Energy Consumption Analysis of Instruction Cache Prefetching Methods

Morteza Baradaran
Department of Computer Science
University of Virginia
Charlottesville, USA
rgq5aw@virginia.edu

Ali Ansari
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
aansaricesut@gmail.com

Mohammad Sadrosadati
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
m.sadr89@gmail.com

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
azad@sharif.edu

***Abstract*—Frequent instruction cache (L1-I) misses pose a significant performance bottleneck in modern processors, especially for applications with large instruction footprints, such as server applications. To address the L1-I misses, there have been various proposals for L1-I prefetching over the past two decades. The designers of L1-I prefetchers primarily focused on enhancing performance while minimizing the area overhead. However, they paid little attention to the resulting increase in energy consumption due to incorporating an L1-I prefetcher. Furthermore, prior works assume L1-I prefetcher's energy consumption is mainly due to its area overhead. In this work, we demonstrate that a substantial proportion of the energy consumption associated with using an L1-I prefetcher is attributed to the increased L1-I accesses initiated by the L1-I prefetcher. To compensate for the energy consumption of more accesses to the L1-I, we propose an approach to decrease energy per access to the L1-I by reducing its associativity. Our experimental results demonstrate that reducing L1-I associativity from 8 to 2 effectively reduces the energy consumption of L1-I prefetchers. As a result, the energy saving achieved through our approach (on average $113.7\ nJ/ki$) compensates for the energy consumption overhead caused by the L1-I prefetcher on the baseline system, with the average and the highest energy overhead at $41.6\ nJ/ki$ and $74.8\ nJ/ki$, respectively, while the associated performance loss (0.8% on average) remains negligible.**

***Index Terms*—energy consumption, instruction prefetching, instruction cache, cache associativity**

I. Introduction

Emerging server workloads are characterized by large instruction working sets. These working sets can easily overwhelm private instruction caches, leading to frequent instruction cache (L1-I) misses. Therefore, the frequent L1-I misses become a key source of performance degradation in modern processors [2], [7], [10], [11]. In order to overcome this problem, a number of L1-I prefetching techniques have appeared in the past two decades [1]–[5], [7].

Prior research on L1-I prefetching techniques has primarily focused on improving performance while minimizing on-chip area overhead [1], [4], [5]. These techniques indirectly contribute to lower energy consumption by reducing the storage requirements of the L1-I prefetcher. However, reducing the energy consumption of prefetchers has not been a main objective in prior work. Furthermore, in the light of dark silicon constraints [15], on-chip components, including prefetchers, must operate with high energy efficiency, and simply turning off the prefetcher is not a viable option, particularly for high-performance server workloads [16]. Therefore, reducing the energy consumption of L1-I prefetchers has become an important research area.

The energy consumption incurred by the system through utilizing an L1-I prefetcher can be attributed to two sources: (1) The energy consumed by the on-chip metadata storage of an L1-I prefetching scheme and (2) The additional energy consumption that is imposed on the L1-I and L2 caches as a result of the increased accesses originating from the L1-I prefetcher. In this work, we show that a substantial fraction of the energy consumption imposed on the system in the presence of an L1-I prefetcher is caused by the implicit pressure of the L1-I prefetcher on the cache hierarchy, i.e., L1-I and L2, rather than by the prefetcher's metadata. In order to accomplish this objective, we evaluate 4 different L1-I prefetchers RDIP [1], FNL-MMA [4], MANA [5] and PIF [2] in terms of their energy consumption and performance.

We propose a novel approach to tackle the issue of excessive energy consumption overhead imposed on the cache hierarchy while employing an L1-I prefetcher. Our approach involves changing the configuration of the L1-I cache to reduce the energy consumed per L1-I access. We study the energy per access of L1-I with different associativities and sizes to achieve an optimum configuration for the L1-I. Then, we compare the energy consumption and performance of the selected L1-I prefetching methods with our proposed L1-I configuration against the baseline system, a system without an L1-I prefetcher. Our observations indicate that using the proposed L1-I configuration, specifically by changing the associativity from 8 to 2, results in a substantial decrease

979-8-3503-8160-3/23/$31.00 ©2023 IEEE
DOI 10.1109/SBAC-PADW60351.2023.00019

in energy per access and total energy consumption overhead of the L1-I cache (on average $113.7\ nJ/ki$), with minimal impact on performance (up to 0.8% on average.) This energy consumption reduction is enough to effectively mitigate the energy consumption overhead caused by the L1-I prefetcher in comparison to the baseline system, with the average and the highest energy overhead at $41.6\ nJ/ki$ and $74.8\ nJ/ki$, respectively. This highlights the effectiveness of our proposed approach in mitigating the excessive energy consumption imposed on the cache hierarchy due to L1-I prefetchers.

To further reduce the energy consumption overhead of using an L1-I prefetcher, we examined the impact of altering the front-end configuration of one of our selected prefetchers, namely MANA, on both the system's energy consumption and overall performance. The energy-optimized front-end configurations demonstrate $25.6\ nJ/ki$ lower energy consumption for L1-I, L2, and L1-I prefetcher storage (including static and dynamic overhead) compared to the performance-optimized configurations, with less than 1% performance loss. Our findings highlight the crucial role of front-end selection in creating a more energy-efficient prefetcher solution.

The rest of the paper is organized as follows: Section II presents our evaluation methodology and baseline system as well as the four selected L1-I prefetchers; Section III explores the results and evaluations while also elaborating on the motivation behind considering modifications to the L1-I configuration to compensate for energy conservation resulting from the L1-I prefetcher; Section IV discusses the findings of our study and delves into their consequential implications; Section V discusses the background of our selected L1-I prefetching methods. Finally, Section VI concludes this work.

