Planaria: Pattern Directed Cross-page Composite Prefetcher

Yuhang Liu, Mingyu Chen
Institute of Computing Technology, Chinese Academy of Sciences,
State Key Laboratory of Processor Chips, China
Beijing, China

ABSTRACT

Given the memory wall, the performance of the memory system significantly influences the user experience of mobile phones. The system cache (SC), located on the memory side, is shared among all CPUs and GPUs within the mobile phone, serving as the last line of defense before resorting to time-consuming off-chip memory access. Managing SC is a challenge due to its large working set and irregular access patterns. Despite occupying a substantial on-chip area, SC's effectiveness in terms of hit rate is relatively low. It has been observed that neither state-of-the-art cache replacement policies nor increasing cache size significantly improve SC performance. Prefetchers designed for higher-level caches cannot be seamlessly applied to SC due to the absence of the required program counter (PC) on the memory side and the violation of stringent power constraints in mobile phones by aggressive prefetch traffic.

In this study, we present Planaria, comprising two sub-prefetchers (SLP and TLP) and a coordinator, aiming to achieve high accuracy and coverage simultaneously. The two sub-prefetchers exploit intra- and inter-page regularities through self and transfer learning, respectively. The coordinator explicitly decouples the learning and issuing phases of the sub-prefetchers. Experimental results demonstrate that Planaria has enhanced overall system performance in terms of instructions per cycle (IPC) by an average of 28.9%, 21.9%, and 15.3% over no prefetcher, BOP, and SPP, respectively. Moreover, Planaria proves to be power-efficient, incurring only a 0.5% increase in power consumption, while BOP and SPP increase power consumption by 13.5% and 9.7%, respectively.

1 INTRODUCTION

As mobile phone applications are increasingly memory-intensive, the efficiency of memory system significantly affects the overall performance and thus the user experience of a mobile phone system. Mobile phones exhibit a notably higher average memory access time (AMAT) compared to desktops and servers, primarily due to their use of low-power DDR (LPDDR) memory for energy conservation [4].

As the lowest level cache in the memory hierarchy, system cache (SC) serves as the last defense line before resorting to the time-consuming off-chip memory access. It plays a crucial role in mitigating memory access latency and power consumption. However, the existing performance of SC is far lower than the expectations of the designers. The reason is that the high-level caches have already filtered much spatial and temporal locality, leaving SC few opportunities to exploit. Consequently, the highly irregular accessing patterns cause the low hit rate of SC and complicate the prefetchers' prediction of the address for future access. To validate this analysis, we evaluated the state-of-the-art prefetchers SPP [6] and BOP [8], which rely on identifying regularities from the accessing sequence and predicting the next accessing address. Our experimental results indicate that SPP and BOP only achieve a modest reduction in average memory access time (AMAT) by 10.8% and 3.3%, respectively. Moreover, both prefetchers incur additional memory traffic overhead, with SPP at 15.9% and BOP at 23.4%.

To enhance the performance of the system cache (SC) without introducing excessive additional memory traffic, it is essential to design a dedicated hardware prefetcher. Through memory trace analysis, we identified two distinct types of accessing regularities at the system cache level that can be leveraged for optimization: intra-page regularity and inter-page regularity. Here, intra-page regularity represents that a group of blocks within a memory page (referred to as the footprint snapshot) are accessed repeatedly during a certain time interval. Inter-page regularity represents that memory pages close in address space usually have similar footprint snapshots and thus can learn from each other.

In this study, we propose a hardware prefetcher, Planaria, which includes two sub-prefetchers and a coordinator. The two sub-prefetchers are designed to utilize intra- and inter-page regularities. Specifically, the intra-page sub-prefetcher issues prefetch requests for a memory page by referencing the history information of the page itself. The inter-page sub-prefetcher allows the memory pages close in address space to share their patterns. To coordinate the operation of sub-prefetchers, the coordinator decouples the hardware prefetching as a "pattern learning + prefetch issuing" process and manages the learning and issuing operations separately.

In this paper, we make the following main contributions:

❶ We propose a self-learning driven sub-prefetcher (SLP) to handle intra-page accessing patterns.

❷ We propose a transfer-learning driven sub-prefetcher (TLP) to handle inter-page accessing patterns. TLP increases the prefetch coverage by referencing the patterns of neighboring pages.

