Reducing Data Movement and Energy in Multilevel Cache Hierarchies without Losing Performance: Can you have it all?

Jiajun Wang*, Prakash Ramrakhyani†, Wendy Elsasser† and Lizy Kurian John*
*University of Texas at Austin. jiajunwang@utexas.edu, ljohn@ece.utexas.edu
†Arm Research. Prakash.Ramrakhyani@arm.com, Wendy.Elsasser@arm.com

Abstract*—Optimizing a multilayer cache hierarchy involves a careful balance of data placement, replacement, promotion, bypassing, prefetching, etc. to capture the various properties of access streams. Often getting good performance involves aggressively orchestrating the movement of the data to be available at the appropriate layers of the cache hierarchy at appropriate times. However, it has been popularly recognized that aggressive movement of data results in high energy consumption. State-of-the-art caching policies such as Hawkeye and MPPPB yield excellent performance but incur more data movement compared to policies such as CHAR, and Flexclusion. Considering the energy cost of data movement, we architect a FILtered Multilevel (FILM) caching policy, which yields good performance with reduced levels of data movement. It achieves this by identifying appropriate cache layers for each block of data using a bloom filter and table based predictors. The bloom filter helps to overcome the challenges associated with capturing PC-based information in exclusive caches in an efficient manner. When there is free space in the bypassed cache layer, FILM overrides the initial prediction and allows cache block installation into the cache level achieving more low latency hits. FILM also incorporates an explicit mechanism for handling prefetches, which allows it to train differently for data from demand requests versus prefetch requests. By incorporating quick detection and correction of stale/incorrect bypass decisions, FILM significantly reduces cache block installations and data movement, resulting in up to 10% reduction in dynamic energy at the LLC and DRAM compared with Hawkeye_EX or MPPPB_EX. Considering energy-delay product as a metric, FILM is 10%, 11%, and 5% better than Hawkeye_EX, MPPPB_EX, and CHAR respectively.

I. Introduction

Performance and power consumption are usually at odds while designing circuits and systems. Microprocessor performance scaling in recent years has been achieved by scaling throughput, i.e. by processing more threads concurrently using increased core counts and by employing techniques like Simultaneous Multithreading (SMT) and SIMD. While performance gains from such scaling can serve the increasing demand on computational power, the number of cores and threads are limited by the restricted power and energy budget.

A large consumer of this energy budget is the memory hierarchy. Often caches account for more than 50% of on-chip die area and consume a significant fraction of static and dynamic power. Therefore, increasing the efficiency of caches becomes crucial. The number of threads (and cores) in modern microprocessor SoCs has been steadily increasing as exemplified in 48 thread Arm based solutions [1] and recently announced 36 thread Intel solutions [2]. This has caused the cache capacities to reach the limits of power and die-area constraints. From the the first generation of Intel Core i7 chip to the most recent Intel Core i9 design, the Last Level Cache (LLC) capacity per core has been held near constant at around 1 to 2MB during the past ten generations of chips. At the capacity limits, caches are often still under-provisioned for data-intensive workloads and under-utilized for cache-insensitive workloads. Regardless of whether the large cache is fully utilized by workloads or not, the energy and power cost is always too high to be ignored.

To improve the performance of data-intensive workloads, prior research has looked at redistributing the available SRAM capacity across the various levels in the cache hierarchy and shows that many emerging workloads benefit from a larger L2 size [3]–[5]. Several recently announced microprocessor products appear to have conformed to this recommendation [6], [7]. Opting for larger L2 sizes however, implies that there would be greater overhead to maintain the inclusive property. In prior work, relaxing the inclusion requirement of LLCs has been shown to be beneficial with 3-12% improvements reported [3], [8]. We also observe that out of all dynamic blocks in SPEC CPU2006 suite, more than 70% never get reused at L2 and more than 50% never get reused at the LLC. In Figure 1, we categorize dynamic cache blocks into four groups based on whether it gets reused in L2 or L3 after it gets evicted out of L1. We see that only 20% of dynamic blocks get reused in both L2 and L3. The remaining 80% of the dynamic blocks have an optimal cache location. Insertion of these blocks into other cache levels uses up critical cache space and consumes energy without bringing any performance benefit. This observation motivates exclusive L2 caches.

While trying to optimize the aforementioned types of cache hierarchies, there are many choices in policies for replacement and/or bypassing. We compare the performance and data movement of three bypass and insertion algorithms for exclusive last level caches, CHAR [5], MPPPB_EX [9] and Hawkeye_EX [10], using workload mixes from SPEC CPU2006 suite. Hawkeye_EX and MPPPB_EX are PC-correlated algorithms and were originally designed for inclusive caches. PC-correlated algorithms are popular and yield high-performance in the field of inclusive or near-inclusive cache design. The PCs used in such algorithms correspond to those instructions that cause the

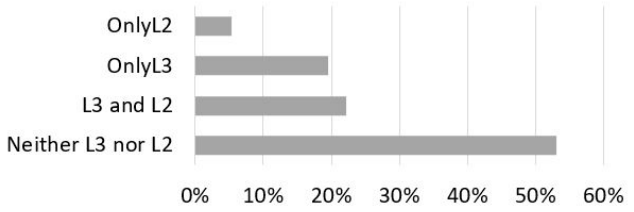


Fig. 1: Percentage of L1 evicted cache blocks getting reused at L2 and L3 in SPEC CPU2006 (average)

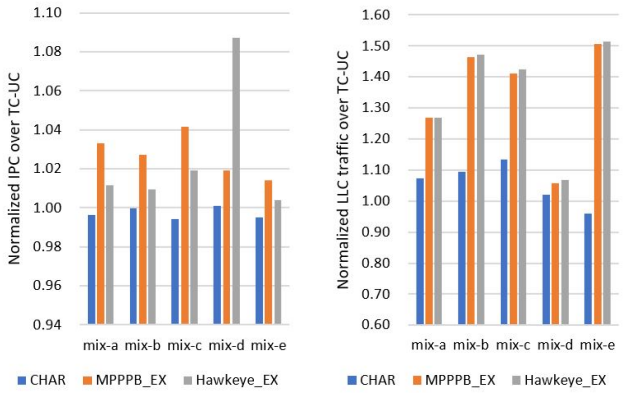
cachelines to be fetched from memory. For inclusive caches this PC also corresponds to the instruction that installs the line in the LLC. It is convenient to maintain this PC signature in just the LLC because in inclusive caches any given cacheline has its longest residence in the LLC. There are no PC-correlated algorithms tailored for exclusive caches because the required PC information is not available in exclusive caches. For exclusive caches, where lines are inserted into the lower level caches upon eviction, the PC information gets lost unless it is passed along with the cacheline across all the levels in the hierarchy. This can lead to inefficient use of space and also further exacerbate the problem of data movement.

Exclusive cache adaptations of the three PC-correlated algorithms mentioned are devised by allowing unlimited hardware overhead to store training data. The comparison result is shown in Figure 2, with IPC and LLC traffic normalized over TC-UC [8]. We see that MPPPB_EX and Hawkeye_EX show better performance compared to CHAR, whereas CHAR generates less LLC traffic than MPPPB_EX and Hawkeye_EX. Specifically, Hawkeye_EX demonstrates a 9% performance improvement compared to CHAR in mix-d by exploiting the data locality of lbm workload, whereas CHAR shows as large as 50% less LLC traffic than Hawkeye_EX and MPPPB_EX in mix-e due to reduced data movement from L2 to LLC (i.e., L2 eviction installed in LLC) of the bwaves workload. The high-performance of MPPPB_EX and Hawkeye_EX are certainly desirable, but the low data traffic of CHAR is also advantageous. Prior art [11] shows that data movement energy is orders-of-magnitude larger than computation energy and the energy dissipation of future systems will be dominated by data movement. Energy-efficient architectures are required to reduce the amount of data movement and exploit data locality to reduce the distance of moving data. The objective of this paper is to create a new algorithm that gives both the high speedup and low data traffic (low data movement and hence low energy).

In this paper, we present a replacement and bypassing algorithm which yields the performance of state-of-the-art caching schemes, but with much reduced data movement, data traffic and energy.

