-chip prediction and prefetch filtering. Section IV presents a novel approach that unifies these two techniques in a single method.

IV. Two Level Perceptron Prediction

This paper proposes the *Two Level Perceptron (TLP)* predictor, a two level cooperative prediction scheme that leverages neural methods to perform cost-effective off-chip prediction for demand load requests combined with adaptive L1D prefetch filtering. TLP is composed of two microarchitectural perceptron predictors named First Level Perceptron (FLP) predictor and Second Level Perceptron (SLP), respectively. TLP is motivated by the four findings of Section II.

Findings 1 and 2 demonstrate that Hermes exacerbates the pressure on the memory subsystem that modern workloads inject, particularly for memory intensive workloads from domains like graph-processing. In addition, Finding 3 indicates that a selective delay mechanism can potentially mitigate this pressure. The FLP design, described in Section IV-A, is motivated by these three findings. FLP includes a novel selective delay mechanism to only trigger speculative requests to DRAM for highly confident off-chip predictions. Finding 4 indicates the potential of guiding prefetch filtering via off-chip prediction. The SLP design, described in Section IV-B, exploits this potential and incorporates a novel feature based on FLP output. Section IV-C presents our complete proposal, TLP, a multi-Level perceptron combining FLP and SLP. TLP is novel in three ways: i) it incorporates a new selective delay mechanism to reduce pressure on the memory subsytem; ii) it leverages off-chip prediction to guide prefetch filtering; and iii) it constitutes the first hardware proposal targeting both off-chip prediction and prefetch filtering. In addition, TLP is the first multi-level perceptron hardware approach that can be effectively applied due to its low area requirements. Finally, Section IV-D details the hardware requirements of TLP.

Legacy Hermes features	• PC $\oplus$ cacheline offset • PC $\oplus$ byte offset • PC + first access • Cacheline offset + first access • Last-4 load PCs
Leveling feature	• FLP prediction + cacheline offset

TABLE I: List of features used by the FLP and the SLP.

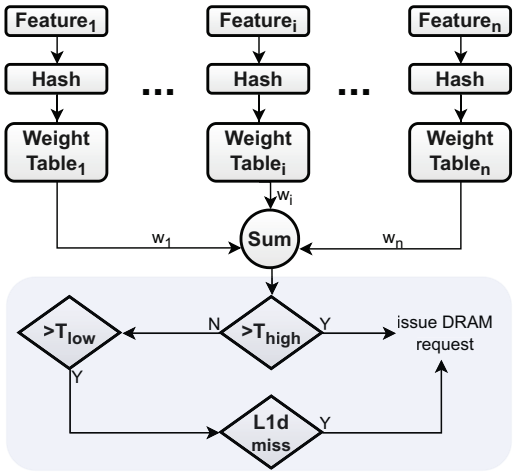


Fig. 7: Flowchart of FLP. Diamonds indicate decision points.

A. First Level Perceptron (FLP) Predictor

FLP is an off-chip predictor based on a micro-architectural hashed perceptron predictor that dynamically decides whether to consume the off-chip prediction in the core (*i.e.*, in parallel with the L1D lookup since L1D caches are typically implemented as VIPT structures), or upon an L1D miss. This delayed decision mechanism is driven by two threshold values: τ_{high} and τ_{low}. Perceptron confidence values greater than τ_{high} indicate a high probability for the corresponding load request to miss in all cache levels, values lower than τ_{low} indicate the opposite, and intermediate values indicate the need for delaying the decision upon an L1D miss. FLP takes into account several program features to predict whether a demand load

request will miss in the cache hierarchy or not. Our exploration indicates that the features used in the original Hermes [13] work provide good predictions and that adding more features provides marginal benefits. Thus, FLP uses the same set of features as the original Hermes prediction, presented in Table I (*c.f.*: Legacy Hermes features). The selected features correlate the probability of a demand load request going off-chip with a history of PCs and accessed memory regions. Each FLP feature is associated with a weight table which is composed of confidence counters.

Figure 7 presents a flowchart of FLP's operation and illustrates how the confidence value produced by FLP is used to drive the off-chip prediction mechanism. Upon a demand load request, FLP is consulted by the core. FLP uses the selected program features to index its weight tables, then reads out and sums the corresponding weights to produce a confidence value. Then, the confidence value is compared to the τ_{high} threshold. A confidence value greater than τ_{high} indicates a high probability for the corresponding load request to miss in all caches. In this case, FLP issues a speculative DRAM request from the core in parallel with the L1D lookup as first-level caches are typically implemented as VIPT structures. However, if the confidence value does not exceed τ_{high} but does exceed the τ_{low} threshold, the probability of the load demand request to miss in all cache levels is not considered high enough to benefit from a speculative DRAM request. Thus, the request is flagged as predicted off-chip and is sent to the L1D cache. In Section III-A, we observed that the probability of a load demand requiring an access to the DRAM tends to rise with each successive cache level traversed. Therefore, if this request results in an L1D miss, the flag bit is read, and a speculative DRAM request is issued from the L1D. Thus, FLP addresses our third analysis finding and avoids sending useless DRAM requests for loads that might hit in the on-chip caches. Finally, if the confidence value exceeds none of the two thresholds, the demand load request continues like a normal request without triggering speculative DRAM access.

The FLP is trained upon completing a memory access, (*i.e.,* when the memory block is returned to the core from the cache hierarchy). When the request comes back to the core, the FLP checks if the request was a true off-chip load request (*i.e.*, if this request required a DRAM access). If the request was a true off-chip load request, the predictor's corresponding weights are trained positively. Conversely, if the request was not a true off-chip load request, the predictor's corresponding weights are trained negatively.

B. Second Level Perceptron (SLP) Predictor

The SLP is a perceptron-based off-chip predictor conceived to be used in the context of L1D prefetch filtering. The SLP design is motivated by the observation that off-chip prediction can be leveraged to design effective L1D prefetch filters. Section III-C justifies this observation. SLP can be used to improve the performance of any generic L1D prefetcher since it makes no assumption regarding the L1D prefetcher design.

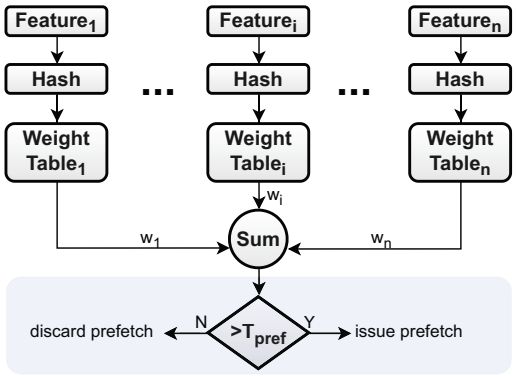


Fig. 8: Flowchart of SLP. Diamonds indicate decision points.