❸ We propose Planaria, which integrates the SLP and TLP. Experimental results show that on the targeted mobile phone applications, Planaria has reduced AMAT over no prefetcher and the state-of-the-art prefetchers, BOP and SPP by 24.3%, 21.3% and 15.1%, respectively. While BOP and SPP increase the power consumption of memory systems by 13.5% and 9.7%, Planaria only incurs 0.5% extra power consumption.

The rest of this paper is organized as follows. Section 2 presents the skeleton of Planaria. Section 3 and Section 4 illustrate the design of the intra- and inter-page sub-prefetchers, respectively. Section 5

DAC'24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3656499

provides details about the experimental setup. Section 6 presents the evaluation results and analysis. Section 7 overviews the related work. Finally, Section 8 concludes our study.

2 THE SKELETON OF PLANARIA

Planaria has two key components, including two sub-prefetchers (i.e., Self-Learning directed Prefetcher (SLP) and Transfer Learning directed Prefetcher (TLP)). SLP exploits the intra-page spatial locality, which will be introduced in Section 3. TLP utilizes the similarity of accessing patterns of neighboring memory pages, which will be introduced in Section 4.

Figure 1 shows the structure and operation of Planaria. Planaria operates by taking demand requests as input and producing prefetch requests as output. First, Planaria transfers the address and arrival time of the current demand request to each sub-prefetchers and launches their learning phases. Second, the learning phase learns the memory accessing pattern and delivers the meta-data to the issuing phase. Meta-data comprises the page number and the bitmap-pattern representing which blocks should be prefetched in that memory page. Planaria selects only one sub-prefetcher through a specific rule to enable its issuing phase. The selection rule enables the SLP-issuing preferentially and enables TLP-issuing only when SLP does not have history information to support generating prefetching requests. The selected sub-prefetcher executes its issuing phase to calculate the prefetching address. Finally, the generated prefetch requests are inserted into the prefetch queue.

The coordination framework of Planaria has harvested the benefits and overcome the shortcomings of prior coordination mechanisms. Prior proposals [3, 5, 7, 11], integrate different sub-prefetchers, where each sub-prefetcher is monolithic. This monolithic structure means that the *learning* and *issuing* phases of each sub-prefetcher were indivisible. One key insight of our design is that, the learning phase and the issuing phase of each sub-prefetcher need to be decoupled and managed separately. In the learning phase, each sub-prefetcher learns the pattern from all the memory accesses and provides the meta-data for calculating prefetch addresses. In the issuing phase, each sub-prefetcher generates prefetch requests according to the meta-data delivered by the learning phase. Specifically, the learning phase of all the sub-prefetchers should be enabled simultaneously, allowing each sub-prefetcher to observe the complete memory accessing sequence and thus it is full-pattern directed. Conversely, the issuing phase of the sub-prefetchers is enabled selectively to balance the prefetching accuracy and coverage.

3 INTRA-PAGE SUB-PREFETCHER

3.1 Observation of Intra-page Accessing Pattern

Observation 1: Spatial locality is more dominant than temporal locality for the intra-page memory accessing pattern of system cache level.

Figure 2 shows the characteristics of the intra-page accessing pattern. The X-axis presents the arrival time (cycle) of the accesses, and the Y-axis is the block number in a memory page, which is from 0 to 63 when the size of a page is 4KB and the size of a block is 64B. As shown in Figure 2, we can get the following characteristics.

❶ Several different blocks in a memory page are accessed during a brief time interval, highlighting distinctive spatial locality. If these accessed blocks are regarded as a snapshot, the constituent and structure of the snapshot are stable. The stability will be validated by the experiment introduced in Figure 4. ❷ The reuse distance of the snapshots is usually long, indicating a limited temporal locality. ❸ The access order of these blocks is non-deterministic, resulting in a highly unpredictable sequence of deltas.

3.2 Self-Learning Directed Prefetcher

According to observation 1, we propose an intra-page prefetcher: SLP (an acronym for Self-Learning directed Prefetcher) to exploit the spatial locality of an intra-page pattern. SLP records the footprint snapshot for memory pages that have been accessed recently and prefetches all the other blocks in a footprint snapshot whenever any blocks of the snapshot have been accessed.

