Untangling the Low Accuracy of the Entangling Instruction Prefetcher

Vedant Kalbande*, Biswabandan Panda*, Alexandra Jimborean†, Alberto Ros†
*Indian Institute of Technology Bombay
†University of Murcia

***Abstract*—The Entangling Instruction Prefetcher (EIP) is a state-of-the-art mechanism designed to improve instruction-fetch timeliness. While earlier studies report an average prefetch accuracy of 72% and as high as 91%, more recent evaluations using different methodologies—most notably the gem5 simulator—report lower accuracies of 44% on average and even below 30%.**

This work aims to shed light on this controversy by providing a robust and faithful open-source implementation within the gem5 simulation framework. In addition, we analyze several design choices that influence EIP's accuracy and performance highlighting common pitfalls when porting EIP across simulators. In our experiments, using the same benchmarks as prior studies, our gem5 version of EIP achieves an average prefetch accuracy of 64.35%, demonstrating that its accuracy remains stable across the simulators.

***Index Terms*—Instruction prefetching, accuracy, modeling.**

I. Introduction

The code footprint of datacenter applications is growing with an estimated increase of nearly 30% every year [1], [2]. This trend put enormous pressure on the cache hierarchy as the instruction working set exceeds the capacity of lower-level caches, such as the first-level instruction cache (L1I) and even the second-level cache (L2). CPUs have adopted decoupled front-ends through fetch-directed instruction prefetching (FDIP) [3], [4] to tolerate L1I cache miss latencies and maintain a steady flow of instructions through the processor front-end. However, FDIP fails to hide the latency of instruction cache misses in scenarios such as branch mispredictions and large miss latencies. In these scenarios, dedicated L1I prefetchers play an important role.

The Entangling Instruction Prefetcher (EIP) [5], [6], winner of the 1st Instruction Prefetching Championship (IPC-1), represents a state-of-the-art approach to L1I prefetching aimed at improving timeliness and accuracy. EIP operates by entangling trigger instructions (source) with their future target instructions (destinations) through a correlation mechanism that considers the latency of L1I misses, enabling instructions to fill the L1I cache precisely when they are needed. By having multiple destinations for a single source, EIP naturally complements FDIP by exploring multiple control-flow paths, mitigating stalls induced by branch mispredictions. Early evaluations of EIP [5], [6] considered only correct-path execution, while a subsequent wrong-path-aware design [7] demonstrated that it remains effective under realistic front-ends.

The problem. EIP reports an average prefetch accuracy of 72% [6], that is, out of 100 L1I fills due to prefetch requests 72 cachelines are demanded before eviction. Similar accuracy trends were later observed [8] across different prefetcher sizes using the same evaluation framework (the ChampSim simulator [9]). Recent studies, such as Priority-Directed Instruction Prefetching (PDIP) [10] and Hierarchical Prefetching [11], also focus on instruction prefetching and evaluate EIP as a state-of-the-art baseline for comparison. However, these studies reported lower accuracy when reevaluating EIP alongside their proposed mechanisms: 44% in PDIP [10] and nearly 30% in Hierarchical Prefetching [11] (Figure 1). Notably, these evaluations are performed using the gem5 simulator [12], an execution-driven simulator that directly runs benchmark binaries, whereas ChampSim is a trace-based simulator. While these discrepancies in accuracy may initially appear to stem from methodological differences, such as the use of the gem5 simulator and distinct benchmark suites, our investigation revealed challenges in implementing EIP within gem5 that warrant further clarification.

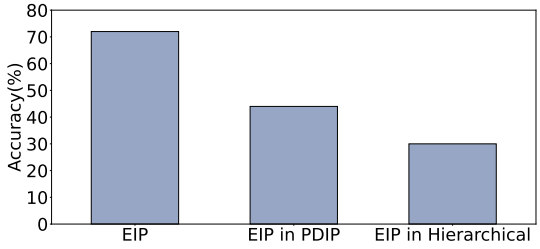

Fig. 1. Prefetch accuracy of EIP as reported by different instruction prefetching papers EIP [7], PDIP [10], and Hierarchical [11]

