Dead-Block Filter for Prefetcher Optimization

Himanshu Galav
Department of Electrical Engineering
IIT Ropar
Punjab, India
2021eeb1177@iitrpr.ac.in

***Abstract*—In the realm of cache prefetching using Deep Learning Models, the oversight of DeadBlocks, i.e., prefetched addresses that remain unused and contribute to cache pollution, is a notable gap. This paper proposes leveraging DeadBlocks to examine predicted prefetch addresses, thus mitigating unnecessary cache pollution. We introduce a method that counts the total accesses to a block to track DeadBlocks efficiently. Previous research has introduced various Dead Block Predictors, each with its approach. Our study focuses on implementing a Dead Block Filter for enhancing the prefetching efficiency of all prefetchers. Experimental results demonstrate an improvement in Instruction Per Cycle (IPC) and a reduction in useless prefetches, leading to an overall enhancement in prefetcher efficiency.**
Github Repository: Link

I. Introduction

Cache prefetching is a critical technique utilized in modern computer systems to enhance memory access latency and overall system performance. By predicting future memory accesses and prefetching the corresponding data into the cache, prefetchers aim to reduce the latency associated with accessing data from main memory. However, prefetching mechanisms often face the challenge of unnecessary cache pollution, where prefetched data remains unused and occupies valuable cache space.

One significant contributor to cache pollution is the prefetching of addresses that correspond to DeadBlocks—memory blocks or cache lines that are prefetched but ultimately remain unused by subsequent memory accesses. The presence of DeadBlocks not only wastes memory bandwidth but also leads to reduced cache effectiveness by displacing potentially useful data.

In this paper, we propose a novel approach to address the issue of cache pollution by leveraging DeadBlocks to examine predicted prefetch addresses. By integrating DeadBlock prediction mechanisms into prefetching strategies, we aim to distinguish between useful and unnecessary prefetches, thereby optimizing cache utilization and enhancing prefetcher efficiency.

In the following sections, we provide an overview of related work in the field of DeadBlock prediction and prefetching mechanisms. We discuss various Dead Block Predictors proposed in the literature and outline their contributions to prefetching efficiency. Subsequently, we present our approach to integrating DeadBlock prediction into prefetching strategies, followed by experimental results demonstrating the effectiveness of our approach in improving cache performance metrics.

II. Background And Related Work

Previous research has explored various approaches to predict and mitigate the impact of DeadBlocks on cache performance. Several Dead Block Predictors have been proposed, each offering unique strategies for identifying and handling DeadBlocks.
We summarize some of the key Dead Block Predictors and their contributions below:

A. Trace-Based Predictor

Introduced by Lai et al., this predictor prefetches data into DeadBlocks in the L1 data cache based on reference traces. The predictor collects a trace of instruction addresses accessing a particular block, predicting DeadBlocks based on the theory that the same trace leading to the last access for one block will lead to the last access for other blocks.

B. Time-Based Predictor

Proposed by Hu et al., this predictor learns the live duration of a block and predicts it dead if it is not accessed for a certain number of cycles. Abella et al. also propose a similar predictor based on the number of references rather than cycles, aimed at reducing cache leakage.

C. Cache Burst Predictor

Cache bursts can be utilized with various Dead Block Predictors, consisting of all contiguous accesses to a block while in the Most Recently Used (MRU) position. This predictor updates predictions only on each burst rather than each reference, reducing the number of accesses and updates to the prediction table.

D. Counting-Based Predictor

Proposed by Kharbutli and Solihin, this predictor tracks the number of accesses to each block and predicts a block dead if it has been accessed more often than the previous generation. Each entry in the predictor table includes a confidence counter to improve prediction accuracy.

These Dead Block Predictors, among others, have been applied to prefetching and block replacement strategies to improve cache efficiency. However, the integration of DeadBlock prediction into prefetching mechanisms remains an ongoing area of research, which this paper aims to address by proposing a novel approach to optimize prefetcher efficiency and mitigate cache pollution.