Different from previous researches [1, 16], which map the bitmap-pattern to the signature comprising the program counter (PC) and the trigger access, SLP exploits PN as the signature to index the footprint snapshot. This design is motivated by the expensive cost of obtaining a PC at the low level of the memory hierarchy and the difficulty of distinguishing different PCs from numerous devices in a heterogeneous computer system. While using only the page number (PN) as a signature might potentially reduce prefetch accuracy if the access pattern of a memory page changes frequently during program phase switches, our quantitative experiments reveal minimal alterations in the footprint snapshot; that is, the constituent and structure of the snapshot remain stable.

We conducted an experiment to quantitatively validate the similarity of footprint snapshots across different program phases. The methodology is illustrated in Figure 3. Initially, we determined the window size by counting the number of accessed blocks in a page. Subsequently, for each page, we recorded the blocks accessed within a window and compared them with those of the preceding window. The overlap rate was employed as a metric for the similarity between two time windows. This rate was calculated by dividing the number of blocks accessed in both the current and previous windows by the total number of blocks accessed in the current window. We analyzed the memory traces of all the targeted applications and calculated the average overlap rate of the windows for all pages. The results are shown in Figure 4. The average overlap rate of the applications is more than 80%. As a qualitative experiment, the results are strong enough to support that the snapshot change is limited during the switch of program phases. Therefore, using the page number as the signature of a footprint snapshot is feasible and accurate.

Figure 1 illustrates the structure and operation of SLP, where page number (PN) is used as the index of the tables. The DRAM comprises four channels, each with a respective SC. A 4KB memory page is partitioned into four segments, each consisting of 16 blocks, and each segment is statically mapped to a specific DRAM channel. Consequently, the prefetcher of the SC in each channel utilizes a 16-bit bitmap pattern to track whether the corresponding block has been accessed. SLP uses the *last access time* field and time-out mechanism to evict the outdated entries. Upon the arrival of a demand access, the accumulation table (AT) will check whether some blocks of the same page have been accessed (Step ❶). If AT fails to find a matching entry, the request is inserted into the filter

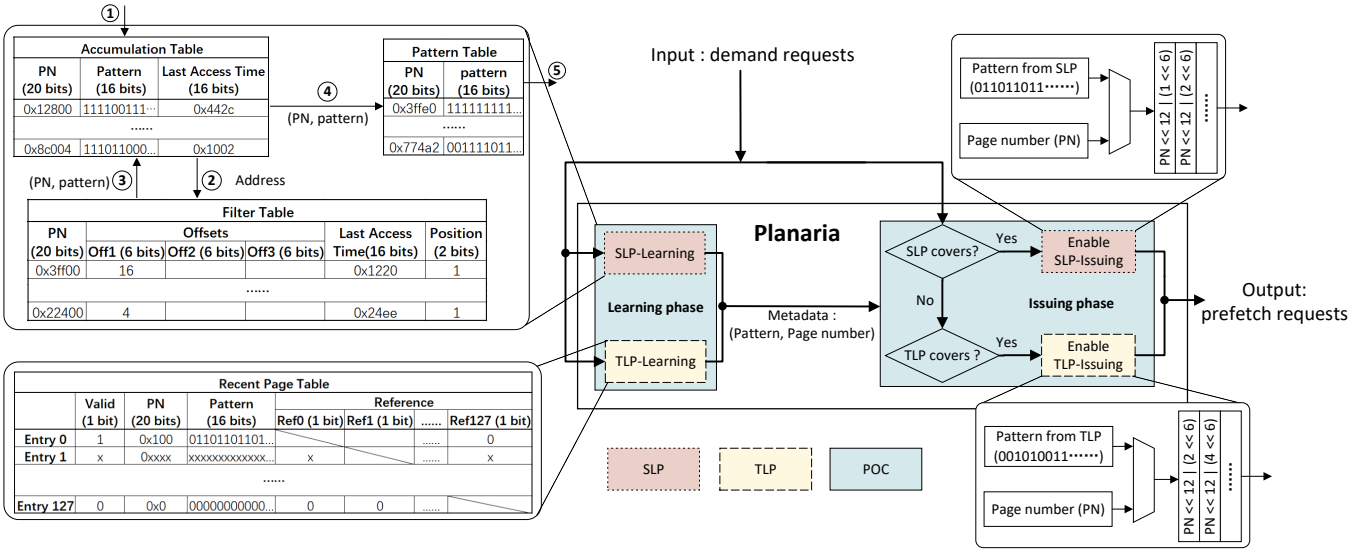

Figure 1: The structure of Planaria.