Aimed at addressing these issues and to devise an effective predictor for an efficient and scalable multi-level exclusive cache hierarchy, this paper makes the following contributions:

- FILM, a locality filtering mechanism utilizing a bloom filter and predictors to capture PC-based guidance in a multi-level exclusive cache hierarchy with minimal hardware overhead.
- A method to learn about the correctness of bypass decisions and to adaptively guide data placement into appropriate cache layers based on data reuse patterns.
- Explicit incorporation of prefetch logic, with FILM differentiating prefetch and demand requests and with prefetch aware training/learning of bypass/placement decisions.
- Adopt state-of-the-art work to exclusive caches and show results for multicore multi-programmed system, demonstrating significant energy efficiency improvements and reduction in on-chip data movement. FILM improves overall energy efficiency by 9%, compared to the second highest of 4% from CHAR.



(a) Performance (b) LLC traffic

Fig. 2: Normalized performance and LLC traffic of state of the art caching schemes over TC-UC. mix-a: libquantum, bwaves, gcc, milc; mix-b: GemsFDTD, bwaves, gcc, xalancbmk; mix-c: GemsFDTD, libquantum, leslie3d, xalancbmk; mix-d: lbm, gcc, gobmk, wrf; mix-e: GemsFDTD, bwaves, perlbench, wrf

The rest of this paper is organized as follows. Section II discusses the background of exclusive cache hierarchies. Section III describes the design details of the proposed FILM scheme. We describe our evaluation methodology in Section IV and show the performance and energy result of FILM in Section V. Section VI summarizes prior research in the domain. Finally we summarize our paper in Section VII.

II. Background

While exclusive caches bring high-performance as suggested above, data movement in exclusive hierarchies is different as compared to inclusive hierarchies. In contrast to inclusive hierarchies, only the top level cache of the multi-level exclusive caches is filled with LLC miss data, and the remaining levels serve as victim caches [12], which get filled upon evictions from the upper cache level regardless of cacheline dirty status. If a cacheline in the lower level receives a hit, the cacheline is promoted to the top level and gets invalidated from the

current level to maintain uniqueness. This policy is referred to as "invalidate on hit".

In this work, we use the RRIP [3] optimized for exclusive caches as our replacement policy. The "invalidate on hit policy" poses challenges on replacement policies that are not designed with exclusive hierarchies in mind. For example, the RRIP [13] replacement policy learns re-reference behavior by attaching an RRPV per cacheline. However, such re-reference information is lost as the cacheline is invalidated on hit. To address this challenge, Jaleel et al. [3] presented modifications required for RRIP to be applied to exclusive LLC by adding an SFL3 (Serviced From L3) bit per cacheline and condensing the re-reference information into the SFL3 bit. Specifically, the SFL3 bit is set when a cacheline gets hit at L3. On LLC insertion, if the line was originally served from memory (SFL3 is zero), it is predicted as reuse in the distant future; if the line was originally served from L3 (SFL3 is one), it is predicted as reuse in the near future. This paper extends this idea to both exclusive L2 and exclusive LLC by adding an SFL2 (Serviced From L2) bit. SFL2 and SFL3 are set when a cacheline sees a hit and serves the data request from L2/L3, and are reset when the cacheline is evicted from L2/L3 to make room for new blocks.

III. Design

In this section, we introduce the design of FILM. FILM predicts the reuse of cache blocks at each cache level, and guides evicted cache blocks to insert into the right level rather than trickle down through the various layers in the cache hierarchy.

Figure 3 illustrates how FILM is integrated into the cache hierarchy and how it closely interacts with all levels. Although FILM is a single centralized component, it is not on the critical path. The training process of FILM, which does not require instant feedback, is also off the critical path. FILM's training process is triggered by the three types of cache activities shown in the Figure 3, ① data block installs from main memory to the top level cache due to LLC misses, ② data block hits at lower level caches, and ③ unused data evictions from lower level caches. By leveraging cacheline address and cacheline reuse behavior, FILM trains its prediction model to get the optimal cache insertion decision. On LLC misses (activity ①), apart from receiving training information, FILM also sends L2 and LLC bypass hints to data blocks. The bypass hints are stored along with the cacheline at the cost of 2-bit overhead. The access latency of FILM is orders-of-magnitude lower than the long DRAM access latency and thus can be overlapped with DRAM access latency.

FILM is composed of two hardware structures, a *Prediction Learning Table* and a *Result Table*. The Prediction Learning Table learns data locality of an individual memory instruction by observing the history of past data accesses, and uses this learning to make bypass decisions for future accesses. FILM uses a centralized structure to store the locality of memory instructions. This centralized structure is more space-efficient in comparison to holding the memory instruction locality

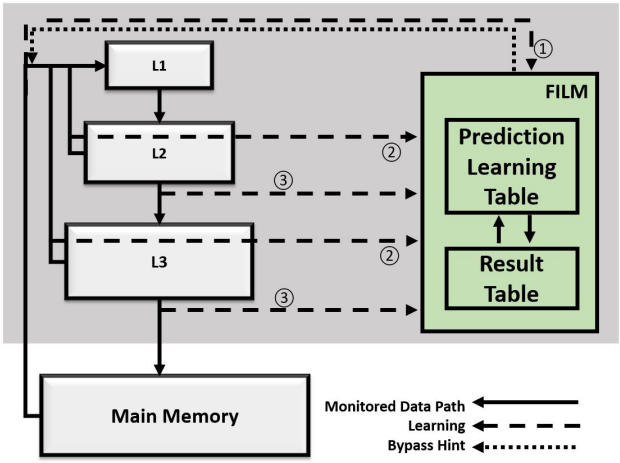


Fig. 3: Overview of the proposed FILM system

along with cachelines in all the levels of the cache hierarchy. FILM achieves good learning accuracy with a reasonably small number (16) of entries in this table.

To avoid information loss, when a trained entry is evicted from the Prediction Learning Table, its learning is captured in the Result Table to inform future bypass decisions. The Result Table also provides an initial value for the Prediction Learning Table when an instruction is reallocated back to the table. The Result Table is indexed using a hash of the memory instruction PC. This hash is stored in the *Tag* field of the Prediction Learning table. Each entry in the Result Table is just 2 bits, much smaller than Prediction Learning Table Entry. Therefore, we can maintain a lot of (2048) entries in this table. Thus, with judicious allocation of available resources, we combine the benefits of a wide but shallow learning table and a narrow but deep result table. This helps with optimizing the solution while maintaining a tight overall hardware budget. FILM can be applied together with other cache replacement policies as FILM only provides bypass/insertion hints.

FILM adapts PC-correlated locality filtering approaches for exclusive cache hierarchies. We select PC as the training heuristic because we observe a good correlation between a memory instruction and the locality of the data accessed by the instruction. Figure 4 shows that the majority of active instructions which make intensive data requests in the SPEC CPU2006 workloads have stable data locality behavior at L2 and L3 of exclusive caches. We define a memory instruction to have stable data locality at a specific cache level if more than 90% of data blocks accessed by the instruction are within the same range of reuse distance (e.g., always hit or always miss in the cache). This observation suggests that if historical data accesses made by an instruction do not benefit from caching at a given level, then future accesses from the same instruction can bypass that cache. In section III-A we discuss how this intuition is applied to train FILM for demand requests. Although PC-correlated algorithms have been proposed in prior work [9], [10], [14], they focus on inclusive cache hierarchies, where the

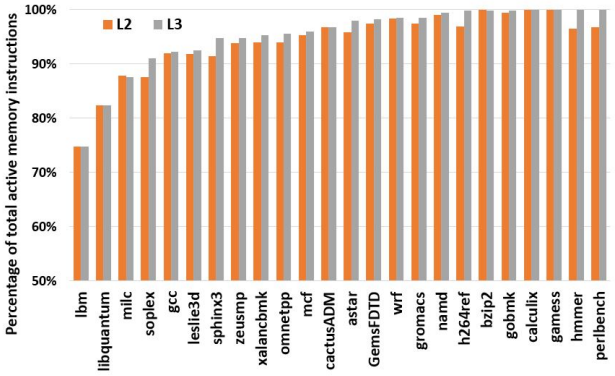


Fig. 4: Percentage of memory instructions with stable data locality

memory instruction information is available during the data block insertion. A direct adaptation of prior art requires storing memory instruction PC along with cachelines, which introduces a significant amount of storage overhead. We present how FILM overcomes this challenge of building the one-to-one relationship between a data block and its instruction using bloom filters in the section III-A2. In section III-B we present how we can train FILM for prefetch requests by adding only one entry per prefetcher. Further, we bring out that these prefetcher entries do not even require a bloom filter.

