Overhead	2-core	4-core
Core RRPV (2 bits per core per set)	4 KB	16 KB
Block RRPV bits (16 bits per set)	16 KB	32 KB
MLP Cost storage (24 bits per set)	24 KB	48 KB
Net overhead	44 KB	96 KB
% overhead over LLC	1.1%	1.2%

Table 4: Storage overhead of AB-Aware over SRRIP

6 RELATED WORK

Both academic and industrial research communities have made significant contributions in the development of efficient cache management policies. We summarize prior research work in the area of improving LLC performance through efficient management.

Qureshi *et al.* [10] proposed a low-cost run-time mechanism to compute MLP-cost of in-flight misses and uses the MLP-cost to reduce isolated misses. This replacement scheme uses LRU as its baseline policy. Hence, the policy is not efficient for a multicore system when a STR application high data demand is running with a CF application having low data demand. In such a case, STR application will thrash the data of CF from the shared cache. Qureshi *et al.* proposed a Utility-based Cache Partitioning (UCP) [11] scheme to minimize the miss rate of all the competing applications. It uses Utility Monitor (UMON) counters to find the utility of providing extra *ways* to an application. Based on statistics received from each UMON, replacement policy logically partitions the cache to have the lowest number of misses.

Qureshi *et al.* [9] show that on an average more than 60% of the cache lines are not re-referred before getting evicted as LLC receives filtered data locality. Such *zero reuse* cache lines adversely affect the performance of other applications under LRU policy, as these cache lines spend time traversing from MRU to LRU position before eviction, and therefore consume cache resources without any returns in terms of reuse. *LRU Insertion Policy (LIP)* [9] inserts cache blocks directly at LRU position, so they get evicted without spending much time in cache.

Jaleel *et al.* [6] proposed *Static Re-reference Interval Prediction (SRRIP)*, which predicts re-reference intervals of cache blocks. Instead of maintaining recency counters, SRRIP maintains M-bit saturating counter, which they call *Re-Reference Prediction Value (RRPV)*. $RRPV = 0$ denotes near re-reference interval and $RRPV = 2^M - 1$ denotes distant re-reference interval. SRRIP inserts blocks at $RRPV = 2^M - 2$, which is intermediate re-reference value. Upon hit, the block is promoted to $RRPV = 0$ (promotion policy), which is equivalent to MRU position. SRRIP tries to preserve the data-sets of the application showing good locality, however, some workloads have very high access rates compared to others. Such workloads may still interfere with cache blocks of recency-friendly workloads, especially, if they have low access rates. STR applications may produce misses with high rate, and CF blocks may get evicted before getting the first hit. To address this issue, Lathigara *et al.* [7] propose to add one more RRPV counter at core/application level in *Application Behavior-aware Re-reference Interval Prediction (ABRIP)* policy. However, ABRIP doesn't perform well when workload mixes are CF-CF.

7 CONCLUSIONS & FUTURE WORK

LLC is generally shared by multiple applications having diverse nature and different access pattern. Applications also have different MLP-behavior. Some applications have more parallel misses, whereas other applications have many isolated misses. Conventionally used LRU-based cache replacement policies are unaware of such application behaviors. We proposed an application behavior aware replacement policy AB-Aware. It considered both application's MLP-behavior and data-reuse behavior to manage shared LLC. The proposed policy reduced the negative interference among applications by assimilating each block's recency information to understand application's behavior as a whole. It also reduces the miss-penalty associated with each LLC miss. Experimental results showed up to 15.85% performance improvement (1.69% on an average) for 2-core configuration and up to 23.8% IPC improvement (8.71% on an average) for 4-core configuration over SRRIP.

We plan to investigate the benefits of using Auxiliary Tag Directory (ATD) to dynamically choose the best performing policy among SRRIP and AB-Aware as in few workload mixes SRRIP outperforms AB-Aware policy. AB-Aware policy increases the total number of misses for few workload mixes, we plan to study the energy cost for these increased misses and increased activity in MSHR due to updates in every cycle.

REFERENCES

[1] L. A. Belady. 1966. A Study of Replacement Algorithms for a Virtual-storage Computer. *IBM Syst. J.* 5, 2 (June 1966), 78–101.

[2] Shekhar Borkar and Andrew A. Chien. 2011. The Future of Microprocessors. *Commun. ACM* 54, 5 (May 2011), 67–77.

[3] Trevor E. Carlson, Wim Heirman, Stijn Eyerman, Ibrahim Hur, and Lieven Eeckhout. 2014. An Evaluation of High-Level Mechanistic Core Models. *ACM TACO* (2014).

[4] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17.

[5] Aamer Jaleel, William Hasenplaugh, Moinuddin Qureshi, Julien Sebot, Simon Steely, Jr., and Joel Emer. 2008. Adaptive Insertion Policies for Managing Shared Caches. In *PACT-17*. 208–219.

[6] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. 2010. High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP). In *ISCA-37*. 60–71.

[7] P. Lathigara, S. Balachandran, and V. Singh. 2015. Application Behavior Aware Re-reference Interval Prediction for Shared LLC. In *ICCD-33*. 172–179.

[8] Harish Patil, Robert Cohn, Mark Charney, Rajiv Kapoor, Andrew Sun, and Anand Karunanidhi. 2004. Pinpointing Representative Portions of Large Intel® Itanium® Programs with Dynamic Instrumentation. In *Micro-37*. 81–92.

[9] Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. 2007. Adaptive Insertion Policies for High Performance Caching. In *ISCA-34*. 381–391.

[10] Moinuddin K. Qureshi, Daniel N. Lynch, Onur Mutlu, and Yale N. Patt. 2006. A Case for MLP-Aware Cache Replacement. In *ISCA-33*. 167–178.

[11] Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *Micro-39*. 423–432.

[12] Carole-Jean Wu, Aamer Jaleel, Margaret Martonosi, Simon C. Steely, Jr., and Joel Emer. 2011. PACMan: Prefetch-aware Cache Management for High Performance Caching. In *Micro-44*.