# Perceptron Learning Driven Coherence Aware Reuse Prediction for Last-level Caches

Snehil Verma*
snehilv@iitk.ac.in

***Abstract*—Inclusive caches have been widely used in Chip Multiprocessors in favor of the simplicity of cache coherence protocol. However, it trades the coherence simplicity for effective cache capacity unlike the non-inclusive caches. Cache blocks that get evicted from the last-level cache (LLC) need to be invalidated in higher levels of cache in order to maintain the inclusion property. It is possible that highly referenced blocks in the higher levels of cache (L1 and L2) can get selected as victims in the LLC, which may incur more coherence traffic and limit system performance. Thus, it is important to predict highly reused blocks in the whole cache hierarchy and incorporate the prediction with cache management policies for better performance. We propose Coherence-Aware Reuse Prediction (CARP) to use cache coherence information of an application as a feature in perceptron learning based reuse prediction. We use the same for better cache management at the LLC. We observe that CARP provides marginal performance improvement for most of the parallel applications from PARSEC benchmark suite. Additionally, we show the effectiveness of CARP by running multiple parallel applications concurrently.**

## I. Introduction

Modern Chip Multiprocessor (CMP) designs mostly have inclusive cache hierarchy [14] since it simplifies the coherence protocol. To enforce the inclusion property, the data cached in higher level caches should be a subset of lower level caches. Since the inclusive last level caches (LLCs) are often unaware of the temporal locality of the higher level caches, blocks with high temporal locality in higher level may be consequently evicted from the cache hierarchy if the LLC decides to replace it. This may limit the performance for inclusive caches due to smaller effective capacity, compared to non-inclusive and exclusive caches. To enjoy the benefits of inclusive cache without sacrificing performance, it is essential to keep the important blocks in the cache hierarchy.

One metric to evaluate a block's importance is its reuse, that tells the frequency of the block re-references. A smart cache management policy should keep the blocks with high reuse and replace those with lesser or no reuse. The presence of no-reuse blocks in the cache is one of the major factors that affects cache utilization. A cache block is considered no-reuse if it is not re-referenced until its eviction. The impact of no-reuse blocks on the cache performance is two-folded. It occupies the space in the cache unnecessarily and thereby other useful blocks (for near future accesses) may get evicted. Hence, the replacement policies must be aware of the no-reuse blocks in the cache in order to select the victims during a cache line eviction. Reuse predictors are researched to detect useful blocks that tend to be accessed again before it is evicted [2], [5]. However, these techniques only evaluate multi-programmed workloads and neglects the sharing for multi-threaded workloads.

*Work done in collaboration with Jiayi Huang and Pritam Majumder

Multi-threaded shared memory applications executing on CMPs require communication between the threads mapped on different cores through the shared LLC. In addition, highly re-referenced blocks in higher cache levels tend to have low reuse in LLCs. In such cases, most replacement policies ignore the workload preference of shared data over private data. They are also unaware of the reuse of the blocks in the higher level caches and tend to replace those blocks. Significant research has been done to predict block reuse in caches for multi-programmed single-threaded workloads. In shared caches, the criticality of cache blocks varies among the sharers and sharing degrees (number of sharers), especially for multi-threaded workloads.

The coherence information such as coherence states history, sharers and sharing degree, indicates how the block is utilized within the cache hierarchy, which can be used to guide for better reuse prediction. The extent of predicting inter-core reuse of shared data can be improved by introducing coherence-awareness to optimize replacement or insertion policies. The reuse of cache blocks can be learned using machine learning techniques [5]. In this work, we propose to include one hot encoding of block sharers and number of sharers as features to explore the design space for reuse prediction using perceptron learning.

