PACMan: Prefetch-Aware Cache Management for High Performance Caching

Carole-Jean Wu*[§] Aamer Jaleel† Margaret Martonosi* Simon C. Steely Jr.† Joel Emer†‡

Princeton University*
Princeton, NJ
{carolewu,mrm}@princeton.edu

Intel Corporation, VSSAD†
Hudson, MA
{aamer.jaleel,simon.c.steely.jr,
joel.emer}@intel.com

Massachusetts Institute of Technology‡
Cambridge, MA

ABSTRACT

Hardware prefetching and last-level cache (LLC) management are two independent mechanisms to mitigate the growing latency to memory. However, the interaction between LLC management and hardware prefetching has received very little attention. This paper characterizes the performance of state-of-the-art LLC management policies in the presence and absence of hardware prefetching. Although prefetching improves performance by fetching useful data in advance, it can interact with LLC management policies to introduce application performance variability. This variability stems from the fact that current replacement policies treat prefetch and demand requests identically.

In order to provide better and more predictable performance, we propose Prefetch-Aware Cache Management (PACMan). PACMan dynamically estimates and mitigates the degree of prefetch-induced cache interference by modifying the cache insertion and hit promotion policies to treat demand and prefetch requests differently. Across a variety of emerging workloads, we show that PACMan eliminates the performance variability in state-of-the-art replacement policies under the influence of prefetching. In fact, PACMan improves performance consistently across multimedia, games, server, and SPEC CPU2006 workloads by an average of 21.9% over the baseline LRU policy. For multiprogrammed workloads, on a 4-core CMP, PACMan improves performance by 21.5% on average.

Categories and Subject Descriptors

B.8.3 [**Hardware**]: Memory Structures

General Terms

Design, Performance

Keywords

Prefetch-Aware Replacement, Reuse Distance Prediction, Shared Cache, Set Dueling

[§]A large part of this work was performed while Carole-Jean Wu was an intern at Intel/VSSAD.

MICRO '11 December 3-7, 2011, Porto Alegre, Brazil

1. Introduction

Technology trends show that main memory speeds significantly lag behind processor speeds. In response, both industry and academia have invested significant efforts in improving a processor's memory subsystem. One approach relies on efficiently managing the on-chip last-level cache (LLC) through intelligent cache replacement. Such proposals reduce LLC interference by dynamically modifying the cache insertion policy to prioritize the most reused and important data. Another popular approach is to *prefetch* data into the cache hierarchy *before* the actual reference. While prefetching can hide memory latency and improve performance significantly, it can severely degrade performance in the event of untimely and/or inaccurate prefetch requests. To avoid prefetcher pollution, recent studies have proposed alternative cache insertion policies for prefetch requests [18, 31]. Unfortunately, these proposals incur significant hardware overhead. In contrast, this paper proposes simple and low-overhead dynamic prefetch-aware LLC management to provide consistent performance improvements in the presence of hardware prefetching.

State-of-the-art cache replacement policies improve performance over the baseline least-recently-used (LRU) replacement policy by preserving the useful working set in the cache [2, 5, 12, 16, 17, 23, 26, 27, 30, 32, 35]. However, the majority of recent replacement policy proposals are evaluated in the absence of hardware prefetching. In the presence of prefetching, we find that intelligent replacement policies may provide minimal performance improvements or unexpectedly degrade performance. This stems from the fact that the majority of replacement policy proposals make the *same* replacement policy decisions for prefetch and demand requests.

This paper proposes *Prefetch-Aware Cache Management (PACMan)*. The goals of PACMan are (1) avoid cache pollution due to harmful prefetch requests and (2) retain cache lines that cannot be easily prefetched, and hence are more valuable in the cache. PACMan accomplishes both goals by recognizing that prefetch and demand requests can be treated differently on cache hits and cache insertions. In doing so, we show that PACMan reduces prefetcher induced cache pollution and combines the performance benefits from hardware prefetching and intelligent cache management.

We apply PACMan to the recent Dynamic Re-Reference Interval Prediction (DRRIP) [12] replacement policy. We show that the additional information on the type of request (demand or prefetch) can improve the re-reference predictions made by DRRIP. In the presence of hardware prefetching, PACMan significantly improves application performance by an average of 21.9% while DRRIP alone improves performance by only 5.8%. For server workloads, where prefetcher pollution is particularly problematic, PACMan improves performance by an average of 27.5% over LRU.

Furthermore, our evaluations with 4-core multi-programmed work-

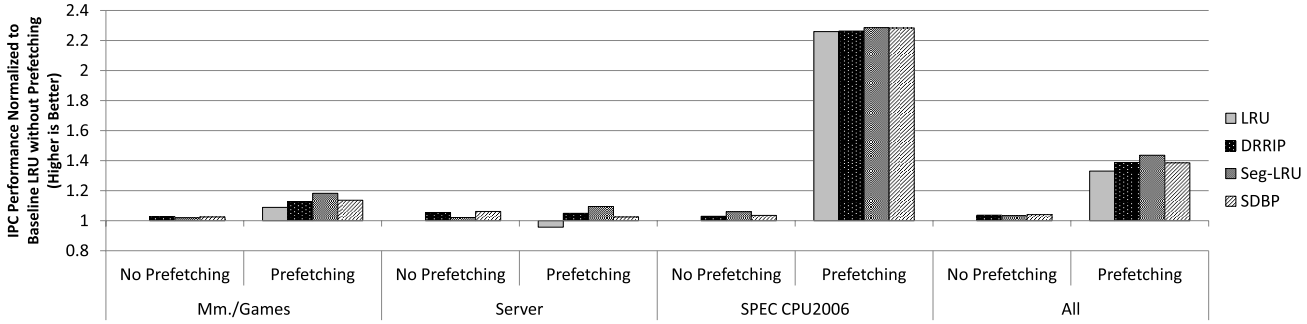

Figure 1: Hardware prefetching significantly improves application performance. See Section 5 for methodology details.

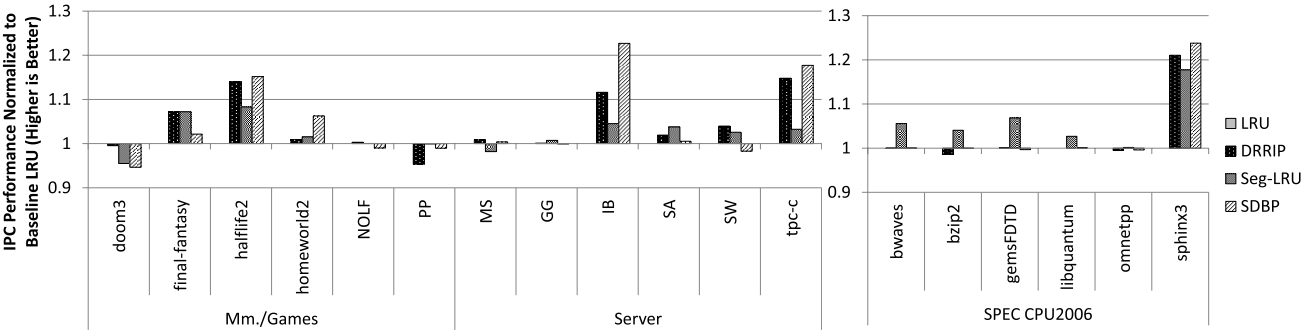

Figure 2: Performance of individual applications under cache management policies without prefetching: Cache management schemes, DRRIP, Seg-LRU, and SDBP, effectively improve performance in the absence of prefetching.

loads show that PACMan effectively eliminates inter-core, as well as intra-core, prefetch-induced interference in shared LLCs. In the presence of hardware prefetching, PACMan improves multi-programmed workload performance by an average of 21.5% while DRRIP alone improves performance by 7.8% over LRU.

Overall, the contributions of the paper are as follows:

- This paper provides a comprehensive performance evaluation of state-of-the-art cache management policies in the presence and absence of hardware prefetching. While an intelligent cache management technique can improve application memory performance, prefetching can increase performance even more by fetching useful data in advance to the cache. However, we show that prefetching *alone*, without the help of a prefetch-aware cache management technique, can interact poorly with LLC management, often causing performance degradation and variability.
- In addition to improving performance significantly, PACMan effectively eliminates performance variability introduced by prefetching. PACMan's ability to provide quality of service is particularly important for server applications, for which performance guarantee is a high priority.
- Finally, PACMan insights presented in this paper are independent from the underlying replacement policy. PACMan can be built on top of any ordered replacement policy. PACMan's simple and elegant prefetch-aware decision offers significant performance improvements without incurring additional overhead.