Goal and contributions. Our goal is to investigate the reported drop in accuracy when evaluating EIP within the gem5 framework. To this end, we start from a gem5 configuration modeling a decoupled front-end [13] and implement the latest open-source version of EIP [7][1] both with and without its wrong-path awareness (WPA) feature, identifying several subtleties that are easy to overlook during implementation. In addition, because both existing gem5 implementations of EIP [10], [11] train the prefetcher at *retire*; preserving its timely property with that scheme is not straightforward. We describe how to adapt the training while maintaining EIP's timely behavior. Finally, we examine the released code for Hierarchical Prefetching[2,3] and note several design decisions that diverge from the original EIP implementation. Our im-

[1]https://github.com/alberto-ros/EntanglingInstructionPrefetcher
[2]https://github.com/CRAFT-THU/gem5-hp
[3]PDIP artifact contains only the binary, without the source code.

plementation of EIP in gem5 achieves an average accuracy of 64.35% across a benchmark suite comparable to that used in prior work (PDIP [10]), confirming EIP's robustness across simulation platforms. We report an average speedup of 3.1%, primarily driven by the `verilator` workload. We open source our implementation for the research community.

II. Background on EIP

This section provides a brief overview of the EIP prefetcher to set the foundation for porting EIP (originally implemented in ChampSim) to the gem5 simulator.

The two key concepts of EIP are *basic blocks* and *entangling pairs*. A basic block corresponds to a sequence of linear accesses (either accessing the same cacheline as before or the next). The head of a basic block is the first accessed cacheline in a basic block. All heads of basic blocks that miss in L1I are stored in a History Table along with their timestamp. This table is used to form entangled pairs: A source cache line (the trigger) and a destination cache line (the target to be prefetched). The mechanism is built in two phases: training and prefetching.

Training: EIP measures the latency of L1I misses and on a cacheline fill it links (entangles) the head of the basic block of the incoming cacheline (destination) with the head of basic block accessed earlier enough to have time to prefetch the missing cacheline (sourcet). These pairs are stored along with the size of basic blocks in the Entangled Table. Each pair is also associated with a 2-bit confidence counter, which is incremented (or saturated) on accurate prefetches and decremented on inaccurate ones. To save space, destinations are compressed and stored relative to the source.

Prefetching: When a source cacheline is accessed, EIP prefetches both the source's entire basic block (for spatial locality) and the entangled destination's basic block considering the confidence counter assigned to the pair is not zero. EIP handles variations in latency through this confidence counter.

A. Wrong-Path-Aware EIP (EIP-WPA)

Wrong-path accesses affect EIP during training and prefetching. During *training*, EIP may detect basic blocks and create entangled pairs during execution of wrong-path instructions. If they are stored in the Entangled Table, it would be polluted with useless information, possibly evicting correct-path information. During *prefetching*, prefetch requests may be triggered by wrong-path instructions. These requests, once sent into the memory hierarchy, are difficult to cancel even if the triggering instruction is later squashed. Fortunately, the major source of performance loss comes from wrong-path training, while prefetch requests triggered from the wrong path have only a minor impact [7]. Therefore, EIP must primarily prevent wrong-path training, without interfering with its timeliness.

There are two main methods to minimize wrong path interference: (1) training speculatively at fetch and squash on branch mispredictions, or (2) training at retire. EIP-WPA [7] implements the first one. The key goal is to ensure that if a basic block or entangled pair is created during wrong-path execution, it does not enter the Entangled Table. This is achieved by informing the prefetcher whenever a squash event (e.g., branch misprediction recovery) occurs, and applying three microarchitectural techniques:

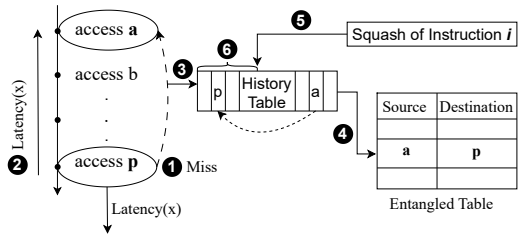

Fig. 2. Overview of Wrong-Path-Aware EIP for a superscalar CPU

Squashing the History Table: Wrong-path instructions can pollute the History Table and become invalid sources or destinations. To prevent this, each History entry is tagged with an instruction sequence ID. On a squash, the processor informs the prefetcher of the ID of the mispredicted branch. All History entries created after this branch are then flushed. This way only correct-path instructions remain in the History Table, while wrong-path accesses are discarded, hopefully before they influence future entangling.

Updating Runtime Basic Block Information: Whenever a branch misprediction is detected, the prefetcher adjusts the size of the current basic block. The block is truncated at the mispredicted branch, so only correct-path instructions remain part of the block. This correction has the largest positive impact, as in most cases only correct-path basic block information is propagated into the Entangled Table.