II. Methodology

In this section, we describe the tools and methodologies that we employed to conduct our research and evaluate each of the selected prefetching methods. We also provide an overview of the four prefetching methods we used in our performance and energy consumption analysis.

A. Simulation Infrastructure

To analyze the performance and energy consumption of the selected prefetchers, we use ChampSim [8] simulator and CACTI-7 [14], respectively.

ChampSim [8] is a trace-driven simulator provided by the 1st Instruction Prefetching Championship (IPC-1 [9]). We use Champsim with the default IPC-1 configurations. We launch all simulations with 50 public benchmarks provided by IPC-1. To make results more accurate, we allocate 50 million instructions to warm up the system, including the caches, the branch predictor, and prefetchers' metadata. The subsequent 50 million instructions are used to collect the evaluation metrics, such as Instructions Per Cycle (IPC) and access counters.

CACTI [14] is an integrated tool to calculate cache access time, cycle time, area, leakage, and dynamic power model for modern computer architectures. We use CACTI-7 to measure the energy per access, dynamic energy, and static energy of tables and structures exploited in our selected prefetchers.

B. Competing L1-I Prefetchers

In order to comprehensively understand the energy consumption of different prefetching techniques, we choose four representative L1-I prefetchers. We select two of the prefetchers, MANA [5] and FNL-MMA [4], from methods proposed in the Instruction Prefetching Championship (IPC-1 [9]). Due to the storage constraints of the IPC-1, the designers of MANA and FNL-MMA also implemented a storage-efficient version of their methods. These storage-efficient versions minimize the metadata size of the L1-I prefetcher, making them suitable for systems with limited storage resources. The other two L1-I prefetching methods, PIF [2] and RDIP [1], are widely recognized approaches that require significantly larger metadata. PIF is known for its high-performance capabilities but requires large metadata, i.e., 236 KB. On the other hand, RDIP exhibits lower performance compared to PIF, but its metadata size is relatively smaller. Further details about the four selected L1-I prefetchers are outlined in section V. Below, we provide additional information regarding the configurations of both the selected prefetchers and our baseline system, which we use in our evaluations.

baseline: Table I summarizes our baseline system configuration, against which all performance gains and energy consumption changes are compared. Notably, our baseline system L1-I does not utilize any prefetcher.

TABLE I
Baseline Characteristics

Parameter	Value
Core	14 nm, a single 4 GHz OoO core, 352-entry ROB, 128-entry Load Queue, 72-entry Store Queue
Fetch Unit	32 KB L1-I, 8-way, 64B block size, 4-cycle latency, hashed-perceptron branch predictor [13], 2K entry Branch Target Buffer, 8 MSHRs
L1-D Cache	48 KB, 12-way, 64B block size, 5-cycle latency, 16 MSHRs, next-line prefetcher
L2 Cache	512 KB, 8-way, 10-cycle latency, Signature Path Pattern (SPP) [12] prefetcher
Last Level Cache	2 MB, 16-way, 20-cycle latency, 64 MSHRs

RDIP: We choose all the parameters based on the original proposal of RDIP [1]. The Miss Table is a 4K entry, 4-way set associative table, where each entry holds three trigger addresses and the associated footprint of the detected instruction cache misses. This prefetcher imposes 83 KB storage on the baseline design.

PIF: The history buffer of PIF can accommodate up to 32K spatial regions. To find a record in the history buffer, PIF uses an index table that holds pointers to the history buffer's entries. We model this index table with a 4-way set associative and 2K sets, as in the original proposal. PIF imposes over 236 KB of storage overhead to the baseline design. The temporal

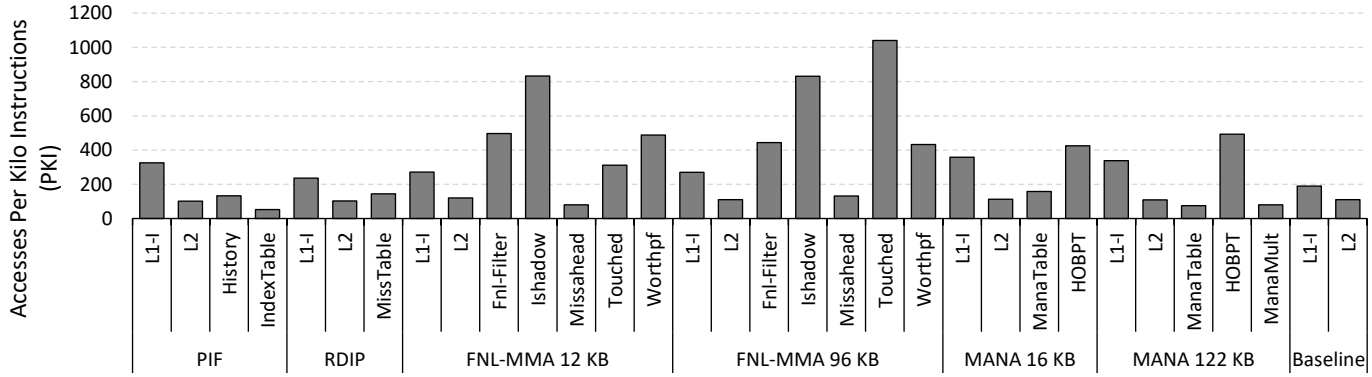

Fig. 1. Accesses per Kilo instructions for all the metadata used in the 4 selected prefetchers with a 32 KB and 8-way L1-I

compactor contains eighteen spatial regions, and the lookahead is five, as in the original proposal. We also use four SABs where each one tracks twelve consecutive spatial regions [2].