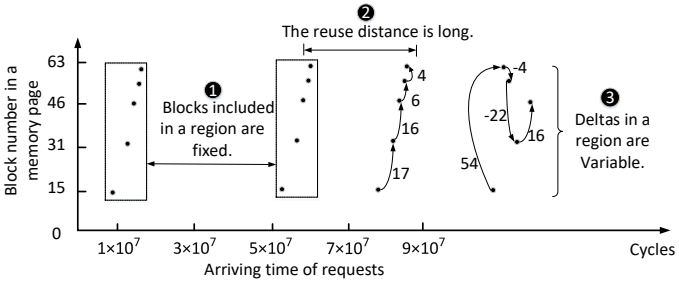

Figure 2: The footprint snapshot of a memory page.

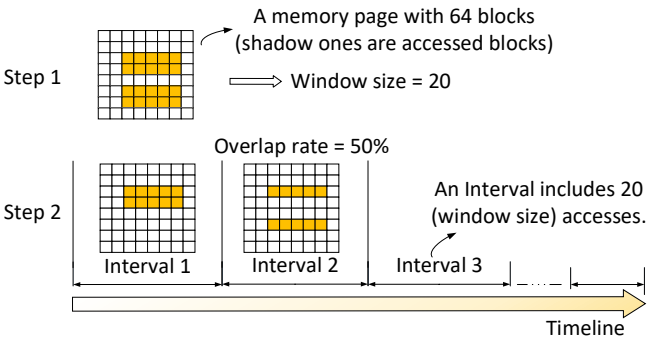

Figure 3: The method for calculating the overlap rate.

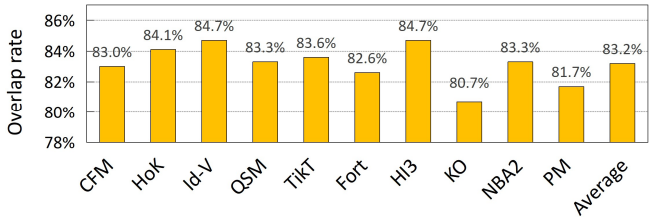

Figure 4: The overlap rate of different applications.

table (FT) (Step ❷). FT is responsible for filtering snapshots that include too few blocks. Once an FT entry has recorded three offsets (three demand requests), the request is then inserted into AT (Step ❸). When an AT entry is evicted due to time-out, SLP interprets this as the detection of a complete and stable snapshot, and the recorded bitmap pattern is transferred to the Pattern History Table (PT) (Step ❹). Based on PT, prefetch requests will be generated if the demand request is a cache miss (Step ❺).

4 INTER-PAGE SUB-PREFETCHER

4.1 Observation of Inter-page Accessing Pattern

Observation 2: Significant fractions of memory pages can learn memory accessing pattern from their neighboring pages on system cache level.

We conducted an experiment to quantitatively validate observation 2. First, we represent the accessing pattern with a bitmap for every memory page, using "1" and "0" to denote whether a given block in the page has been accessed or not. Subsequently, if the difference between the bitmap of two pages is small (e.g., under a threshold, 4 bits) and the two pages are close in address space (i.e., the difference between page numbers is under the *distance threshold*, the two pages will be regarded as learnable neighbors. The fraction of memory pages exhibiting this neighboring property for various distance thresholds is depicted in Figure 5. The results show that on average 26.95% (or 39.26%) of memory pages have neighboring property when the distance threshold is set as 4 (or 64).

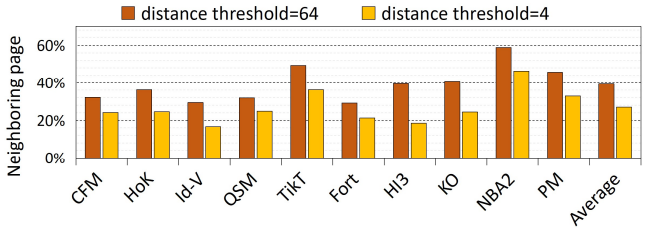

Figure 5: The proportion of learnable neighboring pages of the targeted applications.

4.2 Transfer-Learning Directed Prefetcher

To exploit the inter-page regularity, we design an inter-page prefetcher: TLP (an acronym for Transfer-Learning directed Prefetcher) based on observation 2. Figure 6 illustrates the fundamental concept of TLP. Memory page A does not have self-learned metadata to generate prefetch requests due to the lack of the history information. Instead, it has two candidate neighboring pages to learn from: page B and C, because the difference of their page numbers is under the threshold (i.e., 64). The common pattern of page A and B involves 6 blocks, whereas the common pattern of page A and C involves 3 blocks. Therefore, page B, rather than C, exhibits greater similarity to page A in terms of memory access patterns. Thus, page A utilizes the pattern learned from page B to generate prefetches. On page A, the blocks indicated by the learned pattern but not yet accessed will be prefetched.

