International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

PP-Bridge: Establishing a Bridge between the Prefetching and Cache Partitioning

Purnendu Das [1], Nurulla Mansur Barbhuiya [2], Bishwa Ranjan Roy [3,*]
[1,2,3] Department of Computer Science
Assam University, Silchar
Assam, India
e-mail: [1]purnen1982@gmail.com, [2]nurullabarbhuiya@gmail.com, [3]brroy88@gmail.com
*Corresponding Author

Abstract— Modern computer processors are equipped with multiple cores, each boasting its own dedicated cache memory, while collectively sharing a generously sized Last Level Cache (LLC). To ensure equitable utilization of the LLC space and bolster system security, partitioning techniques have been introduced to allocate the shared LLC space among the applications running on different cores. This partition dynamically adapts to the requirements of these applications. Prefetching plays a vital role in enhancing cache performance by proactively loading data into the cache before it get requested explicitly by a core. Each core employs prefetch engines to decide which data blocks to fetch preemptively. However, a haphazard prefetcher may bring in more data blocks than necessary, leading to cache pollution and a subsequent degradation in system performance. To maximize the benefits of prefetching, it is essential to keep cache pollution to a minimum. Intriguingly, our research has uncovered that when existing prefetching techniques are combined with partitioning methods, they tend to exacerbate cache pollution within the LLC, resulting in a noticeable decline in system performance. In this paper, we present a novel approach aimed at mitigating cache pollution when combining prefetching with partitioning techniques.

Keywords-Covert Channel Attack; Side Channel Attack; Shared Memory; Last Level Cache; Flush+Reload.

I. INTRODUCTION

The effective management of cache resources is becoming increasingly critical in the current context as a result of the advancements that have been made in processor design [1, 2]. The need for improved cache architecture and management strategies has emerged as a consequence of the rising number of cores contained into chip multicore processors (CMP) as well as the requirements placed on applications [1, 3]. Consequently, a great number of researchers attempted to investigate the shared and private cache design systems, each of which offers a variety of benefits and drawbacks [1]. There is no interference between applications while using private cache; nevertheless, because these caches are of a limited capacity, they are unable to lower miss rate very effectively. Because shared caches have a greater capacity than private caches, they are more effective at reducing the number of cache misses [1, 3]. Therefore, modern multicore processor have private upper level caches for each core and a sharable large sized Last Level Cache (LLC) which is accessible to all the core. An example of multicore processor is shown in the figure-1. However, because shared caches are subject to interference, it is possible that they will not improve performance or quality of service (QoS). Some application may unnecessarily consume more cache space than the other applications. Cache partitioning schemes [4–7] come into play as a result to handle this limitation. It partitions the LLC among the different applications running in the system such that the cache space can be used fairly.

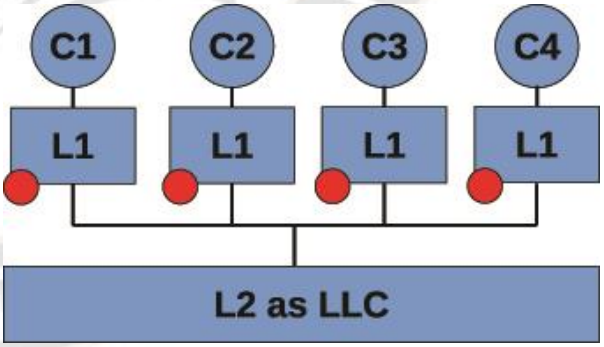

Fig.1: An Example of multicore processor with four cores. Each core has a private L1 cache and all the core share a common larger sized L2 cache as LLC. The red circles are prefetch engine placed near to each core.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

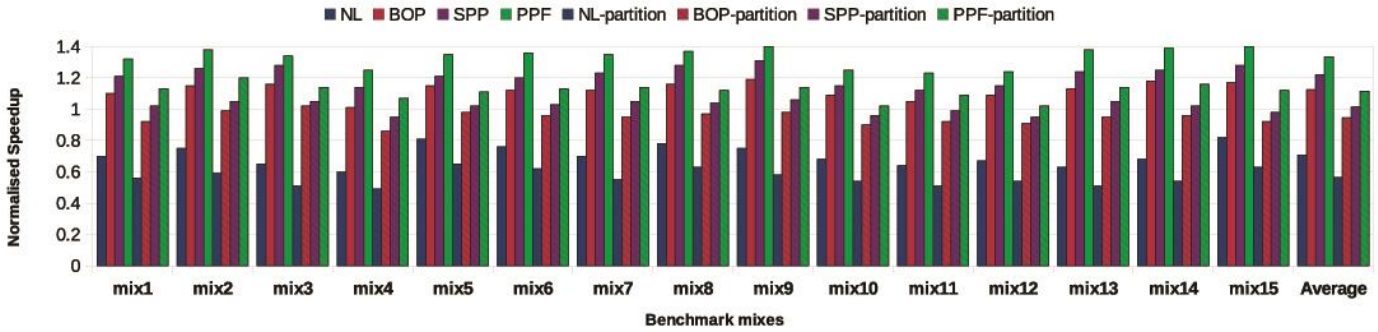

Fig. 2: Comparison of normalised speedup of different prefetching techniques with and without partitioning. The last four (cross-lined) bars for each mixes are for prefetching with partitioning.