MANA: We select all the configuration parameters based on the original proposal. MANA [5], [6] uses a 4K entry table of 1K sets to store the spatial regions. Each MANA Table record consists of 7 bits to indicate the index in High-Order-Bits Patterns' Table (HOBPT), a 2-bit partial tag, an 8-bit footprint, and a 12-bit pointer to the successor spatial region. Moreover, it uses a 128-entry, 8-way set associative HOBPT. The overall storage budget of MANA in the storage-efficient format is 15 KB. We also use the extended format of MANA by just changing the size of MANA Table and HOBPT to 16K entry and 1024-entry, respectively, and adding a 4K entry MANA-Multiple Table. Hence, the extended format of MANA requires 122 KB storage.

FNL-MMA: The storage budget required for the extended format of the FNL-MMA [4] is close to 96 KB: 408 bytes for the I-Shadow cache (192 17-bit entries), 8 KB for the Touched table (64K 1-bit entries), 16 KB for the WorthPF table (64K 2-bit entries), 71 KB for the miss ahead prediction table (8K 71-bit entries), 116 bytes for the MMA filter (16 58-bit addresses) and 136 bytes for the FNL filter (128 17-bit entries). In the storage-efficient format, the size of the WorthPF, Touched, and miss ahead tables are divided by 8, and hence the overall size, in this case, is about 12 KB.

C. Energy Consumption Calculation Methodology

The energy consumption of an L1-I prefetcher is derived from two primary sources: (1) The energy consumption related to the prefetcher's metadata, and (2) The energy consumption overhead imposed on the baseline system due to an increased number of accesses to the cache hierarchy in the presence of the L1-I prefetcher, including dynamic and static energy consumption. In this section, we explain our evaluation methodology for quantifying the energy consumption of the four selected L1-I prefetchers.

First, we focus on the evaluation method to determine energy consumption related to the L1-I prefetcher's metadata. As described in sections II-B and V, the metadata storage in each L1-I prefetcher scheme consists of one or two primary, large tables, as well as several smaller structures such as queues and buffers. We assume that the majority of the energy consumption in these methods belongs to the large tables. Based upon this assumption, we count the number of demand accesses to each of the large tables using ChampSim. Next, we calculate the dynamic energy of each table by multiplying the energy per access of each table entry obtained from CACTI-7 with the number of accesses obtained in the previous step. We also calculate the static energy related to each table by multiplying the leakage power (reported by CACTI-7) by the simulation time. Finally, we sum the dynamic and static energy numbers to obtain the total energy consumed by the tables of the L1-I prefetcher.

We calculate the energy consumption overhead imposed on the cache hierarchy (L1-I and L2) in the presence of an L1-I prefetcher. To do so, we first compute the energy consumption of L1-I and L2 in both the system with and without the L1-I prefetcher. Then, we subtract the two obtained values to calculate the additional energy consumption imposed on the system due to the presence of the L1-I prefetcher. To compute the dynamic energy consumption of L1-I and L2 caches, we count the number of demand accesses to L1-I and L2 using ChampSim. Then, we calculate the dynamic energy of L1-I and L2 by multiplying the energy per access of L1-I and L2 obtained from CACTI-7 with the number of accesses obtained in the previous step.

As mentioned before, we count the additional number of accesses initiated from the L1-I prefetcher to its metadata as well as L1-I and L2 throughout simulations using ChampSim. To provide a normalized perspective, we divide the obtained numbers by the total number of instructions (50 million) and multiply the result by 1000. This normalization allows us to present the data in a standardized format and facilitate comparisons across different prefetchers. By multiplying the attained normalized number of accesses by the energy per access obtained from CACTI-7, we calculate the normalized energy consumption (nJ/ki).

III. Evaluation

In this section, we elaborate on our analysis of the 4 selected L1-I prefetching methods, focusing on their performance and

energy consumption. Then, we will explore the motivation driving the adjustment of L1-I configurations, aiming to compensate for energy consumption overhead introduced to the baseline system by the L1-I prefetcher. In addition, we evaluate our proposed approach by comparing it to the baseline system in terms of both performance and energy consumption. Finally, we discuss the impact of modifying the prefetcher's front-end configuration, specifically focusing on reducing energy consumption without significant performance degradation. Our baseline system in all experiments has a 32 KB 8-way set associative instruction cache without an L1-I prefetcher. More details about our evaluation methodology and the four L1-I prefetchers we have chosen are provided in sections V and II-B.

Figure 1 presents the number of accesses per Kilo instructions to the L1-I prefetcher tables, L1, and L2 for the selected L1-I prefetchers. This figure reveals that FNL-MMA experiences more access to its tables than other prefetchers. Furthermore, the figure indicates a significant increase in the number of accesses to the L1-I and L2 in the presence of prefetchers, compared to the baseline system, with the additional accesses to L1-I substantially far surpassing those to L2. Therefore, to address the additional energy consumption imposed on the system due to these increased accesses to L1-I, we propose an approach to decrease the energy per access in L1-I in the following section.