## II. Background and Motivation

Cache replacement has been an active field of research in computer architecture. Though the LRU replacement policy has been a de-facto policy for a long time, it is not robust for recency-unfriendly cache access patterns, such as the thrashing access patterns and streaming access patterns. It is important to detect the cache blocks with high reuse to avoid unnecessary evictions and misses. Research towards this involves efforts to accurately predict block reuse and adjusting cache line replacement and insertion by learning re-reference behavior of the program. Qureshi et al. [6] improves the LRU Insertion Policy to insert the most recently used block in the LRU position in eviction chain, rather than the MRU position. This is enhanced in the Bimodal Insertion Policy (BIP) that adapts to changes in the working set and their Dynamic Insertion Policy (DIP) chooses between BIP and the traditional LRU policy depending on which policy incurs fewer misses. The Static Re-Reference Interval Prediction (SRRIP) [1] scheme predicts the re-reference distance of new cache entries and places them at a near-immediate position, rather than a distant one (as in the case of LIP). DRRIP is the DIP-equivalent for this policy. Petoumenos et al.s instruction-based reused distance predictor [3] uses the temporal characteristics of a cache block to predict its reuse distance at run-time, based on the access patterns of the instructions (PCs). The Signature-Based Hit Predictor (SHiP) [4] uses memory region, program counter, and instruction sequence history based signatures to predict the re-reference behavior of a cache block. Sampling Dead Block Prediction [2] samples a small number of sets out of the entire cache using partial tags for reuse prediction. It uses the address of the last memory access instruction for the prediction. In Xie et al.s [7] work, a new cache management approach has been proposed that combines dynamic insertion and promotion policies to provide the benefits of cache partitioning, adaptive insertion, and capacity stealing all with a single mechanism.

Researches in the recent years are adopting machine learning for cache management to further reduce the miss rate and enhance performance. Teran et al. [5] has applied perceptron learning using tags and PCs to learn the correlations between past cache access pattern and future accesses, which drives replacement and bypass optimization.

Most of the above work are evaluated using single-threaded workloads, while cache management from the perspective of multi-threaded applications is rarely explored. And for shared memory, the coherence information captures the utilization of the cache block and also the program behavior. However, it is not exploited to improve LLC efficiency.

In this research, we propose coherence-aware reuse prediction with perceptron learning for multi-threaded applications.

## III. Related Work

This section visits the state of the art in cache management from three perspectives, namely, reuse prediction and replacement policies for LLCs, inclusive cache management and sharing awareness in multi-core systems for multi-threaded applications.

*1) Reuse Prediction:* Khan et al. [2] introduces sampling dead block prediction. This technique predicts whether a cache block is dead based on the program counter and drives dead block replacement and bypass optimization. It maintains a sample set of the cache and three prediction tables, which are updated by accesses and evictions to the sampler set. Teran et al.s adopts [5] on perceptron learning for reuse prediction shows a higher accuracy over the SDBP prediction technique It uses perceptron learning to utilize features such as data and instruction addresses to find correlations between past cache access behavior and future accesses, which in turn help to determine replacement and bypass optimization.

*2) Inclusive Cache Management:* Cache management are researched to bridge the gap between inclusive caches and non-inclusive/exclusive caches [11], [12]. Tian et al. [11] introduced Temporal-Based Multilevel Correlating management (TMC) to detect temporal-aware LLC replacement candidates. Jallel et al. [1] proposed Temporal Locality Aware (TLA) cache management policies to allow an inclusive LLC to be aware of the temporal locality of lines in the core caches and avoid evicting these blocks from the LLC.

*3) Sharing Awareness Cache Management:* Panda and Balachandran [16] introduced the idea for bringing coherence and sharing awareness in replacement policies for parallel applications. Later, Natarajan and Chaudhuri [13] characterized the contribution of LLC hits for shared and private blocks in multi-threaded applications, which shows that shared blocks are more important than private blocks. They also analyze how sharing-awareness can reduce LLC miss by over LRU policy. This is of great interest to us for understanding the potential benefits of avoiding LLC evictions of temporal blocks at higher cache level due to sharing-unawareness.

Further, Banerjee's thesis [17] presents a more comprehensive study of sharing behavior in the LLC and provides a descriptive approach of making replacement policies sharing-aware with the help of various signatures.

In this project, we propose to use coherence information as learning features and rely on machine learning techniques to learn the correlation among the features and reuse, and use it to guide LLC replacement policy.