Cache prefetching is a crucial strategy employed to address the memory-wall problem, primarily aimed at mitigating latency to access memory blocks [8–10]. This technique revolves around the anticipation and retrieval of future memory block accesses from the main memory which is likely to be demanded by the processing cores. These fetched blocks are aptly termed "prefetched blocks", while those requested directly by the core are referred to as "demand blocks". There exist two main types of prefetchers: software prefetchers and hardware prefetchers. Software prefetchers, which are static in nature, heavily rely on compilers and cannot adapt to the dynamic behavior of the applications running in the system [9]. In contrast, hardware prefetchers incorporate a dedicated hardware circuit within the processor, commonly known as a prefetch engine. This hard-ware component excels at efficiently predicting future memory block accesses based on historical patterns [9,11,12]. Hardware prefetchers can be further classified into spatial and temporal prefetchers, depending on their focus on spatial or temporal locality in previously accessed memory patterns [9,13,14]. Spatial prefetchers operate by analyzing the past memory accesses demanded by different cores and predicts the future access by observing the fixed behavioral patterns in those memory accesses. Consequently, purely spatial prefetchers exhibit remarkable accuracy in predicting regular memory access patterns but falter when confronted with irregular memory access address patterns. On the other hand, temporal prefetchers retain knowledge of past memory accesses. When encountering a previously seen memory address, they prefetch the consecutive addresses stored in their memory, known as metadata. However, this approach demand significant memory resources. In the context of this work, our focus is exclusively on spatial prefetching.

Inaccurate or aggressive prefetching may prefect block which will never access by the core. Such prefetch blocks are fetched unnecessary and also creates pollution in the cache. The existing prefetching ideas like next line prefetcher (NL), BOP [15], PPF [16], and SPP [17] etc, improve the system performance with limited cache pollution. However we have observed that when these ideas are applied in presence of cache partitioning the cache pollution increases. None of these existing prefetching techniques are proposed considering the partitioning cache. All these techniques assume that the entire LLC can be accessible to all the cores and the underlying replacement policy can replace a block from any ways as a victim. With some experimental analysis we have observed that the performance of these prefetch techniques decreases in presence of partitioning. Figure 2 shows the performance of a 4-core system with different prefetching techniques. The comparison is shown for with and without applying partitioning on top of prefetching. Different spec cpu 2006 applications [18] are used to prepare the benchmark mixes for this experiment. The details about the experimental setup is discussed in Section IV. It can be observed from the figure that all the prefetching techniques show less performance improvement while combining with cache partitioning. As per our knowledge, no work has been done so far to jointly analyze the behavior of prefetching and cache partitioning.

In this paper we have proposed an idea to efficiently use the prefetching techniques on cache partitioning. Our proposed idea can be used to combine any prefetching and partitioning technique. We call our proposed technique as PP-Bridge, a bridge between prefetching and partitioning. The primary idea of PP-Bridge is divided into two parts. First is to prevent the prefetching techniques to partition the LLC as per their aggressiveness. Second is to prevent the prefetching to evict demands blocks from its own partition. Combining both these ideas, the proposed PP-Bridge reduces the additional cache pollution caused by combining with partitioning. The detailed description about the idea is given in Section III. The major works done in this paper are as follows:

- We have proposed an alternative technique for periodically deciding the LLC partitions such that the prefetch

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

aggressiveness cannot control the decision of LLC partitioning.

- To reduce the impact of cache pollution at LLC, we have proposed a technique to keep the demand blocks in the cache for longer duration.

- Combining the above two techniques the performance of the system increases significantly.
- The proposed technique is experimentally compared with the existing prefetching techniques.

The rest of the paper is organized as follows. The background and related works are discussed in the next section. The proposed idea is discussed in Section III. The experimental analysis is discussed in Section IV. Some sensitive issues related to the proposed idea are discussed in Section V. Finally the paper concludes at Section VI.

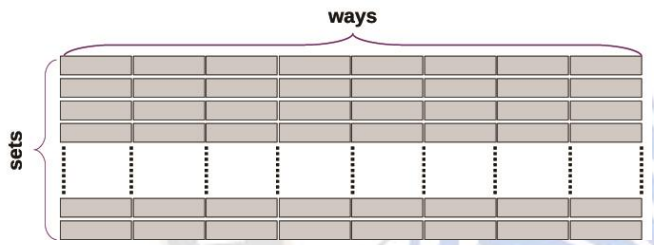

Fig. 3: Example of an 8-way set-associative cache with S number of sets.

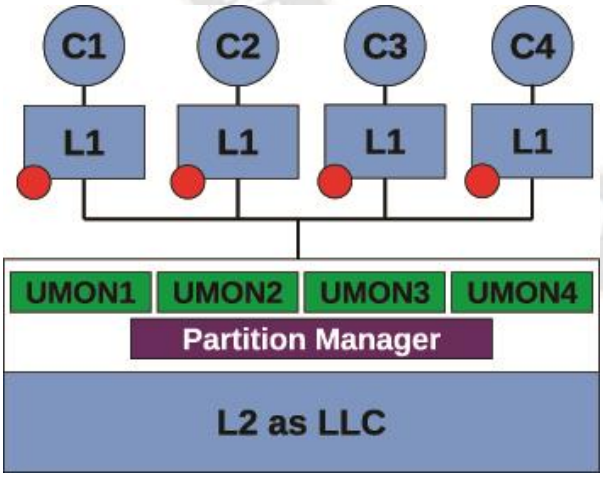

Fig. 4: An Example of the cache partitioning technique as mention in UCP [4]. The red circles are prefetch engine placed near to each core.

II. BACKGROUND

A. Set-Associative Cache and Replacement Policies

In this study, all considered cache memories are of the set-associative type. Each cache is configured with a fixed number of sets, and a block can only be mapped to a specific set based on the set-index bits within the block's address. The number of blocks that a set can accommodate is determined by the associativity of the cache. An N-way set-associative cache, for instance, can hold N blocks within each set. For illustration, refer to Figure 3, which depicts an 8-way set-associative cache. Throughout this paper, we employ the notation N to denote the cache associativity, and S to represent the number of sets within the Last-Level Cache (LLC).