2. Motivation

To understand the effects of hardware prefetching thoroughly, we evaluate replacement policies in the presence and absence of hardware prefetching. For the purpose of this paper, we model a stream prefetcher that closely models the mid-level cache (MLC) stream prefetcher of an Intel Core i7 [8]. In addition, we implement three recent replacement policy proposals: DRRIP [12], Segmented-LRU (Seg-LRU) [5], and Sampling Deadblock Prediction (SDBP) [16] and evaluate their performance in the presence of hardware prefetching.

2.1 Performance Variability under Various Cache Replacement with Prefetching

Figure 1 illustrates a summary of application performance (for each workload category) in the presence of hardware prefetching. Overall, the figure shows that prefetching significantly improves application performance by roughly 35%. However, when we look at the performance of individual applications from each workload category more closely, we find that the performance of prefetching varies significantly across the different cache management policies. Figures 2 and 3 highlight this performance variability in the absence and presence of prefetching respectively. In both figures, the x-axis represents the different workloads under study while the y-axis represents the performance relative to the baseline LRU replacement policy without prefetching.

Figure 2 shows that, in the absence of prefetching, intelligent cache management schemes can improve application performance for a variety of applications. However, in the presence of prefetching, we observe very different behavior. For example, `final-fantasy` receives 2-8% performance improvement in the absence of prefetching. However, as Figure 3 shows, in the presence of prefetching, its performance varies from -12% to 8% across the four different underlying cache replacement policies.

In addition to performance variability across cache replacement policies, prefetching can severely degrade performance. This is particularly true for server applications where performance degrades by more than 20% (e.g. `tpc-c`). For some workloads (e.g. `IB` and

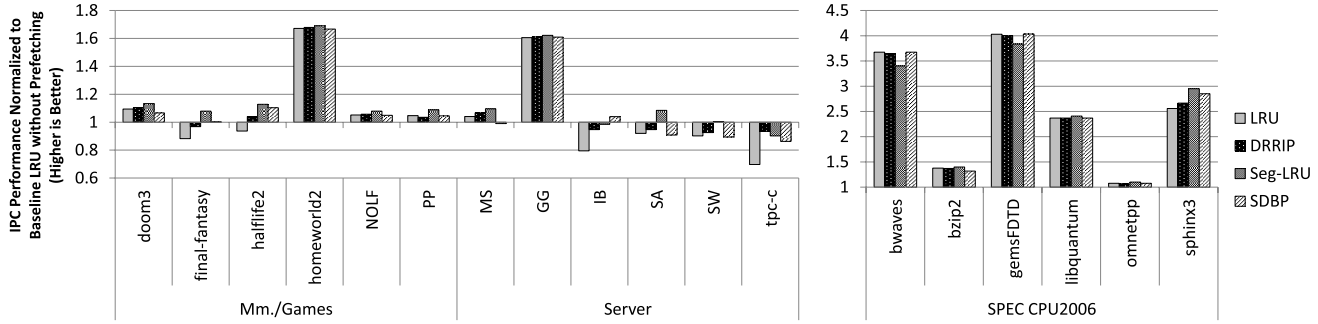

Figure 3: Performance of individual applications under cache management policies in the presence of prefetching: Prefetcher cache interference not only can cause performance degradation but also introduces performance variability. Note: Y-axes use different scales.

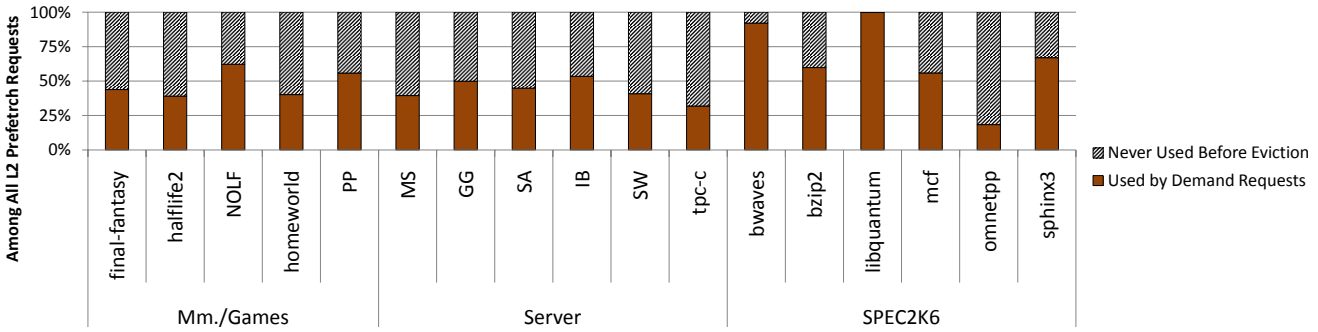

Figure 4: Reuse of prefetched cache lines at the L2 and last-level L3 caches. This is in part due to prefetcher LLC pollution on inaccurate prefetches, but also due to the temporal filtering of effective prefetches by references in smaller caches.

`tpc-c`), intelligent cache replacement policies can address some of the performance degradation caused by prefetcher pollution.

Most importantly, Figure 3 illustrates that the performance improvements from intelligent replacement policies drop significantly in the presence of prefetching. For example, in the absence of prefetching, `gemsFDTD` and `sphinx3` receive roughly 7% and 24% performance improvements respectively from an intelligent replacement policy. However, in the presence of prefetching, intelligent replacement yields negligible performance benefits for `gemsFDTD` while the benefits for `sphinx3` reduce to roughly 15% over the baseline LRU. This raises the question as to whether there exists a prefetch-aware cache management policy that can further improve performance of such applications in the presence of prefetching.

2.2 Low LLC Reuse of Prefetch Requests

To further motivate the need for a prefetch-aware cache management policy, Figure 4 illustrates the reuse characteristics of lines prefetched into the LLC. The x-axis represents the different workloads while the y-axis represents the fraction of cache lines inserted by a prefetcher that are re-referenced by a demand request. On average, 47% of prefetched cache lines are never reused by a demand request.

Low reuse of prefetched cache lines at the LLC can be due to prefetcher pollution. Such behavior is indicative of workloads whose performance degrades in the presence of prefetching (e.g. `tpc-c`). Alternatively, low reuse also occurs because of timely prefetched data in the smaller, upper-levels of the cache hierarchy (L2 cache in our case). Subsequent demand requests are directly serviced by the prefetched cache lines inserted into the L2 cache, and never reach the LLC. As a result, the filtering of temporal locality by the smaller caches causes low reuse of the prefetched lines in the LLC (e.g. `homeworld` and SPEC CPU2006 workloads).

In conclusion, the appearance of zero-reuse prefetched cache lines at the LLC and the reduced gains from intelligent replacement policies in the presence of hardware prefetching motivate alternative policies for handling prefetch requests at the LLC. The next section proposes such a policy.

3. Prefetch-Aware Cache Management

Despite the ubiquitous industry use of hardware prefetching, existing cache management policies do not distinguish between prefetch and demand requests. This leaves a significant opportunity to improve cache management by making *Prefetch-Aware Cache Management (PACMan)* decisions.

From a cache management perspective, prefetch requests have different properties than a demand request. In general, cache lines inserted into the LLC by demand requests are more likely to be performance-critical than prefetch requests. Thus, a replacement policy may benefit by giving more preference to items fetched by demand requests versus those from prefetch requests. This work explores different replacement policies for prefetch requests.

Cache replacement policies essentially predict the re-reference behavior of cache lines [12]. The natural opportunity to make re-reference predictions is on cache insertions and cache hits. For example, LRU predicts that a missing cache line will be re-referenced soon and always inserts the missing line at the "MRU position" of the LRU chain. Similarly, LRU predicts that a line receiving a cache hit will also be re-referenced soon and moves the line to the MRU position.

Traditionally, the majority of replacement policies assign the *same* re-reference predictions for demand and prefetch requests. To tackle the problem of prefetch-induced cache interference, we propose to make re-reference predictions at the granularity of a request type

	Baseline DRRIP	PACMan-M on DRRIP		PACMan-H on DRRIP		PACMan-HM on DRRIP	
SRRIP	**All**	**Demand**	**Prefetch**	**Demand**	**Prefetch**	**Demand**	**Prefetch**
Insertion	2	2	3	2	2	2	3
Re-Reference	0	0	0	0	No Update	0	No Update
BRRIP	**All**	**Demand**	**Prefetch**	**Demand**	**Prefetch**	**Demand**	**Prefetch**
Insertion	Mostly 3	Mostly 3	Mostly 3	Mostly 3	Mostly 3	Mostly 3	Mostly 3
Re-Reference	0	0	0	0	No Update	0	No Update

Figure 5: PACMan RRPV Assignment under DRRIP. We use the BRRIP configuration in [12] for all PACMan schemes under DRRIP: 95% of cache lines are inserted with an RRPV of 3, and 5% are inserted with an RRPV of 2.

