Crossing the Boundary: Virtual-Address Based Inter-Page Prefetching for Lower Level Caches

Ho Je Lee
Yonsei University
Seoul, South Korea
hoje.lee@yonsei.ac.kr

Won Woo Ro
Yonsei University
Seoul, South Korea
wro@yonsei.ac.kr

Abstract

Physical-address (PA) based L2 prefetchers are designed to capture long-range access patterns, yet they are fundamentally unable to issue page-cross prefetches because of security issues. As a result, modern processors conservatively drop any predicted access that crosses a 4 KB boundary, truncating otherwise continuous access streams and reintroducing compulsory misses at every page transition. This architectural restriction leaves a critical gap in the hierarchy: the L1 prefetcher observes virtual-address continuity but lacks the lookahead distance to exploit it, while lower-level prefetchers possess the lookahead but cannot cross page boundaries.

We propose VIP (Virtual Inter-page Prefetcher), a lightweight mechanism that safely enables page-cross prefetching on behalf of PA-based prefetchers. VIP monitors the L1D miss stream in the virtual-address domain, learns inter-page stride patterns using a compact IP-correlated predictor, and issues physical prefetches only after a TLB lookup verifies correctness and permissions. VIP introduces only a very small storage overhead and requires no modification to existing L1/L2 prefetchers. Across comprehensive evaluations, VIP consistently improves performance, delivering a 4.77% geometric-mean speedup over the DPC-4 state-of-the-art baseline (Berti + Pythia) across 132 traces.

1 Background & Motivation

This section reviews how modern multi-level prefetchers operate, why lower-level prefetchers cannot cross page boundaries, and how this limitation restricts their ability to capture long-range access patterns. We then quantify the performance impact of these boundary-induced stalls and show the structural blind spot for page cross prefetching in current multi-level prefetchers.

1.1 Multi-Level Prefetchers

Data prefetching is essential for hiding the latency of memory accesses. Hardware prefetchers track access patterns and proactively insert data into the cache hierarchy, enabling subsequent requests to hit without delay. Such mechanisms can operate at multiple levels, from the private L1 data cache to the L2 and shared LLC.

Roles of Multi-Level Prefetchers: Prefetchers placed at the private L1 data cache aim to capture short-distance spatial or temporal locality that closely reflects the program's original access stream. Their proximity to the core enables highly timely prefetches, often bringing lines into the L1 before the corresponding demand arrives. However, L1 prefetchers operate under tight constraints: limited lookahead distance, small storage budgets, and a strong need to avoid cache pollution in the latency-critical L1. Consequently, L1 cache prefetchers adopt conservative, low-latency pattern detection and provide only shallow lookahead.

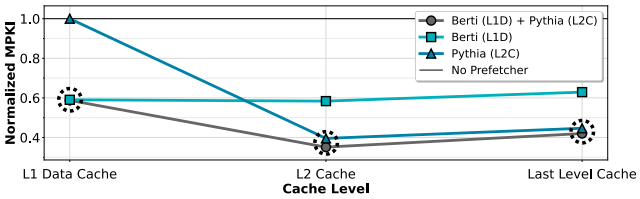

Figure 1: Normalized demand MPKI at each cache level under different prefetcher configurations, normalized to a no-prefetcher baseline.

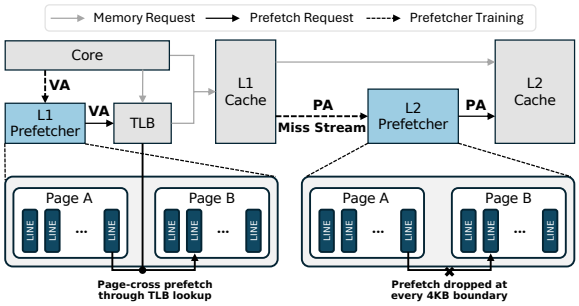

Figure 2: L1 prefetchers (VA-based) can cross page boundaries, while L2 prefetchers (PA-based) are restricted from crossing pages.