When a new block must be inserted into a non-empty set, the replacement policy comes into play by removing an existing block from the set. Replacement policies adhere to three primary rules: insertion, promotion, and eviction. The insertion rule governs the placement of a newly arriving block into its appropriate position within the cache. Promotion, on the other hand, comes into play when a cache hit occurs, leading to the promotion of the relevant block. The block that is selected for eviction from the set is known as the victim block, and the eviction rule is responsible for selecting this victim block for replacement. In accordance with established replacement policies, any process can evict a block belonging to another process if it is chosen as the victim block. However, when two processes are not permitted to evict each other's blocks from the cache, they are considered isolated from each other.

B. Cache Partitioning

Cache partitioning is a technique used to fairly divide the LLC among the multiple applications or cores. The applications running on a core can be divided into cache friendly (CF) and non cache friendly (NCF) applications. The CF applications, with larger cache space, reduce the number of misses. The NCF application need only a limited cache space. Any cache space beyond that cannot reduce the number of misses for NCF applications. A common example of NCF is the streaming applications. Hence, uniformly dividing the LLC space among all the applications leads to the inaccurate utilization of the LLC. This is because an NCF applicant may evict all the important blocks of a CF application if care has not been taken. Cache partitioning is a technique proposed to prevent this situation.

1) Utility Based Cache Partitioning: The phenomena of way-based cache partitioning is first observed in Utility Based Cache Partitioning (UCP) [4]. In general, UCP analyses the information gathered from each application's demand while it is operating by using a low-cost monitoring circuit.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

After doing so, it uses this data to spread the ways that cache is used throughout the cores of CMP. The partition set that is handed to the user by UCP is referred to as the target partition. If there is a CMP with eight cores (core-0, core-1, core-2, core-3, core-4, core-5, core-6, and core-7) and an N-way set associative LLC, then the target partition can be expressed as (a, b, c, d, e, f, g, h), where $a+b+c+d+e+f+g+h = N$. Here a is the ways count reserved for core-0, b is the ways count reserved for core-1, and so on. If the partition once decided, does not change during the execution of the computer then such partitions are called static partition.

In case of dynamic partition UCP observes the behavior of a selective number of sets called sample sets. The partition changes dynamically based on the behavior of these sample sets. To observe the utilization of each core, UCP maintain one module called UMON for each core. The UMON records all the cache request received by the LLC from a particular core. Please note that UMON records only the request coming for the sample sets. The information collected by all the UMONs are then shared with a common partition manager who decides the final partition. The procedure repeats after a fixed cycle of execution. Figure 4 shows an example of such cache partitioning.

C. Cache Prefetching

Ishhi and colleagues [19] introduced the AMPM prefetcher, a technique designed to identify common strides within frequently accessed memory regions known as hot zones. AMPM accomplishes this by employing pattern matching. When a pattern is discovered in the hot zone, AMPM begins prefetching along the projected strides. Pugsley et al. suggested Sandbox prefetching [20], which uses a bloom filter to initially add prefetch addresses. These addresses are then checked against the contents of the bloom filter. When the prefetch accuracy exceeds a predefined threshold, real prefetches are triggered.

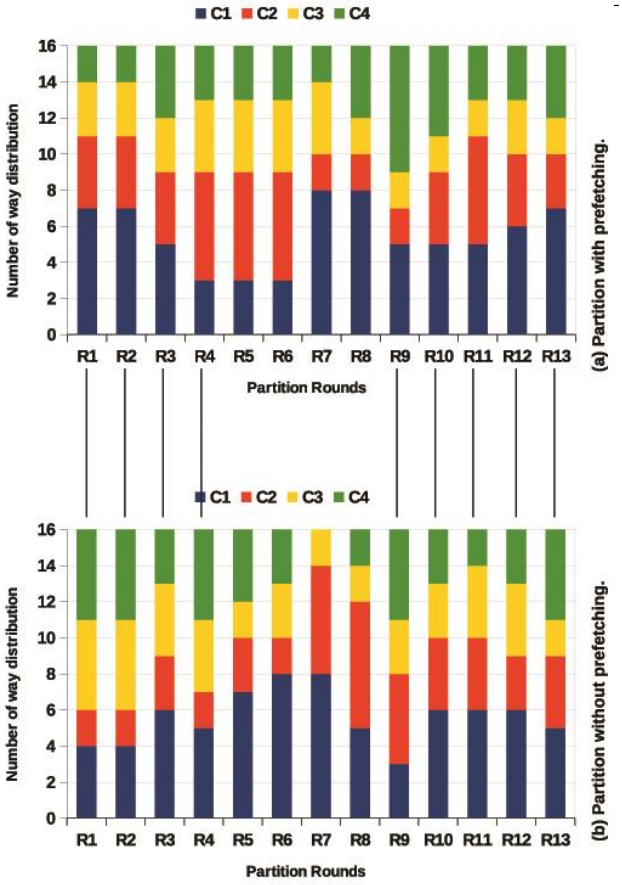

Fig. 6: The LLC partitioning shows with and withour considering the prefetch request to take the decision of partitioning. The results shows for multiple prefetch rounds for a sing benchmark mix.