SLP uses several program features to perform effective prefetch filtering at L1D. Our feature exploration indicates that FLP's features can be also used in the context of L1D prefetch filtering. Therefore, SLP uses the same as FLP, presented in Table I, but these features are adapted to use physical addresses in place of virtual addresses as SLP is placed after the L1D cache. Additionally, SLP makes use of a new feature denoted as FLP prediction + offset in Table I. This feature combines the FLP output bit of the cache block from which the prefetch request originated with the offset of the prefetched cache block in its physical memory page. The rationale of this feature is to correlate the probability of an L1D prefetch request going off-chip when a certain cache line offset is touched with the off-chip prediction decision related to the block that triggered the prefetch request. The SLP produces a binary off-chip prediction when an L1D prefetch request is issued.

Figure 8 presents a flowchart of the SLP operation. SLP is consulted when the L1D prefetcher issues a prefetch request. The confidence value is built similarly to the FLP. The output value is compared to the τ_{pref} threshold. If it exceeds τ_{pref}, the prefetch is considered as eventually requiring a DRAM access and, therefore, likely useless. In this situation, the prefetch request is discarded. Conversely, if the confidence value does not exceed τ_{pref}, the prefetch request is processed as usual by the cache hierarchy.

SLP is trained in a similar way as FLP (cf. Section IV-A). Upon the completion of an L1D prefetch request, the predictor's weights are trained positively or negatively depending on whether or not the prefetch request was served off-chip.

C. Building a Multi-Level Perceptron Predictor

This section presents our complete proposal, Two Level Perceptron (TLP) predictor, a hierarchical neural prediction scheme that combines FLP and SLP predictors, presented in Sections IV-A and IV-B, respectively.

Figure 9 shows the design and the operation of TLP. Upon a load demand access, the core consults FLP to obtain a confidence value $Conf$ driving the off-chip prediction ❶. This prediction can give one of the three following outcomes: (i) the load request is predicted to be off-chip with high confidence ($Conf > \tau_{high}$), thus a speculative DRAM request is thrown

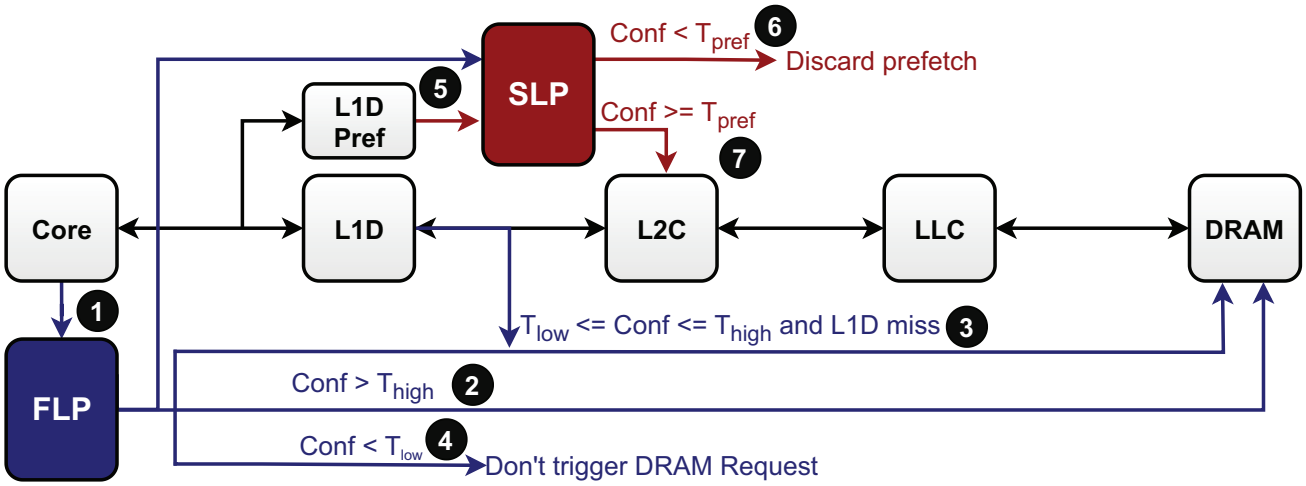


Fig. 9: Organization and operation of the Two Level Perceptron (TLP) prediction mechanism.

from the core (2) besides the regular load demand access; (ii) the load request is predicted to be off-chip with low confidence ($\tau_{low} \leq Conf \leq \tau_{high}$), thus the speculative DRAM request will be thrown only if the load misses in the L1D (3); (iii) the load request is predicted to be on-chip; therefore no additional action is taken (4) besides triggering the regular demand access. Metadata relative to the prediction (hashed PC, history of last load PCs, and perceptron confidence value) are stored in the matching Load Queue entry for later training and an off-chip prediction tag is set in the load request thrown to the cache hierarchy depending on the FLP prediction.

SLP is consulted upon L1D prefetch requests (5). To make a prediction, SLP takes as input the metadata attached to the prefetch request and the off-chip prediction tag attached to the demand load request from which the prefetch request originates. This information is used to produce an off-chip $Conf$ prediction specific to L1D prefetch request. This prediction can result in two possible outcomes: i) the prefetch request is predicted to be off-chip ($Conf < \tau_{pref}$) and the prefetch request is discarded (6), and ii) the prefetch request is predicted to be on-chip ($Conf \geq \tau_{pref}$) and the prefetch request is processed as usual by the cache hierarchy (7). Similarly to FLP, SLP stores metadata relative to its prediction in the L1D MSRH entries for later training.

The training routines of the FLP and the SLP are triggered upon completion of the corresponding requests, (*i.e.*, for FLP when the load request returns to the core and SLP when the prefetch request is served), as Sections IV-A and IV-B explain.

D. TLP Hardware Requirements and Latency

Table II breaks down the hardware requirements of TLP into its various components. TLP only requires 6.98KB of additional storage per core. Similar to Hermes [13], FLP requires 3.21KB of storage for its prediction tables and 0.42KB of storage for the metadata in the Load Queue entries for training purposes. SLP requires 3.29KB of storage for its prediction tables as we make the addition of a new feature, and 0.06KB of additional storage in the L1D MSHR entries for training purposes. In total, TLP requires only 6.99KB of extra storage, making it a low overhead design combining off-chip prediction and prefetch filtering. Similarly to previous work [13], we consider a 6-cycles latency when either FLP or SLP trigger a speculative DRAM access.

Component	Description	Size
FLP	• Perceptron weight tables: 2.58KB • Page buffer: 0.63KB	3.21KB
SLP	• Perceptron weight tables: 2.66KB • Page buffer: 0.63KB	3.29KB
Load Queue metadata	Hashed PC: 32b; Last-4 PC: 10b; First access: 1b; perceptron confidence value: 5b	0.42KB
L1D MSHR metadata	Hashed PC: 32b; Last-4 PC: 10b; First access: 1b; perceptron confidence value: 5b; prediction: 1b	0.06KB
Total		6.98KB

TABLE II: Storage overhead of TLP.