## IV. Coherence-Aware Reuse Prediction

The main idea is to use coherence information of the cache blocks as extended learning features to guide reuse prediction. In shared-memory systems, coherence information such as coherence states, sharers and number of sharers tells how the cache block is utilized in the running multi-threaded application. The Perceptron reuse prediction [5] works well for multi-programmed workloads by learning the cache access pattern and program phase using combination of tags and PCs. Using Perceptron reuse prediction as basis, we extend the feature set to include number of sharers and onehot encoding of sharers to explore coherence awareness for reuse prediction. We derive five alternatives using these extra features. Unlike the Perceptron, the coherence-aware schemes interact with LLC more tightly since they need to query the coherence information from LLC. For misses in LLC, there is no valid coherence information for the access. Fortunately, we can precisely speculate such information since it will be the first access to the new block. So, the number of sharers and sharers encoding will be 1 and ($1 \ll$ requester id). Figure 1 shows the organization and data-path of the predictor with enhanced part highlighted.

*1) Bias:* In Bias, we use the number of sharers of the requested line and use it as bias for the perceptron prediction. Simply, the new prediction becomes $y_{out} = y_{perceptron} - bias$. The intuition behind this is that cache blocks with higher sharing degree tend to have high reuse in the cache hierarchy.

*2) NumSharers and NumSharersHash:* In these alternatives, we query the number of sharers and use it as another

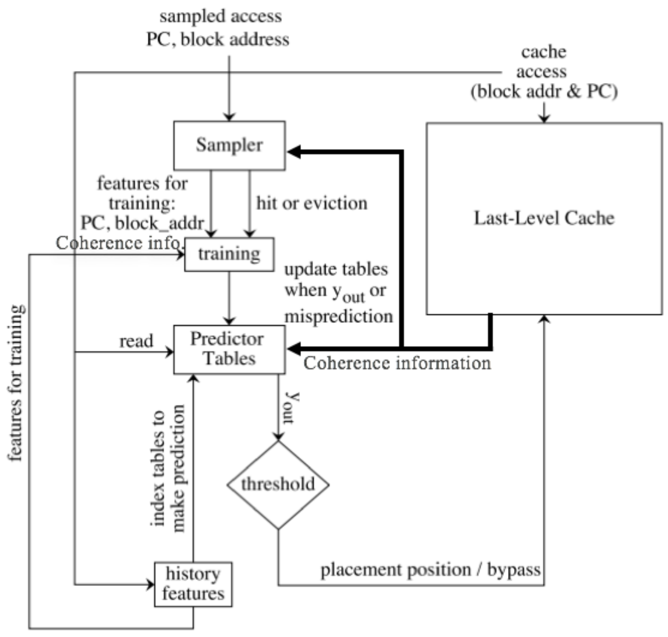

Fig. 1. Coherence-Aware Reuse Predictor Organization and Datapth [5].

feature to index its weight table. The difference between NumSharers and NumSharersHash is the latter XORs the number of sharers with the PC as other features. So the NumShares only needs a table with number of entries same as the core numbers.

*3) OneHot and OneHotHash:* OneHot represents the one hot encoding of the sharers of a cache block. We use this as an extended feature for Perceptron learning. Similar to NumSharersHash, OneHotHash XORs the encoding value with PC and compute the index for the corresponding table.

## V. Methodology

### A. Simulation Configuration

We use an execution-driven simulator ZSim [10] to model detail micro-architectural behaviors. ZSim supports multi-threaded applications and models the cache hierarchy in detail. The L2 is configured as inclusive due to the limits of ZSim that it only support non-inclusion for LLC. We configured the system with 8 OOO cores with parameters shown in Table I.

TABLE I
System Configuration

| | |
|---|---|
| **Cores** | Westmere-like OOO at 2.4 GHz, 8 cores (MT) |
| **L1 caches** | 32 KB, 4-way set-assoc, split D/I, 3 cycles |
| **L2 caches** | Private, 256 KB, 8-way set-assoc, inclusive, 7 cycles |
| **L3 cache** | Shared, 4, 8, 16 MB, inclusive, 8-way set-assoc, 27 cycles |
| **Coherence** | MESI, 64B lines |
| **Memory** | DDR3-1333 MHz, 4 ranks/channel, 4 channels |

### B. Workloads

We use 8 multi-threaded applications and kernels from the PARSEC [8] benchmark suite for evaluation. We exclude *ferret, raytrace* and *vips* due to compilation issues and *facesim* due to long simulation time. All benchmarks are using pthread implementations and run in a combination of two with 4 threads each, using large input data set. We annotate the benchmarks and fast-forward to the parallel phases for detailed simulation. All the runs are executed to the end.