In contrast, prefetchers at lower levels (L2 or LLC) target access patterns that span longer distances. Because these prefetchers see a miss stream rather than the full demand stream, they operate on sparser patterns but can exploit much larger state and lookahead due to their larger capacity. L2/LLC prefetchers are therefore responsible for recovering long-range locality, reducing miss penalties that would otherwise propagate to main memory. Their design often prioritizes aggressive coverage and higher bandwidth utilization, as pollution risks are lower than in L1.

Effect of Multi-Level Prefetchers: The distinct roles of L1 and lower-level prefetchers are also reflected quantitatively in their impact on demand miss behavior. To illustrate this, we conduct a simple experiment measuring the demand MPKI (misses per kilo-instruction) at each cache level under different prefetcher configurations, normalized to a no-prefetcher baseline. We compare an L1-only configuration using Berti [11], an L2-only configuration using Pythia [1], and a multi-level configuration that combines both. As shown in Figure. 1, the combined multi-level design achieves an L1 MPKI reduction comparable to that of the L1-only prefetcher, while simultaneously achieving L2 and LLC MPKI reductions comparable to those of the L2-only prefetcher. This observation indicates that

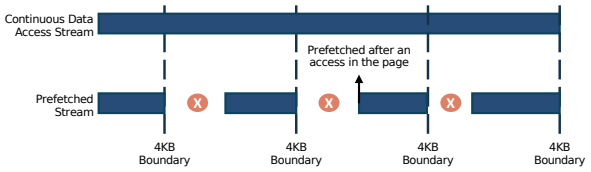

Figure 3: Illustration of boundary-induced prefetch delays under physical-address–based prefetching.

each prefetcher remains effective at the cache level it targets, even when both are enabled together. In particular, the L2 prefetcher contributes most of the MPKI reduction at the L2 and LLC by capturing long-range access patterns that are inherently difficult for L1 prefetchers to exploit due to their limited lookahead and tight capacity constraints.

Insight I. Each prefetcher is most effective at its intended cache level, with L2 prefetchers capturing long-range access patterns beyond the reach of L1 prefetchers.

Limitation of Physical Address Based Prefetchers: Despite the effectiveness of lower-level prefetchers at capturing long-range access patterns, these prefetchers face an inherent architectural limitation rooted in their reliance on physical addresses. As illustrated in Figure. 2, L1 prefetchers operate in the virtual-address domain and can consult the TLB before issuing a prefetch, allowing them to safely cross page boundaries. In contrast, L2 and LLC prefetchers operate after address translation and observe only physical addresses, losing visibility into the program's virtual address continuity. As a result, they cannot determine how consecutive virtual pages are mapped to physical frames. Physical fragmentation can cause adjacent virtual pages to reside in non-contiguous or unrelated physical locations, making page-cross prediction in the physical domain unreliable. Moreover, without consulting the TLB, lower-level prefetchers cannot verify whether a predicted physical access corresponds to a valid page mapping with appropriate permissions. Allowing such speculative cross-page prefetches could therefore lead to unsafe accesses and potential side-channel vulnerabilities [3, 4, 13]. As a result, modern processors conservatively forbid physical-address–based prefetchers from issuing any request that is predicted to cross a 4 KB boundary, even when the access pattern remains highly predictable. This restriction fundamentally limits the long-range coverage achievable by deeper-level prefetchers.

Insight II. Page boundaries force physical address based prefetchers to drop predictable accesses, sharply limiting their long-range coverage.

1.2 Impact of Missing Page Cross Prefetching

We now examine the quantitative impact of missing page-cross prefetching. Even when long-range access patterns are predictable, lower-level prefetchers are forced to stop at every 4 KB boundary, introducing avoidable compulsory misses that disrupt otherwise smooth access streams. These boundary-induced stalls accumulate across large, multi-page data structures, effectively capping the achievable benefits of long-range prefetching.