V. Experimental Methodology

A. Simulation Methodology

We evaluate our proposal using ChampSim [22], a detailed trace-based simulator that models a 4-wide out-of-order CPU. We consider a baseline system similar to the Intel Cascade Lake microarchitecture [1]. Table III presents the specific configuration details of the baseline system. Regarding hardware prefetching, we use state-of-the-art prefetchers in both the L1D and the L2C. At the L1D level we consider both the Instruction Pointer Classification Prefetcher (IPCP) prefetcher [39] and the Berti prefetcher [37]. At the L2 level we use the SPP prefetcher [33], which brings prefetched blocks into either the L2C or the LLC depending on its internal prefetch logic.

B. Workloads

Our evaluation considers a large set of applications spanning different benchmark suites. Specifically, we consider workloads

Component	Description
Branch Predictor	hashed-perceptron
CPU	3.8 GHz, 4-wide out-of-order processor 6-stage pipeline, 224-entries re-order buffer
L1 ITLB	64-entry, 4-way, 1cc, 8-entry MSHR, LRU
L1 DTLB	64-entry, 4-way, 1cc, 8-entry MSHR, LRU
L2 TLB	1536-entry, 12-way, 8cc, 16-entry MSHR, LRU
L1I Cache	32 kB, 8-way, 4cc, 10-entry MSHR, LRU
L1D Cache	32 kB, 8-way, 4cc, 10-entry MSHR, LRU, IPCP [39] or Berti [37]
L2 Cache	1 MB, 16-way, 10cc, 16-entry MSHR, LRU, SPP [33]
LLC	1.375 MB per core, 11-way, 36/56cc, 64-entry MSHR, LRU
DRAM	16 GB, DDR4 SDRAM single-core data-rate: 12.8 GB/s per core multi-core data-rate: 3.2 GB/s per core $t_{RP} = t_{RCD} = t_{CAS} = 24$ cycles

TABLE III: System configuration.

from SPEC CPU 2006 [2] and SPEC CPU 2017 [3] benchmark suites. In addition, we consider graph-processing applications included in the GAP benchmark suite [11]. Specifically, we use six graph-processing kernels from GAP: Breadth-First Search (BFS) is a fundamental graph traversal algorithm; Page Rank (PR) iteratively updates per-vertex ranks until convergence; Connected Components (CC) applies the Shiloach-Vishkin [46] algorithm to compute the largest connected components of the graph; Betweenness Centrality (BC) uses the Brandes algorithm [16] to approximate the per-vertex centrality scores; Triangle Count (TC) counts the number of triangles in the graph; and, finally, Single-source Shortest Paths (SSSP) uses δ-stepping [35] to return the distance of all vertices of a graph to a given source vertex. Table IV shows the main characteristics of these six applications, including the size of property array elements, and input parameters such as the execution style (push or pull), or the use of frontiers.

For each graph-processing kernel, we consider 6 different input graphs that feature different sizes and distributions of node degrees (*e.g.,* power-law, normal, etc.). Different degree distributions produce different memory access patterns. For instance, when node degrees are distributed following a power-law function, there are a few highly connected graph nodes that yield more data reuse opportunities than vertices with a few connections. Table V lists all considered input graphs.

In addition, we only consider workloads for which the baseline system shows LLC MPKI greater than 1. This filters out workloads and leaves us with 31 GAP workloads and 24 SPEC workloads.

All workload traces have been obtained using the SimPoint methodology [40] to identify at least one SimPoint representative of each workload. Each SimPoint is 1 billion instructions long and characterizes a different phase of these workloads, similar to prior work [30], [32], [45], [49].

Section VI refers to SPEC 2006 and SPEC 2017 workloads as SPEC and to GAP workloads as GAP.

C. Single-Core Evaluation

Our set of single-core workloads contains 55 distinct workloads: 31 possible combinations of graph-processing kernels and input graphs, described in Section V-B, and 24 SPEC CPU

	BC [11]	**BFS** [11]	**CC** [11]	**PR** [11]	**TC** [11]	**SSSP** [11]
irregData ElemSz	8 B + 4 B	4 B	4 B	4 B	4 B	4 B
Execution style	Push-Mostly	Push & Pull	Push-Mostly	Pull-Only	Push-Only	Push-Only
Use Frontier	Yes	Yes	No	No	No	Yes

TABLE IV: Graph kernels

	Web [11]	**Road** [11]	**Twitter** [11]	**Kron** [11]	**Urand** [11]	**Friendster** [53]
# Vertices (in M)	50.6	23.9	61.6	134.2	134.2	65.6
# Edges (in M)	1,949.4	58.3	1,468.4	2,111.6	2,147.4	3,612.1

TABLE V: Input Graphs

2006 [2] and SPEC CPU 2017 [3] benchmarks. All considered workloads experience at least 1 Miss per Kilo Instructions (MPKI) in the baseline system that Table III describes. Each workload is executed for 100 million instructions to warm up the memory hierarchy and the other microarchitectural structures, and it is executed for an additional set of 100 million instructions to obtain performance data. We run experiments evaluating the impact of using larger numbers of instructions (500 million warmup instructions, 1 billion simulation instructions), and observe identical trends with negligible differences in terms of IPC.

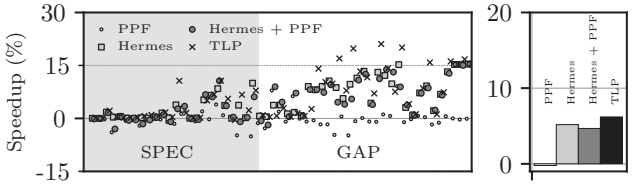
D. Multi-Core Evaluation

We generate multi-core workload mixes using the same methodology as previous work [13]. We consider either single-core GAP workloads or single-core SPEC workloads to create both homogeneous and heterogeneous multi-core workload mixes. To generate the homogeneous ones, we randomly select 50 single-core workloads and run four instances of each workload, one per core. For the heterogeneous mixes, we randomly select 50 combinations of four single-core workloads. In total, we consider 50 homogeneous and 50 heterogeneous four-core workloads. We do this process for both SPEC and GAP benchmark suites, meaning that our multi-core evaluation campaign is composed of 200 workloads. Finally, our multi-core experiments use the same number of warmup and simulation instructions as the single-core scenario.

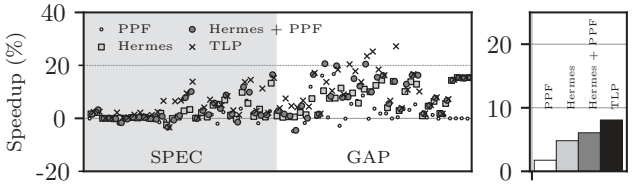
Our performance results concerning multi-core workloads report the weighted speedup normalized to the baseline. This metric is commonly used to evaluate multi-core workloads [30], [45], [50] since it avoids performance overestimation due to high-IPC threads. The metric is computed as follows: for each single-core workload, we compute its IPC in a multi-core scenario shared with the other co-running single-core workloads (IPC_{shared}), and its IPC running in isolation on the same system (IPC_{single}). We then compute the weighted IPC of the mix as the weighted sum of $IPC_{shared}/IPC_{single}$ for all the benchmarks in the mix, and we normalize this weighted IPC with the weighted IPC of the baseline design.