### C. Replacement Policies

We implement SDBP [2] and Perceptron [5]. Based on Perceptron, we extend the feature set with coherence information (e.g. number of sharers and one hot encoding of sharers) and derive five alternatives: Bias, NumSharers, NumSharersHash, OneHot and OneHotHash. These predictors are trained with a sampling set and predicts reuse for every cache access. The threshold values we used are same as those in [5].

The baseline replacement policy is LRU (Sharers Aware), which first selects the blocks with least number of sharers and then it applies LRU policy. With reuse prediction, the replacement policy prioritizes the blocks without reuse for victim selection, then LRU is applied. We also evaluate normal LRU (no sharers aware) replacement policy. Bypass optimization is not implemented due to time limit as it needs to go through the coherence actions in inclusive caches. We leave it for future work.

## VI. Results

In this section, we report the simulation results for different policies. We analyzed the results to show performance speedup and cache misses for various techniques.

TABLE II
Abbreviations used for benchmarks

| | | | |
|---|---|---|---|
| **blackscholes** | bl | **fluidanimate** | fl |
| **bodytrack** | bo | **freqmine** | fr |
| **canneal** | ca | **streamcluster** | st |
| **dedup** | de | **x264** | x2 |

*1) Cache misses (MPKI):* Fig. 2 reports the misses per kilo instructions (MPKI) for various techniques over LRU policy with 4 MB LLC configuration. With the increase in the LLC size the overall MPKI reduces for all the techniques except for perceptron with bias as a feature, for which the MPKI remains more or less a constant.

For a 16 MB LLC, LRU (no sharers aware), LRU and SDBP replacement policies outperform other replacement policies, in general. Almost all the benchmark combinations show changes when moved from 16 MB LLC to 8 MB LLC. On average, perceptron with bias as a feature, shows much less MPKI, dropped to 52%, w.r.t LRU. Other variations of perceptron show lower MPKI w.r.t LRU as well. Interestingly, for a combination of *streamcluster* with *x264*, *freqmine*, *fluidanimate*, *canneal*, *bodytrack* and *blackscholes* (i.e. every other benchmark except *dedup*), with perceptron bias replacement policy MPKI dropped to at least 30% w.r.t LRU. For a combination of *x264 & canneal*, MPKI respective to every