A. Handling demand request

FILM leverages the observation that data blocks touched by the same memory instruction tend to have similar caching behavior. Thus, by learning the caching behavior of a memory instruction through access history, the caching behavior of future data blocks from the same instruction can be predicted. Building this one-to-one relationship between a data block and its instruction is challenging for exclusive cache hierarchies. Storing the PC information in the cache along with the tag/data blocks for a sampled set (training set) is not viable for exclusive caches as the overhead for just a sampled set can be dramatic. We end up having to store the PC information across all the levels in the hierarchy. This is because every cache-line in an exclusive cache hierarchy has only one unique location across all the cache levels. If we store the PC information in just the last level, we lose the ability to track reuse behavior when a cache-line is promoted to the upper levels (L1, L2). Further, unless the PC information is stored in the L1 when the line is first brought in from memory, it will not be available when the cache-line eventually reaches the exclusive LLC. Increasing the size of L1 tag structures to store additional meta-data will likely have effects on its cycle-time and therefore adversely impact performance. Additionally, ferrying the bypass related meta-data along with data across the various levels in the hierarchy further impacts the critical path and worsens the problem of data movement. To circumvent these challenges FILM employs centralized training and inference structures that are off the critical path of the cache hierarchy. We describe the key components of FILM in the subsections below.

1) ***Prediction Learning Table****:* The Prediction Learning Table forms the core of FILM's bypass mechanism. It is a multiported table based training structure, where each entry corresponds to one memory instruction. FILM selects data blocks mapped to a few LLC sampled sets as its training set. Once FILM is able to retrieve the memory instruction information of a training data block, it trains bypass heuristics for this memory instruction at all cache levels except for L1. Due to constrained training storage budget, the Prediction Learning Table keeps track of a limited number (e.g.,16) of memory instructions. When reaching the Prediction Learning Table entry limit, the instruction with the least frequent memory accesses would be evicted to make room for new instruction.

As illustrated in Figure 5, each table entry contains a *Tag* field which is a hash of the memory instruction PC and is used to index the Result Table, a *Footprint Container*, *ReuseCnt* fields for L2 and L3 respectively, and *Fill* fields to record current L2 and L3 insertion decisions. The footprint container represents the cache footprint of the associated memory instruction. An entry in this table learns the reuse behavior for cache-lines fetched from DRAM by the memory instruction associated with the entry. As cachelines move across the hierarchy, we capture their re-use behavior to the learning table entry that is identified by the footprint container of the entry. The ReuseCnt fields are used to track the number of cache reuses encountered by the data blocks fetched by one memory instruction. The ReuseCnt fields are fixed width saturating counters(e.g., 3-bit), which get incremented on data hit, or get decremented on unused data eviction. *L2Fill* and *L3Fill* fields record the latest training result, a bypass/insert hint. They are initialized according to the value in the Result Table. An L2ReuseCnt/L3ReuseCnt value reaching the maximum value triggers the L2Fill/L3Fill field to change to "Insert", whereas value decreasing to zero triggers changes to "Bypass". When a new entry is allocated, the initial value of a ReuseCnt field is set based on the status of the corresponding Fill hint, i.e., set to maximum value if the Fill hint is "Insert" and to zero if the Fill hint is "Bypass". The initial status of a Fill hint is determined by the Result Table, which will be introduced in Section III-A4. Before training starts, all entries in the Result table are initialized to 'Insert". Therefore, for a first-time trained memory instruction, its Fill hints are initialized to "Insert" and the ReuseCnt fields are initialized to the maximum value.

2) ***Footprint Container****:* Each entry in the Prediction Learning table represents a single memory instruction but needs to train on all the cachelines the instruction brings in to the cache hierarchy. Storing the addresses of all the corresponding cache lines in the footprint container of entry can make the Prediction Learning Table prohibitively large. Therefore, how to efficiently associate data blocks fetched by a memory instruction with its corresponding entry in the Prediction Learning Table is a crucial problem. To address this challenge, FILM applies a bloom filter [15], a space-efficient probabilistic data structure which can rapidly determine whether a data element belongs to a data set or not. Every Prediction Learning table entry is assigned to a separate bloom filter. In our work, we use the

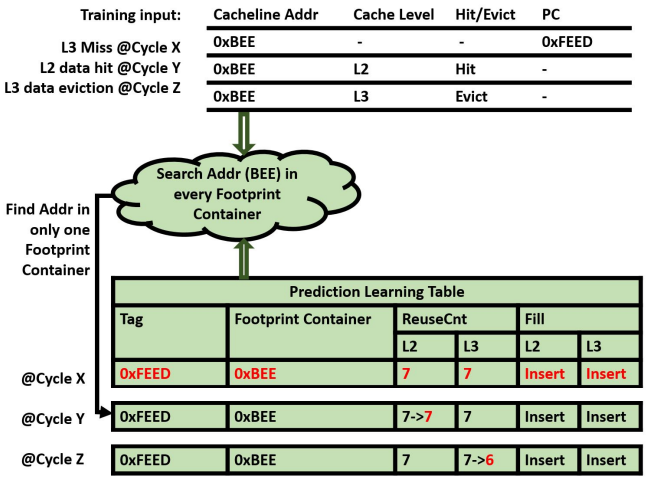


Fig. 5: Training of FILM on demand-fetched blocks. One Prediction Learning Table entry update at three different cycles.

most basic design of a bloom filter, which is in the form of a bit vector. The bit vector of bloom filter has a fixed size (e.g., 4098 bits).

To add an element to the bloom filter, the element is hashed a few times, and the bits in the bit vector at the index of those hashes are set to 1. To test for membership, an element is hashed with the same hash functions. The element is not in the set if any value at index bits is not set, otherwise it could be in the set. Since there is no way to delete an element from the bloom filter, the chance of bloom filter reporting false positive membership increases as the number of inserted elements grows. Therefore, we reset the bloom filter periodically, after every 256 insertions, to maintain a low false positive rate of 1%. We name the bloom filter the *footprint container*, as what it records is essentially the memory footprint of an instruction.

*3) **Learning Process**:* To train the prediction model, FILM requires information including cacheline address, the level from which the data block gets hit or evicted without reuse, and the hashed PC of the memory instruction if it is an LLC miss. FILM leverages SFL2 and SFL3 bits to indicate whether a data block has been reused during its stay at L2 and L3. Jaleel et al. presented how these bits are required for RRIP if it has to be applied to exclusive caches. Details on RRIP's adaptation for exclusive caches are included in the Section II. The hashed PC is used to find the right entry in the Prediction Learning Table, and is stored in the Tag field if a new table entry is allocated. As described in Figure 3, FILM's learning process is triggered by the three types of cache activities: LLC misses, cache hits and unused data evictions.

On LLC misses, the data address is shifted by the size of a cacheline to form a cacheline address, and the cacheline address is inserted into the bloom filter. For example, as shown in Figure 5, an LLC miss to data 0xBEE happens at cycle X. The PC (0xFEED) of the memory instruction and the cacheline address (0xBEE) are sent to FILM's Prediction Learning Table. A new table entry is allocated to the memory instruction. PC (0xFEED) is stored in the Tag field. Fill hints are set based on the Result table, which are all "Insert" in this example. ReuseCnt fields are initialized based on the Fill hints. The cacheline address (0xBEE) is inserted into the Footprint Container of the table entry.