III. Approach

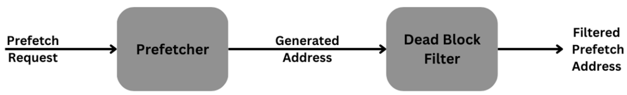


Fig. 1. Model of Cascaded Dead-Block Filter with the Prefetcher

Our approach aims to integrate DeadBlock prediction mechanisms into existing prefetching strategies to mitigate cache pollution and enhance prefetcher efficiency. We begin by conducting simulations using the champsim simulator across various trace files, including bc-0, bc-5, pr-14, sssp-14, bfs-14, and cc-5. This initial step allows us to identify DeadBlocks—memory blocks or cache lines that are prefetched but ultimately remain unused by subsequent memory accesses.

Upon identifying DeadBlocks, we store this data for subsequent processing. In the second phase of our approach, we implement a processing pipeline to handle the DeadBlock data. This involves parsing the DeadBlock file, which contains both DeadBlock addresses and their associated Program Counter (PC) information. Subsequently, we utilize an unordered_map data structure to efficiently store and manage DeadBlock information.

During prefetcher operation, the DeadBlock data stored in the unordered_map is leveraged to filter out predicted prefetch addresses that correspond to DeadBlocks. By scrutinizing prefetch addresses against the DeadBlock information, we aim to prevent unnecessary prefetches, thereby reducing cache pollution and enhancing overall prefetcher efficiency.

Experimental Results

In our experimental evaluation, we observed significant improvements in cache performance metrics, particularly Instruction Per Cycle (IPC), prefetch efficiency, and cache utilization. By incorporating DeadBlock prediction mechanisms into prefetching strategies, we achieved notable enhancements in the following aspects:

A. IPC Improvement

Our approach led to a notable increase in IPC, indicating improved overall system performance. By reducing unnecessary prefetches and cache pollution, more CPU cycles were utilized effectively, resulting in higher throughput and efficiency.

B. Reduction in Useless Prefetches

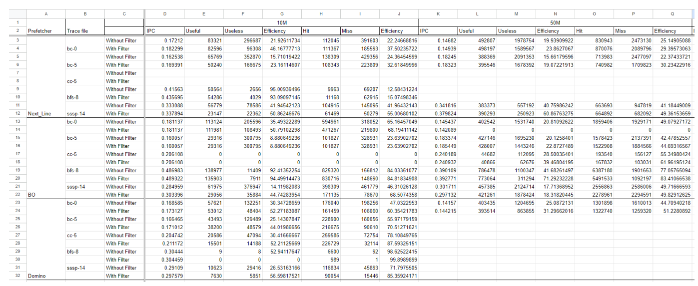
We observed a significant decrease in the issuance of useless prefetches, which are prefetch addresses that correspond to DeadBlocks. By filtering out these unnecessary prefetches, we minimized cache pollution and improved prefetcher accuracy.

C. Increase in Useful Prefetches

Conversely, we noted an increase in useful prefetches—prefetch addresses that correspond to memory accesses actually utilized by subsequent instructions. This indicates that our approach effectively distinguishes between useful and unnecessary prefetches, leading to better prefetcher performance.

D. Overall Efficiency Enhancement

The combined effect of reducing useless prefetches, increasing useful prefetches, and improving cache utilization resulted in an overall enhancement in prefetcher efficiency. By optimizing cache prefetching mechanisms through DeadBlock prediction, we achieved improved system performance and resource utilization.