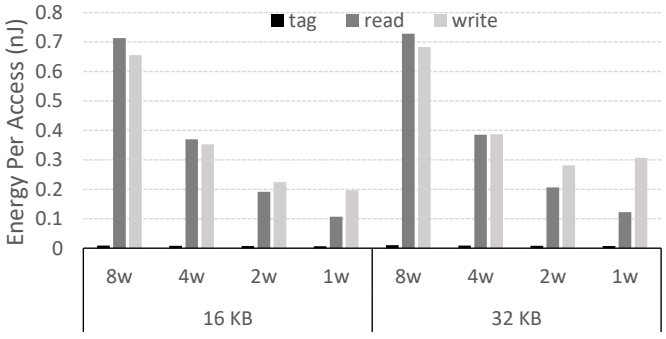

Fig. 2. Energy per access for different L1-I configurations

A. Changing L1-I Configuration

As stated earlier, a large fraction of the energy consumption induced by an L1-I prefetcher is due to the amplified number of accesses to the L1-I, resulting in higher dynamic energy consumption. One approach to mitigate the effect of enormous additional accesses to the L1-I initiated by the L1-I prefetcher is to decrease the energy per access of L1-I. In this section, we aim to explore alternative configurations for the L1-I cache that effectively decrease its energy per access.

Figure 2 shows the energy per access (including read, write, and tag lookup operations) of L1-I for various associativities ($1, 2, 4$, and 8-way) and sizes (32 and 16 KB.) By reducing the associativity of a 32 KB L1-I from 8 to 2, the energy of read and write access will decrease around $3\times$ and $2\times$, respectively. However, as can also be seen in the figure, changing the L1-I size from 32 KB to 16 KB does not noticeably affect the energy per access. Hence, reducing associativity from 8 to 2 is sufficient to significantly decrease the energy per access, thereby effectively mitigating the additional energy consumption resulting from the increased accesses from the L1-I prefetcher to the L1-I cache.

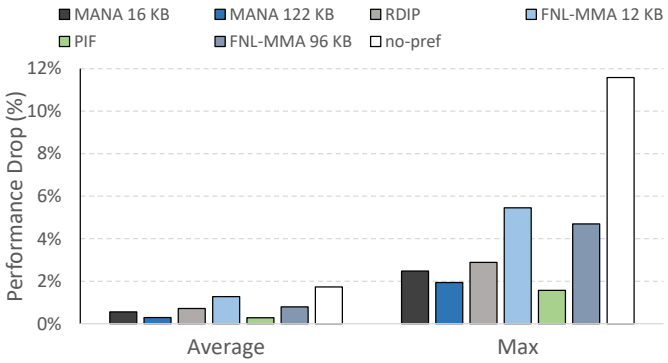

Fig. 3. Average and Min performance drop of the L1-I 32 KB, 2-way

In addition, the main purpose of higher cache associativity is to alleviate cache conflicts and prevent cache thrashing, which arises when cache lines are frequently evicted and reloaded due to limited associativity. However, it's important to note that in server workloads, the predominant issue is often capacity misses rather than conflict misses. Given the proactive role of the L1-I prefetcher in fetching a significant portion of future demanded blocks to the L1-I cache, larger associativity (i.e., 8-way) may not be essential anymore.

To evaluate the effectiveness of the L1-I prefetcher in compensating for lower L1-I associativity, we assess our proposed L1-I configuration (32 KB and 2-way) with and without the utilization of L1-I prefetchers. Figure 3 presents the performance drop of changing associativity of a 32 KB L1-I cache from 8-way to 2-way across four selected L1-I prefetchers. The figure displays the geomean and maximum performance drop of the 50 workloads, alongside the baseline's maximum and average performance drop. By decreasing the L1-I associativity from 8 to 2, the performance decreases about 1.2% on average and about 12% in the worst case in the baseline system, which is a system without an L1-I prefetcher. It can be inferred that by presenting an L1-I prefetcher to the system, regardless of the specific prefetcher employed, we observe a minimal decrease in average performance for the proposed L1-I configuration. Therefore, we conclude that decreasing the associativity of the L1-I will not affect the performance significantly in the presence of the L1-I prefetchers.

In summary, our proposed L1-I configuration to reduce its energy per access would be 32 KB and 2-way, and we assess this configuration using the 4 selected prefetchers in the next section.

B. Evaluation of the selected L1-I prefetchers

Figure 4 illustrates the average speedup achieved by employing each of the selected L1-I prefetchers alongside a 32

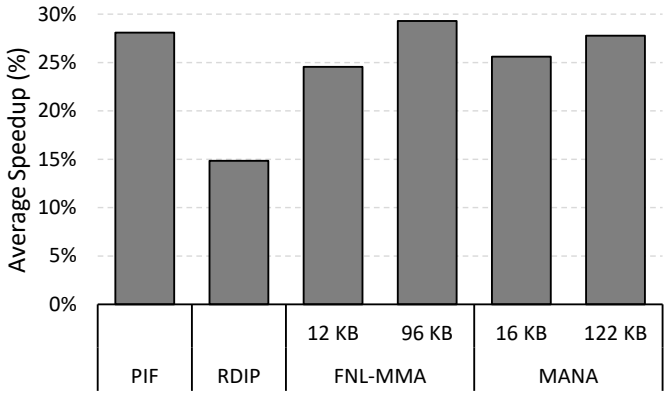

Fig. 4. Average speedup of 4 selected prefetchers (32 KB, 8-way L1-I)

KB 8-way L1-I cache over the baseline system. For MANA and FNL-MMA, we also measured the speedup obtained when using their storage-efficient formats, namely 16 KB for MANA and 12 KB for FNL-MMA. Based on figure 4, PIF, MANA 122 KB, and FNL-MMA 96 KB perform similarly in terms of speedup, ranging from 27% to 29%. Furthermore, if we use the storage-efficient format of MANA and FNL-MMA, the performance degradation ranges from 2% to 5%. Additionally, the performance improvement achieved by using the RDIP prefetcher is not substantial (15%) compared to the other prefetchers.