E. Alternative Techniques

Besides `TLP`, we consider the following techniques in our evaluation: (i) the Perceptron-based Prefetch Filtering (`PPF`) [14], a perceptron-based predictor that filters inaccurate prefetch requests, thus increasing the accuracy of the underlying

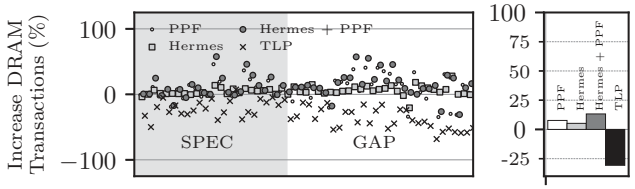


(a) IPCP

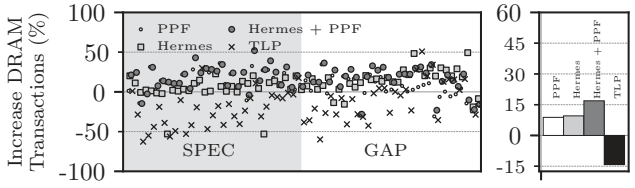


(b) Berti

Fig. 10: Performance evaluation in the single-core scenario.



(a) IPCP



(b) Berti

Fig. 11: Increase in DRAM transactions in the single-core scenario. Lower is better.

prefetcher. PPF is located at the L2C since it is built on top of the SPP prefetcher. When using PPF, we configure SPP as previous work indicates [14] to fully exploit the advantages of PPF. (ii) `Hermes` [13], the state-of-the-art off-chip predictor that removes long-lasting load requests from the critical path by issuing speculative requests to the DRAM controller. (iii) `Hermes+PPF`, a scheme that uses both `Hermes` as off-chip predictor and `PPF` as a prefetch filter.

VI. Evaluation

A. Single-Core Evaluation

This section evaluates `TLP` in the single-core context following the methodology that Section V presents. Figure 10 shows the performance gains provided by `PPF`, `Hermes`, `Hermes+PPF`, and `TLP` in the single-core context over a baseline described in Section V, which has no off-chip predictor neither L1D prefetch filter. Specifically, Figures 10a and 10b present performance results when using IPCP and Berti as L1D prefetchers, respectively. Both figures display speedup with respect to the baseline system in the y-axis. The x-axis shows the selected SPEC and GAP workloads. For each benchmark suite we sort workloads in increasing order of MPKI in the baseline system. This evaluation indicates that `TLP` significantly outperforms state-of-the-art approaches for off-chip prediction (`Hermes`), prefetch filtering (`PPF`), and a combination of them (`Hermes+PPF`). In the scenario considering IPCP as L1D prefetcher, `TLP` yields 6.2% geometric mean speedup with respect to the baseline system while `PPF`, `Hermes`, and `Hermes+PPF` bring -0.2%, 5.2%, and 4.7% geometric mean speedups, respectively. When considering the Berti prefetcher, `TLP` yields 8.1% geometric mean speedup as compared to 1.7%, 4.8%, and 6.1% for `PPF`, `Hermes`, and `Hermes+PPF`, respectively. `TLP` achieves larger performance gains for GAP than SPEC. Since GAP workloads are strongly memory bound, the reductions in terms of DRAM transactions that `TLP` achieves compared to `Hermes` particularly benefit GAP.

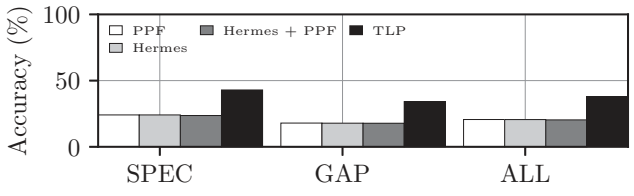
To identify the source of TLP performance improvements we quantify the impact of `PPF`, `Hermes`, `Hermes+PPF`, and `TLP` in terms of number of DRAM accesses. Figure 11 shows this evaluation. Specifically, Figure 11a presents results obtained using IPCP, while Figure 11b considers Berti. The y-axis displays the increase in DRAM transactions processed by the memory controller in the single-core context while the x-axis shows the SPEC and GAP workloads sorted in terms of MPKI. When using IPCP, `TLP` reduces DRAM transactions by an average of 30.7% over the baseline while `PPF`, `Hermes`, and `Hermes+PPF` increase average DRAM transactions by 7.7%, 5.2%, and 13.3%, respectively. When considering Berti, `TLP` reduces the number of DRAM transactions by an average of 14.2% over the baseline, while `PPF`, `Hermes`, and `Hermes+PPF` trigger of 8.8%, 9.6%, and 16.9% additional DRAM transactions, respectively.

To further explain the performance gain obtained by `TLP`, we evaluate the accuracy of the considered L1D prefetchers (IPCP and Berti) when `PPF`, `Hermes`, `Hermes+PPF`, and `TLP` operate in the system. Figure 12 presents the results. Specifically, Figures 12a and 12b present the accuracy of the IPCP and Berti prefetchers, respectively. The key takeaway of this comparison is that `TLP` increases the accuracy of the L1D prefetchers. Across all SPEC and GAP workloads, IPCP experiences an average accuracy of 20.6%, 20.6%, 20.3%, and 38.0% when `PPF`, `Hermes`, `Hermes+PPF`, and `TLP` operate in the system, respectively. Finally, we observe similar behavior for the Berti prefetcher; Figure 12b reveals that Berti experiences the highest accuracy with `TLP`.

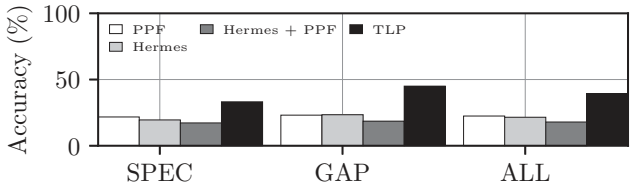
Data in Figures 11 and 12 indicate that `TLP` successfully reduces the number of DRAM transactions that state-of-the-art off-chip prediction and prefetch filtering approaches trigger.