Michaud et al. [15] proposed the optimal-offset Prefetcher (BOP), a dynamic approach for determining the optimal block offset at runtime. BOP employs a Recent Record database and a best-offset learning strategy to analyse the cache access patterns for 52 distinct offsets It identifies and monitors global deltas to accurately depict cache request patterns and to ensure prompt prefetching for enhanced efficiency. Lookahead

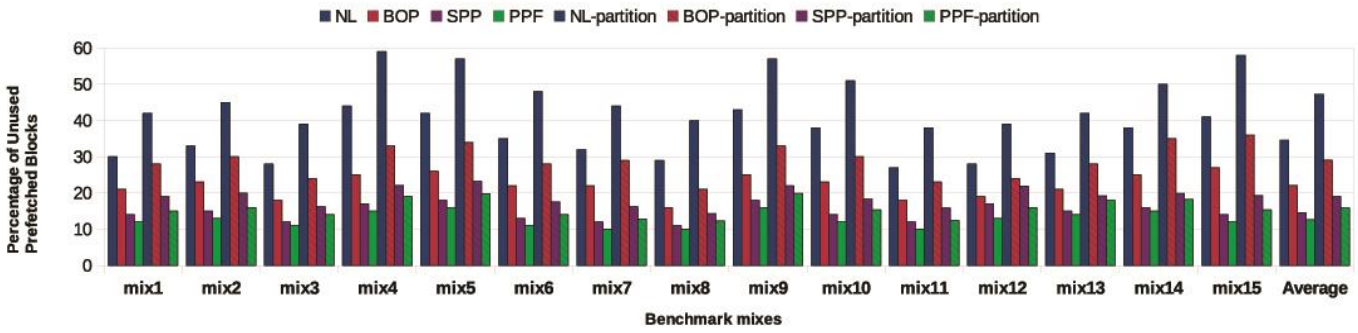

Fig. 5: Comparison of cache pollution done by prefetching techniques with and without combining with partitioning. Here the cache pollution is measures as the percentage of prefetch blocks not requested (or used) by any core.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

prefetchers such as SPP [17], KPC [21], and PPF [16] store the past record of cache access patterns in a compact signature form. When a signature match is obtained, the compact history is used to prefetch blocks. Cache pollution is minimized by training the prefethers by feeding the performance parameter of LLC so that undesirable prefetches can be avoided.

DSPatch [22] is a prefetching approach described by Bera et al. that describes cache access patterns inside a page as a bit-vector. To eliminate unwanted prefetches, DSPatch maintain two different bit-patterns of spatial access, one targeting prefetch coverage while the other targeting prefetch precision. DSPatch prefetch cache blocks by analysing one of these bit-patterns during runtime based on DRAM bandwidth utilisation information. It is observed from the finding that prefetched blocks frequently get stale after their initial access, Shesadri et al. devised ICP [23], which lowers the priority of cache blocks that are prefetched for the first time.

Domino prefetcher [24] follows temporal locality to issue prefetches. It keeps the record of the last two misses to predict the future cache access. Irregular Stream Buffer (ISB) [25] advances previous techniques by incorporating a structural address space with spatially ordered addresses. This involves converting non-contiguous addresses that are temporally related in the physical address space into contiguous addresses within the structural address space.

Managed Irregular Stream Buffer (MISB) [26] improves on this solution by effectively handling the metadata by prefetching, correcting for delays in receiving metadata from off-chip. Triage [27] improves on the preceding solution by lowering the amount of space required for metadata storage and minimizing off-chip access traffic. Triage implies that a section of the LLC can be used to store metadata because a small fragment of it is sufficient for issuing prefetch request, and the advantages of prefetching outweigh the advantages of a bigger cache.

III. The Proposed Idea

A. Motivation

The existing prefetching works [15, 17, 19] never con-sider the impact of partitioning on the prefetching. A partitioned cache provides restricted cache size to each core hence aggressive prefetching creates cache pollution in some partitions. Another important observation is that the involvement of the prefetch requests in taking the decision of partitioning has limitations. A core using aggressive prefetching may take a larger partition in the cache. Figure 5 shows the cache pollution caused by different prefetching techniques observed with and without partitioning. A fixed static partitioning is used for this experiment. It can be observed that the prefetching techniques show high cache pollution when combined with cache partitioning. Here the cache pollution is measured as the percentage of prefetched blocks not requested by any core. The pollution is increased by 26%, 23%, 19%, and 16% in case of NL, BOP, SPP, and PPF respectively. Figure 6 shows the difference in partition sizes with and without considering the prefetching requests while deciding the partition of a 16-way associative LLC. Both upper and lower part of the figure shows the distribution of 16 ways among the four cores.

The y-axis shows how the 16-ways are distributed among the four cores (C1, C2, C3, and C4). The x-axis shows the partitioning decision in multiple partitioning rounds. The upper part of the figure shows the partition when UMONs also record the prefetching requests. The lower part of the figure shows the partition when UMONs do not consider the prefetching requests. Comparing the upper and lower figure, for each round, it can be observed that the partition distribution are not same. In the upper part, some cores get high partition when it is actually not required.