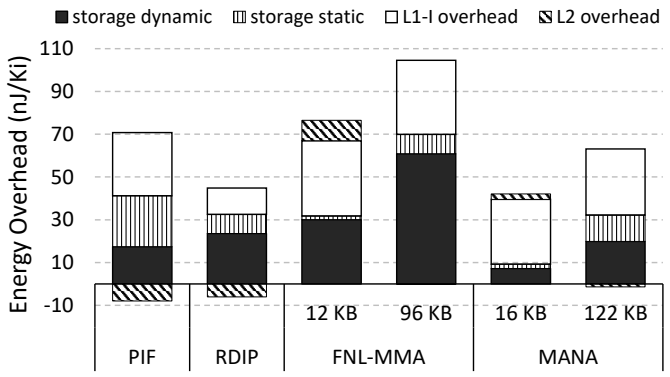

Fig. 5. Energy overhead of 4 selected prefetchers (32 KB, 8-way L1-I)

Figure 5 presents the energy consumption overhead of employing each of the selected L1-I prefetchers alongside a 32 KB 8-way L1-I cache, as compared to the baseline system. This overhead includes both the dynamic and static energy consumption of the L1-I prefetcher, as well as the additional energy consumption resulting from increased accesses to L1-I and L2 caches initiated from the selected prefetchers. Further details on the methodology used for calculating the energy consumption of an L1-I prefetcher can be found in section II. As shown in the Figure 5, MANA 16 KB exhibits the lowest storage energy (dynamic + static) overhead (less than 10 nJ/ki), which is mainly due to its significantly smaller metadata size. Additionally, FNL-MMA 96 KB exhibits the highest dynamic storage-related energy consumption (60 nJ/ki), which is attributed to its frequent access to large bit-wise structures during run time. In terms of static energy consumption of L1-I prefetcher's metadata, PIF, renowned for its extensive metadata requirements, demonstrates the highest level of static energy consumption (20 nJ/ki).

Regarding the energy consumption overhead imposed on the cache hierarchy, as depicted in Figure 5, the L2 energy consumption overhead is 9.5 nJ/ki for FNL-MMA 12 KB and 2.5 nJ/ki for MANA 16 KB. This additional energy overhead on L2 for the storage-efficient format of FNL-MMA and MANA signifies that reducing the size of metadata in the L1-I prefetcher leads to increased pressure on the L2 due to additional accesses to L2 required to bring the demanded blocks by the prefetcher. Furthermore, L2 energy consumption overhead is negative for PIF, RDIP, FNL-MMA 96 KB, and MANA 122 KB. This suggests that employing L1-I prefetchers can reduce the number of accesses to L2, resulting in decreased (negative in the figure) energy consumption compared to the baseline system. However, the energy consumption overhead of L1-I is substantially high (28.8 nJ/ki on average) across all the selected L1-I prefetchers. This observation motivates us to adopt our proposed approach, aiming to reduce the energy per access and, hence, the overall energy consumption of L1-I in the presence of L1-I prefetchers.

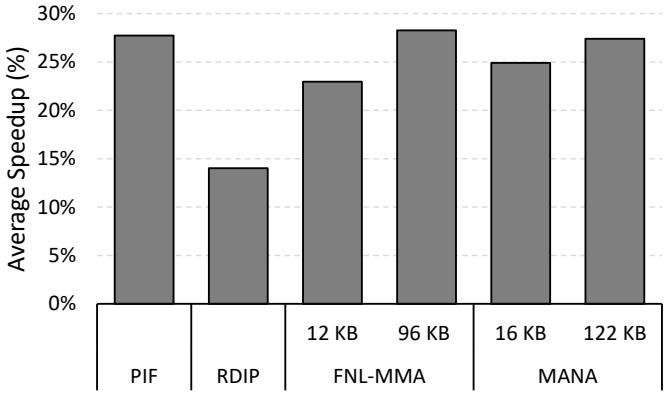

Fig. 6. Average speedup of 4 selected prefetchers (32 KB, 2-way L1-I)

C. Evaluation of the Selected L1-I Configuration

In order to mitigate the energy consumption overhead due to the presence of the L1-I prefetchers, we use our proposed configuration (i.e., 32 KB and 2-way) for the L1-I and repeat the experiments in the previous section. Figure 6 demonstrates the average speedup achieved by employing each of the four selected prefetchers in a system with our proposed L1-I configuration, compared to the baseline system. As compared to the 32 KB 8-way, decreasing the L1-I associativity will not necessarily lead to considerable performance loss, with an average of 0.8 % and the worst-case scenario of 1.5 %.