(i.e. demand or prefetch). Specifically, we investigate replacement policies that apply the baseline re-reference predictions for demand requests, but apply modified re-referenced predictions for prefetch requests. We propose four types of *prefetch-aware* cache management policies:

- PACMan on Misses (**PACMan-M**): PACMan-M modifies the re-reference prediction *only when a prefetch request misses in the cache*. In particular, PACMan-M predicts that all prefetch requests will be re-referenced in the distant future. For example, like previous LRU-based studies, PACMan-M can be implemented by inserting *all* prefetch requests at the LRU position [18, 31]. In doing so, PACMan-M reduces the average in-cache lifetime of a prefetched cache line, in effect prioritizing demand requests over prefetch requests. In the event the line is subsequently re-referenced, the replacement state is updated on the cache hit. However, if there is no subsequent request, PACMan-M is useful for avoiding prefetcher pollution.
- PACMan on Hits (**PACMan-H**): PACMan-H modifies the re-reference prediction *only when a prefetch request hits in the cache*. Unlike a conventional replacement policy that always updates re-reference predictions on cache hits, PACMan-H chooses not to update the re-reference prediction on prefetch cache hits. In doing so, PACMan-H treats "prefetchable" requests with lower priority. In doing so, PACMan-H enables useful, but hard to prefetch, demand requests to be retained in the cache.
- PACMan on Hits and Misses (**PACMan-HM**): PACMan-HM modifies the re-reference predictions *for prefetch request cache hits and misses*. PACMan-HM gives shorter lifetime to *all* prefetch requests by using the PACMan-M policy on prefetch misses and the PACMan-H policy on prefetch hits. PACMan-HM is useful for avoiding prefetcher pollution and retaining hard to prefetch lines.
- Dynamic PACMan (**PACMan-DYN**): Statically predicting the re-reference pattern of prefetch requests can significantly degrade performance of workloads that benefit from prefetching. To ensure robust prefetch-aware replacement, we propose to dynamically detect the re-reference behavior of prefetch requests. PACMan-DYN uses Set Dueling [27] to dynamically predict the re-reference behavior of prefetch requests.

4. PACMan Implementation

PACMan can be applied to any existing state-of-the-art cache management policy. We implement PACMan upon the recently proposed Dynamic Re-Reference Interval Prediction (DRRIP) replacement policy [12] because of its simple design and low implementation complexity.

4.1 DRRIP Background

Instead of using a 4-bit LRU counter per cache line for a 16-way set-associative cache, DRRIP uses an M-bit counter per cache line to store the lines re-reference prediction value (RRPV). The RRPV in essence controls the lifetime of a cache line. In general, large RRPVs cause insertion closer to the "LRU position" and, therefore, have a short lifetime in the cache. In contrast, small RRPVs comparatively have a longer lifetime in the cache.

We base PACMan on 2-bit DRRIP, where each cache line can have RRPV between 0 and 3. DRRIP uses Set Dueling [28] to choose between its two component policies: SRRIP and BRRIP. SRRIP always inserts cache lines with an RRPV of 2 while BRRIP inserts most of cache lines with an RRPV of 3. On hits, DRRIP updates the RRPV of a cache line to 0. On replacements, DRRIP chooses a cache line with RRPV of 3 for replacement. If no block with RRPV of 3 is found, all RRPVs in the set are incremented and the search for a replacement block is repeated.

4.2 Applying PACMan to DRRIP

As summarized in Figure 5, PACMan's RRPV assignment is based on the DRRIP policy as follows:

PACMan-M: PACMan-M modifies DRRIP's insertion policy on cache misses. If a miss is due to a demand request, PACMan-M follows the baseline DRRIP insertion policy and inserts these cache lines with an RRPV of 2 (or 3). However, if the cache miss is due to a prefetch request, PACMan-M always inserts these lines with an RRPV of 3.

PACMan-H: PACMan-H modifies DRRIP's policy on cache hits. If the hit is due to a demand request, PACMan-H follows the baseline DRRIP policy and updates the line's RRPV to 0. However, if the cache hit is due to a prefetch request, PACMan-H does not modify the line's RRPV state. In doing so, PACMan-H explicitly prioritizes retaining more "valuable" (harder to prefetch) blocks instead of "prefetchable" blocks in the cache[1].

PACMan-HM: PACMan uses the PACMan-M policy at cache misses and follows the PACMan-H policy at cache hits.

4.3 PACMan-DYN

To differentiate whether a cache line is brought into the LLC by a demand versus prefetch request, prior work [1] proposed using an additional prefetch bit per cache block. To avoid this overhead, **PACMan-DYN** instead uses Set Dueling Monitors (SDMs) to determine which applications benefit from prefetching. An SDM estimates the number of cache misses for any given policy by permanently dedicating a few cache sets to follow that policy. The rest of

[1]DRRIP [12] assigns re-reference predictions based on the expected re-reference pattern. PACMan-H, however, assigns re-reference predictions based on the expected *value* of the cache line. For example, it is more "valuable" to retain cache lines that are harder to prefetch.

```
if(cacheMiss && isSDMSet_SRRIP+PACManH[setIndex])
     cnt_SRRIP+PACManH      += 2;
     cnt_SRRIP+PACManHM     -= 1;
     cnt_BRRIP+PACManHM     -= 1;

if(cacheMiss && isSDMSet_SRRIP+PACManHM[setIndex])
     cnt_SRRIP+PACManH      -= 1;
     cnt_SRRIP+PACManHM     += 2;
     cnt_BRRIP+PACManHM     -= 1;

if(cacheMiss && isSDMSet_BRRIP+PACManH[setIndex])
     cnt_SRRIP+PACManH      -= 1;
     cnt_SRRIP+PACManHM     -= 1;
     cnt_BRRIP+PACManHM     += 2;

if(cacheMiss && isFollowerSet[setIndex])
     // Use the policy with the minimum cnt_policy
     int usePolicy = findMinimum();
     Assign RRPV based on usePolicy;
```

Figure 6: PACMan-DYN Algorithm.

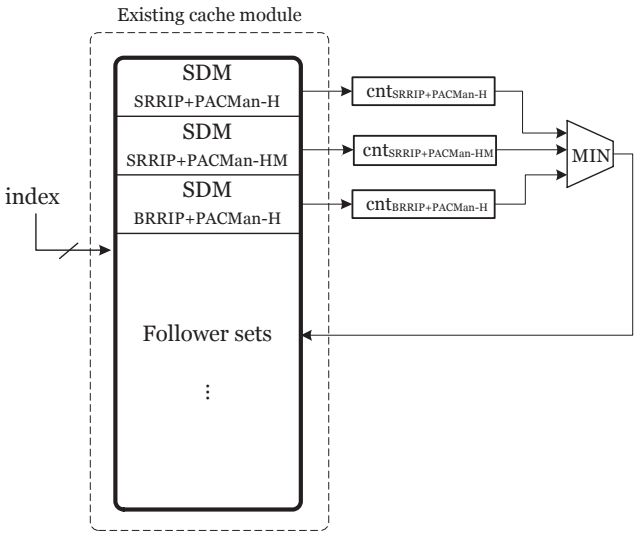

Figure 7: PACMan-DYN Implementation.

the cache sets then follow the policy whose SDM gives the fewest cache misses.

A brute force approach to use Set Dueling for determining the best PACMan prefetching and cache replacement combination requires 8 SDMs. As you will see in the result section, PACMan-H always performs better and provide more consistent gains than the underlying DRRIP replacement policy. As a result, we switch our baseline to PACMan-H and use Set Dueling to determine dynamically whether to use the PACMan-M component.

PACMan-DYN implements 3 SDMs: $\text{SDM}_{SRRIP+PACMan_H}$, $\text{SDM}_{SRRIP+PACMan_{HM}}$, and $\text{SDM}_{BRRIP+PACMan_H}$ over the 2-bit implementation of DRRIP. $\text{SDM}_{SRRIP+PACMan_H}$ follows the cache insertion and update policy based on SRRIP and PACMan-H, $\text{SDM}_{SRRIP+PACMan_{HM}}$ follows the cache insertion and update policy based on SRRIP and PACMan-HM, and $\text{SDM}_{BRRIP+PACMan_H}$ follows the cache insertion and update policy based on BRRIP and PACMan-H. PACMan-DYN does not implement the additional $\text{SDM}_{BRRIP+PACMan_{HM}}$ policy because, for cache misses, most references, regardless of prefetch or demand, are inserted with an RRPV of 3. In other words, the $\text{SDM}_{BRRIP+PACMan_{HM}}$ and $\text{SDM}_{BRRIP+PACMan_H}$ policies are essentially identical.