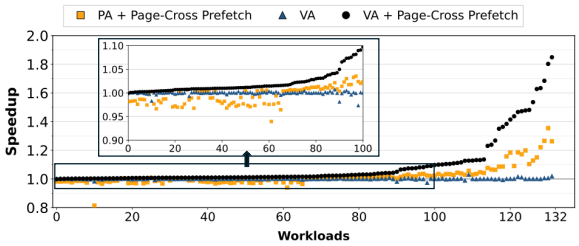

Figure 4: Speedup of three L2C prefetcher variants—PA+cross, VA-only, and VA+cross—each applied on top of Berti (L1D) + Pythia (L2C) with large page allocation enabled, evaluated over 132 workloads.

Missed Benefits: To illustrate the performance impact of missing page-cross prefetching, we consider a simple yet representative example, shown in Figure. 3. Suppose an application generates a continuous and highly predictable data access stream that naturally spans multiple pages. While the access pattern itself exhibits strong regularity, physical-address–based prefetchers are constrained by page boundaries and cannot issue prefetches beyond a 4 KB boundary. Consequently, prefetching within a new page can only begin after an access to that page is observed. This requirement introduces a delay at every page transition, effectively truncating an otherwise continuous access stream. As a result, long-range access patterns are repeatedly truncated at page boundaries, preventing lower-level prefetchers from sustaining timely and continuous lookahead across pages.

Performance Headroom: Although L2 prefetchers cannot, in real hardware, operate on virtual addresses or safely issue page-cross prefetches with physical addresses, we evaluate such hypothetical configurations to quantify the performance headroom left unexploited by current designs. Figure. 4 shows this headroom using the baseline of Berti (L1D) [11] + Pythia (L2C) [1] with large page allocation [2] enabled. The figure compares three L2C prefetcher (Pythia) variants: a physical-address prefetcher with page-crossing enabled, and a virtual-address prefetcher with page-crossing disabled and enabled. The PA+cross configuration provides limited benefit and sometimes degrades performance, while the VA-only configuration avoids these inaccuracies but still incurs a compulsory miss at every page boundary. In contrast, the VA+cross variant consistently improves performance across nearly all 132 workloads, with steady gains for most applications and substantial improvements for those exhibiting long, virtually contiguous access patterns.

These results reveal two key insights. First, page-cross prefetching is broadly beneficial: eliminating boundary-induced compulsory misses helps most workloads, and those that traverse multi-page, virtually contiguous data structures can achieve particularly large speedups. Second, large pages reduce—but do not eliminate—physical fragmentation. Misalignments within or across large-page regions disrupt the physical stride pattern, so PA-based page-cross predictions often point to the wrong physical page and degrade accuracy. It is also worth noting that many production servers and data-center systems rely primarily on 4 KB pages due to practical limitations of large pages such as the need for contiguous physical memory, defragmentation overheads, and increased tail latency—as highlighted by prior studies [7, 8, 12]. Virtual-address

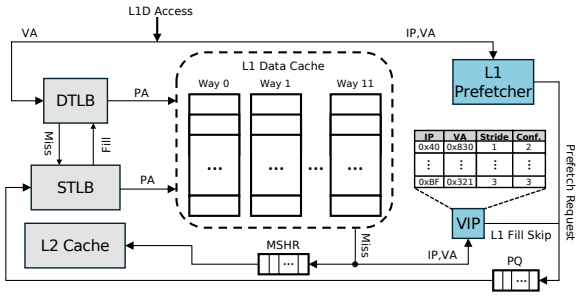

Figure 5: Virtual Inter-page Prefetcher Design Overview

prefetching, on the other hand, preserves the program's logical continuity and enables accurate page-cross prediction, explaining the consistent advantage of the VA+cross configuration.

Insight III. Virtual address prefetching predicts cross page locality more accurately than physical address prefetching, showing huge performance headroom when page-bound stalls are removed.