B. The Main Idea (PP-Bridge)

Because of the issues mentioned above in this work we have been motivated to propose a technique to efficiently combine the prefetching and cache partitioning. The propose PP-Bridge has two important part:

- Reducing the inter-application prefetch pollution the pollution created by the prefetched blocks of an application by evicting the demand blocks of other applications from LLC.
- Reducing the intra-application prefetch pollution the pollution created by the prefetched blocks of an application by evicting the demand blocks of the same application.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

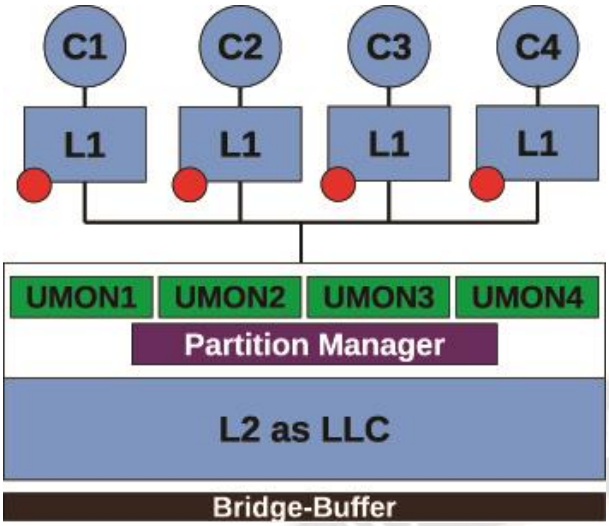

Fig. 7: Example of a 4-core processor including the modules required to implement PP-Bridge. The red circles are prefetch engine placed near to each core.

Reducing inter-application prefetch pollution: The main cause of this type of pollution is already discussed in Section III-A (Figure 5). To reduce this pollution we propose not to involve the prefetch block requests while computing the LLC partitions. Hence, from the sample sets, only the demand requests are monitored by the UMON module. The partition is decided based on the demand request observed from each core in the UMON. Please note that, in this work, we assume one application running in each core. The UMON circuit as discussed in Section II-B1, records all the demand requests and periodically analyzes them to change the LLC partition. The main advantage of such restriction in PP-Bridge is that now an aggressively prefetched application cannot demand larger cache space in the LLC. The partitioning is provided based on the demand request. When a demand request comes for an already prefetch request, the UMON considers that request for partitioning. Hence, if a core needs more prefetching blocks and the prefetcher is accurate, the core eventually gets a larger partition. However, a core with an aggressive prefetcher with high cache pollution cannot get a high partition LLC. As per our knowledge, none of the existing ideas have separated the prefetch and demand blocks while partitioning.

Reducing intra-application prefetch pollution: Even after partitioning the LLC as per the demand requests there is another issue that needs to be addressed. An application after the partition can use the LLC space only assigned to it. Hence a partitioning decided by demand block may face a challenge to handle the pollution created by the aggressive prefetcher. A demand-based cache partitioning prevents the inter-application cache pollution. However, the strategy may cause intra-application cache pollution. To overcome this intra-application pollution, we have slightly modified the block eviction policy (replacement policy). If a demand block from the LLC is evicted by an prefetch block then the evicted block is placed in a small buffer. Otherwise, the evicted block is written back to the main memory (if dirty). We call this buffer as bridge-buffer. In future, when a core requests for the block, it is searched simultaneously in both the LLC and the bridge-buffer. A hit either in the cache or bridge-buffer sends the block to the upper level immediately. A hit in the bridge-buffer also moves the block from the buffer to LLC again. Please note that no prefetched block can be placed in the bridge-buffer. Also a demand block if evicted by another demand block is not inserted into the bridge-buffer. This buffer reduces the impact of cache pollution as the incorrectly evicted demand blocks from the cache are given a chance to remain in the chip. Hence, intra-application cache pollution cannot reduce the number of demand hits in the system.

Figure 7 shows all the components of a multicore processor with the support of PP-Bridge. The processor is considered as a 4-core processor. The bridge-buffer used in our experiments has 32-entries. Which means that the buffer can hold at max 32 demand blocks into it.

C. Hardware Overhead

The major additional hardware required by PP-Bridge is the bridge-buffer. Excluding this buffer, all the modules shown in Figure 7 are required in the existing design on top of which PP-Bridge is proposed. Since the buffer can hold 32-entries, the total additional storage required is 2.04KB, which is just 0.04% of the total LLC size. The cache block size is considered as 64 bytes. To calculate the storage overhead we have considered an 12-bit additional storage per block for the tag entry.

TABLE I. Simulation parameters of the baseline system.

Core	Out-of-order, 4 GHz, 4-core.
L1I	32 KB, 8-way, 4 cycles
L1D	48 KB, 12-way, 5 cycles
L2	512 KB, 8-way, 10 cycles, LRU
LLC	4 MB, 16-way, 20 cycles, LRU

IV. Experimental Analysis

Our experiments were conducted using the Champsim trace-based simulator [28], which is well-regarded and has been utilized in recent competitions related to data prefetching and cache replacement. We extended the Champsim simula-tor to accommodate our proposed threat model. The system parameters employed for these experiments are detailed in

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

Table I. In all our experiments, we employed a 4-core sys-tem configuration as outlined in Table I. To create diverse scenarios, we used various applications from the SPEC CPU 2006 benchmark suite [18]. Each experiment involved all four cores concurrently running a distinct application from the SPEC CPU 2006 suite. Therefore, for every run, we assembled a mix of four benchmark applications from SPEC CPU 2006. To ensure that the caches were properly primed, we executed warming runs spanning 50 million instructions. Subsequently, we executed the applications for a minimum of 200 million instructions collectively to calculate the weighted speedup as a performance metric. Our multi-core evaluation encompassed both homogeneous and heterogeneous traces. We generated a total of 15 trace mixes by combining cache-friendly and memory-intensive applications. In cases where a core completed its instructions ahead of others, we replayed the instructions until all cores finished their execution.

A. Performance Evaluation

The proposed PP-Bridge technique is applied on top of a prefetching techniques (on top of partitioning) have improved after applying PP-Bridge.

The proposed PP-Bridge improves the speedup by 22%, 21%, 18%, and 15% over the corresponding technique where PP-Bridge is not applied. The main reason of this improvement is the reduction in both inter-application and intra-application pollution. The detail analysis about the prefetch pollution is discussed next.