The PACMan-DYN algorithm is described in Figure 6 and its implementation is presented in Figure 7. Each SDM has an associated counter. When a demand cache miss[2] occurs at a set within a particular SDM, the associated counter is incremented by 2 and the counters associated with the other two SDMs are decremented by 1. If any counter is saturated, all counters remain the same. This policy selection process is based on a scheme proposed in [21] where the goal is to choose the best out of 3 different policies. For cache misses occurring at follower sets, PACMan-DYN finds the minimum of the three SDM counters and uses the corresponding policy for inserting and updating the references.

[2]Prefetch and writeback misses do not update the counters.

5. Experimental Methodology

5.1 Simulation Infrastructure

We evaluate PACMan using the the simulation framework released by the First JILP Workshop on Computer Architecture Competitions [13]. This Pin-based [20] CMP$im [10] simulation framework models a 4-way out-of-order processor with a 128-entry reorder buffer and a three-level cache hierarchy. The three-level cache hierarchy is based on an Intel Core i7 system [8]. The L1 and L2 caches use LRU replacement and our replacement policy studies are limited to the LLC. We model a per-core streamer prefetcher with 16 stream detectors that trains on L2 cache misses. The prefetcher issues prefetch requests from the L2 cache and inserts prefetched cache lines into the L2 and LLC.

Table 1: Architectural parameters of the simulated system.

MSHR: 32 entries allowing up to 32 outstanding misses
L1 Inst. Caches: 32KB, 4-way, Private, 1 cycle
L1 Data Caches: 32KB, 8-way, Private, 1 cycle
Mid-Level L2 Caches: 256KB, 8-way, Private, 10 cycles
LLC: 1MB per-core, 16-way, Shared, Non-inclusive, 30 cycles
Main Memory: 32 outstanding requests, 200 cycles

We also model an interconnect with a fixed average latency. Traffic entering the interconnect is modeled using a fixed number of miss status handling registers (MSHRs). Modeling MSHR contention reflects the latency penalty of excess traffic in the system. All transactions, including all data demand and prefetch requests, contend for the MSHRs. Table 1 lists the architectural parameters for the memory hierarchy and Table 2 summarizes the configurations used for the comparison replacement policy proposals.

5.2 Workload Construction

We perform our evaluation for state-of-the-art cache replacement schemes and PACMan using both sequential and multiprogrammed workloads. For single-program studies, we use a collection of applications from multimedia and PC games (Mm.), enterprise server (Srvr.), and SPEC CPU2006 (Spec) [29] applications. While these applications are quite diverse, most of our performance analysis focuses on 18 applications (which span the three categories) that are

Table 2: Description and configuration for the three state-of-the-art cache management techniques under study: DRRIP, Seg-LRU, and SDBP. The configuration is selected to represent the best performance in each scheme.

Policies	Description and Parameters
DRRIP	A cache insertion/replacement scheme using re-reference interval prediction to assign lifetime to cache blocks [12]. 10-bit policy selector; 32 sets per set-dueling monitor; 2-bit RRIP counters.
Seg-LRU	A cache replacement and bypassing scheme based on a segmented-LRU cache [5]. Adaptive bypassing enabled for Seg-LRU and LRU; Aging enabled for Seg-LRU; Initial bypassing probability is 64; Second minimum bypassing probability is 1/4096.
SDBP	A cache replacement and bypassing scheme based on instruction-aware deadblock prediction [16]. 32-set 12-way sampler; Three 4096-entry skew table; True 16-way LRU policy.

cache intensive. We conclude with a performance summary that includes all 65 applications.

The SPEC CPU2006 reference traces were collected using PinPoints [24] for the reference input set, while other application traces were collected on a hardware tracing platform. The hardware-traced workloads include both operating system and user-level activity while the SPEC CPU2006 workloads only include user-level activity. These workloads were run for 250 million instructions.

To evaluate cache management techniques for the shared LLC, we also construct 161 heterogeneous mixes of multiprogrammed workloads. First, we use 35 mixes of Mm. applications, 35 mixes of Srvr. applications, and 35 mixes of Spec programs. Finally, we create another 56 combinations of 4-core workloads comprising randomly-chosen applications that are mixed from different categories. The heterogeneous mixes of different workload categories are used as a proxy for a virtualized system. We run each application for 250 million instructions and collect the statistics with the first 250 million instructions completed. If the end of the trace is reached, the model rewinds the trace and restarts automatically. This simulation methodology is similar to recent work on shared caches [3, 11, 12, 19, 35].

6. Performance Evaluation

6.1 Overview Results for PACMan

Figure 8 compares the sequential application performance results for PACMan-M, PACMan-H, PACMan-HM, and PACMan-DYN with the baseline LRU scheme (LRU) and DRRIP. In general, PACMan-M or PACMan-H *alone* improves performance significantly over both LRU and DRRIP under prefetching across all types of workloads. PACMan-HM, combining the performance benefits from PACMan-M and PACMan-H, shows further performance improvement, except for `bwaves` and `gemsFDTD`. This is because PACMan-HM actively de-prioritizes prefetch requests over demand requests at cache hits and misses. As a result, applications which do not suffer from prefetch-induced cache pollution originally, e.g. `gemsFDTD`, experience performance degradation under PACMan-HM. PACMan-DYN resolves this negative performance impact. This is because PACMan-DYN eliminates performance degradation for applications that actually benefit from aggressive prefetching by giving beneficial prefetch requests higher priority. Finally, while PACMan-HM is effective, PACMan-DYN improves performance even more for applications like `sphinx3` because of its dynamic adjustment. Most importantly, PACMan-DYN consistently improves performance over the baseline for *all* applications.

6.2 Results for Sequential Workloads

Figure 9 compares the overall performance results for sequential applications. While DRRIP under prefetching improves performance by an average of 5.8% over the baseline LRU, PACMan-DYN improves performance by 21.9%. PACMan's prefetch-aware management is particularly important for server applications. It further improves server workload performance by an average of 27.5% over LRU. Looking more closely at SPEC CPU2006 applications, although application performance is less sensitive to replacement policies in the presence of prefetching, we find that PACMan-DYN *still* further improves SPEC CPU2006 workload performance. While DRRIP gives 1% performance benefit compared to LRU in the presence of prefetching, PACMan-DYN improves performance for SPEC CPU2006 applications more significantly by an average of 14.3%. Overall, PACMan-DYN demonstrates an effective cache management by differentiating prefetch and demand requests at cache hits and cache misses.

6.3 Performance Consistency

We categorize the sequential applications into three distinct classes, as summarized in Table 3. Class I applications represent those experiencing prefetch-induced cache interference and, as a result, seeing prefetch-induced performance degradation. Class II applications represent those that receive performance gains under intelligent cache management schemes [33] and can also benefit from prefetching. Finally, Class III applications represent those that are

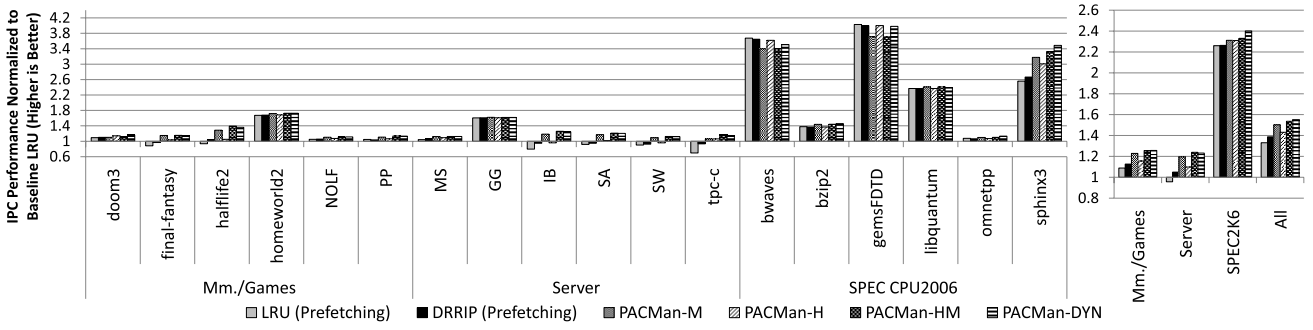

Figure 8: Performance comparison for PACMan-M, PACMan-H, PACMan-HM, and PACMan-DYN.