On training events triggered by unused data eviction or cache hits, FILM retrieves the PC of the memory instruction which initially fetched the data from DRAM by looking for the cacheline address among all the footprint containers in a time multiplexed fashion. The searching process can be pipelined and is not on the critical path. If a single membership is reported for a cacheline address, then FILM constructs the one to one mapping between the data and the instruction. The data locality of the memory instruction is learned based on data reuse information. Alternatively, training activities are not performed for the following two situations: one is when no residency in the Prediction Learning Table is detected, which is possible because FILM tracks a limited number of memory instructions and footprint containers get reset periodically; and the other situation is one when the address is found in more than one bloom filter due to false positive membership reporting. In the latter case, FILM decides not to train to avoid training noise. Figure 5 illustrates Prediction Learning Table entry updates due to L2 data hits at cycle Y and LLC unused evictions at cycle Z respectively. After searching through all the footprint containers, cacheline address 0xBEE, which has previously been added to the Footprint Container of the instruction 0xFEED during cycle X, is reported as single membership in table entry 0xFEED. The ReuseCnt number of this table entry is updated based on the data reuse information. The Fill hints stay same as no new threshold has been reached.

*4) **Result Table**:* Upon LLC miss, data blocks consult FILM about whether to bypass L2 or LLC in the future. The Prediction Learning Table is the first-hand source of bypass hints when a PC match is found. However, if there is no PC match, FILM relies on the Result Table to handle cases when data blocks cannot receive bypass hints from the Prediction Learning Table. The Result Table is a direct-mapped structure indexed by the hashed instruction pointer. Each table entry has two bits, representing L2 and L3 bypass hints separately, and their initial value are set to be "Insert". When there is an LLC demand miss, the PC of the demand request is used to index the Result Table and read the L2 and L3 fill decision. Once the data block is installed directly into L1, the decision is kept with the data block along with other metadata. The 2 bit overhead per cacheline is acceptable, and it helps guide data insertion as a complement to the Prediction Learning Table. Another important function of the Result Table is to provide initial Fill value for a newly allocated Prediction Learning Table entry. When a Prediction Learning Table entry is evicted, the trained bypass hints of the instruction are stored into its corresponding Result Table entry, and such that next time when this instruction gets reallocated to the Prediction Learning Table, it has warmed-up bypass hints.

*5) **Detect stale bypass decisions using empty blocks**:* The optimal bypass hints dynamically change along with the

program execution. The reason is because cache bypass of one group of data blocks changes the reuse distance of other groups. A previous bypass decision becomes stale and does not work in the future as the reuse distance profile changes dynamically. One example is the case when cache accesses exhibit a thrashing access pattern, e.g., a memory instruction repetitively reading K data blocks which happen to map to the same set of an N-way associative ($N < K$) cache. The optimal solution is to keep N data blocks in the cache and bypass the rest ($K - N$). An algorithm without error detection will predict that none of the future data blocks from the same instruction should be inserted into the cache. Whereas an optimal algorithm should allow at least N data blocks to be inserted to guarantee data reuse at the best effort.

A stale bypass hint is difficult to detect because the data block following the bypass hint is discarded, leaving no chance to prove its locality from cache hits. Thus, FILM is designed with a "utilize empty blocks" rule to provide opportunities to detect stale bypass decisions. The rule is explained as follows. Let us consider the L2 cache as an example. A data block is inserted into L2 cache due to available free space even though FILM suggests to bypass L2. The bypass L2 hint is stored along with this block. On a subsequent hit to the same block at L2, FILM trains its model and considers the L2 "Bypass" hint as stale after seeing that a block marked as bypass gets reused. In addition to increasing the L2ReuseCnt counter, FILM would immediately flip the L2 fill hint from "Bypass" to "Insert" based on the single error. Prior art either does not have an error detection scheme and always performs data bypassing based on prediction, or inserts blocks if there is free space cache without any further activities on error detection.

Although the "utilize empty blocks" rule could cause useless data block insertions (given that FILM suggests bypass), we argue that it does not cause additional performance degradation due to two reasons. One is that it does not pollute caches as it uses free cache space without causing any eviction. The other reason is that high performance cache replacement policy protects cache blocks with frequent reuse and selects cache blocks with less or no reuse as the victim, such that wasted insertions from the "utilize empty blocks" rule are evicted to make room for new blocks.

B. Handling Prefetch

FILM's training on prefetched blocks is performed at the granularity of a prefetcher. For example, for a system with L1 and L2 prefetchers, FILM sets up two entries in the Prefetch Prediction Learning Table, with each entry representing one prefetcher. The footprint container field is not required, because each cacheline can pinpoint which prefetcher initially fetched the block by storing prefetch identifier information (*PfId*) in the tag store. Cache blocks fetched by the same prefetcher will have the same PfId. Prefetch identifier helps to distinguish prefetched blocks from regular blocks (whose PfId is zero). When a prefetched block serves a demand request, it is promoted from a prefetched block to a regular block and the PfId is reset to zero. The future training process on this block is handled the

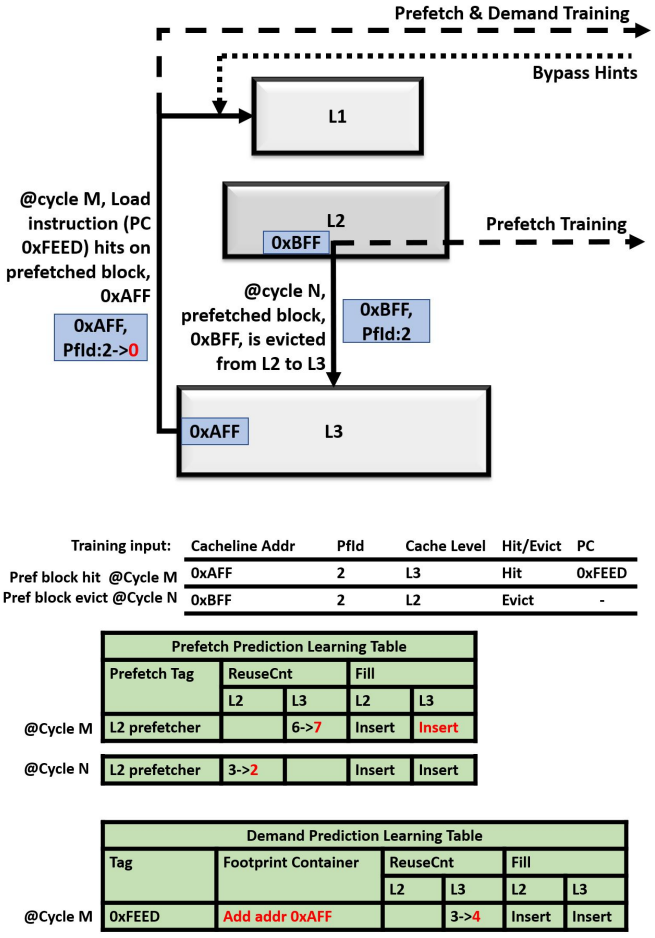


Fig. 6: Training on prefetched blocks. Showing two different scenarios at two different cycle.

same as a regular block. Note that a prefetched block serving prefetch requests from upper levels does not change the PfId value of the block. Basically, the prefetch identifier of a cache block provides three hints to FILM. Firstly, it indicates whether the training target is a regular memory instruction or certain data prefetcher. Secondly, if the target is a data prefetcher, PfId points to the prefetcher for which FILM is to be trained. Thirdly, PfId being non-zero indicates that a prefetched block has not served a demand request yet.

Training on prefetch requests occurs when a prefetched block is hit by either prefetch requests or demand requests, and when a prefetched block is evicted. When a prefetched block sees a demand request hit, FILM performs three operations. Firstly, the L2ReuseCnt or L3ReuseCnt of the corresponding prefetcher is incremented based on whether the hit occurs at L2 or L3. Secondly, the cacheline address of the hit block is added to the footprint container of the demand request, in preparation for future training on this instruction. The ReuseCnt of the Demand Prediction Learning Table entry also increments. PfId of this cache block is reset. Thirdly, the demand request PC is used to

consult the Prediction Learning Table and the Result Table to get suggested L2 and L3 bypass hints, which is sent back to the data. When a prefetched block serves a prefetch request (hit), the L2ReuseCnt/L3ReuseCnt of the corresponding prefetcher is incremented. When a regular block serves a prefetch request (hit), the regular block is demoted to a prefetched block and has its PfId set, but there is no Prediction Learning Table updates. When an unused prefetch block is evicted, the L2ReuseCnt or L3ReuseCnt of the corresponding prefetcher is decremented. Making L2/L3 bypass decisions and correcting stale bypass decisions are the same as handling demand requests. Figure 6 illustrates the above operation using two different examples. In the first example at cycle M, a data block prefetched by L2 services a demand request at L3. In the second example at cycle N, a prefetched block initiated by L2 prefetcher gets evicted out of L2 without usage. For this case, the prefetched block is inserted into the L3 based on the Insert L3 bypass hint from the Prefetch Prediction Table, but the Demand Prediction Table is not updated.