Delaying Insertion into the Entangled Table: In the original EIP implementation, once a new basic block or entangled pair is detected, it is inserted immediately into the Entangled Table. If this happens on the wrong path, the pollution is permanent because cleaning the Entangled Table is difficult. To solve this, Entangling WPA keeps new blocks and pairs temporarily in the History Table instead of inserting them directly. Since the History is flushed on each squash, wrong-path data is automatically removed. Only when entries are evicted from the History (FIFO order) are they promoted into the Entangled Table, which significantly increases the likelihood that only correct-path information is preserved.

B. Illustrated overview

Figure 2 illustrates how the EIP-WPA works at the high level. Upon an instruction-cache miss (❶), the miss latency is computed, which is used to entangle the missed basic block with an appropriate source basic block entry in the History Table (❷, ❸). When an entry is evicted from the History Table, it is inserted in the Entangled Table along with its destination entangled pairs (❹). In addition, upon a pipeline squash for instruction *i*, all History Table entries corresponding to wrong-path instructions following *i* are identified with the help of sequence ID and then flushed(❺, ❻). Overall, EIP-WPA preserves the high accuracy and coverage of the original EIP, while eliminating the main source of inefficiency caused by wrong-path execution.

III. Untangling EIP implementation in gem5

10K feet view of gem5 prefetching framework. Gem5 provides a base prefetching framework, `BasePrefetcher`, which manages how prefetch requests are issued. Prefetchers in gem5 only need to generate target addresses and push them into a prefetch queue. The `BasePrefetcher` then issues a prefetch request from that queue only when the address is not present in the cache or MSHR, and it is also responsible for translation when the prefetcher generates virtual addresses. We leverage this framework to implement EIP.

Training on all types of accesses. By default, gem5 invokes the prefetcher only on demand misses, and not on all demand accesses (i.e., both hits and misses). However, EIP requires training on all demand accesses regardless of whether they result in a cache hit or miss. Therefore, the `prefetch_on_access` flag should be enabled for the EIP prefetcher. Furthermore, gem5 does not implicitly invoke the prefetcher on prefetch hits, i.e., a demand access hits a cache line that was previously brought in by a prefetcher. So, to train on them as well, the `prefetch_on_pf_hit` flag should be enabled for EIP. Our results show that the average accuracy drops by 10% if EIP is not trained on prefetch hits. We observed however that open-source prior-work re-implementations of EIP in gem5 do not train the prefetcher on `prefetch_on_pf_hit`.

Synchronizing shadow structures. The open-source EIP [6] code relies on extending the L1I cache and MSHR with metadata by creating a shadow copy of the cache tags and MSHR tags, rather than directly adding extra fields to the existing hardware components. This approach enables seamless integration into the gem5 prefetching framework without requiring modifications to the simulator's core structures. *However, maintaining synchronization between these shadow structures and the actual caches remains critical for achieving high prefetching accuracy and performance,* since all events that modify the tags in the simulator structures needs to inform the prefetcher.

An integration challenge arises because a shadow MSHR entry must be created *only when a prefetch request is actually issued*. Since this decision is made by the `BasePrefetcher`, the `BasePrefetcher` must be extended to create a corresponding entry in the shadow MSHR whenever a prefetch request is emitted. In addition, when EIP enqueues an address into the prefetch queue, EIP passes the associated source set and source way information. This metadata is required to correctly populate the shadow MSHR and update the Entangled Table. We observed that available EIP implementations in gem5 extend existing core structures directly rather than using shadow structures within the prefetcher, which complicates the porting process and increases the risk of inconsistencies (see Section V). Our solution, however, correctly updates the shadow structures in the EIP code.

IV. Evaluation

A. Simulation methodology

We use gem5 [12] with FDIP support [13] to model a detailed out-of-order superscalar CPU in full-system (FS) mode running Ubuntu 18.04 (Linux kernel 4.15). Our experimental setup is based on the simulation infrastructure developed by the EMISSARY authors, and as workloads we utilize the EMISSARY checkpoints of datacenter applications publicly released as part of their artifact [14]. Our configuration closely resembles Intel's Alder Lake like cache hierarchy [15], with an LRU based replacement policy at different cache level, summarized in Table I. The baseline employs the FDIP instruction prefetcher [3], and the cache hierarchy is configured as non-inclusive. Each datacenter workload is simulated for 100M instructions following a warm-up of 50M instructions.