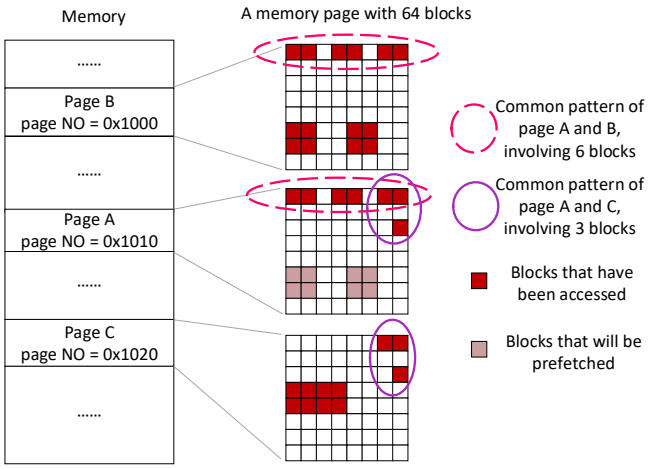

Figure 6: The diagram of the TLP sub-prefetcher.

As shown in Figure 1, the major component of TLP is the Recent Page Table (RPT) which has 128 entries. RPT is indexed by page number and each entry contains a 16-bit bitmap to record whether the blocks have been accessed recently. To effectively check whether a page can refer to another page, 128 1-bit "Ref" fields are allocated in each entry. If the difference between the page number of entry *i* and entry *j* is larger than a threshold, the *j*th "Ref" in entry *i* and the *i*th "Ref" in entry *j* are set as "1"; otherwise "0". If the size of RPT is *N*, then each entry will have *N-1* useful "Ref" items, because referring to a page itself is meaningless.

Table 1: Experimental setup configurations.

Memory trace-generating system configurations		
Mobile phone	**CPUs**	8 cores, ARM 2 Cortex-A76, 2.6 GHz, 512KB L2 2 Cortex-A76, 1.92 GHz, 512KB L2 4 Cortex-A55, 1.8 GHz, 128KB L2 Three-level caches, 4MB L3, with CPU-side hardware prefetchers
	GPUs	ARM Mali-G76MP10, 720 MHz, Two-level caches
	Others	NPU, ISP, DSP
	Process	7nm
Memory system configurations		
Simulator	**System Cache**	4 MB, 16-way per set 64B per cache block
	DRAM	4 channels, 1 rank per channel, 8 banks per channel, LPDDR 4 tRAS=51, tRCD=16, tRRD=12, tRC=76, tRP=16, tCCD=8, tRTP=9, tWTR=12, tWR=22, tRTRS=2, tRFC=216, tFAW=48, tCKE=9, tXP=9, tCMD=1, Burst Length=16, DDR Frequency=1.2GHz, FR-FCFS

Figure 1 illustrates the structure and operation of TLP. Initially, RPT is empty. When page $0x100$ is accessed, TLP allocates a new entry (entry 0) in RPT. As other RPT entries are invalid at present, all "Ref" fields in entry 0 are set as "0". Then, page $0x110$ is accessed. TLP allocates a new entry (entry 2) and sets Ref0 as "1" because page $0x100$ and page $0x110$ are neighboring pages in space. After several memory accesses, page $0x110$ is accessed again. TLP finds the Ref0 of entry 2 is "1" and the bitmaps of entry 0 and entry 2 have four same bits. In this case, TLP predicts that page $0x100$ and page $0x110$ have a similar memory accessing pattern. Comparing the bitmaps of entry 0 and entry 2, if a bit in entry 0 is "1" but in entry 2 is "0", TLP will prefetch the block on page $0x110$.

5 EVALUATION METHODOLOGY

Table 1 displays the architecture parameters of the baseline system. We prefer "physical running to collect real traces + trace-driven simulation + RTL performance evaluation" to full-system simulation. Table 2 shows the targeted applications which run on mobile phones and are in the top10 list of a region [9, 10]. We modified DRAMSim2 [12] to evaluate different candidate architectures, including LPDDR chips. DRAMSim2 reads memory traces as input, simulates the progress of processing these memory requests and output the statistics of performance. We implemented and synthesized the baseline system and Planaria using Verilog to further evaluate the overall performance and estimate the area overhead.