Figure 7 showcases the energy consumption overhead of employing each of the selected L1-I prefetchers in a system with our proposed L1-I configuration in comparison to the baseline system. As shown in the figure, there is a substantial

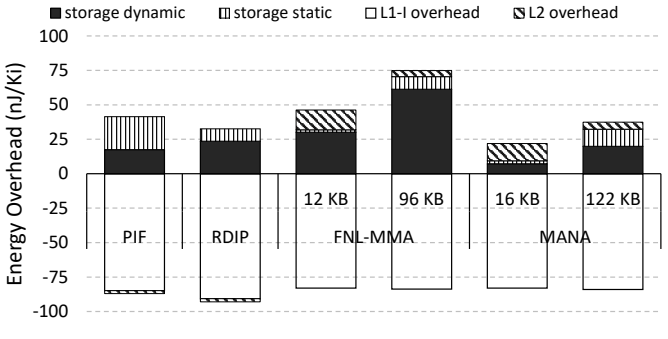

Fig. 7. Energy overhead of 4 selected prefetchers (32 KB, 2-way L1-I)

decrease in L1-I energy consumption ($-84.9\ nJ/ki$ on average) among all of the selected L1-I prefetchers compared to the baseline system. This amount of energy reduction leads to saving a significant amount of energy ($28.8 - (-84.9) = 113.7\ nJ/ki$ on average), in comparison to the 32 KB 8-way configuration results (which is demonstrated in the previous section.) Therefore, the energy savings from reducing the associativity of the L1-I cache are sufficient to offset the energy consumption overhead of the L1-I prefetcher (including L1-I prefetcher energy and L2 energy overhead.), with the average and the highest overhead observed at $41.6\ nJ/ki$ and $74.8\ nJ/ki$, respectively. Moreover, there is a modest increase in L2 energy overhead ($5.9\ nJ/ki$ on average) compared to the 32 KB 8-way configuration. This implies that there is slightly more pressure on L2 as a result of the lower associativity in L1-I. However, the amount of increase in L2 energy overhead is negligible compared to the amount of the reduction in L1-I energy consumption.

TABLE II
MANA FRONTENDS

	Lookahead	SAB count	SAB size	SRQ size
Aggressive	5	3	12	18
Semi-conservative	3	3	12	18
Conservative	3	1	5	8

D. L1-I Prefetcher Front-end Sensitivity Analysis

To address the energy overhead of using the L1-I prefetcher, there are two potential approaches. First, we can consider modifying the L1-I configuration, as discussed in the previous sections. Secondly, adjusting the configurations of the L1-I prefetcher can also help mitigate the impact of energy overhead on the baseline system. In this section, we investigate the effects of changing the front-end configuration of one of the selected prefetchers (MANA [5]) on both performance and energy consumption. To do so, we conduct a detailed analysis of the front-end components of MANA, which includes the SAB (Sequential Access Buffer), SAB count, SRQ (Sequential Request Queue), and lookahead. We propose three distinct configurations for the MANA front-end: Aggressive, Semi-conservative, and Conservative. Each of these configurations are defined in Table II for clarity and reference.

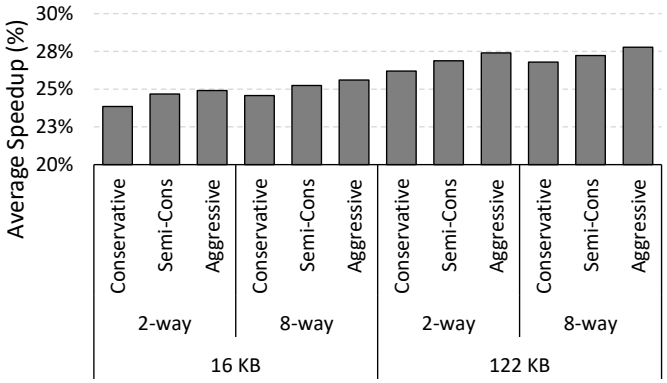

Fig. 8. Average speedup sensitivity analysis for MANA prefetcher

It can be inferred from the table that by transitioning from the Aggressive configuration to the Conservative configuration, the decrease in the lookahead and size of the prefetching buffers signifies a shift from a performance-optimized configuration to an energy-optimized configuration. In this experiment, our focus is to analyze the speedup and energy consumption overhead achieved by employing various front-end configurations of MANA. We compare the results obtained from both MANA 16 KB and MANA 122 KB configurations, in combination with L1-I caches of 32 KB and 2-way as well as 32 KB and 8-way. Figures 8 and 9 present the speedup and energy consumption overhead achieved by the different front-end configurations compared to the baseline system.

Figure 8 demonstrates that by transitioning MANA front-end from conservative to aggressive and increasing the L1-I associativity as well as the L1-I prefetcher's metadata size, there is a continuous increase in performance. Moreover, the performance gap between the aggressive and semi-conservative approaches remains negligible, averaging at only 0.4%. In addition, figure 9 illustrates that switching MANA front-end from conservative to aggressive leads to a considerable increase in the L2 energy consumption, averaging at $5.9\ nJ/ki$. The primary reason for the increase in L2 energy consumption is the higher number of accesses to L2 caused by the larger lookahead of the prefetcher in the aggressive configuration. Additionally, it is evident that increasing the metadata size results in a higher energy consumption overhead for the L1-I prefetcher, including both static and dynamic energy consumption.

Among the three evaluated configurations of MANA, the semi-conservative configuration exhibits an average of $25.6\ nJ/ki$ less overall energy consumption overhead compared to the other two configurations. The overall energy consumption overhead encompasses the energy consumption of L1-I, L2, and L1-I prefetcher, including both static and dynamic energy consumption. Therefore, the semi-conservative configuration of MANA front-end appears to effectively address the energy consumption overhead caused by the L1-I

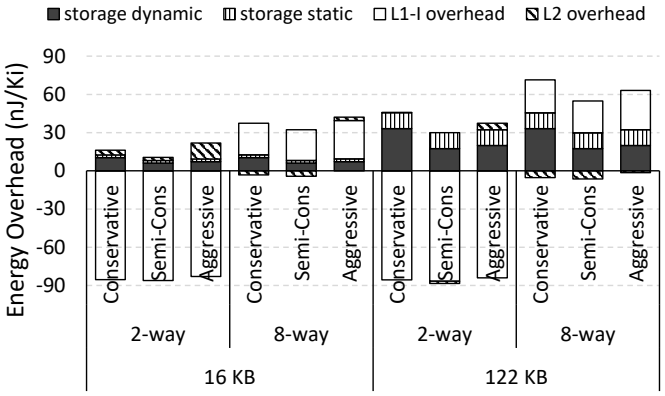

Fig. 9. Energy consumption sensitivity analysis for MANA prefetcher

prefetcher over the baseline system. Finally, it can be concluded from figure 9 that by combining the semi-conservative configuration with lower associativity in L1-I (2-way), we can achieve even greater energy savings.