IV. Evaluation Methodology

Our simulations are performed on a cycle-accurate simulator, which is an extended version of the 2nd Cache Replacement Champion (CRC2) simulator [16]. Table I shows detailed simulator parameters. The memory subsystem consists of a three-level cache hierarchy and a detailed DRAM model. Evaluations are conducted on multicore systems with prefetching enabled. We apply both traditional data prefetcher designs such as next-line prefetchers, as well as one of the state-of-the-art prefetching schemes, VLDP [17]. We use Simpoint traces of SPEC CPU2006 workloads provided by the 2nd Cache Replacement Contest [16]. In multi-core experiments, cores that finish executing early would restart execution from beginning in order to continue adding pressure to shared cache and memory. In the multi-core experiments, each core runs 250M instructions with a warm up length of 10M instructions. We use McPAT [18] to estimate the dynamic power and energy consumed by the various policies. The system energy reported in this paper includes core energy, fabric energy, shared LLC energy and DRAM energy.

We compare FILM design against seven cache replacement and bypass algorithms. **TC-UC** [8] and **DRRIP_EX** [3] learn global caching priorities for exclusive caches. Both policies use a three-bit re-reference counter per cacheline. TC-UC is implemented with bypass and aging policies, which corresponds to the "Bypass+TC_UC_AGE_x8" policy in their paper [8]. Both **FLEXclusion** [19] and **CHAR** [5] focus on reducing on-chip multi-level cache bandwidth via relaxing cache inclusion policy. For CHAR we use the address space correlation scheme and implement the "CHAR-C4" policy which is tailored for an exclusive cache model and does not de-allocate a block from LLC on hit. For FLEXclusion, it operates in both Aggressive and Bypass mode. While the above four schemes are address space correlated, we also include code space correlated schemes, namely **SHiP++_EX** [20], **Hawkeye_EX** [10], and **MPPPB_EX** [9]. We have adopted these schemes to exclusive caches by storing the instruction pointer information along with data block and tailored their RRIP-based replacement policies for an exclusive cache model based on Jaleel's prior work [3]. The implementations are based on the code submitted by the respective authors to the CRC2. Specifically, the implementation of SHiP++_EX is based on SHiP++ [20], which further improves the performance of the SHiP policy.

TABLE I: Simulation parameters

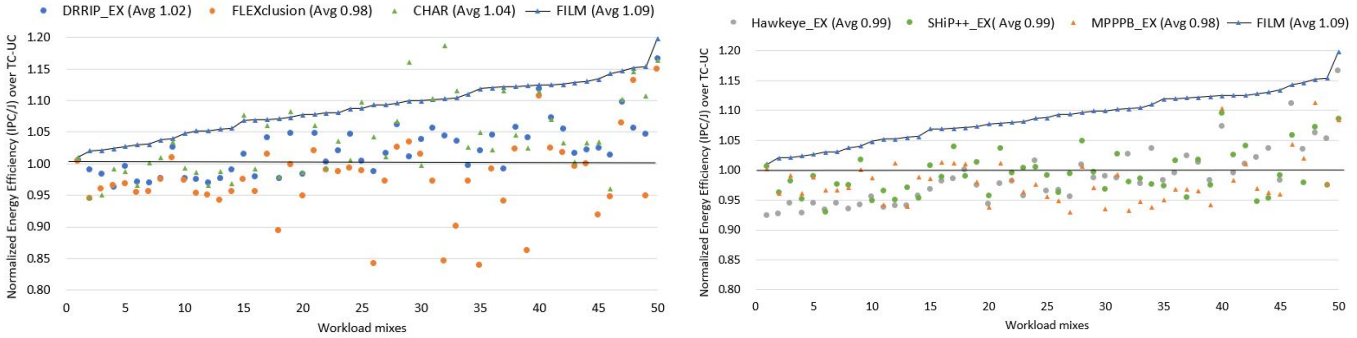
Four cores	out-of-order cores, 4.5GHz, 6-wide pipeline, 72-entry load queue, 56 entry store queue maximum 2 loads and 1 stores be issued every cycle
Branch Predictor	bimodal branch prediction, 16384 entries, 20 cycle mispredict latency
Private L1	64KB, 8-way associative, 8 MSHR entries RRIP replacement policy, nextline prefetcher, 4 cycle latency
Private L2	512KB, 8-way associative, 16 MSHR entries RRIP replacement policy, VLDP prefetcher, additional 8 cycle latency
Shared LLC	8MB, 16-way associative, 32 MSHR entries RRIP replacement policy additional 20 cycle latency
DRAM	4GB off-chip memory. 1 channel. 1600 MT/s Read queue length 48 per channel Write queue length 48 per channel tRP = 11 cycle, tRCD = 11 cycle, tCAS = 11 cycle

V. Results

A. Multicore Evaluation

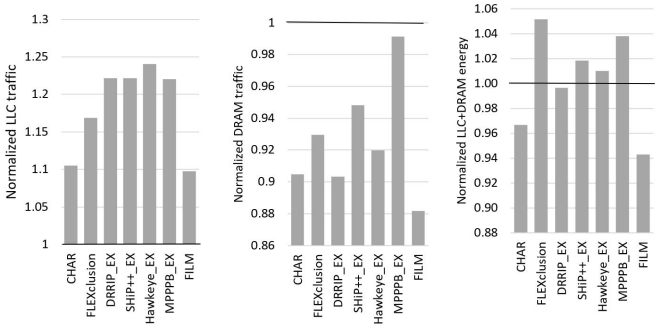
We evaluate FILM on a series of 4-core multi-programmed workloads, with a wide variation in the data reuse characteristics and cache capacity sensitivity of the co-running programs. The workload mixes are made of both streaming-oriented workloads and reuse-oriented workloads. In the following sections, we evaluate FILM and other policies from the perspective of energy efficiency, data movement and performance. An early finished workload continues executing and stressing shared resources(e.g., LLC and main memory) until the slowest one completes, however, the energy and performance of a workload is computed based on the data of first 250 million instructions. We use throughput (i.e., total IPC) over entire system energy as the metric to demonstrate energy efficiency. LLC accesses and DRAM accesses are used as metrics to evaluate the amount of data movement controlled by the evaluated policies. LLC accesses (LLC traffic) consists of all kinds of LLC accesses, including load/store access, prefetch requests and L2 evictions. DRAM accesses (DRAM traffic) consists of all the LLC misses. IPC speedup is used to summarize the performance impact of a policy on multicore workloads.

1) Energy Efficiency: Figure 7 compares the energy efficiency of CHAR, FLEXclusion, DRRIP_EX, SHiP++_EX, Hawkeye_EX and MPPPB_EX, with the number normalized to the baseline TC-UC. The mixes are arranged in the increasing order of FILM's normalized energy efficiency. The comparisons are demonstrated in two panels based on whether the policies were originally proposed to handle exclusive caches or not. The policies in the first class (CHAR, FLEXclusion, DRRIP_EX) as well as the baseline(TC-UC) are address-correlated. In contrast,



(a) (b)

Fig. 7: Energy efficiency(IPC/J) of FILM and other schemes. Results normalized to TC-UC. FILM (line with blue triangle) constructs the upper envelope.