We implemented a hardware module for monitoring the memory bus within the mobile phone to record the memory access trace. Each trace entry includes the physical access address, the access type (i.e., read or write), the request device ID (i.e., CPU, GPU, DSP, etc.) and the access arrival time. The memory traces are real records of physical mobile phones, which can accurately reflect the impacts of the deep memory hierarchy, the complex software stack and the user activities on memory accesses. The power model

Table 2: The targeted representative applications.

Workloads	Description	Length (M)	Abbr.
Cross Fire Mobile	First-person shooter	67.48	CFM
Honor of Kings	Multiplayer MOBA	71.37	HoK
Identity V	Asymmetric battle arena	68.27	Id-V
QQ Speed Mobile	3D racing mobile game	69.45	QSM
TikTok	Short video sharing app	70.82	TikT
Fortnite	Multiplayer battle royale	66.71	Fort
Honkai Impact 3	3D action game	67.65	HI3
Knives Out	Multiplayer battle royale	68.00	KO
NBA 2K19	Basketball game	67.71	NBA2
PUBG Mobile	Multiplayer battle royale	67.71	PM

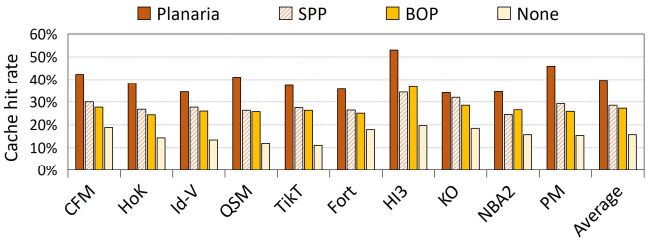

Figure 7: Hit rate of SC with different prefetchers.

provided by the manufacturers of mobile phones is embedded into our simulator.

6 RESULTS AND ANALYSIS

This section shows and analyses the experimental results.

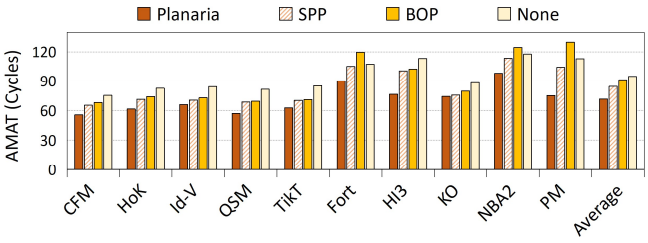

Figure 8: AMAT of the memory system with different prefetchers.

Figure 7 and 8 display the performance improvement of the targeted applications in terms of AMAT and the hit rate of SC, respectively. Planaria has reduced AMAT by 24.3%, 21.3%, and 15.1% over no prefetcher, BOP, and SPP, respectively. In the majority of instances, an increase in the cache hit rate corresponds to a decrease in Average Memory Access Time (AMAT). Nevertheless, in the cases of applications Fort, NBA2, and PM, the utilization of BOP augments the cache hit rate while elevating AMAT compared to scenarios devoid of prefetchers. We evaluated the time of each stage for individual access and found that the discerned rationale behind BOP's suboptimal performance in the contexts of Fort, NBA2, and PM lies in its generation of numerous superfluous prefetches, thereby engendering a substantial augmentation in memory traffic. Conversely, Planaria increases the cache hit rate without much extra memory traffic, which indicates higher accuracy.

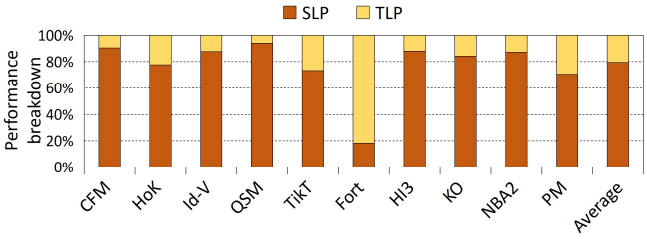

Figure 9: Planaria performance breakdown.