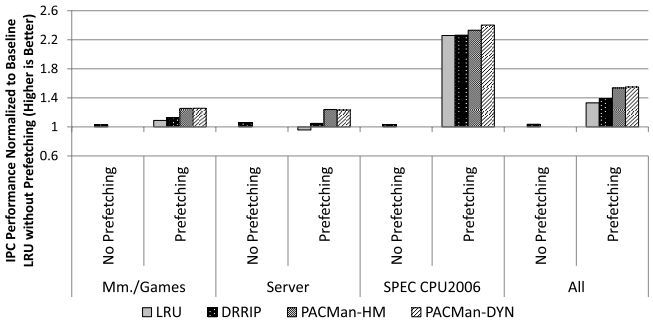

Figure 9: Performance comparison for sequential workloads.

not cache sensitive but benefit from prefetching. Typically, streaming and compute-bound applications fit into Class III.

Overall, PACMan-DYN consistently and effectively improves performance for applications in a diverse range of workloads. Figure 10 shows the consistent performance improvement under PACMan-HM (except for `bwaves` and `gemsFDTD`) and PACMan-DYN for all multimedia, games, server, and SPEC CPU2006 applications. This is because PACMan-DYN can exploit the distinct access characteristics of prefetch versus demand references and dynamically determine better cache insertion and cache update policies based on its observation. As a result, PACMan-DYN can manage the degree of prefetch-induced LLC interference more effectively.

PACMan-DYN's ability to eliminate performance variability caused by prefetch-induced cache interference is particularly important for applications in Class I, which experience the most significant cache pollution among the three categories. While DRRIP under prefetching improves the performance of Class I applications by an average of 1.13X over the baseline LRU scheme, PACMan-DYN improves the performance even more by an average of 1.44X and by as much as 1.67X for `tpc-c`.

For Class II applications, which not only are sensitive to the underlying cache management schemes but also receive benefit from prefetching, we show that both PACMan-HM and PACMan-DYN can further improve the performance significantly and consistently for all applications in this category. In particular, among all Class II applications, PACMan-DYN improves `sphinx3`'s performance the most: 1.36X over the baseline LRU scheme under the influence of prefetching.

Finally, for Class III applications, which neither suffer from prefetch-induced interference nor receive performance benefit with larger cache sizes, PACMan-DYN can sustain the performance improvement from prefetching by accurately identifying beneficial prefetching.

On average, in the presence of prefetching, PACMan-DYN improves application performance consistently by an average of 21.9% over the baseline LRU scheme. More importantly, PACMan-DYN is able to provide performance predictability in the presence of hardware prefetching. This performance guarantee is particularly important for server applications, for which performance quality of service is an important requirement.

6.4 PACMan for Multiprogrammed Workloads

Prefetch-induced cache interference becomes more complicated and severe for multiprogrammed workloads. Applications not only experience intra-application cache interference from their own prefetch requests, but also cross-application prefetch-prefetch and prefetch-demand interference. As a result, the degree of cache contention is also higher than that observed in sequential workloads.

To implement PACMan-DYN for multiprogrammed workloads, we investigate two approaches: **PACMan-DYN-Local** and **PACMan-DYN-Global**. PACMan-DYN-Local duplicates the group of SDMs directly from PACMan-DYN for sequential applications. Similar to the adaptive insertion policy [11], each application has its own SDMs for the three policies described in Section 4.3. The follower sets then use the best of the three policies based on the counters associated with each application. Based on each indi-

Table 3: Applications selected to represent 3 distinct categories under prefetching influence.

Class I: Harmful Prefetching
final-fantasy (Mm.), halflife2 (Mm.), SA (Srvr.), IB (Srvr.), SW (Srvr.), tpc-c (Srvr.)
Class II: Beneficial Prefetching, Cache Sensitive
doom3 (Mm.), NOLF (Mm.), homeworld (Mm.), PP (Mm.), MS (Srvr.); bzip2 (Spec), omnetpp (Spec), sphinx3 (Spec)
Class III: Beneficial Prefetching, Cache Insensitive
GG (Srvr.), bwaves (Spec), gemsFDTD (Spec), libquantum (Spec)

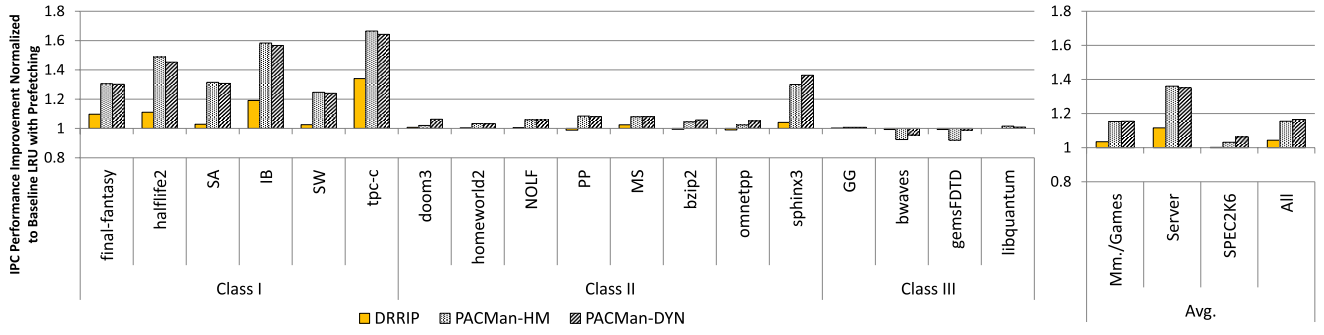

Figure 10: Performance comparison for applications from 3 different classes (See Table 3). Both PACMan and PACMan-DYN can effectively and consistently improve application performance when compared to other state-of-the-art LLC management schemes.

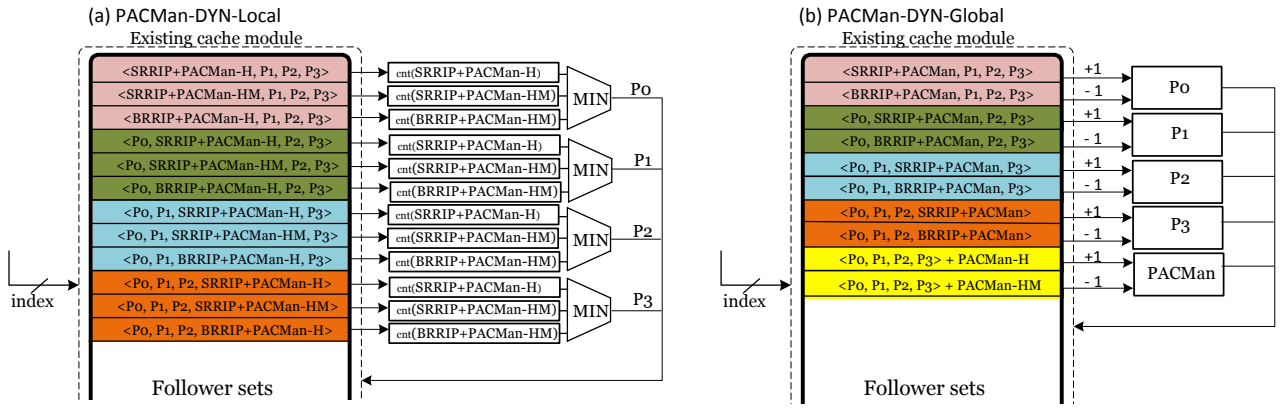

Figure 11: PACMan-DYN implementation for the 4-core workloads. (a) PACMan-DYN-Local: A group of 3 SDMs is used to determine the best per-application-basis PACMan policy. For example, the three top-most SDMs are dedicated to `App0`. References of `App0` follow the PACMan policy pre-assigned to each of its SDMs. References from the other three applications follow the policies determined by the per-core policy selection counters: P_1, P_2, or P_3. (b) PACMan-DYN-Global: A group of 2 SDMs is dedicated to each application. Similar to how the default DRRIP works, $application_i$ follows the policy determined by the per-application policy counter, P_i. In addition, two SDMs are used to determine which of the PACMan policies (PACMan-H or PACMan-HM) the follower sets use.

vidual application's prefetching preference, PACMan-DYN-Local follows the best local, per-application basis policy. Figure 11(a) depicts PACMan-DYN-Local's implementation.