(a) LLC traffic (b) DRAM traffic (c) LLC+DRAM energy

Fig. 8: The traffic and energy of shared memory resource (LLC and DRAM) of FILM and other schemes. Results normalized to TC-UC. The lower the better.

all the policies in the second class (SHiP++_EX, Hawkeye_EX and MPPPB_EX) are PC-correlated and require maintaining the program counter of the load/store instructions along with the cacheline. Additionally, CHAR and FLEXclusion, from the first group, and the baseline TC-UC policy have L2 eviction bypassing LLC mode, whereas DRRIP_EX and policies in the second group do not enable data bypassing.

Our first observation is that FILM constantly achieves higher energy efficiency than the baseline whereas the profile of other policies fluctuates dramatically. Compared to the baseline, the energy efficiency of FILM varies from 1% to a gain of 20%, whereas FLEXclusion shows the largest swing between a loss of 8% to a gain of 16%. Our second observation is that FILM has the highest average energy efficiency improvement of 9%, beating the second largest value of 4% from CHAR by 5%. In other words, given the same amount of energy supply FILM is able to achieve 9% higher performance than the baseline, compared to the range of -2% to 4% performance improvement from other policies.

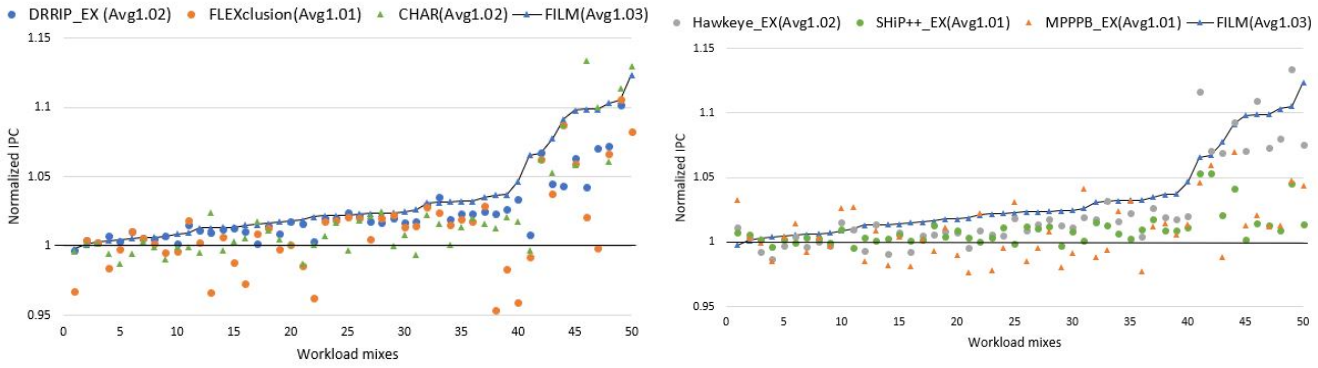
2) Data Movement: Data movement is a major factor contributing to the energy consumption difference among all the policies as it affects both LLC energy and DRAM energy. In order to understand why one policy consumes more/less energy than another, we summarize the average normalized LLC and DRAM traffic as well as the total energy of the two shared memory resources of all the workload mixes in Figure 8a. From the figure, we observe that FILM consumes the least amount of shared memory energy compared to other schemes because it generates the smallest average number of LLC and DRAM traffic. It is also noted that all the evaluated policies introduce more LLC traffic compared to the baseline policy with fewer DRAM accesses. The reason is that the baseline policy performs LLC bypass more aggressively compared to the other schemes. Aggressive LLC bypassing helps reduce LLC energy, but results in wasting more DRAM energy due to the increasing LLC misses.

Comparing to the policies with no LLC bypassing (DRRIP_EX, SHiP++_EX, Hawkeye_EX and MPPPB_EX), FILM saves LLC traffic by selectively installing L2 evictions into LLC. The average normalized LLC traffic for no-bypassing policies are around 1.22X, which is 0.12X more than the 1.1X of FILM. The LLC bypass rate of FILM varies between 1% to 46% (with median value of 20%), which account for up to 0.3X less LLC traffic compared to no-bypassing policies.

Compared with data-bypassing policies, FILM has similar average LLC traffic compared to CHAR and 0.7X less than FLEXclusion, and FILM has the lowest DRAM traffic(0.88X) compared to CHAR(0.9X) and FLEXclusion(0.93X).

3) Performance: Figure 9 summarizes the performance speedup of various algorithms normalized to the baseline TC-UC. The mixes are arranged in the increasing order of FILM's normalized throughput. The average performance of FILM is generally better than FLEXclusion and SHiP++, and looks on par with other schemes. Specifically, FILM outperforms DRRIP_EX on workload mixes with lbm and sphinx3. FILM beats DRRIP_EX in lbm and sphinx3 in terms of single core performance by more than 10%. FILM shows its performance advantage on cache capacity sensitive workloads by increasing the effective cache capacity via reduced insertions, minimizing shared cache capacity contention in the multicore scenario.

Among all the four PC-correlated policies, SHiP++_EX shares the most common thoughts with FILM. The largest



(a) (b)

Fig. 9: IPC of FILM and other schemes. Results normalized to TC-UC. FILM constructs the upper envelope.

TABLE II: Comparison between FILM and basic RAP and RAP enhanced with FILM-like training on prefetch relative to TC-UC

Metric	Basic RAP	Enhanced RAP	FILM
Hardware overhead per core	72KB	72KB	8.5KB
Energy efficiency (IPC/J)	0.91	0.96	1
DRAM traffic (access count)	2	1.06	1
LLC traffic (access count)	1.38	1	1
Performance (IPC)	0.89	0.98	1

TABLE III: FILM hardware budget (per core)

Component	Parameter	Budget
Prediction Learning Table (Demand + Prefetch)	16 entries, 11-bit Tag, 4096-bit footprint container 8-bit ReuseCnt+Fill	8 KB
Result Table	2048 entries, 2-bit entry	0.5KB

TABLE IV: Overhead Comparison (per core)

CHAR	SHiP++_EX	Hawkeye_EX	MPPPB_EX	FILM
2.25KB	77.5KB	86KB	97KB	8.5KB

difference is that FILM relies on bypassing dead blocks to avoid triggering the replacement policy and protecting critical data, whereas SHiP++_EX (as well as Hawkeye_EX and MPPPB_EX) always inserts dead blocks with the highest eviction priority. Take the mix50 in the Figure 9b as an example, FILM outperforms SHiP++_EX, Hawkeye_EX and MPPPB_EX in this workload mix, which consists of one cache capacity sensitive workload (sphinx3), one streaming workload (libquantum), and two workloads with small working set size (astar and wrf). FILM distills the streaming pattern and minimizes the LLC data replacement due to this workload, thus increasing the LLC hit rate of the data with reuse, which is retained in the cache. Another difference is that FILM detects any stale bypass decisions and updates its prediction model, whereas SHiP++_EX does not do any error detection or correction. There is no obvious performance winner among CHAR, Hawkeye_EX, MPPPB_EX and FILM. Hawkeye_EX outperforms FILM on a few workload mixes, and its multicore throughput speedup comes from its performance improvement on one workload, lbm. However, even for the workload mixes with 8% performance difference between FILM and Hawkeye_EX, FILM achieves the same energy efficiency as Hawkeye_EX as FILM generates 25% less LLC traffic.

Co-running application could experience IPC reduction due to the negative interference between the multiple tenants contending for shared cache resources. Compared to the baseline, FILM is able to restrict the performance degradation of single application within 2%, whereas other policies lead to single application performance degradation of 4% to 10%.