Figure 9 illustrates the breakdown of the performance improvement of Planaria. SLP contributes to nearly 80% of overall improvement. For applications CFM, QSM, HI3, KO and NBA2, the effect of TLP is limited. Patterns existing in CFM, QSM, HI3, KO and NBA2 are what SLP excels at. The limitation of TLP for those applications is due to the lack of opportunities to issue prefetches. Especially, TLP contributes to most of the improvement for Fort. When SLP frequently fails to issue prefetches, TLP which is the low priority sub-prefetcher gets a chance to issue prefetches. That is, TLP contributes more to the overall improvement at this time.

Planaria incurs an average additional power consumption of merely 0.5%, whereas BOP and SPP result in 13.5% and 9.7%, respectively. For each application, Figure 10 displays the power consumption of memory system when SC employing different prefetchers. Planaria increases the memory system's power consumption by 0.2% to 2.8%. For the applications HI3 and PM, Planaria even reduces the power consumption of memory system by 3.3% and 1.2%, respectively. The storage of Planaria is 345.2KB, which is only 8.4% of the capacity of 4MB SC.

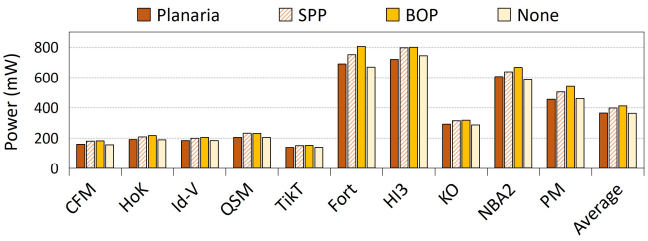

Figure 10: Power consumption of the memory system with different prefetchers.

7 RELATED WORK

We review the related work of the three components of Planaria, respectively. Intra-page prefetchers can be classified into delta-based prefetchers, spatial prefetchers and content-directed prefetchers. Delta-based prefetchers [6, 8, 15] learn the pattern of the delta history to predict future deltas and add current access address and delta to calculate the prefetch address. Due to the filtering effect of higher-level caches and out-of-order scheduling in the core and the

cache/memory sub-systems, it is difficult to detect a regular delta sequence on SC. Section 6 has validated that the typical delta-based prefetchers cannot perform well for SC.

Spatial prefetchers exploit spatial locality by online [1, 16] or offline profiling [17] or compiler hints [18]. Spatial prefetchers mainly rely on a PC to exploit the correlations between instruction flow and data flow. However, it is expensive to transfer the PC from multiple cores to low-level cache [13]. We proposed a practical prefetcher (i.e., SLP) that does not need a PC to address this issue.

Content-directed prefetchers [2, 20] track the delicate relationship between "data flow" and "address flow", the completeness of which has been undermined by the filtering effect of the high-level caches. As it is difficult to detect indirect accessing patterns for SC, we cannot apply a content-directed prefetcher to SC.

Existing inter-page prefetchers [6, 14] try to launch the prefetching without relearning any delta history patterns or doing any other warmup in the new memory page. However, making a prediction based on small global history tables shared by all pages would incur many mispredictions, which cannot be tolerated by the strict power consumption constraints of mobile phones and also causes cache pollution. To improve the prefetching accuracy, our study proposes more restrictions on inter-page prefetching, i.e., comparing the difference of page numbers and bitmap-patterns between two pages. Another difference between our scheme and prior studies is that, our scheme focuses on learning the spatial bitmap-pattern rather than the delta-based pattern of a memory page, and then conditionally applies it to another page.

The coordinator of existing hybrid prefetchers can be divided into two categories: serial coordinators and parallel coordinators. Serial coordinators enable the sub-prefetchers in priority order. TPC [7] includes three sub-prefetchers and assumes that each sub-prefetcher can recognize the boundary of its expertise. Thus, the serial coordinator of TPC is a hardwired decision logic that enables each sub-prefetcher in turn. Parallel coordinators enable all the sub-prefetchers in parallel. ISB [5] and MISB [19] combine an irregular prefetcher with a regular prefetcher, enabling the irregular prefetcher and regular prefetcher simultaneously to improve the prefetch coverage. Our coordinator is different from the serial or parallel coordinators, but harvests their benefits through "parallel training and serial issuing". It explicitly decouples the learning and issuing phases of the sub-prefetchers and manages them separately to improve the accuracy and coverage simultaneously.