B. Multi-Core Evaluation

This section evaluates the performance of `TLP` in the multi-core scenario following the methodology that Section V presents. In addition, this section indicates the contribution of each specific `TLP` component to final performance (Section

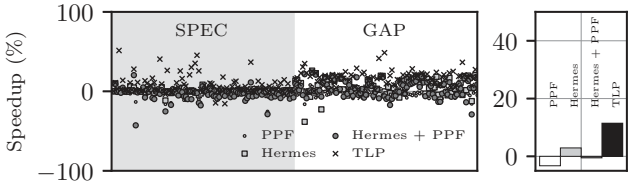


(a) IPCP

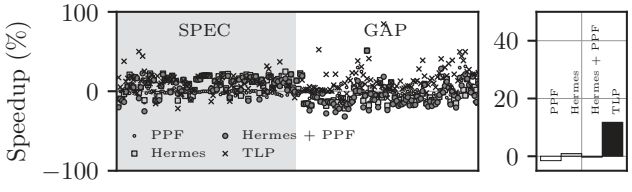


(b) Berti

Fig. 12: Accuracy of the L1D prefetchers.

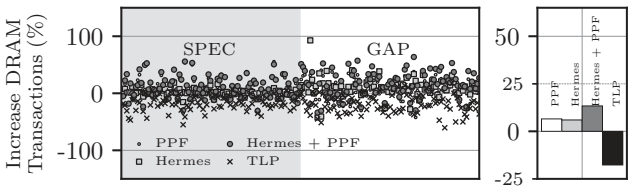


(a) IPCP

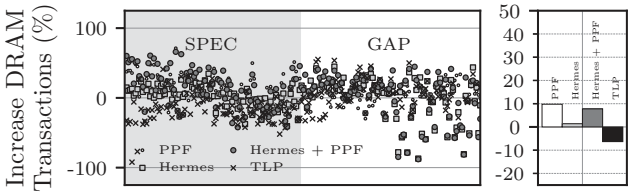


(b) Berti

Fig. 13: Performance improvement in the multi-core scenario.



(a) IPCP



(b) Berti

Fig. 14: Increase in DRAM transactions in the multi-core scenario. Lower is better.

VI-B1), and evaluates TLP considering different DRAM bandwidth scenarios (Section VI-B2).

Figure 13 shows the performance gains provided by PPF, Hermes, Hermes+PPF, and TLP in the multi-core context over the baseline system that Section V describes. Specifically, Figures 13a and 13b present performance results considering IPCP and Berti, respectively. Both figures display speedup over the baseline system in the y-axis. The x-axis shows the multi-core SPEC and GAP workloads sorted in increasing order in terms of MPKI. The sorting is done independently within each benchmark suite. Considering the IPCP prefetcher, TLP improves geometric mean performance by 11.5% as compared to -3.3%, 3.0%, and -0.5% for PPF, Hermes, and Hermes+PPF, respectively. When considering Berti as L1D prefetcher we observe similar trends. Specifically, TLP yields a 11.8% geometric mean speedup over the baseline as compared to -1.5%, 1.0%, and -0.3% for PPF, Hermes, and Hermes+PPF, respectively. The main takeaway of this experiment is that TLP provides significantly higher multi-core performance than all considered prior proposals.

To explain the source of TLP performance improvements in the multi-core context, we quantify the impact of PPF, Hermes, Hermes+PPF, and TLP on the number of DRAM accesses. Figure 14 shows the increase in terms of DRAM transactions over the baseline system that Section V describes. Figures 14a and Figure 14b show the impact on DRAM transactions when IPCP and Berti operate at L1D, respectively. Considering the IPCP prefetcher, TLP reduces the number of DRAM transactions by an average of 17.7% over the baseline while PPF, Hermes, and Hermes+PPF increase DRAM transactions by 6.5%, 6.0%, and 13.4%, respectively. We observe a similar behavior when Berti is used as L1D prefetcher: TLP reduces the average DRAM transactions by 6.3% while PPF, Hermes, and Hermes+PPF increase DRAM transactions by 9.8%, 1.4%, and 7.8%, respectively. The main takeaway is that TLP outperforms prior approaches for off-chip prediction and prefetch filtering in multi-core contexts since it significantly reduces the DRAM pressure.

1) Performance Contribution of Each TLP Component: This section evaluates the contribution of each specific TLP component to the final performance. To do so, we consider five different scenarios besides TLP: i) FLP, which consits of just the FLP predictor without the selective delay mechanism. ii) SLP, which consists of just the SLP predictor. iii) Two-Step Predictor (TSP), which consists of FLP without the selective delay mechanism, and SLP without the feature based on FLP output. TSP consumes FLP predictions before the completion of L1D accesses , as Hermes does. Therefore, the difference between TSP and Hermes is the use of SLP. iv) Delayed TSP, a technique similar to TSP with the exception that always delays the consumption of FLP predictions upon L1D misses. v) Selective TSP, an evolution of Delayed TSP that uses selective delay. Finally, we consider TLP. The difference between TLP and Selective TSP is that TLP uses a feature based on the output of FLP to drive the predictions of SLP. Figure 15 shows the performance of these

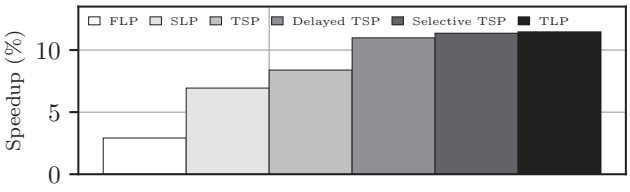
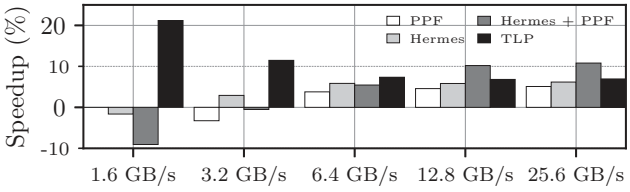
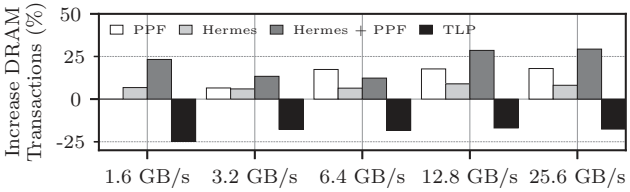


Fig. 15: Performance contribution of each TLP component.



(a) Performance improvement.



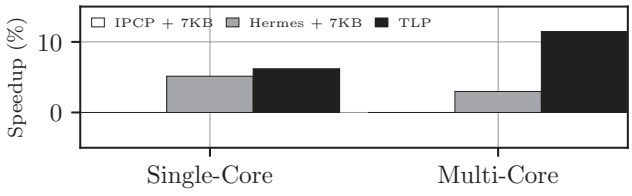
(b) Increase in DRAM transactions.

Fig. 16: Impact of DRAM bandwidth on geometric mean performance and average number of DRAM transactions in the multi-core context.