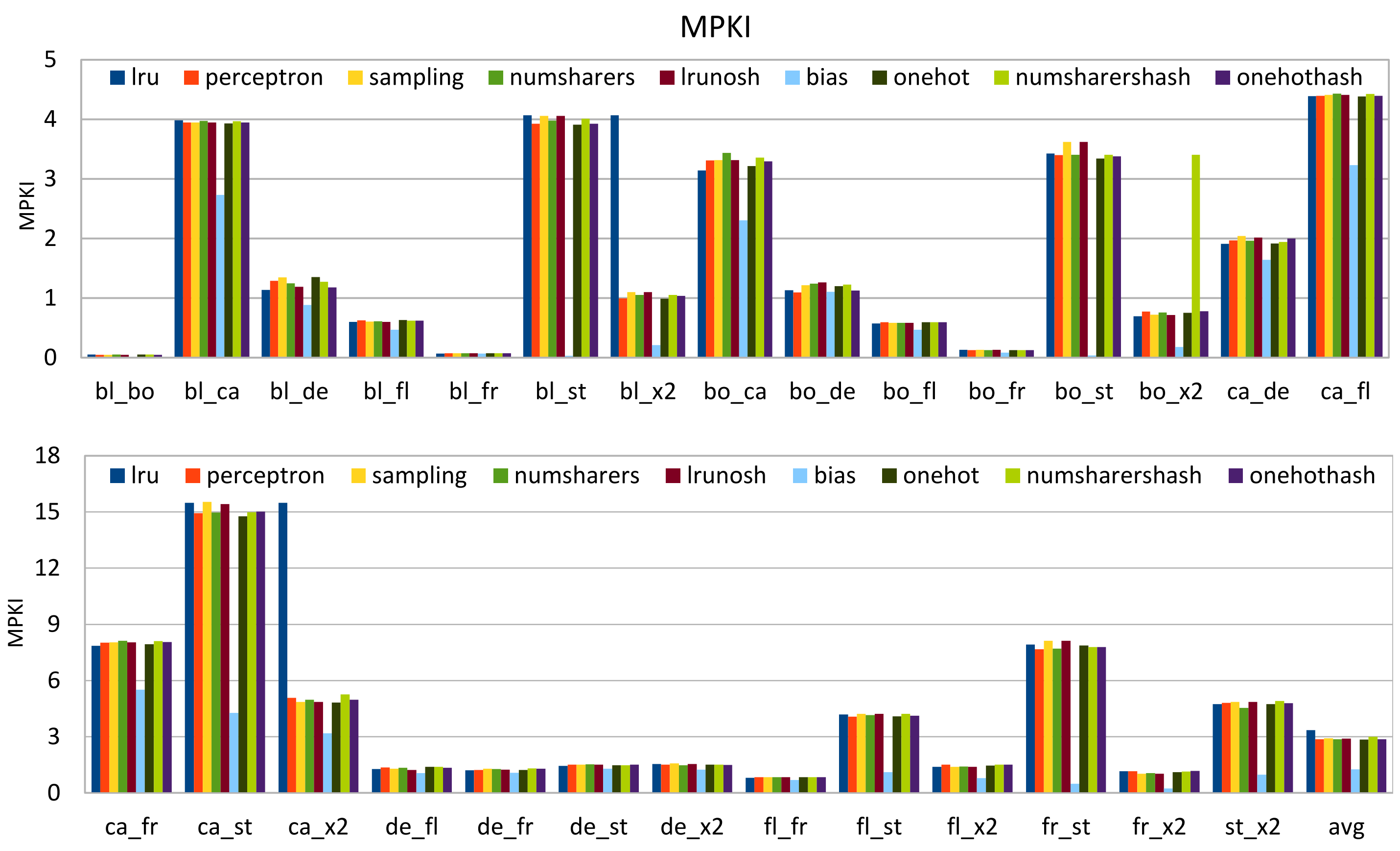

Fig. 2. MPKI for combination of PARSEC benchmarks for 4 MB LLC

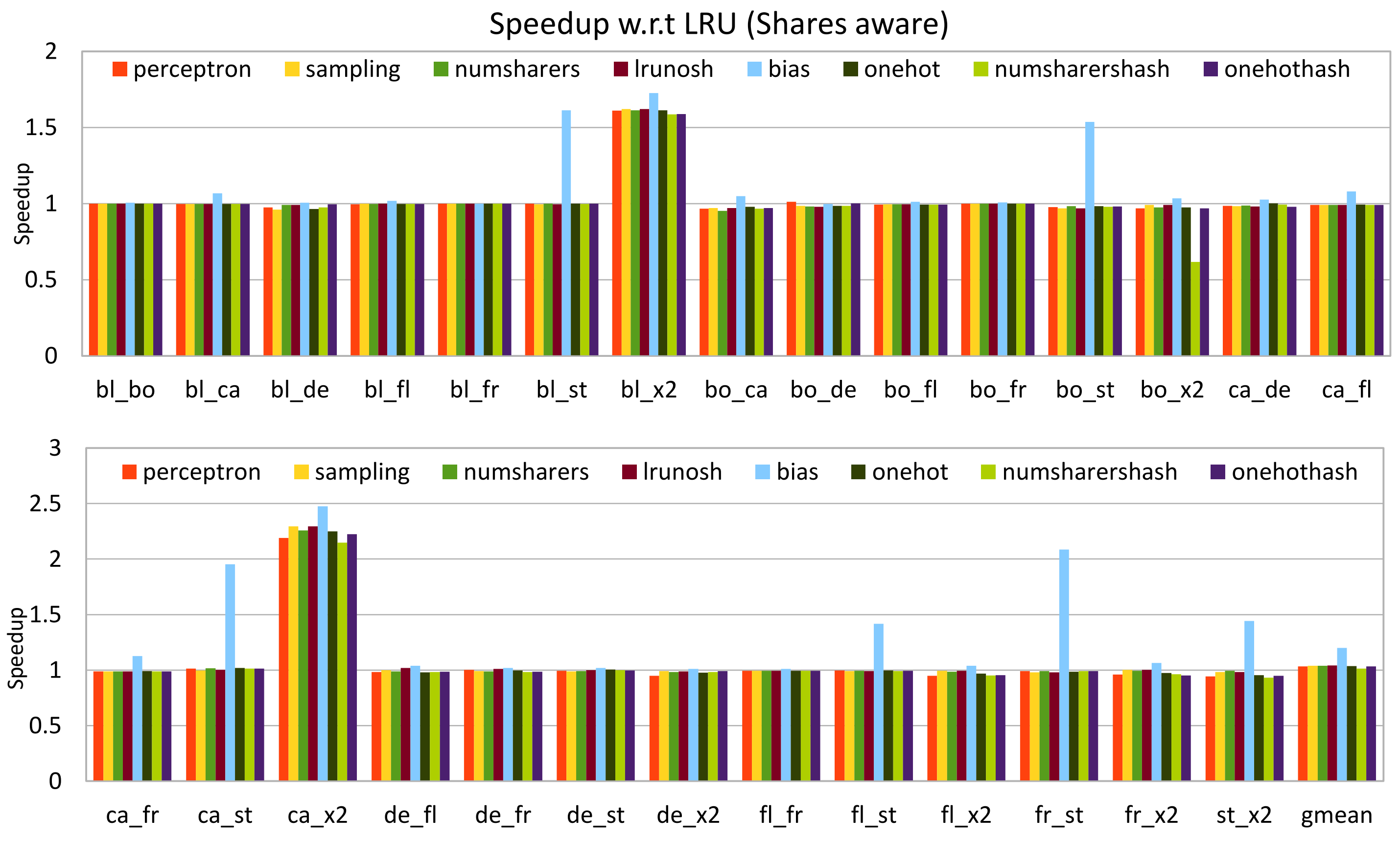

Fig. 3. Speedup for combination of PARSEC benchmarks for 4 MB LLC

replacement policy, except numsharers, dropped to around 30%, w.r.t LRU.

Using a 4 MB LLC, we observe similar results as observed with an 8 MB LLC. On an average, perceptron with bias as a feature, shows much less MPKI, dropped to 40%, w.r.t LRU. Other variations of perceptron show lower MPKI w.r.t LRU as well. For a combination of *streamcluster* with *x264*, *freqmine*, *fluidanimate*, *canneal*, *bodytrack* and *blackscholes* (i.e. every other benchmark except *dedup*), with perceptron bias replacement policy MPKI dropped to at least 25% w.r.t LRU. For a combination of *x264 & blackscholes* and *x264 & canneal*, MPKI respective to every replacement policy dropped to around 33% and 25% respectively, w.r.t LRU.

*2) System performance (Normalized IPC):* We use normalized IPC as performance metric since we start the detailed simulation from the beginning of parallel phases and run the application till the end. Fig. 3 shows the speedups of various replacement policies normalized to LRU, with a 4 MB LLC configuration.

For a 16 MB LLC, it is observed that LRU (no shares aware), LRU and SDBP replacement policies outperform other replacement policies, in general. For a combination of *streamcluster & x264*, numsharershash achieves speedup of around 6% over LRU.