B. Cache Pollution

As mentioned in Section III, the proposed PP-Bridge reduces both inter-application and intra-application cache pollution. Figure shows the reduction in cache pollution in PP-Bridge over the existing prefetch+partition setup. The comparison setup is already discussed in Section IV-A. Please note that the result shown in this figure is recorded after only applying the inter-application reduction technique proposed by PP-Bridge. It can be observed from the figure that the pollution is reduced by 12%, 11%, 10%, and 9% over the corresponding

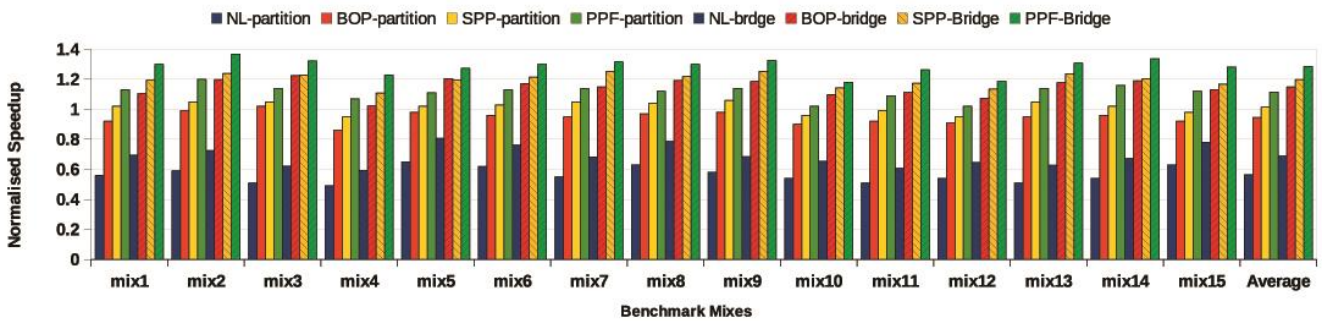

Fig. 8: Performance comparison of PP-Bridge with various prefetching techniques while implemented on top of partitioning.

baseline system where both prefetching and partitioning are already applied. For the experiments we have used different prefetching techniques like NL, BOP [15], SPP [17], and PPF [16]. The partitioning technique use is UCP. As mentioned in Section I, our proposed PP-Bridge helps to efficiently combine both prefetching and partitioning.

Figure 8 shows the efficiency of PP-Bridge in terms of normalised speedup. The figure shows the comparison of each prefetching technique with and without PP-Bridge. In the figure, the prefetching technique without PP-Bridge has "partituion" as suffix while the prefetching with PP-bridge has a suffix of "bridge". Please note that PP-bridge also use a modified UCP-based partitioning technique as discussed in Section III-B. The last four bars for each mix, are for the PP-bridge applied over different prefetching techniques. It can be observed from the figure that the performance of the prefetch+partition setup.

A further improvement in cache pollution is shown while the intra-application pollution is also controlled by efficiently using the bridge-buffer as discussed in Section III-B. Figure 10 shows the final reduction in cache pollution after implementing both inter-application and intra-application pollution reduction technique of PP-Bridge. It can be observed from the figure that the pollution is reduced by 19%, 18%, 17%, and 16% over the corresponding prefetch+partition setup. The improvements shown in this figure reflects the importance of reducing both inter-application and intra-application pollution while applying prefetching with partitioning. As a result, the system get performance benefit as shown in Figure 8.

C. Efficiency of Bridge-Buffer

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

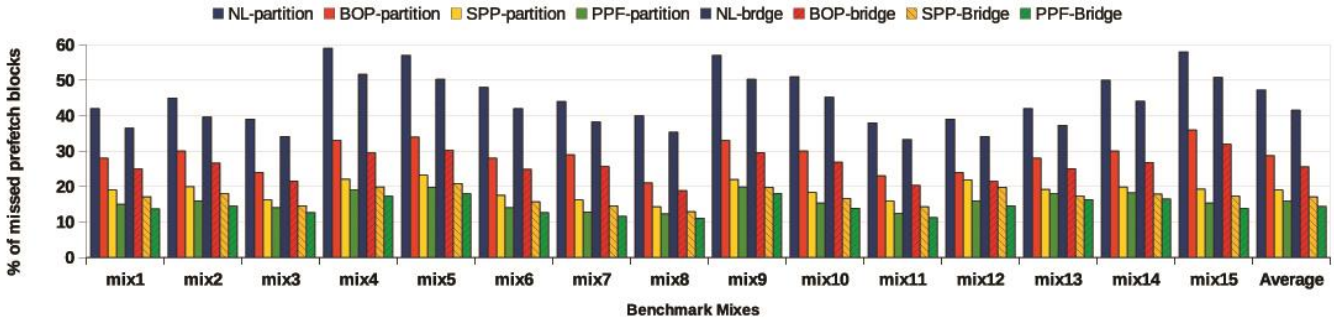

Fig. 9: Cache pollution of the prefetch techniques with partitioning. The pollution for PP-Bridge is shown only with the PP-Bridge technique used to reduce the inter-application pollution