TABLE I
Parameters of the simulated system

Core (Aarch, 64-bit ARM)	
Fetch Target Queue	24 entry, 192 instructions
Branch Predictor	TAGE, 16K-entry BTB
Fetch/Decode/Commit	8 wide
ROB/Issue/Load/Store Queue	512 / 240 / 128 / 72
Cache Hierarchy (Non-inclusive)	
L1I	32KB, 8-way, 4 cycles, 16 MSHRs
L1D	64KB, 8-way, 6 cycles, 16 MSHRs
L2	1MB, 16-way, 16 cycles, 32 MSHRs
LLC	2MB, 16-way, 22 cycles, 64 MSHRs
EIP Parameters	
Entangled Table	4K entries (256 sets, 16 ways)
History Table	32 entries, 6 bits for basic block sise
History Table searches	5 entries for merging, 24 for entangling

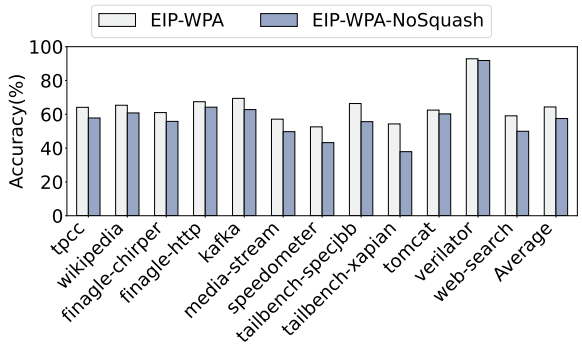

Fig. 3. Prefetch accuracy of different configurations of EIP-WPA

After incorporating the gem5-specific changes described in Section III, we evaluate EIP-WPA in detail. To assess the impact of wrong-path training, we also consider a variant that does not flush the history table on squashes (EIP-WPA-NoSquash). Both variants use the configuration parameters described in Table I.

B. Results

Accuracy. Prefetch accuracy is calculated as the fraction of useful prefetches over the total number of useful and useless prefetches. Figure 3 shows the accuracy of EIP-WPA and EIP-WPA-NoSquash. EIP-WPA achieves an average accuracy of 64.35%, demonstrating its effectiveness in execution-driven simulators such as gem5. EIP-WPA-NoSquash attains a lower average accuracy (57.50%), yet it remains higher than the values reported in previous gem5-based studies. The decrease in accuracy for EIP-WPA-NoSquash across all benchmarks indicates the negative effects of wrong-path training.

Speedup. Figure 4 reports the speedup as the improvement in IPC relative to a baseline configuration with FDIP but

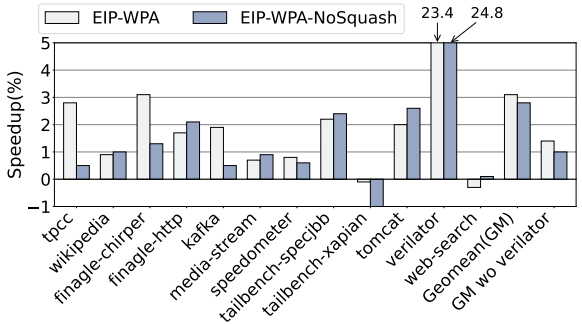

Fig. 4. Speedup over baseline FDIP without a dedicated L1I prefetcher

without any additional prefetcher. Across 12 datacenter benchmarks, EIP-WPA achieves an average speedup of 3.1%. This improvement is primarily due to a reduction in demand MPKI from 40.67 to 32.58 in L1I. `Verilator` is an outlier, exhibiting a speedup of 23.4%. Excluding `verilator`, the average speedup is about 1.4%, with moderate performance gains observed across the majority of the remaining benchmarks. The EIP-WPA-NoSquash configuration achieves an average speedup of 2.8% (1.0% without `verilator`). The slight performance improvement observed in some benchmarks for EIP-WPA-NoSquash compared to EIP-WPA is primarily due to more aggressive prefetching, as the History Table is not frequently squashed.

C. Sensitivity study of Entangled Table in EIP-WPA

Beyond a correct implementation, a sensitivity analysis is recommended to fine-tune the prefetcher for the new framework. Increasing the Entangled Table size can significantly enhance performance: with 8K entries geomean speedup increases to 5.8% (1.4% without `verilator`), while with 16K entries increases to 7.9% (2.0% without `verilator`).