1.3 Structural Blind Spot Between L1 and L2 Prefetchers

Modern L1 and L2 prefetchers are effective within their respective roles, yet fail to complement each other across page boundaries due to a fundamental mismatch in capability. L1 prefetchers operate in the virtual-address domain and preserve page continuity, but are constrained to short-range lookahead by tight capacity and timeliness requirements. In contrast, L2 prefetchers can sustain long-range lookahead, but operate exclusively on physical addresses and therefore lose visibility into virtual page continuity. As a result, no prefetcher in the hierarchy simultaneously possesses both virtual-address visibility and long-range lookahead, leaving page-cross locality fundamentally unexploited in existing designs.

Insight IV. L1 prefetchers see virtual page continuity but lack lookahead, while L2 prefetchers have lookahead but cannot observe virtual continuity—creating a structural blind spot for page-cross prefetching.

2 Virtual Inter-page Prefetcher

To overcome the blind spot, we propose VIP (Virtual Inter-page Prefetcher), a light-weight extension to the L1 prefetcher that enables inter-page prefetching on behalf of L2. Unlike conventional L1 prefetchers, which serve only L1D, VIP's role is to learn the virtual-address stride behavior of the L2 cache, and provide timely page-cross prefetches that lower-level prefetchers cannot generate.

2.1 Overview and Integration into the L1 Pipeline

Figure 5 presents the overall design of VIP. Conventional L2 prefetchers operate exclusively on the L1 miss stream—the filtered sequence of requests that escape the L1D. VIP is designed to observe the exact same miss stream, but in the virtual-address (VA) domain instead of the physical domain. Modern processors store both the virtual address (VA) and instruction pointer (IP) inside each MSHR entry [9] to support replay, page-fault handling, and coherence interactions. VIP leverages this existing metadata to tap into the L1 miss stream as it enters the MSHR, before physical address translation discards virtual contiguity. Because VAs preserve the program's logical page-to-page layout, this VA-domain view reveals inter-page spatial patterns that disappear once addresses are translated into fragmented physical frames.

Table 1: Storage overhead of VIP

Structure	Description	Size
VIP Stride Table	64 sets × 1 way = 64 entries Each entry stores (bit-level): • Last virtual line number (42 bits) • Last stride (8 bits) • Confidence counter (4 bits) • IP tag (6 bits) • Valid bit (1 bit) Total per entry: 61 bits Total for 64 entries: 3,904 bits (≈488 B)	~0.48 KB
Control Registers	• Prefetch-degree thresholds • Confidence thresholds	< 16 B
Total		**~0.49 KB**

Upon each L1 miss, the VA and IP are forwarded to VIP, enabling it to learn the same long-range access patterns that an L2 prefetcher would attempt to learn—except without losing virtual continuity. Once VIP predicts the next virtual address, it performs a TLB lookup to translate the prediction into a physical address, ensuring permission checks and correctness while avoiding the security issues inherent in physical page-cross speculation. Although VIP is positioned near the L1 prefetching stage, its role is not to improve L1 behavior. VIP is designed specifically to augment the L2 prefetcher by supplying the missing page-cross predictions that physical-address-based L2 designs cannot generate. Filling VIP predictions into the L1D would not improve timeliness for these L2-bound accesses and would instead introduce unnecessary pollution in the latency-critical L1 cache. Accordingly, VIP bypasses the L1D entirely and injects its prefetches directly toward the L2 side, aligning its behavior with the level of the hierarchy where page-cross prefetching is both meaningful and effective.

VIP is designed as a lightweight extension unit that operates alongside the existing L1 and L2 prefetchers rather than replacing or modifying them. Both the L1 and L2 prefetching mechanisms remain entirely unchanged. At the same time, VIP's predictions naturally complement the L2 prefetcher to support inter-page prefetching, while the conventional L2 prefetcher continues to handle all intra-page patterns in the physical domain. This clean division allows VIP to supply long-range, page-cross lookahead without duplicating or disrupting the L2 prefetcher's behavior, providing additive benefit while preserving the simplicity and stability of the baseline multi-level prefetching hierarchy.