To explore a more global prefetch decision for the shared LLC, we also propose PACMan-DYN-Global. PACMan-DYN-Global chooses a uniform global prefetch cache insertion and update decision for all per-core prefetchers. In PACMan-DYN-Global, each application in the multiprogrammed workload has two SDMs that follow the baseline DRRIP policies: SRRIP and BRRIP. To determine whether prefetching is beneficial globally to the LLC, PACMan-DYN-Global implements two additional SDMs: SDM_{PACMan_H} and $\text{SDM}_{PACMan_{HM}}$. SDM_{PACMan_H} follows the better of the two policies for each application together with the PACMan-H policy. Similarly, $\text{SDM}_{PACMan_{HM}}$ follows the better policy for each application along with the PACMan-HM policy for prefetch requests. Note, as Figure 11(b) shows, since each application in PACMan-DYN-Global uses only two SDMs, only one saturating counter (instead of three for PACMan-DYN-Local) is required for each application. Furthermore, PACMan-DYN-Global requires fewer total SDMs than in PACMan-DYN-Local.

6.5 Results for Multiprogrammed Workloads

Figure 12 shows that, in the presence of prefetching, DRRIP can improve performance by an average of 7.8% over the baseline LRU already. This is because although DRRIP does not directly distinguish between demand and prefetch requests, its intrinsic scan- and thrash-resistant properties can identify aggressive, but low-reuse, prefetch requests and reduce interfering prefetch requests in the LLC. However, PACMan-HM with explicit prefetch-aware cache management improves performance even more: an average of 22.9% for multiprogrammed workloads. For the multiprogrammed workloads studied in this paper, PACMan-DYN-Local and PACMan-DYN-Global did not perform better than PACMan-HM because of the SDM learning overhead; however, PACMan-DYN-Local and PACMan-DYN-Global are still more robust when prefetching preference varies over time in the workload. Nonetheless, for all 161 multiprogrammed workloads, both PACMan-DYN-Local and PACMan-DYN-Global still offer significant performance improvements over LRU by an average of 20.9% and 21.5% respectively. Since PACMan-DYN-Global requires less SDM learning overhead while offering slightly better performance gain over PACMan-DYN-Local, it is preferable to use PACMan-DYN-Global for the shared LLC. For the rest of the paper, for the shared LLC, we will refer PACMan-DYN-Global as PACMan-DYN.

If we focus on the performance improvement for the multiprogrammed mixes of SPEC CPU2006 applications, we do not see much performance difference across the different state-of-the-art cache management policies when considering prefetching. This is similar to what we observe for sequential SPEC CPU2006 applications. However, both PACMan-HM and PACMan-DYN demonstrate their effective management by using prefetch-aware cache insertion and cache hit policies. As a result, PACMan-HM and

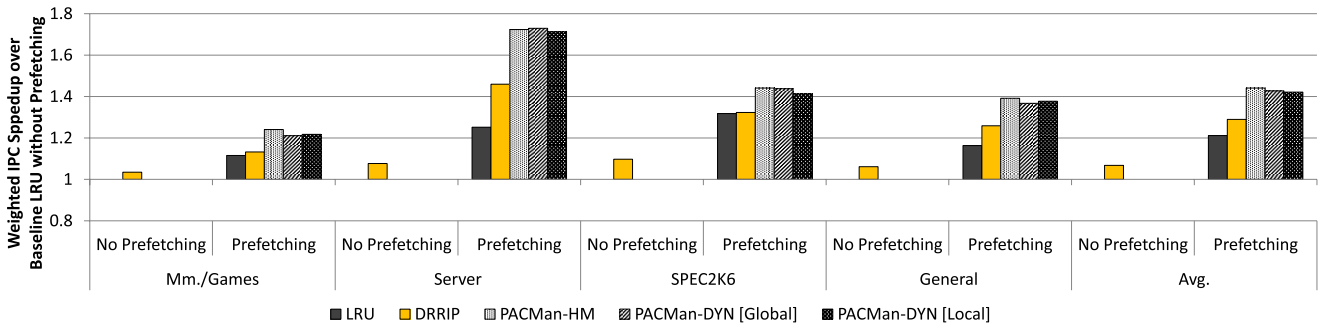

Figure 12: Performance comparison for multiprogrammed workloads.

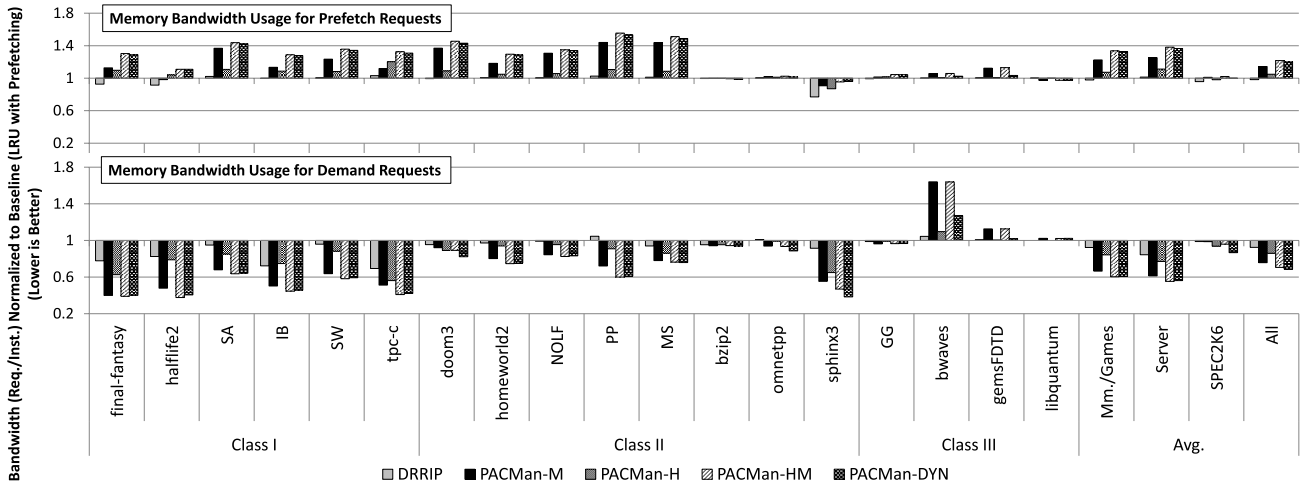

Figure 13: PACMan's impact on memory bandwidth.

PACMan-DYN improve the performance of multiprogrammed SPEC CPU2006 workloads by an average of 12.3% and 12.0% over the baseline LRU under prefetching. Finally, similar to sequential server workloads, PACMan-DYN's prefetch-aware cache management decisions improve multiprogrammed *server workloads* the most: an average of 47.8% (46.1% for PACMan-DYN-Local). This is a considerable improvement over the 20.8% offered by DRRIP.

6.6 Impact on Memory Bandwidth

Prefetching can adversely affect the bandwidth requirements to memory. This section evaluates the influence of PACMan's cache management decision on memory bandwidth. Figure 13 compares PACMan's impact on bandwidth between the LLC and the main memory. The upper portion of the figure plots the number of *prefetch requests* to memory per instruction, normalized to the baseline LRU scheme. The lower portion of the figure plots the number of *data demand requests* to memory per instruction normalized to the baseline LRU scheme under prefetching. On average, PACMan-HM increases the bandwidth usage from hardware prefetchers by 22% while PACMan-DYN incurs less memory bandwidth overhead (by an average of 20%). This increase in memory bandwidth due to prefetching is because PACMan-HM does not cache many prefetched cache lines in the LLC that are considered harmful. This results in more memory traffic for prefetched requests.

When comparing PACMan's impact on data demand memory bandwidth usage, we see a significant reduction for all PACMan schemes. This is because PACMan can significantly decrease the number of demand cache misses at the LLC, and, as a result, improve the data demand memory bandwidth between the LLC and the main memory. For the selected applications, while DRRIP under prefetching can reduce demand memory bandwidth by an average of 7.6%, PACMan-HM and PACMan-DYN improve demand memory bandwidth more significantly by an average of 29.5% and 31.3% respectively.