8 CONCLUSION

Hardware prefetching is a pivotal technique for optimizing memory system performance, which impacts significantly the overall computer system efficiency. However, designing a hardware prefetcher that achieves both high performance and power efficiency poses a challenge, considering the specific characteristics of the targeted applications, architectural features, and the stringent power consumption constraints of mobile phones. To tackle this challenge, we leverage the following insights: the dominant regularity of the System Cache (SC) reveals itself as spatial locality rather than temporal locality. Additionally, within the targeted applications, both intra- and inter-page regularities exist concerning spatial locality. We have devised two sub-prefetchers utilizing self-learning and transfer-learning to exploit intra- and inter-page regularities, respectively. Furthermore, we have decoupled prefetching into 'the learning phase and the issuing phase' and introduced a coordinator to seamlessly integrate the two sub-prefetchers.

ACKNOWLEDGMENTS

This work is partially supported by the National Key Research and Development Program of China under Grant No. 2023YFB4503904, and the Innovation Funding of ICT, CAS under Grant No. E361100.

REFERENCES

[1] Rahul Bera, Anant V Nori, Onur Mutlu, and Sreenivas Subramoney. 2019. Dspatch: dual spatial pattern prefetcher. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 531–544.

[2] Robert Cooksey, Stephan Jourdan, and Dirk Grunwald. 2002. A stateless, content-directed data prefetching mechanism. *ACM SIGPLAN Notices*, 37, 10, 279–290.

[3] Eiman Ebrahimi, Onur Mutlu, and Yale N Patt. 2009. Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems. In *proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture*. Raleigh, North Carolina, USA, 7–17.

[4] John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach*. Elsevier.

[5] Akanksha Jain and Calvin Lin. 2013. Linearizing irregular memory accesses for improved correlated prefetching. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 247–259.

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

[7] Sushant Kondguli and Michael Huang. 2018. Division of labor: a more effective approach to prefetching. In *proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. Los Angeles, CA, USA, 83–95.

[8] Pierre Michaud. 2016. Best-offset hardware prefetching. In *proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Barcelona, Spain, 469–480.

[9] Qianfan. [n. d.] Rank of mobile games in China. https://zhishu.analysys.cn/public/view/wTopApp/wTopApp.html. Accessed Nov 4, 2021. ().

[10] Qimai. [n. d.] Rank of games in China. https://www.qimai.cn/rank/marketRank/market/3/category/154/date/2019-11-08. Accessed Nov 4, 2021. ().

[11] Joseph Rogers. 2019. Effects of an lstm composite prefetcher. (2019).

[12] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. Dramsim2: a cycle accurate memory system simulator. *IEEE computer architecture letters*, 10, 1, 16–19.

[13] Subhash Sethumurugan, Jieming Yin, and John Sartori. 2021. Designing a cost-effective cache replacement policy using machine learning. In *proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Seoul, South Korea, 291–303. DOI: 10.1109/HPCA51647.2021.00033.

[14] Manjunath Shevgoor, Sahil Koladiya, Rajeev Balasubramonian, Chris Wilkerson, Seth H Pugsley, and Zeshan Chishti. 2015. Efficiently prefetching complex address patterns. In *proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Waikiki, HI, USA, 141–152.

[15] Alan Jay Smith. 1982. Cache memories. *ACM Computing Surveys (CSUR)*, 14, 3, 473–530.

[16] Stephen Somogyi, Thomas F Wenisch, Anastassia Ailamaki, Babak Falsafi, and Andreas Moshovos. 2006. Spatial memory streaming. *ACM SIGARCH Computer Architecture News*, 34, 2, 252–263.

[17] Peter Van Vleet, Eric Anderson, Lindsay Brown, J-L Baer, and Anna Karlin. 1999. Pursuing the performance potential of dynamic cache line sizes. In *proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*. Lisbon, Portugal, 528–537.

[18] Zhenlin Wang, Doug Burger, Kathryn S McKinley, Steven K Reinhardt, and Charles C Weems. [n. d.] Guided region prefetching: a cooperative hardware/software approach. In *proceedings of the Annual International Symposium on Computer Architecture, 2003. Proceedings.* San Diego, California, USA, 388–398.

[19] Hao Wu, Krishnendra Nathella, Dam Sunwoo, Akanksha Jain, and Calvin Lin. 2019. Efficient metadata management for irregular data prefetching. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–13.

[20] Xiangyao Yu, Christopher J Hughes, Nadathur Satish, and Srinivas Devadas. 2015. Imp: indirect memory prefetcher. In *proceedings of the 48th International Symposium on Microarchitecture*. Waikiki, HI, USA, 178–190.