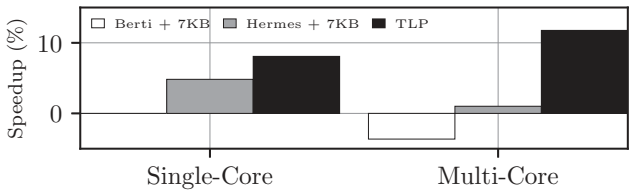
six approaches when IPCP operates at L1D. We observe that by incrementally adding parts of our design to form our final proposal, TLP, we can compound performance as FLP, SLP, TSP, Delayed TSP, and Selective TSP yield respectively 2.9%, 6.9%, 8.4%, 10.2%, and 11.4% geometric mean speedups over the baseline. Our final proposal, TLP provides a 11.5% speedup over the baseline, justifying our design choices. Although the difference between TLP and Selective TSP in terms of geometric mean speed-ups is rather small, it becomes larger for workloads with high correlation between off-chip load demand requests and off-chip L1D prefetch requests like bc.road. We observe a very similar behavior when we consider Berti as L1D prefetcher.

2) Sensitivity Analysis on DRAM Bandwidth: This section evaluates TLP, PPF, Hermes, and Hermes+PPF on scenarios with 1.6 GB/s per core up to 25.6 GB/s per core. Figures 16a and 16b show the geometric mean performance and the impact on DRAM accesses of TLP of these four approaches in the multi-core context, respectively.

Figure 16a indicates that TLP outperforms the other approaches under memory bandwidth regimes between 1.6 and 6.4 GB/s per core. The improvement achieved by TLP in the 1.6 GB/s scenario is 21.2%, while TLP obtains a 6.9% geometric mean speedup over the baseline when there are 25.6 GB/s per core available. Even in scenarios with unrealistically large memory bandwidth per core ratios (*e.g.*, 12.8 or 25.6 GB/s per core), TLP outperforms Hermes and PPF since it avoids cache pollution due to inaccurate prefetching. In these



(a) IPCP



(b) Berti

Fig. 17: Performance improvement of designs enhanced with TLP's storage budget.

scenarios, the unrealistic aboundance of memory bandwidth allows Hermes+TLP to deliver larger performance than TLP.

Figure 16b shows the impact in terms of DRAM transactions for all considered approaches on five memory bandwidth per core scenarios. TLP achieves a remarkable reduction in terms of DRAM transactions over the baseline in all scenarios. Specifically, TLP decreases DRAM transactions from 24.8% (1.6 GB/s core scenario) to 17.6% (25.6 GB/s per core scenario) as compared to the baseline.

C. Designs Enhanced with TLP's Storage Budget

In addition to the results provided in the previous sections, we also evaluate other designs leveraging 7KB of extra storage over IPCP, Berti, and Hermes. We compare them to TLP. Figure 17 shows the evaluation of these designs in both single-core and multi-core contexts. In the single-core context, adding 7KB of extra storage to IPCP and Berti does not leverage any performance benefits over the baseline. When Hermes is enhanced with 7KB of extra storage, it provides performance improvements close to the baseline Hermes, *i.e.*, 5.2% and 4.8% geometric mean speedup for IPCP and Berti, respectively. In comparison, TLP leverages 6.2% and 8.1% geometric mean speedup for IPCP and Berti. In the multi-core context, we observe a similar behavior where adding extra storage to the prefetchers does not leverage performance improvements. Finally, we observe that Hermes shows a similar behavior as its counterpart using no extra storage.

VII. Related Work

To the best of our knowledge, this is the first work to provide a cooperative solution for off-chip prediction and adaptive prefetch filtering using neural methods. Sections II, III-B, and VI describe, analyze and compare our proposal against Hermes [13], the state-of-the-art off-chip predictor, respectively. Sections II and VI compare our proposal against PPF [14], the state-of-the-art prefetch filter. This section focuses on other related work targeting memory hierarchy optimizations.

Hit/Miss Prediction. Jalili and Erez [26] proposed *LP*, a scheme that uses a flat-array to track the residency of cache lines in the cache hierarchy. This flat-array is stored in a reserved section of DRAM, and a small cache keeps recently used entries of the flat-array for future predictions. *LP* presents several challenges. First, it can have a high false-positive prediction rate. Second, the size of *LP* can grow very large, leading to significant latency and storage overheads. Third, *LP* does not address the large bandwidth consumption of cache prefetchers. In contrast, TLP requires only 4KB per core of storage overhead, and 56 additional bits per prefetch request, while producing accurate off-chip predictions for both load and prefetch requests. With its more streamlined approach, TLP offers a promising solution to the challenges presented by other prefetching methods, paving the way for improved performance and efficiency in modern computing systems.

Data Prefetching. Stream and strided cache prefetchers are unable to effectively prefetch for the indirect memory access patterns of graph-processing workloads [7], [10]. Yu et al. [55] propose a microarchitectural prefetcher that identifies and prefetches indirect memory access patterns without requiring any application nor software information. Ainsworth et al. [6] propose a prefetcher that leverages application-level information to capture indirect memory access patterns. Basak et al. [10] propose DROPLET, a physically decoupled prefetcher that takes into account the reuse distances when applying prefetching for different graph types. Although effective, these hardware prefetchers increase memory bandwidth consumption. In contrast, our proposal reduces the cost of hardware prefetching while keeping its advantages, as Section VI shows.

Cache Bypassing. Recent research has proposed several complex cache replacement and bypassing policies [25], [30], [45], [52], that have demonstrated significant performance gains in general-purpose computing applications. However, recent studies [27] show that these policies are ineffective when applied to workloads managing irregular and sparse structures like graphs due to the irregularity of the memory access patterns that these workloads display.

Memory Optimizations for Graph-Processing Applications. Recent work demonstrates the benefits of optimizing the memory hierarchy for graph applications. Ozdal et al. [38] use scratchpads to store vertex and edge data of graph-processing applications, while Gonzalez et al. [23] employ a large eDRAM scratchpad to accommodate larger volumes of graph data than the conventional SRAMs. Several prior works [5], [20], [36], [47], [56] reduce the latency cost of graph memory accesses by executing graph-processing operations close to DRAM, partially hiding the latency cost of the corresponding memory accesses. TLP complements these works since it improves the cache management of a wide range of applications.

Redesigning the Cache Hierarchy. The Distill Cache [41] approach reserves a section of the L2C to place the used words of a cache line when that line is elected for eviction. This design improves the use of cache storage capacity by evicting just the unused words of each cache line. In contrast, our proposal manages the pervasiveness of highly irregular access patterns and dynamically classifies memory accesses as either regular or irregular. By labeling some memory accesses as not cache-friendly, we avoid cache pollution and useless cache look-ups. The Victim Cache [31] proposal is a small fully-associative cache, found on the refill path of the LLC. It contains eviction victims of the cache to which it is attached and tries to decrease conflict misses. On LLC misses, both the LLC and the Victim Cache are looked-up; if the requested cache block is found in the Victim Cache, the LLC victim and the Victim Cache entry are swapped, thus lowering the miss latency. On a Victim Cache miss, the block is fetched from DRAM and the LLC victim is inserted in the Victim Cache. While the Victim Cache has been proven effective at improving the performance of SPEC workload [2], [3], it relies heavily on spatial locality as it inserts caches victims. Our proposal does not rely on locality assumptions and shortcuts the cache hierarchy when it is predicted to be inefficient.