2.2 Learning Algorithm and Confidence-Guided Prefetching

VIP employs an IP-correlated stride learning model that operates in the virtual-address domain. This lightweight design is intentionally chosen for two reasons. First, stride-based prediction is particularly

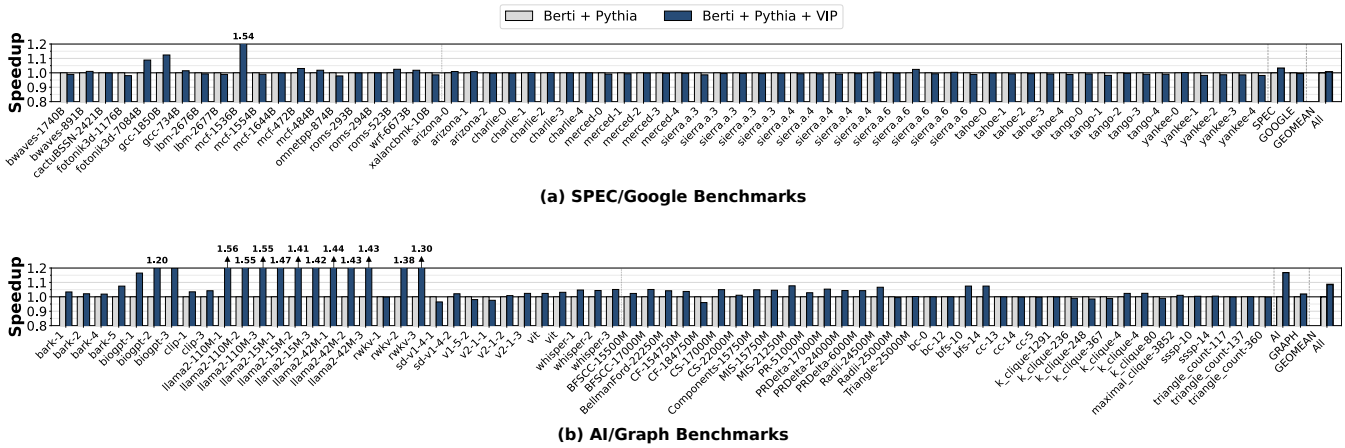

Figure 6: Speedup of VIP combined with Berti (L1D) and Pythia (L2C), evaluated on (a) 43 Google + 20 SPEC CPU2017 traces and (b) 32 AI + 37 Graph traces, normalized to Berti (L1D).

effective at detecting page-cross behavior, as many long-range sequential streams maintain a consistent virtual stride even when they span multiple pages. Second, VIP is implemented alongside the L1 prefetcher, where storage budgets are extremely tight; therefore, its predictor must be simple enough. More sophisticated correlation- or history-based models would require larger state and deeper history tables, which are difficult to place near the L1 pipeline and are less robust to the noise present in the VA miss stream.

For each instruction pointer, VIP keeps only the last virtual address and its associated stride, increasing confidence when the same stride reappears and resetting confidence when the pattern diverges. This confidence value determines the prefetch degree, enabling aggressive lookahead only for stable patterns while remaining conservative for irregular streams. Through this confidence-guided, low-overhead mechanism, VIP provides timely page-cross prefetches that lower-level prefetchers fundamentally cannot issue, restoring long-range locality without requiring any modification to the L1 prefetcher and the L2 prefetcher itself.