V. Discussion

EIP on retire. EIP-WPA trains speculatively on L1I accesses and flushes the History Table upon a pipeline squash. However, wrong-path information may still reach the Entangled Table. To avoid this, the prefetcher can instead be invoked at instruction retirement, as in the EIP implementations used by PDIP [10] and Hierarchical Prefetching [11].

With this approach, when an instruction retires, the prefetcher is invoked and the corresponding new basic block is either inserted into the History Table or merged with an existing entry. If the retired instruction was an instruction cache miss, an entangled pair is formed and updated in the Entangled Table. This requires searching the History Table to identify the source head of basic block that arrived at least miss-latency-cycles (of the current instruction) earlier to achieve timeliness at the time of prefetching. The search is performed by iterating over the History Table and selecting entries whose timestamps satisfy Equation 1, where TT denotes the timestamp at L1I access time associated with the current basic block (currentBB) or a basic block at position i in the History Table, and the right side is the miss latency of the current retired instruction.

$$TT_{\text{currentBB}} - TT_{\text{i}} \geq \text{MissLatency}_{\text{currentInst}} \quad (1)$$

We found that existing gem5 implementations of EIP rely on retire timestamps for computing Equation 1. *We note that the timestamps recorded at retire do not accurately reflect the timing relationships observed at fetch, which are essential for timely entangling pair formation.* To mitigate this issue, an auxiliary buffer can be introduced to record basic block timestamps at fetch time, so that they can later be used by the prefetcher when populating the History Table at retire, ensuring timely entangling of pairs.

Other differences. We find that existing gem5 implementations omit confidence updates for timely prefetching, a mechanism explicitly included in the original EIP. They also differ in destination compression, supporting only four entries instead of six. Additional inconsistencies made correctness verification difficult, so we opted to develop and release an independent implementation.

VI. Conclusion

This work revisits the Entangling Instruction Prefetcher by providing a complete and validated implementation within the gem5 simulator. Our analysis clarifies the sources of previously reported discrepancies in accuracy and demonstrates that EIP retains high effectiveness when evaluated under consistent simulation environments. To foster reproducibility and further research on instruction prefetching, we will release the complete gem5-compatible codebase, allowing the community to build upon this foundation, explore new design variations, and fairly compare future proposals.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (819134), from the MCIN/AEI/10.13039/501100011033/ and the "ERDF A way of making Europe", EU (PID2022-136315OB-I00), and from the FSRM/10.13039/100007801 (23039/GERM/25).

References

[1] G. Ayers *et al.*, "Asmdb: understanding and mitigating front-end stalls in warehouse-scale computers," in *ISCA*, 2019.
[2] S. Kanev *et al.*, "Profiling a warehouse-scale computer," in *ISCA*, 2015.
[3] G. Reinman *et al.*, "Fetch directed instruction prefetching," in *MICRO*, 1999.
[4] Y. Ishii *et al.*, "Re-establishing fetch-directed instruction prefetching: An industry perspective," in *ISPASS*, 2021.
[5] A. Ros and A. Jimborean, "The entangling instruction prefetcher," in *The 1st Instruction Prefetching Championship (IPC1)*, May 2020.
[6] A. Ros and A. Jimborean, "A cost-effective entangling prefetcher for instructions," in *ISCA*, 2021.
[7] A. Ros and A. Jimborean, "Wrong-path-aware entangling instruction prefetcher," *IEEE Transactions on Computers*, 2024.
[8] G. Chacon *et al.*, "Composite instruction prefetching," in *ICCD*, 2022.
[9] "The championship simulator: Architectural simulation for education and competition," https://arxiv.org/pdf/2210.14324, 2020.
[10] B. R. Godala *et al.*, "Pdip: Priority directed instruction prefetching," in *ASPLOS*, 2024.
[11] T. Zhang *et al.*, "Hierarchical prefetching: A software-hardware instruction prefetcher for server applications," in *ASPLOS*, 2025.
[12] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011.
[13] "Gem5-emissary," https://github.com/PrincetonUniversity/gem5_FDIP.git, 2023.
[14] B. R. Godala, "Datacenter workloads of emissary," https://drive.google.com/file/d/1ac60R-nuENQjw-rRBR-0S9rYQEEuCvyp/view, 2024.
[15] "Intel alderlake," https://en.wikipedia.org/wiki/Alder_Lake, 2021.