4) Comparison with RAP: As both FILM and RAP [21] have the same goal of optimizing data placement across the cache hierarchy, we complete our evaluation with RAP comparison in this section. RAP is compared separately because it does not have a special training mechanism for prefetched blocks. With prefetching disabled, FILM's performance on the multi-programmed mixes beats RAP by 7%. With prefetching enabled, the performance of the original version of RAP on the system with prefetching enabled is poor, because RAP uses PC as its training metric and the PC of a prefetched block is zero (prefetcher does not have PC). To make fair comparison, we enhance RAP with FILM-like training schemes on prefetch and show the result of the enhanced RAP in Table II. One difference between RAP and FILM is that for cachelines with frequent accesses(hits), RAP learns from only the first hit and evictions, whereas FILM learns from all the hits and evictions. Another difference is that to avoid losing a global view of data movement under heavily data bypassing, RAP dedicates few sets as learning sets which are not affected by the RAP bypass algorithm, whereas FILM follows its "utilize empty line" rule. Enhanced RAP has a 72KB hardware cost as it extends the metadata field of each cacheline in the cache subsystem with 12-bit PC. With such significant overhead, the energy efficiency of Enhanced RAP is 4% less than FILM. Both Enhanced RAP and FILM cut down LLC traffic due to their support on cache level bypassing. RAP has 6% more DRAM accesses and 2% less performance compared with FILM as its bypassing tends to get overly aggressive.

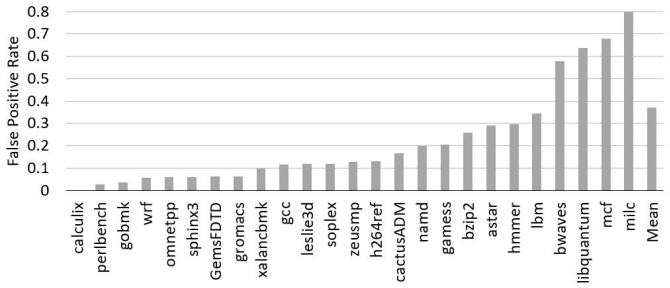


Fig. 10: Rate of multiple entry matches reported by FILM's bloom filter.

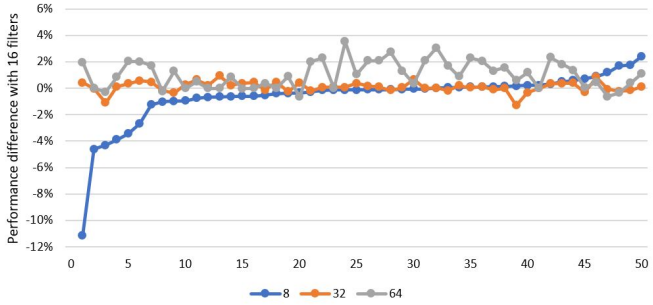


Fig. 11: Performance sensitivity to the number of bloom filters. IPC normalized to 16 bloom filters.

B. Hardware Cost and Design Decisions

1) Hardware overhead comparison: Table III shows the hardware budget of FILM's two main memory components, Prediction Learning Table and Result Table. FILM's total hardware budget is 8.5KB per core, which is 0.3% more SRAM than the three-level cache hierarchy, in exchange for a significant reduction in data movement and a dramatic improvement on energy efficiency. Table IV compares the hardware budgets for the evaluated replacement policies. TC-UC and DRRIP_EX are not listed because they, as well as other six schemes, share the common overhead of three bits per cacheline to store re-reference counter. FLEXclusion is not listed because it leverages pre-existing data paths with only four registers overhead. SHiP++_EX, Hawkeye_EX and MPPPB_EX have 72KB more overhead than the number claimed in their paper due to the overhead of 14-bit PC-based signature stored with cacheline.

2) Bloom filter analysis: As mentioned earlier, we use a bloom filter in FILM, to tie all cache blocks fetched from DRAM by the same memory instruction to one table entry. When training, we use this bloom filter to identify which specific memory instruction to train. FILM stops training on a data address when a multiple match is detected. Figure 10 illustrates the false positive rate at which FILM's bloom filter reports multiple entries matched one address among SPEC CPU2006 workloads. We see that the false positive rate is greater than 10% for more than half of the workloads. One may be surprised that FILM's training accuracy is not seriously impacted even with such high rates of multiple matches. We observe that for workloads like libquantum and bwaves, the valid training points are only around 30% of the entire training set, and this observation suggests high information redundancy in the training set. In other words, although we do not train FILM based on the entire history of data accesses generated by a given instruction, a small portion of the history provides sufficient information to train a decision that is as good as the one made with all the history. Further, this observation adds confidence to our design choice of using a few sampled cache sets for training FILM, as opposed to tracking all the sets.

Figure 11 illustrates the impact on performance as the number of bloom filters in the Prediction Learning Table increases from 8 to 64. The performance number is normalised to 16 entries, which is the same number showing in Table III.

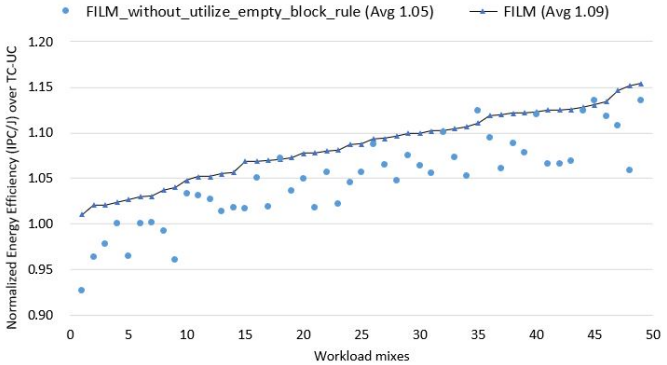


Fig. 12: Energy efficiency of FILM and FILM without "utilize empty block rule". Results normalized to TC-UC. FILM is at upper envelope.

We observe a huge performance jump when the number of bloom filters increases from 8 to 16. Performance difference between 16 and 32 bloom filters is negligible. Performance increases by 2% as the number quadruples from 16 to 64. Thus, the least amount we could possible use to maintain performance is 16.

3) Impact of "Utilize empty blocks" rule: The "Utilize empty blocks" rule inserts evicted cache blocks into lower caches when there is empty space in the cache, regardless of FILM's bypassing hints. If such blocks see hits in their new home, it guides FILM to dynamically adjust its outdated bypass decisions. The CHAR algorithm uses empty block as well. However, FILM uses empty blocks to create opportunities for detecting stale bypassing hints, whereas CHAR does not perform any special training on data inserted into empty blocks. Figure 12 compares normalized energy efficiency over TC-UC between FILM and another FILM implementation which does not apply the "Utilize empty blocks" rule and always bypasses data block based on hints. Always bypassing reduces the number of LLC installs and saves cache energy, at the cost of losing performance and increasing DRAM energy. Figure 12 illustrates that "utilize empty block" rule contributes to an average of 4% energy efficiency improvement compared to TC-UC.

VI. Related Work

There have been several studies on intelligent cacheline bypass/placement. A group of researchers have studied the energy and performance impact of cache bypass on the first level cache [22]–[26], while another group of researchers focus on the last level [27]–[36]. All these techniques only target single level of cache without addressing the problem from the perspective of the entire cache hierarchy.

Gaur et al. explore insertion and bypass algorithms for exclusive LLCs and propose a number of design choices for selective bypassing and insertion age assignment (TC-UC) [8]. LLC bypass and age assignment decisions are based on two properties of a block, trip count and use count.

Chaudhuri et al. propose CHAR [5], cache hierarchy-aware replacement algorithms for inclusive LLCs and applies the same algorithms to implement efficient bypass techniques for exclusive LLCs in a three-level hierarchy. The CHAR algorithm learns the reuse pattern of the blocks residing in the L2 cache to generate selective replacement hints to the LLC.

Sim et al. propose FLEXclusion [19] which dynamically switches between exclusion and non-inclusion depending on workload behavior. While FLEXclusion dynamically changes between the exclusive and non-inclusive to get the benefit of high performance of exclusive caches and low data traffic of non-inclusive caches, FILM's goal is to improve exclusive caches performance and reduce data traffic by learning bypass hints.

Wu et al. propose Signature based Hit Predictor (SHiP) [14], a sophisticated cache insertion mechanism. SHiP predicts whether the incoming cache line will receive a future hit by correlating the re-reference behavior of a cache line with a unique signature, such as memory region, program counter, or instruction sequence history based signatures. The SHiP implementation compared in our work uses program counter as the signature.

Jain et al. propose Hawkeye [10], a cache replacement policy which learns from Belady's algorithm by applying it to past cache accesses to inform future cache replacement decisions. Hawkeye is consisted of an OPTgen algorithm which uses the notion of liveness intervals to reconstruct Belady's optimal solution for past long cache accesses, and a predictor which learns OPT's behavior of past PCs to inform eviction decisions for future loads by the same PCs.