IV. Discussion

This section delves into two aspects of our study's findings and their implications, focusing on the key topics raised during our evaluation.

Compatibility with Virtual Memory (VIPT): Changing the associativity of a 32 KB L1-I cache from 8-way to 2-way leads to a $4\times$ expansion in set size. This adjustment mandates the increase in page size from 4 KB to 16 KB to maintain the existing virtual address translation speed while employing a Virtually Indexed Physically Tagged Cache (VIPT), incorporating extra tag bits to cover the expanded sets. Moreover, it's common to employ larger page sizes, such as 16 KB, in modern operating systems like Apple's M-series [18]. However, it's important to note that employing larger page sizes can potentially increase vulnerability to internal fragmentation in some workloads [20] [17]. As another approach to address this issue, we noted that a 2-way L1-I cache experiences lower access latency than its 8-way counterpart, potentially reducing the necessity of relying solely on VIPT to maintain virtual address translation speed.

Comparing the Energy Savings of Our Approach vs. the Way-Prediction Technique [19]: Both methodologies aim to minimize way probes during cache set lookups to enhance energy efficiency. First, we observed that the 2-way L1-I cache incurs considerably less area overhead than the 8-way cache. While the 2-way L1-I cache consumes considerably less area overhead than the 8-way cache, implementing a way-prediction technique introduces additional overhead. Additionally, our approach involves lowering energy consumption by reducing associativity and leveraging a strong prefetcher. In contrast, the way-prediction technique anticipates future access ways, leading to fewer way probes in a set. The way-prediction technique achieves, on average, 70% reduction in ED (average energy consumption per cache access $\times$ average energy consumption per cache access), whereas our approach ensures at least a twofold decrease in energy per access for L1-I cache. Notably, the way-prediction method achieves 90% hit rate in specific scenarios, indicating that its prediction accuracy falls short in 10% of lookups [19]. In conclusion, these two approaches share a common goal and can complement each other to achieve enhanced performance and energy savings.

V. Background

In this section, we review four instruction prefetching methods for servers proposed in the last decade, which we selected for our evaluations.

Proactive Instruction Fetch (PIF) [2], a temporal instruction access-based prefetcher, provides a more substantial enhancement, compared to its predecessors [7]. Temporal prefetchers are based on repetitive behavior in the sequences of instruction cache accesses or misses. By recording and replaying the sequence of prior cache accesses [2] or misses [7], temporal prefetchers succeed in eliminating future instruction cache misses. PIF records the sequence of spatial regions in a circular history buffer. Although PIF effectively minimizes cache misses, it imposes substantial storage overhead on the processor, i.e., 236 KB per core.

RAS Directed Instruction Prefetcher (RDIP) [1] attempts to offer a significantly lower storage cost compared to PIF without sacrificing performance. The designers of RDIP note that the current state of the return-address-stack (RAS) can provide a distinct and precise representation of the program's state. To exploit this observation, RDIP XORs the four top entries of the RAS and calls it a signature. Then it assigns the observed instruction cache misses to the corresponding signature and stores them in a set associative table. Using the unique signature, RDIP retrieves miss patterns from the table. RDIP achieves a significant reduction in per-core storage, lowering it to over 60% of the storage required by PIF. While RDIP requires considerably lower storage as compared to PIF, its required storage budget is still considerable.

Microarchitecting an Instruction Prefetcher (MANA) [6] is another L1-I prefetching technique that leverages temporal correlation to achieve its benefits. By exploiting the temporal correlation, this prefetcher offers a smaller number of distinct records as well as compact metadata records in the prefetching-related tables. Moreover, MANA stores high-order bits of tag in a set associative table named high-order-bits patterns' table (HOBPT). In addition, to provide a space-efficient method and take advantage of temporal correlation among spatial records, MANA proposed the idea of chaining spatial metadata records, which yields a robust and timely prefetching method.

FNL+MMA [4] is another instruction cache prefetcher that monitors only demand misses and stores them in a small tag-only cache named I-Shadow. This method uses a simple next-line prefetcher to predict the near future accesses. To prevent the main issue of next-line prefetching, i.e., over-fetching and subsequent cache pollution, the FNL+MMA method exploits the Footprint Next-line prefetcher (FNL). The FNL incorporates a filtering mechanism to anticipate

"imminent" future accesses, the data accesses that will be required in the immediate or near future.

MANA and FNL-MMA further enhance their proposed methods with a storage-efficient format that minimizes the size of the prefetcher's metadata while maintaining a high level of performance.