			10M							50M						
Prefetcher	Trace file		IPC	Useful	Useless	Efficiency	Hit	Miss	Efficiency	IPC	Useful	Useless	Efficiency	Hit	Miss	Efficiency
		Without Filter	0.17212	83321	296687	21.92611734	112045	391603	22.24668816	0.14682	492807	1978754	19.93909922	830943	2473130	25.14905088
	bc-0	With Filter	0.182299	82596	96308	46.16777713	111367	185593	37.50235722	0.14939	498197	1589567	23.8627067	870076	2089796	29.39573063
		Without Filter	0.162538	65769	352870	15.71019422	138309	429356	24.36454599	0.18245	388369	2091353	15.66179596	713983	2477097	22.37433721
	bc-5	With Filter	0.169391	50240	166675	23.16114607	108343	223809	32.61849996	0.18323	395546	1678392	19.07221913	740982	1709823	30.23422916
		Without Filter														
	cc-5	With Filter														
		Without Filter	0.41563	50564	2656	95.00939496	9963	69207	12.58431224							
	bfs-8	With Filter	0.435695	54286	4029	93.09097145	11168	62915	15.07498346							
		Without Filter	0.333088	56779	78585	41.94542123	104915	145095	41.96432143	0.341816	383373	557192	40.75986242	663693	947819	41.18449009
Next_Line	sssp-14	With Filter	0.337894	23147	22362	50.86246676	61469	50279	55.00680102	0.379824	390293	250923	60.86763275	664892	682092	49.36153659
	bc-0	Without Filter	0.181137	113124	205596	35.49322289	594961	318052	65.16457049	0.145437	402542	1531740	20.81092622	1859406	1929171	49.07927172
		With Filter	0.181137	111981	108493	50.79102298	471267	219800	68.19411142	0.142089	0	0	0	0	0	0
	bc-5	Without Filter	0.160057	29316	300795	8.880649236	101827	328931	23.63902702	0.183374	427146	1695230	20.1258401	1578423	2137391	42.47852557
		With Filter	0.160057	29316	300795	8.880649236	101827	328931	23.63902702	0.185449	428007	1443246	22.8727489	1522908	1884566	44.69316567
	cc-5	Without Filter	0.206108	0	0	0	0	0	0	0.240189	44682	112095	28.50035401	193540	156127	55.34980424
		With Filter	0.206108	0	0	0	0	0	0	0.240932	40866	62676	39.46804195	167832	103031	61.96195124
	bfs-8	Without Filter	0.486983	138977	11409	92.41352254	825320	156812	84.03351077	0.390109	786478	1100347	41.68261497	6387180	1901653	77.05765094
		With Filter	0.489322	135903	7911	94.49914473	830716	148690	84.81834908	0.392771	773064	311294	71.29232228	5491533	1092197	83.41066538
	sssp-14	Without Filter	0.284959	61975	376947	14.11982083	398309	461779	46.31026128	0.301711	457385	2124714	17.71368952	2556863	2586006	49.71666593
BO		With Filter	0.303396	29056	35884	44.74283954	171135	78670	68.5074358	0.297132	421261	1878424	18.31820445	2278961	2294591	49.82912625
	bc-0	Without Filter	0.168585	57621	132251	30.34728659	176040	198256	47.0322953	0.14157	403435	1204695	25.0872131	1301898	1610013	44.70940218
		With Filter	0.173127	53012	48404	52.27183087	161459	106060	60.35421783	0.144215	393514	863855	31.29662016	1322740	1259320	51.2280892
	bc-5	Without Filter	0.166465	43493	129489	25.14307847	228900	180056	55.97179159							
		With Filter	0.171012	38200	48579	44.01986656	216675	90610	70.51271621							
	cc-5	Without Filter	0.204742	20586	47094	30.41666667	259585	72754	78.10849765							
		With Filter	0.211172	15501	14188	52.21125669	226729	32114	87.59325151							
	bfs-8	Without Filter	0.30444	9	8	52.94117647	6600	92	98.62522415							
		With Filter	0.304459	0	0	0	989	1	99.8989899							
	sssp-14	Without Filter	0.29109	10623	29416	26.53163166	116834	45893	71.7975505							
Domino		With Filter	0.297579	7630	5851	56.59817521	90054	15446	85.35924171							

Fig. 2. Results with and without deadblock filter

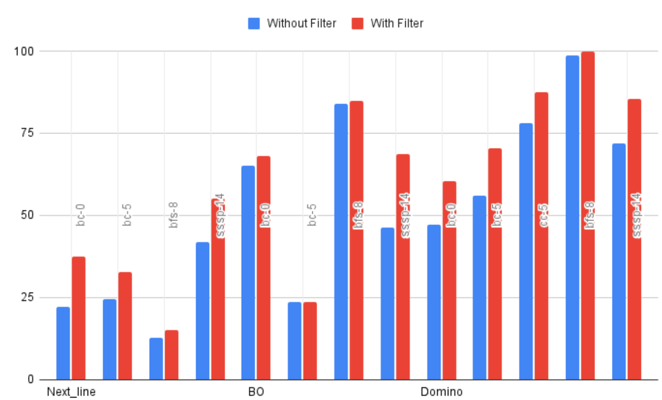


Fig. 3. Chart for Efficiency in terms of Hits & Misses

Results: Link to sheet

These experimental results underscore the effectiveness of integrating DeadBlock prediction mechanisms into prefetching strategies to optimize cache performance and enhance overall system efficiency. By accurately distinguishing between useful and unnecessary prefetches, our approach offers a practical solution for mitigating cache pollution and improving prefetcher performance in real-world scenarios