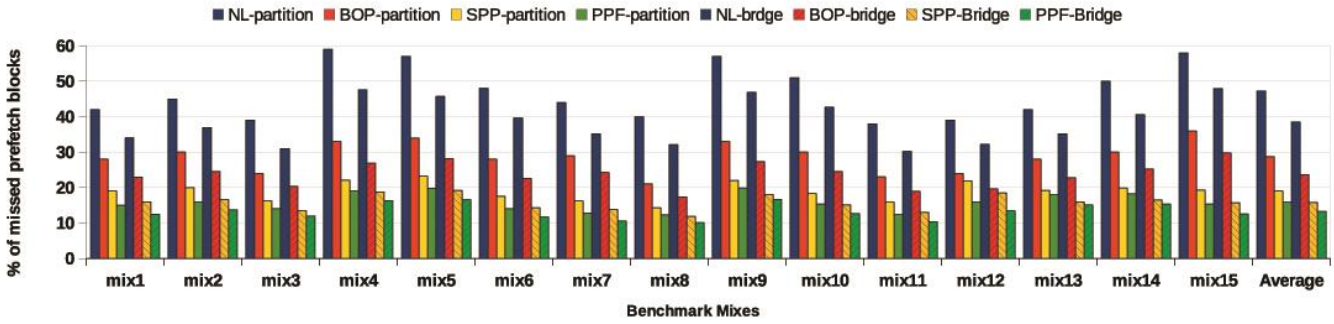

Fig. 10: Cache pollution of the prefetch techniques with partitioning. The pollution for PP-Bridge is shown after reducing both the inter-application and intra-application pollution.

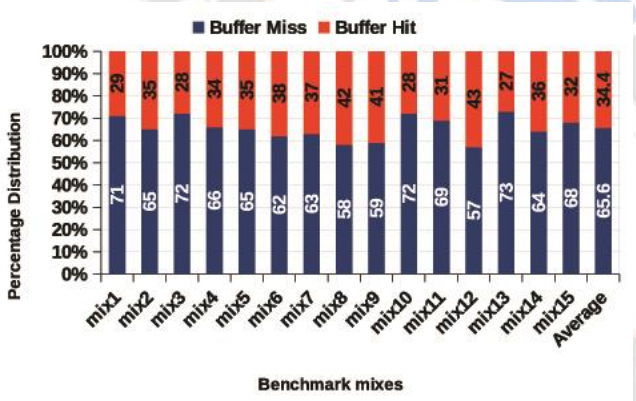

Fig. 11: The percentage distribution of hits and misses in bridge-buffer of the proposed PP-Bridge.

Figure 11 shows the percentage distribution of hit and miss in the bridge-buffer. A block, if missed in LLC, first search in bridge-buffer. A hit in the bridge-buffer eliminates the needs to going to main memory. Hence the total LLC misses can be divided into total bridge-buffer hits and total bridge-buffer misses. From the figure, it can be observed that on average 34.4% LLC misses are getting hit in the bridge-buffer which helps in improving the performance of the system.

V. Other Sensitive Issues

A. Applying PP-Bridge with non-partitioned LLC

The proposed PP-Bridge is only applicable when the prefetching techniques are applied on a a partitioned LLC. It does not show significant improvement when the prefetching techniques are applied over an LLC which is not partitioned dynamically. Since the main motive of this work is to enhance the performance of prefetching in a partitioned LLC, its under-performance in non-partition LLC does not fall under the scope of this paper. Also, LLC partitioning now a days is not just used for fairly distributing the LLC. It is also used to prevent the applications from side channel [29] and covert channel attacks [30], [31]. Hence, a secure LLC may also have partition which is a perfect example of the need of an idea like PP-Bridge

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

B. Scalability of PP-Bridge

The proposed PP-Bridge can be used for any large sized multicore system. Though the proposed idea is only experimented for a single bank LLC, the idea can also be applied to the modern multi-banked LLC [2] without any major hardware overhead. The additional hardware will be required for the bridge-buffer required for in the PP-Bridge. In case of a multi-banked LLC, one bridge-buffer will be required for each LLC bank.

VI. CONCLUSION

Contemporary computer processors are equipped with multiple cores, each having its dedicated private cache memories, while they collectively share a spacious Last Level Cache (LLC). To ensure equitable allocation of LLC resources among concurrently executing applications on different cores and bolster system security, partitioning techniques have been introduced. These techniques enable the dynamic partitioning of the LLC space based on the varying demands of the running applications. Prefetching mechanisms are employed to proactively load data blocks into the cache before they are requested by the core, ensuring quicker access. Prefetching decisions are typically made by specialized prefetch engines associated with each core. However, unbridled prefetching can result in an excess of cached blocks that may never be utilized, leading to cache pollution and performance degradation. To reap the benefits of prefetching while minimizing this pollution, it becomes imperative to strike a balance.

Surprisingly, our research has unveiled that when existing prefetching techniques are combined with partitioning strategies, they tend to exacerbate cache pollution within the LLC, consequently undermining system performance. In this paper, we propose an innovative approach to mitigate cache pollution arising from the combination of prefetching and partitioning, aiming to enhance overall system performance. The proposed PP-Bridge can be used as a bridge between any spatial prefetching techniques with UCP-based partitioning techniques.