For an 8 MB LLC, perceptron with bias as a feature achieves geometric mean speedup of 15% over LRU. Other variations of perceptron show marginal improvement over LRU as well. Similar to MPKI, for a combination of *streamcluster* with *x264*, *freqmine*, *fluidanimate*, *canneal*, *bodytrack* and *blackscholes* (i.e. every other benchmark except *dedup*), bias achieves a speedup of at least 38% over LRU. For a combination of *x264 & canneal*, every replacement policy, except numsharers, achieve a speedup of at least 120%, over LRU.

Similarly for a 4 MB LLC, perceptron with bias as a feature achieves geometric mean speedup of 20% over LRU. Other variations of perceptron show marginal improvement over LRU as well. For a combination of *streamcluster* with *x264*, *freqmine*, *fluidanimate*, *canneal*, *bodytrack* and *blackscholes* (i.e. every other benchmark except *dedup*), bias achieves a speedup of at least 40% over LRU. For a combination of *x264 & blackscholes* and *x264 & canneal*, every replacement policy achieve a speedup of 60% and 120% respectively, over LRU.

## VII. Conclusion and Future Work

This project explores the design space of perceptron-based reuse prediction using cache coherence information, specifically, number of sharers and one hot encoding of sharers. We derive five alternatives and for 4 MB & 8 MB LLC they all show improvement, on an average. Especially, perceptron implemented with bias as a feature outperforms every other replacement policy and shows a major improvement.