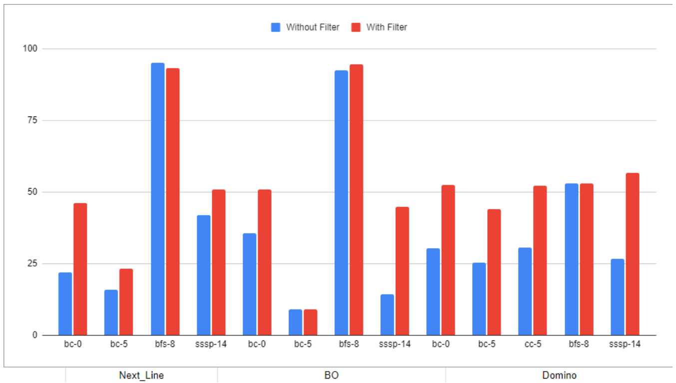


Fig. 4. Chart for Efficiency in terms of Usefull & Useless

Conclusion and Future Work

In conclusion, this paper presents a novel approach to optimize cache prefetching mechanisms by leveraging DeadBlocks to mitigate cache pollution and enhance prefetcher efficiency. By integrating DeadBlock prediction mechanisms into prefetching strategies, we effectively distinguish between useful and unnecessary prefetches, thereby optimizing cache utilization and improving overall system performance.

Our experimental results demonstrate significant improvements in cache performance metrics, including Instruction Per Cycle (IPC) enhancement, reduction in useless prefetches, and increased useful prefetches. These findings highlight the effectiveness of our approach in optimizing prefetcher efficiency and mitigating the impact of DeadBlocks on cache performance.

Furthermore, our approach extends beyond prefetching optimization. The Dead Block filters developed can also be leveraged to eliminate DeadBlocks present in the cache, thereby improving cache replacement policies. By accurately identifying and handling DeadBlocks, cache replacement algorithms can make more informed decisions, leading to better cache utilization and performance

Additionally, the Dead Block filters can enhance prefetcher design by providing valuable feedback on prefetch accuracy and effectiveness. By incorporating DeadBlock prediction mechanisms into prefetcher architectures, prefetchers can dynamically adjust prefetching strategies based on the presence of DeadBlocks, thereby further improving prefetcher efficiency and reducing cache pollution.

Looking ahead, future work could focus on refining DeadBlock prediction mechanisms to make them more adaptive and efficient in real-time scenarios. Developing more sophisticated Dead Block Predictors capable of dynamically adjusting prediction parameters based on program behavior and cache characteristics could lead to even greater improvements in cache performance.

Furthermore, the integration of DeadBlock prediction into multi-level cache hierarchies and shared cache environments represents an interesting avenue for exploration. By extending DeadBlock prediction mechanisms to higher-level caches and shared cache architectures, we can further optimize cache utilization and prefetcher efficiency across diverse computing environments.

Overall, this paper provides a foundation for future research in the optimization of cache prefetching mechanisms and the mitigation of cache pollution. By harnessing DeadBlocks as a valuable resource for cache management, we can unlock significant improvements in system performance and efficiency in cache-intensive computing environments.

Acknowledgment

We would like to express our sincere gratitude to Dr. T.V. Kalyan for his invaluable guidance and support throughout the duration of the CS531 course on Memory Systems and Architecture. Dr. Kalyan's expertise, encouragement, and insights have been instrumental in shaping our understanding of cache prefetching mechanisms and DeadBlock prediction.

We would also like to extend our appreciation to the teaching assistants, Sourabh and Prathmesh, for their assistance and support during the course. Their dedication to assisting students and clarifying concepts has been immensely beneficial in our learning journey.

We are grateful to Dr. Kalyan and the teaching assistants for their unwavering commitment to fostering a conducive learning environment and facilitating our academic growth.

References

[1] P. Zhang, A. Srivastava, A. V. Nori, R. Kannan, and V. K. Prasanna, "Fine-grained address segmentation for attention-based variable-degree prefetching," in Proceedings of the 19th ACM International Conference on Computing Frontiers, 2022.

[2] "ChampSim". 2017. https://github.com/ChampSim/ChampSim.

[3] A. Lai, C. Fide, B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In ISCA-28, 2001.