VI. Conclusion

A significant performance bottleneck in modern server working sets is frequent L1-I cache misses. Over the past two decades, researchers presented numerous L1-I prefetching techniques to tackle the issue of L1-I cache misses. While previous works focused on performance enhancement and minimizing area overhead of L1-I prefetchers, researchers gave little attention to the resulting increase in energy consumption from incorporating an L1-I prefetcher. In this work, we proposed a survey on the energy consumption and performance of four selected prefetchers. Moreover, we showed that a significant portion of the energy consumption associated with L1-I prefetchers is due to the increased L1-I accesses initiated by the prefetcher. To mitigate this energy consumption, we proposed an approach to reduce the energy per access by decreasing the L1-I associativity. Our observations showed that reducing the L1-I associativity from 8 to 2 effectively mitigates energy consumption, on average $113.7\ nJ/ki$. The achieved energy savings compensate for the energy consumption overhead imposed by the L1-I prefetcher over the baseline system, with the average and the highest energy overhead at $41.6\ nJ/ki$ and $74.8\ nJ/ki$, respectively. Additionally, the associated performance loss remains negligible, with an average of 0.8%. We also investigated the impact of altering the front-end configuration of an L1-I prefetcher on performance and energy consumption. Our results demonstrated that using the energy-optimized front-end for MANA yields a substantial reduction of $25.6\ nJ/ki$ in energy consumption (including L1-I, L2, and L1-I prefetcher storage), with a minimal performance loss of less than 1%.

References

[1] A. Kolli, A. Saidi and T. F. Wenisch, *RDIP: Return-address-stack Directed Instruction Prefetching,* 2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Davis, CA, 2013, pp. 260-271.

[2] Michael Ferdman, Cansu Kaynak, and Babak Falsafi, *Proactive instruction fetch,* 2011 In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44). Association for Computing Machinery, New York, NY, USA, 152–162. DOI:https://doi.org/10.1145/2155620.2155638.

[3] Rakesh Kumar, Boris Grot, and Vijay Nagarajan, *Blasting through the Front-End Bottleneck with Shotgun,* 2018 In Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18). Association for Computing Machinery, New York, NY, USA, 30–42. DOI:https://doi.org/10.1145/3173162.3173178.

[4] André Seznec, *The FNL+MMA Instruction Cache Prefetcher,* IPC-1 - First Instruction Prefetching Championship, May 2020, Valence, Spain.

[5] Ansari A, Golshan F, Lotfi-Kamran P, Sarbazi-Azad H, *MANA: Microarchitecting an instruction prefetcher,* IPC-1 - First Instruction Prefetching Championship, May 2020, Valence, Spain.

[6] A. Ansari, F. Golshan, P. Lotfi-Kamran and H. Sarbazi-Azad, *MANA: Microarchitecting an Instruction Prefetcher,* in IEEE Transactions on Computers, vol. 72, no. 3, pp. 732-743, 1 March 2023, doi: 10.1109/TC.2022.3176825.

[7] M. Ferdman, T. F. Wenisch, A. Ailamaki, B. Falsafi and A. Moshovos, *Temporal instruction fetch streaming,* 2008 41st IEEE/ACM International Symposium on Microarchitecture, Lake Como, 2008, pp. 1-10, doi: 10.1109/MICRO.2008.4771774.

[8] N. Gober, G. Chacon, L. Wang, Paul V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, *The Championship Simulator: Architectural Simulation for Education and Competition*, https://doi.org/10.48550/arXiv.2210.14324 arXiv-issued DOI via DataCite, Oct 2022.

[9] "IPC-1," https://research.ece.ncsu.edu/ipc/.

[10] L. Spracklen, Yuan Chou and S. G. Abraham, *Effective instruction prefetching in chip multiprocessors for modern commercial applications,* 11th International Symposium on High-Performance Computer Architecture, San Francisco, CA, USA, 2005, pp. 225-236, doi: 10.1109/HPCA.2005.13.

[11] C. Kaynak, B. Grot, and B. Falsafi, *SHIFT: Shared History Instruction Fetch for Lean-core Server Processors,* in Proceedings of the 46th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO), Dec. 2013, pp. 272–283.

[12] Jinchun Kim, Seth H Pugsley, Paul V Gratz, AL Narasimha Reddy, Chris Wilkerson, and Zeshan Chishti, *Path confidence based lookahead prefetching,* In 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 1–12.

[13] David Tarjan and Kevin Skadron, *Merging path and gshare indexing in perceptron branch prediction,* ACM transactions on architecture and code optimization (TACO) 2, 3 (2005), 280–300.

[14] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas, *CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories,* ACM Trans. Archit. Code Optim. 14, 2, Article 14 (July 2017), 25 pages. DOI:https://doi.org/10.1145/3085572.

[15] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam and D. Burger, "Dark silicon and the end of multicore scaling," 2011 38th Annual International Symposium on Computer Architecture (ISCA), San Jose, CA, USA, 2011, pp. 365-376.

[16] Imani HamidReza, Anderson Jeff, El-Ghazawi Tarek, "iSample: Intelligent Client Sampling in Federated Learning", 2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC), pp. 58-65.

[17] A. Shekar, M. Baradaran, S. Tajdari, and K. Skadron, "HashMem: PIM-based Hashmap Accelerator," arXiv:2306.17721 [cs.AR], 2023.

[18] https://www.wwdcnotes.com/notes/wwdc20/10214/

[19] K. Inoue, T. Ishihara and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," Proceedings. 1999 International Symposium on Low Power Electronics and Design (Cat. No.99TH8477), San Diego, CA, USA, 1999, pp. 273-275, doi: 10.1145/313817.313948.

[20] Ted G. Lewis, Brian J. Smith, and Marilyn Z. Smith. 1974. Dynamic memory allocation systems for minimizing internal fragmentation. In Proceedings of the 1974 annual ACM conference - Volume 2 (ACM '74). Association for Computing Machinery, New York, NY, USA, 725–728.