Jimenez et al. propose Multiperspective Placement, Promotion, and Bypass (MPPPB) [9], a technique that predicts the future reuse of cache blocks using seven different types of features to capture various program properties and memory behavior. The set of features used in MPPPB include data address, last miss, offset and program counter. Its predictor is organized as a hashed perceptron predictor indexed by a diverse set of features, and the final prediction result is an aggregation of many predictions taking into account each prediction's confidence.

Sembrant et al. present a Reuse Aware Placement (RAP) policy [21] to optimize data movement across the entire cache hierarchy. RAP dynamically identifies data sets and measures their reuse at each level in the hierarchy. Each cache line is associated with a data set and consults that data set's policy upon eviction or installation. RAP selects a group of cachelines (called learning blocks) to help adapt changes in application and instruction behavior by ignoring bypass decisions upon installation. Compared with FILM, RAP experiences performance degradation as an incorrect bypass decision may have caused additional cache misses before it is corrected. Another factor leading to RAP's low performance is the absence of making special training effort on the prefetch requests. Moreover, RAP involves huge hardware overhead as it requires every cache block in the cache hierarchy to maintain a 12-bit large instruction pointer field.

VII. Conclusion

Due to the inherent difference of data block insertion and movement between an exclusive hierarchy and an inclusive/non-inclusive hierarchy, prior work, which is PC-correlated and is designed with non-inclusive caches in mind, cannot be easily applied to exclusive caches. Moreover, a holistic approach to manage data placement is essential for high cache performance and efficient resource utilization. Therefore, the authors propose FILM, a locality filtering mechanism to adaptively guide data placement into appropriate cache layers based on data reuse patterns. With a PC-based prediction scheme, FILM utilizes bloom filters to record the memory instruction PC of data blocks, incurring minimal cache overhead for meta-data transmission and storage. Additionally, FILM is able to quickly detect and correct any stale bypass decisions. FILM also does special training on prefetch requests, and makes prefetch aware learning of bypass/placement decisions.

Compared to a competitive baseline (TC-UC), FILM improves the average energy efficiency of multicore multi-programmed system by an of average 9% (maximum 20%), beating the second-highest average energy efficiency improvement from CHAR by 5%, and is constantly more energy efficient than other PC-correlated schemes. Moreover, FILM cuts down wasteful cache block insertions and data movement, and generates on average 12% less LLC traffic and 4% less DRAM traffic than other PC-correlated schemes.

VIII. Acknowledgements

This research has been supported in part by NSF grants 1725743, 1745813, and 1763848. A part of this research was conducted during an internship at Arm. We also wish to acknowledge the computing time we received on the Texas Advanced Computing Center (TACC) system. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the National Science Foundation or other sponsors. We would also like to thank the anonymous reviewers for their helpful suggestions to improve the paper.

REFERENCES

[1] A. Chandrasekher, "Meet qualcomm centriq 2400, the world's first 10-nanometer server processor." [Online]. Available: www.qualcomm.com/news/onq/2016/12/07/meet-qualcomm-centriq-2400-worlds-first-10-nanometer-server-processor

[2] J. Hofmann, G. Hager, G. Wellein, and D. Fey, "An analysis of core- and chip-level architectural features in four generations of intel server processors," 02 2017.

[3] A. Jaleel, J. Nuzman, A. Moga, S. C. Steely, and J. Emer, "High performing cache hierarchies for server workloads: Relaxing inclusion to capture the latency benefits of exclusive caches," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015.

[4] P. Kundu, M. Annavaram, T. Diep, and J. Shen, "A case for shared instruction cache on chip multiprocessors running oltp," *SIGARCH Comput. Archit. News*, vol. 32, Sep. 2003. [Online]. Available: http://doi.acm.org/10.1145/1024295.1024297

[5] M. Chaudhuri, J. Gaur, N. Bashyam, S. Subramoney, and J. Nuzman, "Introducing hierarchy-awareness in replacement and bypass algorithms for last-level caches," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2012.

[6] Wikipedia-contributors, "List of intel xeon microprocessors," 2018. [Online]. Available: https://en.wikipedia.org/wiki/List-of-Intel-Xeon-microprocessors-Skylake-based-Xeons

[7] Wikipedia-contributors, "Epyc," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Epyc

[8] J. Gaur, M. Chaudhuri, and S. Subramoney, "Bypass and insertion algorithms for exclusive last-level caches," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, June 2011.

[9] D. A. Jiménez and E. Teran, "Multiperspective reuse prediction," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3123939.3123942

[10] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016.

[11] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "Gpus and the future of parallel computing," *IEEE Micro*, vol. 31, Sep. 2011.

[12] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on*. IEEE, 1990.

[13] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *SIGARCH Comput. Archit. News*, vol. 38, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1816038.1815971

[14] C. J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely, and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2011.

[15] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, Jul. 1970. [Online]. Available: http://doi.acm.org/10.1145/362686.362692

[16] CRC2-2017, "The 2nd cache replacement championship," 2017. [Online]. Available: https://github.com/ChampSim/ChampSim

[17] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015.

[18] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009.

[19] J. Sim, J. Lee, M. K. Qureshi, and H. Kim, "Flexclusion: Balancing cache capacity and on-chip bandwidth via flexible exclusion," *SIGARCH Comput. Archit. News*, vol. 40, Jun. 2012. [Online]. Available: http://doi.acm.org/10.1145/2366231.2337196

[20] A. J. M. Q. V. Young, C. Chou, "Ship++: Enhancing signature-based hit predictor for improved cache performance," 2017. [Online]. Available: https://crc2.ece.tamu.edu/?$page_id = 53$

[21] A. Sembrant, E. Hagersten, and D. Black-Schaffer, "Data placement across the cache hierarchy: Minimizing data movement with reuse-aware placement," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016.

[22] J. D. Collins and D. M. Tullsen, "Hardware identification of cache conflict misses," in *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society, 1999.

[23] M. A. Z. Alves, K. Khubaib, E. Ebrahimi, V. Narasiman, C. Villavieja, P. O. A. Navaux, and Y. N. Patt, "Energy savings via dead sub-block prediction," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*. IEEE, 2012.

[24] Y. Etsion and D. G. Feitelson, "Exploiting core working sets to filter the l1 cache with random sampling," *IEEE Transactions on Computers*, vol. 61, 2012.

[25] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun, "A modified approach to data cache management," in *Proceedings of the 28th annual international symposium on Microarchitecture*. IEEE Computer Society Press, 1995.

[26] J. Jalminger and P. Stenstrom, "A novel approach to cache block reuse predictions," in *Parallel Processing, 2003. Proceedings. 2003 International Conference on*. IEEE, 2003.

[27] J. Ahn, S. Yoo, and K. Choi, "Dasca: Dead write prediction assisted stt-ram cache architecture," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*. IEEE, 2014.

[28] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*. IEEE, 2012.

[29] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," *IEEE Transactions on Computers*, vol. 57, 2008.

[30] L. Li, D. Tong, Z. Xie, J. Lu, and X. Cheng, "Optimal bypass monitor for high performance last-level caches," in *Parallel Architectures and Compilation Techniques (PACT), 2012 21st International Conference on*. IEEE, 2012.

[31] L. Xiang, T. Chen, Q. Shi, and W. Hu, "Less reused filter: improving l2 cache performance via filtering less reused lines," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009.

[32] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*. IEEE, 2008.

[33] E. Teran, Y. Tian, Z. Wang, and D. A. Jiménez, "Minimal disturbance placement and promotion," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016.

[34] D. A. Jiménez, "Insertion and promotion for tree-based pseudolru last-level caches," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013.

[35] D. A. Jiménez, "Dead block replacement and bypass with a sampling predictor," in *JWAC 2010-1st JILP Worshop on Computer Architecture Competitions: cache replacement Championship*, 2010.

[36] P. Faldu and B. Grot, "Leeway: Addressing variability in dead-block prediction for last-level caches," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept 2017.