2.3 Storage Overhead

Table 1 summarizes the hardware cost of VIP. The primary structure is the VIP stride table, organized as 64 entries with only 61 bits per entry, storing the last virtual line number, last stride, a small confidence counter, an IP tag, and a valid bit. This compact design—totaling roughly 0.48 KB—is sufficient because VIP operates on the L1 miss stream, which is already a filtered, spatially coherent view of the program's access behavior. As a result, VIP does not require large associative structures or deep per-entry metadata; the miss stream's natural sparsity and locality allow a small predictor to capture the long-range, inter-page stride patterns that lower-level PA-based prefetchers miss. Empirically, we find that expanding entry size or adding additional metadata offers no meaningful performance improvement, since the L1 miss stream does not justify a more complex or larger design. Beyond the stride table, VIP uses fewer than 16 B of control registers (prefetch-degree thresholds and confidence limits), and requires no modifications to the L2 cache or the physical-address-based prefetching pipeline. Overall, VIP adds only ~0.49 KB of hardware state per core—a negligible footprint compared to modern cache and prefetching structures—while enabling accurate page-cross lookahead that L2/LLC prefetchers fundamentally cannot provide.

In the DPC-4 submission, VIP is evaluated alongside existing L1D and L2C prefetchers. The L1D prefetcher (Berti) uses approximately 29.5 KB, and the L2C prefetcher (Pythia) uses approximately 25.5 KB. Adding VIP's ~0.49 KB state at L1 keeps the total storage well within the DPC-4 limits of 32 KB for L1D and 128 KB for L2 prefetchers. Thus, the proposed design fully complies with the DPC-4 storage budget requirements.

3 Evaluation

Evaluation Summary: Figure 6 summarizes the per-benchmark performance impact of VIP across all 132 workloads. Overall, VIP consistently improves performance over Berti (L1D) + Pythia (L2C), achieving a **4.77%** geometric-mean speedup. AI workloads benefit the most (16.83% geomean), as they frequently traverse long, virtually contiguous regions where each page transition would otherwise incur an unavoidable L2/LLC miss. SPEC CPU2017 and Graph workloads show more moderate gains (3.30% and 1.96% geomean), reflecting their mix of spatially regular and irregular phases. In contrast, Google production traces exhibit a small performance regression (-0.46% geomean). These workloads are typically server-style applications characterized by control-heavy execution and limited long, contiguous data streams, which reduces their sensitivity to data-side miss reduction [6]. As a result, improving data-side locality alone provides modest benefit and can slightly increase contention for shared cache capacity, occasionally displacing instruction or other useful cache lines. In addition, instruction-side stalls remain a significant contributor to performance in such workloads. In real systems, widely deployed mechanisms such as a decoupled front-end or fetch-directed instruction prefetching (FDIP) [5, 10] would mitigate these instruction-side effects, making the data-side benefits of VIP more visible. Because such mechanisms are not modeled in our evaluation framework, the observed regression reflects

workload characteristics and resource interactions rather than a fundamental limitation of page-cross data prefetching.

Overall, VIP delivers consistent improvements across data-intensive workloads without pathological slowdowns, demonstrating that virtual-address–guided inter-page prefetching effectively addresses a critical gap in existing lower-level data prefetchers.

4 Discussion and Future Work

Architectural Integration and Optimization Opportunities: There remain several opportunities to further improve VIP beyond the DPC-4 implementation constraints. In our current prototype, VIP issues its prefetch requests through the L1 prefetch queue and relies on L1-side structures due to the competition rules. In practical hardware, VIP could instead translate its predicted virtual address via a TLB lookup and directly enqueue the resulting physical request into the L2 prefetch queue. This would avoid using L1 MSHRs, ports, and bandwidth, reducing contention in the latency-critical L1 datapath and potentially improving performance further.

Algorithmic Design Choices: A key design decision in VIP is the use of a lightweight IP-stride predictor. We adopt this simple mechanism because IP-stride prefetching already captures the dominant form of inter-page locality—contiguous or near-contiguous miss streams across virtual pages—while avoiding the higher storage, latency, and integration complexity of more sophisticated predictors. Rather than increasing predictor complexity, we focus on preserving simplicity and low overhead. Although feedback-based suppression could further improve accuracy, we show that a simple IP-stride design alone is sufficient to achieve meaningful performance gains, leaving such enhancements as future work.