In particular, for applications that do not suffer much from prefetch-induced cache pollution i.e. `gemsFDTD`, PACMan-DYN's dynamic detection and adjustment for cache insertion and update policies help to minimize the memory bandwidth consumption. While PACMan-HM increases `gemsFDTD`'s demand memory bandwidth by 12.6%, PACMan-DYN can reduce this significant bandwidth overhead to merely 2%. Finally, we note that PACMan-DYN is

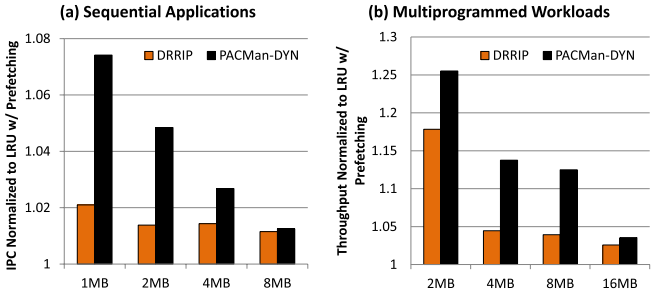

Figure 14: PACMan-DYN sensitivity to cache sizes for (a) sequential and (b) multiprogrammed workloads.

essentially a lightweight dynamic method for detecting prefetch-induced cache pollution. As such, we foresee future methods where PACMan-DYN is used to modulate prefetcher aggressiveness to further reduce memory bandwidth requirement caused by today's aggressive hardware prefetchers.

6.7 PACMan-DYN Sensitivity to Cache Sizes and Various Cache Hierarchies

To analyze the effectiveness of PACMan under various cache sizes, we perform a cache size sensitivity study for PACMan-DYN. Figure 14(a) shows that for sequential workloads, PACMan-DYN consistently outperforms DRRIP. The performance benefit levels off for an 8MB LLC. This is because an 8MB cache is adequate to accommodate both prefetch and demand references. For multiprogrammed workloads, the LLC experiences higher cache contention. Figure 14(b) illustrates that, for cache sizes ranging from 2MB to 16MB, PACMan consistently outperforms LRU and DRRIP for all shared cache sizes.

Although this work evaluates the various PACMan insertion and promotion policies for non-inclusive cache hierarchy, the core idea of the prefetch-aware cache management applies to strictly inclusive [9] and exclusive [15] cache hierarchies as well. For the inclusive cache hierarchy, we apply exactly the same PACMan-DYN insertion and hit promotion policies. For the exclusive cache hierarchy, we retain the same capacity in the three-level hierarchy by

doubling the L2 cache size and halving the LLC size. Since exclusion requires lines evicted from the L2 cache to be filled into the LLC, to implement PACMan-DYN, the L2 cache must remember which L2 cache lines were brought in by prefetch requests. This can be accomplished by adding a *prefetch* bit per line in the L2 cache to remember whether the L2 cache line was brought in by a demand or a prefetch request. To distinguish between demand and prefetched lines, this *prefetch* bit at the L2 cache is used by PACMan-DYN to determine the LLC insertion policy. Across all workloads, our evaluation of PACMan-DYN for the strictly inclusive and exclusive cache hierarchies see as much as 28% performance improvement (and 5% on average) over LRU: this doubles the performance gain over DRRIP.

6.8 Hardware Overhead

Table 4 compares hardware overhead and performance for several prior state-of-the-art schemes, along with PACMan-HM and PACMan-DYN. We show performance for the 18 sequential applications considered thus far, and also for the full 65-application suite from which they were drawn. PACMan-DYN uses the same amount of hardware as DRRIP but, on average, it offers 16.2% more performance improvement for the selected sequential applications and 13.8% more for multiprogrammed workloads. Seg-LRU offers the best performance of the prior schemes, but uses much more hardware implementation overhead. In contrast, PACMan-DYN improves performance even more than Seg-LRU—an additional 10.6%. Also importantly, PACMan-DYN offers a much simpler implementation than Seg-LRU, with little hardware beyond its underlying replacement policy. Finally, we note that while the performance improvements for the full 65-application suite (including non-cache-intensive applications) are lower than those for the 18 cache-intensive applications, PACMan-DYN's performance improvements are still considerable. They still greatly exceed that of the other possible approaches. Overall, PACMan-DYN offers a simple and practical cache design for managing LLC prefetching influence.

6.9 Results Summary

To summarize, we provide the first characterization for prefetch-induced LLC interference within an application, as well as across applications, under state-of-the-art cache capacity management techniques. We illustrate that, in the presence of hardware prefetching, performance benefits using prior proposals become more varied. Some applications experience significant performance degradation. To tackle this problem, we propose a simple prefetch-aware cache management scheme, PACMan-DYN. PACMan-DYN offers a more consistent performance improvement across applications for a diverse range of workloads: multimedia, games, server, and SPEC CPU2006. In the presence of prefetching, PACMan-DYN improves the performance of the 65 sequential and 161 multiprogrammed workloads by an average of 9.2% and 21.5% respectively over the baseline LRU scheme while DRRIP improves performance by 2.9% and 5.8%. This work shows the need of cache management schemes to recognize the unique access behavior for demand and prefetch requests in future design. Finally, while we build PACMan-DYN upon DRRIP, our insights to the interaction between demand and prefetch requests are not limited to a DRRIP-based cache; the same ideas hold regardless of underlying cache management policies.

7. Related Work

7.1 Dynamic Prefetching Control

Prefetching takes advantage of available memory bandwidth to reduce memory latencies. However, the benefits of aggressive prefetching heavily rely on its accuracy. Other prior work [1, 4, 6, 7, 14, 18, 22, 25, 31, 34, 36] improves prefetching accuracy and reduces cache pollution caused by prefetch requests. All of these work mainly focus on designs that reduce per-core interference between prefetch and demand requests for different cores in CMP systems.

Lin, Reinhardt, and Burger [18] proposed a mechanism to reduce DRAM latencies by modulating prefetcher issue rate, maximizing DRAM low hit rate, and always inserting prefetch requests with the lowest priority in the LLC. Their proposed mechanism is static and always gives prefetch references lower priority by inserting them in the LRU position. Our paper, on the other hand, introduces several policy variations including a dynamic prefetch-aware cache management solution.

Alameldeen and Wood [1] proposed to use an additional prefetch bit per cache block to estimate the degree of prefetch-induced cache pollution and adjust the aggressiveness of hardware prefetchers accordingly. Our cache pollution estimation scheme proposed here accurately approximates when prefetching harms performance, but at negligible hardware overhead.

Srinath et al. [31] built a feedback-directed prefetching mechanism which uses an additional prefetch bit per cache block to help identify prefetcher accuracy. Furthermore, additional hardware is required to help identify the degree of prefetch-induced cache pollution. Again, the light-weight cache pollution estimation mechanism proposed in PACMan can be used in [31] and the insights discovered in PACMan can help further improve the feedback directed prefetching scheme's decision on cache fill and hit policies for demand and prefetch cache blocks.

Wu and Martonosi [34] characterized real-system cache pollution caused by hardware prefetchers and proposed a dynamic prefetcher manager that modulates the aggressiveness of prefetchers at runtime. Their real-system prefetch manager implementation offers an orthogonal and coarser-grained approach to mitigate prefetch-induced cache pollution without hardware changes.

Table 4: Hardware overhead and performance comparison for prior state-of-the-art schemes, PACMan-HM, and PACMan-DYN.

Policies	%IPC vs. LRU (18 rep. apps)	%IPC vs. LRU (65 apps)	Hardware Overhead**	16-way 1MB LLC
LRU	33.0	29.3	$n * \log_2 n$	8KB
DRRIP	38.8	32.2	2n	4KB
Seg-LRU***	44.4	33.6	5n + 5.68KB	15.68KB
SDBP***	40.4	32.6	5n + 13.75KB	23.75KB
PACMan-HM	53.6	38.0	2n	4KB
PACMan-DYN	55.0	38.5	2n	4KB

**Assuming an n-way set-associative cache, HW overhead is measured in bits required per cache set.
***Hardware overhead for Seg-LRU and SDBP scales up as the number of cores sharing the LLC increases.

On the other hand, Ebrahimi et al. [4] proposed coordinated control of CMP prefetchers to reduce inter-core prefetcher-prefetcher LLC interference. However, this approach requires modification to prefetcher organization in hardware whereas PACMan leverages hardware implementation of the underlying cache replacement policy and with little additional overhead.

To our knowledge, there has not been any research that analyzes cache management schemes under prefetching in detail. We are the first to compare the performance of three state-of-the-art cache management techniques, DRRIP, SLRU, and SDBP, in the presence of prefetching. We further propose PACMan, a novel and practical prefetch-aware cache management scheme to address both intra- and inter-core prefetch-induced cache interference.

7.2 LLC Interference Characterization and Cache Capacity Management

The LLC is one of the most critical shared resources in CMPs. For performance isolation, quality of service, and better system throughput for the LLC, there has been research addressing cache capacity management problems [2, 5, 12, 16, 17, 23, 26, 30, 32, 35]. However, real systems have many contributing factors to LLC interference including per-core prefetchers. This interference becomes more complicated as multiple applications and multiple per-core prefetchers share the LLC simultaneously.