VIII. Conclusions

This work introduces *Two Level Perceptron (TLP)* predictor, a neural approach that leverages two perceptron predictors to apply off-chip prediction to both demand and prefetch requests. This technique prevents memory bandwidth waste due to inaccurate prefetch requests or wrong off-chip prediction, and avoids cache pollution due to prefetching. We evaluate TLP against several previous approaches (PPF [14] and Hermes [13]) considering 55 single-core workloads, 200 multi-core workloads, and two state-of-the-art L1D prefetchers, IPCP [39] and Berti [37]. TLP achieves a 11.5% geometric mean speedup when deployed on a multi-core system using the IPCP L1D prefetcher, while the best previous approach, Hermes, delivers 3.0%. When considering Berti, Hermes delivers 0.9% geometric mean speedup while TLP obtains 11.8% improvement. Our evaluation also demonstrates that TLP significantly reduces the overhead in terms of DRAM bandwidth transactions that previous approaches incur in all considered scenarios.

Acknowledgments

The authors are grateful to the anonymous MICRO 2023 and HPCA 2024 reviewers for their valuable comments and constructive feedback that significantly improved the quality of the paper. This work has been partially supported by the European HiPEAC Network of Excellence, by the Spanish Ministry of Science and Innovation MCIN/AEI/10.13039/501100011033 (contracts PID2019-107255GB-C21 and PID2019-105660RB-C22) and by the Generalitat de Catalunya (contract 2021-SGR-00763). This work is supported by the National Science Foundation through grant CCF-1912617 and generous gifts from Intel. Marc Casas has been partially supported by the Grant RYC-2017-23269 funded by MCIN/AEI/10.13039/501100011033 and by ESF Investing in your future. Els autors agraeixen el suport del Departament de Recerca i Universitats de la Generalitat de Catalunya al Grup de Recerca ”Performance understanding, analysis, and simulation/emulation of novel architectures” (Codi: 2021 SGR 00865).

APPENDIX

A. Abstract

Our artifact provides i) the implementation of TLP, ii) the simulation infrastructure, iii) the set of workloads, iv) scripts for launching the experiments, and v) Python scripts bundled in Jupyter notebooks to exploit the simulation results and reproduce some of the key figures of this paper.

B. Artifact check-list (meta-information)

- **Program:** Memory traces of SPEC 2006 [2], SPEC 2017 [3], and GAP [11] workloads.
- **Compilation:** GNU GCC and CMake.
- **Metrics:** Performance improvements, reduction in DRAM transactions, statistics on inaccurate off-chip predictions, L1D useful & useless prefetches, and L1D prefetchers' accuracy.
- **Output:** We provide scripts that generate all single-core figures (Figures 1, 2, 4, 5, 6, 10, 11, 12).
- **Experiments:** We provide scripts that submit the required jobs. The only requirement is a SLURM manager.
- **How much disk space required (approximately)?:** 140GB.
- **How much time is needed to prepare workflow (approximately)?:** About 1 hour.
- **How much time is needed to complete experiments (approximately)?:** About 12 hours.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:**
- **Workflow framework used?:** SLURM for job management.
- **Archived (provide DOI)?:** The code is available at https://doi.org/10.5281/zenodo.10100304. The trace set is available in 3 volumes:
  - Volume 1: https://doi.org/10.5281/zenodo.10083542
  - Volume 2: https://doi.org/10.5281/zenodo.10088347
  - Volume 3: https://doi.org/10.5281/zenodo.10088525

C. Description

1) How to access: Our artifact is available at https://doi.org/10.5281/zenodo.10100304.

2) Hardware dependencies: Any hardware capable of compiling and running ChampSim [4].

3) Software dependencies: Our artifact depends on the following tools: CMake, Jupyter, Python 3.8.10, matplotlib, and SLURM.

4) Data sets: Memory traces of SPEC 2006 [2], SPEC 2017 [3], and GAP [11] workloads.

D. Installation

First, Download the artifact from our GitHub repository using the appropriate `git clone` command. Second, download the trace set from the following Zenodo records:

- Volume 1: https://doi.org/10.5281/zenodo.10083542
- Volume 2: https://doi.org/10.5281/zenodo.10088347
- Volume 3: https://doi.org/10.5281/zenodo.10088525

E. Experiment workflow

To reproduce all the single-core figures of this work, take the following steps:

- `cd TLP-HPCA30-artifact`
- Move the three volumes of the traces artifact to the root of the code's artifact.
- Extract the traces using `tar -xMf TLP-HPCA30-artifact-traces.VOLUME1.tar`. This command is interactive and will request you to provide the name of the next archive's volume as follows `TLP-HPCA30-artifact-traces.VOLUME2.tar`, etc. A new directory named `traces/` will be present in the artifact directory.
- set paths and username in `scripts/-run_single_core.sh` (lines 4, 7, 8, 10, and 11), `scripts/run_single_core_legacy.sh` (lines 4, 7, 8, 10, and 11), `scripts/run_single_core.job` (lines 6, 9, 10, and 12).
- Execute `./scripts/compile_single_core.sh` to compile the binaries.
- Execute `scripts/run_single_core.sh` and `scripts/run_single_core_legacy.sh` to launch all single-core simulations.

Running all the jobs takes around 12 hours, depending on the cluster and the number of jobs that can be launched in parallel.

F. Evaluation and expected results

When all jobs are finished, generate the single-core figures using the Jupyter notebooks provided in the `notebooks` directory. We recommend using the Jupyer extension in the VS Code editor, as it is how the workflow was originally designed.

The single-core figures will be available in the `plots/` directory.

G. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

REFERENCES

[1] Cascade lake - microarchitectures - intel - WikiChip. [Online]. Available: https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake#Memory_Hierarchy

[2] SPEC CPU® 2006. [Online]. Available: https://www.spec.org/cpu2006/

[3] SPEC CPU® 2017. [Online]. Available: https://www.spec.org/cpu2017/

[4] "ChampSim," https://crc2.ece.tamu.edu/, 2021, [Online].

[5] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 105–117.

[6] S. Ainsworth and T. M. Jones, "Graph Prefetching Using Data Structure Knowledge," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1–11. [Online]. Available: https://doi.org/10.1145/2925426.2926254

[7] V. Balaji, N. Crago, A. Jaleel, and B. Lucia, "P-OPT: Practical Optimal Cache Replacement for Graph Analytics," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 668–681.

[8] V. Balaji and B. Lucia, "When is graph reordering an optimization? studying the effect of lightweight graph reordering across applications and input graphs," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 203–214.