REFERENCES

[1] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, *"Multi-Core Cache Hierarchies," Synthesis Lectures on Computer Architecture, vol. 6, no. 3, pp. 1–153, 2011.*

[2] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, *"A NUCA Substrate for Flexible CMP Cache Sharing," in Intl. Conf. on Supercomputing, p. 31–40, 2005.*

[3] S. Das and H. K. Kapoor, *"Dynamic Associativity Management in Tiled CMPs by Runtime Adaptation of Fellow Sets," Transactions on Parallel and Distributed Systems, vol. 28, no. 8, pp. 2229–2243, 2017.*

[4] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: *A low-overhead, high-performance, runtime mechanism to partition shared caches," in Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 39, pp. 423–432, 2006.*

[5] C. Yang, L. Liu, K. Luo, S. Yin, and S. Wei, "Ciacp: *A correlation-and iteration-aware cache partitioning mechanism to improve performance of multiple coarse-grained reconfigurable arrays," IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 1, pp. 29–43, 2016.*

[6] G. Saileshwar, S. Kariyappa, and M. Qureshi, *"Bespoke cache enclaves: Fine-grained and scalable isolation from cache side-channels via flex-ible set-partitioning," in 2021 International Symposium on Secure and Private Execution Environment Design (SEED), pp. 37–49, IEEE, 2021.*

[7] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Pono-marev, *"Non-Monopolizable Caches: Low-Complexity Mitigation of Cache Side Channel Attacks," Transactions on Architecture and Code Optimization, vol. 8, no. 4, pp. 1–21, 2012.*

[8] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, *"Path Confidence based Lookahead Prefetching," in 49th Annual International Symposium on Microarchitecture, pp. 1–12, 2016.*

[9] S. Mittal, "A Survey of Recent Prefetching Techniques for Processor Caches," *ACM Computing Surveys, vol. 49, no. 2, pp. 35:1–35:35, 2016.*

[10] D. Deb, J. Jose, and M. Palesi, *"COPE: Reducing Cache Pollution and Network Contention by Inter-Tile Coordinated Prefetching in NoC-Based MPSoCs," ACM Trans. Des. Autom. Electron. Syst., vol. 26, no. 3, pp. 1–31, 2021.*

[11] S. Vanderwiel, S. Vanderwiel, D. J. Lilja, and D. J. Lilja, "*A Survey of Data Prefetching Techniques," tech. rep., 1996.*

[12] C.-J. Wu, A. Jaleel, M. Martonosi, S. C. Steely, and J. Emer, "*PACMan: Prefetch-Aware Cache Management for high performance caching," in 2011 44th Annual IEEE/ACM International Symposium on Microarchi-tecture (MICRO), pp. 442–453, 2011.*

[13] S. Mittal, "*A Survey of Recent Prefetching Techniques for Processor Caches," vol. 49, no. 2, 2016.*

[14] B. Falsafi and T. F. Wenisch, *A Primer on Hardware Prefetching. Morgan & Claypool Publishers, 2014.*

[15] P. Michaud, "*Best-Offset Hardware Prefetching," in High Performance Computer Architecture, pp. 469–480, 2016.*

[16] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz, and D. A. Jimenez,′ "*Perceptron-Based Prefetch Filtering," in 46th International Symposium on Computer Architecture, pp. 1–13, 2019.*

[17] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "*Path Confidence based Lookahead Prefetching," in 49th International Symposium on Microarchitecture, pp. 1–12, 2016.*

[18] J. L. Henning, "*SPEC CPU2006 Benchmark Descriptions," ACM SIGARCH Computer Architecture News, vol. 34, no. 4, pp. 1–17, 2006.*

[19] Y. Ishii, M. Inaba, and K. Hiraki, "*Access Map Pattern Matching for Data Cache Prefetch," in Supercomputing, pp. 499–500, 2009.*

[20] S. H. Pugsley, Z. Chishti, C. Wilkerson, P. Chuang, R. L. Scott, A. Jaleel, S. Lu, K. Chow, and R. Balasubramonian, "*Sandbox Prefetching: Safe Run-time Evaluation of Aggressive Prefetchers," in High Performance Computer Architecture, pp. 626–637, 2014.*

[21] J. Kim, E. Teran, P. V. Gratz, D. A. Jimenez,´ S. H. Pugsley, and C. Wilkerson, "*Kill the Program Counter: Reconstructing Program Behavior in the Processor Cache Hierarchy," in Architectural Support for Programming Languages and Operating Systems, pp. 737–749, 2017.*

[22] R. Bera, A. V. Nori, O. Mutlu, and S. Subramoney, "*DSPatch: Dual Spatial Pattern Prefetcher," in Microarchitecture, pp. 531–544, 2019.*

[23] V. Seshadri, S. Yedkar, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, *" Mitigating Prefetcher-Caused Pollution Using Informed Caching Policies for Prefetched Blocks," ACM Trans. Archit. Code Optim., vol. 11, no. 4, pp. 51:1–51:22, 2015.*

[24] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, *"Domino Temporal Data Prefetcher," in International Symposium on High Per-formance Computer Architecture, pp. 131–142, 2018.*

[25] A. Jain and C. Lin, *"Linearizing Irregular Memory Accesses for Im-proved Correlated Prefetching," in 46th International Symposium on Microarchitecture, p. 247–259, 2013.*

[26] H. Wu, K. Nathella, D. Sunwoo, A. Jain, and C. Lin, *"Efficient Metadata Management for Irregular Data Prefetching," in 46th International Symposium on Computer Architecture, p. 449–461, 2019.*

[27] H. Wu, K. Nathella, J. Pusdesris, D. Sunwoo, A. Jain, and C. Lin, *"Temporal Prefetching Without the Off-Chip Metadata," in 52nd Annual International Symposium on Microarchitecture, p. 996–1008, 2019.*

[28] Online Available: *https://github.com/ChampSim/ChampSim, Champsim Simulator.*

[29] Q. et. al., *"CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping," in MICRO 2018.*

[30] J. Kaur and S. Das, *"A survey on cache timing channel attacks for multicore processors," Journal of Hardware and Systems Security, vol. 5, no. 2, pp. 169–189, 2021.*

[31] J. Kaur and S. Das, *"TPPD: Targeted Pseudo Partitioning based Defence for cross-core covert channel attacks," Journal of Systems Architecture, vol. 135, p. 102805, 2023.*