Prior work often focuses on characterizing inter-core cache interference only; they neglect other contributing factors to LLC interference in real systems. As characterized in the paper, hardware prefetching implemented in all of today's high performance systems influences memory sub-system performance significantly. To the best of our knowledge, we are the first to analyze prefetch-induced interference for private and shared LLCs, offer a comprehensive study on the effect of prefetching for the state-of-the-art cache management techniques, and propose a novel management solution addressing this problem.

8. Conclusion

The most important research impact of our work is to highlight the need for cache research studies to consider prefetching and other real-system effects when evaluating management schemes and other design proposals. As characterized in the paper, hardware prefetching implemented in today's high performance systems significantly influences memory sub-system performance. In the presence of prefetching, performance benefits using prior proposals become more varied. Some applications experience significant performance degradation. Other applications, which originally see significant performance improvement from an intelligent cache management scheme in the absence of prefetching, do not continue to see these gains when prefetching effects are accounted for.

To address the challenges of performance degradation and variability under prefetching, we propose a novel Prefetch-Aware Cache Management technique (PACMan) to mitigate prefetch-induced LLC interference. We build PACMan on top of a state-of-the-art replacement policy, DRRIP, by modifying cache insertion and cache hit promotion policies differently for prefetch and demand requests to (1) avoid cache pollution due to harmful prefetch requests, and (2) retain cache lines that cannot be easily prefetched in the cache. The proposed dynamic PACMan policy implements Set Dueling to determine when prefetching is beneficial to running applications. Thus, unlike prior work, PACMan can exploit benefits of hardware prefetching without additional hardware overhead.

PACMan can effectively and consistently reduce the degree of prefetch-induced cache pollution, a prominent issue particularly in server workloads. As a result, in the presence of prefetching, server workload performance is significantly improved by an average of 27.9% over the baseline LRU scheme. Moreover, PACMan can further improve the performance of SPEC CPU2006 and other applications which benefit hardware prefetching originally. On average, while the best of the state-of-the-art techniques improves the performance of the diverse range of applications by an average of 11.4% over LRU, PACMan can improve performance by an average of 21.9% over LRU and by 16.2% over DRRIP under prefetching.

Furthermore, we evaluate PACMan with multiprogrammed workloads, for which prefetch-induced interference is a more complicated problem. Workloads not only experience intra-core demand-prefetch interference, but aggressive per-core prefetchers worsens the problem by introducing inter-core prefetch-prefetch and prefetch-demand interference. We show PACMan can eliminate intra-core, as well as inter-core, prefetch-induced interference for shared LLCs. As a result, the performance of multiprogrammed workloads is improved by an average of 21.5% over LRU and by 13.8% over DRRIP under prefetching. Finally by more effectively caching the most useful demand- and prefetch-requested data, PACMan also reduces demand memory bandwidth considerably: an average of roughly 31.3%. Overall, the detailed analysis of state-of-the-art cache management policies in this paper can guide future work in memory hierarchy architecture and design regarding the importance and challenges of prefetching.

9. Acknowledgements

We thank the entire Intel VSSAD group for their support and feedback during this work. We also thank Yu-Yuan Chen, Daniel Lustig, and the anonymous reviewers for their useful insights related to this work. This material is based upon work supported by the National Science Foundation under Grant No. CNS-0509402 and CNS-0720561. The authors also acknowledge the support of the Gigascale Systems Research Center, one of six centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity. Carole-Jean Wu is supported in part by an Intel PhD Fellowship.

10. References

[1] A. Alameldeen and D. Wood. Interactions between compression and prefetching in chip multiprocessors. In *Proceedings of the 35th International Symposium on Computer Architecture*, 2007.

[2] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005.

[3] M. Chaudhuri. Pseudo-LIFO: The foundation of a new family of replacement policies for LLCs. In *Proceedings of the 42nd International Symposium on Microarchitecture*, 2009.

[4] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt. Coordinated control of multiple prefetchers in multi-core systems. In *Proceedings of the 42nd International Symposium on Microarchitecture*, 2009.

[5] H. Gao and C. Wilkerson. A dueling segmented LRU replacement algorithm with adaptive bypassing. In *Proceedings of the 1st JILP Workshop on Computer Architecture Competitions*, 2010.

[6] I. Hur and C. Lin. Memory prefetching using adaptive stream detection. In *Proceedings of the 39th International Symposium on Microarchitecture*, 2006.

[7] S. Iacobovici, L. Spracklen, S. Kadambi, Y. Chou, and S. G.

Abraham. Effective stream-based and execution-based data prefetching. In *Proceedings of the 18th International Conference on Supercomputing*, 2004.

[8] Intel Core i7 Processors http://www.intel.com/products/processor/corei7/.

[9] A. Jaleel, E. Borch, M. Bhandaru, S. C. Steely Jr., and J. Emer. Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (TLA) cache management policies. In *Proceedings of the 43rd International Symposium on Microarchitecture*, 2010.

[10] A. Jaleel, R. S. Cohn, C.-K. Luk, and B. Jacob. CMP$im: A Pin-based on-the-fly multi-core cache simulator. In *Proceedings of the 4th Workshop on Modeling, Benchmarking and Simulation*, 2008.

[11] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. C. Steely Jr., and J. Emer. Adaptive insertion policies for managing shared caches. In *Proceedings of the 17th International Conference on Parallel Architecture and Compilation Techniques*, 2008.

[12] A. Jaleel, K. B. Theobald, S. C. Steely Jr., and J. Emer. High performance cache replacement using re-reference interval prediction (RRIP). In *Proceedings of the 38th International Symposium on Computer Architecture*, 2010.

[13] JILP Workshop on computer architecture competitions http://jilp.org/jwac-1/.

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

[15] N. P. Jouppi and S. J. E. Wilton. Tradeoffs in two-level on-chip caching. In *Proceedings of the 21st International Symposium on Computer Architecture*, 1994.

[16] S. M. Khan, Y. Tian, and D. A. Jiménez. Dead block replacement and bypass with a sampling predictor. In *Proceedings of the 43rd International Symposium on Microarchitecture*, 2010.

[17] S. Kim, D. Chandra, and Y. Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004.

[18] W.-f. Lin, S. Reinhardt, and D. Burger. Reducing DRAM latencies with an integrated memory hierarchy design. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture*, 2001.

[19] G. Loh. Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy. In *Proceedings of the 42nd International Symposium on Microarchitecture*, 2009.

[20] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005.

[21] P. Michaud. The 3P and 4P cache replacement policies. In *Proceedings of the 1st JILP Workshop on Computer Architecture Competitions*, 2010.

[22] K. Nesbit, A. Dhodapkar, and J. Smith. AC/DC: An adaptive data cache prefetcher. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004.

[23] K. Nesbit, J. Laudon, and J. Smith. Virtual private caches. In *Proceedings of the 35th International Symposium on Computer Architecture*, 2007.

[24] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. In *Proceedings of the 37th International Symposium on Microarchitecture*, 2004.

[25] J.-K. Peir, S.-C. Lai, S.-L. Lu, J. Stark, and K. Lai. Bloom filtering cache misses for accurate data speculation and prefetching. In *Proceedings of the 16th International Conference on Supercomputing*, 2002.

[26] M. Qureshi and Y. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th International Symposium on Microarchitecture*, 2006.

[27] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely Jr., and J. Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 35th International Symposium on Computer Architecture*, 2007.

[28] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A case for MLP-aware cache replacement. In *Proceedings of the 34th International Symposium on Computer Architecture*, 2006.

[29] Standard Performance Evaluation Corporation CPU2006 Benchmark Suite. http://www.spec.org/cpu2006/.

[30] S. Srikantaiah, M. Kandemir, and M. J. Irwin. Adaptive set pinning: Managing shared caches in chip multiprocessors. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008.

[31] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt. Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, 2007.

[32] E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, 2002.

[33] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr., and J. Emer. SHiP: Signature-based hit predictor for high performance caching. In *Proceedings of the 44th International Symposium on Microarchitecture*, 2011.

[34] C.-J. Wu and M. Martonosi. Characterization and dynamic mitigation of intra-application cache interference. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, 2011.

[35] Y. Xie and G. Loh. PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches. In *Proceedings of the 37th International Symposium on Computer Architecture*, 2009.

[36] X. Zhuang and H.-H. Lee. Reducing cache pollution via dynamic data prefetch filtering. In *IEEE Trans. Comput.*, volume 56, January 2007.