In near future, we plan to analyze and explain the results. For this we need to obtain the maximum achievable performance and observe the number of blocks evicted for every number of sharers.

Further, we plan to explore the design space by exploiting various coherence information and their combination such as coherence states transitions and history, and identify the effective features and their applications. In addition, we will implement bypass optimization in ZSim to support in inclusive cache hierarchy.

## References

[1] Jaleel, Aamer, Kevin B. Theobald, Simon C. Steely Jr, and Joel Emer. ”High performance cache replacement using re-reference interval prediction (RRIP).” *In ACM SIGARCH Computer Architecture News, vol. 38, no. 3, pp. 60-71. ACM, 2010.*

[2] Khan, Samira Manabi, Yingying Tian, and Daniel A. Jimenez. ”Sampling dead block prediction for last-level caches.” In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 175-186. *IEEE Computer Society, 2010.*

[3] Petoumenos, Pavlos, Georgios Keramidas, and Stefanos Kaxiras. ”Instruction-based reuse-distance prediction for effective cache management.” Systems, Architectures, Modeling, and Simulation, 2009. *SAMOS'09. International Symposium on. IEEE, 2009.*

[4] Wu, Carole-Jean, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely Jr, and Joel Emer. ”SHiP: Signature-based hit predictor for high performance caching.” *In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 430-441. ACM, 2011.*

[5] Teran, Elvira, Zhe Wang, and Daniel A. Jimnez. ”Perceptron learning for reuse prediction.” *In Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on, pp. 1-12. IEEE, 2016.*

[6] Qureshi, Moinuddin K., et al. ”Adaptive insertion policies for high performance caching.” *ACM SIGARCH Computer Architecture News. Vol. 35. No. 2. ACM, 2007.*

[7] Xie, Yuejian, and Gabriel H. Loh. ”PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches.” *ACM SIGARCH Computer Architecture News. Vol. 37. No. 3. ACM, 2009.*

[8] Bienia, Christian, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. ”The PARSEC benchmark suite: Characterization and architectural implications.” *In Proceedings of the 17th international conference on Parallel architectures and compilation techniques, pp. 72-81. ACM, 2008.*

[9] Teran, Elvira, Yingying Tian, Zhe Wang, and Daniel A. Jimnez. ”Minimal disturbance placement and promotion.” *In High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on, pp. 201-211. IEEE, 2016.*

[10] Sanchez, Daniel, and Christos Kozyrakis. ”ZSim: fast and accurate microarchitectural simulation of thousand-core systems.” *ACM SIGARCH Computer Architecture News 41, no. 3 (2013): 475-486.*

[11] Tian, Y., Khan, S.M. and Jimenez, D.A. ”Temporal-based multilevel correlating inclusive cache replacement.” *ACM Transactions on Architecture and Code Optimization (TACO), 10(4), p.33., 2013*

[12] Jaleel, A., Borch, E., Bhandaru, M., Steely Jr, S.C. and Emer, J. ”Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (tla) cache management policies.” *In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (pp. 151-162). IEEE Computer Society, 2010, December.*

[13] Natarajan, R. and Chaudhuri, M. ”Characterizing multi-threaded applications for designing sharing-aware last-level cache replacement policies.” *In Workload Characterization (IISWC), 2013 IEEE International Symposium on (pp. 1-10). IEEE, 2013, September.*

[14] Lempel, Oded. ”2nd Generation Intel Core Processor Family: Intel Core i7, i5 and i3.” *Hot Chips 23 Symposium (HCS), 2011 IEEE. IEEE, 2011.*

[15] Chen, Y., Li, W., Kim, C. and Tang, Z. ”Efficient shared cache management through sharing-aware replacement and streaming-aware insertion policy.” *In Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on (pp. 1-11). IEEE, 2009, May.*

[16] Panda, Biswabandan and Balachandran, Shankar ”CSHARP: Coherence and SHaring Aware Replacement Policies for Parallel Applications.” *In Proceedings of 24th IEEE International Conference on Computer Architecture and High Performance Computing. IEEE, 2012.*

[17] Banerjee, Subarno (supervisor Chaudhuri, Mainak). ”Mining Signatures of Inter-core Sharing Behaviour in the Last-Level Caches of Multi-core Processors.” *M.Tech thesis, IIT Kanpur. 2015, June.*