[9] V. Balaji and B. Lucia, "Combining data duplication and graph reordering to accelerate parallel graph processing," in *Proceedings of the 28th International symposium on high-performance parallel and distributed computing*, 2019, pp. 133–144.

[10] A. Basak, S. Li, X. Hu, S. M. Oh, X. Xie, L. Zhao, X. Jiang, and Y. Xie, "Analysis and optimization of the memory hierarchy for graph processing workloads," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 373–386.

[11] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

[12] S. Beamer, K. Asanović, and D. Patterson, "Reducing pagerank communication via propagation blocking," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 820–831.

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

[14] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 1–13.

[15] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 72–81. [Online]. Available: https://doi.org/10.1145/1454115.1454128

[16] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[17] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[18] T. D. Bui, S. Ravi, and V. Ramavajjala, "Neural graph learning: Training neural networks using graphs," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, ser. WSDM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 64–71. [Online]. Available: https://doi.org/10.1145/3159652.3159731

[19] O. Evelien and R. Rousseau, "Social network analysis: A powerful strategy, also for the information sciences," *Journal of Information Science*, vol. 28, no. 6, p. 441–453, December 2002.

[20] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, 2015, pp. 113–124.

[21] E. Garza, S. Mirbagher-Ajorpaz, T. A. Khan, and D. A. Jiménez, "Bit-level perceptron prediction for indirect branches," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 27–38. [Online]. Available: https://doi.org/10.1145/3307650.3322217

[22] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," 2022.

[23] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph Processing in a Distributed Dataflow Framework," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14. USA: USENIX Association, 2014, p. 599?613.

[24] C. T. Have and L. J. Jensen, "Are graph databases ready for bioinformatics?" *Bioinformatics*, vol. 29, no. 24, pp. 3107–3108, 10 2013. [Online]. Available: https://doi.org/10.1093/bioinformatics/btt549

[25] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 60–71. [Online]. Available: https://doi.org/10.1145/1815961.1815971

[26] M. Jalili and M. Erez, "Reducing load latency with cache level prediction," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, April 2022, pp. 648–661.

[27] A. V. Jamet, L. Alvarez, D. A. Jiménez, and M. Casas, "Characterizing the impact of last-level cache replacement policies on big-data workloads," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2020, pp. 134–144. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/343622/IISWC20-paper.pdf?sequence=1

[28] D. Jimenez, "Piecewise linear branch prediction," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 382–393.

[29] D. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, ser. HPCA '01, 2001, pp. 197–206.

[30] D. A. Jiménez and E. Teran, "Multiperspective reuse prediction," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO-50 '17, IEEE. New York, NY, USA: Association for Computing Machinery, 2017, pp. 436–448. [Online]. Available: https://doi.org/10.1145/3123939.3123942

[31] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *ACM SIGARCH Computer Architecture News*, vol. 18, no. 2SI, pp. 364–373, 1990.

[32] S. M. Khan, Y. Tian, and D. A. Jiménez, "Sampling dead block prediction for last-level caches," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 175–186.

[33] J. Kim, S. H. Pugsley, P. V. Gratz, A. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[34] G. Memik, G. Reinman, and W. Mangione-Smith, "Just say no: benefits of early cache miss determination," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, 2003, pp. 307–316.

[35] U. Meyer and P. Sanders, "δ-stepping: a parallelizable shortest path algorithm," *Journal of Algorithms*, vol. 49, no. 1, pp. 114–152, 2003.

[36] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 457–468.

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

[38] M. M. Ozdal, S. Yesil, T. Kim, A. Ayupov, J. Greth, S. Burns, and O. Ozturk, "Energy Efficient Architecture for Graph Analytics Accelerators," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 166–177. [Online]. Available: https://doi.org/10.1109/ISCA.2016.24

[39] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 118–131.

[40] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using simpoint for accurate and efficient simulation," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1, pp. 318–319, 2003.

[41] M. K. Qureshi, M. A. Suleman, and Y. N. Patt, "Line Distillation: Increasing Cache Capacity by Filtering Unused Words in Cache Lines," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 250–259.

[42] A. Sembrant, E. Hagersten, and D. Black-Schaffer, "The direct-to-data (d2d) cache: Navigating the cache hierarchy with a single lookup," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, p. 133–144, jun 2014. [Online]. Available: https://doi.org/10.1145/2678373.2665694

[43] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The evicted-address filter: A unified mechanism to address both cache pollution and thrashing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 355–366. [Online]. Available: https://doi.org/10.1145/2370816.2370868

[44] 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," vol. 11, no. 4, jan 2015. [Online]. Available: https://doi.org/10.1145/2677956

[45] Z. Shi, X. Huang, A. Jain, and C. Lin, "Applying deep learning to the cache replacement problem," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 413–425.

[46] Y. Shiloach and U. Vishkin, "An o (log n) parallel connectivity algorithm," Computer Science Department, Technion, Tech. Rep., 1980.

[47] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating Graph Processing Using ReRAM," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 531–543.

[48] E. Teran, Z. Wang, and D. A. Jiménez, "Perceptron learning for reuse prediction," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[49] G. Vavouliotis, L. Alvarez, V. Karakostas, K. Nikas, N. Koziris, D. A. Jiménez, and M. Casas, "Exploiting page table locality for agile tlb prefetching," in *Proceedings of the 48th International Symposium on Computer Architecture*, 2021.

[50] G. Vavouliotis, G. Chacon, L. Alvarez, P. V. Gratz, D. A. Jiménez, and M. Casas, "Page size aware cache prefetching," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 956–974.

[51] H. Wei, J. X. Yu, C. Lu, and X. Lin, "Speedup graph processing by graph ordering," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1813?1828. [Online]. Available: https://doi.org/10.1145/2882903.2915220

[52] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr, and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 430–441.

[53] J. Yang and J. Leskovec, "Community-affiliation graph model for overlapping network community detection," in *2012 IEEE 12th international conference on data mining*. IEEE, 2012, pp. 1170–1175.

[54] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan, "Speculation techniques for improving load related instruction scheduling," *SIGARCH Comput. Archit. News*, vol. 27, no. 2, p. 42–53, may 1999. [Online]. Available: https://doi.org/10.1145/307338.300983

[55] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "IMP: Indirect Memory Prefetcher," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 178–190. [Online]. Available: https://doi.org/10.1145/2830772.2830807

[56] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 544–557.

[57] Y. Zhang, V. Kiriansky, C. Mendis, S. Amarasinghe, and M. Zaharia, "Making caches work for graph analytics," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 293–302.

[58] X. Zhuang and H.-H. Lee, "A hardware-based cache pollution filtering mechanism for aggressive prefetches," in *2003 International Conference on Parallel Processing, 2003. Proceedings.*, 2003, pp. 286–293.