Training Lower-Level Prefetchers at L1D: An interesting design question is whether it is beneficial to train lower-level prefetchers, such as L2 prefetchers, directly at the L1D miss stream—for example, training Pythia on L1 misses while still inserting prefetches into the L2. While this approach could, in principle, expose richer access patterns, it introduces several practical challenges.

First, training a full-fledged L2 prefetcher at the L1 significantly increases pressure on the already tight L1-side storage budget. More importantly, operating such a prefetcher in the virtual-address domain would require a TLB lookup for every predicted prefetch, substantially increasing TLB bandwidth demand and translation overhead. In contrast, VIP limits TLB interaction to inter-page predictions only, where virtual-address reasoning is fundamentally required, thereby amortizing translation overhead while preserving correctness. In our preliminary exploration, we did not observe substantially larger performance gains from training a more complex L2-style prefetcher at L1D compared to VIP's targeted inter-page approach. This suggests that selectively enabling virtual-address translation only for page-cross events is a more effective design point than universally applying it to all L2 prefetches. A more comprehensive evaluation of this design space, including the performance and overhead trade-offs of frequent TLB lookups, is left as an interesting direction for future work.

References

[1] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A customizable hardware prefetching framework using online reinforcement learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1121–1137.

[2] Dimitrios Chasapis, Georgios Vavouliotis, Daniel A Jiménez, and Marc Casas. 2025. Instruction-Aware Cooperative TLB and Cache Replacement Policies. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 619–636.

[3] Yun Chen, Lingfeng Pei, and Trevor E Carlson. 2021. Leaking control flow information via the hardware prefetcher. *arXiv preprint arXiv:2109.00474* (2021).

[4] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. 2016. Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 368–379.

[5] Yasuo Ishii, Jaekyu Lee, Krishnendra Nathella, and Dam Sunwoo. 2021. Re-establishing fetch-directed instruction prefetching: An industry perspective. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 172–182.

[6] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In *Proceedings of the 42nd annual international symposium on computer architecture*. 158–169.

[7] Vasileios Karakostas, Chrysa Papagiannopoulou, and Stefanos Kaxiras. 2016. Redundant Memory Mappings: Exploiting the Potential of Page Overlays. In *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*. 49–61.

[8] Juncheng Li, Dong Niu, Zhenhua Li, and Yiming Li. 2019. Understanding and Analyzing Huge Page Management in Cloud Environments. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 35–48.

[9] OpenXiangShan. 2021. XiangShan: An Open-source High-performance Out-of-order RISC-V Processor. https://github.com/OpenXiangShan/XiangShan/blob/mdp-train-fix/src/main/scala/xiangshan/cache/dcache/mainpipe/MissQueue.scala.

[10] Glenn Reinman, Brad Calder, and Todd Austin. 1999. Fetch directed instruction prefetching. In *MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE, 16–27.

[11] Alberto Ros. 2021. Berti: An Accurate Local-Delta Data Prefetcher. In *Proceedings of the 3rd Data Prefetching Championship (DPC-3)*. https://dpc3.compas.cs.stonybrook.edu/pdfs/Berti.pdf.

[12] Christos Vavouliotis, Vasileios Karakostas, Muhammad Shafique, Kostas Psarris, and Stefanos Kaxiras. 2021. Morrigan: Designing Translation-Optimized TLB and Cache Hierarchies. In *Proceedings of the 54th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1059–1072.

[13] Jose Rodrigo Sanchez Vicarte, Michael Flanders, Riccardo Paccagnella, Grant Garrett-Grossman, Adam Morrison, Christopher W Fletcher, and David Kohlbrenner. 2022. Augury: Using data memory-dependent prefetchers to leak data at rest. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1491–1505.